Resultants Meet Resultant: Improving CICO-1 and CICO-2 Attacks on ZK-Friendly Permutations

Antoine Bak[1,2], Augustin Bariant[3], Maël Hostettler[4], and Vincent Neiger[5]

[1] Inria, Paris, France
[2] DGA, Paris, France
[3] ANSSI, Paris, France
[4] Ruhr University Bochum, Bochum, Germany
[5] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
antoine.bak@inria.fr, augustin.bariant@ssi.gouv.fr
mael.hostettler@telecom-sudparis.eu, vincent.neiger@lip6.fr

Abstract. The increasing usage of Zero-knowledge proof protocols has raised the need for cryptographic primitives that are efficient in that setting, called Arithmetization-oriented primitives. The security of such permutations is commonly evaluated with the CICO-k problem.

The best known CICO-1 attack against ZK-Friendly permutations over $\mathbb{F}_q^t$ based on α-inversions $x \mapsto x^{1/\alpha}$ exploits resultants (ASIACRYPT 2024, CRYPTO 2025). It starts from *one* input variable x and introduces a temporary variable after each α-inversion. With an efficient procedure to eliminate temporary variables, the attack reaches a time and memory complexity of $\tilde{\mathcal{O}}(D_I(2-1/\alpha)^n)$, where D_I is the ideal degree of the CICO-1 modeling and n is the number of α-inversions. In this work, we study such an approach using *two* input variables x_1, x_2, and we generalize the temporary variable elimination to that setting. Subsequently, we present a new CICO-2 attack framework and a new Start-From-The-Middle (SFTM) CICO-1 attack framework.

Both our attacks rely on fast bivariate resultants for their final bivariate system solving step. Using resultant algorithms with complexity almost linear in D_I, our CICO-2 and CICO-1 attacks reach a complexity almost linear in $\alpha^n D_I$ and in D_I, respectively, which is a first theoretical improvement. Designing an efficient implementation of these resultant algorithms remains a challenge, so for our practical contributions we turn to Villard's algorithm (ISSAC 2018). After adapting it to our context, we obtain practical complexities $\tilde{O}((\alpha^n D_I)^{\gamma_2})$ and $\tilde{O}(D_I^{\gamma_1})$ for CICO-2 and CICO-1 respectively, where $1.2 \leq \gamma_1 \leq 1.25 \leq \gamma_2 \leq 1.33$ depending on the chosen linear algebra exponent $2 \leq \omega \leq 3$.

Our attacks improve upon the best known ones against several instances of `Anemoi`, *Rescue* and Griffin, successfully breaking 128-bit and 256-bit security instances of *Rescue* in the CICO-1 setting and full-round instances of `Anemoi` and Griffin in the CICO-2 setting for the first time. Our implementation of the attack confirms the practicality of the approach.

1 Introduction

Over the past decade, there has been growing interest in advanced protocols such as Zero-Knowledge proofs, Multi-Party Computation, and Fully Homomorphic Encryption. Most of these protocols operate over finite fields, using addition and multiplication as their fundamental operations, and some applications require symmetric primitives that are compatible with those operations. Converting traditional primitives such as the AES or SHA3 into finite field operations is prohibitively expensive in practice. Instead, cryptographers have developed dedicated primitives designed to minimize the number of finite field operations; these primitives are known as Arithmetization-Oriented (AO) primitives. The computational cost of such primitives depends on the protocol, but it is generally related to the number of finite field multiplications required.

The case of Zero-Knowledge protocols is somewhat special. Hash functions play a central role in commitments: they are evaluated by the prover in any environment but verified within the protocol. In this context, an AO hash function is considered efficient if it can be verified efficiently in finite fields, rather than simply evaluated efficiently. We refer to [12] for more details. In this setting, mappings of the form $x \mapsto x^{1/\alpha}$ with small α, called α-inversions, are commonly used as components in ZK-friendly AO hash functions, such as *Rescue* [1], Arion [33], `Anemoi` [12], Griffin [19], or XHash8/12 [2]. These algorithms use the sponge construction [10] from AO permutations, and employ α-inversions to drastically increase the univariate degree of the permutations with little verification cost overhead in a ZK setting.

The security of such permutations is commonly analyzed with the Constrained-Input Constrained-Output (CICO) problem [9].

Definition 1 (CICO-k). *Let $\Pi : \mathbb{F}_q^t \mapsto \mathbb{F}_q^t$ be a permutation and $k \leq t$. The CICO-k problem consists in finding an input/output pair $(\mathbf{x}, \mathbf{y})$ of the permutation with the first k coordinates of both $\mathbf{x}$ and $\mathbf{y}$ being 0, i.e.*

$$\Pi(0, \ldots, 0, x_{k+1}, \ldots, x_t) = (0, \ldots, 0, y_{k+1}, \ldots, y_t).$$

The CICO-k problem on a permutation Π is tightly linked with the preimage resistance of the hash function using Π as underlying permutation in the sponge construction [9]. The CICO-k problem can be solved generically, using in average q^k (inverse) permutation calls, but knowing if faster dedicated attacks exist on a fixed permutation requires cryptanalysis. Since the statistical properties of finite field multiplications are strong, especially when the field is large, statistical cryptanalysis such as differential or linear cryptanalysis are often less threatening than algebraic attacks to solve this type of problem. This is the main reason for the differences in approach of cryptanalysis of traditional and AO primitives.

Existing CICO cryptanalysis. The most efficient attacks against AO permutations are a special type of algebraic attacks known as polynomial-solving attacks. In this type of attacks, the attacker models the CICO-k solutions with

a set of polynomial equations, which he solves using off-the-shelf tools from computer algebra, such as Gröbner basis algorithms or, when available, faster dedicated tools that exploit the structure of the polynomial system. Among the CICO-k problems, the CICO-1 problem has received the most attention against AO primitives [7,6,28,40,5]. The problem is particularly relevant when the underlying field is large (typically, a cardinality q of ≥ 256 bits), since it might directly affect the security of the corresponding hash function. More generally, it also serves as a basis of comparison and gives tools to cryptographers for further analysis of CICO-k problems with greater k. For instance, the CICO-1 FreeLunch attack [6] was the base of study of the CICO-k CheapLunch attack [3].

CICO-1 cryptanalysis of ZK-friendly permutations. When the AO permutation is based on a round function of low degree, such as in Poseidon [20], the most efficient way to solve the CICO-1 problem is to set the input to affine polynomials in an unknown variable x in such a way that is satisfies the input constraint of the CICO-1 problem. Then, the permutation is evaluated on $\mathbb{F}_q[x]^t$ (rather than $\mathbb{F}_q^t$) and the first output element of the permutation is a polynomial $P(x) \in \mathbb{F}_q[x]$ whose roots in $\mathbb{F}_q$ define valid solutions for the CICO-1 problem. All one has to do is recover its roots, in quasi-linear time in the degree of P. See [7] for more details.

However, against ZK-friendly permutations based on α-inversions, the round function is no longer of low-degree. Instead, the traditional way of solving CICO-1 against these permutations is to build a system of low-degree multivariate equations modeling CICO-1 solutions and then solve it. To do so, the attacker first sets the input to affine polynomials in the variable x that satisfies the input CICO-1 constraint. He then evaluates the permutation on polynomials, except the α-inversions, which he treats differently: he adds a variable z_i after each such α-inversion, and stores the equation linking z_i^α to the input of the α-inversion in a polynomial system. Eventually, the first output element of the permutation is a polynomial in $\mathbb{F}_q[x, z_1, ..., z_n]$ (where n α-inversions required a new variable) that the attacker adds to the polynomial system. The produced polynomial system has a triangular structure, which was exploited with Gröbner bases techniques in the FreeLunch attack [6], and more recently with resultant techniques in [40,5]. A crucial observation of [5] is that multiplying polynomials of $\mathbb{F}_q[x, z_1, ..., z_n]/\langle z_1^\alpha - P_1(x), \ldots, z_n^\alpha - P_n(x, z_1 \ldots z_{n-1})\rangle$ can be done somewhat efficiently with the Kronecker trick, adding "only" an overhead factor of $(2 - 1/\alpha)^n$ (omitting logarithmic factors) compared to the size of the polynomials, when they are dense in their degree.

The SFTM approach. The Start-From-The-Middle (SFTM) approach was introduced at ASIACRYPT 2024 [40] to solve CICO-1 problems on *Rescue* and `Anemoi`. The main idea is to split the permutation into $\Pi = \Pi_1 \circ \Pi_0^{-1}$, and to set the middle state as an affine space $\mathcal{L}(x, y)$ in two variables x, y. The attacker then evaluates Π_1 in $\mathbb{F}_q[x, y]$ (starting from the input $\mathcal{L}(x, y)$), and adds a new variable z_i and a new equation for each encountered α-inversion. He performs

similar computations for Π_0 with a new equation and variable per α-inversion. All-in-all, the attacker models the CICO-1 solutions with a system with $n+2$ equations and variables, where n is the number of α-inversions in Π_0 and Π_1. The authors of [40] then remark that the z_i's can be eliminated with resultants, yielding a bivariate system in x and y, and which they solve by computing the determinant of a Sylvester matrix using fast Lagrange interpolation techniques. The bottleneck of the attack of [40] lies in solving such a bivariate system.

Contribution. In this work, we describe a generalization of the variable elimination procedure from [5]. Using this generalization, combined with efficient bivariate solving techniques, we provide a generalization of the resultant attack to the CICO-2 problem, and improve the CICO-1 attacks on some primitives. For both applications, we support our theoretical claims with practical implementations on round-reduced versions of the primitives.

Generalizing the variable elimination procedure from [5]. We generalize the algorithm from [5] that multiplies polynomials in $\mathbb{F}_q[x, z_1, ..., z_n]/\langle z_1^\alpha - P_1(x), \ldots, z_n^\alpha - P_n(x, z_1 \ldots z_{n-1})\rangle$ to quotient rings of the form $\mathbb{F}_q[x_1, \ldots, x_k, z_1, ..., z_n]/\langle z_1^\alpha - P_1(x_1, \ldots, x_k), \ldots, z_n^\alpha - P_n(x_1, \ldots, x_k, z_1 \ldots z_{n-1})\rangle$. Interestingly, as long as $k \geq 2$, such a multiplication can be performed in time that is quasi-linear in the size of the output. This is in contrast to the case $k = 1$ where there is a $(2 - 1/\alpha)^n$ overhead. Using the iterated resultant technique, this leads to an efficient procedure to get a polynomial in $\langle Q(x_1, \ldots, x_k, z_1 \ldots z_n), z_1^\alpha - P_1(x_1, \ldots, x_k), \ldots, z_n^\alpha - P_n(x_1, \ldots, x_k, z_1 \ldots z_{n-1})\rangle \cap \mathbb{F}_q[x_1, \ldots, x_k]$, that is eliminating the variables z_i from the system of equations.

