IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2020, No. 1, pp. 289–312. DOI:10.13154/tosc.v2020.i1.289-312

Algebraic and Higher-Order Differential Cryptanalysis of Pyjamask-96

Christoph Dobraunig[1], Yann Rotella[1,2] and Jan Schoone[1]

[1] Digital Security Group, Radboud University, Nijmegen, The Netherlands
[2] Laboratoire de Mathématiques de Versailles, UVSQ, CNRS, Université Paris-Saclay, Versailles, France

christoph@dobraunig.com, yann.rotella@uvsq.fr, j.schoone@cs.ru.nl

Abstract. Cryptographic competitions, like the ongoing NIST call for lightweight cryptography, always provide a thriving research environment, where new interesting ideas are proposed and new cryptographic insights are made. One proposal for this NIST call that is accepted for the second round is Pyjamask. Pyjamask is an authenticated encryption scheme that builds upon two block ciphers, Pyjamask-96 and Pyjamask-128, that aim to minimize the number of AND operations at the cost of a very strong linear layer. A side-effect of this goal is a slow growth in the algebraic degree. In this paper, we focus on the block cipher Pyjamask-96 and are able to provide a theoretical key-recovery attack reaching 14 (out of 14) rounds as well as a practical attack on 8 rounds. We do this by combining higher-order differentials with an in-depth analysis of the system of equations gotten for 2.5 rounds of Pyjamask-96. The AEAD-scheme Pyjamask itself is not threatened by the work in this paper.

Keywords: cryptanalysis · NIST call for lightweight cryptography · Pyjamask · algebraic cryptanalysis · higher-order differentials · symmetric cryptography

1 Introduction

Reducing the number of multiplications within cryptographic primitives is a quite recent trend that was initiated by LowMC [ARS+15], which lead to many interesting design approaches like FLIP [MJSC16], Kreyvium [CCF+18], MiMC [AGR+16], or Rasta [DEG+18]. The prime focus of these designs is to provide benefits in the areas of fully homomorphic encryption (FHE), multi-party computation (MPC), or post-quantum signature schemes [CDG+17, CDG+19]. Moreover, reducing the number of multiplications, or — more generally — the number of nonlinear building blocks provides benefits in the area of side-channel countermeasures. In particular, the costs of masking can be reduced as already brought forward by the block cipher Noekeon [DPVR00]. Pyjamask [GJK+19], an entry to the NIST lightweight call, follows this direction in a more aggressive manner than Noekeon and introduces two block ciphers Pyjamask-96 and Pyjamask-128, claiming to need the smallest number of binary multiplications (ANDs) per input bit processed, not considering designs like LowMC [ARS+15], or Rasta [DEG+18] that follow a more unconventional design approach. Note that there exist also other designs, which achieve a small number of AND per processed bit apart from block cipher based designs, e.g., constructions based on Xoofff [DHVV18], or the authenticated encryption schemes Ascon [DEMS19], Keyak [BDP+14], and Xoodyak [DHP+19].

The quest on reducing the number of multiplications has motivated cryptanalysts to look at these newly proposed constructions. This is because the nonlinear elements in ciphers provide the necessary confusion part in the confusion and diffusion duality [Sha49] that most of the modern designs still follow. Hence, new attacks and insights have

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

been published that took advantage of this, like, e.g., analysis that exploits the low algebraic degree of the functions [DEM15, DLMW15], the sparsity of nonlinear elements per encryption round [RST18], or other structural properties inherited from the quest to reduce the number of multiplications [DLR16].

The design of Pyjamask-96 follows quite conventional principles. We have the iterated application of a round function consisting of the parallel applications of 3-bit S-boxes, a linear diffusion layer, and the key addition where the round keys are derived via a linear key schedule. In contrast to other designs like LowMC [ARS+15], or Zorro [GGNPS13] that use incomplete S-box layers to reduce the number of nonlinear elements, Pyjamask-96 uses a complete one. Hence, the only option of Pyjamask-96 to reduce the number of multiplications within the circuit is to aggressively reduce the number of rounds. In this paper, we will analyze the effects of this aggressive reduction and present attacks on round-reduced-versions of Pyjamask-96 that take advantage of a combination of higher-order differential properties, together with the low monomial count we have in the equations that we want to solve. An overview of our results is shown in Table 1 and Table 2. Note that memory complexity is negligible, hence not included in the table. We want to emphasize that our attacks just focus on the block cipher Pyjamask-96 and that especially the attacks on the higher number of rounds do not apply when Pyjamask-96 is used in the context of the authenticated encryption scheme Pyjamask, partially due to the high data complexity. Hence, our attacks do not invalidate the security of any of the full-round authenticated encryption schemes specified in Pyjamask.

Table 1: Overview of the results on our key recovery attacks on Pyjamask-96.

Rounds	Time (in Pyjamask-96 calls)	Data (in blocks)	Reference
14/14	2^{115}	2^{96}	Section 6
13/14	2^{125}	2^{94}	Section 5
13/14	2^{99}	2^{96}	Section 6
12/14	2^{96}	2^{96}	Section 6
11/14	2^{91}	2^{95}	Section 7
10/14	2^{83}	2^{87}	Section 7
9/14	2^{67}	2^{71}	Section 7
8/14	2^{35}	2^{39}	Section 7
7/14	2^{27}	2^{23}	Section 7

Table 2: Overview of the results on our key recovery attacks on Pyjamask-96-AEAD.

Rounds	Time (in Pyjamask calls)	Data (in blocks)	Reference
7/14	2^{86}	2^{41}	Section 8

2 Description of Pyjamask

Pyjamask is a block cipher-based authenticated encryption scheme designed by Dahmun Goudarzi, Jérémy Jean, Stefan Kölbl, Thomas Peyrin, Matthieu Rivain, Yu Sasaki, and Siang Meng Sim, which has passed into the second round of the NIST lightweight competition. The mode of operation chosen by the authors of Pyjamask is the OCB AEAD mode of operation [KR14]. Pyjamask comes with two variants, one with a 128-bit block-size and one with a 96-bit block-size. Both use a 128-bit secret key.

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	x_{27}	x_{28}	x_{29}	x_{30}	x_{31}
x_{32}	x_{33}	x_{34}	x_{35}	x_{36}	x_{37}	x_{38}	x_{39}	x_{40}	x_{41}	x_{42}	x_{43}	x_{44}	x_{45}	x_{46}	x_{47}	x_{48}	x_{49}	x_{50}	x_{51}	x_{52}	x_{53}	x_{54}	x_{55}	x_{56}	x_{57}	x_{58}	x_{59}	x_{60}	x_{61}	x_{62}	x_{63}
x_{64}	x_{65}	x_{66}	x_{67}	x_{68}	x_{69}	x_{70}	x_{71}	x_{72}	x_{73}	x_{74}	x_{75}	x_{76}	x_{77}	x_{78}	x_{79}	x_{80}	x_{81}	x_{82}	x_{83}	x_{84}	x_{85}	x_{86}	x_{87}	x_{88}	x_{89}	x_{90}	x_{91}	x_{92}	x_{93}	x_{94}	x_{95}

Figure 1: The state of Pyjamask-96.

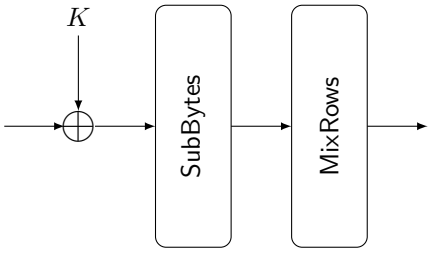

Figure 2: The round-function of Pyjamask-96.

In this paper, we focus on the analysis of the underlying block cipher with the 96-bit block-size, denoted as Pyjamask-96. Our attack is not directly applicable on Pyjamask-128 as the algebraic degree of Pyjamask-128 increases faster compared to Pyjamask-96 and the degree of the equations is one of the crucial points of our cryptanalysis. Hence, we only give the description of the 96-bit version Pyjamask-96. Pyjamask-96 is an iterated block cipher that consists of a data path and a key schedule. The key schedule is linear and the key state can be seen as a matrix of size 4×32. The data path consists of fourteen rounds that we will describe next. A state in the data path can be seen as an array of size 3×32 as in Figure 1.

2.1 Round function

The number of rounds of Pyjamask-96 is 14. One round function is applied on the 3×32 state and consists of the following operations, see also Figure 2:

- AddRoundKey: Bitwise addition of the round key defined by the key schedule.
- SubBytes: The same 3-bit S-box is applied to each of the 32 columns of the state.
- MixRows: For $i \in \{0, 1, 2\}$ we compute the rows of the updated state as $\boldsymbol{M}_i \cdot R_i^T$, where the matrices $\boldsymbol{M}_i$ are defined as 32×32 circulant binary matrices.

After the last round, another key addition (AddRoundKey) is performed.

