IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2020, No. 1, pp. 266–288. DOI:10.13154/tosc.v2020.i1.266-288

Spectral analysis of ZUC-256

Jing Yang[1], Thomas Johansson[1] and Alexander Maximov[2]

[1] Dept. of Electrical and Information Technology, Lund University, Lund, Sweden
{jing.yang,thomas.johansson}@eit.lth.se
[2] Ericsson Research, Lund, Sweden
alexander.maximov@ericsson.com

Abstract. In this paper we develop a number of generic techniques and algorithms in spectral analysis of large linear approximations for use in cryptanalysis. We apply the developed tools for cryptanalysis of ZUC-256 and give a distinguishing attack with complexity around 2^{236}. Although the attack is only 2^{20} times faster than exhaustive key search, the result indicates that ZUC-256 does not provide a source with full 256-bit entropy in the generated keystream, which would be expected from a 256-bit key. To the best of our knowledge, this is the first known academic attack on full ZUC-256 with a computational complexity that is below exhaustive key search.

Keywords: ZUC-256 · Stream Cipher · 5G Mobile System Security.

1 Introduction

ZUC is the stream cipher being used as the core of 3GPP Confidentiality and Integrity Algorithms UEA3 & UIA3 for LTE networks [ETS11a]. It was initially proposed in 2010 as the candidate of UEA3 & UIA3 for use in China. After external and public evaluation and two ZUC workshops, respectively in 2010 and 2011, it was ultimately accepted by 3GPP SA3 as a new inclusion in the LTE standards with a 128-bit security level, i.e., the secret key is 128-bit long.

Like most stream ciphers, ZUC has a linear part, which is an LFSR, and a non-linear part, called the F function, to disrupt the linearity of the LFSR contribution. The design is different from common stream ciphers which are often defined over binary fields $GF(2)$ or extension fields of $GF(2)$, the LFSR in ZUC is defined over a prime field $GF(p)$ with $p = 2^{31} - 1$ while the registers in F are defined over $GF(2^{32})$. There is a bit-reorganization (BR) layer between the LFSR and F serving as a connection layer to extract bits from the LFSR and push them into F. Thus standard cryptanalysis against common stream ciphers can not be directly applied to ZUC and till now, there is no efficient cryptanalysis of ZUC with an attack faster than exhaustive key search.

After ZUC was announced, there were a number of research work conducted to evaluate the cipher [ETS11b], [STL10], [WHN+12]. A weakness in the initialization phase was found in [STL10], [WHN+12] and this directly resulted in an improved version. After the adoption as the UEA3 & UIA3 standard, there were additional work in cryptanalysis of ZUC. A guess-and-determine attack on ZUC was proposed in [GDL13] based on half-words, i.e. 16-bit blocks, by splitting the registers in the LFSR and FSM into high and low 16 bits, where some carry bits are introduced due to the splitting. It requires 6 keystream words and the complexity is $O(2^{392})$, which is, however, higher than exhaustive key search. In [ZFL11], a differential trail covering 24 rounds of the initialization stage is given, but this does not pose a threat since ZUC has 32 initialization rounds. [LMVH15] also shows that weak inputs do not exist in ZUC when it is initialized with 32 rounds. These results indicate that ZUC is resistant against common attacks.

Received: 2019-11-23, Accepted: 2020-01-23, Published: 2020-05-07

In January 2018, ZUC-256 was announced as the 256-bit version of ZUC [Tea18] in order to satisfy the 256-bit security level requirement of 5G from 3GPP [3GP18]. Compared to ZUC-128, the structure of ZUC-256 remains the same, while only the initialization and message authentication code generation phases are improved to match with the 256-bit security level. Subsequently, in July 2018, a workshop on ZUC-256 was held and some general cryptanalyses were presented, but no obvious weaknesses of ZUC-256 were found. To conclude, till now, there are no efficient cryptanalysis techniques succeeding to reduce the claimed security levels of ZUC (128-bit or 256-bit).

In this paper, we propose a distinguishing attack on ZUC-256 with computational complexity around 2^{236}, by linearly approximating the non-linear part F and the different finite fields between the LFSR and F. The important techniques we employ to find a good linear approximation and compute the bias are called *spectral tools* here for cryptanalysis, using e.g., the Walsh Hadamard Transform(WHT) and the Discrete Fourier Transform (DFT). The spectral tools for cryptanalysis are widely used in linear cryptanalysis to, for example, efficiently compute the distribution or the bias of a linear approximation, since there exist fast algorithms for WHT and DFT which can reduce the computational complexity from $O(N^2)$ to $O(N \log N)$ [MJ05], [LD16]. It is also widely used to investigate the properties of Boolean functions and S-boxes, which can be considered as vectorial Boolean functions, like correlation, autocorrelation, propagation characteristics and value distributions [NH07], [HN12]. We explore the use of WHT and DFT and find new results about efficiently computing the bias or correlations. Importantly, we show how a permutation or a linear masking in the time domain would affect the spectrum points in the frequency domain for widely used operations and components, such as $\boxplus, \oplus$, and S-boxes. Based on that, we give a number of further results on how to choose linear maskings in the time domain by considering the behavior of noise variables in the frequency domain such that a decent approximation with a large bias can be found.

We employ the new findings in spectral analysis of ZUC-256 and use them to develop a distinguishing attack. Even though the distinguishing attack is not a very strong one, it indicates that ZUC-256 can not achieve the full 256-bit security level under this case.

The rest of this paper is organized as follows. We first give the general design and structure of ZUC-256 in Section 2 and then the spectral analysis techniques are given in Section 3. After that, we in Section 4 give a distinguishing attack on ZUC-256 using the spectral tools. Specifically, we first derive a linear approximation in Subsection 4.1; and then we show how to efficiently derive the bias of the approximation in Subsection 4.2 $\sim$ Subsection 4.4 by using the spectral analysis and a technique called "bit-slicing technique"; and lastly we give the distinguishing attack based on the derived approximation. In Section 5, we conclude the paper.

2 Description of ZUC-256

In this section we give a brief description of the ZUC-256 algorithm. Basically, the structure of ZUC-256 is exactly the same as that of ZUC-128, except that the length of the secret key K is changed to be 256-bit long and the loading process of the key and IV is modified accordingly [Tea18]. ZUC-256 takes a 256-bit secret key K and a 128-bit initial vector IV as input and produces a sequence that is usually called *keystream*. In this paper, we use $Z^{(t)}$ to denote the generated keystream block at a time instance t for $t = 1, 2, \ldots$. In ZUC-256, each keystream block is a 32-bit word, so we write $Z^{(t)} \in GF(2^{32}),\ t = 1, 2, \ldots$. Furthermore, each (K, IV) pair should produce a unique keystream sequence, and in practice K is usually fixed and the IV value varies to generate many different keystream sequences.

The overall schematic of the ZUC-256 algorithm is shown in Figure 1. It consists of three layers: the top layer is a linear feedback shift register (LFSR) of 16 stages; the

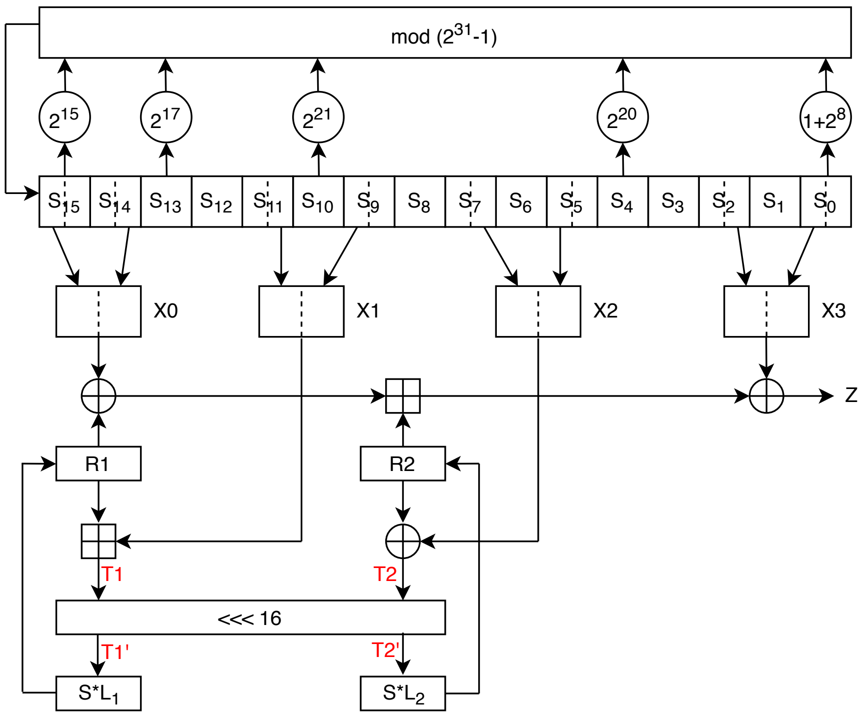

Figure 1: The keystream generation phase of the ZUC-256 stream cipher

bottom layer is a nonlinear block which is called F function; while the middle layer, called bit-reorganization (BR) layer, is a connection layer between the LFSR and F. Now we would give some details of the three layers, and for more details we refer to the original design document [ETS11a].

The LFSR Layer

The LFSR part consists of 16 cells denoted $(s_0, s_1, ..., s_{15})$ each holding 31 bits and giving 496 bits in total. Every value in the cells is an element from the finite field $GF(p)$, where $p = 2^{31} - 1$ and it can be written in a binary representation as

$$x = x_0 + x_1 2 + \ldots + x_{30} 2^{30},$$

where $x_i \in \{0, 1\}$ for $0 \leq i \leq 30$. Then $2^k \cdot x \mod p$ is computed as $x \lll_{31} k$, where $\lll_{31} k$ is the 31-bit left circular shift by k steps. This makes the implementation quite efficient. One can see that the LFSR in ZUC is operating over a prime field instead of $GF(2)$ or $GF(2^n)$ as most stream ciphers do. This makes it insusceptible to common linear cryptanalysis. The feedback polynomial of the LFSR is given by:

$$P(x) = -x^{16} + 2^{15}x^{15} + 2^{17}x^{13} + 2^{21}x^{10} + 2^{20}x^4 + (1 + 2^8) \equiv 0 \mod p.$$

$P(x)$ is a primitive polynomial over $GF(p)$ and this ensures that the LFSR sequence is an m-sequence with period $p^{16} - 1 \approx 2^{496}$. If we denote the LFSR state at clock t as $(s_0^{(t)}, s_1^{(t)}, ..., s_{15}^{(t)})$, then at the next clock $t + 1$, s_i is shifted to s_{i-1}, i.e., $s_i^{(t)} = s_{i-1}^{(t+1)}$, for $1 \leq i \leq 15$, while $s_{15}^{(t+1)}$ is updated by:

$$s_{15}^{(t+1)} = 2^{15}s_{15}^{(t)} + 2^{17}s_{13}^{(t)} + 2^{21}s_{10}^{(t)} + 2^{20}s_4^{(t)} + (1 + 2^8)s_0^{(t)} \mod p.$$

If $s_{15}^{(t+1)} = 0$, then set $s_{15}^{(t+1)} = p$ (i.e., the representation of element 0 is the binary representation of p).

