Key-Recovery Attacks on Full KRAVATTE

Colin Chaigneau[1], Thomas Fuhr[2], Henri Gilbert[1,2], Jian Guo[3], Jérémy Jean[2], Jean-René Reinhard[2] and Ling Song[3,4]

[1] UVSQ, Versailles, France
Colin.Chaigneau@uvsq.fr

[2] ANSSI Crypto Lab
51, boulevard de La Tour-Maubourg
75700 Paris 07 SP
<Firstname.Lastname>@ssi.gouv.fr

[3] Nanyang Technological University, Singapore
{guojian,songling}@ntu.edu.sg

[4] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, China

Abstract. This paper presents a cryptanalysis of full KRAVATTE, an instantiation of the `Farfalle` construction of a pseudorandom function (PRF) with variable input and output length. This new construction, proposed by Bertoni et al., introduces an efficiently parallelizable and extremely versatile building block for the design of symmetric mechanisms, e.g. message authentication codes or stream ciphers. It relies on a set of permutations and on so-called rolling functions: it can be split into a compression layer followed by a two-step expansion layer. The key is expanded and used to mask the inputs and outputs of the construction. KRAVATTE instantiates `Farfalle` using linear rolling functions and permutations obtained by iterating the `Keccak` round function.

We develop in this paper several attacks against this PRF, based on three different attack strategies that bypass part of the construction and target a reduced number of permutation rounds. A higher order differential distinguisher exploits the possibility to build an affine space of values in the cipher state after the compression layer. An algebraic meet-in-the-middle attack can be mounted on the second step of the expansion layer. Finally, due to the linearity of the rolling function and the low algebraic degree of the `Keccak` round function, a linear recurrence distinguisher can be found on intermediate states of the second step of the expansion layer. All the attacks rely on the ability to invert a small number of the final rounds of the construction. In particular, the last two rounds of the construction together with the final masking by the key can be algebraically inverted, which allows to recover the key.

The complexities of the attacks devised are far below the security claimed for the latest published KRAVATTE specifications published on the IACR ePrint and for a strengthened version of KRAVATTE that has been recently presented at ECC 2017.

Keywords: Cryptanalysis · Higher Order Differential · Algebraic Attack · Linearization

1 Introduction

`Farfalle` is an efficiently parallelizable permutation-based construction of a variable input and output length *pseudorandom function* (PRF) recently proposed by Bertoni et al. [BDH+16]. It represents an extremely versatile building block for the design of

symmetric mechanisms. It can indeed be used either directly as a message authentication code (MAC), as the keystream generation part of a stream cipher, as a key derivation function (KDF), or otherwise in a mode of operation allowing to convert it into a more complex mechanism, for instance an authenticated encryption scheme or a block cipher of variable block length.

`Farfalle` takes as input a key and a (sequence of) data string(s) of arbitrary length(s) and produces an output of arbitrary length. Its construction involves two basic ingredients: a set of *permutations* of a b-bit state, and a family of so-called *rolling functions* used to derive distinct b-bit *mask values* from a b-bit *mask key* or more generally b-bit variants of a b-bit state.

The `Farfalle` construction consists of a *compression layer* followed by an *expansion layer*. The compression layer produces a single b-bit *accumulator value* from a tuple of b-bit blocks representing the input data. The expansion layer first non-linearly transforms the accumulator value into a b-bit *rolling state* and then non-linearly transforms a tuple of variants of this rolling state, produced by iterating the rolling function into a tuple of (truncated) b-bit output blocks. Both the compression and the expansion layer involve b-bit mask values derived from the key by the *key derivation* part of the construction.

An efficient instantiation of the `Farfalle` construction named KRAVATTE is also specified in [BDH+16, version 20170717:134002]. It is depicted in Figure 1. The underlying components are a set of `Keccak-`p permutations of a $b = 1600$-bit state, and a family of simple $\mathbb{F}_2$-linear rolling functions. The variants of KRAVATTE are addressed according to the number of rounds in the internal permutations: n_b rounds for the key derivation, n_c rounds for the compression layer, n_d rounds for the non-linear transformation applied of the accumulator, and n_e rounds for the expansion layer. The specifications of KRAVATTE published on the IACR ePrint in July 2017 use $(n_b, n_c, n_d, n_e) = (6, 6, 4, 4)$ and an announcement at ECC 2017 of a strengthened variant [BDH+17b] uses $(n_b, n_c, n_d, n_e) = (6, 6, 6, 6)$.

Our Contributions

In this paper, we present three families of attacks against full KRAVATTE, whose time and data complexities are far below the security claimed by the designers. Furthermore, one of them can even be successfully applied to strengthened variants of KRAVATTE.

The first two attack strategies focus on the expansion layer *after* the application of its initial non-linear transformation. They exploit that all output blocks are generated from the same initial rolling state, and the small number of `Keccak-`p rounds between the rolling state diversification and the block outputs. They require a long KRAVATTE output generated from a single and possibly unknown message. The compression layer and the derivation of the rolling state from the accumulator value do not contribute any security against these attacks. The third strategy focuses on a property of the compression layer.

Meet-in-the-Middle Algebraic Attack. The first attack can be seen as a meet-in-the-middle (MITM) algebraic attack, and bears some resemblance to the meet-in-the-middle approach applied to interpolation attacks [JK97]. The rolling state and the output masking key are the unknowns of an algebraic system built by forming expressions of the same intermediate state, either by forward computation from the rolling state, or by backward computation from the output. The expansion mechanism makes it possible to collect enough equations to solve the system by linearization.

Linear Recurrence Distinguisher. The second attack strategy leverages the linear branch diversification mechanism of the expansion layer: the rolling state can be assimilated to a short LFSR state, due to the restriction of the rolling function to only 320 bits of the 1600-bit state. As a consequence, the linear complexity of the sequence of blocks obtained by application to consecutive rolling state values of a small number of the low-degree

`Keccak` round function is also limited, i.e., this sequence satisfies a linear recurrence of order far smaller than what is expected from the size of the state. Furthermore, the recurrence polynomial of this sequence of blocks can be derived at a moderate cost. This observation provides the linear recurrence distinguisher used in our attack.

Higher Order Differential Distinguisher. The last attack strategy is essentially a higher order differential distinguisher. First, the compression layer of `Farfalle` produces an accumulator state equal to the *exclusive or* of non-linear permutations of the b-bit blocks representing the input data. This property allows the compression layer to satisfy the design requirements of being efficiently parallelizable and *incremental*.[1] However, it also allows an adversary to construct simple structures of 2^n n-block input values whose images by the compression layer form an affine subspace of dimension n of $\{0,1\}^b$.

Moreover, KRAVATTE relies on the `Keccak`-p permutation, whose round function has an algebraic degree only two and the rolling function is $\mathbb{F}_2$-linear. Therefore, if we denote by r the number of `Keccak`-p rounds of the partial computation —on input the accumulator state and up to ϵ final `Keccak`-p rounds— of any of the output blocks of the expansion layer, the algebraic degree of this partial expansion is upper bounded by 2^r. This implies that if $n > 2^r$, the sum of the outputs of this partial expansion over the accumulator values associated with one of the structures mentioned above is equal to zero. This observation provides the higher order differential distinguisher used in our attack.

Last Round Attacks. The attacks all rely on the capacity to "invert" up to two of the last rounds of the expansion layer despite a final masking of the output values by a key block. This can be done algebraically, by expressing the intermediate values as a function of the KRAVATTE output block and of the unknown key block, setting up a system of multivariate polynomial equations, and solving this system by linearization. Surprisingly, this is more efficient than expected from the algebraic degree of the inverse of the last rounds due to the limited diffusion in a small number of iterations of the inverse round function of `Keccak`.

This notably offers the possibility to leverage distinguishers on partial versions of KRAVATTE, and then mount key-recovery attacks on the full primitive. We give in Table 1 a list of the key-recovery attacks that are described in the rest of the paper.

Optimizations. Various technical improvements can be applied to the attack strategies in order to optimize the time, memory, or data complexity. We already note that some of these techniques improve some of the complexities but downgrade some others, which makes the selection of improvements a trade-off process. We discuss these optimizations in a dedicated section (Section 5) after the presentation of the attack strategies.

Organization. We give a description of KRAVATTE in Section 2, an instantiation of the `Farfalle` construction. In Section 3, we describe a MITM algebraic attack and an attack based on the linear recurrence distinguisher of KRAVATTE partial expansion layer. Both attack strategies focus on the expansion layer of KRAVATTE. In Section 4, we describe a higher order differential attack on KRAVATTE. In Section 5, we describe technical optimizations that can be applied to the attack strategies in order to improve their complexities, and provide a selection of attacks optimized either for time, memory or data complexity. Finally, we discuss in Section 6 the insights gained from these attacks.

2 Specifications of `Farfalle` and KRAVATTE

In this section, we give a description of permutation-based mode `Farfalle` and its original instantiation KRAVATTE, which is based on the permutation used in `Keccak` [BDPA11,

[1] *Incremental* means that if two input data share the same prefix, their compression layer computations can be partly shared.

Table 1: Key-recovery attacks against KRAVATTE instantiations for several (n_d, n_e) values. All attacks are independent of n_b and n_c, and $\star$ means that n_d can take any value. The reference points to the section describing the attack type. The complexity figures are obtained after the selection of optimizations described in Section 5.

$(\mathbf{n_d}, \mathbf{n_e})$	Type	Data	Memory	Time	Reference
		blocks	bits	elementary op.	
$(4, 4)$	Higher Order	$2^{74.7}$	$2^{62.3}$	$2^{112.2}$	Section 4
$(\star, 4)$	MITM	$2^{27.8}$	$2^{76.9}$	$2^{115.3}$	Section 3.1
$(\star, 4)$	Linear Recurrence	$2^{51.2}$	$2^{51.2}$	$2^{65.1}$	Section 3.2
$(\star, 4)$	Linear Recurrence	$2^{29.9}$	$2^{62.3}$	$2^{87.0}$	Section 3.2
$(\star, 6)$	Linear Recurrence	$2^{88.4}$	$2^{88.4}$	$2^{134.6}$	Section 3.2

NIS14]. The two primitives `Farfalle` and KRAVATTE have both been designed by Bertoni et al., originally published on the IACR ePrint in [BDH+16], and strengthened versions have been accepted at ToSC and will be presented at the FSE 2018 conference [BDH+17a].