Resultant-based CICO-2 attacks. Our approach mostly follows the modeling from [5], except that we introduce two variables x_1, x_2 in the input, and get two polynomials $Q_1, Q_2 \in \mathbb{F}_q[x_1, x_2, z_1, \ldots, z_n]$ representing the output constraints. Using our variable elimination procedure on Q_1 and Q_2, we get a bivariate system in x_1, x_2, that we can solve using the Villard algorithm [38]. We apply this result on Griffin and `Anemoi`, giving the first full-round breaks in the CICO-2 setting for some versions of Griffin with 128-bit security, and some versions of `Anemoi` with 256-bit security.

Improved Start-from-the-middle (STFM) CICO-1 attacks. The second application is a better STFM attack on the CICO-1 problem. The asymptotic complexity of our STFM attack improves upon the complexity of the best attack in the literature, i.e. the resultant attack [40,5]. We give a careful complexity analysis of the attack, including an adaptation of Villard's algorithm to compute bivariate resultants from a system of dense bivariate polynomials in their total degrees. We apply our attack to instances of *Rescue* and `Anemoi`, yielding the first attacks on full-round instances of *Rescue* with 128-bit and 256-bit security.

Outline. In Section 2, we set up our notations and give an overview of known results on the complexity of polynomial operations and properties of resultants. In Section 3, we describe our generalized variable elimination procedure and our adaptation of Villard's algorithm to polynomials dense in their total degree. In Section 4, we describe our CICO-2 and Start-from-the-Middle CICO-1 attacks, and analyze their complexity. In Section 5, we apply the attacks to *Rescue*, `Anemoi` and Griffin, and present our experimental results.

2 Background

2.1 Notations

In this paper, $\mathbb{F}_q$ denotes the finite field with q elements. For a given field $\mathbb{F}$ (for example $\mathbb{F} = \mathbb{F}_q$), $\mathbb{F}[x_1, \ldots, x_k]$ denotes a polynomial ring over $\mathbb{F}$ with k variables, that we sometimes shorten into $\mathbb{F}[\mathbf{x}]$ (where $\mathbf{x}$ denotes $x_1, \ldots, x_k$). Indices are used to denote degree constraints: for example $\mathbb{F}[\mathbf{x}]_{<d}$ is the set of polynomials of degree strictly less than d.

We say that a monomial $m = \mathbf{x}^\alpha = \prod_{i=1}^k x_i^{\alpha_i}$ belongs to a polynomial $P \in \mathbb{F}[\mathbf{x}]$, and we write $m \in P$, if the coefficient of the monomial m in the expression of P on the monomial basis of $\mathbb{F}[\mathbf{x}]$ is nonzero. Fix a nonzero polynomial $P \in \mathbb{F}[\mathbf{x}]$. Its total degree is $\mathrm{tdeg}(P) = \max\{\sum_{i=1}^k \alpha_i \mid \mathbf{x}^\alpha \in P\}$. More generally, its $\mathbf{w}$-weighted degree, for a given weight $\mathbf{w} = (w_{x_1}, \ldots, w_{x_k}) \in (\mathbb{R}^+)^k \setminus \{(0, \ldots, 0)\}$, is defined as

$$\mathrm{wdeg}_{\mathbf{w}}(P) = \max\left\{\sum_{i=1}^k w_{x_i}\alpha_i \mid \mathbf{x}^\alpha \in P\right\} .$$

The partial degree of P w.r.t. x_i, or its x_i-degree for short, is denoted by $\deg_{x_i}(P)$ and is equal to the degree of P in x_i. By convention, we set these degrees to $-\infty$ for $P = 0$.

In this work, we will sometimes use two types of variables, x_i's and z_i's, and denote $R_{k,n} = \mathbb{F}[x_1, \ldots, x_k, z_1, \ldots z_n]$. The previous notations naturally extend to the z_i's. We call the variables $x_1, \ldots, x_k$ the $\mathbf{x}$-variables, and the variables $z_1, \ldots, z_n$ the $\mathbf{z}$-variables.

In complexity bounds, we use the soft-O notation which indicates asymptotic bounds with hidden polylogarithmic factors: $g(n) = \tilde{O}(f(n))$ when there exists $d \in \mathbb{N}$ such that $g(n) = O(f(n)\log(f(n))^d)$.

2.2 Polynomial operations

In this paper, we extensively exploit fast polynomial operations with quasi-linear complexities.

Fast polynomial multiplication. We express complexity bounds using a time function $\mathsf{M}(d)$ for the complexity of polynomial multiplication [17, Def. 8.26], meaning that two polynomials $P, Q \in \mathbb{F}[x]_{\leq d}$ can be multiplied using at most

$\mathsf{M}(d)$ operations in $\mathbb{F}$. Using an FFT-based approach [14], we can take $\mathsf{M}(d)$ to be quasi-linear in d, more precisely $\mathsf{M}(d) = O(d\log(d)\log(\log(d))) = \tilde{O}(d)$; better bounds are known when $\mathbb{F} = \mathbb{F}_q$ [22, Thm. 8.4] (see also [21], in the bit complexity model). A property that we will use in the paper is that asymptotically, $\mathsf{M}(a) + \mathsf{M}(b) \leq \mathsf{M}(a+b)$. Then, two multivariate polynomials $P, Q \in \mathbb{F}[\mathbf{x}]$ of degree d_i in x_i can be multiplied in $O(\mathsf{M}(\prod_{i=1}^{k}(2d_i+1)))$ operations in $\mathbb{F}$, using Kronecker substitution [17, Sec. 8.4]. In particular, these univariate and k-variate multiplications have quasi-linear complexity with respect to the size of a dense representation of $P \cdot Q$.

Fast univariate root-finding. Let $P \in \mathbb{F}_q[x]$ be a polynomial of degree d, with $d \in O(q)$. As showed in [17, Algo. 14.15], finding all roots of P in $\mathbb{F}_q$ can be done by first computing the GCD of P with $x^q - x$, which yields a polynomial $Q \in \mathbb{F}_q[x]$ which splits as a product of $r = \deg(Q)$ factors of degree 1, and then running an equal-degree factorization algorithm to find these factors which provide the roots of P. The last factorization step costs $O((\log(q)+\log(r))\,\mathsf{M}(r)\log(r))$ operations in $\mathbb{F}_q$ [17, Thm. 14.11]. Since $r \leq d$ this is in $O(\mathsf{M}(r)\log(q)\log(r))$. For computing the GCD we first find $x^q \bmod P$, in $O(\mathsf{M}(d)\log(q))$ field operations, and then use the half-gcd in $O(\mathsf{M}(d)\log(d))$ field operations [17, Cor. 11.9]. We get a total of $O((\mathsf{M}(d) + \mathsf{M}(r)\log(r))\log(q))$ operations in $\mathbb{F}_q$ to find all roots of P, where the number r of these roots is found out during the computation and is typically negligible compared to d.

Resultants. Several components of the paper and of our practical implementation of our attack make use of resultants. In our use cases, the domain $\mathcal{R}$ will be a polynomial ring (univariate or multivariate).

Definition 2 (Sylvester matrix, resultant). *Let $\mathcal{R}$ be an integral domain and let $P = P_0 + P_1 x + \cdots + P_d x^d$ and $Q = Q_0 + Q_1 x + \cdots + Q_e x^e$ be two polynomials in $\mathcal{R}[x]$ of respective degrees d and e. Then, the* Sylvester matrix *of P and Q is*

$$\mathsf{S} = \begin{bmatrix} P_0 & P_1 & \cdots & P_d & & \\ & \ddots & \ddots & & \ddots & \\ & & P_0 & P_1 & \cdots & P_d \\ Q_0 & Q_1 & \cdots & Q_e & & \\ & \ddots & \ddots & & \ddots & \\ & & Q_0 & Q_1 & \cdots & Q_e \end{bmatrix} \in \mathcal{R}^{(d+e)\times(d+e)},$$

where there are e rows with coefficients of P and d rows with coefficients of Q, and where blank entries are zero. The resultant *of P and Q (with respect to the variable x), denoted by $\mathrm{Res}_x(P,Q)$, is defined as the determinant $\det(\mathsf{S}) \in \mathcal{R}$.*

Proposition 1 ([15, Chap. 3, §1]). *With the notation in Definition 2, let further $(\mu_1, \ldots, \mu_d)$ and $(\nu_1, \ldots, \nu_e)$ be the respective roots of P and Q, counted*

with multiplicity, in the algebraic closure of $\mathcal{R}$. Then,

$$\mathrm{Res}_x(P,Q) = P_d^d Q_e^e \prod_{i=1}^{d}\prod_{j=1}^{e}(\mu_i - \nu_j) = P_d^d \prod_{i=1}^{d} Q(\mu_i) = (-1)^{de} Q_e^e \prod_{j=1}^{e} P(\nu_j).$$

Proposition 2 ([16, Chap. 3, §6]). *With the notation in Definition 2 and $\langle P, Q\rangle$ the ideal spanned by P and Q, we have that:*

$$\mathrm{Res}(P,Q) \in \langle P, Q\rangle \cap \mathcal{R} \ .$$

3 Quasi-triangular ideals and resultants

In this section, we recall and generalize some results established in [5] and adapt some results on bivariate resultants. We recall that $R_{k,n} = \mathbb{F}[x_1, \ldots, x_k, z_1, \ldots z_n]$ and start with a definition of quasi-triangular ideal.

Definition 3 (Quasi-triangular ideal). *We say that $I \subset R_{k,n}$ is a* quasi-triangular *ideal if it is of the form:*

$$I = \langle z_i^\alpha - P_i \mid 1 \leq i \leq n\rangle \ ,$$

where $P_i \in R_{k,i-1}$ for $1 \leq i \leq n$.

For such a quasi-triangular ideal I, we denote $I_j = \langle z_i^\alpha - P_i \mid 1 \leq i \leq j\rangle$ for $1 \leq j \leq n$. In [5], the authors solve CICO-1 against several AO permutations by finding a univariate polynomial in x in the ideal spanned by the following equations:

$$\begin{cases} z_j^\alpha = P_j(x, z_1, \ldots, z_{j-1}) \ , \ 1 \leq j \leq n \\ Q(x, z_1, \ldots, z_n) = 0 \ , \end{cases}$$

which is of the form $\langle I, Q\rangle$ where $I \subset R_{1,n}$ is quasi-triangular. To that end, they present an efficient representation and multiplication procedure in $R_{1,n}/I$.

3.1 Multiplication modulo quasi-triangular ideals

Inspired by [5], we provide an efficient multiplication procedure in $R_{k,n}/I$ where I is quasi-triangular and where $k \geq 2$, while [5] only considered the case $k = 1$. First, we represent elements of the quotient ring $R_{k,n}/I$ under the infinite $\mathbb{F}$-basis:

$$\mathcal{B}_{k,n} = \{\mathbf{x}^{\mathbf{a}} \cdot \mathbf{z}^{\mathbf{b}} \mid \mathbf{a} \in \mathbb{N}^k, \mathbf{b} \in \{0, \ldots, \alpha - 1\}^n\} \ .$$

A polynomial of $R_{k,n}/I$ represented as a $\mathbb{F}$-linear combination of elements of $\mathcal{B}_{k,n}$ is said to be reduced.

