Full Round Zero-sum Distinguishers on TinyJAMBU-128 and TinyJAMBU-192 Keyed-permutation in the Known-key setting

Orr Dunkelman[1], Shibam Ghosh[2(✉)] and Eran Lambooij[3]

Department of Computer Science, University Of Haifa, Haifa, Israel
[1]orrd@cs.haifa.ac.il
[2]sghosh03@campus.haifa.ac.il
[3]eran@hideinplainsight.io

Abstract. TinyJAMBU is one of the finalists in the NIST lightweight standardization competition. This paper presents full round practical zero-sum distinguishers on the keyed permutation used in TinyJAMBU. We propose a full round zero-sum distinguisher on the 128- and 192-bit key variants and a reduced round zero-sum distinguisher for the 256-bit key variant in the known-key settings. Our best known-key distinguisher works with 2^{16} data/time complexity on the full 128-bit version and with 2^{23} data/time complexity on the full 192-bit version. For the 256-bit version, we can distinguish 1152 rounds (out of 1280 rounds) in the known-key settings. In addition, we present the best zero-sum distinguishers in the secret-key settings: with complexity 2^{23} we can distinguish 544 rounds in the forward direction or 576 rounds in the backward direction. For finding the zero-sum distinguisher, we bound the algebraic degree of the TinyJAMBU permutation using the monomial prediction technique proposed by Hu *et al.* at ASIACRYPT 2020. We model the monomial prediction rule on TinyJAMBU in MILP and find upper bounds on the degree by computing the parity of the number of solutions.

1 Introduction

Lightweight cryptographic primitives are essential for providing security for highly resource-constrained devices that transmit sensitive information. Thus, recent years have seen a substantial increase in the development of lightweight symmetric cryptographic primitives. As a response, NIST started the lightweight cryptography competition [25] in 2018. The competition aims to standardize lightweight authenticated encryption algorithms and lightweight hash functions. In 2021, NIST announced ten finalists out of the initial 56 candidates.

TinyJAMBU [32,33], proposed by Wu *et al.*, is one of the finalists of the NIST lightweight competition. The design principle of TinyJAMBU follows the sponge duplex mode using a keyed permutation. The core component of TinyJAMBU is a keyed permutation derived from a lightweight NFSR that contains a single NAND gate as the non-linear component.

TinyJAMBU uses two different keyed permutations with the same key K. These are similar in structure but differ only in the number of rounds. We denote these two permutations $\mathcal{P}^a$ and $\mathcal{P}^b$. The permutation $\mathcal{P}^a$ is used in steps where no output is observed. $\mathcal{P}^b$ is used in the first initialization step and when an internal state value is partially leaked. In the original submission of TinyJAMBU, the round number of $\mathcal{P}^a$ was 384 for all variants. The round number of $\mathcal{P}^b$ was 1024, 1152, and 1280 for 128-, 192-, and 256-bit key variants, respectively.

1.1 Existing Analysis on TinyJAMBU Permutation

The designers of TinyJAMBU [32,33] provide a rigorous security analysis of the underlying permutation against various attacks. From the differential and linear attack perspectives, the designers [32] count the least number of active NAND gates to claim security against those attacks. Later it was improved by Saha *et al.* in [23] using (first order) correlated NAND gates. The authors in [23] propose differential characteristics through 338 and 384 rounds of the keyed permutation with probabilities of $2^{-62.68}$ and $2^{-70.68}$, respectively. These attacks lead to a forgery attack in the TinyJAMBU mode. In response to this result, the designers of TinyJAMBU increased the number of rounds of $\mathcal{P}^a$ from 384 to 640 [33]. More recently Dunkelman *et al.* reported related-key forgery attacks on the full TinyJAMBU-192 and TinyJAMBU-256 (after the tweak) [11].

Sibleyras *et al.* [24] discuss slide attack on the full round TinyJAMBU permutations. These reported attacks are key recovery attacks on TinyJAMBU permutation with data/time complexities of $2^{65}, 2^{66}$, and $2^{69.5}$ for 128-, 192-, and 256-bit key variants, respectively.

Regarding algebraic attacks, the designers of TinyJAMBU [32,33] claim that all the ciphertext bits are affected by the input bits after 598 rounds. This statement was supported by a recent work by Dutta *et al.* [12], where the authors bound the degree using the monomial prediction technique [17] and showed that TinyJAMBU is secure against 32-sized cube attacks after 445 rounds. In addition, they propose cube distinguishers in the weak-key setting for 451 rounds and 476 rounds of TinyJAMBU permutation and the size of the weak-key set is 2^{108}. Another cryptanalysis from an algebraic perspective was reported recently by Teng *et al.* [26], where the authors propose cube attacks against TinyJAMBU. The authors propose basic cube distinguisher on 438 rounds TinyJAMBU by considering TinyJAMBU mode as a black-box.

1.2 Our Contributions

This paper aims to study the algebraic properties of TinyJAMBU permutation as a standalone primitive. Notably, we construct zero-sum [2,3,5] distinguishers that distinguish the TinyJAMBU keyed permutation in practical complexity. Zero-sum distinguishers [2,3,5] can be used to distinguish permutations from a random permutation by suggesting a subset of inputs whose corresponding outputs are summed to zero. One way to do so is to bound the algebraic degree of the output bits, as a bit of degree d is balanced over any cube of degree $d+1$. The most precise technique for upper bounding the degree of a Boolean function is the *Monomial prediction technique*, proposed at ASIACRYPT 2020 [17]. The *Monomial prediction technique* recursively predicts the existence of a monomial in the polynomial representation of the output bits. One can model the rules of monomial prediction using automatic tools like MILP [17] or CP/SAT [14].

We use zero-sum distinguishers both in the known-key [20] and the secret-key settings [19,21]. We show that one can bound the degree of an output bit after 544 rounds of the TinyJAMBU permutation by 22 by carefully choosing the cube variables. Furthermore, we show that the degree of the inverse TinyJAMBU permutation increases slower than in the forward direction. Using this, we show that in the backwards direction, a bit can be represented as a degree 22 polynomial after 576 rounds. Combining these results, we then show full round zero-sum distinguishers on 128- and 192-bit key versions and reduced round distinguisher on 256-bit key version in the known key settings. Our best zero-sum distinguishers are summarized in Table 1, which are the best-known algebraic distinguishers on TinyJAMBU permutation till date.

Key size	#Rounds	Complexity	Model	Type	Section
all	480(/1024)	2^{16}	Secret Key	ZS	5
all	544(/1024)	2^{23}	Secret Key	ZS	5.1
128	1024	2^{16}	Known Key	ZS	5.1
192	1152	2^{23}	Known Key	ZS	5.2
256	1152(/1280)	2^{23}	Known Key	ZS	5.2

Table 1: Zero-Sum (ZS) Distinguishers on the full Round TinyJAMBU permutation.

1.3 Paper Structure

In Section 2, we define the notation for the paper as well as giving a short introduction of the monomial prediction technique. We give a short overview of the TinyJAMBU mode and keyed primitive in Section 3. Then, in Section 4, we show how to build a zero-sum distinguisher on the full keyed permutation of TinyJAMBU-128 and TinyJAMBU-192. In Section 5 we show how improve data/time complexity of the zero-sum distinguisher on the full keyed permutation of TinyJAMBU-128 and TinyJAMBU-192 Finally, we conclude the paper in Section 6.

2 Preliminaries

2.1 Notations

The size of a set $\mathbb{S}$ is denoted as $\|\mathbb{S}\|$. The Hamming weight of $a \in \mathbb{F}_2^n$ is defined as $wt(a) = \sum_{i=1}^{i=n} a_i$. We use bold lowercase letters to represent vectors in a binary field. For any n-bit vector $\mathbf{s} \in \mathbb{F}_2^n$, its i-th coordinate is denoted by s_i, thus we have $\mathbf{s} = (s_{n-1}, ..., s_0)$. $\mathbf{0}$ represents the binary vector with all elements being 0. For any vector $\mathbf{u} \in \mathbb{F}_2^n$ and $\mathbf{x} \in \mathbb{F}_2^n$, we define the *bit product* as $\mathbf{x}^{\mathbf{u}} = \prod_{i=1}^{n} {x_i}^{u_i}$.

Let $Y \subseteq \mathbb{F}_2^n$ be a multi-set of vectors. A coordinate position $0 \leq i < n$ is called a *balanced position* if $\bigoplus_{\mathbf{y} \in Y} y_i = 0$.

2.2 Boolean Functions and Upper Bounds on the Degree

An n-variable Boolean function f is a function from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. If f is a Boolean function then there exists a unique multivariate polynomial in

$$\mathbb{F}_2[x_0, x_1, ..., x_{n-1}]/(x_0^2 + x_0, x_1^2 + x_1, ..., x_{n-1}^2 + x_{n-1})$$

such that

$$f(x_0, x_1, ..., x_{n-1}) = \bigoplus_{\mathbf{u} \in \mathbb{F}_2^n} a_{\mathbf{u}}^f \mathbf{x}^{\mathbf{u}}.$$

This multivariate polynomial is called the algebraic normal form (ANF) of f. In this paper, we primarily look at the algebraic degree of a Boolean function. The definition of the algebraic degree of a Boolean function is as follows

Definition 1. *The algebraic degree of a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is d if d is the degree of the monomial with the highest algebraic degree in the ANF of f, i.e.,*

$$d = \max_{\mathbf{u} \in \mathbb{F}_2^n, a_{\mathbf{u}}^f \neq 0} wt(\mathbf{u}).$$

Many well-known attacks such as integral attacks [18], higher-order differential attacks [21], cube attacks [9], and zero sum distinguishers [4] exploit a low degree of a Boolean function. Hence, the problem of finding the degree of a cryptographic function is important in cryptanalysis. Numerous methods for computing (or bounding) the degree of cryptographic Boolean functions have been proposed in the literature. A study on various degree evaluation methods can be found in [7].