2.1.1 The S-box

The 3-bit S-boxes of the SubBytes layer have Algebraic Normal Form (ANF): $S(x_0, x_1, x_2) = (y_0, y_1, y_2)$, where y_0, y_1, y_2 satisfy:

$$\begin{aligned} y_0 &:= x_0x_2 + x_1 \\ y_1 &:= x_0x_1 + x_0 + x_1 + x_2 \\ y_2 &:= x_1x_2 + x_0 + x_1 + 1 \end{aligned}$$

Hence, the S-box is a quadratic mapping. As the S-box is a 3-bit permutation, the inverse of the S-box is also quadratic. The ANFs of the 3 component functions of the inverse of

the S-box are given by:

$$\begin{aligned} x_0 &:= y_0y_1 + y_2 + 1 \\ x_1 &:= y_1y_2 + y_0 + y_1 + y_2 + 1 \\ x_2 &:= y_0y_2 + y_1 + y_2 + 1 \end{aligned}$$

2.1.2 MixRows

The MixRows layer of Pyjamask-96 consists of three binary circulant matrices of size 32×32 that are applied to the three 32-bit rows of the current state. The three matrices are defined as:

$$\begin{aligned} \boldsymbol{M}_0 &= \mathrm{cir}([1,1,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,1,1,1,0,0,0,1,0]), \\ \boldsymbol{M}_1 &= \mathrm{cir}([0,1,0,0,0,0,1,0,0,0,0,0,0,1,1,1,0,1,0,0,0,0,0,1,0,1,1,0,0,0,1,1]), \\ \boldsymbol{M}_2 &= \mathrm{cir}([0,0,0,0,0,0,0,0,1,0,1,0,0,1,1,1,1,1,0,0,1,1,0,1,0,0,1,0,0,1,0,1,1]), \end{aligned}$$

where cir is the function that creates a circular matrix with the input as its first row. The Hamming weight of the vectors that define the matrices is 11 for $\boldsymbol{M}_0$ and $\boldsymbol{M}_1$, and 13 for $\boldsymbol{M}_2$. The inverse of the MixRows layer can also be defined as the application of 3 circulant matrices. The Hamming weight of the vector that defines $\boldsymbol{M}_1^{-1}$ is 13. The two vectors defining $\boldsymbol{M}_0^{-1}$ and $\boldsymbol{M}_2^{-1}$ are of Hamming weight 11.

2.2 The Key Schedule

The key schedule of Pyjamask-96 determines round keys according to the following key schedule. As the state in Pyjamask-96 is a 3×32-bit array, a round key will always be determined by taking the first three rows of the key state. The key state is a 128-bit state (4 32-bit rows) and the first round key consists of just the first three rows. The remaining round keys are determined by the following key schedule. The first step consists of the application of a 4×4 matrix that operates independently on each column, allowing a bitslice implementation. This matrix is

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

The next step consists of applying a circulant matrix of size 32×32, defined as

$$\boldsymbol{M}_K = \mathrm{cir}([1,0,1,0,1,0,0,1,1,1,0,0,1,1,1,0,1,1,0,0,0,0,0,0,1,0,0,0,1,1,1,0]),$$

to the first row of the key state, while left-rotating the other rows in the key state by 8, 15 and 18 positions respectively. Finally, a four byte round constant is added bitwise to various parts of the state. This constant is given by the list

$$[0,0,1,0,0,1,0,0,0,0,1,1,1,1,1,1,0,1,1,0,1,0,1,0,1,0,0,0,*,*,*,*],$$

where the $(*,*,*,*)$ is the binary representation of the round number $n \in \{0,\ldots,13\}$. The first byte is added to the first byte of the last row. The second byte is added to second byte of the third row, and so on. Only the last half of the last byte changes throughout the rounds of the key schedule. Then the key schedule provides the first three rows as the new round key and the steps are iterated.

2.3 Implementation Cost of Pyjamask-96

As our upcoming analysis is a key recovery attack where the time complexity is dominated by binary operations, we estimate the costs of a software implementation of Pyjamask-96 for comparison.

- **Costs of SubBytes:** SubBytes uses a 3-bit S-box of degree two. Hence, we assume that a software implementation of this S-box needs at least 3 nonlinear operations to compute the degree two terms of the S-box in each output bit and 3 linear operations to add the linear parts to the output. Needing a total of 6 32-bit instructions for SubBytes.
- **Costs of MixRows:** MixRows consists of three matrix multiplications. In the following we only want to count the number of XOR operations needed to apply a matrix, neglecting the necessary storing or rotations. We assume that an implementation of one matrix multiplication needs at least 4 32-bit XOR operations since the hamming weight of the circular matrices involved is at least 11. Hence, we assume that the linear layer of Pyjamask-96 then needs $4 + 4 + 4 = 12$ 32-bit instructions.
- **Costs of AddRoundKey:** We assume that add key needs 3 32-bit XOR operations.
- **Costs of Key Schedule:** We assume that the cost of applying $\boldsymbol{M}$ is 6 32-bit XOR. We estimate the costs of the matrix multiplication with 4 32-bit instructions (as for the matrices of the linear layer). We do not count the 3 left-rotations, since they might be integrated in upcoming instructions. Finally, the constant addition requires 4 instructions. This counting gives us a total of $6 + 4 + 4 = 14$ 32-bit instructions per round (for the key schedule).

To sum up, we assume that one round of Pyjamask-96 needs at least $14+6+12+3 = 35$ 32-bit instructions. As Pyjamask-96 consists of 14 rounds, and one separate AddRoundKey, one evaluation of Pyjamask-96 for a given key and plaintext then needs at least $14 \cdot 35 + 3 = 493$ 32-bit instructions.

3 Attack principle

The small number of ANDs in Pyjamask is compensated by the cost of a strong linear layer, leading to very high diffusion after only a few rounds. Hence, it is likely that Pyjamask withstands attacks based on differential and linear cryptanalysis.

However, a low degree in the equations involved by the application of the cipher may lead to vulnerabilities with regard to higher-order differential cryptanalysis [Lai94], cube attacks [DS09] or algebraic cryptanalysis [Cou03, CM03].

Our idea is to combine integral distinguisher, algebraic cryptanalysis and Guess-and-Determine techniques, in order to attack Pyjamask-96. The attack is practical for a round-reduced version and academical for the full-round version. Figure 3 schematizes the principle of our cryptanalysis.

Integral distinguisher. The integral distinguisher (that we will explain in Section 4) allows us to distinguish 11 rounds of Pyjamask-96 from a random permutation with a probability close to 1. Using only degree arguments, we can distinguish up to only 10 rounds. However, thanks to the parallel application of the S-boxes, we can input specific vector spaces, allowing us to gain one more round (at the cost of making some pre-computation). Since the inverse of the S-box is also quadratic, we can apply the same distinguisher on the inverse of Pyjamask-96 for the same amount of rounds. Furthermore, since MixRows

comes after SubBytes and MixRows is linear, we can include that as "half" a round to get the following property:

$$\sum_{X \in \mathcal{X}} \mathsf{Pyj}_K^{-11.5}(X) = c$$

where $c \in \mathbb{F}_2^{96}$ is a precomputed constant (expressible in some key-bits) and $\mathcal{X}$ a subset of $\mathbb{F}_2^{96}$ that is specifically chosen (again, see Section 4). $\mathsf{Pyj}_K^{-11.5}$ denotes the last 11.5 rounds of Pyjamask backwards, or the first 11.5 rounds of $\mathsf{Pyjamask}^{-1}$.

Getting key-bits equations. We apply our distinguisher to a chosen set of ciphertexts, and solve a system of equations involving the first three round keys. Namely, we have to solve a system of equations of the form

$$\sum_{P \in \mathcal{P}} \mathsf{Pyj}_K^{2.5}(P) = c \tag{1}$$

where $\mathcal{P}$ denotes the corresponding subset of $\mathbb{F}_2^{96}$ defined as the pre-image of $\mathcal{X}$ through Pyjamask.

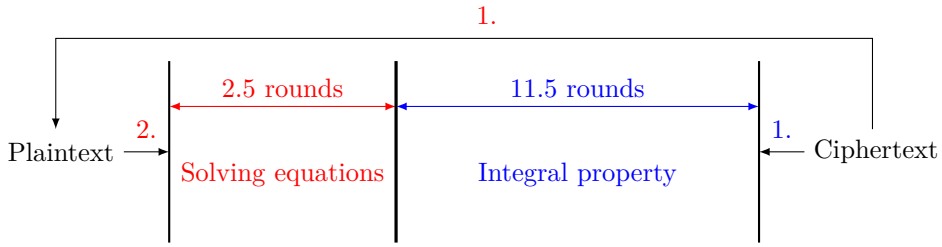

Figure 3: General setting of our cryptanalysis for the full-round attack. The integral property and the solving of equations.

Guess-and-Determine. If we want to attack 13 rounds (as described in Section 5), we can combine an 11 round distinguisher with a 1.5 round key recovery. After 1.5 rounds, a single output bit of the key schedule (or of the round function) does not depend on all cipher key-bits. So, we can apply a simple Guess-and-Determine technique in order to filter the keys faster than exhaustive search. However, for 2.5 rounds of Pyjamask a single output bit depends on almost all cipher key-bits, preventing us from using the simple Guess-and-Determine technique.

Solving polynomial equations. Although we cannot use the simple Guess-and-Determine technique, the number of monomials occurring in Equation 1 can be upper bounded and evaluated (see Section 6.2). This number of monomials occurring in the equations gives a bound on the number of bit-operations needed to solve the system by linearization technique. However, the number of possible equations we have is very limited. Finally, in order to solve the system, we need to combine Guess-and-Determine techniques and advanced linearization techniques in order to solve the corresponding system.

