Integral Attacks on Pyjamask-96 and Round-Reduced Pyjamask-128

Jiamin Cui[1,2], Kai Hu[1,2], Qingju Wang[3], and Meiqin Wang[1,2](✉)

[1] School of Cyber Science and Technology, Shandong University, Qingdao 266237, Shandong, China.
cuijiamin@mail.sdu.edu.cn,hukai@mail.sdu.edu.cn,mqwang@sdu.edu.cn,
[2] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Qingdao, Shandong, China
[3] SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg.
qingju.wang@uni.lu

Abstract. In order to provide benefits in the areas of fully homomorphic encryption (FHE), multi-party computation (MPC), post-quantum signature schemes, or efficient masked implementations for side-channel resistance, reducing the number of multiplications has become a quite popular trend for the symmetric cryptographic primitive designs. With an aggressive design strategy exploiting the extremely simple and low-degree S-box and low number of rounds, Pyjamask, the fundamental block cipher of the AEAD with the same name, has the smallest number of AND gates per bit among all the existing block ciphers (except LowMC or Rasta which work on unconventional plaintext/key sizes). Thus, although the AEAD Pyjamask stuck at the second round of the NIST lightweight cryptography standardization process, the block cipher Pyjamask itself still attracts a lot of attention. Not very unexpectedly, the low degree and the low number of rounds are the biggest weakness of Pyjamask. At FSE 2020, Dobraunig et al. successfully mounted an algebraic and higher-order differential attack on full Pyjamask-96, one member of the Pyjamask block cipher family. However, the drawback of this attack is that it has to use the full codebook, which makes the attack less appealing. In this paper, we take integral attacks as our weapon, which are also sensitive to the low degree. Based on a new 11-round integral distinguisher found by state-of-the-art detection techniques, and combined with the relationship between round keys that reduces the involved keys , we give the key recovery attack on the full Pyjamask-96 without the full codebook for the first time. Further, the algebraic and higher-order differential technique does not work for Pyjamask-128, the other member of the Pyjamask block cipher family. To better understand the security margin of Pyjamask-128, we present the first third-party cryptanalysis on Pyjamask-128 up to 11 out of 14 rounds.

Keywords: Pyjamask, Lightweight cipher, Integral Attack, Division Property, Monomial Prediction

1 Introduction

Recently, block ciphers implemented in resource-constrained environments have received a lot of attention with the increasing deployments of small computing devices. To meet such trend, NIST initiated a lightweight cryptography (LWC) competition for developing a standard of lightweight cryptographic algorithms.

Pyjamask, as an authenticated encryption with associated data (AEAD) scheme which targets at side-channel resistance [12], has been selected as one of the second round candidates for NIST LWC competition. The mode of operation chosen by the authors is OCB AEAD [17]. The underlying block ciphers contain two versions: Pyjamask-96 and Pyjamask-128, named according to the block size. To allow efficient masked implementations, especially for high-order masking, the S-box layer aggressively minimizes costs in terms of AND gates, which also leads to relatively slow growth in algebraic degree. At FSE 2020, Dobraunig et al. [10] noticed this vulnerability and presented an 11.5-round[4] higher-order differential distinguisher. Combined with the solution of linearized systems for monomials and the guess-and-determine strategy, they proposed a chosen-ciphertext attack for the full-round Pyjamask-96 using the whole codebook. Later, Tian and Hu [22] found a 10-round integral distinguisher based on the division property [23,26] that leads to an 11-round attack of Pyjamask-96 with time complexity $2^{93.8}$. So far, there is no third-party analysis for Pyjamask-128.

The integral attack was firstly proposed by Daemen et al. [7] to evaluate the security of the block cipher Square, and later formalized by Knudsen and Wagner [16]. It mainly has two steps: the construction of an integral distinguisher followed by a key recovery step. In order to construct an integral distinguisher, a structure of plaintexts is chosen firstly and encrypted for a few rounds. If the corresponding state has an integral property, an integral distinguisher is obtained, which can be used to perform the key recovery attack with many techniques.

Detecting integral distinguishers. Currently, the division property, proposed as a generalized integral property by Todo at EUROCRYPT 2015 [24], is the most efficient and accurate method of detecting the integral distinguishers. It can better exploit the algebraic degree information and identify balanced output bits. Its powerfulness was undoubtedly demonstrated by the break of full MISTY1 [23]. However, the original division property is word-oriented, i.e., it only exploits the algebraic degree of the non-linear exponents, and could not utilize the internal structure of ciphers in a fine-grained way. So Todo and Morii introduced the bit-based division property at FSE 2016 [26], including the conventional bit-based division property and the three-subset bit-based division

[4] This 11.5-round distinguisher works under the chosen-ciphertext scenario, so the last `MixRows` can be removed naturally, i.e., without the `MixRows` operation, the distinguisher is actually 11 full rounds. Our 11-round distinguisher (introduced later) works under the chosen-plaintext setting, so we cannot remove the `MixRows` operation for the distinguisher. However, equivalently, in the key-recovery phase, we can ignore the last `MixRows` operation also.

property. In [29], Wang et al. showed that the three-subset bit-based division property could be used to recover the exact superpoly in the cube attack. This observation was soon refined by Hao et al. in [13] as the three-subset bit-based division property without unknown subsets (3SDPwoU). Recently, Hebborn et al. pointed out that the idea behind the 3SDPwoU from the perspective of the parity set [5] is that the 3SDPwoU actually determined the existence or absence of a certain monomial in the polynomial of the cipher output. At ASIACRYPT 2020, Hu et al. [15] proposed the concept of the monomial prediction, which is another language for the division properties from the viewpoint of the polynomial directly. By counting the so-called monomial trails, they are able to determine if a monomial of the plaintext or IV appears in the polynomial of the cipher output. Indeed, the monomial prediction and the 3SDPwoU were proved to be equivalent [15].

In practice, searching for the division properties or the monomial trails is time and memory consuming where the complexity is usually the exponential function of the block size. At ASIACRYPT 2016 [30], Xiang et al. introduced the Mixed Integral Linear Programming (MILP) models for the conventional division properties and thereafter the MILP models have been the dominant tool in this area. Later, the integral attacks on dozens of symmetric primitives are then improved [28,27,18,6,8].

Techniques used in the key recovery attacks. When mounting the key recovery attack, we guess some involved round keys, partially decrypt the corresponding ciphertexts backwards to the tail of the integral distinguishers and check whether the summation of the statements is zero. Since the presentation of the integral attack, many refined key-recovery techniques are presented. The partial sum technique was one of the most important tools, which was introduced by Ferguson et al. in [11] to improve the time complexity of integral attacks. The statement after the distinguisher, the ciphertexts and the involved round keys can be separated into several parts and considered independently. Thus, we can reuse some derived partial sum and optimize the time complexity. The original attack target was AES. They dramatically reduced the complexity of the 6-round attacks on AES by a factor of 2^{28}. Recently, this powerful method has lead to a significant reduction in the time complexity of the integral attack on the full MISTY1 [23,4].

Motivation. In order to provide benefits in the areas of fully homomorphic encryption (FHE), multi-party computation (MPC), or post-quantum signature schemes, reducing the number of multiplications has become a quite popular trend for the symmetric cryptographic primitive designs. Unlike earlier designs such as LowMC [3], or Rasta [9] and many others that follow an unconventional design approach, e.g., incomplete non-linear layer, Pyjamask follows a classical design approach to benefit from the mature cryptanalysis methods and security arguments. Within this design space, it aggressively reduces the number of rounds of the internal block ciphers and makes Pyjamask one of the existing members with the smallest number of AND gates per bit processed. Moreover, the low number of nonlinear building blocks of Pyjamask allows side-channel countermeasures. Although the AEAD Pyjamask was not selected as the finalists of the NIST LWC competition, the block cipher Pyjamask still attracts a lot of attention because of its low multiplication complexity characteristic. The most important side effect of the low number of multiplicative operations is the slow growth of algebraic degree, which makes it vulnerable to the attacks such as the higher-order differential attacks [10] and the integral attacks [22]. However, the only attack on the full Pyjamask-96 has to take the whole codebook. We are naturally interested in whether we can improve the attacks without using the whole codebook. The algebraic degree of Pyjamask-128 grows much faster than Pyjamask-96, which the technique in [10] heavily relies on. As a result, their technique fails when applied to Pyjamask-128. On the other hand, block ciphers with 128-bit block size are more popular since almost all protocols support 128-bit block ciphers (for the compatibility with AES) rather than 96-bit ones. Thus, it is of special significance to study the security margin of this 128-bit version of Pyjamask. Thanks to the development of the division properties recently, we have more powerful tools to detect the integral distinguishers for Pyjamask. Thus the security of Pyjamask can be better evaluated by more refined integral attacks.

Our contributions. In this paper, we take state-of-the-art techniques for detecting division properties to give a more fine-grained study of the security strength of Pyjamask-96 and Pyjamask-128 against the integral attacks. To find more integral properties, we construct the MILP models for the encryption algorithm considering the effect of the round keys for Pyjamask simultaneously. 11- and 9-round integral distinguishers are established for Pyjamask-96 and Pyjamask-128, respectively. Based on the new 11-round integral distinguisher for Pyjamask-96, capturing the relationship between the round keys that dramatically reduces the number of involved subkey bits in the key recovery phase from 165 to 121 bits, we manage to attack the full Pyjamask-96 without the full codebook for the first time. Equipped with the partial sum and equivalent key skills, the complexities for some fewer rounds of Pyjamask-96 are also improved. We also give the first third-party cryptanalysis on Pyjamask-128 up to 11 (out of 14) rounds, which helps to better understand its security margin. All the results are summarized in Table 1. The full version of the paper can be found in [1].

Outline. The rest of this paper is organized as follows. In Section 2, we introduce some background knowledge needed in this paper. The technique of obtaining the integral distinguishers is described in Section 3. In Section 4, we mainly describe the key recovery attack on 13-round and 14-round Pyjamask-96. The attack on Pyjamask-128 is present in Section 5. Finally, the paper is concluded in Section 6.