2.1 The `Farfalle` Construction for Permutation-Based PRFs

KRAVATTE is a permutation-based variable input and output length pseudo-random function. It takes as input a key and a sequence of bit strings, and returns an arbitrary-length output. It relies on the `Farfalle` construction, which allows to build a PRF from parallel applications of fixed permutations. In this article and without loss of generality, we focus on input sequences that contain only one bit string, while the general construction allows for vectors of bit strings. Throughout this paper, the *exclusive or* of bits or b-bit blocks is denoted additively by "+".

`Farfalle` makes use of four permutations (possibly identical or related), denoted p_b, p_c, p_d and p_e of a b-bit block. Its instantiation requires the definition of three so-called *rolling functions*, denoted $roll_c$, $roll_e$ and $roll_f$. These functions should ensure that for an unknown value x, an adversary cannot predict the value of any number of iterations of the rolling functions $roll^i(x)$, nor the value of $roll^i(x) + roll^j(x)$ for $i \neq j$.

The `Farfalle` construction takes as input a key K and a message M. For an ℓ_i-block input message and a ℓ_o-block output, `Farfalle` consists of the three following steps:

Mask derivation: The key K is padded into a b-bit string $K\|10^*$, on which the permutation p_b is applied and yields $\mathbf{k}^{in} = p_b(K\|10^*)$. Denoting $\mathbf{k}^{out} = roll_c^{\ell_i+1}(\mathbf{k}^{in})$, $\ell_i+\ell_j$ masks are then computed as $\mathbf{k}_i^{in} = roll_c^i(\mathbf{k}^{in})$ for $i = 0, \ldots, \ell_i - 1$, and $\mathbf{k}_j^{out} = roll_f^j(\mathbf{k}^{out})$ for $j = 0, \ldots, \ell_o - 1$.

Compression layer: The message M is padded into a ℓ_i sequence of b-bit blocks m_i, by appending a 1-bit and a sequence of 0-bits. Then, one compresses these data into a single b-bit block accumulator x, by XOR-ing a key mask to each block, applying the permutation p_c to the results, and XOR-ing all the results together: $\texttt{Acc}(M) = \sum_i p_c\left(m_i + \mathbf{k}_i^{in}\right)$.

Expansion layer: In a first step, the permutation p_d is applied on the accumulator to get $\mathbf{y} = p_d(\texttt{Acc}(M))$. Then, in a second step, ℓ_o output blocks $\mathbf{z}^j$ are computed from this value by applying consecutively a rolling function, the permutation p_e, and an XOR with the key mask $\mathbf{k}_j^{out}$: namely, $\mathbf{z}^j = p_e\left(roll_e^j(\mathbf{y})\right) + \mathbf{k}_j^{out}$ for $j = 0, \ldots, \ell_o - 1$.

The output of `Farfalle` is the concatenation of bit strings $\mathbf{z}^0\| \cdots \|\mathbf{z}^{\ell_o-1}$.

2.2 The KRAVATTE Pseudo-Random Function

KRAVATTE is an instantiation of the `Farfalle` construction, which specifies the internal components. The Figure 1 outlines the overall primitive that relies on four different `Keccak-`p permutations [NIS14] on a block size of $b = 1600$ bits. The only distinction between those four permutations named p_b, p_c, p_d and p_e lies in their number of rounds, that we denote respectively by n_b, n_c, n_d and n_e.

Since the first publication, the designers substantially changed the construction of `Farfalle` and KRAVATTE, and as of the latest KRAVATTE specification [BDH+16], the permutations p_b and p_c consist of $n_b = n_c = 6$ rounds of the `Keccak-`p permutation, while the remaining two permutations contain $n_d = n_e = 4$ rounds. In a private communication,[2] upon discovery of the higher order differential attack described in Section 4, the designers considered increasing the numbers of rounds to $(n_b, n_c, n_d, n_e) = (6, 6, 6, 6)$. The resulting updated version has been presented by the designers at the ECC 2017 conference [BDH+17b].

To conveniently address the various versions throughout the paper, we use the notation KRAVATTE-(n_d, n_e) to refer to a version with specific n_d (resp. n_e) number of rounds in the permutation p_d (resp. p_e). Our results are independent of p_b and p_c, which is why we do not mention n_b and n_c.

Like in `Keccak`, the 1600-bit state is represented as a $5 \times 5 \times 64$ three-dimensional bit array B, where each bit is denoted $B_{x,y,z}$, with $x, y = 0, \ldots, 4$ and $z = 0, \ldots, 63$. Arithmetic operations performed on indices x, y and z are reduced modulo 5, 5 and 64, respectively, and we omit the modulo for the sake of simplicity.

Additionally, while `Farfalle` uses several rolling functions, KRAVATTE only relies on one, that we simply denote by $roll$, and whose n-th iteration is depicted as $\textcircled{n}$ on Figure 1. More precisely, $roll_b = roll_c = roll_d = roll$ and $roll_e$ is the identity. The $roll$ function transforms a state A into $B = roll(A)$ as follows:

$$\begin{aligned}
&B_{x,y,z} \leftarrow A_{x,y,z} && \text{if } y < 4,\\
&B_{x,4,z} \leftarrow A_{x+1,4,z} && \text{if } x < 4,\\
&B_{4,4,z} \leftarrow A_{0,4,z-7} + A_{1,4,z} && \text{if } z > 60,\\
&B_{4,4,z} \leftarrow A_{0,4,z-7} + A_{1,4,z} + A_{1,4,z+3} && \text{if } z \leq 60.
\end{aligned}$$

Security Claims. In the original document, the designers of KRAVATTE claim a security of 256 bits when the amount of data does not exceed 2^{137} input and output blocks, that is $\ell_i + \ell_o \leq 2^{137}$.

2.3 Round Function of the `Keccak-`p Permutation

We now give a brief description of the `Keccak-`p permutation, which can also be found in [NIS14]. It is based on the iteration of a round function, defined as the composition of the following operations (in this order), that each produce a state A' from A:

Linear Diffusion θ: The sum of the five bits of columns with indices $(x-1, z)$ and $(x+1, z-1)$ are added to bit each bit (x, y, z) of the state:

$$A'_{x,y,z} \leftarrow A_{x,y,z} + \sum_{j=0}^{4} A_{x-1,j,z} + \sum_{j=0}^{4} A_{x+1,j,z-1}.$$

Lane-Wise Rotation ρ: Each lane of the state is rotated by a different number of positions, whose exact values are not relevant for the description of the attacks.

[2] November 5, 2017.

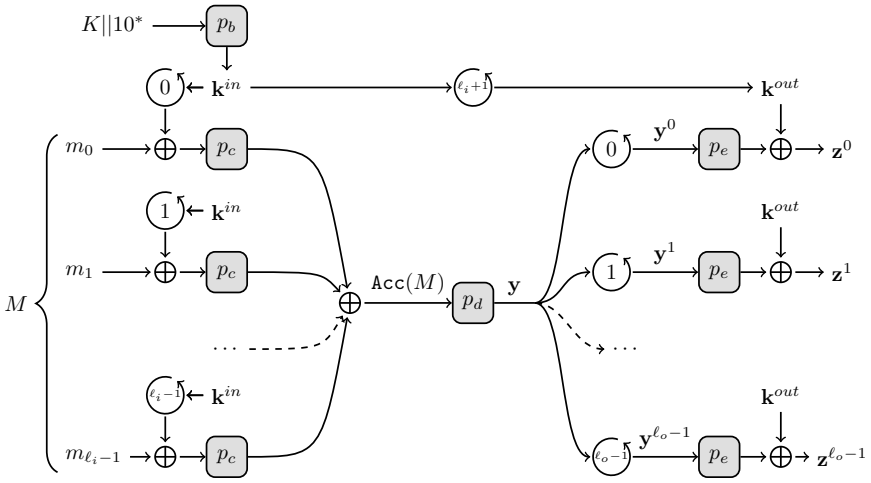

Figure 1: The KRAVATTE primitive. The input message M is padded and split into the b-bit blocks m_i. The function $\textcircled{n}$ refers to the linear function $x \to roll^n(x)$.

Lane-Preserving Permutation π: Lanes positions are switched according to a constant pattern: $A'_{x,y,z} \leftarrow A_{x+3y,x,z}$

Substitution Layer χ: A 5-bit Sbox of degree two is computed on each row (y, z) of the state. More specifically, each output bit depends on three input bits by the following equation (omitting y and z indices):

$$A'_x \leftarrow A_x + \overline{A_{x+1}} \cdot A_{x+2}.$$

Constant Addition ι: A round constant produced by an LFSR is XOR-ed to the lane indexed by $(0, 0)$. We omit the exact values of the constants, as they are not relevant to understand the paper. We refer the interested reader to [NIS14] for more details.

In the remaining of the paper, we also use the inverse of the Keccak-p round function, obtained by inverting the sequence of operations. The transformations ι, ρ and π all have straightforward inverses. The inverse Sbox χ^{-1} has algebraic degree three, and omitting y and z indices, its polynomial expression is given by

$$A'_x \leftarrow \overline{A_{x+1}} \cdot \overline{A_{x+3}} \cdot A_{x+4} + \overline{A_{x+1}} \cdot A_{x+2} + A_x.$$

The transformation θ^{-1} is a high-density linear layer whose exact expression is not relevant for the analysis conducted in the paper. It consists in XOR-ing to each bit of the state the sum of all five bits of about half of the columns of the state, and we note that, for a given column, the value XOR-ed to all the five positions is the same.

3 Algebraic Cryptanalysis of Full KRAVATTE

In this section, we describe key-recovery algebraic attacks against KRAVATTE-(n_d, n_e) for any n_d and $n_e \in \{4, 6\}$ and for a single message. These attacks rely on a remark on the linearization of the algebraic systems describing iterated Keccak-p^{-1} rounds, on the structure of the expansion layer in the KRAVATTE construction, and on the small number n_e of Keccak-p rounds in the p_e permutation. We note that these attacks are entirely independent of the compression layer of KRAVATTE as well as the application of p_d on the accumulator that initiates the expansion layer.