The BR Layer

The BR layer is the connection layer between the LFSR and F. It extracts 128 bits from the LFSR and forms four 32-bit words $X0, X1, X2, X3$ with the first three being fed to F and the last one XOR-ed with the output of F to finally generate the keystream symbol. For a cell s_i in the LFSR, the low and high 16 bits are extracted as:

$$s_{iL} = s_i[0...15] \quad \text{and} \quad s_{i_H} = s_i[15...30].$$

Then $X0, X1, X2, X3$ are constructed as follows:

$$X0 = s_{15H}||s_{14L}, \qquad X1 = s_{11L}||s_{9H}, \qquad X2 = s_{7L}||s_{5H}, \qquad X3 = s_{2L}||s_{0H},$$

where $h||l$ denotes the concatenation of two 16-bit integers h and l into a 32-bit one, with l being the least significant bits and h being the most significant bits of the result. Then $X1, X2$ will be sent into F to update the registers there.

The Non-linear Layer F

The nonlinear layer F has two internal 32-bit registers $R1$ and $R2$ being updated through linear and nonlinear operations. It is a compression function taking $X0, X1, X2$ as the input and producing one 32-bit word which would be used to generate the keystream symbol as below:

$$Z^{(t)} = ((R1^{(t)} \oplus X0) \boxplus_{32} R2^{(t)}) \oplus X3.$$

Then F is updated by:

$$\begin{aligned} T1 &= R1^{(t)} \boxplus_{32} X1, \\ T2 &= R2^{(t)} \oplus X2, \\ R1^{(t+1)} &= S(L_1(T1_L||T2_H)), \\ R2^{(t+1)} &= S(L_2(T2_L||T1_H)). \end{aligned}$$

Here $S = (S0, S1, S0, S1)$ is a 32×32 S-box composed of four juxtaposed S-boxes, where S_0 and S_1 are two different 8-to-8-bit S-boxes. L_1, L_2 are two 32×32 linear transforms which are defined as follows:

$$\begin{aligned} L_1(X) &= X \oplus (X \lll_{32} 2) \oplus (X \lll_{32} 10) \oplus (X \lll_{32} 18) \oplus (X \lll_{32} 24), \\ L_2(X) &= X \oplus (X \lll_{32} 8) \oplus (X \lll_{32} 14) \oplus (X \lll_{32} 22) \oplus (X \lll_{32} 30). \end{aligned}$$

Just like other stream ciphers, ZUC-256 uses an initialization phase before generating a keystream sequence, to fully mix the secret key and IV. During the initialization phase, the key and IV are loaded into the LFSR registers and the cipher runs 32 iterations with the output from the F function being fed back to the LFSR instead of producing keystream symbols. After the initialization, the cipher enters the keystream mode, with the first output word from F being discarded and the following outputs forming the keystream symbols by XOR-ing with $X3$. Since the attack in this paper only uses the keystream mode, we do not give the details of the initialization mode, but refer to the design document for the details [ETS11a], [Tea18].

3 Spectral tools for cryptanalysis

In multidimensional linear cryptanalysis one often has to deal with large distributions, and be able to find good approximations with large biases that can further be used in an attack. In this section, we give several techniques in spectral analysis which help to efficiently explore a good linear approximation and compute its bias. We will later use most of the presented techniques in cryptanalysis of ZUC-256.

Notations. Let $X^{(1)}, X^{(2)}, \ldots, X^{(t)}$ be t independent random variables taking values from an alphabet of n-bit integers, such that the total size of the alphabet is

$$N = 2^n.$$

For a random variable X, let the sequence of $X_k, k = 0, 1, \ldots, N-1$ represent the distribution table of X, i.e., $X_k = Pr\{X = k\}$, or a sequence of occurrence values in the time domain, e.g. $X_k =$ the number of occurrences of $X = k$. If such a sequence of numbers would be normalized by dividing each entry by the total number of occurrences, we would talk about an empirical distribution or a *type* [CT12].

We will denote by $\mathcal{W}(X)$ the N-point Walsh-Hadamard Transform (WHT) and by $\mathcal{F}(X)$ the N-point Discrete Fourier Transform (DFT). Individual values of the transforms will be addressed by $\mathcal{W}(X)_k$ and $\mathcal{F}(X)_k$, for $k = 0, 1, \ldots, N-1$. We will denote by $\hat{X}_k$ the spectrum value of a point k, i.e., $\hat{X}_k = \mathcal{W}(X)_k$ or $\hat{X}_k = \mathcal{F}(X)_k$, depending on the context. The values $\hat{X}_k$, for $k = 0, 1, \ldots, N-1$, in the frequency domain constitute the *spectrum* of X.

When considering Boolean operations, such as $k \cdot M$, where k is an n-bit integer (or an index) and M is an $n \times n$ Boolean matrix, it should be understood as that the integer k is 1-to-1 mapped to a Boolean vector of length n containing the corresponding bits of the integer k in its binary representation. Then a Boolean multiplication is performed modulo 2, and the resulting Boolean vector can thus be 1-to-1 mapped back to an n-bit integer.

WHT and DFT. The DFT is defined as:

$$\hat{X}_k = \mathcal{F}(X)_k = \sum_{j=0}^{N-1} X_j \cdot e^{-\frac{i2\pi}{N}kj}, \quad \text{for } k = 0, 1, ..., N-1,$$

where $\omega_0 = e^{-i2\pi/N}$ is a primitive N-th root of unity. Every point value $\mathcal{F}(X)_k$ is a complex number with the real part Re() and imaginary part Im(), i.e., $\hat{X}_k = \text{Re}(\hat{X}_k) + i \cdot \text{Im}(\hat{X}_k)$. WHT is a special variant of DFT and it is defined as

$$\hat{X}_k = \mathcal{W}(X)_k = \sum_{j=0}^{N-1} X_j \cdot (-1)^{k \cdot j},$$

where $k \cdot j$ now denotes the bitwise dot product of the binary representation of the n-bit indices k and j. I.e., one can rewrite the dot product in the vectorial binary form:

$$k \cdot t = (k_0, k_1, \ldots, k_{n-1}) \cdot (j_0, j_1, \ldots, j_{n-1})^T \mod 2,$$

where k_i, j_i are the i-th bits of the binary representation of k and j, for $i = 0, 1, \ldots, n-1$. Every $\mathcal{W}(X)_k$ has only the real part and it is an integer.

The squared magnitude at a point k is derived by $|\hat{X}_k|^2 = \text{Re}(\hat{X}_k)^2 + \text{Im}(\hat{X}_k)^2$. The point $k = 0$ in the spectrum represents the sum of all values in the time domain for both WHT and DFT cases, i.e.,

$$|\hat{X}_0| = \sum_{j=0}^{N-1} X_j. \tag{1}$$

There are many well-known fast algorithms computing DFT or WHT in time $O(N \log N)$ and this makes the spectral transform widely used in cryptanalysis and in many other areas as well.

Convolutions. A typical operation in linear multidimensional cryptanalysis is to compute the distribution of a noise variable which is the sum ($\oplus$ or $\boxplus$) of other noise variables (referred to as sub-noise variables). While computing the distribution directly in the time domain might be complicated, the complexity could be largely reduced when using DFT and WHT [MJ05] through:

$$\begin{aligned} (X^{(1)} \boxplus X^{(2)} \boxplus \ldots \boxplus X^{(t)})_k &= \mathcal{F}^{-1}(\mathcal{F}(X^{(1)}) \cdot \mathcal{F}(X^{(2)}) \cdot \ldots \cdot \mathcal{F}(X^{(t)}))_k, \\ (X^{(1)} \oplus X^{(2)} \oplus \ldots \oplus X^{(t)})_k &= \mathcal{W}^{-1}(\mathcal{W}(X^{(1)}) \cdot \mathcal{W}(X^{(2)}) \cdot \ldots \cdot \mathcal{W}(X^{(t)}))_k, \end{aligned} \tag{2}$$

where $\cdot$ is the point-wise multiplication of two spectrum vectors. In particular, the overall complexity is now $O(t \cdot N \log N)$.

3.1 Precision problems and the bias in the frequency domain

The bias of a multidimensional noise variable X is often expressed in the time domain as the *Squared Euclidean Imbalance (SEI)*, which is also called the *capacity* in some papers [HN12], defined in [BJV04] as follows:

$$\epsilon(X) = N \sum_{i=0}^{N-1} (X_i/f - 1/N)^2, \tag{3}$$

where $f = \sum_{i=0}^{N-1} X_i$ is the normalization factor, used in case when the distribution table of X is not normalized. For example, the table in the time domain for X stores the number of occurrences of each entry. If the distribution table of X is already normalized then $f = 1$, as expected for the sum of all probabilities.

It is known that to distinguish a noise distribution X with the above bias $\epsilon(X)$ from random using a hypothesis testing, one needs to collect $O(1/\epsilon(X))$ samples from this distribution [BJV04], [HG97].

Precision problems. Assume that we want to compute the bias of a noise variable X, which is the sum ($\oplus$ or $\boxplus$) of t other sub-noises $X^{(1)}, \ldots, X^{(t)}$ using the convolution formulae given in Equation 2. If the expected bias is $\epsilon(X) \approx 2^{-p}$, then in practice we would expect to have probability values around $2^{-n} \pm 2^{-p/2-n}$, in average, and then a float data type should be able to maintain at least $O(|p/2|)$ bits of precision for every value of X_k in the time domain, conditioned that the float data type has the exponent field (e.g., data types `float` and `double` in standard C).

For example, when we want to compute a bias $\epsilon > 2^{-512}$ ($p = 512$) then underlying data types for float or integer values should hold *at least* 256 bits of precision. This forces a program to utilize a large number arithmetic (e.g., BIGNUM, Quad, etc), which requires larger RAM and HDD storage, and expensive computation time.

In the following, we show that the bias of X may be computed in the frequency domain without having to switch to the time domain, and the required precision may fit well into the standard type `double` in C/C++.

Theorem 1. *For an n-bit random variable X with either normalized or non-normalized probability distribution $(X_0, X_1, ..., X_{N-1})$ and its spectrum $(\hat{X}_0, \hat{X}_1, ..., \hat{X}_{N-1})$, computed either in DFT or WHT, the bias $\epsilon(X)$ can be computed in the frequency domain as the sum of normalized squared magnitudes of all nonzero points, where the zero point, $\hat{X}_0$, serves as the normalization factor, i.e.,*

$$\epsilon(X) = \frac{1}{|\hat{X}_0|^2} \sum_{i=1}^{N-1} |\hat{X}_i|^2.$$

Proof. From Equation 1 we get that the normalization factor is $f = |\hat{X}_0|$. The SEI expression can be written as $\epsilon(X) = N \sum_{i=0}^{N-1} (X_i/f - U_i)^2$, where U is the uniform distribution. According to Parseval's theorem, we can derive $\epsilon(X) = N \sum_{i=0}^{N-1} |X_i/f - U_i|^2 = N \cdot \frac{1}{N} \sum_{i=0}^{N-1} |\mathcal{F}(X/f - U)_i|^2 = \sum_{i=0}^{N-1} |\hat{X}_i/f - \hat{U}_i|^2$. Since $\hat{X}_0 = f, \hat{U}_0 = 1$, and $\hat{U}_k = 0$ for $k = 1, 2, \ldots, N-1$, we get that $\epsilon(X) = \sum_{i=1}^{N-1} |\hat{X}_i/f|^2$, from which the proof follows. $\square$

Theorem 1 means that the required precision of values in the frequency domain can be as small as just a few bits, but the exponent value must be correct and preserved. In C/C++ it is therefore good enough to store the spectrum of a distribution in type `double` which has 52 bits of precision and the smallest exponent it can hold is 2^{-1023}. We can barely imagine cryptanalysis where the expected bias will be smaller than that (and if it will, we can always change the factor $\hat{X}_0$ to a larger value).