Complexity for attacking round-reduced versions. An advantage of our attack is that it can be also applied easily to a lower number of rounds. Hence, we can accurately compute the complexities of our cryptanalysis for round-reduced versions of Pyjamask-96 (see Table 1 for the complexities and Section 7 for the computation).

4 Integral distinguisher on 11 rounds

The core of our attack (and its variants) consists of applying an integral distinguisher with probability close to 1 on 11 rounds of Pyjamask-96. The reason for this is that in 11 rounds of Pyjamask-96, when using higher-order derivatives, we obtain expressions that always yield zero, while for a random permutation this has a very low probability.

4.1 Degrees of Pyjamask

As described by Boura et al. in [BCD11, BC13], there is an upper bound for the degree of the round-reduced functions. This upper bound will increase asymptotically to the length of the input. In particular, specific formulae for the upper bound are deduced. Using this, the authors of Pyjamask computed the upper bounds. We point out that the degrees computed by the designers [GJK+19] for Pyjamask-128 are too conservative with regards to the bounds provided by [BCD11]. With the bounds as in Table 3, we cannot mount this attack on Pyjamask-128. However, the bound of Pyjamask-96 matches [BCD11].

Table 3: Upper bounds on the degrees of each round of Pyjamask.

Round	1	2	3	4	5	6	7	8	9	10	11	12+
Pyjamask-96	2	4	8	16	32	64	80	88	92	94	95	95
Pyjamask-128	3	9	27	81	112	122	126	127	127	127	127	127

As you can see, the degree of any of the 96 component functions of Pyjamask-96 is upper bounded by 94 after 10 rounds. As the key schedule of Pyjamask is linear, any component function, seen as a Boolean function on $128 + 96 = 224$ variables, is of degree smaller than 94.

4.2 Reminder: Higher-order derivatives

Definition 1 (Derivative [Lai94])**.** For a Boolean function $F\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ and an element $a \in \mathbb{F}_2^n$ we can define the derivative of F with respect to a, $\Delta_a F$ as:

$$\Delta_a F(x) = F(x + a) + F(x).$$

In [Lai94], Lai also defined higher-order derivatives, that are obtained just by iterating the derivative. In particular, Lai proved that when iterating derivatives over linearly independent a_i we get

$$\Delta_{a_1}\Delta_{a_2}\cdots\Delta_{a_k}F(x) = \sum_{v\in V} F(x+v), \tag{2}$$

where V is the vector space spanned by $a_1, \ldots, a_k$.

Definition 2 (Higher-order derivatives)**.** For a Boolean function $F\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ and a vector space $V \subset \mathbb{F}_2^n$ we define the higher-order derivative with respect to V as the function $\Delta_V F$ given by:

$$\Delta_V F(x) = \sum_{v\in V} F(x+v).$$

For any Boolean function F, Lai also showed that $\deg \Delta_V F \leq \deg F - \dim V$. *A fortiori*, for any vector space V of dimension strictly greater than k, we have

$$\Delta_V F = 0\,.$$

We can replace vector space in the above with affine space, as we have

$$\sum_{v \in A} F(x+v) = \sum_{v \in V} F(x+v+b) = \Delta_V F(x+b).$$

Hence, as the component functions are of degree 94, we know from Equation 2 that for any affine space V of dimension 94 that

$$\sum_{v \in V} \mathsf{Pyj}_K^{10}(x+v)$$

is constant.

4.3 Our specific input affine spaces

Since the substitution layer of Pyjamask-96 consists of the parallel application of 3-bit S-boxes, we can input specific affine spaces of dimension 94 that will even yield that

$$\sum_{v \in V} \mathsf{Pyj}_K^{11}(x+v)$$

is constant.

We consider vector spaces of the form $\mathcal{V} = V_0 \bigoplus \mathcal{U}$, where we have $V_0 = \{\mathbf{0}, v_0\}$ with v_0 any non-zero vector of the form

$$v_{0,i} = \begin{cases} 0 & \text{if } i \not\equiv 0 \pmod{32}; \\ * & \text{if } i \equiv 0 \pmod{32}, \end{cases}$$

with $* \in \mathbb{F}_2$. Thus $\dim V_0 = 1$. For $\mathcal{U}$ we have the 93-dimensional vector space that has as a basis

$$\{e_1, \ldots, e_{31}, e_{33}, \ldots, e_{63}, e_{65}, \ldots, e_{95}\},$$

where e_i is the standard basis vector that is zero everywhere except in position i. Indeed, $\dim \mathcal{V} = \dim V_0 + \dim \mathcal{U} = 94$.

Theorem 1. *Let S denote a SubBytes layer that applies 3-bit S-boxes in parallel on columns of a state, let F denote any Boolean function of degree ≤ 94 and let $\mathcal{V}$ be as above. We write $x = (x_0, x_1, \ldots, x_{95})$ and $k = (k_0, k_1, \ldots, k_{95})$ in $\mathbb{F}_2^{96}$. Then the value of*

$$\sum_{v \in \mathcal{V}} (F \circ S)(x+k+v)$$

is constant and depends on at most 3 key-bits.

Proof. Since $\mathcal{V} = V_0 \oplus \mathcal{U}$ we get

$$\sum_{v \in \mathcal{V}} (F \circ S)(x+k+v) = \sum_{u \in \mathcal{U}} (F \circ S)(x+k+u) + (F \circ S)(x+k+v_0+u)$$

Now, we define $\overleftarrow{x}$ and $\vec{x}$ as

$$\begin{aligned} \overleftarrow{x} &= (x_0, 0, \ldots, 0, x_{32}, 0, \ldots, 0, x_{64}, 0, \ldots, 0) \\ \vec{x} &= (0, x_1, \ldots, x_{31}, 0, x_{33}, \ldots, x_{63}, 0, x_{65}, \ldots, x_{95}). \end{aligned}$$

The same notational conventions hold for k, u, v_0. Furthermore we introduce two maps $S_0, S' \colon \mathbb{F}_2^{96} \to \mathbb{F}_2^{96}$ where S_0 applies the S-box to the first column and the zero-mapping to

all other columns. The map S' then applies the zero-mapping to the first column and the S-box to all other columns. (In this definition, we assume that elements in 96 are again represented as 3×32 arrays.) Since the S-boxes are applied in parallel on columns, we get:

$$= \sum_{u \in \mathcal{U}} F(S_0(\overleftarrow{x} + \overleftarrow{k} + \overleftarrow{u}) + S'(\vec{x} + \vec{k} + \vec{u})) + F(S_0(\overleftarrow{x} + \overleftarrow{k} + \overleftarrow{v_0} + \overleftarrow{u}) + S'(\vec{x} + \vec{k} + \vec{v_0} + \vec{u}))$$

Since $\overleftarrow{u} = \mathbf{0}$ and $\vec{v_0} = \mathbf{0}$ we can leave them out:

$$= \sum_{u \in \mathcal{U}} F(S_0(\overleftarrow{x} + \overleftarrow{k}) + S'(\vec{x} + \vec{k} + \vec{u})) + F(S_0(\overleftarrow{x} + \overleftarrow{k} + \overleftarrow{v_0}) + S'(\vec{x} + \vec{k} + \vec{u}))$$

By writing $y = S_0(\overleftarrow{x} + \overleftarrow{k}), y + v_0' = S_0(\overleftarrow{x} + \overleftarrow{k} + \overleftarrow{v_0}), u' = S'(\vec{x} + \vec{k} + \vec{u})$, we get:

$$= \sum_{u' \in \mathcal{U}} F(y + u') + F(y + v_0' + u')$$

which is just

$$= \sum_{v' \in \mathcal{V}'} F(y + v')$$

where $\mathcal{V}' = \{\mathbf{0}, v_0'\} \oplus \mathcal{U}$. By the previous section, this result is a constant C, since $\deg F \leq \dim \mathcal{V}'$. This constant depends on $\mathcal{V}'$ and the value y. Since $\mathcal{U}$ does not depend on any of the key variables or plaintext variables, we see that the value of C only depends on at most 3 key-bits and 3 plaintext-bits. □

The previous result holds for Pyjamask-96 and thus for our attack, where F is just Pyj_K^{10}. It is now manageable to pre-compute the corresponding algebraic expression (on the 3 key-bits involved) by pre-computing all the following values:

$$\sum_{v \in \mathcal{V}} (F \circ S)(x + k + v)$$

for all possible v_0 as above, and $x \in \left(\mathbb{F}_2^{96} / \{0, v_0\}\right) \setminus \{0\}$ and k_0, k_{32}, k_{64} (all keys of the form $k = (*, 0, \ldots, 0, *, 0, \ldots, 0, *, 0, \ldots, 0)$, again $* \in \mathbb{F}_2$). We will then recover 7×3 expressions of $\sum_{v \in \mathcal{V}} (F \circ S)(x + k + v) = F(k_0, k_{32}, k_{64})$.