Monomial weights. Following [5], we define monomial weights to estimate the complexity of the multiplication in $R_{k,n}/I$, as follows:

- $w_{x_i} = 1$ for all $1 \leq i \leq k$.
- $w_{z_j} = \mathrm{wdeg}(P_j(x_1, \ldots, x_k, z_1, \ldots, z_{j-1}))/\alpha$ for all $1 \leq j \leq n$.

We also adapt the reduction algorithm from [5] to a case with more than one $\mathbf{x}$-variable:

Algorithm 1 $\mathrm{Reduce}_j(g(x_1, \ldots, x_k, z_1, \ldots, z_j), I_j)$

Input: A polynomial $g \in \mathbb{F}_q[x_1, \ldots, x_k, z_1, \ldots, z_j]$, where $\deg_{z_i}(g) < 2\alpha - 1$ for $1 \leq i \leq j$, and a quasi-triangular ideal $I_j = \langle z_i^\alpha - P_i \mid 1 \leq i \leq j \rangle$.
Output: The reduced form of g with only monomials in $\mathcal{B}_{k,j}$.

1: **if** $i = 0$ **then**
2: return g
3: write g as $g = \sum_{i=0}^{2\alpha-2} g_i(x_1, \ldots, x_k, z_1, \ldots, z_{j-1}) z_j^i$
4: $\rho \leftarrow \mathrm{Reduce}_{j-1}(g_{\alpha-1}, I_{j-1}) \cdot z_j^{\alpha-1}$
5: **for** $i = 0$ to $\alpha - 2$ **do**
6: $\rho \leftarrow \rho + \mathrm{Reduce}_{j-1}(g_i + \mathrm{Reduce}_{j-1}(g_{\alpha+i}, I_{j-1}) \cdot P_j, I_{j-1}) \cdot z_j^i$
▷ $2\alpha - 1$ *calls to* Reduce_{j-1} *in total*
7: **return** ρ

This allows to derive the following result.

Proposition 3 (Multiplication in $R_{k,n}/I$). *Given a quasi-triangular ideal $I \subset R_{k,n}$ and two reduced polynomials $P, Q \in R_{k,n}/I$, the polynomial $S = \mathrm{Reduce}_n(P \cdot Q, I) \in R_{k,n}/I$ can be computed in its reduced form in time:*

$$O\left(n \cdot \mathsf{M}\big((2\alpha-1)^n (d_P + d_Q)^k\big)\right) = \tilde{O}\left((2\alpha-1)^n (d_P + d_Q)^k\right) ,$$

where $d_P = \mathrm{wdeg}(P), d_Q = \mathrm{wdeg}(Q)$.

Proof. We mostly follow the steps from [5, Algorithm 1]. We denote by $d_{\mathbf{x}}$ the maximum degree of the x_i's manipulated by the algorithm, and by $C(j, d_{\mathbf{x}})$ the cost of a call to Reduce_j on a polynomial with degree in the x_i's bounded by $d_{\mathbf{x}}$. We can prove the following loop invariants:

- At each step, the degree in the x_i is upper bounded by $d_{\mathbf{x}} \leq d_P + d_Q$.
- When calling Reduce_j, we make $2\alpha - 1$ calls to Reduce_{j-1}, and $\alpha - 1$ multiplications of reduced polynomials whose result will have degree in the x_i at most $d_{\mathbf{x}}$. Said otherwise:

$$C(i, d_{\mathbf{x}}) \leq (2\alpha-1) C(i-1, d_{\mathbf{x}}) + (\alpha-1)\, \mathsf{M}\big((2\alpha-1)^{i-1} d_{\mathbf{x}}^k\big) .$$

We have that $C(0, d_{\mathbf{x}}) = 0$. We prove now by induction that for all i:

$$C(i, d_{\mathbf{x}}) \leq i \cdot \mathsf{M}\big((2\alpha-1)^i d_{\mathbf{x}}^k\big) .$$

This is true for $i = 0$, and if it is true for $i-1$, then:

$$\begin{aligned} C(i, d_{\mathbf{x}}) &\leq (2\alpha-1)C(i-1, d_{\mathbf{x}}) + (\alpha-1)\,\mathsf{M}\big((2\alpha-1)^{i-1} d_{\mathbf{x}}^k\big) \\ &\leq (2\alpha-1)(i-1)\cdot \mathsf{M}\big((2\alpha-1)^{i-1} d_{\mathbf{x}}^k\big) + (\alpha-1)\,\mathsf{M}\big((2\alpha-1)^{i-1} d_{\mathbf{x}}^k\big) \\ &\leq i \cdot \mathsf{M}\big((2\alpha-1)^{i} d_{\mathbf{x}}^k\big) \ . \end{aligned}$$

Applying this result to $i = n$ and using $d_{\mathbf{x}} \leq d_P + d_Q$ yields

$$C(n, d_{\mathbf{x}}) = O\left(n \cdot \mathsf{M}\big((2\alpha-1)^n (d_P + d_Q)^k\big)\right) \ .$$

□

3.2 Resultant computation

We first recall the following result, which is a particular case of Proposition 1 applied to Q and $z_i^\alpha - P_i$.

Proposition 4 (Similar to [5], Proposition 6). *Let $\overline{\mathbb{F}}$ and $\overline{R}_{i-1}$ denote respectively the algebraic closure of $\mathbb{F}$ and $R_{k,i-1}$. The following holds:*

$$\mathrm{Res}_{z_i}(Q, z_i^\alpha - P_i) = \pm \prod_{j=1}^{\alpha} Q(\theta \chi^j) \ ,$$

where $\chi \in \overline{\mathbb{F}}$ is a α-th primitive root of unity, and $\theta \in \overline{R}_{i-1}$ is an α-th root of P_i.

Proof. Let $\theta \in \overline{R}_{i-1}$ such that $\theta^\alpha = P_i$ and $\chi \in \overline{\mathbb{F}}$ a primitive α-th root of 1. Then, the polynomial $z_i^\alpha - P_i \in R_{k,i-1}[z_i]$ factors out as:

$$z_i^\alpha - P_i = \prod_{j=1}^{\alpha} (z_i - \theta \chi^j) \ .$$

Using the last equality in Proposition 1 yields the result. □

Proposition 5. *Let $k \geq 2$ and $Q \in R_{k,j}/I_j$. The resultant $\mathrm{Res}_{z_j}(Q, z_j^\alpha - P_j) \in R_{k,j-1}/I_{j-1}$ is of weighted degree bounded by $\alpha \cdot \mathrm{wdeg}(Q)$ and can be computed in its reduced form in time:*

$$\tilde{O}\left((2\alpha-1)^{j+1} (\alpha \cdot \mathrm{wdeg}(Q))^k\right) \ .$$

Proof. To prove this statement, we use Proposition 4 by adding variables θ, χ and defining the quasi-triangular-like ideal[6]:

$$\tilde{I}_j = \langle z_1^\alpha - P_1, \ldots, z_{j-1}^\alpha - P_{j-1}, \theta^\alpha - P_j, \chi^{\alpha-1} + \cdots + \chi + 1 \rangle \ .$$

[6] The degree of the last equation is $\alpha - 1$ so the ideal is not "stricto sensu" quasi-triangular, but the multiplication bounds from Proposition 3 still apply. As a side note, one could replace the last equation by $\chi^\alpha - 1$ to get a quasi-triangular ideal.

Using Proposition 3, we can multiply two polynomials P, Q in

$$\mathbb{F}[\mathbf{x}, z_1, \ldots, z_{j-1}, \theta, \chi]/\tilde{I}_j$$

in time

$$\tilde{O}\left((2\alpha-1)^{j+1}(d_P + d_Q)^k\right) \ .$$

To compute the product from Proposition 4, we make use of a product tree, as described in [4, Appendix A]. That is, we start by multiplying the $Q(\theta\chi^j)$ two by two to get $\lceil \alpha/2 \rceil$ polynomials of weighted degree at most $2\mathrm{wdeg}(Q)$, using $\lfloor \alpha/2 \rfloor$ multiplications. Then, we iterate so that at step j, we get $\lceil \alpha/2^j \rceil$ polynomials of weighted degree at most $2^j\mathrm{wdeg}(Q)$, until we get only one polynomial of weighted degree $\alpha\mathrm{wdeg}(Q)$. The total cost is upper bounded by:

$$\begin{aligned}
&\tilde{O}\left(\sum_{i=1}^{\lceil \log_2(\alpha) \rceil} \lfloor \alpha/2^{i+1} \rfloor \cdot (2\alpha-1)^{j+1}(2^i \mathrm{wdeg}(Q))^k\right) \\
&= \tilde{O}\left((2\alpha-1)^{j+1}\mathrm{wdeg}(Q)^k \sum_{i=1}^{\lceil \log_2(\alpha) \rceil} \lfloor \alpha/2^{i+1} \rfloor \cdot 2^{ik}\right)
\end{aligned}$$

The following bound yields the sought result for $k \geq 2$:

$$\sum_{i=1}^{\lceil \log_2(\alpha) \rceil} \lfloor \alpha/2^{i+1} \rfloor \cdot 2^{ik} \leq \alpha/2 \sum_{i=1}^{\lceil \log_2(\alpha) \rceil} 2^{i(k-1)} = \alpha/2 \cdot \frac{2^{(k-1)\lceil \log_2(\alpha) \rceil} - 2}{2^{k-1} - 1} = O\left(\alpha^k\right) \ .$$

□

The procedure `zElim`. We define the procedure `zElim`, whose objective is to eliminate the $\mathbf{z}$-variables in the system. This procedure is defined in Algorithm 2.

Algorithm 2 $\mathsf{zElim}(I, Q)$

Input: quasi-triangular ideal $I \subset R_{k,n}$, reduced polynomial $Q \in R_{k,n}/I_{k,n}$,
Output: a polynomial $Q^{(\mathrm{red})} \in \mathbb{F}[\mathbf{x}] \cap (I + \langle Q \rangle)$

1: $Q^{(n)} \leftarrow Q$
2: **for** $j = n$ down to 1 **do**
3: $\quad Q^{(j-1)} \leftarrow \mathrm{Res}_{z_j}(Q^{(j)}, z_j^\alpha - P_j) \pmod{I_{j-1}}$
4: $Q^{(\mathrm{red})} \leftarrow Q^{(0)}$
5: **return** $Q^{(\mathrm{red})}$

Proposition 6. *Let $k \geq 2$. The procedure* `zElim` *takes as input an quasi-triangular ideal $I \subset R_{k,n}$ and a reduced polynomial $Q \in R_{k,n}/I_{k,n}$ and has the following properties:*

1. $\texttt{zElim}(I, Q) \in (I + \langle Q \rangle) \cap \mathbb{F}[x_1, \ldots, x_k]$.
2. $\mathrm{tdeg}(\texttt{zElim}(I, Q)) \leq \mathrm{wdeg}(Q) \cdot \alpha^n$.
3. *It has complexity*
$$\tilde{O}\left(\alpha^2(\alpha^n \mathrm{wdeg}(Q))^k\right) ,$$

Proof. We define $Q^{(n)} = Q$, and $Q^{(j)}$ the result of the $n-j$-th step of the procedure. The following properties hold by induction on $j = n, \ldots, 1$, using Proposition 2 and 5:

1. $Q^{(j)} \in (I + \langle Q \rangle) \cap \mathbb{F}[x_1, \ldots, x_k, z_1, \ldots, z_j]$.
2. $\mathrm{wdeg}(Q^{(j)}) \leq \mathrm{wdeg}(Q) \cdot \alpha^{n-j}$.
3. The $n-j$-th step has complexity
$$O\left(j \cdot \mathsf{M}\left(\alpha^{(n-j)k}(2\alpha-1)^{j+2}\mathrm{wdeg}(Q)^k\right)\right) = \tilde{O}\left(\alpha^{(n-j)k}(2\alpha-1)^{j+2}\mathrm{wdeg}(Q)^k\right) .$$

Since $Q^{(0)} = \texttt{zElim}(I, Q)$, this proves the first two points of Proposition 6. The third point can be proven by summing the complexity of each step, as follows:

$$\begin{aligned}\sum_{j=1}^{n} \tilde{O}\left(\alpha^{(n-j)k}(2\alpha-1)^{j+2}\mathrm{wdeg}(Q)^k\right) &= \tilde{O}\left(\alpha^{nk}\mathrm{wdeg}(Q)^k\alpha^2 \sum_{j=1}^{n} \frac{(2\alpha-1)^j}{\alpha^{jk}}\right) \\ &= \tilde{O}\left(\alpha^2(\alpha^n \mathrm{wdeg}(Q))^k\right),\end{aligned}$$

since $\frac{2\alpha-1}{\alpha^k} < 5/9$ for $\alpha \geq 3$ and $k \geq 2$. This proves the third point.