2 Preliminaries

2.1 Pyjamask Block Cipher Family

Pyjamask is a family of the block ciphers used by the AEAD Pyjamask, one of the second-round candidates of the NIST LWC competition. In the remaining

Table 1: Comparisons of attack results for Pyjamask-96 and Pyjamask-128

Instance	Approach	#Round	Data	Time	Reference
Pyjamask-96	Higher-order	9/14	2^{71} CC	2^{67}	[10]
	Integral	**9/14**	**2^{66} CP**	**2^{66}**	**Section 4**
	Higher-order	10/14	2^{87} CC	2^{83}	[10]
	Integral	**10/14**	**2^{81} CP**	**2^{81}**	**Section 4**
	Higher-order	11/14	2^{95} CC	2^{91}	[10]
	Integral	11/14	2^{93} CP	$2^{93.8}$	[22]
	Integral	**11/14**	**2^{90} CP**	**2^{90}**	**Section 4**
	Higher-order	12/14	2^{96} CC	2^{96}	[10]
	Integral	**12/14**	**2^{93} CP**	**2^{93}**	**Section 4**
	Higher-order	13/14	2^{96} CC	2^{99}	[10]
	Higher-order	13/14	2^{94} CC	2^{125}	[10]
	Integral	**13/14**	**2^{95} CP**	**$2^{95.2}$**	**Section 4.1**
	Higher-order	14/14	2^{96} CC	2^{115}	[10]
	Integral	**14/14**	**2^{95} CP**	**$2^{122.6}$**	**Section 4.2**
Pyjamask-128	**Integral**	**5/14**	**2^{20} CP**	**2^{23}**	**Section 5**
	Integral	**6/14**	**2^{52} CP**	**2^{54}**	**Section 5**
	Integral	**7/14**	**2^{90} CP**	**$2^{91.5}$**	**Section 5**
	Integral	**8/14**	**2^{90} CP**	**$2^{107.8}$**	**Section 5**
	Integral	**9/14**	**2^{112} CP**	**2^{112}**	**Section 5**
	Integral	**10/14**	**2^{122} CP**	**2^{122}**	**Section 5**
	Integral	**11/14**	**2^{127} CP**	**2^{127}**	**Section 5**

CC – chosen ciphertext; CP – chosen plaintext

sections, Pyjamask is always the name of the block ciphers without ambiguity. Pyjamask includes two members Pyjamask-96 and Pyjamask-128 named according to the block sizes. Both versions take the same 128-bit length key and have the same 14 rounds. The internal states are represented as rectangles with t rows and 32 columns, where $t = 3$ for Pyjamask-96 and $t = 4$ for Pyjamask-128.

In the specification [12], the round function of Pyjamask consists of three steps: `AddRoundKey`, `SubBytes` and `MixRows`. After 14 rounds of iterations, one additional `AddRoundKey` is appended at last. In this paper, we equivalently regard `SubBytes`, `MixRows` and `AddRoundKey` as one full round (see Figure 1) and thus there is a whitening `AddRoundKey` before the first round function. The operations in one round are described as follows,

- `SubBytes`. The same t-bit S-box is applied to each of the 32 columns of the internal state in parallel ($t = 3$ for Pyjamask-96 and $t = 4$ for Pyjamask-128). For `SubBytes`$^{-1}$, the inverse of the S-box is applied. S_3 and S_4 used in Pyjamask-96 and Pyjamask-128 respectively are given by the following table.

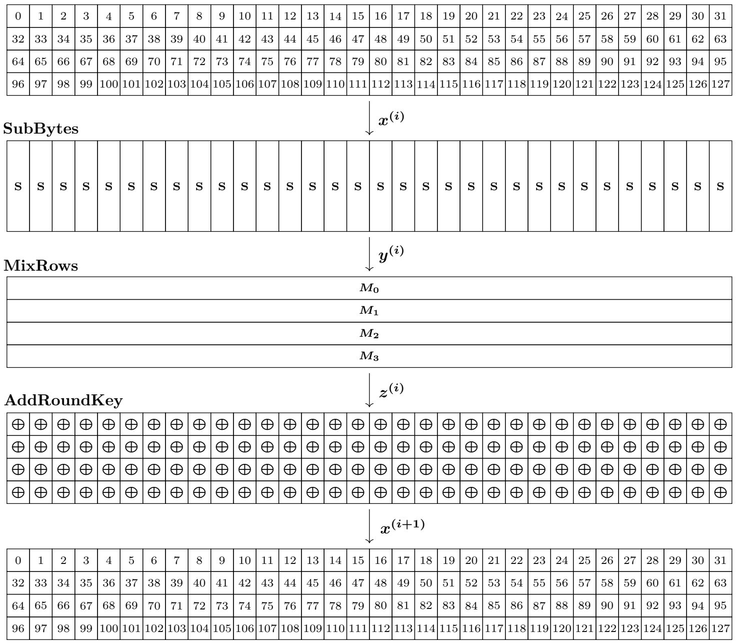

Fig. 1: The round function of Pyjamask-128

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_3(x)$	1	3	6	5	2	4	7	0	–	–	–	–	–	–	–	–
$S_4(x)$	2	D	3	9	7	B	A	6	E	0	F	4	8	5	1	C

- `MixRows`. For each row R_i of the internal state where $0 \leq i < 4$, the updated state can be calculated by $\boldsymbol{M_i} \cdot R_i^T$. The binary matrices $\boldsymbol{M_i}$ used in the `MixRows` layer are 32×32 circulant matrices defined as follows:

$$\boldsymbol{M_0} = \mathtt{cir}([1,1,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,1,1,1,0,0,0,1,0]),$$
$$\boldsymbol{M_1} = \mathtt{cir}([0,1,0,0,0,0,1,0,0,0,0,0,0,1,1,1,0,1,0,0,0,0,0,1,0,1,1,0,0,0,1,1]),$$
$$\boldsymbol{M_2} = \mathtt{cir}([0,0,0,0,0,0,0,0,1,0,1,0,0,1,1,1,1,0,0,1,1,0,1,0,0,1,0,0,1,0,1,1]),$$
$$\boldsymbol{M_3} = \mathtt{cir}([0,1,1,0,0,1,0,0,0,0,0,0,1,0,0,1,0,1,0,1,0,0,1,0,1,0,0,0,1,0,0,1]),$$

  where $\boldsymbol{M_0}$, $\boldsymbol{M_1}$, $\boldsymbol{M_2}$ are used for both Pyjamask-96 and Pyjamask-128, and $\boldsymbol{M_3}$ is only used in Pyjamask-128. The function `cir` generates a matrix where the ith row equals the input vector rotated by i positions to the right. The hamming weight of each row is 11 for $\boldsymbol{M_0}$, $\boldsymbol{M_1}$, $\boldsymbol{M_3}$ and 13 for $\boldsymbol{M_2}$. For $\texttt{MixRows}^{-1}$, the matrices are also circulant and the Hamming weight of each row is 11, 13, 11, 15 for $\boldsymbol{M}_0^{-1}$, $\boldsymbol{M}_1^{-1}$, $\boldsymbol{M}_2^{-1}$, $\boldsymbol{M}_3^{-1}$, respectively.
- `AddRoundKey`. An n-bit round key $k^{(i)}$ for the ith round is extracted from the key schedule and XORed to the internal state, where $0 \leq i \leq 13$ and

$n = 96, 128$. The pre-whitening key (which is also the master key as shown later) is denoted by $k^{(-1)}$.

Key schedule. Pyjamask-96 and Pyjamask-128 share a similar key schedule, where only the sizes of the subkeys differ due to the extra row in Pyjamask-128. Let $k^{(-1)}$ be the master key. The same round function is operated on $k^{(-1)}$ for 14 times, and the round keys $k^{(i)}$ for $0 \leq i \leq 13$ are generated. Note that the key state in each round is loaded into the 128-bit key state in the same ordering as the internal state of the encryption.

- `MixColumns`. Update each column of the key state by the same matrix $\boldsymbol{M}$, where

$$\boldsymbol{M} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

  and $\boldsymbol{M} = \boldsymbol{M}^{-1}$.
- `MixAndRotateRows`. Update the first row of the key state R_0 by $\boldsymbol{M}_k \cdot R_0^T$. The circular matrix $\boldsymbol{M}_k$ is defined as:

$$\boldsymbol{M_k} = \texttt{cir}([1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0]).$$

  The hamming weight of each row is 15 for $\boldsymbol{M_k}$ and 11 for $\boldsymbol{M}_k^{-1}$. R_1, R_2, R_3 are left-rotated by 8, 15 and 18 positions. Namely, they are replaced by $R_1 \lll 8$, $R_2 \lll 15$ and $R_3 \lll 18$, respectively.
- `AddConstant`. Four-byte constants are added bitwisely to different bytes in the key state. The first 28-bit constant is denoted in hexadecimal notation as `0x243f6a8` while the extra 4-bit constant is the binary representation of the round index i.

2.2 Notations

In this paper, the state matrix of Pyjamask at the beginning of round i is denoted by $x^{(i)}$ where $0 \leq i \leq 13$. The state matrix after the `SubBytes` and the `MixRows` operations of round i are denoted by $y^{(i)}$ and $z^{(i)}$, respectively. Bits of every state are labeled by $0, 1, 2, \ldots, 95$ for Pyjamask-96 and $0, 1, \ldots, 127$ for Pyjamask-128, illustrated in Figure 1. The jth row vector of the binary matrix $\boldsymbol{M_i}$ is denoted by $\boldsymbol{M_i}[j]$, $0 \leq j < 32$. We denote the subkey of round i by $k^{(i)}$, and the first (pre-whitening) key by $k^{(-1)}$. In the key-recovery process, we are interested in swapping the order of the `MixRows` operation and the `AddRoundKey`. As these operations are linear or affine they can be interchanged, by first XORing the data with an equivalent subkey and then applying the `MixRows` operation. We denote the equivalent subkey for the altered version by $u^{(i)}$, i.e., $u^{(i)} = \texttt{MixRows}^{-1}(k^{(i)})$. When we interchange the order of the `MixRows` operation of round i and the `AddRoundKey`, we denote the state right after the `AddRoundKey` (and just before the `MixRows` operation) by $\bar{z}^{(i)}$.

2.3 Monomial Prediction

The monomial prediction is a new technique proposed by Hu et al. in [15] to determine whether a monomial appears in any product of the coordinate functions of a vectorial boolean function f. It provides a new perspective from the polynomial for the division properties [26,13]. Let $\boldsymbol{f}\colon \mathbb{F}_2^{n_0} \to \mathbb{F}_2^{n_r}, \boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ be a composite vectorial Boolean function of a sequence of smaller vectorial Boolean functions $\boldsymbol{f}^{(i)}\colon \mathbb{F}_2^{n_i} \to \mathbb{F}_2^{n_{i+1}}, \boldsymbol{x}^{(i+1)} = \boldsymbol{f}^{(i)}(\boldsymbol{x}^{(i)}), 0 \leq i \leq r-1$, i.e.,

$$\boldsymbol{f} = \boldsymbol{f}^{(r-1)} \circ \boldsymbol{f}^{(r-2)} \circ \ldots \circ \boldsymbol{f}^{(0)}.$$

We use the notation $\pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)})$ to represent a monomial of $\boldsymbol{x}^{(i)}$ related to $\boldsymbol{u}^{(i)}$, where $\pi_{\boldsymbol{u}}(\boldsymbol{x})$ stands for $\prod x_i^{u_i}$ and x_i, u_i is the ith coordinate of the vector $\boldsymbol{x}, \boldsymbol{u}$, respectively. Note that $\pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)})$ is also a Boolean function of the $\boldsymbol{x}^{(j)}$ for $j < i$. If the ANF of $\boldsymbol{f}^{(i)}$ is available and relatively simple, we can tell whether the polynomial of $\pi_{\boldsymbol{u}^{(i+1)}}(\boldsymbol{x}^{(i+1)})$ contains the term $\pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)})$ for any $\boldsymbol{u}^{(i)}$ and $\boldsymbol{u}^{(i+1)}$. $\pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)}) \to \pi_{\boldsymbol{u}^{(i+1)}}(\boldsymbol{x}^{(i+1)})$ denotes $\pi_{\boldsymbol{u}^{(i+1)}}(\boldsymbol{x}^{(i+1)})$ contains $\pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)})$ according to [15]. Then we introduce the definition of the monomial trail [15].