4.4 Why choose x to be non-zero?

As the main part of our attack is to recover simple algebraic equations, we also have to guarantee that the expressions we get by doing this are linearly independent. Note that Pyjamask-96 is a block cipher, so that it is a permutation when the key k is chosen. If we choose v_0 as in the preceding section, and define X to be

$$X = \left\{(x_0, 0, \ldots, 0, x_{32}, 0, \ldots, 0, x_{64}, 0, \ldots, 0) \in \mathbb{F}_2^{96} \mid (x_0, x_{32}, x_{64}) \in \mathbb{F}_2^3 / \{0, (v_{0,0}, v_{0,32}, v_{0,64})\}\right\}.$$

Then we have

$$\sum_{x \in X} \sum_{v \in \mathcal{V}} (\mathsf{Pyj}_K^{10} \circ S)(x + k + v) = \sum_{x \in \mathbb{F}_2^{96}} (\mathsf{Pyj}_K^{10} \circ S)(x) = 0$$

since choosing out of the four distinct x, the values in $\sum_{v \in \mathcal{V}} (\mathsf{Pyj}_K^{10} \circ S)(x + k + v)$ partition the entire $\mathbb{F}_2^{96}$ into four pairwise disjoint spaces of dimension 94. The sum over all elements in those four pairwise spaces together then yields a sum over all of $\mathbb{F}_2^{96}$.

This means that the four equations given by $\sum_{v \in \mathcal{V}} (\mathsf{Pyj}_K^{10} \circ S)(x + k + v)$ sum to zero. Hence any fourth is a linear combination of the other three.

Since we want linear independent equations, we need to choose one of those values and do not compute the sum over that x. We choose, without loss of generalization, x such that $(x_0, x_{32}, x_{64}) \in \mathbb{F}_2^3 / \{0, (v_{0,0}, v_{0,32}, v_{0,64})\}$ is not $\mathbf{0}$.

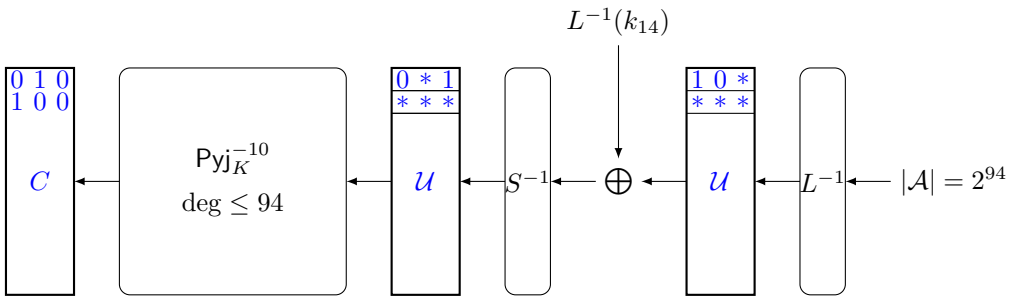

Figure 4: Integral distinguisher on 11.5 rounds of the decryption of Pyjamask-96, for $v_0 = 100$ (as defined in Section 4) and $L^{-1}(k_{14}) = 10 * * \cdots$.

4.5 What do we get?

Putting everything together, for one S-box, we pre-compute all values of the sum, leading to 7×3 expressions on 3 key variables. By doing a Gaussian elimination we delete the equations that are key dependent and keep the other equations. There are 3 key-bits involved, so only 7 different monomials exist. Hence, we can then recover at least $7 \times 3 - 7 = 14$ values of the corresponding sum, with an output being 0 or 1 which does not depend on the secret key. Collecting those values comes naturally with a price of pre-computation that is below the cost of our online attack.

As previously described, we collect our equations by evaluating 11.5 rounds of Pyjamask-96 (almost all Pyjamask-96) for 2^3 possible sub-keys, and thus for all 2^{94} sets present in the sum and also for all choices of v_0 and x. This means that we need to evaluate Pyjamask-96 on its whole codebook, for all possible sub-keys. This leads to a pre-computation complexity of

$$2^3 \times 2^{96} = 2^{99}$$

to get 14 different values. As we will need more equations, we will pre-compute this for all possible S-boxes, that means the pre-computation phase of our attack is then

$$2^3 \times 2^{96} \times 32 = 2^{104}$$

evaluations of Pyjamask-96 to get $14 \times 32 = 448$ equations. We can still assume that all those equations are linearly independent, as the input plaintext $\mathbf{0}$ is never evaluated.

4.6 Going in the other way

The integral property described above comes from the fact that we input affine spaces, and from the fact that the degree of 10 rounds is upper bounded by 94. Hence, we can use the same principle for the inverse of Pyjamask-96 by taking equivalent key k'_{14} as $L^{-1}(k_{14})$ and by changing the inputs to $L^{-1}(C)$ where L denotes MixRows and C the ciphertext.

The bounds on the degree are computed using the algebraic properties of the S-box. Since for a 3-bit S-box, the properties of S and S^{-1} are equivalent, this leads to the same upper bound values of the degrees for both decryption and encryption for Pyjamask-96. Eventually, we can extend the 11-rounds integral distinguisher of Pyjamask-96 to a 11.5 rounds of the inverse of Pyjamask-96 as the linear layer does not impact the degree. The description of this integral distinguisher is depicted in Figure 4.

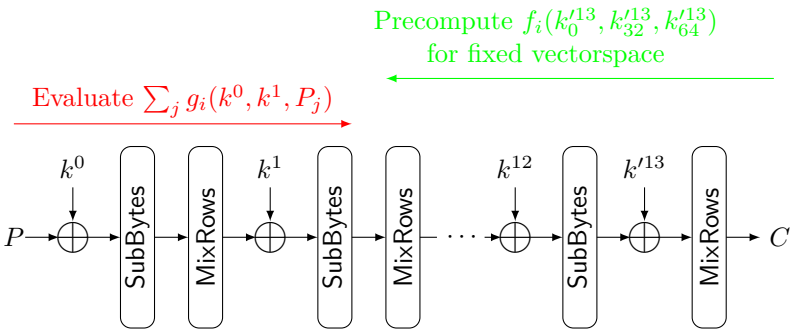

Figure 5: Principle of 13 round attack.

5 A simple attack on 13 rounds

With the knowledge of the previous section, we are able to mount a simple attack on 13 (out of 14) rounds of the block cipher Pyjamask-96. The attack is split into four phases and the concept is outlined in Figure 5. In the first pre-computation phase, we determine the three equations for the three equivalent key-bits $k_0'^{13}$, $k_{32}'^{13}$, and $k_{64}'^{13}$ for three bits of the output of a single S-box of the 2^{nd} round using a specific vector space $\mathcal{V}'$. In the second online phase, we query the plaintexts for the ciphertexts forming $\mathcal{V}'$. In the third phase, we set up three equations in plaintext and key-bits that express the three bits of the output of one S-box in the second round. Then, we compute the sum of them using the queried key-bits in order to end up with three equations just involving the key-bits. In the fourth phase, we solve those equations using brute force in order to reduce the key space by 2^{-3} on average. Next, we explain all steps in more detail.

Pre-computation. As outlined in Subsection 4.6, we can build a vector space $\mathcal{V}'$ in the ciphertexts at the end of the 13^{th} round, so that this space iterates over 94 bits of the outputs of the S-boxes of the 13^{th} round while for one S-box 2 bits are constant. Without loss of generality, we may assume that this S-box is the first S-box and the equivalent key-bits at the output of this S-box are $k_0'^{13}$, $k_{32}'^{13}$, and $k_{64}'^{13}$. When computing the sum of the bits when going 11.5 rounds backwards for $\mathcal{V}'$, this sum is a function $f_i(k_0'^{13}, k_{32}'^{13}, k_{64}'^{13})$, which is in general a different function for each bit. In the pre-computation phase, we aim to recover this function for three bits after going 11.5 rounds backwards (those are three bits at the output of a 2^{nd} round S-box). This can be done by evaluating $\mathcal{V}'$ for all possible assignments of $k_0'^{13}$, $k_{32}'^{13}$, and $k_{64}'^{13}$, while keeping the rest of the key constant. The costs of doing this are $2^3 \cdot 2^{94}$ 11.5 round inverse Pyjamask-96 evaluations.

Online phase. During the online phase, we query the oracle under attack for the vector space of ciphertexts forming $\mathcal{V}'$ and store the corresponding plaintexts. This needs a total of 2^{94} queries to the oracle and 2^{94} space for storing the plaintexts.

Getting equations in the key-bits. Now, it is time to compute the connection to the three bits after the 2^{nd} round S-box from the plaintext side, hence to find $\sum_j \left(g_i(k^0, k^1, P_j)\right)$ that connects with the respective $f_i(k_0'^{13}, k_{32}'^{13}, k_{64}'^{13})$. To do so, we have to set up three equations g_i and evaluate and sum them for all 2^{94} plaintexts. Due to the limited diffusion (see Figure 6) of the single linear layer we have to pass, only 66 bits of k^0 and 66 bits of P_j can appear in g_i, while we only have to take 3 bits of k^1 into account that influence a single S-box. Note that if we compute g_i connecting to the 3 output bits of the 2^{nd} round,

all 3 g_i involve the same bits of k^0, k^1, and P_j. Since the monomials of g_i are at most of degree 4, the maximum number of different monomials is upper bounded by:

$$\#\text{monomials in } g_i(k^0, k^1, P_j) \leq \sum_{i=0}^{4} \binom{66+66+3}{i} = 13\,643\,011 < 2^{23.71}.$$

As a consequence, computing the $\sum_j \left(g_i(k^0, k^1, P_j)\right) = h_i(k^0, k^1)$ for all three equations requires $3 \cdot 2^{23.71} \cdot 2^{94} < 2^{119.3}$ monomial evaluations and additions. The three equations can be stored in $3 \cdot 2^{23.71} < 2^{25.3}$ bits of space. At the end, we have three equations of the form $h_i(k^0, k^1) = f_i(k_0'^{13}, k_{32}'^{13}, k_{64}'^{13})$.