□

The time complexity is essentially optimal, as the output of the procedure is expected to be of size $(\alpha^n \cdot \mathrm{wdeg}(Q))^k$. This is in contrast with the case where $k = 1$ (treated in [5]), where the time complexity of the procedure is in $\tilde{O}((2\alpha-1)^{n+2}\mathrm{wdeg}(Q))$, and the procedure's output size grows with $\alpha^n \mathrm{wdeg}(Q)$.

3.3 Bivariate resultants

The most efficient way to solve a generic bivariate system $P(x, y) = Q(x, y) = 0$ is to compute their bivariate resultant. Several algorithms exist for this purpose. Theoretically, one can use finite field bivariate resultant approaches, such as those due to van der Hoeven and Lecerf [24] or to Villard [39], which have a complexity of $\mathcal{O}(d^{2+\epsilon}\log(q)^{1+o(1)})$ where $d = \max(d_P, d_Q)$ is a bound on the total degrees of P and Q and ϵ is any strictly positive constant. These results rely on algorithms due to Kedlaya-Umans [27], and in particular on an asymptotically efficient

multivariate multi-point evaluation routine. Unfortunately, we are not aware of an efficient implementation of this, and it is currently unclear whether such implementation could be competitive on instances of practical sizes [23, Sec. 8]. In practice, one can instead use a variant of Villard's generic bivariate resultant algorithm [38], hereafter named Algorithm BivariateResultant, and detailed in Algorithm 3 in Appendix A. This leads to the following proposition, whose proof is also given in Appendix A.

Proposition 7. *Let $P, Q \in \mathbb{F}[x, y]$ have total degree d_P, d_Q with $d_P \geq d_Q > 0$, and assume that y^{d_Q} belongs to Q. Algorithm BivariateResultant uses*

$$O\left(\mathsf{M}(d_P^2) + d_P d_Q^{2-1/\omega} \log(d_P)^2 \log\log(d_P)\right)$$

operations in $\mathbb{F}$, and the memory used during the run of the algorithm does not exceed $\tilde{O}(d_P^2 + d_P d_Q^{1+1/\omega})$ elements from $\mathbb{F}$. For generic P and Q of total degrees d_P and d_Q, with $d_P \geq d_Q > 0$, the output polynomial is $R = \mathrm{Res}_y(P, Q)$.

Algorithm BivariateResultant brings two novelties compared to [38]: first, it assumes that P and Q are generic in their total degree, while [38] only tackles the case of polynomials generic in their partial degrees. Second, an initial reduction of P by Q is added to the algorithm, costing $O(\mathsf{M}(d_P^2))$, which leads to a similar kind of unbalanced complexity as in [38].

4 New CICO-1 and CICO-2 attacks based on resultants

In this section, we propose new generic approaches for solving CICO-1 and CICO-2 on AO permutations based on α-inversions.

4.1 A first CICO-2 attack

Here, we describe an attack to solve CICO-2 on AO permutations based on α-inversions. Similarly to the attack from [5], our attack is composed of three steps: `sysGen`, `iteRes` and `bivSol` (the equivalent of `uniSol` in [5]). Our attack strategy can be summarized as follows.

1. **(`sysGen`)** Start from a state comprising 2 variables x_1, x_2, and evaluate the permutation on polynomials. When encountering an α-inversion, create a variable z_i representing its output, add a corresponding equation in the system, and continue the evaluation of the permutation modulo the ideal formed by these equations. These equations yield the following quasi-triangular ideal:
$$I = \langle z_i^\alpha - P_i(x_1, x_2, z_1, \ldots, z_{i-1}) \mid 1 \leq i \leq n \rangle \ .$$
The two output coordinates fixed to 0 by the CICO-2 problem provide equations $Q_1, Q_2 \in R_{2,n}$ where n is the number of α-inversions.
2. **(`iteRes`)** Eliminate the $\mathbf{z}$-variables with iterated resultants by computing $Q_i^{(\mathrm{red})} = \texttt{zElim}(Q_i, I) \in \mathbb{F}[x_1, x_2]$ for $i = 1, 2$.

3. (`bivSol`) Compute $\mathrm{Res}_{x_1}(Q_1^{(\mathrm{red})}, Q_2^{(\mathrm{red})}) \in \mathbb{F}[x_2]$, and compute its root with univariate root finding to get candidate values for x_2. Using these candidates, recover x_1.

Complexity analysis. We let $d_Q = \max(\mathrm{wdeg}(Q_1), \mathrm{wdeg}(Q_2))$. `sysGen` is composed of multivariate polynomial multiplications in $R_{k,n}/I$. The last multiplication dominates the complexity, costing $\tilde{O}((2\alpha-1)^n d_Q^2)$ (cf. Proposition 3). From Proposition 6, `iteRes` costs $\tilde{O}(\alpha^2 \cdot \alpha^{2n} \cdot d_Q^2)$. Since $\mathrm{wdeg}\left(Q_i^{(\mathrm{red})}\right) \leq \alpha^n d_Q$ for $i = 1, 2$, `bivSol` has a complexity of

$$\tilde{O}\left((\alpha^n \cdot d_Q)^{3-1/\omega}\right) ,$$

when using Algorithm BivariateResultant, and

$$O\left((\alpha^n \cdot d_Q)^{2+\epsilon}\right) ,$$

when using the algorithm from van der Hoeven and Lecerf. Thus, `bivSol` is likely to become the bottleneck asymptotically.

Remark. While the ideal degree of the system output by `sysGen` is bounded by $\alpha^n d_Q^2$ (bound reached for CheapLunch systems [3]), `iteRes` increases the ideal degree by a factor of α^n: the final bivariate system has an ideal degree bounded by $\alpha^{2n} d_Q^2$ (bound reached in practice). However, for some values of d_Q and α, this leads to a more competitive complexity than traditional Gröbner basis methods thanks to efficient bivariate solving techniques.

4.2 A CICO-1 attack with SFTM modeling

In this section, we describe our own version of the resultant-based Start-from-the-middle (STFM) attack. Our approach mostly follows the one from [40], with an improved reduction procedure and improved bivariate solving. Let us consider a permutation $\Pi : \mathbb{F}_q^t \mapsto \mathbb{F}_q^t$ and decompose it into two parts: $\Pi = \Pi_1 \circ \Pi_0^{-1}$, as depicted in Figure 1. We call the state after the application of Π_0^{-1} the *middle state*. In order for the SFTM modeling to be applicable, it is necessary that Π_0's high-degree operations are only α-inversions. This is possible if the inverse round function of Π can be expressed as α-inversions and low degree operations, which is the case for `Anemoi` and *Rescue*, but not for Griffin.

In our SFTM attack, we create new variables after each α-inversion of Π_0 and of Π_1. We denote $z_{\mathrm{in},i}$ (resp. $z_{\mathrm{out},i}$) the new variables introduced for each α-inversion of Π_0 (resp. Π_1), and denote $R_{2,n}^{\mathrm{in}} = \mathbb{F}[x_1, x_2, z_{\mathrm{in},1}, \ldots, z_{\mathrm{in},n}]$ and $R_{2,m}^{\mathrm{out}} = \mathbb{F}[x_1, x_2, z_{\mathrm{out},1}, \ldots, z_{\mathrm{out},m}]$. To solve the CICO-1 problem on Π, our SFTM attack is composed of 3 distinct steps:

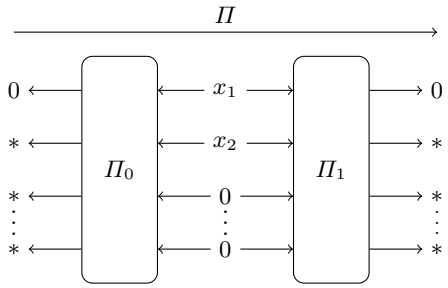


Fig. 1: A permutation $\Pi = \Pi_1 \circ \Pi_0^{-1}$ and its middle state $(x_1, x_2, 0 \ldots, 0)$.

1. (`sysGen`) Evaluate Π_0 (resp. Π_1) on the middle state intepreted as a polynomial, by adding intermediate variables $z_{\text{in},i}$ (resp. $z_{\text{out},i}$) and equations creating a quasi-triangular ideal $I_{\text{in}} = \langle z_{\text{in},i}^{\alpha} - P_{\text{in},i} \mid 1 \leq i \leq n \rangle \subset R_{2,n}^{\text{in}}$ (resp. $I_{\text{out}} \subset R_{2,m}^{\text{out}}$). The first element of the output of Π_0 (resp. Π_1) is a reduced polynomial $Q_{\text{in}} \in R_{2,n}^{\text{in}}/I_{\text{in}}$ (resp. $Q_{\text{out}} \in R_{2,m}^{\text{out}}/I_{\text{out}}$) which should be equal to 0 due to the CICO-1 constraint. This way, a solution to the system

$$\begin{cases} Q_{\text{in}}(x_1, x_2, z_{\text{in},1}, \ldots, z_{\text{in},n}) = 0 \\ Q_{\text{out}}(x_1, x_2, z_{\text{out},1}, \ldots, z_{\text{out},m}) = 0 \\ z_{\text{in},i}^{\alpha} = P_{\text{in},i}(x_1, x_2, z_{\text{in},1}, \ldots, z_{\text{in},i-1}) \text{ for } 1 \leq i \leq n \\ z_{\text{out},i}^{\alpha} = P_{\text{out},i}(x_1, x_2, z_{\text{out},1}, \ldots, z_{\text{out},i-1}) \text{ for } 1 \leq i \leq m \ . \end{cases}$$

yields a solution to the CICO-1 problem.