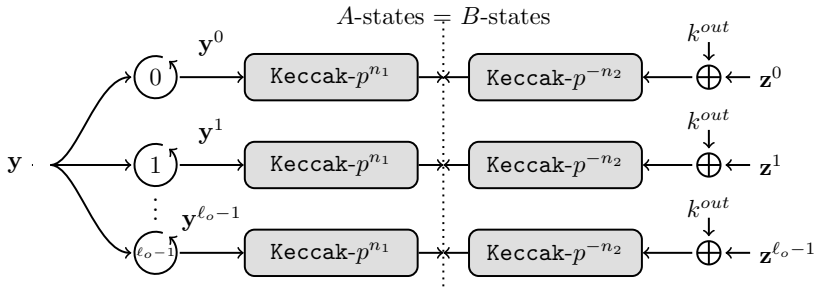

Figure 2: Meet-in-the-middle algebraic attack on KRAVATTE, with n_1 forward and n_2 backward rounds, $n_1 + n_2 = n_e$.

We first describe an attack based on a meet-in-the-middle strategy (Section 3.1), which can be enhanced by an observation borrowed from stream-cipher cryptanalysis (Section 3.2). This last technique can be further improved by refining the study of the linearization of iterated Keccak-p^{-1} rounds, which is covered in Section 5.

3.1 Meet-in-the-Middle Algebraic Attack

We present a key-recovery meet-in-the-middle (MITM) algebraic attack on full KRAVATTE. The key observation underlying the attack is that the *same* unknown value at the output of p_d is used at the input of *all* the branches in the expansion phase. If we denote this value by $\mathbf{y}$, then the j-th output block becomes $\mathbf{z}^j = p_e(roll^j(\mathbf{y})) + \mathbf{k}^{out}$. By considering a system of equations where the unknowns are bits of both $\mathbf{k}^{out}$ and of $\mathbf{y}$, we can mount a meet-in-the-middle attack by equating two states for Branch j: $A^j = B^j$, where A^j corresponds to n_1 forward rounds of Keccak-p applied on $\mathbf{y}^j = roll^j(\mathbf{y})$, and B^j to n_2 backward rounds of Keccak-p applied on $\mathbf{z}^j + \mathbf{k}^{out}$, with $n_1 + n_2 = n_e$. The A-states contain expressions in $\mathbf{y}$ and can be precomputed, while the B-states contain output-dependent expressions in $\mathbf{k}^{out}$. By considering a single input message (possibly unknown) together with its ℓ_o-block output, for ℓ_o sufficiently large, we can collect enough equations to form a system that can be solved through linearization, which recovers $\mathbf{k}^{out}$.

Linearization Principle. Linearization is a well-known technique to solve multivariate polynomial systems of equations. It relies on a fairly simple idea: the system of polynomial equations is turned into a system of linear equations by adding new variables that replace all the monomials of the system whose degree is strictly greater than 1. This linear system of equations can be solved using linear algebra if there are enough equations to make the linearized system overdetermined, typically at least on the same order as the number of variables after linearization. All the attacks from the paper heavily rely on this technique.

In the case of the MITM algebraic attack on KRAVATTE, the middle state can be described as a polynomial expression in $\mathbf{y}$ bits (resp. $\mathbf{k}^{out}$ bits) in the forward (resp. backward) direction. By linearization and summation of both expressions, one gets a linear equation in $\mathbf{y}$ and $\mathbf{k}^{out}$ monomials, with no composite monomial involving both types of unknowns.

Basic Linearization. The most straightforward way to linearize algebraic expressions in n unknowns of degree limited by d is to introduce a new variable for every monomial in the unknowns of degree at most d. The set of monomials considered has cardinality

$$S(n,d) \stackrel{\text{def}}{=} \sum_{i=1}^{d} \binom{n}{i}.$$

This approach can be used directly in the context of the MITM algebraic attack on

Table 2: Number of monomials in input (resp. output) variables after n rounds of Keccak-p or Keccak-p^{-1} for $b = 1600$ ($\log_2$ scale).

n	Keccak-p	Keccak-p^{-1}	
		Basic	Improved
1	20.3	29.3	13.0
2	38.0	77.3	36.5 [**Jean-René:** I get 36.3]
3	69.8	194.0	106.4

KRAVATTE. The algebraic degree of Keccak-p (resp. Keccak-p^{-1}) is two (resp. three) and *roll* is linear, thus the number of monomials involved by a basic linearization is approximately $S(b, 2^{n_1}) + S(b, 3^{n_2})$. We give in Table 2 the number of monomials required to describe the forward and backward parts of the meet-in-the-middle algebraic system.

Improved Linearization in the Backward Direction The number of monomials to consider in the backward direction can be drastically reduced if we take into account the row structure of χ^{-1} non-linear layers and the absence of diffusion before the first χ^{-1} layer encountered. Indeed, through the backward computations, new monomials are only created in χ^{-1} layers through multiplicative combination of input sum of monomials. There are two limiting factors to the combination power of the χ^{-1} layers. First, the χ^{-1} has only degree three, restricting the newly created monomials to the product of at most three input monomials. Secondly, it operates on only five inputs, which has a significant effect since input monomials for the external χ^{-1} layer can be limited by position: as no diffusion takes places after the unmasking by $\mathbf{k}^{out}$, only five $\mathbf{k}^{out}$ bit variables can occur in the monomials occurring at the output of a given Sbox. Consequently, the number of monomials required to express the outputs of an Sbox is upper bounded by $S(5, 3)$, so that expressing the outputs of all Sboxes only requires $N = \frac{b}{5}S(5, 3)$ monomials. The input bits of internal χ^{-1} layers have undergone linear diffusion, so they cannot be restricted in the same manner. However, the degree limitation still applies, and since N monomials can be used to describe the polynomial expressions of all bits before the χ^{-1} layer, the number of monomials that appear in the output bits of this layers is upper-bounded by $S(N, 3)$. This can be iterated to cover more rounds. Note that the improved linearization does not apply in the forward direction due to the application of *roll* and θ prior to the first χ layer. We give in Table 2 estimates for the number of monomials to consider for a small number of Keccak-p rounds in the backward direction.

For $n_e = 4$, choosing $n_1 = n_2 = 2$, KRAVATTE can be attacked by a meet-in-the-middle algebraic attack. The attack requires $\frac{1}{1600}(2^{38.0} + 2^{36.5}) \approx 2^{27.8}$ output blocks to get enough equations, has memory complexity $(2^{38.0} + 2^{36.5})^2 \approx 2^{76.9}$ bits to represent the system and the time for the resolution of the linearized system is at most cubic in the number of monomials, which yields about $(2^{38.0} + 2^{36.5})^3 \approx 2^{115.3}$ elementary operations.

The time complexity to build the system boils down to the construction of the $2^{27.8}$ equations, where each requires to compute the expressions coming from both sides of the meet-in-the-middle. For one equation, the backward contribution is dominated by the product of three polynomials in $320S(5, 3) = 8000$ monomials, while the forward contribution is dominated by the multiplication of two polynomials of at most $S(1600, 2) \approx 2^{20.29}$ monomials in $\mathbf{y}$. All in all, the time complexity is dominated by the time for solving the system.

The above observations about how to linearize the backward computation of up to two last rounds of Keccak-p and the results of Table 2 will be re-used in the key-recovery part

of the attacks introduced in Section 3.2 and Section 4.

3.2 Cancellation of Monomials Using a Linear Recurrence

We now describe a second attack that exploits the linearity of the rolling function to cancel the monomials in $\mathbf{y}$ from the system. Indeed, after the application of p_d, the first half of the remaining expansion layer can be seen as a filtered linear recurrent sequence of states, with update function *roll* and output function $\texttt{Keccak-}p^{n_1}$. Filtered linear recurrences, e.g. filtered LFSR, are classic stream cipher constructions, which have been deeply studied. A line of work [Key76, RH07, RGH07] observes that not only do the LFSR state bits follow linear recurrences, but the same holds for the monomials that are formed from these bits.

In this section, we start by exposing a recurrence polynomial of sequences of values taken by bits of the rolling state. We then show how this can be generalized to obtain a recurrence polynomial for sequence of values taken by products of bits of the rolling state. As the state A^j can be expressed as a sum of such products of $\mathbf{y}^j$ bits, this constitutes a linear complexity distinguisher on partial Kravatte, with the last n_2 $\texttt{Keccak-}p$ rounds and final masking removed. Finally, we show this can be used to combine equations of the system describing the expansion phase of Kravatte to eliminate the monomials in $\mathbf{y}$.

Linear Recurrence of Rolling State Bits. As stated above, the beginning of the expansion layers acts like a LFSR filtered by the fixed non-linear function $\texttt{Keccak-}p^{n_1}$. After the initial value $\mathbf{y} = p_d(\texttt{Acc}(M))$ of the rolling state is formed, it is updated linearly through the rolling function *roll*: the value of the rolling state that appears at the start of Branch j is given by $\mathbf{y}^j = roll^j(\mathbf{y})$. The rolling function *roll* is a linear transformation of the rolling state that leaves Planes 0 to 3 unchanged. The matrix $\mathbf{M}_{roll}$ of size 320 describing how *roll* affects Plane 4 has a primitive characteristic polynomial P_{roll} of degree 320.[3] By the Cayley-Hamilton theorem, we know that $P_{roll}(\mathbf{M}_{roll}) = 0$. We can associate to Bit i of Plane 4 a vector $\mathbf{e}_i$ of the standard basis of $\mathbb{F}_2^{320}$, and the values taken by the Bit i of Plane 4 of the state $\mathbf{y}^j$ in Branch j is then given by $\mathbf{e}_i^T \cdot \mathbf{M}_{roll}^j \cdot \mathbf{y}_{roll}$, where $\mathbf{y}_{roll}$ is the restriction of $\mathbf{y}$ to Plane 4. Then, we observe that the sequence of values taken by a given state bit of the part affected by *roll* over the branches of the expansion layer (see Figure 3) is a linear recurrence sequence with recurrence polynomial P_{roll}. Indeed, noting $P_{roll} = \sum_n c_n X^n$, we have for all j:

$$P_{roll}(\mathbf{y}_i^j) = \mathbf{e}_i^T \cdot \left(\sum_n c_n \mathbf{M}_{roll}^{j+n} \right) \cdot \mathbf{y}_{roll} = \mathbf{e}_i^T \cdot \left(\mathbf{M}_{roll}^j \cdot P_{roll}(\mathbf{M}_{roll}) \right) \cdot \mathbf{y}_{roll} = 0.$$

Furthermore, the bits in Planes 0 to 3 follow a linear recurrence with recurrence polynomial $X + 1$, so all the state bits follow the linear recurrence given by $(X + 1) \cdot P_{roll}$.