A similar technique to compute the bias in the frequency domain has been given in [BJV04], but the probability sequence in the time domain there is the probability differences to the uniform distribution, while the probability sequence here is the original probabilities of the variable X. By this, we could further directly compute the bias of the sum ($\boxplus$ or $\oplus$) of several sub-noises in the frequency domain by combining Theorem 1 and Equation 2: the bias of the $\boxplus$-sum of several sub-noises can be computed by

$$\epsilon(X^{(1)} \boxplus \ldots \boxplus X^{(t)}) = \frac{1}{f} \sum_{k=1}^{N-1} |\mathcal{F}(X^{(1)})_k|^2 \cdot \ldots \cdot |\mathcal{F}(X^{(t)})_k|^2 = \frac{1}{f} \sum_{k=1}^{N-1} \left(\prod_{i=1}^{t} |\mathcal{F}(X^{(i)})_k| \right)^2 ,$$
$$\text{where } \; f = |\mathcal{F}(X^{(1)})_0|^2 \cdot \ldots \cdot |\mathcal{F}(X^{(t)})_0|^2 = \left(\prod_{i=1}^{t} |\mathcal{F}(X^{(i)})_0| \right)^2 , \tag{4}$$

and a similar result holds under the $\oplus$-sum for the WHT case. Note, if we convert each spectrum value $|\mathcal{F}(X^{(i)})_k|$ to $\log_2(|\mathcal{F}(X^{(i)})_k|^2)$ (and, similarly, for the WHT case), then arithmetics in the frequency domain, such as in Equation 4, change from computing products to computing sums. This can give additional speed-up, RAM and storage savings. Later we will show how these results help to find a good approximation.

The main observation and motivation for developing further algorithms. In linear cryptanalysis of stream ciphers where we have an FSM and an LFSR, the approach is usually to first linearly approximate the FSM and get a noise variable X of the linear approximation, then the LFSR contribution in the linear approximation is canceled out by combining several (say t) time instances, such that only noise terms remain. Thus, the final noise is the t-folded noise of X, written as $t \times X$ (i.e., the total noise is the sum of t independent noise variables that follow the same distribution as X), for which the bias is written $\epsilon(t \times X)$. Usually, an attacker tries to maximize this value.

One important observation from Theorem 1 and Equation 4 is that if there is a peak (maximum) value $|\hat{X}_k|$ in the spectrum of X at some nonzero position k, then that peak value will be the dominating contributor to the bias $\epsilon(t \times X)$, as it will contribute $|\hat{X}_k|^{2t}$, while other points in the spectrum of X will have a much less (or even negligible) contribution to the total bias as t grows.

This important observation also affects the case when trying to align the spectrum points from several sub-noises with different distributions to achieve a large bias. We should actually try to move the peak spectrum values of each sub-noise such that they are aligned at some nonzero index k. Then the product of those peak values will result in a large total bias value. This motivates us to develop further algorithms to permute or linearly mask variables and align them at an expected or desired spectrum location k in the frequency domain. In the next sections we will give new findings and algorithms for WHT and DFT cases, which can be helpful in searching for a good linear approximation for common operations in the nonlinear part of a stream cipher, such as $\boxplus$, $\oplus$, S-boxes, etc.

3.2 Algorithms for WHT type approximations

Consider the expression

$$X = M^{(1)}X^{(1)} \oplus M^{(2)}X^{(2)} \oplus \ldots \oplus M^{(t)}X^{(t)}, \tag{5}$$

where distribution tables of $X^{(i)}$'s are known, and an attacker can freely select $n \times n$ full-rank Boolean matrices $M^{(i)}, i = 1 \ldots t$; we want to find a method to efficiently search for the choices of $M^{(i)}$'s to maximize the total bias $\epsilon(X)$. Below we first give a theorem and then an algorithm to achieve this.

Theorem 2. *Given an n-bit variable X and its distribution, for an $n \times n$ full-rank Boolean matrix M we have*

$$\mathcal{W}(M \cdot X)_k = \mathcal{W}(X)_{k \cdot M}$$

Proof. Note that $Pr\{M \cdot X = j\} = Pr\{X = M^{-1} \cdot j\}$, then we have $\mathcal{W}(M \cdot X)_k = \sum_{j=0}^{N-1} X_{M^{-1} \cdot j}(-1)^{k \cdot j} \overset{[i=M^{-1} \cdot j]}{=} \sum_{i=0}^{N-1} X_i(-1)^{k \cdot M \cdot i} = \mathcal{W}(X)_{k \cdot M}$. □

Note that the left-side matrix multiplication $M \cdot X$ is switched to the right-side matrix multiplication $k \cdot M$.

We want to maximize $\epsilon(X)$ in Equation 5 and we know that if spectrum values of $X^{(i)}$'s are aligned after linear masking of $M^{(i)}$'s, we could achieve a large bias. By "aligned" we mean that the largest spectrum magnitudes of each $X^{(i)}$ are at the same location, and this holds for the second largest, the third largest spectrum magnitudes and so on. But in practice, it is unlikely to achieve such a perfect alignment for all spectrum points. Instead, we can try to align n largest spectrum magnitudes and thus getting a decent bias. Algorithm 1 below can be used to achieve this based on Theorem 2.

Algorithm 1 Find $M^{(1)}, \ldots, M^{(t)}$ that maximize spectrum points of X at n indices K

Input The distributions of $X^{(i)}$'s ($1 \le i \le t$) and the index matrix K, which must be an $n \times n$ full-rank Boolean matrix where each row $K_{j,*}$ is a binary form of the j-th spectrum index where we want the best alignment to happen.
Output The $n \times n$ full-rank Boolean matrices $M^{(1)}, M^{(2)}, \ldots, M^{(t)}$

```
1:  procedure WhtMatrixAlign($K, X^{(1)}, \ldots, X^{(t)}$)
2:      for $q = 1, \ldots, t$ do
3:          Compute $W = \mathcal{W}(X^{(q)})$
4:          Let $\{\lambda_1, \lambda_2, \ldots, \lambda_{N-1}\}$ be all nonzero indices sorted as $|W_{\lambda_i}| \ge |W_{\lambda_j}|, i < j$
5:          Construct the $n \times n$ Boolean matrix $\Lambda$ in a greedy approach as follows:
6:          Set a variable $l = 1$ and an $n \times n$ Boolean matrix $\Lambda = \mathbf{0}$
7:          for $i = 0, 1, \ldots, n-1$ do
8:              do
9:                  Set the $i$-th row of $\Lambda$ as $\Lambda_{i,*} = \lambda_l$
10:                 $l := l + 1$
11:             while $rank(\Lambda) \neq (i+1)$
12:         Then we want that $K \cdot M^{(q)} = \Lambda$, from which we derive $M^{(q)} = K^{-1} \cdot \Lambda$.
```

Intuitively, the main trick in Algorithm 1 happens in the step 12. For example, let us take the first row of K as the integer k, and the first row from Λ as the integer λ. The integer λ will eventually be the first value in the sorted list, $\lambda = \lambda_1$, where we have $|\mathcal{W}(X^{(q)})_{\lambda_1}| \to \max$. Following Theorem 2 we then get that the k'th spectrum point of $M^{(q)}X^{(q)}$, expressed as $\mathcal{W}(M^{(q)}X^{(q)})_k = \mathcal{W}(X^{(q)})_{k \cdot M^{(q)}}$, now actually have the largest spectrum value $\mathcal{W}(X^{(q)})_{\lambda_1}$, since $k \cdot M^{(q)} = \lambda = \lambda_1$ by construction in that step 12.

As a comment, in Algorithm 1 we do not really have to sort and find all $N-1$ indices of λ, it is most likely that the inner loop will use just a bit more than n values of the first "best" λ's. Thus, it is enough to only collect the best $c \cdot n$ indices, for some small $c = 2, 3, 4$, out of which the full-rank matrix Λ can be constructed. We note that the algorithm does not necessarily give the best overall bias, but it guarantees that at least n points in the spectrum of X will have the largest possible peak values.

Linear approximation of S-boxes. S-boxes, which can be regarded as vectorial Boolean functions, are widely used in both stream ciphers and block ciphers, serving as the main nonlinear component to disrupt the linearity. Therefore, linear approximations of S-boxes are widely studied in cryptanalysis. For one dimensional approximations of an S-box, i.e., $ax \oplus bS(x)$ where $a, b \in GF(2^n)$ are linear masks, the common way is to construct a linear approximation table (LAT), by trying all possibilities of a, b values. The complexity is $O(2^{2n})$, which is affordable for small S-boxes, e.g., 4-bit, 8-bit. WHT is usually employed to speed up the process. For multiple (vectorized) linear approximations, i.e., $Ax \oplus BS(x)$, where A, B are $n \times n$ full-rank binary masking matrices, testing every choice of A, B would be impossible, and the main task is rather to find A, B such that the linear approximation would be highly biased. Some papers investigated properties of multiple linear approximations, such as [HN12], [HCN19], but there is not much research on how a linear masking in the time domain would affect the spectrum points in the frequency domain, and how to explore good linear maskings to achieve a highly biased approximation. Below we give some new results in these aspects.

Let $S(x)$ be an S-box that maps $\mathbb{Z}_N \to \mathbb{Z}_N$, and $x \in \mathbb{Z}_N$, $N = 2^n$. For the sake of notation in this section the expression of the kind $\mathcal{W}(F(x))$ means the WHT over the distribution table that is constructed through the function $F(x) : \mathbb{Z}_N \to \mathbb{Z}_N$ by running through all values of x.

For an n-bit S-box $S(x)$ and an n-bit integer k, let us introduce the k-th binary-valued (i.e., $\pm 1/N$) function, associated with $S(x)$, as follows

$$B^{[k]}_{\{S(x)\}} = 1/N \cdot (-1)^{k \cdot S(x)}, \quad \text{for } x = 0, 1, \ldots, N-1,$$

where $k \cdot S(x)$ is the scalar product of two binary vectors, i.e., $k \cdot S(x) = \bigoplus_{i=0}^{n-1} k_i \cdot S(x)_i$, and $1/N$ is the normalization factor. Such a combination (without the normalization factor $1/N$) is called a component of the S-box, and for a well-chosen S-box, every component should have good cryptographic properties. We can derive the following results.

Theorem 3. *For a given S-box $S(x)$ and a full-rank Boolean matrix Q we have*

$$\mathcal{W}(S(x) \oplus Q \cdot x)_k = \mathcal{W}(B^{[k]}_{\{S(x)\}})_{k \cdot Q}.$$

Proof. Let X be a *non-normalized* noise distribution of the expression $(S(x) \oplus Q \cdot x)$, where every X_j is the number of different values of x for which $j = S(x) \oplus Q \cdot x$. Then we have:

$$\mathcal{W}(S(x) \oplus Q \cdot x)_k = \frac{1}{N} \sum_{j=0}^{N-1} X_j \cdot (-1)^{k \cdot j} = \frac{1}{N} \sum_{x=0}^{N-1} (-1)^{k \cdot (S(x) \oplus Q \cdot x)} = \sum_{x=0}^{N-1} B^{[k]}_{\{S(x)\}} \cdot (-1)^{k \cdot Q \cdot x},$$

from which the result follows, since the last term is exactly $\mathcal{W}(B^{[k]}_{\{S(x)\}})_{k \cdot Q}$. □

Theorem 3 can now be used to derive a matrix Q such that at least n points in the noise spectrum, where the noise variable is $X = S(x) \oplus Q \cdot x$, will have peak values, thus, making the total bias $\epsilon(X)$ large. Basically, we first search for the $> n$ best one-dimensional linear masks and then we build a matrix Q that contains these best peak values in the spectrum, see Algorithm 2 for details.