Definition 1 (Monomial Trail [15]). *Let $\boldsymbol{x}^{(i+1)} = \boldsymbol{f}^{(i)}(\boldsymbol{x}^{(i)})$ for $0 \leq i < r$. We call a sequence of monomials $(\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}), \pi_{\boldsymbol{u}^{(1)}}(\boldsymbol{x}^{(1)}), \ldots, \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)}))$ an r-round monomial trail connecting $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ and $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ with respect to the composite function $\boldsymbol{f} = \boldsymbol{f}^{(r-1)} \circ \boldsymbol{f}^{(r-2)} \circ \cdots \circ \boldsymbol{f}^{(0)}$ if*

$$\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \to \cdots \to \pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)}) \to \cdots \to \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)}).$$

If there is at least one monomial trail connecting $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ and $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$, we write $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$. Otherwise, $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \not\rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$.

The monomial prediction is another language for the division property from the polynomial viewpoint. The paper [15] has shown the equivalence between 3SDPwoU and the monomial prediction. In theory, they are perfectly accurate in detecting the integral distinguishers by counting the number of monomial trails. However, counting trails are time and memory consuming in most cases especially for the block ciphers, then some trade-off between the accuracy and the efficiency is necessary. The previous division properties (except the 3SDPwoU) can be regarded as the compromised algorithms of the monomial prediction. In this paper, we mainly take the fact in Lemma 1 to search for the integral distinguishers for Pyjamask.

Lemma 1 ([15]). *$\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ if $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \to \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$, and thus $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \not\rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ implies $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \nrightarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$.*

2.4 MILP Modeling for the Monomial Prediction

In this subsection, we denote by $\boldsymbol{x} \in \mathbb{F}_2^n, \boldsymbol{k} \in \mathbb{F}_2^m$ the vectors for the plaintext and the master key, respectively. Let c be a certain output bit of the targeted cipher, then c is a function of $\boldsymbol{x}$ and $\boldsymbol{k}$ written as $c = f(\boldsymbol{x}, \boldsymbol{k})$. For a fix constant $\boldsymbol{u} \in \mathbb{F}_2^n$, we consider the encryption of the following structure of plaintexts

$$\mathbb{X} = \{\boldsymbol{x} \preceq \boldsymbol{u} : \boldsymbol{x} \in \mathbb{F}_2^n\}.$$

Then whether c has the integral property is decided by whether $\bigoplus_{\boldsymbol{x}\in\mathbb{X}} c = \bigoplus_{\boldsymbol{x}\in\mathbb{X}} f(\boldsymbol{x}, \boldsymbol{k})$ is a constant (0 or 1), which is further decided by whether for all possible $\boldsymbol{v} \in \mathbb{F}_2^m \setminus \{\boldsymbol{0}\}$, $\pi_{\boldsymbol{v}}(\boldsymbol{k}) \cdot \pi_{\boldsymbol{u}}(\boldsymbol{x})$ does not appear in the ANF of $c = f(\boldsymbol{x}, \boldsymbol{k})$. According to Lemma 1, if for all possible $\boldsymbol{v} \in \mathbb{F}_2^m \setminus \{\boldsymbol{0}\}$, $\pi_{\boldsymbol{v}}(\boldsymbol{k}) \cdot \pi_{\boldsymbol{u}}(\boldsymbol{x})$ does not have the monomial trails connecting $c = f(\boldsymbol{x}, \boldsymbol{k})$, then $\bigoplus_{\boldsymbol{x}\in\mathbb{X}} c$ is a constant, and c is key-independent.

In [26], Todo and Morri proposed the bit-based division properties, but the time and memory complexities of the corresponding algorithm are $\mathcal{O}(2^n)$ where n is the block size. To search for the division properties efficiently, Xiang et al. introduced the Mixed Integral Linear Programming (MILP) models in [30]. Since then, the MILP model has been the most common tool in the area about the division properties [28,27,18,6,8].

In the monomial prediction, we take a similar method to construct the MILP model by modeling the propagation of the monomial trails. Considering a monomial trail

$$(\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}), \pi_{\boldsymbol{u}^{(1)}}(\boldsymbol{x}^{(1)}), \ldots, \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})),$$

it is enough to only model the vectors of $\boldsymbol{u}^{(i)}$ since the $\boldsymbol{x}^{(i)}$ are only symbolic variables. Then we only need to model the transitions of $(\boldsymbol{u}^{(0)}, \boldsymbol{u}^{(1)}, \ldots, \boldsymbol{u}^{(r)})$. For any Boolean function $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$, every pair of $(\boldsymbol{u}, \boldsymbol{v})$ is a valid monomial trail through $\boldsymbol{f}$ if and only if $\boldsymbol{x}^{\boldsymbol{u}} \rightarrow \boldsymbol{y}^{\boldsymbol{v}}$. Then we add constraints on $(\boldsymbol{u}, \boldsymbol{v})$ to force them be the valid transitions. Since every block cipher can be decomposed into some smaller components such as XOR, COPY, S-box and the linear layer, we introduce the MILP models for these functions.

Model 1 (COPY [13]) *Let $(a) \xrightarrow{COPY} (b_1, b_2, ..., b_m)$ denote the monomial trail through the COPY function, where one bit is copied to m bits. Then, it can be depicted using the following MILP constraints:*

$$\begin{cases} b_1 + b_2 + \ldots + b_m \geq a; \\ a \geq b_i, \ \textit{for all } i \in \{1, 2, \ldots, m\}; \\ a, b_1, b_2, ..., b_m \textit{ are binary variables.} \end{cases}$$

Model 2 (XOR [13]) *Let $(a_1, a_2, ..., a_m) \xrightarrow{XOR} (b)$ denote monomial trail through an XOR function, where m bits are compressed to one bit using an XOR operation. Then, it can be depicted using the following MILP constraints:*

$$\begin{cases} a_1 + a_2 + ... + a_m - b = 0; \\ a_1, a_2, ..., a_m, b \textit{ are binary variables.} \end{cases}$$

Model for S-box [30,29]. Given an S-box sending an n-bit vector to an m-bit vector, the monomial trails through the S-box can be represented as a set of

$(n+m)$-dimensional binary vectors which has a convex hull. With the help of `inequality_generator()` function in Sagemath [2] set of linear inequalities can be derived to describe the H-Representation of this convex hull. Then we use the greedy algorithm [21] to simplify them.

Model for the linear layer [20]. In [20], Sun et al. explained how to deduce a MILP model of the linear layers for the two-subset bit-based division properties. Here we slightly modify it for the monomial trails. Let $\boldsymbol{M}$ be a $n \times n$ matrix over $\mathbb{F}_2$, which can be represented at the bit level and denote

$$M = \begin{pmatrix} m_{0,0} & m_{0,1} & \cdots & m_{0,n-1} \\ m_{1,0} & m_{1,1} & \cdots & m_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n-1,0} & m_{n-1,1} & \cdots & m_{n-1,n-1} \end{pmatrix}$$

where $m_{i,j} \in \{0,1\}$. To represent the monomial trails through the linear layer, we introduce a set of auxiliary binary variables $t_{i,j}(0 \leq i,j \leq n-1)$ to decompose the binary matrix into XOR and COPY operations. Denote $(x_0, x_1, ..., x_{n-1}) \rightarrow (y_0, y_1, ..., y_{n-1})$ as a monomial trail of $\boldsymbol{M}$, then we can construct linear inequality system as $x_j \xrightarrow{COPY} (t_{0,j}, t_{1,j}, ..., t_{n-1,j})$ and $(t_{i,0}, t_{i,1}, ..., t_{i,n-1}) \xrightarrow{XOR} y_i$, where the inequalities for COPY and XOR are from Model 1 and Model 2.

3 Automatic Search Model for Pyjamask and Integral Distinguishers

3.1 MILP Model for Pyjamask-96 and Pyjamask-128

The works in [13,15] have implied that the 3SDPwoU and the monomial prediction can be used to detect the integral properties for ciphers considering the key schedule. In this paper, we regard all the round keys are independent input variables, and the whole model we consider is illustrated in Figure 2. Inspired by [14], we describe both encryption algorithm and the round keys into the MILP model. Suppose the length of the block size and the round key is n ($n = 96$ for

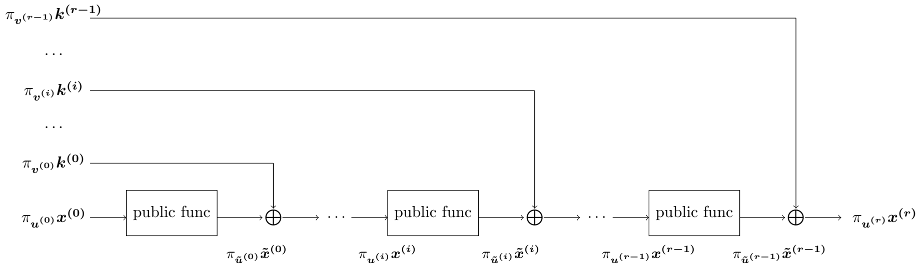
$\pi_{\boldsymbol{v}^{(r-1)}}\boldsymbol{k}^{(r-1)}$
...
$\pi_{\boldsymbol{v}^{(i)}}\boldsymbol{k}^{(i)}$
...
$\pi_{\boldsymbol{v}^{(0)}}\boldsymbol{k}^{(0)}$
$\pi_{\boldsymbol{u}^{(0)}}\boldsymbol{x}^{(0)}$
public func
...
public func
...
public func
$\pi_{\boldsymbol{u}^{(r)}}\boldsymbol{x}^{(r)}$
$\pi_{\tilde{\boldsymbol{u}}^{(0)}}\tilde{\boldsymbol{x}}^{(0)}$
$\pi_{\boldsymbol{u}^{(i)}}\boldsymbol{x}^{(i)}$
$\pi_{\tilde{\boldsymbol{u}}^{(i)}}\tilde{\boldsymbol{x}}^{(i)}$
$\pi_{\boldsymbol{u}^{(r-1)}}\boldsymbol{x}^{(r-1)}$
$\pi_{\tilde{\boldsymbol{u}}^{(r-1)}}\tilde{\boldsymbol{x}}^{(r-1)}$

Fig. 2: General structure of our MILP model

Pyjamask-96 while $n = 128$ for Pyjamask-128). For $0 \leq i < r$, let $\pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)})$ and $\pi_{\tilde{\boldsymbol{u}}^{(i)}}(\tilde{\boldsymbol{x}}^{(i)})$ denote the monomials of the input and output statements of the ith round function, respectively. $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ denotes a monomial of the ciphertext we are interested in. Practically, $\boldsymbol{u}^{(r)}$ is set as a unit vector to study a certain bit of the ciphertext. $\pi_{\boldsymbol{v}^{(i)}}(\boldsymbol{k}^{(i)})$ denotes the monomial of the ith round key. We use inequalities to add constraints for variables $\boldsymbol{u}^{(i)}, \tilde{\boldsymbol{u}}^{(i)}$ and $\boldsymbol{v}^{(i)}$ according to the functions between them (recall $\boldsymbol{x}^{(i)}, \tilde{\boldsymbol{x}}^{(i)}$ and $\boldsymbol{k}^{(i)}$ are only symbolic variables). The whole process is very similar to [30] except,

1. Besides the encryption, the round keys are also considered. The monomials $\pi_{\boldsymbol{v}^{(i)}}(\boldsymbol{k}^{(i)})$ are treated equivalently as $\pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)})$ when we add constraints.
2. The monomial trails through public functions are described in the models introduced in Section 2.4 rather than the constraints for the two-subset bit-based division properties.