Linear Recurrence of Monomials Formed on the Rolling State. This can be generalized to the monomials formed from the bits of the rolling state. In the same way that $\mathbf{M}_{roll}$ describes the evolution of state bits, one can consider the matrix $\mathbf{M}_{roll}^{\leq d}$ describing the evolution through *roll* of monomials of degree at most d in the $\mathbf{y}^j$ state bits, since the transformation is also linear on this set. Indeed, the updated value of every state bit after *roll* is a linear combination of state bits before the update, so the product of d updated values can be written, by developing the product of the linear combinations, as a linear combination of monomials of state bits before update with degree at most d. The characteristic polynomial $P_{roll}^{\leq d}$ of this matrix provides a recurrence polynomial for all $S(320, d)$ monomials of degree at most d in the 320 variables of Plane 4. It is also a recurrence polynomial for monomials of degree at most d in all state variables with at least one variable coming from Plane 4, since the product of variables of Planes 0 to 3 is constant and can be factored out, leaving a monomial of degree strictly less than d of

[3] We explicitly give P_{roll} in Appendix.

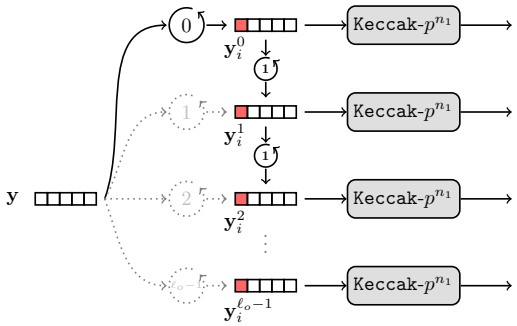

Figure 3: Linear recurrence in the KRAVATTE branches: the sequence $(\mathbf{y}_i^j)_j$ of highlighted bits at a prescribed Position i across the branches $j = 0, \ldots, \ell_o - 1$ follows a linear recurrence described by the polynomial $(X+1) \cdot P_{roll}$.

variables from Plane 4. Monomials with variables only from Planes 0 to 3, together with the constant monomial, are constant and have $X+1$ as recurrence polynomial. Thus, $(X+1) \cdot P_{roll}^{\leq d}$ is a recurrence polynomial for all monomials of degree at most d in all 1600 variables of the rolling state. Since `Keccak-`p has degree two, the recurrence polynomial $(X+1) \cdot P_{roll}^{\leq 2^{n_1}}$ cancels the sequences of all monomials involved in the algebraic expression of the outputs of the first part of the expansion layer. Its degree is $S(320, 2^{n_1}) + 1$. We give estimates of this value for $n_1 = 2, 3, 4$ in Table 3. For larger values of n_1, the technique is not applicable since the degree of the polynomial, e.g., $2^{146.5}$ for $n_1 = 5$, and $2^{227.3}$ for $n_1 = 6$ goes beyond the limit set on the data complexity in the security claims of KRAVATTE.

Computing the Recurrence Polynomial for Monomials of Degree at Most d. Computing the characteristic polynomial of a matrix usually requires to compute a determinant, but can be done in the case of $P_{roll}^{\leq d}$ in time quasilinear in the size of the matrix $\mathbf{M}_{roll}^{\leq d}$, without even forming the matrix, due to algebraic properties of linear recurring sequences. Indeed, it has been shown in [Key76] that the roots of this polynomial are all simple and elements of the algebraic extension $\mathbb{F}_2[X]/P_{roll}$. Denoting by α the class of X, they are given by α^t, were $t \in [1, 2^{320} - 1]$ takes all values with Hamming weight at most d. Thus, $P_{roll}^{\leq d}$ can be formed as the product of $N = S(320, 2^{n_1})$ polynomials of the form $X + \alpha^t$.

In a first stage, the polynomials whose roots are conjugates are multiplied together, resulting in a set of irreducible polynomials in $\mathbb{F}_2[X]$. In a second stage, these polynomials are multiplied together. Multiplication of two polynomials in $\mathbb{F}_2[X]$ of degree at most n can be performed efficiently for large n using the Schönhage algorithm [Sch77], with asymptotic complexity $M(n) = O(n \log n \log\log n)$. This algorithm has been implemented in the `gf2x` library [BGTZ08] and experimental data indicates the hidden constant is small. To take full advantage of fast polynomial multiplications, the computation of $P_{roll}^{\leq d}$ can be performed tree-wise: At each step, polynomials are multiplied by pairs, resulting in a set of half as many polynomials with double degree. Taking into account the asymptotic complexity of fast polynomial multiplication, the complexity T_P of the computation can be estimated by

$$\sum_{i=0}^{\log N} 2^i M\left(\frac{N}{2^i}\right) \leq N \log^2 N \log\log N.$$

We give estimates of this time complexity T_P for small values of n_1 in Table 3.

Impact on the Cryptanalysis of KRAVATTE. Using the linear recurrence given by the polynomial $(X+1) \cdot P_{roll}^{\leq 2^{n_1}} = \sum_j d_j X^j$, we eliminate all monomials in $\mathbf{y}$ from the system.

More precisely, the i-th equation is obtained by summing $A^{i+j} + B^{i+j} = 0$ equations:

$$\sum_j d_j \cdot \texttt{Keccak-}p^{n_1}\left(roll^{i+j}(\mathbf{y})\right) + \sum_j d_j \cdot \texttt{Keccak-}p^{-n_2}\left(\mathbf{k}^{out} + \mathbf{z}^{i+j}\right) = 0,$$

and since the first sum is null due to the relation from the linear recurrence, this yields

$$\sum_j d_j \cdot \texttt{Keccak-}p^{-n_2}(\mathbf{k}^{out} + \mathbf{z}^{i+j}) = 0. \tag{1}$$

As a consequence, in the case $n_2 = 2$, the number of monomials to write this system goes down to $2^{36.5}$, since only the monomials in $\mathbf{k}^{out}$ remain (see Table 2). Each equation of the system requires $S(320, 2^{n_1})$ consecutive output blocks to be formed, but since a sliding-window mechanism can be used to form the equations, only $S(320, 2^{n_1}) + \frac{1}{1600}2^{36.5}$ blocks are necessary to form the system. We can process the available blocks on the fly: For each block, we add its contribution to the system of $\mathbf{k}^{out}$ monomials in the equations prescribed by the recurrence polynomial, i.e., Block $\mathbf{z}_j$ contributes to Equation i if $d_{j-i} = 1$. This does not increase the memory complexity, but increases the time complexity of building the system by a factor of $S(320, 2^{n_1})$.

Applying this attack with $n_1 = 2$, we can attack Kravatte-$(n_d, 4)$ for any n_d. The recurrence polynomial to store has degree $2^{28.7}$, so to collect enough data to solve the linearized system, we need $2^{28.7} + \frac{1}{1600}2^{36.5} \approx 2^{28.8}$ output blocks. Computing the recurrence polynomial requires about $2^{40.7}$ basic operations[4] (see Table 3). Solving the linearized system requires $T_{solve} = (2^{36.5})^3 \approx 2^{109.5}$ operations, and a memory of $\left(2^{36.5}\right)^2 \approx 2^{73}$ bits. However, the most time-consuming part of the attack resides in the construction of the system. For each output block and every bit of the intermediate state, the bit is written as a linearized algebraic expression of bits in $\mathbf{k}^{out}$, depending on $\mathbf{z}^j$, as explained at the end of Section 3.1. Then, this expression is added to the equations it contributes to build, depending on the recurrence polynomial. Every equation is built by the addition of at most $S(320, 2^{n_1})$ contributions, and the cost of adding one contribution is given by the size of an expression, which is about $2^{36.5}$. Computing the algebraic expression of one bit essentially boils down to the multiplication of three polynomials in $320S(5, 3) = 8000$ monomials each that appear in the inversion of $n_2 = 2$ rounds of Kravatte. Overall, constructing the system amounts to approximately

$$T_{build} = \left(S(320, 2^{n_1}) + \frac{1}{1600}2^{36.5}\right) \cdot 1600 \cdot 8000^3 + S(320, 2^{n_1}) \cdot \left(2^{36.5}\right)^2$$

operations. We give estimations of the overall time complexity $T_P + T_{build} + T_{solve}$ in Table 3, and stress again that there are several options to optimize this step, which are addressed in Section 5.

With $n_1 = 4$, the attack breaks the security claim of Kravatte-$(n_d, 6)$ for any n_d. Indeed, the recurrence polynomial has degree $2^{88.4}$ and can be computed in about 2^{104} simple operations, which allows to collect the equations using about $2^{88.4}$ output blocks. Then, the system can be constructed as before (with non-optimized computations, it requires about $2^{161.4}$ basic operations) and solved similarly as in the case $n_1 = 2$.

We summarize the attack complexities in Table 3. Additional optimizations that further lower these complexities are presented in Section 5. In this table, the two first lines provide attacks for the ePrint version of full Kravatte-$(4, 4)$, and the last line gives an attack for the strengthened variant announced at ECC 2017.

[4] We performed this computation and provide the full expression of $P_{roll}^{\leq 4}$ in the `verification_LinearRecurrence` folder of the supplementary material. The code located in the same folder provides an experimental verification of the linear recurrence distinguisher after $n_1 = 2$ rounds.

The Particular Case of One Backward Round. In the case of one backward round, i.e., $n_2 = 1$, Equation (1) can be solved by exhaustive search. Note that the linear layer of the round considered can be removed from the analysis, and thus no diffusion takes place. As a consequence, it is possible to recover $\mathbf{k}^{out}$ Sbox by Sbox, by guessing the five $\mathbf{k}^{out}$ bits corresponding to a given Sbox, and checking that sums of $\chi^{-1}(\mathbf{k}^{out} + \mathbf{z}^j)$ over positions j determined by the recurrence polynomial yields zero, which is the case for the correct guess. For each sum, this gives a $t = 5$-bit test on the $g = 5$ guessed bits of $\mathbf{k}^{out}$. With only one sum, the probability that no false alarm occurs is $(1 - 2^{-t})^{2^g - 1} \approx 0.37$, so the rate of Sboxes with false alarms on corresponding $\mathbf{k}^{out}$ bits is too high to recover the complete $\mathbf{k}^{out}$ by key enumeration. However, with two sums, one gets a $t = 10$-bit test, and the probability of absence of false alarms raises to 0.97, which amounts to about 10 Sboxes with false alarms, making a final offline key candidate enumeration possible.