Canteaut *et al.* proposed a method for upper bounding the algebraic degree of composite functions at EUROCRYPT 2002 [6], which was improved by Boura et al. [3] at FSE 2011 with applications to the Keccak-f function.

In CRYPTO 2017, Liu [22] proposed the *numeric mapping* technique, which is an approach for upper bounding the algebraic degree of a non-linear feedback shift register based stream ciphers. The main idea of this technique is to estimate the degree of a monomial by computing the sum of degrees of all the variables contained in this monomial. Thus, one can use the degree of previous states to estimate the degree of the current state. Using the *numeric mapping* technique, the author found several zero-sum distinguishers for round-reduced Trivium, Kreyvium, and TriviA-SC .

Another approach for the degree evaluation is based on the division property which was proposed as a generalization of the integral property by Todo at Eurocrypt 2015 [27]. It is used in [28] to offer the first attack on the full MISTY1 cipher. The division property proposed by Todo is word-based, i.e., the propagation of the division property captures information only at the word level.

In FSE 2016, Todo and Morii introduced the bit-based division property [29]. However, the accuracy of this approach is determined by the accuracy of the "propagation rules" of the underlying detection algorithms for division properties. The *monomial prediction* technique, proposed at ASIACRYPT 2020 [17], offers an exact method to do so. We use the monomial prediction technique in this paper to find an upper bound on the degree of a Boolean function. In the following subsection we briefly discuss how to use it to compute the algebraic degree.

2.3 Monomial Prediction

Finding algebraic properties, such as the degree of the vectorial Boolean functions corresponding to cryptographic primitives, is usually very hard regarding computational complexity. One way to determine various algebraic properties of a Boolean function is to determine the exact algebraic structure, i.e., predict the presence of particular monomials in its ANF. Monomial prediction technique [17] is an indirect way to determine the presence of a particular monomial in the bit product function of the output bits. In this section, we discuss an overview of monomial prediction technique and how to find the degree of a Boolean function with this technique.

Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a vectorial Boolean function that maps $\mathbf{x} = (x_{n-1}, ..., x_0) \in \mathbb{F}_2^n$ to $\mathbf{y} = (y_{m-1}, ..., y_0)$ with $y_i = f_i(\mathbf{x})$, where $f_i : \mathbb{F}_2^n \to \mathbb{F}_2$ is a Boolean function called the i-th coordinate of f. The monomial prediction problem is to identify if the monomial $\mathbf{x}^{\mathbf{u}}$ is present or absent as a monomial in the polynomial representation of $\mathbf{y}^{\mathbf{v}}$ for some $\mathbf{u} \in \mathbb{F}_2^n$ and $\mathbf{v} \in \mathbb{F}_2^m$. We denote that the monomial $\mathbf{x}^{\mathbf{u}}$ is present in $\mathbf{y}^{\mathbf{v}}$ by $\mathbf{x}^{\mathbf{u}} \to \mathbf{y}^{\mathbf{v}}$, likewise, $\mathbf{x}^{\mathbf{u}} \nrightarrow \mathbf{y}^{\mathbf{v}}$ denotes that $\mathbf{x}^{\mathbf{u}}$ is not present in $\mathbf{y}^{\mathbf{v}}$.

As most of the Boolean functions that appear in symmetric cryptographic primitives are built as a composition of a sequence of vectorial Boolean functions, the authors of [17] proposed a recursive prediction model. Naturally, the function f can be written as a composition of round functions:

$$\mathbf{y} = f(\mathbf{x}) = f^{(r-1)} \circ f^{(r-2)} \circ \cdots \circ f^{(0)}(\mathbf{x})$$

and $\mathbf{x}^{(i+1)} = f^{(i)}(\mathbf{x}^{(i)})$ for all $0 \leq i < r$. We can represent $\mathbf{x}^{(i)}$ as a function of $\mathbf{x}^{(j)}$ for $j < i$ as $\mathbf{x}^{(i)} = f^{(i-1)} \circ \cdots \circ f^{(j)}(\mathbf{x}^{(j)})$.

Definition 2. *(Monomial Trail [17]) Let* $\mathbf{x}^{(i+1)} = f^{(i)}(\mathbf{x}^{(i)})$ *for* $0 \leq i < r$. *We call a sequence of monomials* $((\mathbf{x}^{(0)})^{\mathbf{u}^{(0)}}, (\mathbf{x}^{(1)})^{\mathbf{u}^{(1)}}, ..., (\mathbf{x}^{(r)})^{\mathbf{u}^{(r)}})$ *an r-round*

monomial trail *connecting* $(\mathbf{x}^{(0)})^{\mathbf{u}^{(0)}}$ *to* $(\mathbf{x}^{(r)})^{\mathbf{u}^{(r)}}$ *with respect to the composite function* $f(\mathbf{x}) = f^{(r-1)} \circ f^{(r-2)} \circ \cdots \circ f^{(0)}(\mathbf{x})$ *if*

$$(\mathbf{x}^{(0)})^{\mathbf{u}^{(0)}} \rightarrow (\mathbf{x}^{(1)})^{\mathbf{u}^{(1)}} \rightarrow \cdots \rightarrow (\mathbf{x}^{(i)})^{\mathbf{u}^{(i)}} \rightarrow \cdots \rightarrow (\mathbf{x}^{(r)})^{\mathbf{u}^{(r)}}$$

If there is at least one monomial trail connecting $(\mathbf{x}^{(0)})^{\mathbf{u}^{(0)}}$ *to* $(\mathbf{x}^{(r)})^{\mathbf{u}^{(r)}}$, *we denote it as* $(\mathbf{x}^{(0)})^{\mathbf{u}^{(0)}} \rightsquigarrow (\mathbf{x}^{(r)})^{\mathbf{u}^{(r)}}$.

It is important to note that there can be multiple monomial trails connecting $(\mathbf{x}^{(0)})^{\mathbf{u}^{(0)}}$ to $(\mathbf{x}^{(r)})^{\mathbf{u}^{(r)}}$. This leads to the definition of a *monomial hull* which is the set of all trails connecting $(\mathbf{x}^{(0)})^{\mathbf{u}^{(0)}}$ and $(\mathbf{x}^{(r)})^{\mathbf{u}^{(r)}}$, which is denoted as $\{(\mathbf{x}^{(0)})^{\mathbf{u}^{(0)}} \looparrowright (\mathbf{x}^{(r)})^{\mathbf{u}^{(r)}}\}$. The size of a *monomial hull* can be found recursively as given in the following Lemma 1 from [17].

Lemma 1. *([17]) For* $r \geq 1$, $(\mathbf{x}^{(\mathbf{0})})^{\mathbf{u}^{(0)}} \rightsquigarrow (\mathbf{x}^{(\mathbf{r})})^{\mathbf{u}^{(r)}}$, *then*

$$\|\{(\mathbf{x}^{(\mathbf{0})})^{\mathbf{u}^{(0)}} \looparrowright (\mathbf{x}^{(\mathbf{r})})^{\mathbf{u}^{(r)}}\}\| =$$
$$\begin{cases} 1, & \text{if } r = 1 \\ \displaystyle\sum_{(\mathbf{x}^{(r-1)})^{\mathbf{u}^{(r-1)}} \rightarrow (\mathbf{x}^{(r)})^{\mathbf{u}^{(r)}}} \|\{(\mathbf{x}^{(0)})^{\mathbf{u}^{(0)}} \looparrowright (\mathbf{x}^{(r-1)})^{\mathbf{u}^{(r-1)}}\}\|, & \text{otherwise} \end{cases}$$

It is easy to observe that the presence or absence of a monomial in $(\mathbf{x}^{(r)})^{\mathbf{u}^{(r)}}$, depends on the parity of the size of the *monomial hull*, which is captured in Proposition 1.

Proposition 1. *([17])* $(\mathbf{x}^{(0)})^{\mathbf{u}^{(0)}} \rightarrow (\mathbf{x}^{(r)})^{\mathbf{u}^{(r)}}$ *if and only if* $\|\{(\mathbf{x}^{(0)})^{\mathbf{u}^{(0)}} \looparrowright (\mathbf{x}^{(r)})^{\mathbf{u}^{(r)}}\}\|$ *is odd.*

To conclude, the problem of finding the degree of a polynomial is reduced to finding the parity of the size of the monomial hull of the monomial with the highest degree.

2.4 Computing the Algebraic Degree Using Monomial Prediction

To determine the algebraic degree d, of a Boolean function f, one can use the *monomial prediction technique* [17]. To compute an upper bound on the degree we only need to prove the existence of a monomial $\mathbf{x}^{\mathbf{u}}$ with $wt(\mathbf{u}) = d$ such that $\mathbf{x}^{\mathbf{u}} \rightarrow f$ and $\mathbf{x}^{\mathbf{u}'} \nrightarrow f$ for all $\mathbf{u}'$ with $wt(\mathbf{u}') > wt(\mathbf{u})$.