Initial constraints on $(\boldsymbol{u}^{(0)}, \boldsymbol{v}^{(0)}, \ldots, \boldsymbol{v}^{(r-1)})$. Given a structure of plaintexts that all the active bits are a subset $I \subset \{0, 1, \ldots, n-1\}$, then we use

$$\begin{cases} u_i^{(0)} = 1, & \text{if } i \in I; \\ u_i^{(0)} = 0, & \text{if } i \notin I. \end{cases}$$

to add the initial constraints on $\boldsymbol{u}^{(0)}$. Note we do not add any constraints on $(\boldsymbol{v}^{(0)}, \ldots, \boldsymbol{v}^{(r-1)})$ to allow $(\boldsymbol{v}^{(0)}, \ldots, \boldsymbol{v}^{(r-1)})$ to be free variables over $\mathbb{F}_2^{m \times r}$.

Stopping constraints on $\boldsymbol{u}^{(r)}$. If we consider the integral property of the i'th ciphertext bit, then we use

$$\begin{cases} u_i^{(r)} = 1, & \text{if } i = i'; \\ u_i^{(r)} = 0, & \text{if } i \neq i'. \end{cases}$$

to add the stopping constraints on $\boldsymbol{u}^{(r)}$.

Once we obtain the whole MILP model, we then call the MILP solver to solve the model. If the model is not feasible, for any $(\boldsymbol{v}^{(0)}, \ldots, \boldsymbol{v}^{(r-1)}) \in \mathbb{F}_2^{m \times r}$, $\pi_{\boldsymbol{v}^{(0)}, \ldots, \boldsymbol{v}^{(r-1)}}(\boldsymbol{k}^{(0)}, \ldots, \boldsymbol{k}^{(r-1)}) \cdot \pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ has no trails connecting to $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$, i.e.,

$$\pi_{\boldsymbol{v}^{(0)}, \ldots, \boldsymbol{v}^{(r-1)}}(\boldsymbol{k}^{(0)}, \ldots, \boldsymbol{k}^{(r-1)}) \cdot \pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \not\rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)}), \forall (\boldsymbol{v}^{(0)}, \ldots, \boldsymbol{v}^{(r-1)}) \in \mathbb{F}_2^{m \times r}.$$

According to Lemma 1, $\pi_{\boldsymbol{v}^{(0)}, \ldots, \boldsymbol{v}^{(r-1)}}(\boldsymbol{k}^{(0)}, \ldots, \boldsymbol{k}^{(r-1)}) \cdot \pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ does not appear in $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ for any $(\boldsymbol{v}^{(0)}, \ldots, \boldsymbol{v}^{(r-1)}) \in \mathbb{F}_2^{m \times r}$. Consequently, $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ is zero-sum.

Optional Extra Constraint to Obtain More Integral Property. In the integral attacks, not only the zero-sum ciphertext bits but also the one-sum ciphertext bits are useful. So we add an extra constraint to exclude the case that $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ as follows,

$$\mathcal{M} \leftarrow \sum_{0 \leq i < r,\ 0 \leq j < m} v_j^{(i)} \geq 1. \tag{1}$$

In this case, if the model is not feasible, then

$$\pi_{\boldsymbol{v}^{(0)},\ldots,\boldsymbol{v}^{(r-1)}}(\boldsymbol{k}^{(0)},\ldots,\boldsymbol{k}^{(r-1)})\cdot\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \not\rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)}), \forall(\boldsymbol{v}^{(0)},\ldots,\boldsymbol{v}^{(r-1)}) \in \mathbb{F}_2^{m\times r}\backslash\{\boldsymbol{0}\}.$$

According to Lemma 1, $\pi_{\boldsymbol{v}^{(0)},\ldots,\boldsymbol{v}^{(r-1)}}(\boldsymbol{k}^{(0)},\ldots,\boldsymbol{k}^{(r-1)})\cdot\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ does not appear in $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ for any $(\boldsymbol{v}^{(0)},\ldots,\boldsymbol{v}^{(r-1)}) \in \mathbb{F}_2^{m\times r}\setminus\{\boldsymbol{0}\}$. Consequently, the ith bit of $\boldsymbol{x}^{(r)}$ is key-independent, i.e., zero-sum or one-sum (the concrete property can be determined by an additional experiment). All the source codes are available at https://github.com/iljido/MILP_pyjamask. We refer the readers to our codes for more details of the MILP model.

3.2 Integral Distinguishers of Pyjamask-96 and Pyjamask-128

Since the goal for the first step is to obtain the longest distinguisher, e.g., for Pyjamask-96, we set the 95 bit of the input to active and one bit to constant to find r-round distinguisher. Then the 96 positions of the constant bits are traversed. If there are some balanced bits, we increase the round to $r+1$ and repeat the search process until no balanced bits are available. Then we found a 10.5-round distinguisher for Pyjamask-96 with 96 balanced bits, which can be naturelly extended to 11 rounds since the MixRows is linear. Then, we try to reduce the data complexity by minimizing the number of the active bits of the input as described in [19]. The reduced-round distinguishers can be obtained in the similar way.

For Pyjamask-96, we found distinguishers up to 11 rounds as follows, where $\mathcal{A},\mathcal{C}$ represent ACTIVE, CONSTANT bits respectively. And $\mathcal{B}$ denotes the zero-sum property while $\mathcal{B}_c$ represents the key-independent bits. Note that only for finding the 11-round distinguisher, we use the extra constraint in Equation (1). Comparing to the results without the constraint, we find 32 more key-independent bits (the middle 32 bits denoted by $\mathcal{B}_c^{32}$), which helps to reduce the complexity of the key recovery attack on 14-round Pyjamask-96.

$$\begin{aligned}
(\mathcal{C}^{32},\mathcal{C}^{32},\mathcal{A}^{9}\mathcal{C}^{23}) &\xrightarrow{5R} (\mathcal{B}^{32},\mathcal{B}^{32},\mathcal{B}^{32})\\
(\mathcal{C}^{32},\mathcal{C}^{32},\mathcal{A}^{17}\mathcal{C}^{15}) &\xrightarrow{6R} (\mathcal{B}^{32},\mathcal{B}^{32},\mathcal{B}^{32})\\
(\mathcal{C}^{10}\mathcal{A}^{22},\mathcal{C}^{10}\mathcal{A}^{22},\mathcal{C}^{10}\mathcal{A}^{22}) &\xrightarrow{7R} (\mathcal{B}^{32},\mathcal{B}^{32},\mathcal{B}^{32})\\
(\mathcal{C}^{5}\mathcal{A}^{27},\mathcal{C}^{5}\mathcal{A}^{27},\mathcal{C}^{5}\mathcal{A}^{27}) &\xrightarrow{8R} (\mathcal{B}^{32},\mathcal{B}^{32},\mathcal{B}^{32})\\
(\mathcal{C}^{2}\mathcal{A}^{30},\mathcal{C}^{2}\mathcal{A}^{30},\mathcal{C}^{2}\mathcal{A}^{30}) &\xrightarrow{9R} (\mathcal{B}^{32},\mathcal{B}^{32},\mathcal{B}^{32})\\
(\mathcal{C}^{1}\mathcal{A}^{31},\mathcal{C}^{1}\mathcal{A}^{31},\mathcal{C}^{1}\mathcal{A}^{31}) &\xrightarrow{10R} (\mathcal{B}^{32},\mathcal{B}^{32},\mathcal{B}^{32})\\
(\mathcal{C}^{1}\mathcal{A}^{31},\mathcal{A}^{32},\mathcal{A}^{32}) &\xrightarrow{11R} (\mathcal{B}^{32},\mathcal{B}_c^{32},\mathcal{B}^{32})
\end{aligned}$$

For Pyjamask-128, distinguishers up to 9 rounds are available.

$$\begin{aligned}
(\mathcal{A}^{5}\mathcal{C}^{27},\mathcal{A}^{5}\mathcal{C}^{27},\mathcal{A}^{5}\mathcal{C}^{27},\mathcal{A}^{5}\mathcal{C}^{27}) &\xrightarrow{4R} (\mathcal{B}^{32},\mathcal{B}^{32},\mathcal{B}^{32},\mathcal{B}^{32})\\
(\mathcal{A}^{13}\mathcal{C}^{19},\mathcal{A}^{13}\mathcal{C}^{19},\mathcal{A}^{13}\mathcal{C}^{19},\mathcal{A}^{13}\mathcal{C}^{19}) &\xrightarrow{5R} (\mathcal{B}^{32},\mathcal{B}^{32},\mathcal{B}^{32},\mathcal{B}^{32})\\
(\mathcal{A}^{23}\mathcal{C}^{9},\mathcal{A}^{22}\mathcal{C}^{10},\mathcal{A}^{23}\mathcal{C}^{9},\mathcal{A}^{22}\mathcal{C}^{10}) &\xrightarrow{6R} (\mathcal{B}^{32},\mathcal{U}^{32},\mathcal{B}^{32},\mathcal{U}^{32})\\
(\mathcal{A}^{28}\mathcal{C}^{4},\mathcal{A}^{28}\mathcal{C}^{4},\mathcal{A}^{28}\mathcal{C}^{4},\mathcal{A}^{28}\mathcal{C}^{4}) &\xrightarrow{7R} (\mathcal{B}^{32},\mathcal{U}^{32},\mathcal{U}^{32},\mathcal{U}^{32})\\
(\mathcal{A}^{31}\mathcal{C}^{1},\mathcal{A}^{30}\mathcal{C}^{2},\mathcal{A}^{31}\mathcal{C}^{1},\mathcal{A}^{30}\mathcal{C}^{2}) &\xrightarrow{8R} (\mathcal{B}^{32},\mathcal{U}^{32},\mathcal{B}^{32},\mathcal{U}^{32})\\
(\mathcal{A}^{32},\mathcal{C}^{1}\mathcal{A}^{31},\mathcal{A}^{32},\mathcal{A}^{32}) &\xrightarrow{9R} (\mathcal{B}^{32},\mathcal{U}^{32},\mathcal{B}^{32},\mathcal{U}^{32})
\end{aligned}$$

4 Key Recovery Attack on Pyjamask-96

In this section, we present the key-recovery attacks for up to 14 rounds of Pyjamask-96. Since the attacks on rounds less than 13 is simple, here we only give the details of the attacks on 13- and 14-round Pyjamask-96 based on the same 11-round distinguisher available in Section 3. For a set of 2^{95} plaintexts denoted by $\mathbb{P}$ with the form of $(\mathcal{C}^{1}\mathcal{A}^{31},\mathcal{A}^{32},\mathcal{A}^{32})$, the intermediate state after 11 rounds (say, $x^{(11)}$) has the form of $(\mathcal{B}^{32},\mathcal{B}^{32}_{c},\mathcal{B}^{32})$. Taking a pre-computation with 2^{95} chosen plaintexts, the integral property (zero-sum or one-sum) of the middle 32-bit ciphertext can be determined. Since the complexities of our attack on 14-round Pyjamask-96 are significantly larger than 2^{95} (see Table 1), so the pre-computation is negligible. For the 13-round attack, we only use the 64 bits with zero-sum property.