Algorithm 2 Find Q that maximizes n spectrum points of $S(x) \oplus Q \cdot x$

Input The n-bit S-box $S(x)$
Output The $n \times n$ full-rank Boolean matrix Q

```
1: procedure WhtSBoxApproximation($S(x)$)
2:     Let $\Phi$ be the sorted list of maximum $c \cdot n$ (for some small $c \approx 4$) best triples $(k, \lambda, \omega)$ sorted by the magnitude of $\omega$, where $k$ is the index of the binary-valued function of the S-box, $\lambda$ denotes the index of the spectrum points and $\omega$ is the corresponding spectrum value. If the list is full and we want to add a new triple then the last (worst) list entry is removed.
3:     for $k = 1, \ldots, N-1$ do
4:         Compute $W = \mathcal{W}(B^{[k]}_{\{S(x)\}})$, where $B^{[k]}_{\{S(x)\}} = 1/N \cdot (-1)^{k \cdot S(x)}$
5:         for $\lambda = 1, \ldots, N-1$ do
6:             Consider the triple $(k, \lambda, \omega = |W_\lambda|)$. If $\omega$ is larger than that in the worst triple of $\Phi$ (with the smallest $\omega$), then add $(k, \lambda, \omega = |W_\lambda|)$ to the list.
7:     From the list $\Phi$ use the greedy approach to construct the $n \times n$ full-rank Boolean matrices $K$ and $\mathit{\Lambda}$, similar to how it was done in Algorithm 1.
8:     Set $l = 0$
9:     for $i = 0, 1, \ldots, n-1$ do
10:        do
11:            if $l = |\Phi|$ then
12:                generate the remaining rows of $K$ and $\mathit{\Lambda}$ randomly
13:                exit from the for-loop
14:            Set the $i$-th row of $K$ as the $k$ value of the $l$-th entry of $\Phi$, i.e., $K_{i,*} = \Phi(l).k$
15:            Set the $i$-th row of $\mathit{\Lambda}$ as the $\lambda$ value of the $l$-th entry of $\Phi$, i.e., $\mathit{\Lambda}_{i,*} = \Phi(l).\lambda$
16:            $l := l + 1$
17:        while $rank(K) \neq (i+1)$ or $rank(\mathit{\Lambda}) \neq (i+1)$
18:    Set $Q = K^{-1}\mathit{\Lambda}$.
```

In Algorithm 2, the choice of the parameter c should be such that we would not need to generate final rows of K and $\mathit{\Lambda}$ randomly. Alternatively, one can also modify the algorithm as follows: when a new triple is added to Φ, we run the greedy algorithm and flag records in Φ that are used to construct K and $\mathit{\Lambda}$. After that, the first worst triple in Φ (starting from the end of Φ) that was not flagged is removed if the size of Φ reaches the limit.

The algorithm does not guarantee to get the maximum possible overall bias, but it guarantees that at least the maximum possible peak value will be present in the noise spectrum, which would allow to get a fairly large bias in the end. The complexity is $O(N^2 \log N)$, but in practice there are usually other sub-noises that depend solely on k and λ, which can be used to select a subset of "promising" k and λ values for actual probing of the total noise spectrum, as it will be shown later for the ZUC-256 case.

Other useful formulae on spectral analysis of S-boxes can be derived in Corollary 1, based on Theorem 2 and Theorem 3.

Corollary 1. *Let M, P, Q be $n \times n$ full-rank Boolean matrices, and let $S(x)$ be a bijective S-box over n-bit integers. Then*

$$\mathcal{W}(MS(Px) \oplus Qx)_k = \mathcal{W}(M(S(x) \oplus M^{-1}QP^{-1}x))_k = \mathcal{W}(S(x) \oplus M^{-1}QP^{-1}x)_{k \cdot M} \quad (6)$$

$$= \mathcal{W}(B^{[k \cdot M]}_{\{S(x)\}})_{k \cdot M \cdot M^{-1}QP^{-1}} = \mathcal{W}(B^{[k \cdot M]}_{\{S(x)\}})_{k \cdot QP^{-1}}, \quad (7)$$

$$\mathcal{W}(MS(Px) \oplus Qx)_k = \mathcal{W}(Mx \oplus QP^{-1}S^{-1}(x))_k = \mathcal{W}(B^{[k \cdot QP^{-1}]}_{\{S^{-1}(x)\}})_{k \cdot M}, \quad (8)$$

$$\mathcal{W}(M(S(Px) \oplus Qx))_k = \mathcal{W}(S(x) \oplus QP^{-1}x)_{k \cdot M} = \mathcal{W}(B^{[k \cdot M]}_{\{S(x)\}})_{k \cdot MQP^{-1}}, \tag{9}$$

$$\mathcal{W}(B^{[k \cdot P]}_{\{S(x)\}})_{k \cdot Q} = \mathcal{W}(B^{[k \cdot Q]}_{\{S^{-1}(x)\}})_{k \cdot P}. \tag{10}$$

Theorem 4 (Linear transformation of S-boxes)**.** *Let us consider the following k-th binary-valued function at its spectrum point $\lambda = k \cdot M$, for some full-rank Boolean matrix M, where the original S-box $S(x)$ is linearly transformed with other full-rank Boolean matrices R and Q,*

$$\mathcal{W}(B^{[k]}_{\{RS(Qx)\}})_\lambda. \tag{11}$$

We want to find a set of the best m triples $\{(k, \lambda, \epsilon)\}$ sorted by the maximum bias ϵ. Assume we have a fast method to find best m triples $\{(k', \lambda', \epsilon)\}$ for $\mathcal{W}(B^{k'}_{\{S(x)\}})_{\lambda'}$ instead, then that set can be converted to $\{(k, \lambda, \epsilon)\}$ as follows:

$$\{(k, \lambda, \epsilon)\} = \{(k' \cdot R^{-1},\ \lambda' \cdot Q,\ \epsilon)\}.$$

Proof. $\mathcal{W}(B^{[k]}_{\{RS(Qx)\}})_\lambda = \mathcal{W}(RS(Qx) \oplus Mx)_k = \mathcal{W}(RS(x) \oplus MQ^{-1}x)_k =$
$= \mathcal{W}(B^{[k \cdot R]}_{\{S(x)\}})_{k \cdot MQ^{-1}}$, and the result follows. □

Theorem 5 (S-box as a disjoint combination)**.** *Let us consider an n-bit S-box constructed from t smaller $n_1, n_2, \ldots, n_t$-bit S-boxes $S_1(x_1), S_2(x_2), \ldots, S_t(x_t)$, such that*

$$S(x) = \begin{pmatrix} S_1(x_1) & S_2(x_2) & \ldots & S_t(x_t) \end{pmatrix}^T,$$

where the n-bit input integer x is split into t n_i-bit ($n = \sum_i n_i$) disjoint sub-values as $x = (x_1|x_2|\ldots|x_t)$. Let us also split indices k, λ in a similar way as $k = (k_1|k_2|\ldots|k_t)$ and $\lambda = (\lambda_1|\lambda_2|\ldots|\lambda_t)$. Then we have the following result

$$\mathcal{W}(B^{[k]}_{\{S(x)\}})_\lambda = \prod_{i=1}^{t} \mathcal{W}(B^{[k_i]}_{\{S_i(x)\}})_{\lambda_i}.$$

Proof. Since all x_i's are independent from each other, the combined bias at any point λ is the product of sub-biases at corresponding λ_i's for each k_i-th binary-valued function of the corresponding S-box $S_i(x)$, which can be proved by below.

$$\begin{aligned}
\prod_{i=1}^{t} \mathcal{W}(B^{[k_i]}_{\{S_i(x)\}})_{\lambda_i} &= \Big(\frac{1}{N_1} \sum_{x_1=0}^{N_1-1} (-1)^{k_1 S_1(x_1) \oplus \lambda_1 x_1}\Big) \cdot \ldots \cdot \Big(\frac{1}{N_t} \sum_{x_t=0}^{N_t-1} (-1)^{k_t S_t(x_t) \oplus \lambda_t x_t}\Big) \\
&= \frac{1}{N_1 N_2 \ldots N_t} \sum_{x_1=0}^{N_1-1} \ldots \sum_{x_t=0}^{N_t-1} (-1)^{k_1 S_1(x_1) \oplus \lambda_1 x_1 \oplus \ldots \oplus k_t S_t(x_t) \oplus \lambda_t x_t} \\
&= \frac{1}{N} \sum_{x=0}^{N-1} (-1)^{kS(x) \oplus \lambda x} = \mathcal{W}(B^{[k]}_{\{S(x)\}})_\lambda.
\end{aligned}$$

□

Theorem 4 and Theorem 5 pave the way to compute the bias of any pair (k, λ) in Equation 11 efficiently in time $O(t)$, without even having to construct a large n-bit distribution of the S-box approximation (e.g., $X = RS(Qx) \oplus Mx$), given that $S(x)$ is constructed from smaller S-boxes, which is a common case in cipher designs. E.g., we can simply precompute the tables of $\{(k_i, \lambda_i, \epsilon)\}$ exhaustively for smaller S-boxes, then apply the theorems to compute the bias for a large composite S-box for any pair (k, λ).

For example, let $X = RS(Qx) \oplus Mx$ be the noise variable as the result of an approximation of a large n-bit composite S-box, $RS(Qx)$, where R and Q are some known $n \times n$ Boolean matrices, and Mx is the approximation of that large S-box with a selectable (or given) $n \times n$ full-rank Boolean matrix M. Then, if we want to get the value of some spectrum point k of X we do: compute $\lambda = k \cdot M$ (Theorem 3), then convert the indices as $k' = k \cdot R$ and $\lambda' = \lambda \cdot Q^{-1}$ (Theorem 4), and split them into t $n_1, \ldots, n_t$-bit integers as $k' = (k'_1, \ldots, k'_t)$ and $\lambda' = (\lambda'_1, \ldots, \lambda'_t)$ (Theorem 5). Then, the desired spectrum value at the index k is derived as

$$\mathcal{W}(X)_k = \prod_{i=1}^{t} \mathcal{W}(B^{[k'_i]}_{\{S_i(x)\}})_{\lambda'_i}.$$

Alongside, this also leads to an efficient and fast algorithm to search for the best set of triples $\{(k, \lambda, \epsilon)\}$ in Equation 11, by "reverting" the procedure. These findings have a direct application in the upcoming cryptanalysis of ZUC-256.

General approach of spectral cryptanalysis using WHT. With the tools and methods developed in this subsection, we can now propose a general framework for finding the best approximation, based on probing spectral indices.

1. Derive the total noise expression based on basic approximations and S-box approximations. The noise expression may involve $\oplus$ operations, Boolean matrices multiplications, where some of the matrices can be selected by the attacker.
2. Derive the expression for the k-th spectrum point of the total noise, using the formulae that we found earlier.
3. Convert expressions such as $k \cdot M$, where the matrix M is selectable, to be some parameter λ. If there are more selectable matrices then more λ's can be used.
4. Probe different tuples $(k, \lambda, \ldots)$ to find the maximum peak value in the spectrum for the total noise. The search space for k's and λ's may be shrunk by spectrum values of basic approximations.
5. Convert the best found tuple into the selected matrices, and compute the final multidimensional bias using the constructed matrices.

3.3 Algorithms for DFT type approximations

In this section we provide a few ideas on spectral analysis for DFT type convolutions. Although these methods were not used in the presented attack on ZUC-256, they can be quite helpful in linear cryptanalysis for some other ciphers.

Consider the expression

$$X = c_1 X^{(1)} \boxplus c_2 X^{(2)} \boxplus \ldots \boxplus c_t X^{(t)} \mod N, \tag{12}$$

where, again, $N = 2^n$ and the attacker can choose the constants c_i's, which must be odd, and $X^{(i)}$'s are independent random variables. We will propose the algorithm to find the best combination of the constants c_i's such that the total noise X will have the best peak spectrum value.