To automate this task, the authors in [17] proposed a Mixed Integer Linear Programming (MILP) approach. The core idea is to model the monomial trails of a function with linear inequalities so that only the valid trails satisfy the system. For more information on the MILP modeling, we refer to [15,16,17].

To find the existence of a specific monomial, one can choose a monomial (i.e., the exponent vector $\mathbf{u}$) and check for the feasibility of the MILP model. On the other hand, one can find the maximum degree by setting the objective function of the MILP model to maximize $wt(\mathbf{u})$ according to $\max_{\mathbf{x}^{\mathbf{u}} \rightarrow f} wt(\mathbf{u})$. Finally, to confirm the presence of a monomial, we need to check if the number of solutions is even or odd. To do this, *PoolSearchMode* of the Gurobi[1] solver is used in [17]. The *PoolSearchMode* is implemented by Gurobi solver to systematically search for multiple solutions.

3 The Specification of TinyJAMBU

The TinyJAMBU [32] family of authenticated encryption with associated data (AEAD), is a small variant of JAMBU [31]. It is one of the finalists of the NIST lightweight competition. The design principle of TinyJAMBU is based on the sponge duplex mode using keyed permutations derived from a lightweight NLFSR.

In this paper we call the internal keyed permutation the TinyJAMBU permutation. The round function of the TinyJAMBU permutation is defined as follows: $\mathcal{P}_k : \mathbb{F}_2^{128} \rightarrow \mathbb{F}_2^{128}$ such that for a key bit $k \in \mathbb{F}_2$ and input $\mathbf{x} = (x_{127}, ..., x_0)$, $\mathcal{P}_k(\mathbf{x}) = (z_{127}, ..., z_0)$ where

$$\begin{cases} z_{127} = 1 \oplus k \oplus x_{91} \oplus x_{85}x_{70} \oplus x_{47} \oplus x_0 \\ z_i = x_{i+1}, \text{ for } 0 \leq i \leq 126. \end{cases} \tag{1}$$

We denote the r-round permutation by $\mathcal{P}_{\mathsf{K}}^r$. Given a key $\mathsf{K} = (k_{\kappa-1}, ..., k_0)$ of size κ and a 128-bit input, $\mathcal{P}_{\mathsf{K}}^r$ outputs a 128-bit state by calling the round function r times as follows

$$\mathcal{P}_{\mathsf{K}}^r(\mathbf{x}) = \mathcal{P}_{k_{\kappa-1}} \cdots \circ \mathcal{P}_{k_1} \circ \mathcal{P}_{k_0}(\mathbf{x}),$$

where each subscript i of the key bits are computed as $k_i = k_{i(\mathsf{mod}\ \kappa)}$. The round function of the TinyJAMBU permutation is depicted in Figure 1.

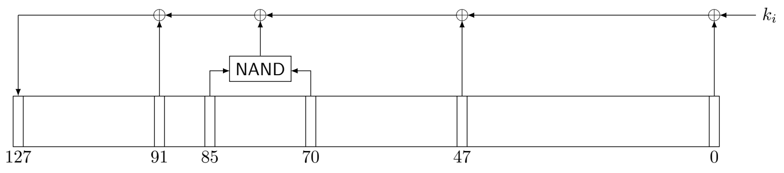

Fig. 1: Round function of the TinyJAMBU permutation.

We define the inverse of $\mathcal{P}_{\mathsf{K}}^r$ as $\mathcal{P}_{\mathsf{K}}^{-r}$, such that for any K and $\mathbf{x}$, $\mathcal{P}_{\mathsf{K}}^{-r}(\mathcal{P}_{\mathsf{K}}^r)(\mathbf{x}) = \mathbf{x}$. We follow the convention that the input to the $\mathcal{P}_{\mathsf{K}}^{-r}$ is rotated, i.e., the i-th input bit becomes $127 - i$ for all $0 \leq i < 128$. The round function of the TinyJAMBU inverse permutation is defined as follows: $\mathcal{P}_k^{-1} : \mathbb{F}_2^{128} \rightarrow \mathbb{F}_2^{128}$ such that for a key bit $k \in \mathbb{F}_2$ and input $\mathbf{z} = (z_{127}, ..., z_0)$, $\mathcal{P}_k(\mathbf{z}) = (x_{127}, ..., x_0)$ where

[1] `https://www.gurobi.com.`

$$\begin{cases} x_{127} = 1 \oplus k \oplus x_{37} \oplus x_{43}x_{58} \oplus x_{81} \oplus x_0 \\ z_i = x_{i+1}, \text{ for } 0 \leq i \leq 126. \end{cases} \tag{2}$$

Given a key $\mathsf{K} = (k_{\kappa-1}, ..., k_0)$, the $\mathsf{TinyJAMBU}$ inverse function is computed as follows

$$\mathcal{P}_{\mathsf{K}}^{-r}(\mathbf{x}) = \mathcal{P}_{k_{\kappa-1}}^{-1} \cdots \circ \mathcal{P}_{k_1}^{-1} \circ \mathcal{P}_{k_0}^{-1}(\mathbf{x}),$$

where each subscript i of the key bits are computed as $k_i = k_{(\kappa-i)(\mathsf{mod}\ \kappa)}$. The round function of the inverse $\mathsf{TinyJAMBU}$ permutation is depicted in Figure 2.

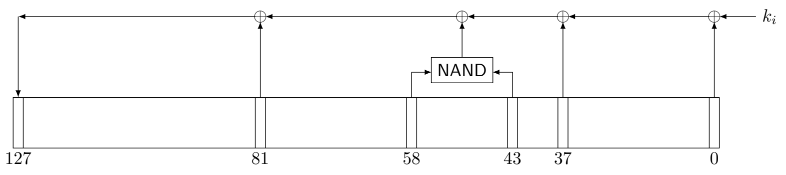

Fig. 2: Round function of the inverse $\mathsf{TinyJAMBU}$ permutation.

$\mathsf{TinyJAMBU}$ has a 32-bit message injection part (rate), a 32-bit squeezing part, and a 96-bit unaltered capacity part. The capacity part of the state is XORed with the 3-bit frame constants denoted by const_i. $\mathsf{TinyJAMBU}$ uses two different keyed permutations using the same key K in different phases of the encryption. We denote them as $\mathcal{P}^a$ and $\mathcal{P}^b$. The only differene between these two permutation is the number of rounds.

$\mathsf{TinyJAMBU}$ has three variants based on key size used in the permutation, whose parameters are listed in Table 2.

Variant	#Rounds $\mathcal{P}^a$	#Rounds $\mathcal{P}^b$	State	Key	Nonce	Tag
TinyJAMBU-128	640	1024	128	128	96	64
TinyJAMBU-192	640	1152	128	192	96	64
TinyJAMBU-256	640	1280	128	256	96	64

Table 2: Parameters of the tweaked $\mathsf{TinyJAMBU}$ [33].

We denote internal state as s. The encryption algorithm of $\mathsf{TinyJAMBU}$ can be divided into the following four phases.

Initialization. In this step $\mathcal{P}^b$ is applied to the 128-bit initial state $s = (0, 0, ..., 0)$ to inject the key into the state. After that, in the nonce setup phase, a 96-bit nonce N is split up into three 32-bit nonce parts $N_0\|N_1\|N_2$ and for each part of the nonce the state is updated with $\mathcal{P}^a$ after which the nonce is added to the most significant part of the state. A depiction of the initialization is given in Figure 3.

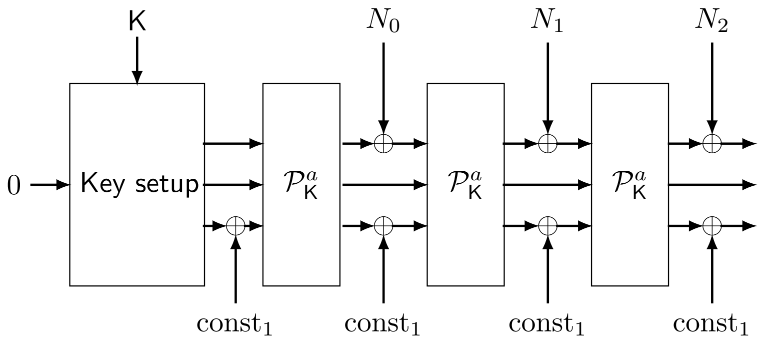

Fig. 3: TinyJAMBU initialization.

Associated Data Processing. The associated data is divided into 32-bit blocks. For each block, the 3-bit frame constant of the associated data phase is XORed with the state and then the state is updated with $\mathcal{P}^a$, after which the 32 bits of the associated data part is XORed with the state. The schematic diagram associated data processing and finalization step is depicted in Figure 4.

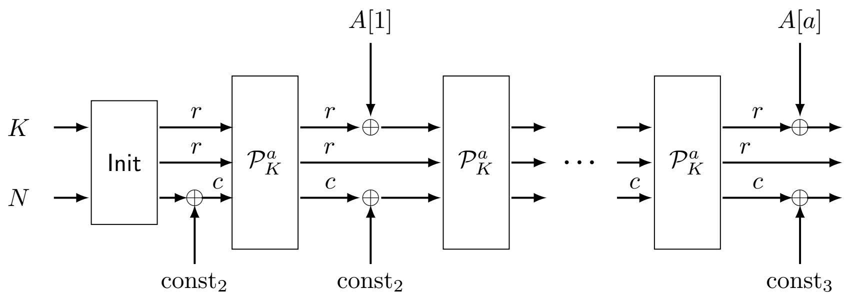

Fig. 4: TinyJAMBU Authenticated Encryption for a blocks of associated data, and m blocks of message.