2. (`iteRes`) Eliminate the variables $\mathbf{z}_{\text{in}}$ through:

$$Q_{\text{in}}^{(\text{red})}(x_1, x_2) = \texttt{zElim}(Q_{\text{in}}, I_{\text{in}}) \ ,$$

and the variables $\mathbf{z}_{\text{out}}$ through:

$$Q_{\text{out}}^{(\text{red})}(x_1, x_2) = \texttt{zElim}(Q_{\text{out}}, I_{\text{out}}) \ .$$

3. (`bivSol`) Compute the solutions in $\mathbb{F}_q^2$ of the polynomial system

$$Q_{\text{in}}^{(\text{red})}(x_1, x_2) = Q_{\text{out}}^{(\text{red})}(x_1, x_2) = 0 \ .$$

A solution to this system leads to a middle state $(x_1, x_2, 0, \ldots, 0)$ which verifies CICO-1 on Π, i.e. such that $(\Pi_0(x_1, x_2, 0, \ldots, 0), \Pi_1(x_1, x_2, 0, \ldots, 0))$ is a correct CICO-1 pair.

Remark. In our attack, depending on the algorithm used for `bivSol`, the optimal decomposition choice for Π can be to voluntarily choose Π_0 and Π_1 so that the total degrees of $Q_{\text{in}}^{(\text{red})}$ and $Q_{\text{out}}^{(\text{red})}$ are unbalanced. More details are provided in the next subsection.

Complexity analysis. Using the results of Section 3, we are able to bound the respective total degrees d_{in} and d_{out} of $Q_{\text{in}}^{(\text{red})}$ and $Q_{\text{out}}^{(\text{red})}$, given the number of variables introduced in input n, the number of variables in the output m, and the degrees of Q_{in} and Q_{out}. These bounds can be used to estimate the complexity of `sysGen`, `iteRes` and `bivSol`.

The cost of `sysGen` is dominated by the cost of the last multiplications to generate Q_{in} and Q_{out}. Following following Proposition 3, they are of complexity respectively

$$O\left(n\,\mathsf{M}\big((2\alpha-1)^n \cdot \text{wdeg}(Q_{\text{in}})^2\big)\right) \text{ and } O\left(m\,\mathsf{M}\big((2\alpha-1)^m \cdot \text{wdeg}(Q_{\text{out}})^2\big)\right) .$$

The cost of `iteRes` can be bounded with the complexity estimates of Proposition 6:

$$\tilde{O}\left((\alpha^{2n+2} \cdot \text{wdeg}(Q_{\text{in}}))^2 + (\alpha^{2m+2} \cdot \text{wdeg}(Q_{\text{out}}))^2\right) .$$

Moreover, the polynomial $Q_{\text{in}}^{(\text{red})}$ has total degree $d_{\text{in}} \leq \alpha^n \cdot \text{wdeg}(Q_{\text{in}})$, and $Q_{\text{out}}^{(\text{red})}$ has degree $d_{\text{out}} \leq \alpha^m \cdot \text{wdeg}(Q_{\text{out}})$.

Following the analysis of Section 3.3, using a theoretical bivariate resultant algorithm based on Kedlaya-Umans, the time complexity of `bivSol` is in $O(\max(d_{\text{in}}, d_{\text{out}})^{2+\epsilon} \log(q)^{1+o(1)})$ for any $\epsilon > 0$. In practice, following Proposition 7, using our adaptation Algorithm BivariateResultant of Villard's algorithm, the complexity of `bivSol` is

$$\tilde{O}\left(d_{\text{in}}^2 + d_{\text{out}}^{2-\frac{1}{\omega}} d_{\text{in}}\right) ,$$

where ω is the exponent of matrix multiplication. Computing the roots of the resultant takes $\tilde{O}(d_{\text{in}} d_{\text{out}})$, which is negligible compared to the bivariate resultant computation.

Remark. Unlike the CICO-2 attack presented in the previous section, `iteRes` is not expected to increase the ideal degree. Indeed, the ideal degree of the system after `sysGen` is bounded by $\alpha^{n+m}\text{wdeg}(Q_{\text{in}})\text{wdeg}(Q_{\text{out}})$, and by $d_{\text{in}} d_{\text{out}} \leq \alpha^{n+m}\text{wdeg}(Q_{\text{in}})\text{wdeg}(Q_{\text{out}})$ after `iteRes`. The two bounds are reached in practice.

Complexity of the attack and decomposition of Π. We now assume that the round function of Π and its inverse have the same structure, degree, and the same number of α-inversions. This is the case for both Anemoi and Rescue. In the "theoretical setting", where a Kedlaya-Umans-based algorithm is used for `bivSol`, a decomposition yielding $d_{\text{in}} \approx d_{\text{out}}$ minimizes the complexity of the attack. Indeed, the algorithms in [24,39] have an expected time depending only on $\max(d_{\text{in}}, d_{\text{out}})^2$. The optimal decomposition of $2r$-round Π therefore consists in taking Π_0 as the inverse of the first r rounds, and Π_1 as the last r rounds. When the number of rounds is odd, the attacker can split the middle round to

minimize $\max(d_{\text{in}}, d_{\text{out}})$. Assuming that we achieve $d_{\text{in}} = d_{\text{out}}$ and $d_{\text{in}} d_{\text{out}} = D_I$, the attack has a complexity of $O(D_I^{1+\epsilon})$ for any $\epsilon > 0$.

In the "practical setting", we use Algorithm BivariateResultant which has complexity

$$\tilde{O}\left(d_{\text{in}}^2 + d_{\text{out}}^{2-\frac{1}{\omega}} d_{\text{in}}\right) .$$

Considering that $d_{\text{in}} d_{\text{out}} = D_I$ is constant, the optimal values for d_{in} and d_{out} are achieved for unbalanced d_{in} and d_{out}, more precisely

$$d_{\text{in}} = D_I^{\frac{2\omega-1}{3\omega-1}} , \qquad d_{\text{out}} = D_I^{\frac{\omega}{3\omega-1}} .$$

One can check that both complexity terms are equal in that case. Assuming that the permutation can be separated in two parts Π_0 and Π_1 yielding such d_{in} and d_{out}, this leads to an attack in complexity:

$$\tilde{O}\left(D_I^{\frac{4\omega-2}{3\omega-1}}\right).$$

This gives a complexity of $\tilde{O}(D_I^{\frac{5}{4}})$ for $\omega = 3$, and $\tilde{O}(D_I^{\frac{6}{5}})$ for $\omega = 2$.

Therefore, for Anemoi and Rescue, taking $\omega = 3$, and if their number of rounds r is divisible by 8, the optimal decomposition is to take Π_0 as the inverse of the first $5r/8$ rounds, and Π_1 as the last $3r/8$ rounds.

Comparison with the improved resultant attack. The improved resultant attack [5] has a complexity of $\tilde{O}(D_I \left(\frac{2\alpha-1}{\alpha}\right)^n)$ where n is the number of α-inversions of the cipher. Our attack has a theoretical complexity bound $O(D_I^{1+\epsilon})$ which provides a direct improvement. Now focusing on the practical setting, let us assume that the logarithmic factors of the attack in [5] are similar to those of Algorithm BivariateResultant and compare the asymptotic behavior of both attacks. One can see that our practical attack has a better complexity when

$$D_I \left(\frac{2\alpha-1}{\alpha}\right)^n > D_I^{\frac{4\omega-2}{3\omega-1}},$$

where n is the number of α-inversions of the permutation. As shown in the next section, this is the case on both Anemoi and Rescue-Prime with $\alpha \leq 7$, even in the worst-case $\omega = 3$.

Time-memory trade-off. In the improved resultant attack, the time complexity is of the same order of magnitude as the memory complexity. This implies that the memory rapidly becomes the bottleneck of the attack. In our case, `iteRes` has a memory complexity of $\tilde{O}(\max(d_{\text{in}}^2, d_{\text{out}}^2))$ which is equal to their time complexity, but `bivSol` has a small gap between its memory and time complexities. In particular, as shown in the proof of Proposition 7, its memory complexity is $\tilde{O}(d_{\text{in}}^2 + m d_{\text{in}} d_{\text{out}}) = \tilde{O}((d_{\text{in}}/d_{\text{out}} + m) D_I)$, where $d_{\text{in}} \geq d_{\text{out}}$ and m

is a parameter that is chosen during the algorithm. If one wishes to optimize the memory complexity (which may be the bottleneck in practice), then one could play with the unbalancedness of d_{in} and d_{out} to keep $d_{\text{in}}/d_{\text{out}}$ on the smaller end, and take $m \approx d_{\text{in}}/d_{\text{out}}$. In that case, the time complexity is impacted, and becomes $\tilde{O}(d_{\text{in}}^2 + (d_{\text{out}} + m^\omega)D_I/m)$, which is $\tilde{O}(d_{\text{in}}^2 + d_{\text{out}}^3 + d_{\text{in}}^\omega d_{\text{out}}^{2-\omega})$. In the extreme case $d_{\text{in}} \approx d_{\text{out}} \approx D_I^{1/2}$ and $m \in O(1)$, this leads to a memory complexity of $\tilde{O}(D_I)$, but the time complexity deteriorates to $\tilde{O}(d_{\text{in}}^2 + d_{\text{out}}^2 d_{\text{in}}) = \tilde{O}(D_I^{1.5})$. (This is essentially the time complexity of the standard bivariate resultant via evaluation-interpolation, with an added first stage of division of P by Q, see Appendix A.1.)

5 Applications

In this section, we estimate the complexity of our attack against the permutations of *Rescue*, `Anemoi`, and Griffin. We apply the SFTM attack for CICO-1 problems on *Rescue* with $t = 3$ and `Anemoi` with $t = 2$. Then, we apply our CICO-2 attack to `Anemoi` with $t = 4$, and Griffin for all $t \geq 4$.

Our complexity estimates are backed up by a practical implementation, which serves both as a way to confirm the polynomial's degree estimates and the overall attack complexity. Our attacks on full-round instances of *Rescue*, `Anemoi` and Griffin are compared to the best attacks of the literature in Table 1.

5.1 Complexity estimate of the attacks

For numerical applications which we compare to previous attacks, we include the logarithmic factors to the complexity formulas. We give here complexity bounds involving $M(d)$, the complexity to multiply polynomials of degree d, which can be replaced by $d\log(d)\log(\log(d))$ to get the logarithmic factors.

First, we recall that the multiplication in $R_{k,n}/I_n$ can be done in $O(n \cdot \mathsf{M}((2\alpha-1)^n(d_P+d_Q)^k))$ following Proposition 3. We now give big-O bounds for Proposition 5 and Proposition 6, which we prove in Appendix B. The bound of Proposition 5 can be written as:

$$O\left(j \cdot M\left((2\alpha-1)^{j+1}\text{wdeg}(Q)^k\right)\right) ,$$

while the bound of Proposition 6 can be expressed as:

$$O\left(M\left(\alpha^2(\alpha^n \cdot \text{wdeg}(Q))^k\right)\right) .$$

From these bounds, we can now straightforwardly compute some big-O bounds for the attacks. Let d_P and d_Q the degrees of the polynomials in the final bivariate resultant step in our attacks. Namely $d_P = \alpha^n \text{wdeg}(Q_{\text{in}})$ and $d_Q = \alpha^m \text{wdeg}(Q_{\text{out}})$ for the SFTM CICO-1 attack, and $d_P = \alpha^n \text{wdeg}(Q_1)$ and $d_Q = \alpha^n \text{wdeg}(Q_2)$ for the CICO-2 attack.