The theorem below would help to decide how to rearrange the spectrum points in the frequency domain to achieve a larger total bias, by multiplication with a constant in the time domain, which is a linear masking.

Theorem 6. *For a given distribution of X and an odd constant c we have*

$$\mathcal{F}(c \cdot X)_k = \mathcal{F}(X)_{k \cdot c \mod N},$$

for any spectrum index $k = 0, 1, \ldots, N-1$.

Proof. $\mathcal{F}(c \cdot X)_k = \sum_{n=0}^{N-1} x_{c^{-1}n} \cdot \left(e^{-i2\pi/N}\right)^{kn} = \sum_{n=0}^{N-1} x_n \cdot \left(e^{-i2\pi/N}\right)^{k \cdot c \cdot n} = \mathcal{F}(X)_{k \cdot c \mod N}$. □

Corollary 2. *Any spectrum value at index $k = 2^m(1+2q)$, for some $m = 0 \ldots n-1, q = 0 \ldots 2^{n-m-1}-1$, can only be relocated to another index k' of the form $k' = 2^m(1+2q')$, for some $q' = 0 \ldots 2^{n-m-1}-1$.*

Proof. The constant c is odd and $c = 1 + 2r$, for some r. If $\mathcal{F}(c \cdot X)_{k'} = \mathcal{F}(X)_k$, we get that $c \cdot k' \equiv k \mod N$, and then $k' \equiv 2^m(1+2q) \cdot (1+2r)^{-1} \mod N$. □

Corollary 3. *Any spectrum value at index $k = 2^m(1+2q)$, for some $m = 0 \ldots n-1, q = 0 \ldots 2^{n-m-1}-1$, can be relocated to the index 2^m in the spectrum by applying the constant $c = 1 + 2q$.*

Proof. $\mathcal{F}(c \cdot X)_{2^m} = \mathcal{F}((1+2q) \cdot X)_{2^m} = \mathcal{F}(X)_{2^m(1+2q)} = \mathcal{F}(X)_k$. □

The results above can be used to solve the problem of finding the constants c_i in Equation 12 such that the spectrum of X would contain the maximum possible peak value.

Algorithm 3 Find c_i's that maximize the peak spectrum point of X in Equation 12

```
Input The distributions of $X^{(i)}$, for $i = 1, 2, \ldots, t$.
Output The coefficients $c_i$, for $i = 1, 2, \ldots, t$.
1: procedure DftConstantsAlign($X^{(1)}, X^{(2)}, \ldots, X^{(t)}$)
2:     Initialize a $t \times n$ matrix $\Psi$ with 0, each cell of which contains the pair $(c, \omega)$.
3:     for $i = 1, \ldots, t$ do
4:         Compute $W = \mathcal{F}(X^{(i)})$
5:         for $m = 0, \ldots, n-1$ and $q = 0, \ldots, 2^{n-m-1}-1$ do
6:             Set $\omega = |W_{2^m(1+2q)}|$
7:             if $\omega \geq \Psi_{i,m}.\omega$ (i.e., the $\omega$ value of the entry in the $i$-th row and $m$-th column in $\Psi$) then set $\Psi_{i,m} = (1+2q, \omega)$
8:     Set $m' = 0$ and $\omega' = 0$
9:     for $m = 0, \ldots, n-1$ do
10:        Compute $\omega = \prod_{i=1}^{t} \Psi_{i,m}.\omega$
11:        if $\omega > \omega'$ then set $m' = m$ and $\omega' = \omega$
12:    for $i = 1, \ldots, t$ do
13:        Assign $c_i = \Psi_{i,m'}.c$ (i.e., the $c$ value of the entry in the $i$-th row and $m'$-th column in $\Psi$)
```

The complexity of the above algorithm is $O(t \cdot N \log N)$.

4 Linear cryptanalysis of ZUC-256

In this section, we perform linear cryptanalysis on ZUC-256. Normally, the basic idea of linear cryptanalysis is to approximate nonlinear operations as linear ones and further to find some linear relationships between the generated keystream symbols or between keystream symbols and the LFSR state words, and thus respectively resulting into a distinguishing attack and correlation attack. In a distinguishing attack over a binary or an extension field over $GF(2)$, the common way is to find LFSR states at several time instances (usually 3, 4 or 5) which are XOR-ed to be zero such that the LFSR contribution in the linear approximation is canceled out while only noise terms remain which would be biased. This common way, however, does not apply well on ZUC, since the LFSR in ZUC is defined over a prime field $GF(2^{31}-1)$ which is different to the extension field $GF(2^{32})$ in the function F.

In this section, we describe a more general approach where the expression that we use to cancel out the LFSR contribution *is directly included in the full noise expression*, which effectively reduces the total noise, i.e., the final bias is larger. This general approach may be used in cryptanalysis of any other stream cipher where an LFSR is involved.

Below we first give our linear approximation of the full ZUC-256, including the LFSR state cancellation process. Then we describe in details how we employ the spectral tools given in Section 3 and a technique we call "bit-slicing" to efficiently compute the bias. Finally, we use the derived linear approximation to launch a distinguishing attack on ZUC-256.

4.1 Linear approximation

Any LFSR's 31-bit word $s_x^{(t)}$ at a time instance t and a cell index x can be expressed as $s_0^{(t+x)}$, for $0 \le t$ and $0 \le x \le 15$. Thus, in this section, we will omit the lower index and refer to an LFSR's word by using the time instance only, i.e., $s^{(t+x)}$. We then try to find a four-tuple of time instances t_1, t_2, t_3, t_4 such that,

$$s^{(t_1)} + s^{(t_2)} = s^{(t_3)} + s^{(t_4)} \mod p \qquad (\text{where } p = 2^{31}-1). \tag{13}$$

Note that for any time offset δ, Equation 13 also holds since the LFSR update is a linear transformation in the ring $1+\mathbb{Z}_p$; i.e., if Equation 13 is true then the following is also true:

$$\forall\delta: \quad s^{(t_1+\delta)} + s^{(t_2+\delta)} = s^{(t_3+\delta)} + s^{(t_4+\delta)} \mod p.$$

At each time instance t_i, we define a 32-bit variable $X^{(t_i)}$ which is the concatenation of the low and high 16-bit parts of $s^{(t_i+a)}$ and $s^{(t_i+b)}$, for some constants $0 \le a, b \le 15$, $a \ne b$, following the description of the BR layer in ZUC-256, i.e., $X^{(t_i)}$ is one of $X0^{(t_i)} = (s_H^{(t_i+15)} || s_L^{(t_i+14)})$, $X1^{(t_i)} = (s_L^{(t_i+11)} || s_H^{(t_i+9)})$, $X2^{(t_i)} = (s_L^{(t_i+7)} || s_H^{(t_i+5)})$, or $X3^{(t_i)} = (s_L^{(t_i+2)} || s_H^{(t_i)})$. Then one can derive the following relation for $X^{(t_i)}$'s according to Equation 13:

$$X^{(t_1)} \boxplus_{16} X^{(t_2)} = X^{(t_3)} \boxplus_{16} X^{(t_4)} \boxplus_{16} C^{(t_1)}, \tag{14}$$

where $\boxplus_{16}$ is the 16-bit arithmetic addition, i.e., addition modulo 2^{16}, of the low and high 16-bit halves of $X^{(t_i)}$'s in parallel. Here $C^{(t_1)} = (C_H^{(t_1)} || C_L^{(t_1)})$ is a 32-bit random carry variable from the approximation of the modulo p, and it can only take the values $C_L^{(t_1)}, C_H^{(t_1)} \in \{0, -1, +1\} \mod 2^{16}$, where the values in the low and high parts of $C^{(t_1)}$ are independent. As an example, the approximation in Equation 14 for $X^{(t_i)} = X1^{(t_i)}$ is then written as:

$$\underbrace{\begin{pmatrix} s_H^{(t_1+9)} \\ s_L^{(t_1+11)} \end{pmatrix}}_{X1^{(t_1)}} \boxplus_{16} \underbrace{\begin{pmatrix} s_H^{(t_2+9)} \\ s_L^{(t_2+11)} \end{pmatrix}}_{X1^{(t_2)}} = \underbrace{\begin{pmatrix} s_H^{(t_3+9)} \\ s_L^{(t_3+11)} \end{pmatrix}}_{X1^{(t_3)}} \boxplus_{16} \underbrace{\begin{pmatrix} s_H^{(t_4+9)} \\ s_L^{(t_4+11)} \end{pmatrix}}_{X1^{(t_4)}} \boxplus_{16} \underbrace{\begin{pmatrix} C1_L^{(t_1)} \\ C1_H^{(t_1)} \end{pmatrix}}_{C1^{(t_1)}}.$$

Next, we would like to derive the distribution of the carries $C_L^{(t_1)}, C_H^{(t_1)}$, and to achieve that, we first give a theorem.

Theorem 7. *Let a modulus p be of the form $p = 2^n - 1$, for an integer $n > 1$. Assume $a_1, a_2, a_3, a_4 \in 1 + \mathbb{Z}_p$, such that $a_1 + a_2 = a_3 + a_4 \mod p$. For integers s and t with $0 \le s < n$ and $1 \le t \le (n - s)$, we extract "middle" t-bit values with the bit-offset s as $A_i^{(s)} = \lfloor a_i/2^s \rfloor \mod 2^t$, for $i = 1, 2, 3, 4$. Then we can get the following approximation*

$$A_1^{(s)} \boxplus_t A_2^{(s)} = A_3^{(s)} \boxplus_t A_4^{(s)} \boxplus_t Q^{(s)} \mod 2^t, \tag{15}$$

where the carry value $Q^{(s)} \in \{0, -1, +1\}$ [1] and it has the distribution $Pr\{Q^{(s)} = 0\} = (2p^2 + 1)/3p^2$, and $Pr\{Q^{(s)} = -1\} = Pr\{Q^{(s)} = +1\} = (p^2 - 1)/6p^2$.

Proof. The proof is given in Appendix A. □

Corollary 4. *The distribution for $C^{(t_1)}$(note here t_1 denotes the time instance) in Equation 14 is as follows:*

$$\begin{aligned} Pr\{C_L^{(t_1)} = 0\} = Pr\{C_H^{(t_1)} = 0\} \approx 2/3, \\ Pr\{C_L^{(t_1)} = -1\} = Pr\{C_H^{(t_1)} = -1\} \approx 1/6, \\ Pr\{C_L^{(t_1)} = +1\} = Pr\{C_H^{(t_1)} = +1\} \approx 1/6. \end{aligned}$$

Proof. Given the relation in Equation 14, we basically need to consider these two 16-bit cases independently (since $a \neq b$): $s_L^{(t_1+a)} \boxplus_{16} s_L^{(t_2+a)} = s_L^{(t_3+a)} \boxplus_{16} s_L^{(t_4+a)} \boxplus_{16} E_L^{(t_1+a)}$ and $s_H^{(t_1+b)} \boxplus_{16} s_H^{(t_2+b)} = s_H^{(t_3+b)} \boxplus_{16} s_H^{(t_4+b)} \boxplus_{16} E_H^{(t_1+b)}$, for some constants $0 \le a, b \le 15, a \neq b$, where the carry $C^{(t_1)}$ is either $(E_L^{(t_1+a)} || E_H^{(t_1+b)})$ or $(E_H^{(t_1+b)} || E_L^{(t_1+a)})$.