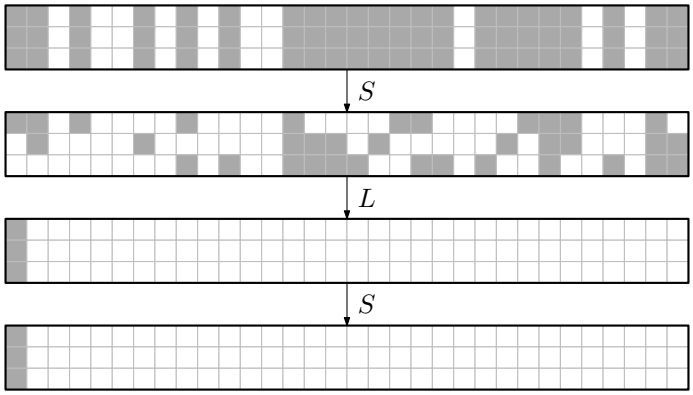

Figure 6: Information needed to compute the output bits of one S-box after 1.5 rounds.

Filter the key. Now we can use the three equations to filter wrong key candidates. Since all three equations use the same 72 bits of key material, we can brute-force for solutions. Again, we can upper bound the number of monomials in $h_i(k^0, k^1)$ by:

$$\#\text{monomials in } h_i(k^0, k^1) \leq \sum_{i=0}^{4} \binom{66+3}{i} = 919\,311 < 2^{19.82},$$

and the number of monomials in $f_i(k_0'^{13}, k_{32}'^{13}, k_{64}'^{13})$ with 2^3. Hence, the cost of reducing the key space by brute-forcing the key-bits is approximately $3 \cdot 2^{72} \cdot 2^{19.82} \cdot 2^3 < 2^{96.41}$ monomial evaluations plus summations.

Total costs of the attack. The most time consuming part of the attack (besides testing all 2^{125} key candidates at the end) is the set up of the equations in the key-bits. This takes approximately $2^{119.3}$ monomial evaluations and bit additions. As detailed in Subsection 2.3, 13 rounds of Pyjamask-96 need $13 \cdot 35 + 3 = 458$ 32-bit instructions. If we compare it with the bit instructions needed to evaluate the equations, this most expensive part translates to

$$\frac{2^{119.3} \cdot 4}{458 \cdot 32} < 2^{107.5}$$

13 round Pyjamask-96 evaluations. Indeed, to add one monomial, we need one XOR and at most 3 ANDs as the monomials are of degree smaller than 4. Moreover, those bit-operations can be converted to 32-bit operations by evaluating every plaintext in parallel over 32-bit registers. The attack gives us a 3-bit advantage on average. Hence, at the end it remains to test all 2^{125} key candidates, which needs $2^{125} + 2^{-96} \cdot 2^{125} \approx 2^{125}$ Pyjamask-96 evaluations.

The biggest space would be used during the online phase if all 2^{94} plaintexts would be stored. However, it is possible to directly use them when computing $\sum_j \left(g_i(k^0, k^1, P_j)\right)$, hence, the biggest space needed is to store the 3 equations $g_i(k^0, k^1, P_j)$ during the summations, which is $2^{25.3}$ bits, or $2^{25.3}/96 < 2^{18.72}$ Pyjamask-96 states. The data complexity is 2^{94} chosen ciphertext-plaintext pairs.

6 Attack on full-round Pyjamask-96

Instead of brute-forcing key-bits, an attacker could also try to solve the system of equations that we obtain after the evaluation of the whole codebook. However, solving polynomial equations is a quite hard problem in general. In symmetric cryptography, we often have plethora of equations (for instance if we take algebraic attacks on stream ciphers) and the critical problem for the cryptanalyst is the complexity of solving a system of high-degree equations. Here, we have a different context and are more in the context of [CDM+18], where the number of equations is much smaller than the number of monomials. Thus, a simple linearization technique will then more likely fail, as the number of monomials we have is much bigger than the size of the set of equations (which is 448). As the number of monomials in the algebraic normal form of one specific bit after 2.5 rounds is the critical point of our cryptanalysis, we evaluate it carefully in the following subsections.

6.1 Two critical metrics

The number of monomials present after 2.5 rounds (or 1.5 rounds for the 13-round attack) is critical for two reasons: first, we need to evaluate those monomials for almost the whole codebook; and second, ideally we want to solve the system of equations using linearization techniques.

The complexity of constructing the system of equations relies on the number of monomials formed with input plaintext variables while the complexity of solving the system relies on the number of monomials formed with secret key variables. The first type of monomials are called *evaluating monomials* and the second type *solving monomials*. Of course, the total number of monomials is at least as big as the number of evaluating monomials or solving monomials.

Example 1. Let us explain what happens with one round of Pyjamask-96 for a bit at position 0 in the state. We denote this bit as b_0. Variables are noted canonically as $K = (k_0, k_1, \ldots, k_{95})$ and $X = (x_0, x_1, \ldots, x_{95})$. By construction, we have $b_0 = (L \circ S)(X + K)$, for one value of plaintext X and round key K.

Hence, we have

$$b_0 = \sum_{i \in \mathcal{I}} (x_i + k_i)(x_{i+64} + k_{i+64}) + x_{i+32} + k_{i+32}$$

where $\mathcal{I}$ is defined by the linear layer: namely $\mathcal{I} = \{0, 1, 3, 8, 13, 18, 19, 24, 25, 26, 30\}$. In this expression of bit b_0, there are exactly $6 \times |\mathcal{I}| = 66$ monomials.

The equations we get are formed by the sum over 2^{94} input plaintexts for a fixed key. So we have only to evaluate the monomials where plaintext bits appear, namely $x_i x_{i+64}$, $x_i k_{i+64}$, $x_{i+64} k_i$ and x_{i+32} for all $\mathcal{I}$ and for all 2^{94} plaintexts. Hence we have 44 evaluating monomials. For our solving monomials, we have to consider terms that contain key-bits, that are $k_i k_{i+64}, x_i k_{i+64}, x_{i+64} k_i$ and k_{i+32} in our example. So we have 44 solving monomials.

6.2 Upper bound of the number of monomials

The cost of solving a system of equations of degree d with n variables is naturally bounded by the complexity of D^3 bit-operations and a data complexity close to D, where D is given by

$$D = \sum_{i=1}^{d} \binom{n}{i}.$$

Indeed, this is the classical cost of solving a linear system of equation, when considering all monomials of degree greater than 2 as new independent variables. However, this is a generic method and it does not use the inherent structure of the considered system.

Generic bound. For Pyjamask-96, we have 128 variables for the key and 96 variables for the plaintext. After 2.5 rounds, the degree of the multivariate polynomial is at most eight, so an upper bound for the number of monomials is

$$\sum_{i=1}^{8} \binom{96+128}{i} = 143\,811\,244\,303\,500 \approx 2^{47}.$$

The number of evaluating monomials is upper bounded by $N_e = \sum_{i=1}^{8} \binom{96}{i} \approx 2^{37}$ and the number of solving monomials is upper bounded by $N_s = \sum_{i=1}^{8} \binom{128}{i} \approx 2^{40.5}$. Then, the cost of the evaluation, in order to get the equations will be $2^{94} \times N_e \times 7$ (see Section 4). After this evaluation, one also need to solve a system of degree 8, with 128 variables, but with only 448 equations. Here, we see that just a degree argument will fail to attack Pyjamask-96, as the number of monomials is too large compared to the number of equations. We then have to take into account that we are not talking about ANY system of equations, we are talking about Pyjamask-96, which is a block cipher with quadratic 3-bit S-boxes.

In the following, we will then go into details in the structure of the system of equations that we have to solve.

Counting through layers. AddRoundKey of course adds one (linear) monomial to the expression. Then we consider the first SubBytes, where in the worst case the number of monomials gets to ten:

$$\begin{aligned} S(P+K)_1 &= (p_0+k_0)(p_1+k_1) + p_0 + k_0 + p_1 + k_1 + p_2 + k_2 \\ &= p_0p_1 + p_0k_1 + k_0p_1 + k_0k_1 + p_0 + k_0 + p_1 + k_1 + p_2 + k_2 \end{aligned}$$

MixRows consists of circulant matrices with Hamming weight 11 or 13 in the last row. Hence, it multiplies the number of monomials by 11 (or 13). So after one round, we have at most 110 monomials in the algebraic expression of every state-bit in the first two rows of the state and 130 monomials in the third row.