For both the CICO-1 and CICO-2 attacks, the asymptotic complexity of `sysGen` is :

$$\begin{aligned} &\text{for CICO-1:} \quad O\left(nM((2\alpha-1)^{n+2}d_P^2/\alpha^{2n}) + mM((2\alpha-1)^{m+2}d_Q^2/\alpha^{2m})\right) \ , \\ &\text{for CICO-2:} \quad O\left(nM((2\alpha-1)^{n+2}d_P^2/\alpha^{2n})\right) \ . \end{aligned}$$

The complexity of `iteRes` is given by:

$$\begin{aligned} &\text{for CICO-1:} \qquad O\left(M((2\alpha-1)^2 d_P^2) + M((2\alpha-1)^2 d_Q^2)\right), \\ &\text{for CICO-2:} \qquad O\left(M((2\alpha-1)^2 d_P^2)\right). \end{aligned}$$

The complexity of `bivSol` is given by Proposition 7:

$$O\left(\mathsf{M}(d_P^2) + d_P d_Q^{2-1/\omega} \log(d_P)^2 \log\log(d_P)\right) \ ,$$

where ω is the exponent of matrix multiplication.

5.2 Application to *Rescue*

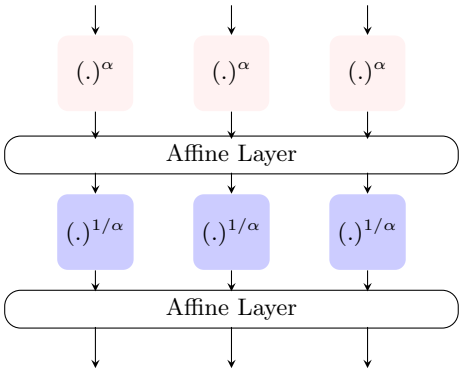


Fig. 2: One round of Rescue–Prime with $t = 3$ branches.

The Rescue–Prime [34] hash function allows many choices for α and the number of branches t. Here focus on $t = 3$.

Round function. The round function of the i-th round R_i, described in Figure 2, is composed of an application of a layer of S-boxes $x \mapsto x^\alpha$, an affine layer corresponding to a multiplication by a MDS matrix $\mathcal{M}$ and a constant addition, a layer of inverse S-boxes $x \mapsto x^{1/\alpha}$, and a second affine layer corresponding to multiplication by $\mathcal{M}$ and a constant addition. The full r-round Rescue Prime permutation is expressed as $\text{Rescue}_r = R_r \circ \cdots \circ R_1$.

Cipher	t	α	Type	128-bit Rounds	128-bit T	256-bit Rounds	256-bit T	Ref
Rescue CICO-1	3	3	Res.	18	137	35	257	[5]
			SFTM-Res.	$11+7$	**112**	$21+14$	**215**	New
	3	5	Res.	14	145	26	262	[5]
			SFTM-Res.	$8+6$	128	$16+10$	**232**	New
	3	7	Res.	12	145	23	269	[5]
			SFTM-Res.	$7+5$	131	$14+9$	**247**	New
`Anemoi` (odd) CICO-1	2	3	Res.	21	80	37	130	[5]
			SFTM-Res.	$12+9$	**70**	$22+15$	**117**	New
	2	5	Res.	21	96	37	157	[5]
			SFTM-Res.	$12+9$	**84**	$22+15$	**141**	New
	2	7	Res.	20	103	36	170	[5]
			SFTM-Res.	$11+9$	**90**	$21+15$	**154**	New
`Anemoi` (even) CICO-1	2	-	Res.	24	110	44	190	[5]
			SFTM-Res.	$14+10$	**105**	$26+18$	**185**	New
`Anemoi` (odd) CICO-2	4	3	GB	14	171	22	269	[13]
			Res.	14	153	22	**237**	New
	4	5	GB	14	212	22	335	[13]
			Res.	14	199	22	308	New
	4	7	GB	13	232	21	374	[13]
			Res.	13	215	21	342	New
`Anemoi` (odd) Jive	4	3	Res.	14	134	22	**209**	New
	4	5	Res.	14	180	22	285	New
	4	7	Res.	13	196	21	319	New
`Anemoi` (even) CICO-2	4	-	GB	15	272	25	451	[12]
			Res.	15	198	25	324	New
`Anemoi-Jive` (even)	4	-	Res.	15	160	25	265	New
Griffin CICO-2	4	3	Res.	15	183	-	-	New
	4	5	Res.	11	177	-	-	New
	4	7	Res.	10	187	-	-	New
	8	3	Res.	11	129	-	-	New
	8	5	Res.	9	135	-	-	New
	8	7	Res.	8	137	-	-	New
	12	3	Res.	10	**108**	-	-	New
	12	5	Res.	9	**125**	-	-	New
	12	7	Res.	8	**125**	-	-	New
	≥ 16	3	Res.	10	**105**	-	-	New
	≥ 16	5	Res.	9	**120**	-	-	New
	≥ 16	7	Res.	8	**119**	-	-	New

Table 1: Time complexity (T) of attacks against full-round 128-bit and 256-bit variants of *Rescue*, `Anemoi` and Griffin, given as the $\log_2$ of the number of operations in $\mathbb{F}_q$, with $\omega = 2.81$, considering the practical Villard-based variant of our attack.

Complexity analysis of the attack. Since there is no linear layer at the input of the permutation, we use the technique of [7] to bypass the first non-linear layer. We therefore consider an instance Π comprised of r rounds such that $r = r_P + r_Q$ with Π_0^{-1} being the first r_P rounds without the first S-box layer, and Π_1 the last r_Q rounds.

Following the `sysGen` procedure described in Section 4.2, we get variables $z_{\text{in},i}, z_{\text{out},i}$ of special form:

$$\begin{cases} z^{\alpha}_{\text{in},j} = P_{\text{in},j}(x_1, x_2)\ ,\ 1 \leq j \leq t. \\ z^{\alpha}_{\text{in},t(i+1)+j} = P_{\text{in},t(i+1)+j}(z_{\text{in},t\cdot i+1}, \ldots, z_{\text{in},t(i+1)-1}),\ 1 \leq i < r_P - 1,\ 1 \leq j \leq t. \\ z^{\alpha}_{\text{out},j} = P_{\text{out},j}(x_1, x_2)\ ,\ 1 \leq j \leq t. \\ z^{\alpha}_{\text{out},t(i+1)+j} = P_{\text{out},t(i+1)+j}(z_{\text{out},t\cdot i+1}, \ldots, z_{\text{out},t(i+1)-1}),\ 1 \leq i < r_Q,\ 1 \leq j \leq t. \end{cases}$$

where all $P_{\text{in},i}, P_{\text{out},i}$ have degree α. Thus, the $\mathbf{z}_{\text{out}}$ and $\mathbf{z}_{\text{in}}$ variables have weight 1. Additionally, the polynomial $Q_{\text{out}}(z_{\text{out},t\cdot(r_Q-1)+1}, \ldots, z_{\text{out},t\cdot r_Q})$ is linear, and the polynomial $Q_{\text{in}}(z_{\text{in},t\cdot(r_P-2)+1}, \ldots, z_{\text{in},t\cdot(r_P-1)})$ has degree α. Thus, the polynomial $Q^{(\text{red})}_{\text{out}}$ will have degree α^{tr_Q}, and the polynomial $Q^{(\text{red})}_{\text{in}}$ degree $\alpha^{t(r_P-1)+1}$.

This gives an ideal degree $D_I = \alpha^{tr_Q+t(r_P-1)+1} = \alpha^{t(r-1)+1}$ for `bivSol`, which is also the ideal degree of the improved resultant attack [40] when only one non-linear layer is bypassed. This gives an attack in complexity:

$$\tilde{O}\left(\alpha^{\frac{(4\omega-2)(t(r-1)+1)}{3\omega-1}}\right).$$

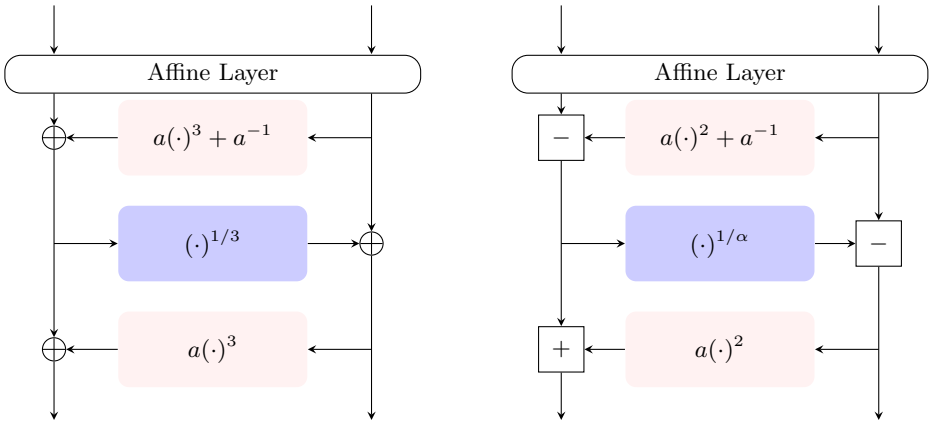
For small values of α, this improves upon the bound of the improved resultant attack, which is $\tilde{O}((2\alpha-1)^{t(r-1)+1})$. Notably, our attack breaks the security claims of the designers of *Rescue* for several 128-bit and 256-bit security variants.

5.3 Application to `Anemoi`

`Anemoi` [12] is a family of ZK-friendly permutations designed to be used either in sponge or in compression mode. `Anemoi` is defined for odd and even characteristics, and for various state sizes, but we only focus on the case of 2 branches for the CICO-1 problem and 4 branches for the CICO-2 problem. We also provide a preimage attack on a 4-to-2 compression function based on a permutation with 4 branches.

Round function. The round function of the i-th round, described in Figure 3, is composed of an affine layer (constant addition $\mathcal{S}_{c_i}$ and matrix $\mathcal{M}$), and a non-linear layer $\mathcal{H}$. The specifications of the non-linear layer of `Anemoi` differs depending on if the base field is of odd or even characteristic. We denote respectively $\mathcal{H}_{o,\alpha}$ and $\mathcal{H}_e$ the non-linear layers of odd and even characteristic `Anemoi`. Those are defined as three-round Feistel networks applied in parallel on pairs of branches.

Keeping the notation $\mathcal{H}$ when the base field characteristic is not specified, the i-th round function can be expressed as: $R_i = \mathcal{H} \circ \mathcal{M} \circ \mathcal{S}_{c_i}$. The full r-round `Anemoi` can then be expressed as $\text{Anemoi}_r = \mathcal{M} \circ R_r \circ \cdots \circ R_1$.



(a) One round of Anemoi with even characteristic.

(b) One round of Anemoi with odd characteristic.

Fig. 3: The round function of Anemoi (case $t = 2$).