The distributions of $E_L^{(t_1+a)}$ and $E_H^{(t_1+b)}$ can be respectively proved through Theorem 7 by setting $n = 31, s = 0, t = 16$ and $n = 31, s = 15, t = 16$. The probability values can be approximated as 2/3 and 1/6 with an error $< 2^{-63}$. □

Next, we list the expressions for generating keystream symbols at time instances t and $t + 1$ as follows,

$$\begin{aligned} Z^{(t)} &= [(T2^{(t)} \oplus X2^{(t)}) \boxplus ((T1^{(t)} \boxminus X1^{(t)}) \oplus X0^{(t)})] \oplus X3^{(t)}, \\ Z^{(t+1)} &= [SL_2(T2'^{(t)}) \boxplus (SL_1(T1'^{(t)}) \oplus X0^{(t+1)})] \oplus X3^{(t+1)}, \end{aligned}$$

where $\boxminus$ is the arithmetic subtraction modulo 2^{32} and $(T1', T2') = (T1, T2) \lll 16$ is a cyclic shift 16 bits to the left. Then we give the full approximation of ZUC-256 based on Equation 13 and its approximation in Equation 14 as follows:

$$\begin{aligned} & M\sigma[Z^{(t_1)} \oplus Z^{(t_2)} \oplus Z^{(t_3)} \oplus Z^{(t_4)}] \oplus [Z^{(t_1+1)} \oplus Z^{(t_2+1)} \oplus Z^{(t_3+1)} \oplus Z^{(t_4+1)}] \\ &= M\sigma N1^{(t_1)} \oplus \bigoplus_{t \in \{t_1, \ldots, t_4\}} M(\underbrace{\sigma T1^{(t)} \oplus \sigma T2^{(t)}}_{=T1'^{(t)} \oplus T2'^{(t)}}) \oplus N2^{(t_1)} \\ &\oplus \bigoplus_{t \in \{t_1, \ldots, t_4\}} (SL_1(T1'^{(t)}) \oplus SL_2(T2'^{(t)})) \\ &= M\sigma N1^{(t_1)} \oplus N2^{(t_1)} \\ &\oplus \bigoplus_{t \in \{t_1, \ldots, t_4\}} \left[M \cdot T1'^{(t)} \oplus SL_1(T1'^{(t)}) \oplus M \cdot T2'^{(t)} \oplus SL_2(T2'^{(t)})\right], \end{aligned}$$

where σ is the swap of the high and low 16 bits of a 32-bit argument, and M is some 32×32 full-rank Boolean matrix that we can choose, which serves as a linear masking

[1] In a special case when $t = 1$ bit the values of $Q^{(s)}$ are $\{0, 1\}$, since $(-1) \equiv 1 \mod 2$; the probabilities of $Pr\{Q^{(s)} = -1\}$ and $Pr\{Q^{(s)} = +1\}$ are then combined into a single case when $Q^{(s)} = 1$.

matrix. The expressions for the noise $N1^{(t_1)}$ (we further split $N1^{(t_1)} = N1a^{(t_1)} \oplus N1b^{(t_1)}$) and noise $N2^{(t_1)}$ are as follows:

$$\begin{aligned} N1a^{(t_1)} = {} & [((T2^{(t_1)} \oplus X2^{(t_1)}) \boxplus ((T1^{(t_1)} \boxminus X1^{(t_1)}) \oplus X0^{(t_1)}))] \\ & \oplus [((T2^{(t_2)} \oplus X2^{(t_2)}) \boxplus ((T1^{(t_2)} \boxminus X1^{(t_2)}) \oplus X0^{(t_2)}))] \\ & \oplus [((T2^{(t_3)} \oplus X2^{(t_3)}) \boxplus ((T1^{(t_3)} \boxminus X1^{(t_3)}) \oplus X0^{(t_3)}))] \\ & \oplus [((T2^{(t_4)} \oplus (X2^{(t_1)} \boxplus_{16} X2^{(t_2)} \boxminus_{16} X2^{(t_3)} \boxminus_{16} C2^{(t_1)})) \boxplus ((T1^{(t_4)} \\ & \boxminus (X1^{(t_1)} \boxplus_{16} X1^{(t_2)} \boxminus_{16} X1^{(t_3)} \boxminus_{16} C1^{(t_1)})) \\ & \oplus (X0^{(t_1)} \boxplus_{16} X0^{(t_2)} \boxminus_{16} X0^{(t_3)} \boxminus_{16} C0^{(t_1)})))] \oplus \bigoplus_{t \in \{t_1,\ldots,t_4\}} (T1^{(t)} \oplus T2^{(t)}), \end{aligned} \tag{16}$$

$$N1b^{(t_1)} = X3^{(t_1)} \oplus X3^{(t_2)} \oplus X3^{(t_3)} \oplus (X3^{(t_1)} \boxplus_{16} X3^{(t_2)} \boxminus_{16} X3^{(t_3)} \boxminus_{16} C3^{(t_1)}),$$

and

$$\begin{aligned} N2^{(t_1)} = {} & [[(SL_2(T2'^{(t_1)}) \boxplus (SL_1(T1'^{(t_1)}) \oplus X0^{(t_1+1)})) \oplus X3^{(t_1+1)}] \\ & \oplus [(SL_2(T2'^{(t_2)}) \boxplus (SL_1(T1'^{(t_2)}) \oplus X0^{(t_2+1)})) \oplus X3^{(t_2+1)}] \\ & \oplus [(SL_2(T2'^{(t_3)}) \boxplus (SL_1(T1'^{(t_3)}) \oplus X0^{(t_3+1)})) \oplus X3^{(t_3+1)}] \\ & \oplus [(SL_2(T2'^{(t_4)}) \boxplus (SL_1(T1'^{(t_4)}) \oplus (X0^{(t_1+1)} \boxplus_{16} X0^{(t_2+1)} \\ & \boxminus_{16} X0^{(t_3+1)} \boxminus_{16} C0^{(t_1+1)}))) \oplus (X3^{(t_1+1)} \boxplus_{16} X3^{(t_2+1)} \boxminus_{16} X3^{(t_3+1)} \\ & \boxminus_{16} C3^{(t_1+1)})]] \oplus \bigoplus_{t \in \{t_1,\ldots,t_4\}} (SL_1(T1'^{(t)}) \oplus SL_2(T2'^{(t)})). \end{aligned}$$

In our analysis we consider noise variables $N1^{(t_1)}$ and $N2^{(t_1)}$ as independent. By this assumption the attacker actually looses some advantage since there is a dependency between, for example, $T1^{(t_1)}, T2^{(t_1)}$ in $N1^{(t_1)}$ and $SL_1(T1'^{(t_1)}), SL_2(T2'^{(t_1)})$ in $N2^{(t_1)}$. The attack can be stronger if we could take into account these dependencies, since then there will be more information in these noise distributions. However, it is practically hard to compute the bias in that scenario.

Next we want to compute the distribution and the bias of the noise terms. However, as one can note, there are many variables involved in each sub-noise expression. For example, the sub-noise $N1a^{(t_1)}$ involves 17 32-bit variables, and 3 C-carries. In order to compute the distribution of $N1a^{(t_1)}$, a naive loop over all combinations of the involved variables would imply the complexity $O(9^3 \cdot 2^{17 \cdot 32})$, which is computationally infeasible.

In the next subsections we make a recap of the bit-slicing technique and show how we adapt it to our case to compute the distributions of the above noise terms.

4.2 Recap on the bit-slicing technique from [MJ05]

Let an n-bit noise variable N be expressed in terms of several n-bit uniformly distributed independent variables, using any combination of bitwise Boolean functions (AND, OR, XOR, etc.) and arithmetical addition $\boxplus$ and subtraction $\boxminus$ modulo 2^n. The distribution of such a noise expression, referred to as a *pseudo-linear function* in [MJ05], can be efficiently derived through the so-called "bit-slicing" technique in complexity $O(k \cdot 2^n + k^2 n \cdot 2^{n/2})$, for some (usually small) k.

The general idea behind the technique is that if we know the set of distributions for $(n-1)$-bit truncated inputs for each possible outcome vector of the sub-carries' values for corresponding arithmetical sub-expressions, then we can easily extend these distributions to the n-bit truncated distributions with a new vector of output sub-carries' values. Then, the algorithm may be viewed as a Markov chain where the nodes are viewed as a vector of probabilities for each combination of sub-carries, and some *transition matrices* are used to go from the $(n-1)$-th state to the n-th state.

Example. Let us explain the technique on a small example. Let $n = 32$ bits and the noise N is expressed in terms of random 32-bit variables A, B, C:

$$N = \underbrace{[\underbrace{(A \boxplus B \boxminus C)}_{\text{inner ADD-1}} \oplus \underbrace{(A \boxplus C)}_{\text{inner ADD-2}}\ ] \boxminus B}_{\text{outer ADD-3}}\,. \tag{17}$$

For each n-bit value X with $(x_{n-1} \dots x_1 x_0)$ as its binary form, we will compute the number of combinations of A, B, C such that the value of N is equal to X.

Carries and the state. Here we have 3 arithmetical parts: two inner and one outer. We express the carries using a vector denoted $(c1, c2, c3)$, where $c1 \in \{-1, 0, +1\}, c2 \in \{0, 1\}, c3 \in \{-1, 0\}$. At each bit position $i, 0 \le i \le n-1$, we would have the input carry vector coming from the first $i-1$ bits, $(c1_{in}, c2_{in}, c3_{in})$, and the output carry vector $(c1_{out}, c2_{out}, c3_{out})$ going to the $(i+1)$-th bit position. Introduce a mapping function τ as: $\tau(c1, c2, c3) = ((c1+1) \cdot 2 + c2) \cdot 2 + (c3+1) \in [0 \dots 11]$, that maps each value of the carry vector to a unique integer index. Thus, at every step i, we will have a column vector V_i of length $k = 12$, each entry of which corresponds to a certain combination of output carries $(c1_{out}, c2_{out}, c3_{out})$, and the values are respectively the number of combinations of the i-bit truncated variables A, B, C such that the first i bits of N are equal to the first i bits of X. The initial vector is $V_0 = (0, \dots 0, 1(\text{ in the index } \tau(c1 = 0, c2 = 0, c3 = 0)), 0, \dots, 0)^{\mathrm{T}}$.

Transition matrices. We will construct two $k \times k$ (12×12) transition matrices, M_0 and M_1, associated with every bit position i for the i-th bit value of X, x_i, being either 0 or 1, such that the vector V_{i+1} is derived by $V_{i+1} = M_{x_i} \cdot V_i$. I.e., when the i-th bit of X, x_i, is 0, we apply M_0, otherwise M_1. These two matrices are constructed as follows: initialize M_0 and M_1 with zeroes; loop through all possible choices of the i-th bits of $A, B, C \in \{0, 1\}^3$ and all possible values of $(c1_{in}, c2_{in}, c3_{in})$; then for each combination we compute the resulting bit $r \in \{0, 1\}$ by evaluating the noise expression, and the vector of output carries $(c1_{out}, c2_{out}, c3_{out})$; we then increase the corresponding matrix cell by 1 as $\texttt{++}M_r[\tau(c1_{out}, c2_{out}, c3_{out})][\tau(c1_{in}, c2_{in}, c3_{in})]$ at the same time. Note, the inner output carries $c1_{out}$ and $c2_{out}$ should not be summed up in the outer output carry $c3_{out}$, while only the resulting 1-bit values of inner sums should go to the outer expression.

In Appendix B we give the code in C for computing the transition matrices M_0 and M_1 for the exampled noise expression given in Equation 17.