Encryption. The plaintext is divided into 32-bit blocks. For each block, the frame bits for encryption are XORed into the state. Then the state is updated with $\mathcal{P}^b$, after which the plaintext block is XORed into the most significant part of the state. Finally, we obtain the 32-bit ciphertext block by XORing bits $95 \ldots 64$ of the state with the plaintext block. Note that, the plaintext and nonce are added to the 32 most significant bits of the state which are $127 \ldots 96$ and the key stream used for encryption, is obtained from bits $95 \ldots 64$.

Finalization. After encrypting the plaintext, the 64-bit authentication tag $T_0 \| T_1$ is generated in two steps. First, the frame bits for the finalization are XORed into the state which is followed by the application of $\mathcal{P}^a$ after which 32-bit T_0 is extracted from bit $95 \ldots 64$ of the state. Then, again the frame bits for the finalization are XORed with the state followed by application of $\mathcal{P}^a$ and 32-bit T_1 is extracted. The schematic diagram plaintext processing and finalization step is depicted in Figure 5.

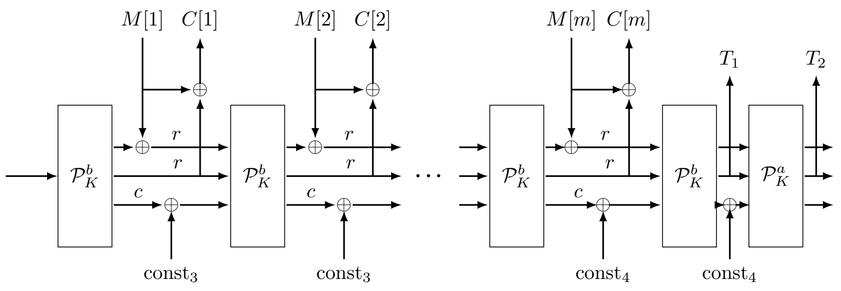

Fig. 5: TinyJAMBU Authenticated Encryption for a blocks of associated data, and m blocks of message.

4 Zero-sum Distinguishers on TinyJAMBU

In this section, we show how to construct a zero-sum distinguisher [2] for the TinyJAMBU permutation. The idea of this distinguisher, as the name 'zero-sum' suggests, is to find a set of inputs and corresponding outputs of an n-bit permutation such that the bits in the inputs and outputs sum to 0 over $\mathbb{F}_2$. In the case of TinyJAMBU we choose an affine vector space ν as a subspace of $\mathbb{F}_2^{128}$ of dimension d. We then show that the polynomial representation of all (or some) of the targeted output bits of the TinyJAMBU permutation have a degree less than d after r rounds. In such a case, the outputs corresponding to all the elements in ν sum to 0 for the targeted bits as the dimension is greater than the degree. In other words, if the algebraic degree of a Boolean function f is less than d, there exists an affine vector space ν of dimension at least d for which $\bigoplus_{\mathbf{x} \in \nu} f(\mathbf{x}) = 0$. The time and data complexity of the attack is $O(2^d)$ and the memory complexity is $O(1)$.

The input set ν usually consists of inputs taking all possible combinations in d input bits and the remaining bits take a fixed value. Thus the input set forms an affine vector space of dimension d. So the resulting output sets are the d-th derivative of the corresponding vectorial Boolean function with respect to ν. Recall that this idea was first proposed as the higher order differential attack [19,21].

In the following we discuss various zero-sum distinguishers on the TinyJAMBU permutation. The basis for all the distinguishers is finding an upper bound on the degree of the polynomial representation of input/output bits. This is done using a MILP model of the monomial prediction trail of the TinyJAMBU permutation.

4.1 MILP Modeling

Let us consider a function $f : \mathbb{F}_2^m \to \mathbb{F}_2^n$ such that $\mathbf{y} = f(\mathbf{x})$. Every pair of $(\mathbf{u}, \mathbf{v})$ is a valid monomial trail through f if and only if $\mathbf{x}^{\mathbf{u}} \to \mathbf{y}^{\mathbf{v}}$. The main motivation of MILP modeling is that it is easy to test validity of a monomial trail using the MILP. Let us consider the following monomial trail

$$(\mathbf{x}^{(0)})^{\mathbf{u}^{(0)}} \to (\mathbf{x}^{(1)})^{\mathbf{u}^{(1)}} \to \cdots \to (\mathbf{x}^{(i)})^{\mathbf{u}^{(i)}} \to \cdots \to (\mathbf{x}^{(r)})^{\mathbf{u}^{(r)}}$$

We consider the transition of the exponents $(\mathbf{u}^{(0))}, \mathbf{u}^{(1))}, ..., \mathbf{u}^{(r))})$ through the round functions and construct a MILP model by modeling the propagation of monomial trails. Any function can be decomposed into smaller operations, namely, COPY, XOR, AND and NOT. We recall the MILP model from [15,16] that supports the following operations.

<u>COPY [15].</u> Consider the function $\mathsf{COPY} : \mathbb{F}_2 \to \mathbb{F}_2^m$ such that $\mathsf{COPY}(a) = (a, a, ..., a)$, i.e., one bit is copied to m bits. Let $(u, (v_1, v_2, ..., v_m))$ denote the monomial trail through the COPY function, then, it can be represented using the following MILP constraints:

$$\begin{cases} v_1 + v_2 + \cdots + v_m \geq u \\ u \geq v_i, \forall i \in \{1, 2, ..., m\} \\ u, v_1, ..., v_m \text{ are all binary variables.} \end{cases}$$

<u>XOR [15].</u> Consider the function $\mathsf{XOR} : \mathbb{F}_2^m \to \mathbb{F}_2$ such that $\mathsf{XOR}(a_1, ..., a_m) = a_1 \oplus \cdots \oplus a_m$. Let $((u_1, u_2, ..., u_m), v)$ denote the monomial trail through the XOR function, then, it can be represented using the following MILP constraints:

$$\begin{cases} u_1 + u_2 + \cdots + u_m - v = 0 \\ u_1, ..., u_m, v \text{ are all binary variables.} \end{cases}$$

<u>AND [15].</u> Consider the function $\mathsf{AND} : \mathbb{F}_2^m \to \mathbb{F}_2$ such that $\mathsf{AND}(a_1, ..., a_m) = a_1 a_2 \cdots a_m$. Let $((u_1, u_2, ..., u_m), v)$ denote the monomial trail through the AND function, then, it can be represented using the following MILP constraints:

$$\begin{cases} v = u_i, \forall i \in \{1, 2, ..., m\} \\ u_1, ..., u_m, v \text{ are all binary variables.} \end{cases}$$

NOT [16]. Consider the function $\mathsf{NOT} : \mathbb{F}_2 \to \mathbb{F}_2$ such that $\mathsf{NOT}(a) = 1 \oplus a$. Let (u, v) denote the monomial trail through the NOT function, then, it can be represented using the following MILP constraint:

$$\begin{cases} v \geq u \\ u, v \text{ are binary variables.} \end{cases}$$

4.2 MILP model for the Monomial Trails of TinyJAMBU

We now discuss a MILP model to capture the valid monomial trails through the TinyJAMBU permutation. We denote the i-th intermediate state of the TinyJAMBU permutation and the TinyJAMBU inverse permutation as $\mathbf{x}^{(i)} = (x^i_{127}, \ldots, x^i_0)$ and $\bar{\mathbf{x}}^{(i)} = (\bar{x}^i_{127}, \ldots, \bar{x}^i_0)$ for $0 \leq i \leq r$, respectively. Thus the input (state) variables are $\mathbf{x}^{(0)} = (x^0_{127}, \ldots, x^0_0)$, and the output (state) variables after r-rounds are $\mathbf{x}^{(r)} = (x^r_{127}, \ldots, x^r_0)$. We let $\mathbf{u}^{(i)} = (u^i_{127}, \ldots, u^i_0)$ and $\bar{\mathbf{u}}^{(i)} = (\bar{u}^i_{127}, \ldots, \bar{u}^i_0)$ denote the exponents of the i-th intermediate state of the TinyJAMBU permutation and the TinyJAMBU inverse permutation, respectively. Also the exponents corresponding to the key variables are denoted by $(u^0_{128+\kappa}, \ldots, u^0_{128})$.