With $(n_1, n_2) = (3, 1)$, it is thus possible to attack KRAVATTE-$(n_d, 4)$ for any n_d with a data complexity of $2^{51.2}$ blocks and a time complexity dominated by the recurrence polynomial computation time $T_p = 2^{65.1}$. The attack requires to precompute and store $P_{roll}^{\leq 3}$ and thus has memory complexity $2^{51.2}$.

Table 3: Degree and computation time of recurrence polynomial for all monomials in $\mathbf{y}$ after n_1 rounds of `Keccak`-p, and attack complexity against KRAVATTE-(n_d, n_e), for any n_d and $n_e = n_1 + n_2$. For optimized attacks, see Section 5.

$\mathbf{n_e}$	$\mathbf{n_1 + n_2}$	$\deg\left(\mathbf{P}_{\mathbf{roll}}^{\leq \mathbf{d}}\right)$	$\mathbf{T_P}$	**Data***	**Memory***	**Time***
4	$2+2$	$2^{28.7}$	$2^{40.7}$	$2^{29.3}$	$2^{73.0}$	$2^{109.5}$
4	$3+1$	$2^{51.2}$	$2^{65.1}$	$2^{51.2}$	$2^{51.2}$	$2^{65.1}$
6	$4+2$	$2^{88.4}$	$2^{104.0}$	$2^{88.4}$	$2^{88.4}$	$2^{161.4}$

*: These complexities are given without the optimizations addressed in Section 5.

4 Higher Order Differential Cryptanalysis of Full KRAVATTE

In this section, we highlight the existence of higher order differential attacks against KRAVATTE. We describe in Section 4.1 a property of the compression layer of `Farfalle`, which weakens the overall construction against higher order differential attacks. The process of our attack is depicted in Section 4.2. To experimentally validate the correctness of the approach, we use a round-reduced variant of KRAVATTE.

In Section 4.3, we show how an adversary can use the higher order distinguisher to mount a chosen-message key-recovery attack against KRAVATTE-(n_d, n_e) such that $n_d + n_e \leq 8$. In the last section of the paper dedicated to various optimizations, we present a variant of this attack allowing to improve the overall data complexity (Section 5.2) and various techniques to substantially decrease the complexities.

4.1 Construction of Affine Spaces in the Accumulator

We describe here a property of the compression layer of `Farfalle`, already identified in [BDH+16, Section 5.4], that enables an adversary to construct an affine space of dimension n in the accumulator block. Given an n-block padded message $M = (m_0, \ldots, m_{n-1})$, we recall that we denote $\texttt{Acc}(M)$ the associated accumulator value $\sum_{i=0}^{n-1} p_c\left(m_i + \mathbf{k}_i^{in}\right)$.

Let $M^0 = (m_0^0, \ldots, m_{n-1}^0)$ and $M^1 = (m_0^1, \ldots, m_{n-1}^1)$ denote an arbitrary pair of padded messages such that $m_i^0 \neq m_i^1$ for all i. These messages are used to build the following structure of 2^n n-block input messages: $\mathcal{S} = \{(m_0^{\epsilon_0}, \ldots, m_{n-1}^{\epsilon_{n-1}}), (\epsilon_0, \ldots, \epsilon_{n-1}) \in \{0,1\}^n\}$.

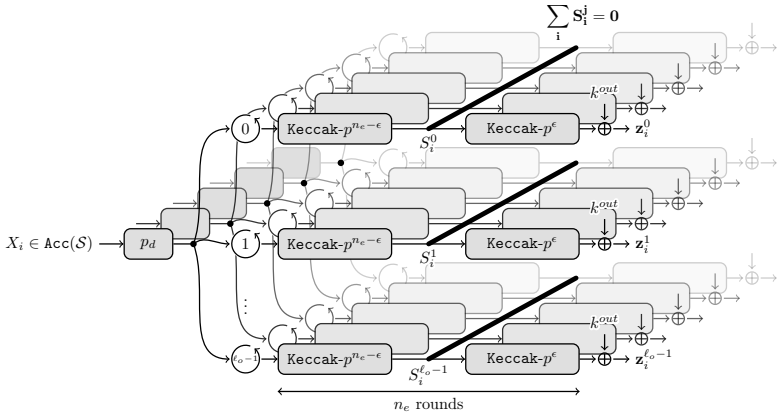

Figure 4: Higher order differential distinguisher on KRAVATTE. Summing over the whole affine space $\mathtt{Acc}(\mathcal{S})$ the states obtained after application of $\ell = n_d + n_e - \epsilon$ rounds to the blocks X_i of the affine space, i.e., summing along every bold line, yields zero.

Let us denote by δ_i the one-block difference $\delta_i = p_c\left(m_i^0 + \mathbf{k}_i^{in}\right) + p_c\left(m_i^1 + \mathbf{k}_i^{in}\right)$. If $n \ll b = 1600$, the δ_i are linearly independent with overwhelming probability. It is easy to see that $\mathtt{Acc}(\mathcal{S})$ is then the n-dimensional affine subspace $\mathtt{Acc}(M_0) + \langle \delta_0, \ldots, \delta_{n-1} \rangle$.

In other words, we can easily build structures of 2^n n-block messages that are transformed by the compression layer into an affine space of one-block accumulator values of dimension n. Note that this does not depend on the number of rounds in p_c.

4.2 Higher Order Differential Attacks Against KRAVATTE

We can use the property of `Farfalle` described above to mount higher order differential attacks on KRAVATTE-(n_d, n_e), as long as $n_d + n_e \leq 8$.

Summing the images of a function f over an affine subspace of dimension n is equivalent to applying the n-th differential of f to an element of the subspace [Lai94]. The round function of the `Keccak-`p permutation used in the KRAVATTE instance is of algebraic degree two. Hence, the partial expansion layer, starting from the accumulator value and applying ℓ permutation layers, is of degree 2^ℓ. By building an affine space of dimension $n = 2^\ell + 1$ with an input structure $\mathcal{S}$ of 2^n messages, each one containing n blocks, the sum over this affine space of the intermediate values after the partial expansion is zero (see Figure 4).

This distinguishing property can then be used to mount last-round attacks: Starting from the KRAVATTE output values of the plaintext in the structure, by inverting the last $\epsilon = n_d - n_e - \ell$ permutation layers of the expansion layer, and summing all the contributions, one gets equations on the output key $\mathbf{k}^{out}$: namely, $\sum_{m \in \mathcal{S}} \mathtt{Keccak}\text{-}p^{-\epsilon}(\mathbf{k}^{out} + \mathbf{z}^m) = 0$.

To demonstrate the validity of the higher order differential distinguisher described above, we applied it on SHORTKRAVATTE. In SHORTKRAVATTE, $n_d = 0$ so that the expansion layer consists of four rounds of the `Keccak-`p permutation, instead of $n_d + n_e = 8$ rounds for the full KRAVATTE instance. Hence, with a structure of 2^{16+1} input messages of 17 blocks, the higher order differential distinguisher spans the whole expansion layer and can be observed by summing directly the output values of the messages in the structure.[5]

4.3 Last-Round Attacks

One Last-Round Attack. One can apply the higher order differential strategy to mount a basic key-recovery attack against KRAVATTE-(n_d, n_e), $n_d + n_e \leq 8$, by considering a 7-round partial expansion layer and a final last-round (i.e., $\epsilon = 1$). This implies to use structures of

[5] The C++ source code that demonstrates this distinguisher appear in the `verification_HigherOrder` folder of the supplementary material.

$2^{2^7+1} = 2^{129}$ messages of 129 blocks. The higher order differential distinguisher continues to apply through the linear layer of the last round. Thus, no diffusion takes place in the inverted part of the last round, and the key-recovery method given at the end of Section 3.2 for the case of one backward round applies, with the small variation that one gets $t = 10$-bit tests by requesting two output blocks per message. This attack has a time and data complexity of $2^{129} \times (129 + 2) \approx 2^{136.0}$, and negligible memory complexity.

We have implemented this attack on a reduced version of KRAVATTE where $n_d + n_e = 5$ using structures of 2^{17} messages of 17 blocks, and find that the number of candidates for $\mathbf{k}^{out}$ is reduced from 2^{256} to about 2^{18}. We note that $\mathbf{k}^{out}$ can be uniquely determined using three to six output blocks.[6]

Two Last-Round Attack. We now describe how to improve the time and data complexity, leveraging the analysis of the algebraic expression of Keccak-$p^{-2}(\mathbf{k}^{out} + \mathbf{z})$, in the same way as for the previous attack (Section 3.1). This enables to consider a higher order distinguisher over $\ell = 6$ Keccak-p rounds (i.e., $\epsilon = 2$). The adversary builds a structure $\mathcal{S}$ of 2^{65} plaintexts of 65 blocks as described above. As a consequence, the 2^{65} intermediate values at any bit position before the penultimate non-linear layer sum to zero.

As described in the previous section (see Table 2), the number of monomials involved in the system corresponding to the inversion of the two last rounds of KRAVATTE is about $2^{36.5}$, and the system can be solved if one collects about the same number of equations. These can be obtained considering, for every message of the structure, outputs of $\frac{1}{1600}2^{36.5} \approx 2^{25.9}$ blocks. The total data complexity (expressed as the sum of all input and output blocks) of this attack is therefore $2^{65} \times (65 + 2^{25.9}) \approx 2^{90.9}$ blocks. The system of equations can be computed on the fly, therefore there is no need to store all the inputs and outputs of KRAVATTE. However, storing the system requires $(2^{36.5})^2 = 2^{73.0}$ bits of memory. The evaluation of the time complexity is more involved, as one needs to consider all the steps of the attack. For each equation and each input, the most expensive step consists in computing the penultimate χ^{-1} layer, which requires to get the product of three polynomial expressions, each of which can contain up to 8000 monomials. Therefore, we can estimate the complexity of this step of the attack to $2^{36.5} \times 2^{65} \times 8000^3 \approx 2^{140.4}$ bit operations. Solving the system of equations is far less expensive, as its time complexity is at most cubic in the number of equations, which leads to $(2^{36.5})^3 = 2^{109.5}$ operations.

As we will show in Section 5, these "naive" data and time complexities can be substantially improved using various optimizations.

5 Optimization Techniques for the Cryptanalysis

5.1 Minimizing the Number of Variables for Two Inverse Rounds

In this section, we improve further the linearization of Keccak-p^{-2}. Notations used in the following are summarized on Figure 5.