The general formulae can now be derived as follows:

$$\Pr\{N = (x_{n-1} \dots x_0)\} = \frac{1}{2^{t \cdot n}} \cdot \underbrace{(1, 1, \dots, 1) \cdot \prod_{i=n/2}^{n-1} M_{x_i}}_{\text{High part, } H[(x_{n-1} \dots x_{n/2})]} \cdot \underbrace{\prod_{i=0}^{n/2-1} M_{x_i} \cdot V_0}_{\text{Low part, } L[(x_{n/2-1} \dots x_0)]}\quad, \tag{18}$$

where t is the number of involved random variables, in our example $t = 3$, and $1/2^{t \cdot n}$ is the normalization factor for the distribution. The left-side row vector $(1, 1, \dots, 1)$ due to the last carries are truncated by the modulo 2^n operation and thus all combinations for all carries' outcomes should be summed up to the result.

Precomputed vectors. We intentionally split Equation 18 into two parts, since it shows that the computation of $Pr\{N = X\}$ for all values of $X \in \{0, 1, \dots, 2^n - 1\}$ can be accelerated by precomputing two tables of the middle sub-vectors in Equation 18 for all possible values of the high $(H[(x_{n-1} \dots x_{n/2})])$ and low $(L[(x_{n/2-1} \dots x_0)])$ halves of X, independently. The whole precomputation takes time $O(n \cdot 2^{n/2} \cdot k^2)$. Then the probability $Pr\{N = X\}$ is a simple scalar product, computed in time $O(k)$ as:

$$\Pr\{N = (x_{n-1} \ldots x_0)\} = \frac{1}{2^{t \cdot n}} \cdot H[(x_{n-1} \ldots x_{n/2})] \cdot L[(x_{n/2-1} \ldots x_0)].$$

4.3 Bit-slicing technique adaptation to compute $N1a$, $N1b$ and $N2$

In this section we will describe in more details how we adapt the bit-slicing technique in order to compute the "heaviest" noise $N1a$. The remaining noises are computationally less demanding and can be derived with similar adaptation techniques.

A direct application of the bit-slicing technique to compute $N1a$, given in Equation 16, is complicated due to: (1) we have $\boxplus_{16}$ adders that block some of the sub-carries to propagate between the 15-th and 16-th bits; and (2) we have random C-carries that can have 3 values at the 0-th and 16-th bits.

The first problem is resolved by introducing two special transition matrices $M_r^{(15)}$ (for $r = 0, 1$) which are only applied to the bit 15. In these matrices, output sub-carries in all involved $\boxplus_{16}$ would not propagate to the next bit 16 and are forced to be 0.

The second problem is solved by introducing another two special transition matrices $M_r^{(0)}$ (for $r = 0, 1$) that are only applied to the bits 0 *and* 16. These special matrices take into account C-values that are added to the 0-th and 16-th bits. The important fact here is that all input sub-carries at bit positions where C's are involved are always 0, and this makes it possible to keep the sub-carry values in the expressions like $(X3^{(t_1)} \boxplus_{16} X3^{(t_2)} \boxminus_{16} X3^{(t_3)} \boxminus_{16} C3^{(t_1)})$ in the smaller range $\{-1, 0, +1\}$, since a C-value $\in \{-1, 0, +1\}$ only appears at the first bit under the 16-bit addition/subtraction where input carries are zeroes, and in the next bits $C = 0$. Thus, to construct $M_r^{(0)}$'s, we do the following: loop through the 1-bit random variables involved in the noise expression; loop through sub-carries that propagate over 32 bits; *do not* loop through the carries that are involved in 16-bit propagations; and loop through $C \in \{-1, 0, +1\}$ values. Then, instead of increasing the corresponding entry of $M_r^{(0)}$ by 1, we actually add the product of the probabilities of all involved C-values.

Transition matrices for the remaining bits (except the bits 0, 15, 16) are constructed as usual, but C-values are all 0.

Additional adaptation is done in the part of L/H precomputed tables of vectors. We know that the low and high precomputations meet in the middle at the bits 15 and 16, where all sub-carries in $\boxplus_{16}$ adders vanish to 0. This makes it possible to shrink the size of the vectors and only leave the states with sub-carries that propagate over all 32-bits of the noise expression.

Complexity. For $N1a$ we have the following situation: we have 17 32-bit variables ($T1$ and $T2$ in 4 time instances and $X0, X1, X2$ in 3 time instances); 8 carries that propagate over 32 bits having binary values either $\{0, +1\}$ or $\{-1, 0\}$; 3 carries that propagate over 16 bits in the range $\{-1, 0, +1\}$. Thus we get the dimension of all involved transition matrices by $k = 2^8 \cdot 3^3$, i.e., the matrices are of size $2^{12.8} \times 2^{12.8}$. If each entry of a matrix is of C-type `double` (8 bytes), then one transition matrix occupies around 365Mb of RAM.

The precomputation phase to compute low (L) and high (H) tables of vectors has time complexity around $O(2^{2 \cdot 12.8} \cdot 32 \cdot 2^{16}) = O(2^{46.6})$. The size of the stored L/H vectors was dramatically reduced from the vector lengths $k = 2^{12.8}$ down to the lengths $k' = 2^8$, since there are only 8 binary-valued sub-carries that propagate between the bits 15 and 16, while other carries were "truncated" by applying the matrix $M_r^{(15)}$.

The total time complexity to construct the noise $N1a$ is, therefore, $O(2^{46.6} + 2^{32} \cdot 2^8)$. We compute $N1b$ and $N2$ with a similar adaptation of the bit-slicing technique, but the time complexity there is a lot smaller.

4.4 Spectral analysis to find the matrix M

With the methods presented in Section 3, the spectral analysis becomes rather simple for the ZUC-256 case, and we below give necessary expressions to perform that. Let us recall that the expression for the total noise is:

$$\begin{aligned} N_{tot}^{(t_1)} &= M\sigma N1^{(t_1)} \oplus N2^{(t_1)} \\ &\oplus \bigoplus_{t\in\{t_1,\ldots,t_4\}} \Big[SL_1(T1'^{(t)}) \oplus M\cdot T1'^{(t)} \oplus SL_2(T2'^{(t)}) \oplus M\cdot T2'^{(t)}\Big]. \end{aligned}$$

The spectrum expression at some point k can thus be derived as follows.

$$\begin{aligned} \mathcal{W}(N_{tot}^{(t_1)})_k &= \mathcal{W}(M\sigma N1)_k \cdot \mathcal{W}(N2)_k \cdot \mathcal{W}(SL_1(x)\oplus Mx)_k^4 \cdot \mathcal{W}(SL_2(x)\oplus Mx)_k^4 \\ &= \mathcal{W}(\sigma N1)_\lambda \cdot \mathcal{W}(N2)_k \cdot \mathcal{W}(B^{[k]}_{\{SL_1(x)\}})^4_\lambda \cdot \mathcal{W}(B^{[k]}_{\{SL_2(x)\}})^4_\lambda, \end{aligned}$$

where $\lambda = k\cdot M$.

Our strategy for the spectral analysis was as follows. We selected $\approx 2^{24.78}$ "promising" spectrum points for λ where $|\mathcal{W}(\sigma N1)_\lambda|^2 > 2^{-150}$, and also selected $\approx 2^{18}$ "promising" spectrum points for k where $|\mathcal{W}(N2)_k|^2 > 2^{-80}$. Then we tried all combinations of the selected (k,λ) and computed the total spectrum value. For the computation of spectrum points for S-boxes (e.g., for $\mathcal{W}(B^{[k]}_{\{SL_1(x)\}})^4_\lambda$), we utilized Theorem 4 and Theorem 5, so that we did not have to construct the full 32-bit distributions of the S-box approximations, but exploring a spectrum point in time $O(1)$. The total complexity of the analysis is $\approx O(2^{41})$.

We then collected the best pairs $\{(k,\lambda)\}$ in terms of the largest peak spectrum values, and constructed two full-rank 32×32 Boolean matrices K and Λ from the indices (k,λ) with the greedy approach given in Algorithm 2. Then the matrix M was derived as $M = K^{-1}\cdot\Lambda$.

Results. We only used seven pairs from the result of the spectrum analysis (since many other pairs did not give us full-rank matrices K and Λ), and the remaining 25 rows of K,Λ were randomly generated. Thereafter, we tested that matrix M in the full approximation and received the total bias:

$$\epsilon(N_{tot}^{(t_1)}) \approx 2^{-236.380623}.$$

The 32×32 binary matrix M is given below as a vector of 32-bit integers, where the bit $M_{i,j}$, for $0\le i,j\le 31$, is extracted as $M_{i,j} = \lfloor \mathtt{M}[i]/2^j \rfloor \mod 2$, and in standard C it is then $M_{i,j} =$`(M[i]>>j)&1`.

```
uint32_t M[32] =
{  0x26dad00b, 0x5de94454, 0x3bdfdb0d, 0x1423c42f, 0xc4f35585, 0x1f22e504,
   0xeb07cc1e, 0x3633b301, 0x11b4bca3, 0x6f23b103, 0x912adb7d, 0x6a058e9e,
   0x67d4ef5a, 0xdd0830b6, 0xee579099, 0x9af30192, 0x455d8a7b, 0x22133144,
   0x7fb935a8, 0x4d923b96, 0xc0c9967e, 0x99db94fc, 0x442f1154, 0x17994e1f,
   0x08d2662e, 0xccc8fe9c, 0x994d8fb8, 0xfba4f0dc, 0x462d2a69, 0x373306ed,
   0x91282e11, 0x9b82d788 };
```

4.5 A distinguishing attack on ZUC-256

In a distinguishing attack, an adversary aims to find some linear relationships between the generated keystream symbols by canceling the LFSR contribution in the linear approximation, thus being able to distinguish the keystream sequence from random.

In Subsection 4.1, we have shown that if we could find four time instances t_1,t_2,t_3,t_4 such that $s^{(t_1)}+s^{(t_2)} = s^{(t_3)}+s^{(t_4)} \mod p$, we can build the keystream samples $M\sigma[Z^{(t_1)}\oplus$

$Z^{(t_2)} \oplus Z^{(t_3)} \oplus Z^{(t_4)}] \oplus [Z^{(t_1+1)} \oplus Z^{(t_3+1)} \oplus Z^{(t_3+1)} \oplus Z^{(t_4+1)}]$ to be biased with a bias of $2^{-236.38}$. By collecting around $O(2^{236.38})$ such samples, we could distinguish this sample sequence from random, thus resulting in a distinguishing attack.

The remaining problem is how we can find such a time instance tuple t_1, t_2, t_3, t_4 satisfying the requirement, i.e., $s^{(t_1)}+s^{(t_2)} = s^{(t_3)}+s^{(t_4)} \mod p$. This problem is equivalent to finding a weight 4 multiple of the feedback polynomial, for which there already exist a number of research results [LJ14][YJM19]. We use the algorithm in [LJ14] to solve this problem. But here we should find a weight 4 multiple with two coefficients being 1 and the other two being -1. Let us first figure out how far we should run the cipher, i.e., the degree of the multiple, to find such a tuple. Let q be the expected degree. If we consider all $t \leq q$, we could create $\binom{q}{3}$ different combinations of $s^{(t_1)} + s^{(t_2)} - s^{(t_3)} - s^{(t_4)}$ when we fix one time instance. Since there are 2^{496} possible such combinations, we can expect that we need to go to the length such that $\binom{q}{3} \approx 2^{496}$, resulting in $q \approx 2^{167}$. We use the algorithm in [LJ14] to find the time instance tuple, but note that at the last step, instead of keeping $x^{i_1} + x^{i_2} + x^{i_3} + x^{i_4} = 0 \bmod P(x)$, we should keep $x^{i_1} + x^{i_2} - x^{i_3} - x^{i_4} = 0 \bmod P(x)$. The algorithm requires computational complexity of q and similar storage. For our case, the complexity is 2^{167}. The algorithm to find the time instance tuple can be found in Appendix C.