To represent the feedback polynomial of TinyJAMBU permutation and its inverse, we define the function $\mathsf{CORE} : \mathbb{F}_2^5 \to \mathbb{F}_2^5$ such that

$$\mathsf{CORE}(x_5, x_4, x_3, x_2, x_1, x_0) = (y_5, y_4, y_3, y_2, y_1, y_0)$$

where

$$(y_5, y_4, y_3, y_2, y_1, y_0) = \begin{cases} y_5 = x_5 \\ y_4 = x_4 \\ y_3 = x_3 \\ y_2 = x_2 \\ y_1 = x_1 \\ y_0 = 1 \oplus x_0 \oplus x_1 \oplus x_2 x_3 \oplus x_4 \oplus x_5. \end{cases} \tag{3}$$

The way we model the i-th round works in two steps. At the first step the CORE function is applied to $(k^i_i, x^i_{91}, x^i_{85}, x^i_{70}, x^i_{47}, x^i_0)$, i.e.,

$$(k^{i+1}_i, x^{i+1}_{91}, x^{i+1}_{85}, x^{i+1}_{70}, x^{i+1}_{47}, x^{i+1}_0) = \mathsf{CORE}(k^i_i, x^i_{91}, x^i_{85}, x^i_{70}, x^i_{47}, x^i_0).$$

In the second step all the state variables and key variables are rotated, i.e., $x^{i+1}_j = x^{i+1}_{j+1 \bmod 128}$ and $k^{i+1}_j = k^{i+1}_{j+1 \bmod 128}$. To construct a monomial trail of the round function of TinyJAMBU permutation, we decompose the CORE function given in Equation 3 into COPY, XOR, AND and NOT operations. Then we can model the monomial trails through the CORE function as shown in Table 3, by introducing 7 intermediate variables w_i for $i = 0, 1, ..., 7$. In Table 3, u_i's and v_i's represent the MILP variables to denote exponents of x_i's and y_i's respectively. For the second step we just rotate the indices. In Algorithm 1, we discuss how to generate a complete MILP model for TinyJAMBU permutation, where $\mathcal{L}$ represents the inequalities for the CORE function as given in Table 3.

The model for the inverse TinyJAMBU permutation works almost similarly. The only difference is the input to the CORE function. In the case of the inverse round, the CORE function is applied as follows

$$(k_i^{i+1}, x_{81}^{i+1}, x_{58}^{i+1}, x_{43}^{i+1}, x_{37}^{i+1}, x_0^{i+1}) = \mathsf{CORE}(k_i^i, x_{81}^i, x_{58}^i, x_{43}^i, x_{37}^i, x_0^i).$$

operation	trail	MILP constraints
$x_1 \xrightarrow{\mathsf{COPY}} (z_1, y_1)$	$(u_1, (w_1, v_1))$	$u_1 \leq w_1,\ u_1 \leq v_1,\ w_1 + v_1 \geq u_1$
$x_2 \xrightarrow{\mathsf{COPY}} (z_2, y_2)$	$(u_2, (w_2, v_2))$	$u_2 \leq w_2,\ u_2 \leq v_2,\ w_2 + v_2 \geq u_2$
$x_3 \xrightarrow{\mathsf{COPY}} (z_3, y_3)$	$(u_3, (w_3, v_3))$	$u_3 \leq w_3,\ u_3 \leq v_3,\ w_3 + v_3 \geq u_3$
$x_4 \xrightarrow{\mathsf{COPY}} (z_4, y_4)$	$(u_4, (w_4, v_4))$	$u_4 \leq w_4,\ u_4 \leq v_4,\ w_4 + v_4 \geq u_4$
$x_5 \xrightarrow{\mathsf{COPY}} (z_5, y_5)$	$(u_5, (w_5, v_5))$	$u_5 \leq w_5,\ u_5 \leq v_5,\ w_5 + v_5 \geq u_5$
$(z_2, z_3) \xrightarrow{\mathsf{AND}} z_6$	$((w_2, w_3), w_6)$	$w_6 == z_2,\ w_6 == z_3$
$(x_0, z_1, z_6, z_4, z_5) \xrightarrow{\mathsf{XOR}} z_7$	$((u_0, w_1, w_6, w_4, w_5), w_7)$	$w_7 == u_0 + w_1 + w_6 + w_4 + w_5$
$z_7 \xrightarrow{\mathsf{NOT}} y_0$	(w_7, v_0)	$v_0 \geq w_7$

Table 3: Inequalities to represent the CORE function.

4.3 Degree Estimation of the TinyJAMBU Permutation

We now bound the degree of TinyJAMBU permutation using the MILP model generated in Algorithm 1. In the following discussion, we consider the key $\mathsf{K} = (k_{\kappa-1}, ..., k_0)$ as a constant. Thus, the j-th state bit after i rounds of $\mathcal{P}_{\mathsf{K}}$, x_j^i, is a polynomial on $\mathbf{x}^{(0)} = (x_{127}^0, \ldots, x_0^0)$, and we need to determine the degree of the polynomial over $\mathbf{x}^{(0)} = (x_{127}^0, \ldots, x_0^0)$. In all of our attacks, we fix our target output (state) bit to the 127-th bit and thus for an r-round permutation $\mathbf{u}^{(r)} = (u_{127}^r, ..., u_0^r)$ is

$$u_j^r = \begin{cases} 1, & \text{if } j = 127 \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

We discuss the detailed procedures to find the degree of the TinyJAMBU permutation in Algorithm 2, where we call Algorithm 1 to generate a MILP model for the TinyJAMBU. Algorithm 2 works as follows: we first generate a MILP model $\mathcal{M}$ and solve the model to get a possible monomial $\mathbf{m}$ in the polynomial representation of the 127-th bit of the output state. Our next task is to confirm the presence of the monomial $\mathbf{m}$ in the polynomial representation of the 127-th bit of the output state. To do so we generate another MILP model $\mathcal{M}'$ (line 11

Algorithm 1 TinyJAMBU$_{MILP}$ (Generating a MILP model)

```
1: Input: $r$, the targeted number of rounds
2: Output: The MILP model $\mathcal{M}$ for $r$-round TinyJAMBU permutation, MILP
   variable representing initial and final states
3: Declare an empty MILP model $\mathcal{M}$;
4: $\mathcal{M}.var \leftarrow \mathbf{u}^{(0)} = (u^0_{127+\kappa}, \dots, u^0_{128}, u^0_{127}, \dots, u^0_0)$;
5: $\mathcal{M}.var \leftarrow \mathbf{s} = (s_{127+\kappa}, \dots, s_0)$;
6: $\mathbf{s} \leftarrow \mathbf{u}^{(0)}$;
7: for $i = 0; i < r; i = i + 1$ do
8:     $\mathcal{M}.var \leftarrow (w_7, \dots, w_0)$;
9:     $\mathcal{M}.var \leftarrow (v_{128}, v_{91}, v_{85}, v_{70}, v_{47}, v_0)$;
10:    $\mathcal{M}.con \leftarrow \mathcal{L}(s_{128}, s_{91}, s_{85}, s_{70}, s_{47}, s_0, v_{128}, v_{91}, v_{85}, v_{70}, v_{47}, v_0, w_7, \dots, w_0)$;
11:    $s_i = v_i, i \in \{0, 47, 70, 85, 91, 128\}$;
12:    for $j = 0; j < 128; j = j + 1$ do
13:        $s_j \leftarrow s_{j+1 \mod 128}$;
14:    for $j = 128; j < \kappa; j = j + 1$ do
15:        $s_j \leftarrow s_{128+\{(j+1) \mod \kappa\}}$;
16: $\mathbf{u^r} \leftarrow \mathbf{s}$
17: return $\mathcal{M}, \mathbf{u}^0, \mathbf{u}^r$;
```

in Algorithm 2). Here we set the initial variables ($\mathbf{u}'^{(0)}$) of $\mathcal{M}'$ according to the solution of $\mathcal{M}$ and solve it. In this case we count the number of solutions of $\mathcal{M}'$. If the number of solutions is odd, then we confirm that the obtained monomial $\mathbf{m}$ exists (line 19 in Algorithm 2) in the polynomial representation of the 127-th bit of the output state. Otherwise, if the number of solution is even, then the number of trails from the monomial $\mathbf{m}$ is even. So, we remove the solution corresponding to $\mathbf{m}$ (line 22-26 in Algorithm 2) and solve the model $\mathcal{M}$ again. We do this until we get a monomial with an odd number of trails.

We have listed our findings of degree for TinyJAMBU-128 permutation and its inverse permutation in Table 7 and Table 8, respectively, in Appendix A. The source code for MILP modeling can be found in
https://github.com/ShibamCrS/zeroSumDistinguishersOnTinyJambu.git.