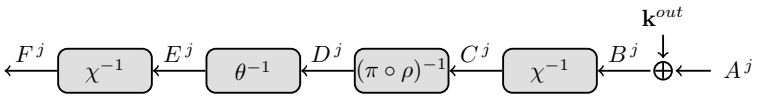

Figure 5: Notations used in Section 5.1.

The attacks described in Section 3 and Section 4 are all based on the construction and the linearization of polynomial expressions whose variables are the bits of $\mathbf{k}^{out}$. Each bit of F^j can be expressed as a low-degree polynomial in key bits $\mathbf{k}^{out}$, whose coefficients are

[6] The source code for this 1-round attack is also provided in the `verification_HigherOrder` folder of the supplementary material.

functions of the output states A^j. We have seen in Section 3.1 that using the row structure of χ^{-1} layers and the absence of diffusion before the first χ^{-1} layer enables to decrease the number of variables to be considered in the linearized expressions. We now show that this can be improved further by additionally considering the number of monomials in the polynomial expression of χ^{-1}, by limiting the number of considered bit positions of F^j and by summing two positions of F^j. We note that this improvement comes at the cost of a slight increase of the data complexity, since only one equation is extracted from each output block. We also note that this last optimization is compatible with the attacks presented in Section 3 and Section 4 because the sum over sets of output block messages they consider are performed consistently on all the positions of the blocks. We study each of the successive layers of the backward computation of KRAVATTE.

External χ^{-1} Layer. The inverse Sbox has algebraic degree three. More precisely, we get the following expressions (we omit indexes y and z and the block number j):

$$\begin{aligned} C_x &= \overline{B_{x+1}B_{x+3}}B_{x+4} + \overline{B_{x+1}}B_{x+2} + B_x \\ &= \left(\mathbf{k}^{out}_{x+1} + A_{x+1} + 1\right)\left(\mathbf{k}^{out}_{x+3} + A_{x+3} + 1\right)\left(\mathbf{k}^{out}_{x+4} + A_{x+4}\right) \\ &+ \left(\mathbf{k}^{out}_{x+1} + A_{x+1} + 1\right)\left(\mathbf{k}^{out}_{x+2} + A_{x+2}\right) + \left(\mathbf{k}^{out}_{x} + A_{x}\right). \end{aligned}$$

Introducing the new variables $w_x = \mathbf{k}^{out}_{x+1}\mathbf{k}^{out}_{x+3}\mathbf{k}^{out}_{x+4} + \mathbf{k}^{out}_{x+1}\mathbf{k}^{out}_{x+2} + \mathbf{k}^{out}_{x}$, $u_x = \mathbf{k}^{out}_{x+3}\mathbf{k}^{out}_{x+4} + \mathbf{k}^{out}_{x+2}$, and $v_x = \mathbf{k}^{out}_{x}\mathbf{k}^{out}_{x+2}$, this can be rewritten as $w_x + P_x(A)$, where $P_x(A)$ is an affine combination of u_x, v_{x+1}, v_{x+4}, $\mathbf{k}^{out}_{x+1}$, $\mathbf{k}^{out}_{x+3}$, $\mathbf{k}^{out}_{x+4}$, with coefficients determined by A. The definition of new variables for the sum of monomials sharing the same coefficient instead of for each monomial enables us to limit the number of variables per bit of the C state to 8, including the variable w with constant coefficient and the coefficient of the degree-0 monomial. The total number of variables over the state is however the same with both approaches. For each of the 320 Sboxes, one generates 10 variables u, v of algebraic degree two in key bits and 5 variables w of algebraic degree three. Taking into account the key bits, the total number of variables is therefore $320 \times (5 + 10 + 5) = 6400$ at that point. This already improves on Section 3.1 since all degree-3 variables are not considered anymore.

Intermediate Inverse Affine Layer. The linear layers consist in the bit-moving layers ρ and π and the linear diffusion layer θ. All these layers do not create new monomials: monomials are simply moved or added to other polynomial expressions. The linear layers only contribute indirectly to the complexity of the algebraic expressions by breaking any independent subset, leading to consider that any bit of the state can be affected by monomials in variables coming from any position of the final state A. This is especially true for the high-diffusion transformation θ^{-1}, whose output bits depend on approximately half of its input state. This has the effect to allow the creation during the next non-linear layer of nearly all the combinations of monomials output by the previous non-linear layer.

More precisely, ρ^{-1} and π^{-1} move every bit of the state. We denote σ the permutation such that bit $\sigma(x, y, z)$ of the state is moved to position (x, y, z). Then, θ^{-1} is a linear diffusion layer with the following property. For each column $D_{x,z}$ of the state, there is a set of bit positions $S_{x,z}$ such that each bit after θ^{-1} is given by

$$E_{x,y,z} = D_{x,y,z} + \sum_{(x',y',z') \in S_{x,z}} D_{x',y',z'}.$$

Therefore, we have

$$\begin{aligned} E_{x,y,z} &= C_{\sigma(x,y,z)} + \sum_{(x',y',z')\in S_{x,z}} C_{\sigma(x',y',z')} \\ &= w_{\sigma(x,y,z)} + P_{\sigma(x,y,z)}(A) + \sum_{(x',y',z')\in S_{x,z}} \left(w_{\sigma(x',y',z')} + P_{\sigma(x',y',z')}(A)\right) \\ &= w'_{x,y,z} + P_{\sigma(x,y,z)}(A) + Q_{x,z}(A). \end{aligned}$$

In this expression, $w'_{x,y,z}$ is a new variable defined as the linear combination of all the w variables involved in the expression of $E_{x,y,z}$, and $Q_{x,z}(A)$ is the sum of the P's over position set $S_{x,z}$. Please note that each $Q_{x,z}$ is considered to potentially involve all the u, v and $\mathbf{k}^{out}$ variables, whereas $P_{x,y,z}$ only has 7 potentially nonzero variables. Also, w variables influence of E and F is completely given by w' variables, which are independent of A. Therefore, w' variables can replace w variables in the description of the algebraic expressions of all F^j.

Partial Internal χ^{-1} Layer. We now consider only two bits of information from the state F^j in a same column, e.g., at Positions $(0,0,0)$ and $(0,1,0)$, and sum their algebraic expressions. With this approach, we cancel out the multiplication of the term contributing the most monomials to the expressions of these bits, decrease the total number of variables and therefore limit the time and memory complexity of our attack by reducing the complexity of the final linearized system. To this end, we denote $P'_{x,y,z} = w'_{x,y,z} + P_{\sigma(x,y,z)}$. Omitting index z, we have:

$$F_{0,y} = \left(\overline{P'_{1,y}} + Q_1\right)\left(\overline{P'_{3,y}} + Q_3\right)\left(P'_{4,y} + Q_4\right) + \left(\overline{P'_{1,y}} + Q_1\right)\left(P'_{2,y} + Q_2\right) + \left(P'_{0,y} + Q_0\right).$$

When considering $F_{0,0} + F_{0,1}$, all products of Q-components cancel out, as Q polynomials are identical over a column. In particular, all arbitrary products of three u, v and $\mathbf{k}^{out}$ variables do not occur anymore. We get:

$$\begin{aligned} F_{0,0} + F_{0,1} = {} & \left(P'_{1,0} + P'_{1,1}\right) Q_3 Q_4 + \left(P'_{3,0} + P'_{3,1}\right) Q_1 Q_4 + \left(P'_{4,0} + P'_{4,1}\right) Q_1 Q_3 \\ & + \left(\overline{P'_{1,0}P'_{3,0}} + \overline{P'_{1,1}P'_{3,1}}\right) Q_4 + \left(\overline{P'_{1,0}}P'_{4,0} + \overline{P'_{1,1}}P'_{4,1}\right) Q_3 \\ & + \left(P'_{2,0} + \overline{P'_{3,0}}P'_{4,0} + P'_{2,1} + \overline{P'_{3,1}}P'_{4,1}\right) Q_1 + \left(P'_{1,0} + P'_{1,1}\right) Q_2 \\ & + \left(P'_{0,0} + \overline{P'_{1,0}}P'_{2,0} + \overline{P'_{1,0}P'_{3,0}}P'_{4,0} + P'_{0,1} + \overline{P'_{1,1}}P'_{2,1} + \overline{P'_{1,1}P'_{3,1}}P'_{4,1}\right). \end{aligned}$$

All the Q polynomials are affine combinations of the same set of all the $320 \times (5+10) = 4800$ $\mathbf{k}^{out}$, u and v variables, and each P' polynomial is an affine combination of 7 variables. Taking into account the constant coefficients of these polynomials, the number of variables required to linearize the expression of $F_{0,0} + F_{0,1}$ is therefore:

$$3 \times 2 \times 8 \times \binom{4801}{2} + \left(3 \times 2 \times 8^2 + 2 \times 2 \times 8\right) \times 4801 + 2 \times \left(8 + 8^2 + 8^3\right),$$

which gives approximately $2^{29.0}$ variables, instead of the approximately $2^{36.5}$ monomials obtained with the simpler bound from Section 3.

Note that there is a trade-off between the number of variables of the linearized system and the number of equations that are obtained from on block. Indeed, by considering more than one pair of positions, additional w' variables have to be considered, together with the monomials resulting from the products of these variables with $\mathbf{k}^{out}$, u and v variables. We do not investigate further this trade-off, since we are mainly concerned with the reduction of the size of the system in order to improve the attacks time complexity, and because this reduction of the system size limits the degradation of the data complexity.

5.2 Super Structure of Input Messages

As already presented in Section 4, the higher order differential distinguisher used for the attack is based on the sum of output messages over a structure of input messages of dimension $n = 65$. Due to the property of `Farfalle` and the algebraic degree of the `Keccak`-p round function, this is guaranteed to lead to a sum of corresponding intermediate states equal to zero. In order to improve the data complexity, we use a super structure of messages, from which structures of dimension 2^n can be extracted. This technique has been used previously, e.g, in [DLMW15].

Principle. Let us consider the set of messages obtained by the concatenation of $n + t$ blocks, Block j being chosen among two possibilities $(m_j^i)_{i\in\{0,1\}}$:

$$\mathcal{S} = \left\{(m_0^{\epsilon_0}, \cdots, m_{n+t-1}^{\epsilon_{n+t-1}}), (\epsilon_0, \cdots, \epsilon_{n+t-1}) \in \{0,1\}^{n+t}\right\}.$$

We can then extract from this super structure several n-dimensional structures by fixing the values of the blocks at t given positions, and subsequently build from each of these structures equations in the output key bits $\mathbf{k}^{out}$.