Thus we succeed to have a distinguishing attack on ZUC-256, for which we need to run the cipher around 2^{236} iterations and collect 2^{236} samples.

5 Conclusions

In this paper, we give a number of spectral tools for linear cryptanalysis and further apply them to ZUC-256 resulting in a distinguishing attack on ZUC-256 faster than exhaustive key search.

We explored how a linear masking in the time domain would affect the spectrum points in the frequency domain under some commonly used operations in cryptography, such as $\boxplus, \oplus$, and S-boxes, in both WHT and DFT types. We also gave a number of results and algorithms about how to find a good linear masking in the time domain by aligning the spectrum points in the frequency domain.

For the distinguishing attack, we first derive a linear approximation of the non-linear part F and the transformation from $GF(p)$ field in LFSR to $GF(2^{32})$ in F. We then employ the spectral tools to find good linear maskings and adapt the bit-slicing technique to efficiently compute the bias of the approximation. The linear approximation is then used to launch a distinguishing attack by finding a weight 4 multiple of the generating polynomial to cancel the contribution from the LFSR. The complexity of the distinguishing attack is $O(2^{236})$. It indicates that ZUC-256 does not provide a source with full 256-bit entropy in the generated keystream, as would be expected from a 256-bit key.

Acknowledgments

We would like to thank all reviewers for providing valuable comments to the manuscript. This work was in part financially supported by the Swedish Foundation for Strategic Research, grant RIT17-0005. The author Jing Yang is also supported by the scholarship from the National Digital Switching System Engineering and Technological Research Center, China.

References

[3GP18] 3GPP TSG-SA. Study on the support of 256-bit algorithms for 5G (release 16), November 2018. https://www.3gpp.org/ftp/tsg_sa/WG3_Security/TSGS3_93_Spokane/Docs.

[BJV04] Thomas Baigneres, Pascal Junod, and Serge Vaudenay. How far can we go beyond linear cryptanalysis? In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 432–450. Springer, 2004.

[CT12] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[ETS11a] ETSI/SAGE. Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3. document 2: ZUC specification, 2011.

[ETS11b] ETSI/SAGE. Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3. document 4: Design and evaluation report, 2011.

[GDL13] Jie Guan, Lin Ding, and Shu-Kai Liu. Guess and determine attack on SNOW 3G and ZUC. *Journal of Software*, 6:1324–1333, 2013.

[HCN19] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. Multidimensional linear cryptanalysis. *Journal of Cryptology*, 32(1):1–34, 2019.

[HG97] Helena Handschuh and Henri Gilbert. χ^2 cryptanalysis of the SEAL encryption algorithm. In *International Workshop on Fast Software Encryption*, pages 1–12. Springer, 1997.

[HN12] Miia Hermelin and Kaisa Nyberg. Multidimensional linear distinguishing attacks and Boolean functions. *Cryptography and Communications*, 4(1):47–64, 2012.

[LD16] Yi Lu and Yvo Desmedt. Walsh transforms and cryptographic applications in bias computing. *Cryptography and Communications*, 8(3):435–453, 2016.

[LJ14] Carl Löndahl and Thomas Johansson. Improved algorithms for finding low-weight polynomial multiples in $F_2[x]$ and some cryptographic applications. *Designs, codes and cryptography*, 73(2):625–640, 2014.

[LMVH15] Frédéric Lafitte, Olivier Markowitch, and Dirk Van Heule. SAT based analysis of LTE stream cipher ZUC. *Journal of Information Security and Applications*, 22:54–65, 2015.

[MJ05] Alexander Maximov and Thomas Johansson. Fast computation of large distributions and its cryptographic applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 313–332. Springer, 2005.

[NH07] Kaisa Nyberg and Miia Hermelin. Multidimensional Walsh transform and a characterization of bent functions. In *2007 IEEE Information Theory Workshop on Information Theory for Wireless Networks*, pages 1–4. IEEE, 2007.

[STL10] B Sun, XH Tang, and C Li. Preliminary cryptanalysis results of ZUC. In *Proc. of the Record of the 1st International Workshop on ZUC Algorithm*, 2010.

[Tea18] ZUC Design Team. The ZUC-256 Stream Cipher, 2018. http://www.is.cas.cn/ztzl2016/zouchongzhi/201801/W020180126529970733243.pdf.

[WHN+12] Hongjun Wu, Tao Huang, Phuong Ha Nguyen, Huaxiong Wang, and San Ling. Differential attacks against stream cipher ZUC. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 262–277. Springer, 2012.

[YJM19] Jing Yang, Thomas Johansson, and Alexander Maximov. Vectorized linear approximations for attacks on SNOW 3G. In *27th Annual Fast Software Encryption Conference, FSE 2020*, 2019.

[ZFL11] Chunfang Zhou, Xiutao Feng, and Dongdai Lin. The initialization stage analysis of ZUC v1. 5. In *International Conference on Cryptology and Network Security*, pages 40–53. Springer, 2011.

A The proof of Theorem 7

Case when $s = 0$**.** Given a_1, a_2 and a_3, the value of a_4 would be fixed. Thus, the total number of all possible combinations of a_i's is p^3.

When $a_1 + a_2 = a_3 + a_4 = k$ for $2 \leq k \leq 2p$, the carry value $Q^{(0)} = 0$ for any t. One can get that there are $(k-1)^2$ solutions for the combinations of a_i's when $2 \leq k \leq p+1$, and $(2p+1-k)^2$ solutions when $p+2 \leq k \leq 2p$. Then the probability $Pr\{Q^{(0)} = 0\}$ in this case is calculated as $(1 + 2^2 + ...(p-1)^2 + p^2 + (p-1)^2 + ... + 2^2 + 1)/p^3 = (2p^2+1)/3p^2$.

Similarly, we can get that when $a_1+a_2 = a_3+a_4+p$ or $a_1+a_2+p = a_3+a_4$, which are two equally likely events, the carry values of $Q^{(0)}$ would respectively be $\pm p \mod 2^t = \pm(2^n - 1) \mod 2^t = \pm 1 \mod 2^t$, both with equal probability $(1 - (2p^2+1)/3p^2)/2 = (p^2-1)/6p^2$.

Case when $s \neq 0$**.** Let us define the set $S^{(0)} = \{(a_1, a_2, a_3, a_4) : a_1 + a_2 = a_3 + a_4 \mod p\}$, which corresponds to all p^3 valid combinations of a_i's when $s = 0$, and $S^{(s)} = \{(2^s a_1, 2^s a_2, 2^s a_3, 2^s a_4) : (a_1, a_2, a_3, a_4) \in S^{(0)}\}$ for the case when $s \neq 0$. Clearly, $|S^{(s)}| = |S^{(0)}|$ and each tuple from $S^{(s)}$ also satisfies $2^s a_1 + 2^s a_2 = 2^s a_3 + 2^s a_4 \mod p$, thus, every tuple of $S^{(s)}$ must also be an element of $S^{(0)}$. The mapping $a_i \to 2^s a_i$ is injective since 2^s is invertible modulo p ($2^s \cdot 2^{n-s} = 1 \mod p$). Therefore, we get that $S^{(0)} \to S^{(s)}$ is an injective mapping and the two sets are equal to each other.

Let us pick any tuple $(a_1, a_2, a_3, a_4) \in S^{(0)}$ assuming the case when $s = 0$, then we extract the lower t-bit values of a_i as $A_i^{(0)}$. The corresponding carry value is then derived as $Q^{(0)} = (A_1^{(0)} \boxplus_t A_2^{(0)}) \boxminus_t (A_3^{(0)} \boxplus_t A_4^{(0)}) \mod 2^t$.

Now we observe that with the selected modulus $p = 2^n - 1$, the multiplication $2 \cdot x \mod p$ is just a circular rotation by 1 bit of x to the left. Thus, in the corresponding mapped tuple $(2^s a_1, 2^s a_2, 2^s a_3, 2^s a_4) \in S^{(s)}$ each a_i value is just circularly rotated by s bits to the left. Then the extracted "middle" t-bit values of $2^s a_i$'s, here denoted as $A_i^{'(s)}$'s, are consistent with $A_i^{(0)}$'s. As a consequence, the resulting carry value will also match, $Q'^{(s)} = Q^{(0)}$. I.e., the mapping $S^{(0)} \to S^{(s)}$ is not only injective but also preserves all other properties including the carry values, therefore, the space of carry values and their probabilities for $s \neq 0$ are the same as for the case when $s = 0$.

B Computation of transition matrices for the exampled noise expression given in Equation 17

```c
// M[0] and M[1] for approximation: N = ((A + B - C) ^ (A + C)) - B
#define tau(c1, c2, c3) (((c3 + 1) * 2 + c2) * 3 + c1 + 1)
long long M[2][12][12];
memset(M, 0, sizeof M);

for(int a=0; a<=1; ++a)
for(int b=0; b<=1; ++b)
for(int c=0; c<=1; ++c)
for(int c1in=-1; c1in<=1; ++c1in)
for(int c2in= 0; c2in<=1; ++c2in)
for(int c3in=-1; c3in<=0; ++c3in)
{
        // process subexpression-1 (inner)
        int expr1   = a + b - c + c1in;
        int result1 = expr1 & 1,  c1out  = expr1 >> 1;

        // process subexpression-2 (inner)
        int expr2   = a + c + c2in;
        int result2 = expr2 & 1,  c2out  = expr2 >> 1;

        // process subexpression-3 (outer)
        // (!) note that c1out and c2out are not included
        int expr3   = (result1 ^ result2) - b + c3in;
        int result3 = expr3 & 1,  c3out  = expr3 >> 1;

        // mapping of in/out carries into indices in the range [0..11]
        int in  = tau(c1in , c2in , c3in );
        int out = tau(c1out, c2out, c3out);

        // add 1 combination to the corresponding in/out carries
        M[result3][out][in] += 1;
}
```

C The algorithm to find a multiple of $P(x)$

Algorithm 4 Finding a multiple of $P(x)$ with weight 4 and two nonzero coefficients being 1 and the other two being -1

Input Polynomial $P(x)$, a small integer b

Output A polynomial multiple $K(x) = P(x)Q(x)$ of weight 4 and expected degree 2^d with two of the nonzero coefficients being 1 and the other two being -1

1. From $P(x)$, create all residues $x^{i_1} \bmod P(x)$, for $0 \leq i_1 < 2^{d+b}$ and put $(x^{i_1} \bmod P(x), i_1)$ in a list $\mathcal{L}_1$. Sort $\mathcal{L}_1$ according to the residue value of each entry.

2. Create all residues $x^{i_1} + x^{i_2} \bmod P(x)$ such that $\phi(x^{i_1} + x^{i_2} \bmod P(x)) = 0$, for $0 \leq i_1 < i_2 < 2^{d+b}$ and put in a list $\mathcal{L}_2$. Here $\phi()$ means the d least significant bits. This is done by merging the sorted list $\mathcal{L}_1$ by itself and keeping only residues $\phi(x^{i_1} + x^{i_2} \bmod P(x)) = 0$. The list $\mathcal{L}_2$ is sorted according to the residue value.

3. In the final step we merge the sorted list $\mathcal{L}_2$ with itself to create a list $\mathcal{L}$, keeping only residues $x^{i_1} + x^{i_2} - x^{i_3} - x^{i_4} = 0 \bmod P(x)$, i.e., $x^{i_1} + x^{i_2} = x^{i_3} + x^{i_4} \bmod P(x)$.