As is well known, once an integral characteristic is found, we can take it to mount a key-recovery attack. If we denote by f the Boolean function that represents the mapping from the ciphertext of Pyjamask-96 to one of the balanced intermediate bit of $x^{(11)}$ (the output of the integral distinguisher), then we are interested in the following equation

$$\sum_{p\in\mathbb{P}} x^{(11)}[i] = \sum_{c\in\mathbb{C}} f(c) = 0 \tag{2}$$

where $x^{(11)}[i]$ is any one balanced bit and $\mathbb{C}$ is the corresponding ciphertext sets encrypted from $\mathbb{P}$. In the process of evaluating Equation (2), we guess the involved subkey bits used in f, and check whether Equation (2) holds. Those subkey values which violate Equation (2) will be filtered out and discarded, and the remaining are the candidates of the correct subkeys.

4.1 Attack on 13-Round Pyjamask-96

By appending two rounds after the distinguisher, we can get a 13-round key recovery attack on Pyjamask-96. Note that we have changed the order of the `MixRows` and `AddRoundKey`, so the last operation of the 13-round Pyjamask is the `MixRows` that can be ignored and then the ciphertext is actually $\bar{z}^{(12)}$. Firstly, we try to write out explicitly the mapping f from $\bar{z}^{(12)}$ to any one balanced bit of $x^{(11)}$. Without loss of generality, here we take $x^{(11)}[0]$ as an example.

Since the Boolean function of f is too complicated, we split it into two steps and in each step, we make clear the subkey bits we need to guess.

Step 1: Express $x^{(11)}[0]$ by $\bar{z}^{(11)}$. We first express $x^{(11)}[0]$ in a polynomial of $\bar{z}^{(11)}$, according to the ANF of the inverse of S_3

$$\begin{aligned} x^{(11)}[0] &= y^{(11)}[0] \cdot y^{(11)}[32] + y^{(11)}[64] + 1 \\ &= (\bar{z}^{(11)}[0] + u^{(11)}[0])(\bar{z}^{(11)}[32] + u^{(11)}[32]) + (\bar{z}^{(11)}[64] + u^{(11)}[64]) + 1. \end{aligned}$$

If we have known $\bar{z}^{(11)}[0], \bar{z}^{(11)}[32]$ and $\bar{z}^{(11)}[64]$, through guessing $u^{(11)}[0], u^{(11)}[32]$ and $u^{(11)}[64]$ we can compute $x^{(11)}[0]$.

Step 2: Express respectively $\bar{z}^{(11)}[0], \bar{z}^{(11)}[32], \bar{z}^{(11)}[64]$ by $\bar{z}^{(12)}$. We first express $\bar{z}^{(11)}[64]$ in a polynomial of $\bar{z}^{(12)}$ as an example, the processes for $\bar{z}^{(11)}[0]$ and $\bar{z}^{(11)}[32]$ are similar.

$$\begin{aligned} \bar{z}^{(11)}[64] &= \boldsymbol{M}_2^{-1}[0] \cdot (x^{(12)}[64], x^{(12)}[65], \dots, x^{(12)}[95])^T \\ &= \sum_{i \in I_2} x^{(12)}[i] \\ &= \sum_{i \in I_2} (y^{(12)}[i-64] \cdot y^{(12)}[i] + y^{(12)}[i-32] + y^{(12)}[i] + 1) \\ &= \sum_{i \in I_2} ((\bar{z}^{(12)}[i-64] + u^{(12)}[i-64]) \cdot (\bar{z}^{(12)}[i] + u^{(12)}[i]) \\ &\quad + (\bar{z}^{(12)}[i-32] + u^{(12)}[i-32]) + (\bar{z}^{(12)}[i] + u^{(12)}[i]) + 1) \end{aligned} \tag{3}$$

where I_2 is a set of indices corresponding to the coefficient of $\boldsymbol{M}_2^{-1}[0]$ (recall that $\boldsymbol{M_i}[j]$ is the jth row of $\boldsymbol{M_i}$). For $\bar{z}^{(11)}[0]$ and $\bar{z}^{(11)}[32]$, the index sets are I_0 and I_1, which corresponds to $\boldsymbol{M}_0^{-1}[0]$ and $\boldsymbol{M}_1^{-1}[0]$ respectively. At first glance, to calculate $\bar{z}^{(11)}[64]$, we need to guess $u^{(12)}[i-64], u^{(12)}[i-32]$ and $u^{(12)}[i]$ for each $i \in I_2$, totally $3 \times |I_2|$ subkey bits. However, Equation (3) can be rewritten as

$$\begin{aligned} \bar{z}^{(11)}[64] = & \sum_{i \in I_2} \bar{z}^{(12)}[i-64]\bar{z}^{(12)}[i] + \sum_{i \in I_2} u^{(12)}[i]\bar{z}^{(12)}[i-64] \\ & + \sum_{i \in I_2} \bar{z}^{(12)}[i]u^{(12)}[i-64] + \sum_{i \in I_2} u^{(12)}[i-64]u^{(12)}[i] \\ & + \utilde{\sum_{i \in I_2} \bar{z}^{(12)}[i-32]} + \sum_{i \in I_2} \bar{z}^{(12)}[i] \\ & + \underline{\sum_{i \in I_2} u^{(12)}[i-32]} + \sum_{i \in I_2} u^{(12)}[i] + 1. \end{aligned} \tag{4}$$

The subkey bits in the underlined term $\sum_{i \in I_2} u^{(12)}[i-32]$ are independent of other subkey or intermediate state bits, so we can regard the whole as one equivalent subkey bit. Consequently, we now need to guess much less (equivalent) subkey bits, totally $2 \times |I_2| + 1$ bits. For $\bar{z}^{(11)}[0]$ and $\bar{z}^{(11)}[32]$, we need $2 \times |I_0| + 1$ and $2 \times |I_1| + 1$ subkey bits respectively. By removing the 11 reusable subkey bits $u^{(12)}[10]$, $u^{(12)}[20]$, $u^{(12)}[24]$, $u^{(12)}[32]$, $u^{(12)}[39]$, $u^{(12)}[44]$, $u^{(12)}[55]$, $u^{(12)}[65]$, $u^{(12)}[73]$, $u^{(12)}[86]$ and $u^{(12)}[94]$, we need 55 subkey bits in $u^{(12)}$ and 3 equivalent subkey bits $\sum_{i \in I_0} u^{(12)}[i+64]$, $\sum_{i \in I_1} u^{(12)}[i-32]$, $\sum_{i \in I_2} u^{(12)}[i-32]$ to compute $\bar{z}^{(11)}[0]$, $\bar{z}^{(11)}[32]$ and $\bar{z}^{(11)}[64]$.

The relevant data bits can be compressed similarly as above. To calculate $\bar{z}^{(11)}[64]$, $3 \times |I_2|$ bits in $\bar{z}^{(12)}$ are required. For $\bar{z}^{(11)}[0]$ and $\bar{z}^{(11)}[32]$, we need $3 \times |I_0|$ and $3 \times |I_1|$ bits in $\bar{z}^{(12)}$, respectively. By removing the reusable bits, totally 66 data bits in $\bar{z}^{(12)}$ are required, which are highlighted gray in Figure 3. However, the underwaved term $\sum_{i \in I_2} \bar{z}^{(12)}[i-32]$ are also independent from other subkey or intermediate state bits. So we can pre-compute the whole as one equivalent data bit for each ciphertext. Then, some of the $u^{(12)}[i]$ make no contribution to the further calculation and can be removed. For $\bar{z}^{(11)}[64]$, we need $2 \times |I_2|$ data bits in $\bar{z}^{(12)}$ and one equivalent data bit $c_2 = \sum_{i \in I_2} \bar{z}^{(12)}[i-32]$. It is the same for $\bar{z}^{(11)}[0]$ and $\bar{z}^{(11)}[32]$. The equivalent data bits are $c_0 = \sum_{i \in I_0} \bar{z}^{(12)}[i+64]$ and $c_1 = \sum_{i \in I_1} \bar{z}^{(12)}[i-32]$, respectively. Consequently, 11 bits in $\bar{z}^{(12)}$ are reduced in total (indicated by a cross in Figure 3). We now need 58 data bits in $\bar{z}^{(12)}$ for the calculation of $x^{(11)}[0]$, including 3 equivalent data bits c_0, c_1 and c_2.

Furthermore, the 3 bit equivalent subkey bit $\sum_{i \in I_0} u^{(12)}[i+64]$, $\sum_{i \in I_1} u^{(12)}[i-32]$, $\sum_{i \in I_2} u^{(12)}[i-32]$ can be merged into the corresponding subkey bits $u^{(11)}[0]$, $u^{(11)}[32]$, $u^{(11)}[64]$ and consider together. Consequently, only 58 (equivalent) key bits are required to calculate $\sum_{p \in \mathbb{P}} x^{(11)}[0]$, including 55 bits in $u^{(12)}$ and 3 bits in $u^{(11)}$.

Partial Sum An additional technique to reduce the complexity of the attack is the partial sum technique described in [11]. It observes that the small S-boxes are applied separately and the output of the `MixRows` is a linear combination of several input bits. Then, the relevant subkey bits are divided into relatively independent parts and guessed one after another. When a part of the subkey is guessed, the corresponding ciphertext can be compressed with the information and stored in a counter for further calculation. Meanwhile, the complexity can also be influenced by the order of guessing. So the trade-off between the increase of the guessed subkey bits and the decrease in the size of the counters deserves consideration. We first consider columns in $\bar{z}^{(12)}$ which correspond to only 2 equivalent subkey bits and then the columns related to 3-bit equivalent subkey.

Key Recovery Procedure. We give the process of the 13-round attack with the observations described before in this section. The attack consists of three phases.