Then we have AddRoundKey. This operation adds 45 monomials to the first 32 coordinates, and 3 to the other 64. The 3 comes from the matrix $\boldsymbol{M}$, while the matrix $\boldsymbol{M}_K$ has Hamming weight 15.

At this point, we have at most 155 monomials. The second SubBytes can create up to 155^2 new monomials (by doing one AND) and $3 \cdot 155$ linear monomials. In total this gives an upper bound of 24 490 monomials.

Again, we multiply by 13 for MixRows and perform AddRoundKey. This operation adds now up to 128 monomials to the first 32 coordinates (as the key schedule reaches full diffusion for the first row after two rounds), and 51 to the other 64 coordinates. The upper bound we get is then $24490 \cdot 13 + 128 = 318\,498$. After applying the last SubBytes, we get $101\,441\,931\,498 \approx 2^{36.56}$ monomials.

We could also count the number of evaluating monomials together with the number of solving monomials more accurately, but as we will see, we can do a smarter thing, leading to the possibility to get the exact number of monomials and not only an upper bound.

Using quadratic properties. We can greatly improve the preceding estimation by adding new independent variables. In a general algebraic attack, one can linearize the nonlinear system, that means consider every nonlinear monomial as new independent variables and then solve a (much bigger) linear system. However, this is only done when there is no known special structure. For Pyjamask-96, we mainly use the fact that the nonlinear layer is quadratic to introduce new variables, drastically decreasing the number of appearing monomials in our equations. For SubBytes we have:

$$S(P+K) = S(P) + S(K) + L_P(K)$$

where $L_P(K)$ is a square matrix of size 96 with coefficients determined by the value of P, applied to K. As an example, we can focus on the first component:

$$\begin{aligned} S(P+K)_0 &= (p_0+k_0)(p_2+k_2) + p_1 + k_1 \\ &= p_0p_2 + p_1 + k_0k_2 + k_1 + p_0k_2 + p_2k_0 \\ &= S(P)_0 + S(K)_0 + p_0k_2 + p_2k_0. \end{aligned}$$

A similar expression holds for the other components as well, where one can notice that the matrix L_P defined above has, for every line and every column only 2 non-zero coefficients that depend on the value of P. For the last component of the S-box, a constant 1 has to be added. In order to reduce the expansion, we will consider this constant as part of the equivalent key $S(K)$.

If we then apply MixRows and another AddRoundKey, we obtain an expression of the form

$$(L \circ S)(P) + (L \circ S)(K_0) + K_1 + \sum_{\substack{i \in I \\ |I|=11,13}} p_i k_j + p_j k_i. \tag{3}$$

In order to reduce the minimal number of monomials occurring in an output bit, we now rename

$$\begin{aligned} \kappa &:= (L \circ S)(K_0) + K_1 \\ P' &:= (L \circ S)(P) \end{aligned}$$

So we introduce $|\kappa| = 96$ new key variables, and $|P'| = 96$ new plaintext variables.

We then continue by counting the number of monomials using the specific structures of Pyjamask-96. Before SubBytes, we have 28 monomials at maximum, one out of κ, one out of P', 26 of the form p_ik_j. The S-box then turns this into $28^2 + 3 \cdot 28 = 868$ monomials at maximum. MixRows gives an upper bound of $13 \cdot 868 = 11\,284$. Considering the third round key K_2 as a new independent key leads to an addition of exactly one (linear) monomial. Lastly, the third SubBytes gives an upper bound on the number of monomials to be

$$(11\,285)^2 + 3 \cdot (11\,285) = 127\,385\,080 \approx 2^{27}.$$

To sum up, the fact that one round is quadratic allows us to decrease the number of monomials by gathering nonlinear monomials that only depend on key variables, in new variables. This can also be done for the second round. Let us reuse Equation 3, and write

$$e_K^P = (L \circ S)(P) + \sum_{\substack{i \in I \\ |I|=11,13}} p_i k_j + p_j k_i.$$

Then, the state after one round can be written as

$$\kappa + e_K^P.$$

Eventually, after another round, the state equals

$$(L \circ S)(\kappa + e_K^P) + K_2 = (L \circ S)(\kappa) + K_2 + (L \circ S)(e_K^P) + \sum_{\substack{i \in I \\ |I|=11,13}} \kappa_j \left(e_K^P\right)_i + \kappa_i \left(e_K^P\right)_j .$$

We replace K_2 by $(L \circ S)(\kappa) + K_2$, allowing us to decrease the size of the system, attacking full Pyjamask-96.

6.3 Exact number of monomials

We could invest more time to upper bound the number of monomials. For instance, by separating which variables are taken into account, or by considering that the number of monomials of a certain degree is limited as explained in the following example.

Example 2. Let us reuse our example on one round of Pyjamask-96 (Example 1). We showed that the number of evaluating monomials after one round was 44, of which 11 are linear. If we use the fact that MixRows transforms these into $11 \cdot 13$ linear monomials, we end up with 143 linear terms. Clearly, this 143 is much higher than 96 (the number of plaintext variables), so we can decrease this 143 to 96. This has implications on the upper bounds.

As the upper bound found in the number of monomials happens to be practical (up to 2^{27}), we can investigate this on a computer and determine the bit corresponding to the equation with the least number of monomials.

Overestimate the number of solving monomials. As the number of solving monomials is a critical point in our attack, we also need some clarification about this number.

We considered all monomials involving key-bits, but this does not take into account that, in order to construct our system, we evaluate the function an even number of times (this is because we input affine spaces at the entry). Thus, we know that the monomials that do not involve plaintext variables will vanish.

Example 3. Let us continue on our example on one round of Pyjamask-96 (Examples 1 and 2). We have

$$b_0 = \sum_{i \in \mathcal{I}} (x_i + k_i)(x_{i+64} + k_{i+64}) + x_{i+32} + k_{i+32} .$$

Hence, $k_i k_{i+64}$ and k_{i+32} will never occur in the constructed system, this leads then to:

- a linear system in key variables (for one round);
- not 44 solving monomials, but 22.

Exact number of monomials. Using SageMath [The19] and the BooleanPolynomialRing library, we are able to construct the list of all monomials that can appear in the equation, using the quadratic properties of the round function. For the code see the supporting material on gitHub[1]. As we can construct this list, we can also find the exact number of solving monomials, together with the number of evaluating monomials. This allows us to determine the exact number of bit-operations needed in our attack.

[1] https://github.com/JJPSchoone/alg-att-pyjamask-supp

The output bit that has the least number of monomials when expressed as a Boolean function is bit 32. More precisely this is true for all state-bits from 32 to 63, as we omit the key-schedule and consider independent round keys. Hence, our metrics are shift-invariant. We also observed that the effective number of key-bits actually involved in the equation is smaller than expected. The results obtained are depicted in Table 4.

Table 4: Maximum number of monomials present in the system for state-bit 32. N_m is the total number of monomials, N_e is the number of monomials needed to evaluate for each plaintext, N_s is the maximum number of monomials present in the system after evaluation and $N_{keybits}$ is the effective number of (equivalent) keybits involved.

Rounds	N_m	N_e	N_s	$N_{keybits}$
1.5	648	571	569	56
2.5	7 642 713	3 910 569	3 829 480	154

In the attack from Section 5 we can substitute the numbers obtained via computer to improve the complexity of the attack. However, for the attack in Section 5, the complexity will still be dominated by 2^{125} Pyjamask-96 evaluations.

6.4 Guess-and-Determine: attacking full Pyjamask-96

A big challenge we face in our attack is the number of equations that we can get, which is limited to 448. Since the number of solving monomials is much bigger than the number of equations, we need to apply other techniques than a trivial linearization for solving the system. We choose a Guess-and-Determine strategy, already investigated for stream ciphers in [EJ02, HR00] and more recently in the context of solving polynomial equations in [DLR16, CDM+18]. This hybrid strategy (Guess-and-Determine), in Gröbner basis algorithms seems to be a very powerful technique when solving over small fields, which fits our cryptanalysis perfectly.

The idea is to guess some variables in order to drastically decrease the number of monomials appearing in our equations.

Example 4. Let us assume that we have the following equation:

$$k_0k_1k_2 + k_1k_2 + k_3 = 0\,.$$

Then, one way to reduce the number of monomials is to guess the value of k_0. If k_0 equals one, the equation we get is $k_3 = 0$. This gives directly the potential set of solutions $1 * *0$. If $k_0 = 0$, we recover $k_1k_2 + k_3 = 0$. Then we guess k_3. If $k_3 = 1$, then $k_1 = k_2 = 1$. If not, we have $k_1k_2 = 0$.

We see by guessing $k_0 = 1$ we reduce the number of possibilities from 16 to 4, and by guessing $k_0 = 0$ we reduce to 3 possibilities.

In larger examples, we also greatly decrease the search space when guessing a few well-chosen bits. The idea is really simple and is the same that is used in [DLR16, CDM+18]. We guess enough key-bits to guarantee that the number of monomials present in the system is smaller than the number of equations we have in our possession.

Computer investigation. Using Sage, we are able to compute a choice of guesses, such that the number of monomials drastically decreases. The algorithm we implement is a greedy algorithm and works as follows. After computing the list of all monomials potentially present in the system of equations, we quotient this set of monomials by every variable, and take as a first guess the variable for which the quotient set of monomials has the smallest size. We continue this process until we are below 448 monomials.