4.4 Basic Zero-sum Distinguisher

We now discuss a basic zero-sum distinguisher on TinyJAMBU-128. To find this we upper bound the degree of TinyJAMBU permutation using Algorithm 2. We set the objective function to maximize the sum of initial variables, i.e., $\sum_{j=0}^{127} u_j^0$ and check for the parity of the number of solutions as in Algorithm 2. With the help of Algorithm 2 we are able to evaluate an upper bound on the algebraic degree of the TinyJAMBU-128 up to 333 rounds. The degree of all the bits in $\mathbf{x}^{(333)}$ is upper-bounded by 38 for 333 rounds. Thus, if we consider an affine subspace ν of dimension 39 and consider the sum of the output states over all

Algorithm 2 Degree estimation of the TinyJAMBU permutation

```
1: Input: $r$, the targeted number of rounds
2: Output: The degree $d$ of $r$-round TinyJAMBU permutation
3: $(\mathcal{M}, \mathbf{u}^{(0)}, \mathbf{u}^{(r)}) \leftarrow \mathsf{TinyJAMBU}_{MILP}(r)$;
4: $\mathcal{M}.constraint \leftarrow \{u^r_{127} = 1\}$;
5: for $i = 0; i < 127; i = i + 1$ do
6:     $\mathcal{M}.constraint \leftarrow \{u^r_i = 0\}$;
7: $\mathcal{M}.objective \leftarrow \max(u^0_{127} + \ldots + u^0_0)$;
8: while true do
9:     $\mathcal{M}.update()$;
10:    $\mathcal{M}.optimize()$;
11:    if $\mathcal{M}.status$ is OPTIMAL then
12:        $d \leftarrow \mathcal{M}.objvalue$
13:        $(\mathcal{M}', \mathbf{u}'^{(0)}, \mathbf{u}'^{(r)}) \leftarrow \mathsf{TinyJAMBU}_{MILP}(r)$;
14:        $u^r_{127} \leftarrow 1$;
15:        for $i = 0; i < 127; i = i + 1$ do
16:            $u^r_i \leftarrow 0$;
17:        for $i = 0; i < 128; i = i + 1$ do
18:            ${u'}^0_i \leftarrow u^0_i.value$;
19:        $\mathcal{M}'.optimize()$;
20:        if $\mathcal{M}'.status$ is OPTIMAL then
21:            if Number of solution in $\mathcal{M}'$ is odd then
22:                return $d$;
23:            else
24:                for $i = 0; i < 127 + \kappa; i = i + 1$ do
25:                    if $u^0_i.value = 0$ then
26:                        $\mathcal{M}.constraint \leftarrow \{u^0_i = 1\}$;
27:                    else
28:                        $\mathcal{M}.constraint \leftarrow \{u^0_i = 0\}$;
```

the elements of ν, we get $\mathbf{0}$, i.e.,

$$\bigoplus_{\mathbf{x}^{(0)} \in \nu} \mathcal{P}_{\mathsf{K}}^{333}(\mathbf{x}^{(0)}) = \mathbf{0}.$$

Which gives us that all the state bits are balanced after 333 rounds.

We can extend this distinguisher to cover more rounds due to the basic nature of the shift register. From the design of the round function, we can observe that each bit is shifted to its previous bit, i.e., $x_j^i = x_{j+1}^i$. Thus after 448 rounds the following 12 bits are still balanced

$$(x_{11}^{(448)}, x_{10}^{(448)}, \ldots x_0^{(448)}).$$

The complexity of this distinguisher is 2^{39}. When the same affine subspace passes through a random permutation, the outputs satisfy the above condition with probability 2^{-12}. Thus, the distinguishing advantage of this distinguisher $1 - 2^{-12}$.

We can increase the distinguishing advantage by considering ℓ affine subspace of dimension 39 instead of taking one affine subspace. This comes at the cost of complexity. In this case the the distinguishing advantage of this distinguisher is $1 - 2^{-12\ell}$ and the complexity is $\ell 2^{39}$. Let us take $\ell = 4$. Then the distinguishing advantage increases to $1 - 2^{-48}$ and complexity of this distinguisher is 2^{41}.

We can increase the number of rounds up to 460 and still can use this zero-sum property. However, the distinguishing advantage of the attack decreases. If $\mathcal{A}$ is the distinguishing advantage, then the number of rounds $r \propto \log(1 - \mathcal{A})$, where $333 \le r \le 460$.

Zero-sum Distinguisher of the Inverse Permutation We now discuss a basic zero-sum distinguisher on the inverse permutation. Similar to the forward direction, here also we set the objective function to maximize $\sum_{j=0}^{127} \bar{u}_j^0$ and to check for the parity of the number of solutions. We noticed in our experiments that in the inverse direction the degree increases significantly slower compared to the forward direction. This is because the update function has only one non-linear operation on bit number 70 and 85 and for the inverse permutation, the first non-linear operation is after 70 rounds, while in the forward direction the first non-linear operation occurs after 42 rounds.

With the help of monomial prediction, we are able to evaluate an upper bound on the algebraic degree of the inverse TinyJAMBU-128 up to 502 rounds. The degree of all the bits in $\bar{\mathbf{x}}^{(502)}$ is upper-bounded by 41 for 502 rounds. Thus if we consider an affine subspace ν of dimension 42 and consider the sum of the output states over all the elements of ν, we get $\mathbf{0}$, i.e.,

$$\bigoplus_{\mathbf{x}^{(0)} \in \nu} \mathcal{P}_{\mathsf{K}}^{-502}(\bar{\mathbf{x}}^{(0)}) = \mathbf{0}.$$

As in the forward direction, we can extend this distinguisher for more rounds using shifting property of the register. After 608 rounds the following 21 bits are still balanced

$$(\bar{x}_{20}^{(608)}, \bar{x}_{19}^{(608)}, ..., \bar{x}_{0}^{(608)}).$$

The data and time complexity of this distinguisher is 2^{42} and the distinguishing advantage of this distinguisher $1 - 2^{-21}$. Similar to the forward direction, we can increase the distinguishing advantage by considering few more subspaces of dimension 41. If we consider 4 subspaces of dimension 41, the distinguishing advantage increases to $1 - 2^{-84}$ and complexity of this distinguisher is 2^{44}.

4.5 Extending to Full Rounds Using Inside-out Approach

We extend the above zero-sum distinguishers to a full round distinguisher using the inside-out approach. We start in the middle of the permutation and compute the degree outwards. We consider an affine subspace ν of dimension d and for all 2^d possible intermediate states we compute the outputs. Suppose that we consider the state after r_1 rounds for an r-round permutation where $r_1 + r_2 = r$. For all these 2^d intermediate states $\mathbf{x}^{(\mathbf{r_1})} \in \mathbb{F}_2^{128}$, we compute $\mathcal{P}^{r_2}(\mathbf{x}^{(\mathbf{r_1})})$ and $\mathcal{P}^{-r_1}(\mathbf{x}^{(\mathbf{r_1})})$. If the degree of both functions $\mathcal{P}^{r_2}$ and $\mathcal{P}^{-r_1}$ are less than d, we get zero-sum on both outputs, i.e.,

$$\bigoplus_{\mathbf{x}^{(\mathbf{r_1})} \in \nu} \mathcal{P}^{r_2}(\mathbf{x}^{(\mathbf{r_1})}) = \bigoplus_{\mathbf{x}^{(\mathbf{r_1})} \in \nu} \mathcal{P}^{-r_1}(\mathbf{x}^{(\mathbf{r_1})}) = \mathbf{0}.$$

The idea is depicted in Figure 6.

$$\begin{tikzcd}
\text{zero-sum} & \nu \arrow[l, "deg(\mathcal{P}^{-r_1}) < d"'] \arrow[r, "deg(\mathcal{P}^{r_2}) < d"] & \text{zero-sum}
\end{tikzcd}$$

Fig. 6: Zero-sum distinguisher from an intermediate vector space ν.

However, to use this distinguisher to distinguish the TinyJAMBU permutation from a random permutation, we need to know the secret key K, i.e., this is a *known-key zero-sum* distinguisher. This cryptographic model, distinguishing a cryptographic permutation from a random permutation with the knowledge of the key, was introduced by Knudsen *et al.* in [20] at ASIACRYPT 2007. The authors in [20] found a distinguisher on the 7-round AES [8] using this model. There are other examples of this approach in the literature, for example on KECCAK and Luffa [3,5], PHOTON [30], Ascon [10] or MiMC [13].

To apply the technique to the TinyJAMBU-128 permutation, we bound the degree of $\mathcal{P}^{r_2}$ and $\mathcal{P}^{-r_1}$ using a MILP model of the monomial prediction rules. We find that the degree of all the bits of $\mathbf{x}^{(332)}$ is upper-bounded by 37 after applying 332 rounds of $\mathcal{P}$. On the other hand, the degree of all the bits of $\bar{\mathbf{x}}^{(482)}$ is upper bounded by 37 for round number 482 of $\mathcal{P}^{-1}$. We set $r_1 = 448$ and $r_2 = 576$ giving $r = r_1 + r_2 = 1024$. If we take $\mathcal{P}^{448}(\mathbf{x}^{(\mathbf{576})}) =$

$(z_{127}, ..., z_0)$, then we get the zero-sum at the bit positions $(z_{10}, z_9, ..., z_0)$. Similarly, if $\mathcal{P}^{-576}(\mathbf{x^{(576)}}) = (y_{127}, ..., y_0)$, then we get the zero-sum at the following 33 bit positions $(y_{32}, y_{31}, ..., y_0)$. Thus we consider an affine vector space ν of dimension 38 and compute outwards by setting the intermediate state to all possible 2^{38} vectors from ν. We collect the outputs of $\mathcal{P}^{448}$ and the outputs of $\mathcal{P}^{-576}$. Finally, we have a set of inputs $X = \{\mathbf{x} = (x_{127}, ..., x_0)\}$ with $\bigoplus_{\mathbf{x}\in X} x_i = 0$ for all $0 \leq i \leq 32$. Also, a set of outputs $Z = \{\mathcal{P}^{1024}(\mathbf{x}) : \mathbf{x} \in X\}$ such that $\bigoplus_{\mathbf{x}\in X} z_i = \bigoplus_{\mathbf{x}\in X} (\mathcal{P}^{1024}(\mathbf{x}))_i = 0$ for all $0 \leq i \leq 10$. Combining the two results we have a zero-sum distinguisher of full round TinyJAMBU-128 permutation with time and data complexity $O(2^{38})$. See Figure 7 for a depiction of the attack.

$$\text{zero-sum} \xleftarrow[deg(\mathcal{P}^{-576}) < 38]{\qquad\qquad} \nu \xrightarrow[deg(\mathcal{P}^{448}) < 38]{\qquad\qquad} \text{zero-sum}$$

Fig. 7: Zero-sum distinguisher on full TinyJAMBU-128.