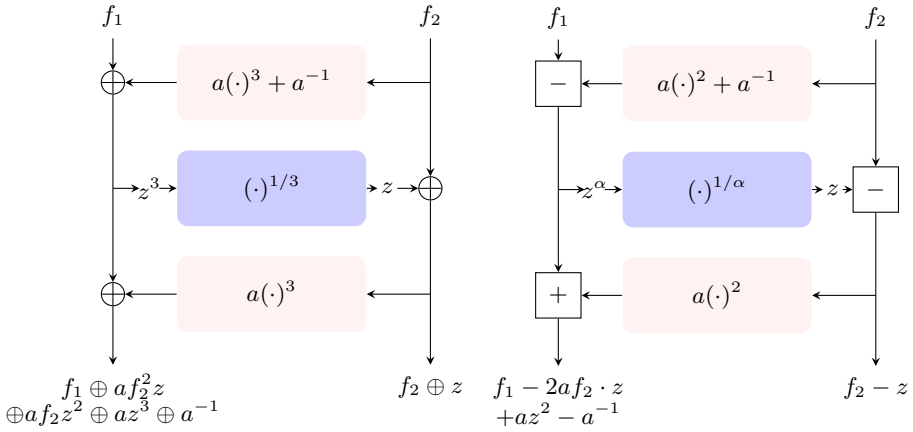
Modeling of a single non-linear layer (odd case). We let an S-box with input f_1, f_2 that are polynomial in x_1, x_2, and in the previous variables z_i that were introduced. The first Feistel step gives a state of the form $(f_1 - a \cdot f_2^2 - a^{-1}, f_2)$. Then we introduce a new variable z as follows: $z^\alpha = f_1 - a \cdot f_2^2 - a^{-1}$. This way, the new variable z gets weight $\leq 2\max(\text{wdeg}(f_1), \text{wdeg}(f_2))/\alpha$. Finally, after the third Feistel step, we get a state whose terms of highest weight is either f_1 or $f_2 \cdot z$ (see Figure 4b). This way, the maximal weight of the state is upper bounded by:

$$\begin{aligned}\max(\text{wdeg}(f_1), \text{wdeg}(f_2 \cdot z)) &\leq \max(\text{wdeg}(f_1), (1 + 2/\alpha)\text{wdeg}(f_2)) \\ &\leq (1 + 2/\alpha)\max(\text{wdeg}(f_1), \text{wdeg}(f_2)) \ .\end{aligned}$$

Each round thus increases the weight of the equations by a degree $(1 + 2/\alpha)$, and adds a new $\mathbf{z}$-variable for each S-box, that is adds $t/2$ $\mathbf{z}$-variables.

The case of the inverse of `Anemoi` S-boxes is very similar, and also leads to an additional variable and a state whose weight is multiplied with $(1 + 2/\alpha)$.

Modeling of a single non-linear layer (even case) In this case, we get a state of the form $(f_1 \oplus a \cdot f_2^3 \oplus a^{-1}, f_2)$ after the first Feistel step, and introduce the new variable z as $z^3 = f_1 \oplus a \cdot f_2^3 \oplus a^{-1}$. This way, the weight of the new variable z equals $\text{wdeg}(f_2)$, and the final state is of degree 3 in z, f_1, f_2. Thus, we introduce a new variable for each S-box, and multiply the weighted degree of the state by 3 for each round.



(a) Modeling of `Anemoi` focused on a single Flystel step (even characteristic).

(b) Modeling of `Anemoi` focused on a single Flystel step (odd characteristic).

Fig. 4: Our modeling of `Anemoi`.

Complexity analysis of the attacks, odd case. We consider an instance comprised of r rounds s.t $r = r_P + r_Q$ with Π_0^{-1} being the first r_P rounds and Π_1 the last r_Q rounds.

CICO-1 attack ($t = 2$) Starting from a middle state of the form (x_1, x_2), we use the results from the previous paragraphs to generate quasi-triangular systems $I_{\text{in}}, I_{\text{out}}$ with r_P $\mathbf{z}_{\text{in}}$-variables and r_Q $\mathbf{z}_{\text{out}}$-variables. As the weighted degree of the state grows with $(1 + 2/\alpha)$ at each round, we get that $\text{wdeg}(Q_{\text{in}}) = (1 + 2/\alpha)^{r_P}$ and $\text{wdeg}(Q_{\text{out}}) = (1 + 2/\alpha)^{r_Q}$.

Using the complexity estimates and decomposition of $r = r_P + r_Q$ from Section 4.2, we get an attack complexity of:

$$\tilde{O}\left((\alpha + 2)^{\frac{(4\omega-2)r}{3\omega-1}}\right).$$

For small α's, this asymptotic complexity improves upon the improved resultant attack of [40], which is $\tilde{O}((\alpha + 2)^r(2 - 1/\alpha)^r)$.

CICO-2 attack ($t = 4$) Starting from an input state $(x_1, x_2, 0, 0)$, we introduce the variables z_{2i-1}, z_{2i} in the two S-boxes at round i. The degree of the state grows with $(1 + 2/\alpha)$ at each round, giving $2r$ $\mathbf{z}$-variables, and equations Q_1, Q_2 of weight $(1+2/\alpha)^r$. Using the complexity estimates from Section 4.1, the attack has complexity:

$$\tilde{O}\left((\alpha(\alpha + 2))^{(3-1/\omega)r}\right).$$

Neglecting for the logarithmic factors, this improves upon the bound $O((\alpha + 2)^{2\omega r})$ from [13] in the practical setting where $\omega = 2.81$. Taking the best known

linear algebra constant, namely $\omega < 2.38$, we still get a better asymptotic complexity than [13] for $\alpha = 3, 5$.

Preimage attack on 4*-to-*2 *Jive mode* We exploit the specifics of the Jive compression mode using a Start-From-The-Middle (SFTM) approach. We start from the $\lceil r/2 \rceil$-th round Flystel, before the application of the $x^{1/\alpha}$. We start with a state of the form:

$$(0, x_1, 0, x_2) \ .$$

This way, the middle $x^{1/\alpha}$ S-box is not activated by those two variables, and the input and output of the Flystel are of degree 2 in x_1, x_2. Then, using a similar method as in the previous paragraph, we can introduce z_i variables in the first $\lceil r/2 \rceil - 1$ rounds in order to get an input of weighted degree at most

$$2 \cdot (1 + 2/\alpha)^{\lceil r/2 \rceil - 1} \ .$$

We also introduce z_i variables in the $r - \lceil r/2 \rceil = \lfloor r/2 \rfloor$ last rounds to get an output of weighted degree at most:

$$2 \cdot (1 + 2/\alpha)^{\lfloor r/2 \rfloor} \ .$$

The Jive mode of operation is defined using a feedforward operation, that is:

$$\mathsf{Jive}[P](\mathbf{x}) = M_{\text{in}} \cdot \mathbf{x} + M_{\text{out}} \cdot P(\mathbf{x}) \ ,$$

where $M_{\text{in}}, M_{\text{out}}$ are 2×4 matrices, and P is the permutation.

Letting Q_1, Q_2 the two equations we obtain when applying that feedforward operation on the input and output states we obtained, we get Q_1, Q_2 of weighted degree at most:

$$\max(2 \cdot (1 + 2/\alpha)^{\lceil r/2 \rceil - 1}, 2 \cdot (1 + 2/\alpha)^{\lfloor r/2 \rfloor}) = 2 \cdot (1 + 2/\alpha)^{\lfloor r/2 \rfloor} \ .$$

Moreover, we had to introduce $n = 2(r-1)$ variables z_i.

Using the previous estimates for Villard algorithm, we get a complexity of:

$$\tilde{O}\left((2\alpha^{r+\lceil r/2 \rceil - 2}(\alpha+2)^{\lfloor r/2 \rfloor})^{3-1/\omega}\right) \ .$$

Note that this attack complexity is lower than the complexity of a CICO-2 attack on the corresponding permutation by a factor close to $(1 + 2/\alpha)^{(3-1/\omega)r/2}$.

Complexity analysis of the attack, even case.

*CICO-*1 *attack (*$t = 2$*)* Similarly to the odd case, we consider an instance comprised of r rounds such that $r = r_P + r_Q$ with Π_0^{-1} being the first r_P rounds and Π_1 the last r_Q rounds. The only difference with odd characteristic case is the weighted degree that gets multiplied by 3 at each round instead of $(1+2/\alpha)$. Using decomposition between r_P and r_Q and complexity estimates from Section 4.2, we get an attack with complexity:

$$\tilde{O}\left(9^{\frac{(4\omega-2)r}{3\omega-1}}\right),$$

which impoves upon the complexity bound of the improved resultant attack of [40], which is $\tilde{O}(9^r(2-1/3)^r)$.

CICO-2 attack ($t = 4$) Similarly as in previous paragraph, we get that Q_1, Q_2 have weight at most 3^r, and we introduce $2r$ $\mathbf{z}$-variables. Thus, we get an attack with complexity:

$$\tilde{O}\left(27^{(3-1/\omega)R}\right) .$$

Applying the complexity estimates from the designers of `Anemoi` for attacks on the CICO-2 problem, we would get attack complexity in $O(R \cdot 81^{R\omega})$. Our attack improves upon those estimates for any value of $\omega \geq 2$.

Preimage attack on 4-to-2 Jive mode The same SFTM modeling as in the odd characteristic case leads to introducing $2(r-1)$ $\mathbf{z}$-variables, and to get equations Q_1, Q_2 of degree at most $3^{\lfloor R/2 \rfloor + 1}$. This leads to an attack with complexity:

$$\tilde{O}\left((3^{2R+\lfloor R/2 \rfloor - 1})^{3-1/\omega}\right) .$$

As in the odd characteristic case, our attack on Jive compression mode is significantly more efficient than our attack on the corresponding CICO problem.

5.4 Application to Griffin

Finally, we provide new attacks on the Griffin CICO-2 problem. While the CICO-1 problem has been treated in [6,5], resulting in full-round breaks for several 128-bits of security instances, larger CICO problems have received less attention. The only work we are aware of in that setting is [3], where the authors provide a modeling of the CICO-k problems, allowing them to derive an upper bound on the degree of the ideal. Note that due to the lack of some regularity assumption, the authors were not able to apply their Cheaplunch framework in the case of Griffin, and thus did not provide complexity estimates for their attacks. In our work, we propose the first attacks on CICO-2 problems on the Griffin permutation.

The Griffin permutation The Griffin round function is the composition of the following steps:

1. A partial S-box step: $(x_1, x_2, x_3, \ldots, x_t) \mapsto (x_1^{1/\alpha}, x_2^{\alpha}, x_3, \ldots, x_t)$.
2. A Horst nonlinear layer: $(x_1, x_2, x_3, x_4, \ldots, x_t) \mapsto (x_1, x_2, x_3 \cdot G_1(x_1, x_2, 0), x_4 \cdot G_2(x_1, x_2, x_3), \ldots, x_t \cdot G_{t-2}(x_1, x_2, x_{t-1}))$, where the G_i are quadratic functions.
3. An affine layer: $\mathbf{x} \mapsto M \cdot \mathbf{x} + c_i$, where c_i is the i-th round constant.