When extracting n-dimensional structures from an $n + t$ super structure, care has to be taken to avoid linear dependencies that decrease the expected amount of equations that can be formed. Indeed, let us consider $\mathcal{S}_*$, an $(n + 1)$-dimensional super structure, and the n-dimensional structures $\mathcal{S}_{0*} = m^0 * \ldots *$, $\mathcal{S}_{1*} = m^1 * \ldots *$, $\mathcal{S}_{*0} = * \ldots * m^0$ and $\mathcal{S}_{*1} = * \ldots * m^1$, where $*$ denotes consecutive positions with two possible values at each position, and 0 (resp. 1) denotes a position with fixed m^0 (resp. m^1), value. Then, we have $\mathcal{S}_{0*} \cup \mathcal{S}_{1*} = \mathcal{S}_* = \mathcal{S}_{*0} \cup \mathcal{S}_{*1}$. As a consequence, given the sum of the intermediate states over three of these five structures, we can derive linearly the sum over the remaining structures, and thus they does not provide additional equations to include into the system.

We can obtain $\binom{n+t}{t}$ structures by selecting m^0 blocks at exactly t positions of the super structure, and keeping the choice among two values on the other n positions. These structures lead to linearly independent equations since the message containing n blocks equal to m^1 at given positions only appear in one specific structure.

Increasing the Number of Structures Extracted from a Super Structure. More generally, we can obtain $S(n+t, t)$ structures by selecting m^0 blocks at any given $0 \leq i \leq t$ positions of the super structure, and keeping the choice among two values on the other n positions. Indeed, let us associate to every structure an integer, whose binary representation represents the indeterminate positions of the structure. Let us also associate to every message an integer whose binary representation is the selection pattern of m^0 and m^1 blocks. Ordering the structures by decreasing value of their representative and the messages by decreasing values of their associated integers, the binary structure/message membership matrix $(\delta_{M_j\in\mathcal{S}_i})_{i,j}$ is in row echelon form (see Figure 6 in Appendix), which proves the linear independence of equations generated from these structures.

Finally, we remark that the $(n + t - i)$-dimensional structures generated above can be generated linearly from the n-dimensional structures where the fixed message blocks are selected from $\{m^0, m^1\}$. It it thus possible to select $S(n + t, t)$ such n-dimensional structures leading to independent equations.

Computing Equations from Super Structures. There are at least two possible strategies to build the equation system using super structures. First, we can store all the data blocks that we get, and compute the system equation by equation, recomputing the contribution from each block of a given structure. Otherwise, we can handle the data block per block, building all the system of equations at once.

The first strategy requires to store all the data blocks during the computation of the system, whereas the second approach has no specific memory requirements and potentially allows to spare computation time. Nevertheless, extra memory can be required for the

construction of the system of equations itself. This will be studied in Section 5.3. In the following, we assume that we select the second approach, and compute all equations simultaneously, handling the available data block by block.

Complexity Analysis. The major benefit of using super structures of input messages is to reduce the data complexity of the attack. By considering super structures of size $n+t$ and ℓ_o output blocks per message, the data complexity is $(n+t+\ell_o)2^{n+t}$ blocks. The number of equations we get is $b\ell_o S(n+t,t)$, assuming one does not use the optimization from Section 5.1 and computes b equations per structure. Otherwise, only one equation is recovered per structure, and the total number of equations is $\ell_o S(n+t,t)$.

We select the parameters of our attack as follows. One aims at getting N_{eq} equations, therefore we need that $b\ell_o S(n+t,t) \geq N_{eq}$ (resp. $\ell_o S(n+t,t) \geq N_{eq}$) if we do not reduce (resp. we reduce) the number of equations. Thus, we choose $\ell_o = \lceil N_{eq}/(S(N+t,t)b) \rceil$ (resp. $\ell_o = \lceil N_{eq}/S(N+t,t) \rceil$. As our aim is to reduce the amount of data necessary for the attack, we then need to choose t so as to minimize $(n+t+\ell_o)2^{n+t}$.

If we combine these super structures with the reduction of the number of equations, we need $N_{eq} \approx 2^{29.0}$. This leads to $t=5$, $\ell_o = 4$ and a data complexity of $2^{74.7}$ blocks. If we use only super structures, we need $N_{eq} \approx 2^{36.5}$ equations. We then compute $t=4$, $\ell_o = 5$ and reach a data complexity of $2^{73.9}$ blocks.

5.3 Counters

We specify here two algorithms that the adversary might use to build a system of equations from the outputs, and study their time and memory complexities, in order to determine the optimal choice in each version of our attack.

Description of the Systems of Equations. The attacks we want to optimize are either based on the higher order differential property or on the use of a polynomial derived from a stream-oriented description of the expansion layer of KRAVATTE. Each equation of the system is derived from a linear relation on bit values two rounds before the output. It is obtained by summing the contributions of several output blocks. The expression of such a contribution requires the computation of the polynomial expression of one bit (or one linear combination of bits) two rounds before the end of KRAVATTE, considering key bits as variables.

In the following, we consider that the bottleneck of such an operation consists in the multiplications of three polynomials that stems from the degree-3 term of the internal χ^{-1} layer. The complexity of such an operation is estimated as the product of the number of terms of each of the three polynomials.

Definitions and Notations. We call *equation* a relation involving key bits and newly introduced key-dependent variables. The equations we use are computed by summing polynomials that depend on one output block. We use the generic term *expression* to refer to such a polynomial, which is computed by inverting two rounds of KRAVATTE. The addition of a given expression when building an equation is called a *contribution*.

We use N_{eq} to denote the total number of equations that is needed to solve the system, which is supposed to be equal to the number of variables. Therefore, we have $N_{eq} = 2^{29.0}$ if optimization of Section 5.1 is implemented, and $2^{36.5}$ otherwise. We denote by S the number of contributions that one needs to add to get each equation. This number depends on the kind of distinguisher that is used. For the 6-round higher order differential distinguisher, we have $S = 2^{65}$. If we use the linear recurrence distinguisher, we need to sum expressions over all the block positions given by the nonzero coefficients of the recurrence polynomial. We estimate it as half the degree of the polynomial (see Table 3). Conversely, we call R the average number of equations each computed expression contributes to. When handling a specific output block, one computes the polynomial expression of bits two rounds before the output, then add its contribution to the R equations (on average) it is involved in.

We also denote by N_{par} the maximum number of equations that are being computed at the same time during the process. When using the linear recurrence distinguisher or the higher order differential distinguisher with super structures, we compute all equations in parallel, and $N_{par} = N_{eq}$, whereas for the higher order differential distinguisher without super structures, we build 1 or 1600 equations in parallel, depending on the number of bits of information we use per block. We also denote by N_{prod} the number of products of three monomials that one needs to compute when inverting the internal χ^{-1} layer. In the general case, N_{prod} is dominated by the product of three polynomials in 8000 variables, which can be considered to cost $8001^3 \approx 2^{38.9}$ elementary operations. If we apply optimizations of Section 5.1, the most expensive part consists in 6 multiplications of 3 polynomials: 1 with 8 nonzero coefficients and 2 with 4801 nonzero coefficients. We then have $N_{prod} = 6 \times 8 \times 4801^2 \approx 2^{30.0}$ elementary operations.

A Direct Approach. The most straightforward way to proceed is to run through all the output blocks that are needed to get enough equations: For each of them, we compute each expression that is needed to build the system, and add it to all equations it contributes to.

Using this technique does not require a specific amount of memory, other than the one used to store the linearized system. We assume that all expressions originating from a single block of output are computed independently. The time complexity to compute each expression in key bits that contributes to the system of equations is N_{prod} operations. The total number of contributions is $N_{eq} \times S$, and therefore the total number of expressions one needs to compute is $N_{eq} \times S/R$. After computing these expressions, one needs to add them to the equations they contribute to. The total number such additions is $N_{eq} \times S$. As the number of equations equals the number of variables, such an addition costs N_{eq} elementary operations. Therefore, the time complexity T_{direct} of this technique is given by

$$T_{direct} = N_{eq} \times S \times \Big(N_{prod}/R + N_{eq}\Big).$$

An Optimization Based on the Use of Parity Counters. Our second technique relies on the following observation. Each of the polynomials that are multiplied during the internal χ^{-1} layer are linear combinations of output bits of the external χ^{-1} layers. Such a bit only depends on the five bits of the output block corresponding to one Sbox output. Moreover, when expanding the products of the internal χ^{-1} layer, one gets the sum of products of three such bits, which only depends on 15 bits of the output blocks, corresponding to the outputs of three Sboxes.

For each equation, the computation of all the S contributions leads to computing several times these products, as soon as S exceeds 2^{15}. Our idea is then to compute these products only once for each value V of the 15 output bits. This can be done in a precomputation step. Then, the contribution of each of these results is added to the final equation if and only if the number of occurrences of V is odd (as the sum of an even number of identical values cancels out in characteristic two). This can be achieved as follows: For each output block, one runs through all useful sets of three Sboxes, and update a parity counter for the 15 output bits of these Sboxes, for each equation the current output block contributes to. Each parity counter consists of 2^{15} parity bits, which are used to store the parity of the number of occurrences of all values of the 15 output bits of the set of three Sboxes defining the counter. Then, one adds to each equation the contribution corresponding of each possible value of each counter.

We denote by N_{ctr} the number of counters that are used for each equation. In the general case, one needs a parity counter set for each possible combination of three output Sboxes, which makes $N_{ctr} = \binom{320}{3} \approx 2^{22.4}$. When the number of equations is optimized, we consider the multiplications of six bits after the external χ^{-1} layer with two dense polynomials. Therefore, one only needs at most $N_{ctr} = 6 \times \binom{320}{2} \approx 2^{18.2}$.

The memory complexity of this step is the amount of memory that is needed to store all the parity bits, which is $M_{ctr} = N_{par} \times N_{ctr} \times 2^{15}$ bits. The time required for the

attack encompasses the updates of parity counters and the addition of contributions of individual counter values to all the equations. The value of each counter only contributes to the coefficients of monomials in key bits at the 15 same positions. Therefore, adding the contribution of such a counter requires at most 2^{15} key additions. For each set of three Sboxes, there are 2^{15} counter values to consider. Therefore, we have: $T_{ctr} = N_{eq} \times N_{ctr} \times (S + 2^{30})$.