1. **Preparing for the counters:**

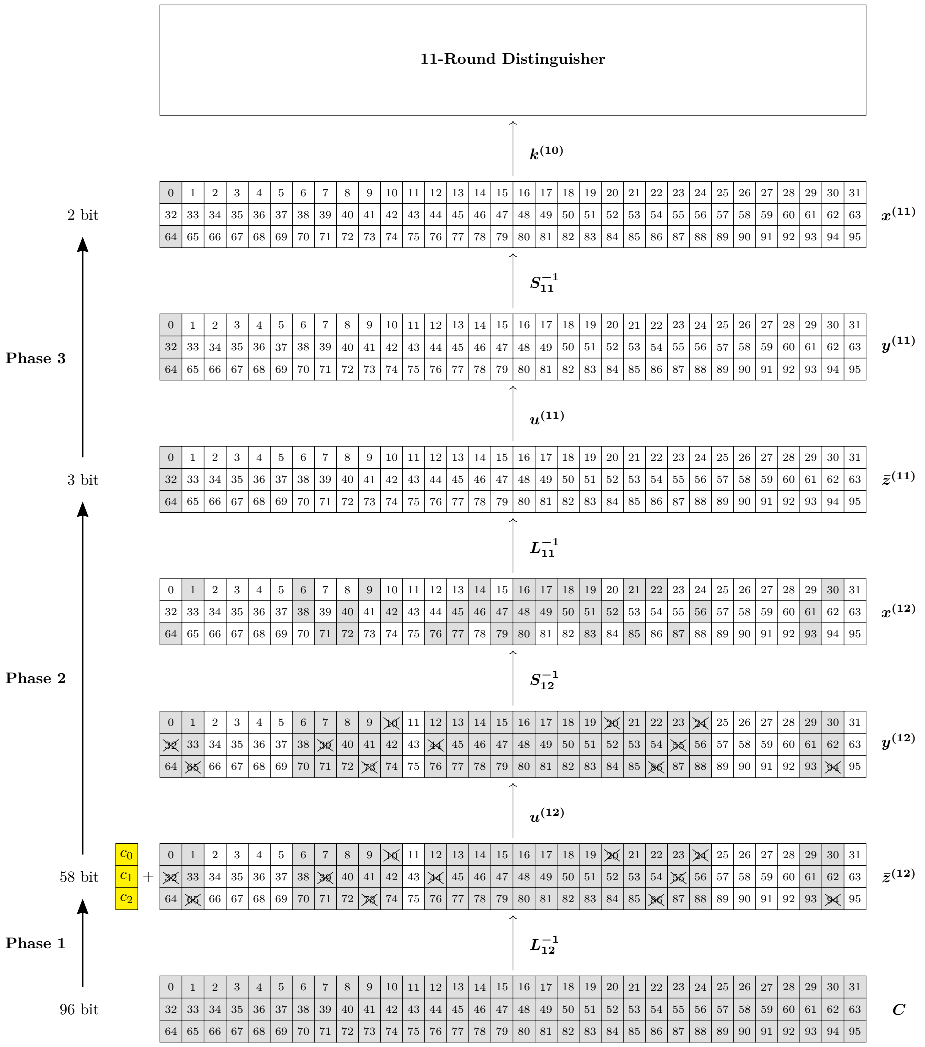

Fig. 3: Key-Recovery of 13-Round Pyjamask-96

1.1 Choose a set of 2^{95} plaintexts $\mathbb{P}$ of the form $(\mathcal{C}^1\mathcal{A}^{31}, \mathcal{A}^{32}, \mathcal{A}^{32})$, $0 \le i < 2^{95}$.

1.2 Allocate a counter $\boldsymbol{T}_0$ of size 2^{66} containing 1-bit values and initialized by 0. For each $P_i \in \mathbb{P}$, call 13-round Pyjamask-96 (without the last MixRows) to encrypt P_i and obtain the corresponding ciphertext $\bar{z}^{(12)}$. Take the cascaded value of the 66-bit value $\bar{z}^{(12)}[0-1]$, $\bar{z}^{(12)}[6-10]$, $\bar{z}^{(12)}[12-24]$, $\bar{z}^{(12)}[29-30]$, $\bar{z}^{(12)}[32-33]$, $\bar{z}^{(12)}[38-42]$, $\bar{z}^{(12)}[44-56]$, $\bar{z}^{(12)}[61-62]$, $\bar{z}^{(12)}[64-65]$, $\bar{z}^{(12)}[70-74]$, $\bar{z}^{(12)}[76-88]$, $\bar{z}^{(12)}[93-94]$ as an index (denoted by i_0) and update $\boldsymbol{T}_0$ by $\boldsymbol{T}_0[i_0] = \boldsymbol{T}_0[i_0] + 1 \pmod 2$.

1.3 Allocate another counter $\boldsymbol{T}_1$ of size 2^{58} containing 1-bit values and initialized by 0. For each $\boldsymbol{T}_0$, calculate the 58-bit index i_1 of $\boldsymbol{T}_1$.

(a) The first 55-bit index of $\boldsymbol{T}_1$ can be calculated by taking out the 11-bit value $\bar{z}^{(12)}[10]$, $\bar{z}^{(12)}[20]$, $\bar{z}^{(12)}[24]$, $\bar{z}^{(12)}[32]$, $\bar{z}^{(12)}[39]$, $\bar{z}^{(12)}[44]$, $\bar{z}^{(12)}[55]$, $\bar{z}^{(12)}[65]$, $\bar{z}^{(12)}[73]$, $\bar{z}^{(12)}[86]$, $\bar{z}^{(12)}[94]$ and cascading the rest bits in order.

(b) The last 3 bits are derived according to $\sum_{i\in I_0} \bar{z}^{(12)}[i+64]$, $\sum_{i\in I_1} \bar{z}^{(12)}[i-32]$ and $\sum_{i\in I_2} \bar{z}^{(12)}[i-32]$.

(c) Update $\boldsymbol{T}_1$ by $\boldsymbol{T}_1[i_1] = \boldsymbol{T}_1[i_1] + 1 \pmod 2$.

2. **Guessing subkeys in $u^{(12)}$:** Based on the 58-bit counter $\boldsymbol{T}_1$ derived, we now guess the 55-bit equivalent subkey in $u^{(12)}$.

2.1 The relevant subkey bits of the column 0, 1, 7, 9, 10, 12, 20, 22, 23, 24, 30 in $\bar{z}^{(12)}$ are guessed separately first. For each 2-bit guess, decrypt through S_3 and calculate the contribution to the corresponding $\bar{z}^{(11)}[i]$, $i = 0, 32, 64$. We treat the cost of the operation at once as 2 times 3-bit S-box computations. In total, 22 subkey bits are guessed in this step and the complexity is around $2^{60} \times 11 \times 2$ 3-bit S-box computations.

2.2 The relevant subkey bits of the remaining 11 columns in $\bar{z}^{(12)}$ are guessed separately. For each 3-bit guess, decrypt through S_3 and calculate the contribution to the corresponding $\bar{z}^{(11)}[i]$, $i = 0, 32, 64$. 33 subkey bits are guessed in this step and the complexity of this step is around $2^{61} \times 11 \times 2$ 3-bit S-box computations.

3. **Guessing subkeys in $u^{(11)}$:** The 3 bits equivalent key in $u^{(11)}$ are guessed in this step. For each guess, decrypt $\bar{z}^{(11)}[0]$, $\bar{z}^{(11)}[32]$, $\bar{z}^{(11)}[64]$ through the 3-bit S-box. Keep the subkey where $(x^{(11)}[0], x^{(11)}[64]) = (0, 0)$. The complexity of this step is around $2^{55+3+3} \times 2$ 3-bit S-box computations.

4. **Exhaustive search the rest of the key:** The key space is reduced by a factor of 2^{-2} for the 58-bit guess. By altering the position of the balanced bit and repeating the above steps, the key space can be further reduced. We choose 18 balanced columns and find the remaining subkey bits by an exhaustive search of 2^{92} possible combinations.

Complexity of the attack In Phase 1.1 and 1.2, the complexity is calculated as 2^{95} 13-round Pyjamask-96 calls and 2^{95} times `MixRows` operations. The cost of Phase 1.3, Phase 2 and Phase 3 is calculated as

$$(2^{60} \times 11 \times 2 + 2^{61} \times 11 \times 2 + 2^{55+3+3} \times 2)/32 \approx 2^{61}$$

times `SubBytes` operations and $2^{66} \times 3/96 = 2^{61}$ times `MixRows` operations. For Phase 4, the same operations repeat for 18 times. The complexity is given as $18 \times 2^{61} + 2^{92}$ 13-round Pyjamask-96 calls. If we regard both `SubBytes` and `MixRows` as half a round of Pyjamask-96, the time complexity is calculated as

$$2^{95} + 2^{95}/2/13 + 18 \times 2^{61}/13 + 2^{92} \approx 2^{95.2}$$

times 13-round Pyjamask-96 calls. The data complexity is 2^{95} plaintexts. As the counters $\boldsymbol{T}_0$ for the 18 positions can be prepared together in Phase 1, the memory complexity is $2^{66} \times 18 \approx 2^{70}$ bits, which is around 2^{67} bytes.

4.2 Attack on full-Round Pyjamask-96

In this section, we extend one more round to get the full-round attack on Pyjamask-96. The order of `MixRows` and `AddRoundKey` has been changed, so the ciphertext is actually $\bar{z}^{(13)}$. Before going any further, we would like to briefly discuss the Fast Fourier Transform (FFT) key recovery technique. Following by the approach in [25], the key recovery process of 13-round attack can be expressed as $\sum_{i=1}^{N} F_{2,K'}(c'_i \oplus K'_r) = 0$, where K'_r and K' denotes the 66-bit $u^{(12)}$ and the 3-bit $u^{(11)}$, respectively. The time complexity is $O(2^3 \times 66 \times 2^{66})$ for one balanced bit. So we believe that the FFT technique can acheive similar performance to our 13-round attack on Pyjamask-96 and the description can be simplified. However, for the full-round attack, this techique fails since we could not truncate K'_r from the 96-bit $u^{(13)}$.

As is seen in the attack on the 13-round case, the critical point of reducing the complexity is to reduce the subkey bits we need to guess and the relevant state bits. However, due to the strong diffusion from the additional round, we need 165 subkey bits to express one balanced bit of $x^{(11)}$, which exceeds the size of master key. To express one balanced bit of $x^{(11)}$, we need 3 bits in $\bar{z}^{(11)}$. Since for one bit in $\bar{z}^{(11)}$ we still need 96 bits in $u^{(13)}$ and at least 33 bits in $u^{(12)}$, we try to split one bit of $\bar{z}^{(11)}$ into small expressions. Here we take $x^{(11)}[0]$ as an example. To express $x^{(11)}[0]$, we need $\bar{z}^{(11)}[0]$, $\bar{z}^{(11)}[32]$, $\bar{z}^{(11)}[64]$, which can be rewritten as $\sum_{i \in I_0} x^{(12)}[i]$, $\sum_{i \in I_1} x^{(12)}[i]$, $\sum_{i \in I_2} x^{(12)}[i]$ respectively. We then split each $I_j, 0 \leq j \leq 2$ into two disjoint sets $I_{j,0}$ and $I_{j,1}$ and make clear the subkey we need to guess for each set. Without loss of generality, we take $\sum_{i \in I_{1,0}} x^{(12)}[i]$ as an example, where $I_{1,0} = \{38, 40, 42, 45, 46, 47, 48\}$.