A (much) better guessing strategy on 13 rounds. For the 13 rounds attack described in Section 5, we needed to brute-force all key-bits involved in the equation. However, the number of monomials (for 13 rounds) that can appear in our equations is very close to 448. It appears that, by guessing only 4 bits (namely bit 1, 13, 25 and 26), the number of monomials decreases directly to 411.

Eventually, we have to guess 4 bits, evaluate those bits in the monomials where they appear (which is bounded by $4 + 4 \cdot 56$) as for 1.5 rounds, the degree is bounded by 2 after linearization and there are only 56 key-bits involved.

As explained in Section 5, for each plaintext, we have to compute one round of Pyjamask-96 without the key schedule (new variables are used to decrease the number of monomials). This costs 35 32-bit instructions. Then, we have to evaluate the monomials in the plaintext bits. For the evaluating monomials, we know that, after linearization and adding new variables, all monomials that we need to evaluate are of degree smaller than 2. Hence the cost of constructing the system is given by

$$\frac{2^{96} \cdot (35 \cdot 32 + 2 \cdot N_e)}{458 \cdot 32} \leq 2^{93.3}$$

Pyjamask-96 evaluations.

One can notice that the cost of solving the system is roughly $2^4(4 + 4 \cdot 56 + 411^3) \approx 2^{30}$ bit-operations, where we bounded Gaussian elimination on an $n \times n$ matrix by $\mathcal{O}(n^3)$. Solving the system needs to be done at most 2^4 times, since we guess 4 bits. Then we need to evaluate our equations for those guesses. The number of places where that needs to be done is $4 + 4 \cdot 56$, since the monomials are at most of degree 2. The cost of solving the system is then much smaller than the cost of constructing the system.

Cost of system building for 14 rounds. For 14 rounds, we first have to construct the system of 448 equations. We have 2^{96} online queries. For all plaintexts, we have to evaluate the monomials present in our system. However, we introduced new variables, corresponding to $(L \circ S)(P)$. Hence, for all 2^{96} plaintexts, we have to apply one round of Pyjamask-96 without key addition. Then, the cost of this is $2^{96} \cdot 35$ 32-bit operations. Afterwards, we need to evaluate the monomials present in the system. One can notice that introducing new variables linearizes the monomials in only key or only plaintext variables, guaranteeing that the evaluating monomials (and solving monomials) have a degree smaller than 4. Eventually, constructing the system has a cost of

$$2^{96} \cdot (35 \cdot 32 + 4 \cdot 3910569 \cdot 411) \leq 2^{128.6}$$

bit-operations. Counting using bit-operations gives a complexity bigger than 2^{128}. However, if we compare to the exhaustive search of Pyjamask-96, the complexity for constructing the system is

$$\frac{2^{128.6}}{493 \cdot 32} \leq 2^{114.7}$$

Pyjamask-96 evaluations.

Cost of system solving for 14 rounds. After constructing the system, we have to solve it. To do so, we first guess the first 96 bits (k^0). It happens (and it is natural) that the remaining maximum number of monomials is 569, as we reduced the system to the 1.5 round attack described previously with a cost of 2^{96} guesses. Naturally, by guessing 4 more bits, namely the same as for 1.5 rounds: $k^1_1 = \kappa_1$, $k^1_{13} = \kappa_{13}$, $k^1_{25} = \kappa_{25}$, $k^1_{26} = \kappa_{26}$, we get a system with at most 411 monomials, with a cost of $2^{95} \cdot 2^4 = 2^{100}$ guesses.

Eventually, the cost (in bit-operations) the complexity for solving the system is

$$2^{100} \cdot (3829480 \cdot 4 \cdot 411 + 411^3) \leq 2^{132.6}$$

bit-operations. The ·4 comes naturally from the fact that all monomials are of degree smaller than 4.

We can improve the previous complexity by making key guesses in a smarter way, by ordering the guesses such that only one bit changes per iteration. This is an example of a Gray code as first described in [Gra47].

Since we only change one bit in each step, only the monomial where this bit occurs have to be evaluated again, since we can retain the older values.

Using our program, we computed the average number of monomials needed to be re-evaluated for all variables that we guess. (See again the supporting material on gitHub[2].) This value is 132767.66. We can now substitute the 3829480 for this much smaller number.

Eventually, all of this can be done using 32-bit instructions, and for each guess we solve a system of size 411 using linearization techniques. If we compare to the exhaustive search of Pyjamask-96, the complexity for solving the system is

$$\frac{2^{100} \cdot (132767.66 \cdot 4 \cdot 411 + 411^3)}{493 \cdot 32} \leq 2^{114.2}$$

Pyjamask-96 evaluations.

The data complexity is just 2^{96} as we need that many queries to the Pyjamask-96 oracle. For the memory complexity that we get, we need to store the three g_i equations during the summations, like we did in the simple attack of Section 5. So here, that is bounded by $2^{30.11}/96 < 2^{23.53}$ Pyjamask-96 states.

7 Round-reduced attacks

As detailed in the previous section, our full-round attack requires the full code book of the block cipher which is very unpractical. When analyzing the security of block ciphers, it is important to know the best attack that is below the security claim, but also to see the best attack that can be practical in order to understand the security level better.

In our case, this is quite straightforward. Indeed, the main complexity of our attack comes from the data complexity, which is directly derived from the integral distinguisher. When reducing the number of rounds, we can obtain the zero-sum property using much less data.

For 14, 13 and 12 rounds, we still need the whole code book to get enough equations. For the other round-reduced versions, we can decrease the data complexity. To keep it simple, we fix the number of equations that we need to collect to $128 = 2^7$. We know that we gain one round for free for the integral distinguisher and we choose to solve a system corresponding to 1.5 rounds . Using Table 3, we get the following complexities. For an 11-round attack, we use an integral distinguisher on $11 - 2 - 1 = 8$ rounds (one round is for free, the two others are used to build the system of equations), meaning that we input affine spaces of dimension 88. As we need 2^7 equations, the data complexity becomes $2^7 \cdot 2^{88} = 2^{95}$. For a 10 round-attack, we need an integral distinguisher on 7 rounds, leading to a data complexity of $2^{80} \cdot 2^7 = 2^{87}$.

Even though we choose to collect only 128 equations, we still get a system of equations with much more monomials. However, by looking carefully at the system, only 19 bits out of 56 shall be guessed, in order to guarantee a number of monomials below 128. The bits that we need to guess are at positions 1, 13, 25, 26, 0, 3, 8, 18, 19, 24, 29, 64, 65, 67, 72, 77, 82, 83 and 88, such that only 114 monomials remain.

[2] https://github.com/JJPSchoone/alg-att-pyjamask-supp

The pre-computation part costs $2^c \cdot 2^3$ Pyjamask-96-calls, where c is the dimension of the affine spaces we use for integral distinguisher (88 for 11 rounds, 80 for 10 rounds, 64 for 9 rounds, 32 for 8 rounds and 16 for 7 rounds). The 2^3 comes from the fact that we consider three equivalent key-bits. (See Subsection 4.5.)

The cost of constructing the system is $2^c(35 \cdot 32 + 2 \cdot 571)$ bit-operations (we have to apply one round of Pyjamask-96 and evaluate the 571 monomials).

The cost for solving the system is the cost for evaluating those 19 bits which is smaller than $(19 + 19 \cdot 56)$ (the maximal number of monomials of degree 2 involving those bits) plus the cost for solving the system with 128 equations, and we have to multiply the sum by 2^{19} as we repeat this for each guess. The cost of solving the equation is then bounded by

$$2^{19}(19 + 19 \cdot 56 + 128^3) \approx 2^{40}$$

bit-operations for all round-reduced version attacked.

Example 5. *(8 rounds of Pyjamask-96)* Since the dimension of the affine spaces for 8 rounds is 32, the pre-computation phase of the attack takes $2^3 \cdot 2^{32}$ Pyjamask-96-calls.

The construction phase then costs $2^{32}(35 \cdot 32 + 2 \cdot 571) \approx 2^{43.14}$ bit-operations. To express this in Pyjamask-96 evaluations, we see that the cost of 8 rounds of Pyjamask-96 gives $8 \cdot 35 + 3$ 32-bit operations. Hence we get

$$\frac{2^{32}(35 \cdot 32 + 2 \cdot 571)}{283 \cdot 32} \approx 2^{29.99}$$

Pyjamask-96-calls.

For the solving of the system we do the same computation and get to $2^{26.9}$ Pyjamask-96-calls.

So the time complexity for the attack on 8 rounds of Pyjamask-96 is bounded by 2^{35}.

8 Application on Pyjamask-96-AEAD

So far, we have focused our analysis solely on the 96-bit block cipher Pyjamask-96. As the NIST candidate Pyjamask-96-AEAD uses Pyjamask-96 in OCB3 [KR14], we discuss how our attack translates to this use. The first restriction lies in the data limit that we have to use less than 2^{48} blocks. We can then apply our attacks needing less than 2^{48} data to Pyjamask-96-AEAD, but with a loss of 2 rounds. In the following, we explain why.