Comparison Between the Two Techniques. From the formulae above, we always have $T_{direct} > N_{eq}^2 \times S$. If T_{ctr} is smaller than this value, the time complexity of the second algorithm is better. This is equivalent to $N_{ctr} \times (S + 2^{30}) < N_{eq} \times S$. Moreover, the number of possible counters is bounded by $\binom{320}{3} \approx 2^{22.4}$, whereas the number of equations is at least $2^{29.0}$ when optimization of Section 5.1 is implemented. Therefore, a sufficient condition for the second algorithm to be more efficient becomes $S + 2^{30} \leq 2^{29.0-22.4}$, which is equivalent to $S \geq 2^{30-6.6} = 2^{23.4}$. In our attacks, S is the number of elements of an affine space of a higher order differential distinguisher or the number of nonzero coefficients of a recurrence polynomial. In all cases we focus on, it is larger than this bound.

The parity counter based attack should therefore be used in any case, unless one aims at optimizing the memory required by the attack and the storage of counter values is its bottleneck in terms of memory complexity.

5.4 Optimizing the Attacks

The high number of different attacks and potential combinations of optimizations makes it difficult to give an exhaustive list of all the possible combinations and their complexities. However, we give numerical applications for a few of them. These complexities are summarized in Table 1. We explain here how we obtain the optimized complexities.

Linear-Recurrence Attack on KRAVATTE-$(\star, 4)$ with $n_e = 2 + 2$. We use both optimizations of Section 5.1 and Section 5.3. As shown in Section 3.2, the precomputation step has a time complexity of $T_P = 2^{40.7}$ elementary operations and a memory complexity of $2^{28.7}$ bits to store the recurrence polynomial. As we reduce the number of variables, we only get one expression per block. The data complexity is then changed to $2^{28.7} + 2^{29.0} \approx 2^{29.9}$ blocks (from $2^{28.7} + \frac{1}{1600}2^{36.5}$). The time complexity to build the system is about $2^{77.5}$ operations, and the memory complexity to store the counters is $2^{62.2}$ bits. Finally, solving the system requires $\left(2^{29}\right)^3 = 2^{87}$ operations, and storing it about 2^{58} bits.

Overall, the time complexity of the attack is 2^{87} basic operations, the memory complexity is $2^{62.3}$ bits and the data complexity is $2^{29.9}$ blocks.

Linear-Recurrence Attack on KRAVATTE-$(\star, 6)$ with $n_e = 4 + 2$. We use the same optimizations for the attack on $n_e = 6$ rounds. The memory required for the attack is now mainly due to the storage of the recurrence polynomial, which requires $2^{88.4}$ bits. As the data complexity mainly comes from the high degree of this polynomial, it is still $2^{88.4}$ blocks as in Section 3.2. Finally, the time complexity of this attack is dominated by the construction of the system, which amounts to $2^{134.6}$ elementary operations.

Higher Order Differential Attack on KRAVATTE-$(4, 4)$ Using All Optimizations. We now focus on attacks based on the higher order differential distinguisher, and first try to apply all three optimizations. As shown in Section 5.2, the data complexity is $2^{74.7}$ blocks. During the construction phase, the memory required for the parity counters is again $2^{62.2}$ bits, and its time complexity is $2^{112.2}$ operations, which is the most time-consuming part of the attack. The memory and time complexities of the system resolution are the same as for the linear-recurrence attack, increasing the memory complexity to $2^{62.3}$ bits.

Memory-optimized Higher Order Differential Attack on KRAVATTE-$(4, 4)$. As the memory required mainly comes from the storage of parity counters, we can drop the optimization

based on super structures. The data complexity goes up to $2^{65} \times (65 + 2^{29}) \approx 2^{94}$ blocks, and the memory complexity drops to the 2^{58} bits required to store the system. The time complexity is left unchanged.

Data-optimized Higher Order Differential Attack on KRAVATTE-**(4, 4).** Similarly, to minimize the amount of data needed for the attack, we can drop the optimization of Section 5.1. As shown in Section 5.2, the data complexity decreases to $2^{73.9}$ blocks, but the number of equations required increases to $2^{36.5}$. The time and memory complexities of the construction step are respectively increased to $2^{123.9}$ operations and $2^{73.9}$ bits. Adding the storage of the system (2^{73} bits), the memory complexity of the attack becomes $2^{74.5}$ bits.

6 Concluding Remarks and Discussion

We proposed in this paper several key-recovery attack strategies breaking the security claims of the recent PRF proposal KRAVATTE. The attacks are primarily focused on either the convergence point or the divergence point of the high-level structure that allows to compress virtually any number of blocks to a single one in an incremental way, and conversely, to expand a single block to almost any number of output blocks. The properties of these two sensitive points of the computation, where all the input information is packed into a single block (right after the compressing phase and right before the second step of the expansion phase), together with the low algebraic degree of the `Keccak`-p permutation, are leveraged in our attacks. From this ambitious and aggressive design structure and the proposed attacks, we would like to draw some high-level conclusions.

First, the non-linear permutations p_c in the compression layer do not prevent the construction of an affine space at its output. This is inherent to the design and cannot be thwarted by simply increasing the number of rounds in p_c. Secondly, the middle non-linear permutation p_d applied to the accumulator does not increase the security of the expansion layer, as one can target the second step of the expansion layer, i.e., applications of p_e to the evolving rolling state, independently. The design incentive was probably to factor out a part of the non-linear transformations of the expansion layer to increase the performances of the output generations, but it appears that such an optimization strongly decreases the security. Finally, the last non-linear permutations p_e used to produce each output block in KRAVATTE have low algebraic degree and are applied after a small linear diversification mechanism. This results in a bit mixing much simpler than expected, which can be distinguished before the end of the expansion layer and used to recover the key by inverting the remaining part.

We note that one can interpret all ours attacks in terms of stream cipher analysis, with either attacks on the IV-processing part (the compression layer) or on the keystream generation part (the expansion layer). Recasting KRAVATTE in this light may help to improve its design. We have no strong views on how best to patch KRAVATTE while retaining its incrementality and parallelizability and without incurring too strong a penalty on its performance.

Acknowledgement

This work has been partially supported by the French Agence Nationale de la Recherche through the BRUTUS project under Contract ANR-14-CE28-0015.

References

[BDH+16] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Farfalle: parallel permutation-based cryptography. Cryptology ePrint Archive, Report 2016/1188, 2016.

[BDH+17a] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Farfalle: Parallel Permutation-Based Cryptography. *IACR Transactions on Symmetric Cryptology*, 2017(4), 2017.

[BDH+17b] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Innovations in Permutation-Based Crypto. Slides from ECC 2017, 2017.

[BDPA11] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The Keccak reference. Round 3 submission to NIST SHA-3, 2011.

[BGTZ08] Richard P. Brent, Pierrick Gaudry, Emmanuel Thomé, and Paul Zimmermann. Faster Multiplication in GF(2)[x]. In Alfred J. van der Poorten and Andreas Stein, editors, *Algorithmic Number Theory, 8th International Symposium, ANTS-VIII, Banff, Canada, May 17-22, 2008, Proceedings*, volume 5011 of *Lecture Notes in Computer Science*, pages 153–166. Springer, 2008.

[DLMW15] Itai Dinur, Yunwen Liu, Willi Meier, and Qingju Wang. Optimized interpolation attacks on LowMC. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 535–560. Springer, Heidelberg, November / December 2015.

[JK97] Thomas Jakobsen and Lars R. Knudsen. The interpolation attack on block ciphers. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 28–40. Springer, Heidelberg, January 1997.

[Key76] E.L. Key. An Analysis of the Structure and Complexity of Nonlinear Binary Sequence Generators. *IEEE Transactions on Information Theory*, 22(6):732–736, 1976.

[Lai94] Xuejia Lai. Higher Order Derivatives And Differential Cryptanalysis. *Kluwer International Series In Engineering And Computer Science*, pages 227–227, 1994.

[NIS14] NIST Computer Security Division. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. FIPS Publication 202, National Institute of Standards and Technology, U.S. Department of Commerce, May 2014.

[RGH07] S. Rønjom, G. Gong, and T. Helleseth. On Attacks on Filtering Generators Using Linear Subspace Structures. In S.W. Golomb, G. Gong, T. Helleseth, and H.-Y. Song, editors, *SSC*, volume 4893 of *Lecture Notes in Computer Science*, pages 204–217. Springer, 2007.

[RH07] S. Rønjom and T. Helleseth. A New Attack on the Filter Generator. *IEEE Transactions on Information Theory*, 53(5):1752–1758, 2007.

[Sch77] Arnold Schönhage. Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Inf.*, 7:395–398, 1977.

A Appendix

We explictly give the degree-320 recurrence polynomial P_{roll} used in Section 3 as

$$\begin{aligned}P_{roll} = {} & x^{320} + x^{262} + x^{246} + x^{234} + x^{218} + x^{204} + x^{202} + x^{198} + x^{191} + x^{186} + x^{183} + x^{182} \\ & + x^{176} + x^{172} + x^{134} + x^{133} + x^{131} + x^{125} + x^{123} + x^{117} + x^{109} + x^{106} + x^{105} + x^{102} \\ & + x^{99} + x^{97} + x^{91} + x^{90} + x^{89} + x^{88} + x^{81} + x^{76} + x^{74} + x^{70} + x^{69} + x^{67} + x^{64} + x^{63} \\ & + x^{61} + x^{60} + x^{59} + x^{58} + x^{55} + x^{53} + x^{48} + x^{47} + x^{45} + x^{44} + x^{41} + x^{39} + x^{38} + x^{35} \\ & + x^{33} + x^{28} + x^{27} + x^{25} + x^{24} + x^{17} + x^{15} + x^{7} + 1.\end{aligned}$$

We note that the minimal polynomial given by the KRAVATTE designers in [BDH+16] corresponds to the inverse matrix $\mathbf{M}_{roll}^{-1}$.

Also, we visually represent the membership matrix discussed in the optimization relying of the super structures from Section 5.2.

	1111	1110	1101	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	...
****	1	1	1	1	1	1	1	1	1	1	1	1	1	...
***0		1		1		1		1		1		1		...
**0*			1	1			1	1			1	1		...
**00				1				1				1		...
*0**					1	1	1	1						...
*0*0						1		1						...
00							1	1						...
0***									1	1	1	1	1	...
0**0										1		1		...
0*0*											1	1		...
00**													1	...

Figure 6: Example of structure/message membership matrix, with $(n, t) = (2, 2)$.