According to Equation (3) and Equation (4), we can express $\sum_{i \in I_{1,j}} x^{(12)}[i]$ by $u^{(12)}$ and $\bar{z}^{(12)}$, and reduce the relevant subkey bits and data bits. In total, we need $2 \times |I_{1,0}|$ data bits in $\bar{z}^{(12)}$ and one equivalent data bit $\sum_{i \in I_{1,0}} \bar{z}^{(12)}[i - 32]$. The relevant key bits are $2 \times |I_{1,0}|$ bits in $u^{(12)}$ (the equivalent key bit $\sum_{i \in I_{1,0}} u^{(12)}[i - 32]$ can be merged into the relevant subkey bit in $u^{(11)}$ and guessed together).

Next, we show how to take the relationship between the round keys to further reduce the $2 \times |I_{1,0}|$ bits in $u^{(12)}$ to $|I_{1,0}|$ bits. In the key schedule, each 128 bit output of the ith round function is composed of the 96-bit round key $k^{(i)} = (k^{(i)}[0], k^{(i)}[1], \ldots, k^{(i)}[95])$ and an additional 32-bit value $\bar{k}^{(i)} = (\bar{k}^{(i)}[96], \bar{k}^{(i)}[97], \ldots, \bar{k}^{(i)}[127])$. In the key recovery phase, we are interested in the relationship between the 128-bit output $(k^{(12)}, \bar{k}^{(12)})$ and $(k^{(13)}, \bar{k}^{(13)})$, So we introduce a new 128-bit transform $\overline{\texttt{MixRows}}$ consisting of 4 matrices $\boldsymbol{M_0}, \boldsymbol{M_1}, \boldsymbol{M_2}$ and $\boldsymbol{E}$, where $\boldsymbol{M_0}, \boldsymbol{M_1}, \boldsymbol{M_2}$ are the three matrices used in `MixRows` and $\boldsymbol{E}$ is a

32×32 identity matrix. Moreover, it is easy to check that $\overline{\texttt{MixRows}}^{-1}$ consists of $\boldsymbol{M_0}^{-1}, \boldsymbol{M_1}^{-1}, \boldsymbol{M_2}^{-1}$ and $\boldsymbol{E}$.

According to the key schedule, let $g : \mathbb{F}_2^{128} \to \mathbb{F}_2^{128}$ be a linear transform and $k^{(12)}[i] = g(k^{(13)}, \bar{k}^{(13)})[i]$, $0 \le i < 96$. The relationship between $u^{(12)}$ and $u^{(13)}$ is as follows,

$$\begin{aligned} u^{(12)}[i] &= \texttt{MixRows}^{-1}(k^{(12)})[i] \\ &= \overline{\texttt{MixRows}}^{-1} \circ g(k^{(13)}, \bar{k}^{(13)})[i] \\ &= \overline{\texttt{MixRows}}^{-1} \circ g \circ \overline{\texttt{MixRows}}(u^{(13)}, \bar{k}^{(13)})[i] \\ &= g(u^{(13)}, \bar{k}^{(13)})[i] \end{aligned}$$

Therefore, $u^{(12)}$ is a linear function of $u^{(13)}$ and $\bar{k}^{(13)}$. Considering the expression of $u^{(12)}[32]$ and suppose $v^{(i)}$ and $w^{(i)}$ are the output of `Mixcolumns` (recall a 4×4 matrix $\boldsymbol{M}$ is used in `Mixcolumns`) and `MixAndRotateRows` of the ith round, respectively, we have

$$\begin{aligned} u^{(12)}[32] &= \boldsymbol{M}^{-1}[1] \cdot (v^{(13)}[0], v^{(13)}[32], v^{(13)}[64], v^{(13)}[96]) \\ &= v^{(13)}[0] + v^{(13)}[64] + v^{(13)}[96] \\ &= \sum_{i \in I_k} w^{(13)}[i] + w^{(13)}[81] + w^{(13)}[110] \\ &= \underline{\sum_{i \in I_k} u^{(13)}[i] + u^{(13)}[81]} + \bar{k}^{(13)}[110] + 1 \end{aligned} \tag{5}$$

where I_k is a set of indices corresponding to the coefficient of $\boldsymbol{M}_k^{-1}[0]$. So $u^{(12)}[32]$ relates to only one bit in $\bar{k}^{(13)}$. For other bits in $u^{(12)}$, things is similar. Moreover, bits in every column of $u^{(12)}$ correspond to the same bit in $\bar{k}^{(13)}$ e.g., $u^{(12)}[0], u^{(12)}[32], u^{(12)}[96]$ all relate to $\bar{k}^{(13)}[110]$. Once we guess the full $u^{(13)}$, we can compute $u^{(12)}$ by the above Equation (5) with the additional knowledge of one corresponding bit in $\bar{k}^{(13)}$. Since the partial value of $u^{(12)}$ related to $u^{(13)}$ (e.g., the underlined part of Equation (5)) can be computed beforehand, for one bit in $\bar{k}^{(13)}$ guessed, we derive the value of three bits in $u^{(12)}$ with only 3 XORs.

We then give the process of the calculation for $\sum_{i \in I_{1,0}} x^{(12)}[i]$, $I_{1,0} = \{38, 40, 42, 45, 46, 47, 48\}$.

1. **Preparing for the counters:**
   1.1 Choose a set of 2^{95} plaintexts $\mathbb{P}$ with the form of $(\mathcal{C}^1\mathcal{A}^{31}, \mathcal{A}^{32}, \mathcal{A}^{32})$, $0 \le i < 2^{95}$.
   1.2 Allocate a counter $\boldsymbol{T}_0$ of size 2^{96} containing 1-bit values and initialized by 0. For each $P_i \in \mathbb{P}$, call 14-round Pyjamask-96 (without the last `MixRows`) to encrypt P_i and obtain the corresponding ciphertext $\bar{z}^{(13)}$ as an index (denoted by i_0). Update $\boldsymbol{T}_0$ by $\boldsymbol{T}_0[i_0] = \boldsymbol{T}_0[i_0] + 1 \pmod 2$.
2. **Guessing then 96 bits in $u^{(13)}$:**

2.1 Partially decrypt through the 15-bit subkey bits $u^{(13)}[0-4]$, $u^{(13)}[32-36]$, $u^{(13)}[64-68]$. For each 3-bit guess, partially decrypt the ciphertext through S_3. We treat the cost of each above operation as 2 times 3-bit S-box computations, then the complexity is around $2 \times (2^{3+96} + 2^{6+96} + 2^{9+96} + 2^{12+96} + 2^{15+96}) \approx 2^{112.2}$ times 3-bits S-box computation.

2.2 After all 15-bit subkey bits have been guessed, calculate the contribution of the 15 bits in $x^{(13)}$ to the 15 bits in $\bar{z}^{(12)}$. Keep the new 96-bit as the index of the counter. We treat the cost of each above operation as 7/32 times `MixRows` operations, then the complexity is $2^{96+15} \times 7/32 \approx 2^{108.9}$ times `MixRows` operations.

2.3 The remaining 81 bits in $u^{(13)}$ are guessed separately. For each 3-bit guess, partially decrypt the ciphertext through S_3 and calculate the contribution to the 15 bits in $\bar{z}^{(12)}$. We treat the cost of each above operation as 2 times 3-bit S-box computations and $7/(32{\times}11)$ times `MixRows` operations. The complexity of this step is bounded by $2^{15+96+3}{\times}27{\times}2 \approx 2^{119.8}$ times 3-bit S-box computations and $2^{15+96+3} \times 27 \times 7/(32 \times 11) \approx 2^{113.2}$ times `MixRows` operations.

3. **Guessing the 7 bits in $\bar{k}^{(13)}$:** Up to now, we have guessed the 96 bit in $u^{(13)}$. For each 96-bit guess:

3.1 Calculate the 14 expression of $u^{(12)}[i]$ according to the 96 bits in $u^{(13)}$, $i \in I_{1,0}$. The complexity of this step is 2^{96} times one round inverse key schedule.

3.2 Guess 7 bits in $\bar{k}^{(13)}$. For each 1-bit guess, derive the relevant 3-bit $u^{(12)}$ and calculate the contribution to $\sum_{i \in I_{1,0}} x^{(12)}[i]$. The complexity is around $2 \times (2^{96+1+15} + 2^{96+2+13} + 2^{96+3+11} + 2^{96+4+9} + 2^{96+5+7} + 2^{96+6+5} + 2^{96+7+3}) \approx 2^{114}$ times 3-bit S-box computations.

After that, we derive the corresponding value of $\sum_{i \in I_{1,0}} x^{(12)}[i]$ for each 103-bit subkey guess and store them in a table. The complexity is dominated by $(2^{112.2}{+}2^{119.8}{+}2^{114})/32 \approx 2^{114.9}$ times `SubBytes` operations and $2^{108.9}{+}2^{113.2} \approx 2^{113.3}$ times `MixRows` operations. Then the total complexity is bounded by $2^{110.6}$ times 14-round Pyjamask-96.

Then we calculate the table for the other 5 sets . The total complexity is bounded by $2^{110.6} + 2^{108.6} \times 3 + 2^{106.6} \times 2 \approx 2^{111.4}$ times 14-round Pyjamask-96.

After all 6 tables have been established, we then go through the relevant 118-bit $(u^{(13)}, \bar{k}^{(13)})$. For each 118-bit key guess, we search for the corresponding $\bar{z}^{11}[0]$, $\bar{z}^{11}[32]$, $\bar{z}^{11}[64]$. Then we guess the corresponding 3-bit subkey in $u^{(11)}$ and keep the subkey where $x^{(11)}[0]$, $x^{(11)}[32]$, $x^{(11)}[64]$ satisfying the tail of 11-round distinguisher (calculated by 2^{95} 11-round encryption). We treat the cost for searching table as one time 14-round Pyjamask-96 encryption. The time complexity for this step is $2^{118} \times 6 + 2^{118+3} \times 2/(32 \times 2 \times 14) \approx 2^{120.6}$ Pyjamask-96 calls. So the total complexity is

$$2^{95} + 2^{95}/(2 \times 14) + 2^{95} \times 11/14 + 2^{110.6} + 2^{120.6} \approx 2^{120.7}$$

times 14-round Pyjamask-96. The key space is reduced by $1/2^3$ each time. We choose 3 balanced columns and find the remaining key by 2^{119} exhaustive search.