8.1 Losing one round inputing affine spaces

In OCB3, blocks of messages are processed with independent block cipher calls using different masks added to the input and output. Those masks are formed using the following rule: we compute a 96-bit value that depends nonlinearly on the nonce and the key (let us say $O_0(K, N)$ for simplicity here), and another 96-bit value depending only on the key ($E_K(0)$). From this second value, we form a basis of $\mathbb{F}_2^{96}$ denoted by $L_0, L_1, L_2, \ldots, L_{95}$. Each time a new mask is used (for block number i), one has to add $L_{ntz(i)}$ to previously computed mask, where $ntz(i)$ is the number of trailing zero bits in the binary representation of i.

Hence, from block 1 to block $2^k - 1$, $k < 95$, every mask is of the form

$$O_0(K, N) + \sum_{i=0}^{k-1} a_i L_i$$

where $a_i \in \mathbb{F}_2$. This is because, if $i < 2^k - 1$, then $ntz(i) < k$. Moreover, at least one a_i is always non-zero in each expression of the masks. Hence, from $2^k - 1$ blocks, we can choose plenty of affine spaces of dimension $k - 1$, by just choosing one mask value as offset. One then needs to carefully choose corresponding blocks that form an affine space of dimension $k - 1$. We cannot have an affine space of dimension k with 2^k blocks, as L_k will appear in the expression of the later block.

We can then process integral distinguisher using OCB mode, but as we do not know the key, we do not know the value of the masks. Thus we cannot input specific affine spaces. Hence, what we described in Section 4 is not possible respecting the OCB mode. So we lose one round. However, as we are not restricted to specific affine spaces, the number of equations we get will not be critical anymore.

8.2 Losing one round for system solving

In the system solving part, it appeared that a lot of monomials occur in our system. The masks added at the output will also increase the number of monomials, since masks are formed using $E_K(0)$. As we cannot easily express the masks in key bits, we have to consider $E_K(0)$ as new independent keys for solving the system. Therefore, the estimated number of monomials given in Section 6.2 is not valid, but will increase, making a system obtained from 2.5 rounds very hard to solve. However, for 2 rounds, the degree is only 4, and an upper bound on the number of monomials will show that system is solvable.

Let us consider the key used in Pyjamask-96 as 128 bits. Let us also consider mask $O_0(K, N)$ to be 96 new variables, and write L_0 for a new 96-bit value independent from the others. Every mask can then be written as a linear combination of L_0 and $O_0(K, N)$. As a result, the number of evaluating monomials is the same since it comes from the known values (plaintext as we go in the backward direction). However, the system we would have to solve is of degree 4, with 128 variables coming from the key, 96 variables coming from $O_0(K, N)$ and 96 variables coming from L_0. Thus, the number of solving monomials will be bounded by

$$\sum_{i=1}^{4} \binom{320}{i} \leq 2^{28.7}.$$

Note that this is a crude upper bound, improving on this bound might improve the time complexity of the attack.

If we want a zero sum after 5 rounds, we need an affine space of dimension 33. Per affine space we get 96 equations that we can use for solving. Using 40 basis vectors, we can gather $96 \cdot \binom{40}{33} \approx 2^{30.7}$ equations. To sum up, considering the data limit of 2^{48}, we can get distinguishers for a maximum of 5 rounds and can add 2 rounds of key recovery. Hence, for Pyjamask-96-AEAD, we can target $5 + 2 = 7$ rounds.

9 Conclusion

In this paper we investigated the security of Pyjamask, a candidate participating in NIST's lightweight competition. Our focus was on the block-cipher Pyjamask-96 underlying one variant of Pyjamask and we were able to show a full-round (14 rounds) attack using the whole codebook of the block cipher (2^{96} blocks in data complexity). However, the cost of our attack is approximately $2^{114.7}$ Pyjamask-96 evaluations, which is below a brute-force key search for Pyjamask-96 128-bit key. Furthermore, we give a practical attack for Pyjamask-96 reduced to 8 rounds.

In order to make the attack practical, one needs to decrease the size of input vector spaces. A possible topic for further research is to investigate on how to improve upper

bounds on the degree or to find integral distinguishers that are not related with the degree of the permutation.

Our attacks exploit the algebraic properties of Pyjamask-96 through higher-order derivatives, combined with a guess-and-determine strategy for solving linearized systems of monomials. In particular, the rather small block-size of Pyjamask-96 combined with its low-degree round function leads to a rather slow increase of the algebraic degree, which we exploit in our attacks. Furthermore, the fact that a round is quadratic allows us to work with equivalent keys to gain one round in the backward direction (almost) for free.

The AEAD-scheme Pyjamask has an extra layer of protection in the form of the mode (OCB) that ensures that it is not threatened by the work we discussed in this paper.

Acknowledgments

Christoph Dobraunig is supported by the Austrian Science Fund (FWF): J 4277-N38. Yann Rotella and Jan Schoone are supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA.

References

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016*, volume 10031 of *LNCS*, pages 191–219, 2016.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 430–454. Springer, 2015.

[BC13] Christina Boura and Anne Canteaut. On the influence of the algebraic degree of F^{-1} on the algebraic degree of $G \circ F$. *IEEE Trans. Information Theory*, 59(1):691–702, 2013.

[BCD11] Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-order differential properties of keccak and *Luffa*. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011*, volume 6733 of *LNCS*, pages 252–269. Springer, 2011.

[BDP+14] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Keyak, 2014.

[CCF+18] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. *J. Cryptology*, 31(3):885–916, 2018.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 1825–1842. ACM, 2017.

[CDG+19] Melissa Chase, David Derler, Steven Goldfeder, Jonathan Katz, Vladimir Kolesnikov, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Xiao Wang, and Greg Zaverucha. The picnic signature scheme, 2019. Submission to the NIST Post-Quantum Cryptography Standardization Process.

[CDM+18] Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux, Mélissa Rossi, and Yann Rotella. On the concrete security of goldreich's pseudorandom generator. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018*, volume 11273 of *LNCS*, pages 96–124. Springer, 2018.

[CM03] Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 345–359. Springer, 2003.

[Cou03] Nicolas Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *LNCS*, pages 176–194. Springer, 2003.

[DEG+18] 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, *Advances in Cryptology - CRYPTO 2018*, volume 10991 of *LNCS*, pages 662–692. Springer, 2018.

[DEM15] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Higher-order cryptanalysis of lowmc. In Soonhak Kwon and Aaram Yun, editors, *Information Security and Cryptology - ICISC 2015*, volume 9558 of *LNCS*, pages 87–101. Springer, 2015.

[DEMS19] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2, 2019. Submission to NIST lighweight competition.

[DHP+19] Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Xoodyak, a lightweight cryptographic scheme, 2019. Submission to NIST lighweight competition.

[DHVV18] Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. The design of xoodoo and xoofff. *IACR Trans. Symmetric Cryptol.*, 2018(4):1–38, 2018.

[DLMW15] Itai Dinur, Yunwen Liu, Willi Meier, and Qingju Wang. Optimized interpolation attacks on lowmc. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015*, volume 9453 of *LNCS*, pages 535–560. Springer, 2015.

[DLR16] Sébastien Duval, Virginie Lallemand, and Yann Rotella. Cryptanalysis of the FLIP family of stream ciphers. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016*, volume 9814 of *LNCS*, pages 457–475. Springer, 2016.

[DPVR00] Joan Daemen, Michaël Peeters, Gilles Van Assche, and Vincent Rijmen. The NOEKEON block cipher, 2000. Nessie Proposal.

[DS09] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 278–299. Springer, 2009.

[EJ02] Patrik Ekdahl and Thomas Johansson. A new version of the stream cipher SNOW. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography, SAC 2002*, volume 2595 of *LNCS*, pages 47–61. Springer, 2002.

[GGNPS13] Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. Block ciphers that are easier to mask: How far can we go? In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013*, volume 8086 of *LNCS*, pages 383–399. Springer, 2013.

[GJK+19] Dahmun Goudarzi, Jérémy Jean, Stefan Kölbl, Thomas Peyrin, Matthieu Rivain, Yu Sasaki, and Siang Meng Sim. Pyjamask v 1.0, 2019. Submission to NIST lighweight competition.

[Gra47] Frank Gray. Pulse code communication, November 1947. US Patent 2,632,058.

[HR00] Philip Hawkes and Gregory G. Rose. Exploiting multiples of the connection polynomial in word-oriented stream ciphers. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 303–316. Springer, 2000.

[KR14] Ted Krovetz and Phillip Rogaway. The OCB authenticated-encryption algorithm. *RFC*, 7253:1–19, 2014.

[Lai94] Xuejia Lai. *Higher Order Derivatives and Differential Cryptanalysis*, pages 227–233. Springer US, Boston, MA, 1994.

[MJSC16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards stream ciphers for efficient FHE with low-noise ciphertexts. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016*, volume 9665 of *LNCS*, pages 311–343. Springer, 2016.

[RST18] Christian Rechberger, Hadi Soleimany, and Tyge Tiessen. Cryptanalysis of low-data instances of full lowmcv2. *IACR Trans. Symmetric Cryptol.*, 2018(3):163–181, 2018.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715, 1949.

[The19] The Sage Developers. *SageMath, the Sage Mathematics Software System*, 2019. `https://www.sagemath.org`.