5 Improved Zero-sum Distinguisher

In the previous section, we suggested a distinguisher on the full round TinyJAMBU-128 permutation by considering degree of the TinyJAMBU permutation and inverse permutation over all 128 state variables. In this section we improve the time and data complexity of the distinguisher by choosing the proper variables (or in other words a proper affine subspace ν), i.e., we want to find the degree of the TinyJAMBU permutation over some selected variables. This scenario is also similar to the cube attack [9] or a cube tester [1].

Given a Boolean polynomial f in m variables, we choose a set of variables (also called the cube variables in a cube attack) of size n and set the rest of the $m - n$ variables to a constant (typically they are set to zero). We reduced the function f to a function on n-variables. If we can show that the reduced function has degree lower than some value d, we have a zero-sum property.

Proper Choice of Variables Let us consider the following subspace ν of dimension 23 from $\mathbb{F}_2^{128}$

$$\mathbf{x} \in \nu \iff \begin{cases} x_{i+47} = x_i, \text{ if } 0 \leq i \leq 22 \\ 0, \text{ if } 23 \leq i \leq 46 \\ 0, \text{ if } 70 \leq i \leq 127. \end{cases} \tag{5}$$

From the algebraic description of the round permutation we get that after one round, the feedback polynomial is $\mathbf{x}_{127}^{(1)} = 1 \oplus x_0^{(0)} \oplus x_{47}^{(0)} \oplus x_{70}^{(0)} x_{85}^{(0)} \oplus x_{91}^{(0)} \oplus k_0$. If we set $\mathbf{x}^{(0)}$ to any vector from ν, we have $\mathbf{x}_{127}^{(1)} = 1 \oplus k_0$. This is also true for first 23 rounds. Therefore, for the first 23 rounds the degree of the feedback polynomial is 0 in the cube variables. Then, from round 24 to 47, the degree of the feedback polynomials is also 0 in the cube variables. It is now clear how the locations of the fixed 0 bits were chosen - to ensure that as many forward rounds with degree 0 as possible. We conclude the results with the following Lemma 2.

Lemma 2. *If we choose the input state* $\mathbf{x}^{(0)}$ *of the TinyJAMBU permutation from the subspace* ν *as described in* (5), *we get* $deg(x_{127}^{(i)}) = 0$ *for all* $0 \le i \le 47$. *More precisely, if* $\mathcal{P}_{\mathsf{K}}^{47}(\mathbf{x}^{(0)}) = \mathbf{z}$, *then the coordinates of* $\mathbf{z}$ *satisfies the following conditions:*

$$\begin{aligned} z_i &= x_i^{(0)} \textit{ for } 0 \le i \le 22 \\ z_i &= 0 \textit{ for } 22 \le i \le 80 \\ z_i &= 1 \oplus k_{81-i}, \textit{ for } 81 \le i \le 117 \\ z_i &= k_{155-i} \oplus k_{118-i}, \textit{ for } 118 \le i \le 127 \end{aligned}$$

After 47 rounds, the degree is upper bounded by 1 and state bits 23 to 128 do not contain any cube variables. So we do not consider these MILP variables in the objective function.

After 388 rounds the degree in all bits of $\mathbf{x}^{(388)}$ is upper bounded by 22. Consequently, as we have started after 47 rounds, we can compute the degree after 47 more rounds. The degree in all bits of $\mathbf{x}^{(435)}$ after 435 rounds is upper bounded by 22. Using the subspace ν mentioned in (5), after applying the permutation for 544 rounds, i.e., $\mathcal{P}^{(544)}$, we get a zero sum on the following 18 bit positions

$$(x_{17}^{(544)}, x_{16}^{(544)}, ..., x_0^{(544)}).$$

Similar distinguishers also works on TinyJAMBU-192. The results are summarized in Table 4.

key size	#rounds	#balanced bits	complexity
128	480	16	16
128	480	38	18
128	512	9	18
128	544	18	23
128	448	12	39*
192	490	6	16
192	555	8	23
256	490	6	16
256	555	8	23

Table 4: Secret-key zero-sum distinguishers TinyJAMBU permutation. $*$ implies the basic distinguisher of sec 4.4.

5.1 Extending to Full Rounds Using Inside-out Approach

Similar to the basic distinguisher, we can extend this distinguisher to the full round TinyJAMBU-128 without increasing time or data complexity. The advantage we get for forward direction from projecting degree for 47 rounds at the starting, is not possible for the inverse permutation. However, as we discussed earlier, the degree of the inverse permutation already increases slowly.

For the inverse permutation we set the MILP variable as follows $\bar{u}_i = 0$ if $0 \leq i \leq 58$ or $81 \leq i \leq 105$. After 472 rounds we have that the degree in all the bits of $\bar{\mathbf{x}}^{(472)}$ are upper bounded by 22. Thus, after 576 rounds of the inverse permutation we get a zero sum on the following 23 bit positions

$$(\bar{x}_{22}^{(576)}, \bar{x}_{21}^{(576)}, \dots \bar{x}_{0}^{(576)}).$$

Similar distinguishers also works on TinyJAMBU-192. The results are summarized in Table 5.

key size	#rounds	#balanced bits	complexity
128	544	6	16
128	544	16	18
128	576	23	23
128	592	7	23
128	608	21	42*
192	544	6	16
192	597	4	23
256	544	6	16
256	597	4	23

Table 5: Secret-key zero-sum distinguishers TinyJAMBU inverse permutation. $*$ implies the basic distinguisher of sec 4.4.

To extend the above results to full rounds we set $r_1 = 512$ and $r_2 = 512$, and we have a zero-sum distinguisher on full rounds with 50 balanced bits on the output of the forward direction and 87 balanced bits on the output of the inverse direction. Other combinations of r_1 and r_2 are possible with the trade-off on data/time complexity to number of balanced bits. Our best distinguisher on full TinyJAMBU-128 works with the time/data complexity of 2^{16}. For these we set $r_1 = 480$ and $r_2 = 544$ such that $r_1 + r_2 = 1024$. We get 16 balanced bits in the forward direction and 6 balanced bits in the backward direction. Results on full round attack with various combinations of r_1 and r_2 are given in Table 6. Also see Figure 8 for a depiction of the attack.

$$\begin{tikzcd}\text{zero-sum} & \mathcal{V} \arrow[l, "deg(\mathcal{P}^{-512}) < 23"'] \arrow[r, "deg(\mathcal{P}^{512}) < 23"'] & \text{zero-sum}\end{tikzcd}$$

Fig. 8: Improved zero-sum distinguisher on full TinyJAMBU-128.

5.2 Attack on TinyJAMBU-192 and TinyJAMBU-256

The improved distinguisher discussed in the previous of this section also works on the full round TinyJAMBU-192 and reduced round TinyJAMBU-256 in a similar manner. The results are summarized in Table 6.

5.3 Experimental Verification

We conducted experiments to verify the existence of the zero-sum distinguishers. All of our distinguishers on TinyJAMBU-128 are verified with the reference implementation of TinyJAMBU. The source codes of the verification can be found in:

https://github.com/ShibamCrS/zeroSumDistinguishersOnTinyJambu.git.

key size	#rounds		#balanced bits		complexity
	forward	inverse	after $\mathcal{P}$	after $\bar{\mathcal{P}}$	
128	480	544	16	6	16
128	480	544	38	16	18
128	512	512	50	87	23
128	448	576	11	33	38^*
192	555	597	8	4	23
256	555	597	8	4	$23^\dagger$

Table 6: Known-key zero-sum distinguishers on full round TinyJAMBU-128 and TinyJAMBU-192 and reduced round TinyJAMBU-256. $*$ implies basic distinguisher of sec 4.4. † implies reduced rounds.

6 Conclusion

We discussed full round zero-sum distinguishers for the TinyJAMBU permutation, based on the algebraic properties of the permutation. All the distinguishers have practical complexities, that allowed for complete experimental verification of the attacks.

One important note is that the keyed permutation is the main contribution of the TinyJAMBU submission to the NIST lightweight competition. Our attack is not possible inside the mode. Nevertheless, as we have shown, the keyed permutation used in TinyJAMBU is easily distinguished in the known-key setting. Combining this with the fact that the mode is proven to be secure under the assumption that the internal keyed permutation is robust could lead to problems in the future.