The time complexity is $2^{120.7} \times 3 + 2^{120} \approx 2^{122.6}$ times 14-round Pyjamask-96. The memory complexity is around $(2^{103} + 3 \times 2^{102} + 2 \times 2^{101})/2^3 = 2^{101.6}$ bytes.

5 Integral Attacks on Round-Reduced Pyjamask-128

In this section we present the attack on Pyjamask-128 with the distinguishers shown in Section 3. The attack results are summarized in Table 1.

The 11-round attack on Pyjamask-128 is by appending two rounds after the 9-round distinguisher. For a set of 2^{127} plaintexts of the form $(\mathcal{A}^{32}, \mathcal{C}^1\mathcal{A}^{31}, \mathcal{A}^{32}, \mathcal{A}^{32})$, the intermediate state after 9 rounds has the form of $(\mathcal{B}^{32}, \mathcal{U}^{32}, \mathcal{B}^{32}, \mathcal{U}^{32})$. The order of `MixRows` and `AddRoundKey` has been exchanged like 13-round attack of Pyjamask-96. So the ciphertext is actually $\bar{z}^{(10)}$.

To express one bit in $x^{(9)}$, for example $x^{(9)}[0]$, we need 4 bits in $\bar{z}^{(9)}$ and 4 bits in $u^{(9)}$. Using the same method as Equation (3), we can compress the relevant data bits by the equivalent data bits $c_1 = \sum_{i \in I_1} \bar{z}^{(10)}[i-32]$ and $c_2 = \sum_{i \in I_2} \bar{z}^{(10)}[i-32] + \bar{z}^{(10)}[i+32]$. By removing the reusable bits and redundant bits, we need 99 bits in $\bar{z}^{(10)}$ and 2 equivalent data bits. The equivalent key is utilized similarly, and the involved key bits are 99 bits in $u^{(10)}$ and 4 equivalent bits in $u^{(9)}$. Then we guess the subkey bits partially and compress the ciphertext.

For each 101-bit equivalent $\bar{z}^{(10)}$, we guess the 99-bit $u^{(10)}$ partially. Each time, we decrypt the ciphertext through the 4-bit S-box and calculate the contribution to $\bar{z}^{(9)}[0]$, $\bar{z}^{(9)}[32]$, $\bar{z}^{(9)}[64]$, $\bar{z}^{(9)}[96]$. If we treat the cost of each above operation as 2 times 4-bit S-box computation, the complexity of this phase is $(2^{107} \times 21 + 2^{106} \times 3 + 2^{105} \times 2 + 2^{103}) \times 2/32 \approx 2^{107.6}$ times `SubBytes` operation.

Then we guess the 4 bits in $u^{(9)}$ and compare them with the tail of the distinguisher. The complexity for this phase is $2^{99+4+4} \times 2/32 = 2^{103}$ times `SubBytes`. The key space is reduced by $1/2^2$ each time. We choose 3 balanced columns and find the remaining key bits by an exhausitve search of 2^{122} possible combinations. The time complexity is dominated by the 2^{127} times encryption and the memory complexity is dominated by $2^{108} \times 3/2^3 \approx 2^{106.5}$ bytes.

For Pyjamask-128 of round r $(r = 7, 8, 9)$, we append 2 rounds after the $(r-2)$-round distinguisher. The process is similar. The time complexity is dominated by the encryption of plaintext for 10- and 9-round attacks, which are 2^{122} times 10-round Pyjamask-128 and 2^{112} times 9-round Pyjamask-128. For 8-round attack, each time when the key space is reduced by $1/2^2$, we need around $2^{107.6}/(2 \times 8) = 2^{103.6}$ times 8-round Pyjamask-128. We repeat the process for 12 times and the total complexity is $12 \times 2^{103.6} + 2^{106} \approx 2^{107.8}$ times 8-round Pyjamask-128.

For Pyjamask-128 of round r, $r = 5, 6, 7$, we append 1 round after the $(r-1)$-round distinguisher. Then we partially guess 4-bit subkey, decrypt the ciphertexts through S_4 and compare with the tail of the distinguisher. For 7-round Pyjamask-128, the time complexity is $2^{90+4} \times 2/(32 \times 2 \times 7) \approx 2^{86.2}$ times 7-round Pyjamask-128 for each 4-bit guess. We choose 20 balanced columns and the total complexity is $2^{90} + 2^{90}/(2 \times 7) + 20 \times 2^{86.2} + 2^{88} \approx 2^{91.5}$ times 7-round Pyjamask-128. For 6-round Pyjamask-128, the time complexity is composed by 2^{52} times 6-round Pyjamask-128, 2^{52} times $\texttt{MixRows}^{-1}$ and $20 \times 2^{52+4} \times 2/(32 \times 2 \times 6) + 2^{128-4\times 20} \approx 2^{52.8}$ times 6-round Pyjamask-128, which is bounded by 2^{54} times 6-round Pyjamask-128. For 5-round Pyjamask-128, the time complexity is composed by 2^{20} times 5-round Pyjamask-128, 2^{20} times $\texttt{MixRows}^{-1}$ and $32 \times 2^{20+4} \times 2/(32 \times 2 \times 5) \approx 2^{21.7}$ times 5-round Pyjamask-128, which is bounded by 2^{23} times 5-round Pyjamask-128.

6 Conclusion

This paper studies the security strength of the block ciphers Pyjamask-96 and Pyjamask-128 against the integral attacks. With a new MILP model for the division property considering the encryption function and the round keys simultaneously, we detect efficient integral distinguishers for both Pyjamask-96 and Pyjamask-128. For Pyjamask-96, we utilize the 11-round integral distinguisher, combined with a novel property of the key schedule, to mount a key recovery attack for full Pyjamask-96 without the full codebook for the first time. The results for fewer rounds, e.g., 13-round Pyjamask-96 are also improved. What's more, we give the first third-party cryptanalysis on the Pyjamask-128, which sheds more light on its security margin. The integral attacks on both versions of Pyjamask are significant to understand the low degree property of block ciphers.

Acknowledgements. We thank the anonymous reviewers for their valuable comments. This work is supported by the National Natural Science Foundation of China (Grant No. 62032014), the National Key Research and Development Program of China (Grant No. 2018YFA0704702), the Major Basic Research Project of Natural Science Foundation of Shandong Province, China (Grant No. ZR202010220025). Qingju Wang is funded by Huawei Technologies Co., Ltd (Agreement No.: YBN2020035184).

References

1. https://eprint.iacr.org/2021/1572.pdf.
2. https://www.sagemath.org/.
3. Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, 2015.
4. Achiya Bar-On and Nathan Keller. A 2^70 attack on the full MISTY1. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 435–456. Springer, 2016.
5. Christina Boura and Anne Canteaut. Another view of the division property. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 654–682. Springer, 2016.
6. Christina Boura and Daniel Coggia. Efficient MILP modelings for sboxes and linear layers of SPN ciphers. *IACR Trans. Symmetric Cryptol.*, 2020(3):327–361, 2020.
7. Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher square. In Eli Biham, editor, *FSE '97*, volume 1267 of *LNCS*, pages 149–165. Springer, 1997.
8. Patrick Derbez and Pierre-Alain Fouque. Increasing precision of division property. *IACR Trans. Symmetric Cryptol.*, 2020(4):173–194, 2020.
9. Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. Rasta: A cipher with low anddepth and few ands per bit. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 662–692. Springer, 2018.
10. Christoph Dobraunig, Yann Rotella, and Jan Schoone. Algebraic and higher-order differential cryptanalysis of pyjamask-96. *IACR Trans. Symmetric Cryptol.*, 2020(1):289–312, 2020.
11. Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David A. Wagner, and Doug Whiting. Improved cryptanalysis of rijndael. In Bruce Schneier, editor, *FSE 2000*, volume 1978 of *LNCS*, pages 213–230. Springer, 2000.
12. Dahmun Goudarzi, Jérémy Jean, Stefan Kölbl, Thomas Peyrin, Matthieu Rivain, Yu Sasaki, and Siang Meng Sim. Pyjamask: Block cipher and authenticated encryption with highly efficient masked implementation. *IACR Trans. Symmetric Cryptol.*, 2020(S1):31–59, 2020.
13. Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for three-subset division property without unknown subset - improved cube attacks against trivium and grain-128aead. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 466–495. Springer, 2020.
14. Phil Hebborn, Baptiste Lambin, Gregor Leander, and Yosuke Todo. Lower bounds on the degree of block ciphers. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 537–566. Springer, 2020.
15. Kai Hu, Siwei Sun, Meiqin Wang, and Qingju Wang. An algebraic formulation of the division property: Revisiting degree evaluations, cube attacks, and key-independent sums. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 446–476. Springer, 2020.
16. Lars R. Knudsen and David A. Wagner. Integral cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *FSE 2002*, volume 2365 of *LNCS*, pages 112–127. Springer, 2002.

17. Ted Krovetz and Phillip Rogaway. The OCB authenticated-encryption algorithm. *RFC*, 7253:1–19, 2014.
18. Baptiste Lambin, Patrick Derbez, and Pierre-Alain Fouque. Linearly equivalent s-boxes and the division property. *Des. Codes Cryptogr.*, 88(10):2207–2231, 2020.
19. Ling Sun, Wei Wang, and Meiqin Wang. Automatic search of bit-based division property for ARX ciphers and word-based division property. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 128–157. Springer, 2017.
20. Ling Sun, Wei Wang, and Meiqin Wang. Milp-aided bit-based division property for primitives with non-bit-permutation linear layers. *IET Inf. Secur.*, 14(1):12–20, 2020.
21. Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 158–178. Springer, 2014.
22. Wenqiang Tian and Bin Hu. Integral cryptanalysis on two block ciphers pyjamask and uBlock. *IET Inf. Secur.*, 14(5):572–579, 2020.
23. Yosuke Todo. Integral cryptanalysis on full MISTY1. In Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 413–432. Springer, 2015.
24. Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 287–314. Springer, 2015.
25. Yosuke Todo and Kazumaro Aoki. FFT key recovery for integral attack. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis G. Askoxylakis, editors, *Cryptology and Network Security - 13th International Conference, CANS 2014, Heraklion, Crete, Greece, October 22-24, 2014. Proceedings*, volume 8813 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.
26. Yosuke Todo and Masakatu Morii. Bit-based division property and application to simon family. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 357–377. Springer, 2016.
27. Qingju Wang, Lorenzo Grassi, and Christian Rechberger. Zero-sum partitions of PHOTON permutations. In Nigel P. Smart, editor, *CT-RSA 2018*, volume 10808 of *LNCS*, pages 279–299. Springer, 2018.
28. Qingju Wang, Yonglin Hao, Yosuke Todo, Chaoyun Li, Takanori Isobe, and Willi Meier. Improved division property based cube attacks exploiting algebraic properties of superpoly. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 275–305. Springer, 2018.
29. Senpeng Wang, Bin Hu, Jie Guan, Kai Zhang, and Tairong Shi. Milp-aided method of searching division property using three subsets and applications. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 398–427. Springer, 2019.
30. Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 648–678, 2016.