References

1. Aumasson, J., Dinur, I., Meier, W., Shamir, A.: Cube testers and key recovery attacks on reduced-round MD6 and Trivium. In: Symmetric Cryptography. Dagstuhl Seminar Proceedings, vol. 09031. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany (2009)
2. Aumasson, J.P., Meier, W.: Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. rump session of Cryptographic Hardware and Embedded Systems-CHES **2009**, 67 (2009)
3. Boura, C., Canteaut, A.: Zero-sum distinguishers for iterated permutations and application to Keccak-f and Hamsi-256. In: Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 6544, pp. 1–17. Springer (2010)
4. Boura, C., Canteaut, A.: A zero-sum property for the Keccak-f permutation with 18 rounds. In: ISIT. pp. 2488–2492. IEEE (2010)
5. Boura, C., Canteaut, A., Cannière, C.D.: Higher-order differential properties of Keccak and *Luffa*. In: FSE. Lecture Notes in Computer Science, vol. 6733, pp. 252–269. Springer (2011)
6. Canteaut, A., Videau, M.: Degree of composition of highly nonlinear functions and applications to higher order differential cryptanalysis. In: Advances in Cryptology – Proceedings of EUROCRYPT. Lecture Notes in Computer Science, vol. 2332, pp. 518–533. Springer (2002)
7. Chen, S., Xiang, Z., Zeng, X., Zhang, S.: On the relationships between different methods for degree evaluation. IACR Trans. Symmetric Cryptol. **2021**(1), 411–442 (2021)
8. Daemen, J., Rijmen, V.: AES and the wide trail design strategy. In: Advances in Cryptology – Proceedings of EUROCRYPT. Lecture Notes in Computer Science, vol. 2332, pp. 108–109. Springer (2002)
9. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Advances in Cryptology – Proceedings of EUROCRYPT. Lecture Notes in Computer Science, vol. 5479, pp. 278–299. Springer (2009)
10. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Cryptanalysis of Ascon. In: Topics in Cryptology - CT-RSA. pp. 371–387 (2015)
11. Dunkelman, O., Lambooij, E., Ghosh, S.: Practical related-key forgery attacks on the full tinyjambu-192/256. Cryptology ePrint Archive, Paper 2022/1122 (2022), `https://eprint.iacr.org/2022/1122`, `https://eprint.iacr.org/2022/1122`
12. Dutta, P., Rajas, M., Sarkar, S.: Weak-keys and key-recovery attack for TinyJAMBU (05 2022). https://doi.org/10.21203/rs.3.rs-1646044/v1
13. Eichlseder, M., Grassi, L., Lüftenegger, R., Øygarden, M., Rechberger, C., Schofnegger, M., Wang, Q.: An algebraic attack on ciphers with low-degree round functions: Application to full MiMC. In: Advances in Cryptology – Proceedings of ASIACRYPT (1). Lecture Notes in Computer Science, vol. 12491, pp. 477–506. Springer (2020)

14. Hadipour, H., Eichlseder, M.: Integral cryptanalysis of WARP based on monomial prediction. IACR Trans. Symmetric Cryptol. **2022**(2), 92–112 (2022)
15. Hao, Y., Leander, G., Meier, W., Todo, Y., Wang, Q.: Modeling for three-subset division property without unknown subset - improved cube attacks against Trivium and Grain-128AEAD. In: Advances in Cryptology – Proceedings of EUROCRYPT (1). Lecture Notes in Computer Science, vol. 12105, pp. 466–495. Springer (2020)
16. Hebborn, P., Lambin, B., Leander, G., Todo, Y.: Lower bounds on the degree of block ciphers. In: Advances in Cryptology – Proceedings of ASIACRYPT (1). Lecture Notes in Computer Science, vol. 12491, pp. 537–566. Springer (2020)
17. Hu, K., Sun, S., Wang, M., Wang, Q.: An algebraic formulation of the division property: Revisiting degree evaluations, cube attacks, and key-independent sums. In: Advances in Cryptology – Proceedings of ASIACRYPT (1). Lecture Notes in Computer Science, vol. 12491, pp. 446–476. Springer (2020)
18. Knudsen, L., Wagner, D.: Integral cryptanalysis. In: Proceedings of FSE 2002. Lecture Notes in Computer Science, vol. 2365, pp. 112–127. Springer
19. Knudsen, L.R.: Truncated and higher order differentials. In: FSE. Lecture Notes in Computer Science, vol. 1008, pp. 196–211. Springer (1994)
20. Knudsen, L.R., Rijmen, V.: Known-key distinguishers for some block ciphers. In: Advances in Cryptology – Proceedings of ASIACRYPT. Lecture Notes in Computer Science, vol. 4833, pp. 315–324. Springer (2007)
21. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Communications and Cryptography: Two Sides of One Tapestry. pp. 227–233. Springer (1994)
22. Liu, M.: Degree evaluation of NFSR-based cryptosystems. In: Advances in Cryptology – Proceedings of CRYPTO (3). Lecture Notes in Computer Science, vol. 10403, pp. 227–249. Springer (2017)
23. Saha, D., Sasaki, Y., Shi, D., Sibleyras, F., Sun, S., Zhang, Y.: On the security margin of TinyJAMBU with refined differential and linear cryptanalysis. IACR Trans. Symmetric Cryptol. **2020**(3), 152–174 (2020)
24. Sibleyras, F., Sasaki, Y., Todo, Y., Hosoyamada, A., Yasuda, K.: Birthday-bound slide attacks on TinyJAMBU's keyed permutation for all key sizes. In: Fifth Lightweight Cryptography Workshop (2022)
25. Technology, N.: Report on Lightweight Cryptography: NiSTIR 8114. CreateSpace Independent Publishing Platform (2017)
26. Teng, W.L., Salam, M.I., Yau, W., Pieprzyk, J., Phan, R.C.: Cube attacks on round-reduced TinyJAMBU. IACR Cryptol. ePrint Arch. p. 1164 (2021)
27. Todo, Y.: Structural evaluation by generalized integral property. In: Advances in Cryptology – Proceedings of EUROCRYPT (1). Lecture Notes in Computer Science, vol. 9056, pp. 287–314. Springer (2015)
28. Todo, Y.: Integral cryptanalysis on full MISTY1. J. Cryptol. **30**(3), 920–959 (2017)
29. Todo, Y., Morii, M.: Bit-based division property and application to Simon family. In: FSE. Lecture Notes in Computer Science, vol. 9783, pp. 357–377. Springer (2016)
30. Wang, Q., Grassi, L., Rechberger, C.: Zero-Sum Partitions of PHOTON Permutations, Lecture Notes in Computer Science, vol. 10808, pp. 279–299. Springer (2018)
31. Wu, H., Huang, T.: The JAMBU Lightweight Authentication Encryption Mode (v2.1). Submission to CAESAR (2016), `https://competitions.cr.yp.to/round3/jambuv21.pdf`
32. Wu, H., Huang, T.: TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms: Submission to NIST LwC (2019), `https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/tinyjambu-spec-final.pdf`

33. Wu, H., Huang, T.: TinyJAMBU : A family of lightweight authenticated encryption algorithms (version 2) (2021), `https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/tinyjambu-spec-final.pdf`

A The Algebraic Degree of **TinyJAMBU**-128 Permutation and its Inverse

offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0+		1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
20+	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
40+	2	2	2	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
60+	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
80+	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5
100+	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
120+	5	5	5	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	7
140+	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
160+	8	8	8	8	8	8	8	8	8	8	8	8	8	9	9	9	10	10	10	10
180+	10	10	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	12	12	12
200+	12	12	12	12	12	12	12	12	12	12	13	13	13	13	13	13	14	14	14	15
220+	15	15	15	15	15	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
240+	17	17	17	17	17	17	17	17	17	17	17	17	17	18	18	18	19	19	19	20
260+	20	20	21	22	22	22	22	22	22	22	23	24	24	24	24	24	24	25	25	25
280+	25	25	25	26	26	26	26	26	26	26	26	27	27	27	27	27	28	28	28	29
300+	29	29	30	30	30	31	31	31	31	31	31	32	33	33	34	34	34	35	35	35
320+	36	36	36	36	36	36	36	36	36	37	37	37	37	38						

Table 7: Degree of the 127-th bit of TinyJAMBU-128 permutation up to 333 rounds.

offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0+		1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
20+	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
40+	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
60+	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	3
80+	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
100+	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	4	4
120+	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
140+	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
160+	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
180+	5	5	5	5	5	5	5	5	6	6	6	6	6	6	6	6	6	6	6	6
200+	6	6	6	6	6	6	6	6	6	6	6	7	7	7	7	7	7	7	7	7
220+	7	7	7	7	7	7	7	7	7	7	7	7	8	8	8	8	8	8	8	8
240+	8	8	8	8	8	8	8	9	9	9	9	9	9	9	9	9	9	9	10	10
260+	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
280+	10	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	12	12	12	12
300+	12	12	12	12	12	12	12	12	12	12	12	13	13	13	13	13	13	14	14	14
320+	14	14	14	14	14	14	14	14	15	15	15	15	15	15	15	15	15	15	15	15
340+	15	15	15	15	15	15	15	15	15	16	16	16	16	16	16	16	16	16	16	16
360+	16	16	16	16	17	17	18	19	19	19	19	19	19	20	20	20	20	20	20	20
380+	20	20	20	20	20	20	20	21	21	21	22	22	22	22	22	22	23	23	24	24
400+	24	24	24	24	24	24	24	24	24	24	24	24	24	25	25	25	25	25	25	25
420+	26	26	26	26	26	26	26	26	26	26	26	26	26	26	27	27	28	29	29	29
440+	29	29	29	30	30	31	31	31	31	31	31	31	31	31	31	31	31	32	32	32
460+	32	32	32	32	33	33	34	34	35	35	35	36	36	36	36	36	36	36	37	37
480+	37	37	37	38	38	38	39	39	39	39	40	40	40	40	40	40	41	41	41	41
500+	41	41	41																	

Table 8: Degree of the 127-th bit of TinyJAMBU-128 inverse permutation up to 502 rounds.