In addition, the Griffin permutation starts with a linear layer. The Griffin round function is represented in Figure 5.

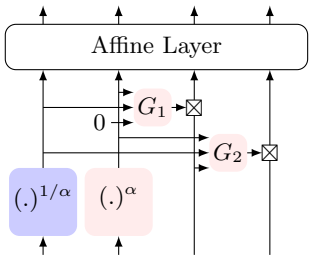


Fig. 5: The Griffin round function in the case $t = 4$.

System generation We start with an input of the form $(x_1, x_2, 0, \ldots, 0)$, and apply the first linear layer. Then, at each round i, we do as follows:

- Let $(f_1^{(i)}(x_1, x_2, z_1, \ldots, z_{i-1}), \ldots, f_t^{(i)}(x_1, x_2, z_1, \ldots, z_{i-1}))$ the state at the input of the i-th round. We introduce the variable z_i as:

$$z_i^\alpha = f_1^{(i)}(x_1, x_2, z_1, \ldots, z_{i-1}) \ ,$$

  that is we set $P_i = f_1^{(i)}$ and add the equation $z_i^\alpha - P_i$ to I.
- The state becomes $(z_i, f_2^{(i)}(x_1, x_2, z_1, \ldots, z_{i-1}), \ldots, f_t^{(i)}(x_1, x_2, z_1, \ldots, z_{i-1}))$.
- Then we apply the S-box $x \mapsto x^\alpha$ to the second coordinate $f_2^{(i)}$, and the Horst nonlinear layer, while performing reductions modulo $I_{i-1} = \langle z_1^\alpha - P_1, \ldots, z_{i-1}^\alpha - P_{i-1} \rangle$. This amounts to applying a function of degree $2\alpha + 1$ to the whole vector.
- Finally, we apply the affine layer to get the $f_j^{(i+1)}$.

After R rounds, we get a state of the form

$$((f_1^{(R+1)}(x_1, x_2, z_1, \ldots, z_R), \ldots, f_t^{(R+1)}(x_1, x_2, z_1, \ldots, z_R))) \ .$$

We set $Q_1 = f_{t-1}^{(R+1)}, Q_2 = f_t^{(R+1)}$.

Using the weights defined as in Section 3, we can easily prove that:

$$\mathrm{wdeg}(Q_1), \mathrm{wdeg}(Q_2) \leq (2\alpha + 1)^R \ .$$

Thus, using the previous complexity analysis with $n = R$ and $d_Q = (2\alpha + 1)^R$, we get an attack with time complexity:

$$\tilde{O}\left((\alpha(2\alpha + 1))^{(3-1/\omega)R}\right) \ .$$

Skipping the first rounds Following [3, Appendix D.2], we can reduce the degree of the equations by skipping the first rounds. Those skipping techniques result in lower values of n and d_Q, and thus in better attacks. We apply those

t	n	d_Q	$\alpha^n \cdot d_Q$
4	R	$(2\alpha+1)^R$	$(\alpha(2\alpha+1))^R$
8	$R-1$	$3\cdot(2\alpha+1)^{R-1}$	$3\cdot(\alpha(2\alpha+1))^{R-1}$
12	$R-2$	$(2\alpha+1)^{R-1}$	$(2\alpha+1)\cdot(\alpha(2\alpha+1))^{R-2}$
≥ 16	$R-2$	$3\cdot(2\alpha+1)^{R-2}$	$3\cdot(\alpha(2\alpha+1))^{R-2}$

Table 2: Round-skipping tricks on Griffin

results to the special case where $k = 2$, and summarize the effect of these skipping tricks on n and d_Q in Table 2.

Applying these results, we get attacks with complexity $\tilde{O}((\alpha^n d_Q)^{3-1/\omega})$ on the Griffin permutation. We applied our attack on several full instances of Griffin with 128 bits of security for several values of t. We see that we get a full round attack as long as $t \geq 12$, for $\alpha = 3, 5, 7$.

5.5 Experimental Attacks

The computational experiments were conducted on a system with a AMD EPYC 9354 processor and 1 TB DDR5 RAM (16 × 64 GB HMCG94AEBRA109N, 4800 MT/s). To ensure as fair a comparison as possible with previous works, all experiments were conducted in a single-threaded setting. However, it is worth noting that many computational steps in our approach could be parallelized to some extent.

From an implementation standpoint, the core computational routines are written in `C++` for performance reasons. The codebase uses the NTL library [36] for univariate polynomial multiplication as it includes an optimized implementation of the small-prime Fast Fourier Transform (as used by [5]), and the PML library [35] to handle generic bivariate polynomials and for the generic resultant's computation using Villard's algorithm. [40] instead uses maple for the core of their experimental attacks. The complete implementation is available on github: github.com/maelhos/resultants-meet-resultant. In all our attacks, we use a 55-bit prime $p =$ `0x64ec6dd0392073`, as in [6] and as in the attack of [40] against Anemoi. Table 3 provides a comparison of our experimental attacks with those the literature.

Remarks. As seen in Table 3, the `bivSol` implementation has an overhead factor compared to our theoretical estimation. This can be the consequence of many unexpected behaviors in the implementation, that we did not further investigate. In order to circumvent this extra factor, we chose to implement different decompositions of the round-reduced permutations (i.e. testing different values of r_P and r_Q for a constant $r_P + r_Q = r$) and put the benchmarks of our attack w.r.t. the different explicit decompositions in Table 3. It should be noted that our implementation uses a matrix multiplication algorithm with $\omega = 3$.

			Overall			iteRes		bivSol		
Cipher	r	Type	T	M	Theor.	T	M	T	M	Ref
Rescue	4	Res.	2.4s	0.1	33	-	-	-	-	[5]
CICO-1	$2+2$	SFTM-Res.	10s	0.1	26	1s	0.1	9s	0.1	New
	5	Res.	6m6s	10.1	41	-	-	-	-	[5]
	$3+2$	SFTM-Res.	24m	9	33	11s	0.4	24m	9	New
	6	Res.	2d4h	570	49	-	-	-	-	[5]
	$3+3$	SFTM-Res.	1d6h	251	40	25m6s	24	1d6h	251	New
	$4+2$	SFTM-Res.	22h17m	295	42	7h56m	295	14h21m	223	New
Anemoi	7	FL	2d	42	-	-	-	-	-	[6]
CICO-1		SFTM-Res.	49m	< 256	40	-	-	-	-	[40]
		Res.	9.5s	0.16	33	-	-	-	-	[5]
	$4+3$	SFTM-Res.	18s	0.16	28	1s	0.09	17s	0.15	New
	$5+2$	SFTM-Res.	36s	1.3	33	24s	1.3	12s	0.7	New
	8	SFTM-Res.	10h	< 256	43	-	-	-	-	[40]
		Res.	1m25s	1.2	37	-	-	-	-	[5]
	$5+3$	SFTM-Res.	2m9s	8.5	33	24s	1.4	1m45s	0.93	New
	9	Res.	13m51s	9.5	40	-	-	-	-	[5]
	$5+4$	SFTM-Res.	35m6s	4.8	34	24s	0.7	34m42s	4.8	New
	$6+3$	SFTM-Res.	25m12s	26.9	38	15m	26.9	10m12s	15.8	New
	10	Res.	2h38m	77	44	-	-	-	-	[5]
	$6+4$	SFTM-Res.	3h22m	28	38	15m	16	3h7m	28	New
	$7+3$	SFTM-Res.	19h17m	381	43	17h29m	381	1h48m	133	New
	11	Res.	1d22h	283	47	-	-	-	-	[5]
	$7+4$	SFTM-Res.	1d9h	432	43	17h18m	432	15h34m	335	New
Anemoi	2	GB	10s	< 48	24	-	-	-	-	[13]
CICO-2		Res.	9s	0.08	26	0.1s	0.02	9s	0.08	New
	3	GB	9m12s	< 48	37	-	-	-	-	[13]
		Res.	4h5m	40	37	36s	1.5	4h4m	40	New
Anemoi	2	Res.	0.1s	0.01	17	< 0.1s	0.01	0.1s	0.01	New
Jive	3	Res.	33s	0.23	27	0.3s	0.02	33s	0.23	New
	4	Res.	12h24m	110	38	1m12s	1.8	12h23m	110	New
Griffin (CICO-2)	4	Res.	59m56s	16	33	4.6s	0.5	59m52s	16	New

Table 3: Summary of practical attacks against round-reduced instances of *Rescue*, Anemoi and Griffin with $\alpha = 3$. For *Rescue* CICO-1, Anemoi CICO-1, Anemoi CICO-2, Anemoi Jive, and Griffin, we used respectively $t = 3, 2, 4, 4, 16$. Time (T) and memory (M) in GigaBytes are provided for the different attacks of the literature, and for each step of our attack. The theoretical complexities (Theor.) of the overall attacks are also given, counted as the $\log_2$ of the number of $\mathbb{F}_q$-operations, with $\omega = 3$.

For attacks on 2 coordinates, we observe that our CICO-2 attacks on Anemoi do not perform as well as the ones from [13]. This can be explained by the fact they used GPU to accelerate some matrix operations. On the other hand, our implementation is completely single-threaded. Some steps of Villard's algorithm can be parallelized, which would lead to much better practical performances of the attack. For the preimage attacks on Anemoi in 4-to-2 Jive mode, we see that we significantly improve upon the attack on the corresponding CICO problem, breaking one more round than what we are able to do for the CICO-2 problem, and being roughly 18 times faster than the best 3-round attack.

Acknowledgements.

The authors would like to thank Luca Campa and Arnab Roy for useful discussions. The authors would like to thank Télécom SudParis for the computational resources provided for the experiments. The work was supported by the European Research Council (ERC) through the grant agreement no. 101041545 "ReSCALE", by the French DGA, and by the Agence nationale de la recherche (ANR) through the projects ANR-23-CE48-0003-01 "CREAM" and ANR-25-CE39-7662 "PQMC".

References

1. Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of symmetric-key primitives for advanced cryptographic protocols. IACR Transactions on Symmetric Cryptology pp. 1–45 (2020)
2. Ashur, T., Kindi, A., Mahzoun, M.: XHash8 and XHash12: Efficient STARK-friendly Hash Functions. Cryptology ePrint Archive, Paper 2023/1045 (2023), https://eprint.iacr.org/2023/1045
3. Bak, A., Bariant, A., Boeuf, A., Briaud, P., Øygarden, M., Phanse, A.: The algebraic cheaplunch: Extending freelunch attacks on arithmetization-oriented primitives beyond CICO-1. IACR Cryptol. ePrint Arch. p. 2040 (2025), https://eprint.iacr.org/2025/2040
4. Bariant, A., Boeuf, A., Briaud, P., Hostettler, M., Øygarden, M., Raddum, H.: Improved resultant attack against arithmetization-oriented primitives. IACR Cryptol. ePrint Arch. p. 259 (2025), https://eprint.iacr.org/2025/259
5. Bariant, A., Boeuf, A., Briaud, P., Hostettler, M., Øygarden, M., Raddum, H.: Improved resultant attack against arithmetization-oriented primitives. In: Tauman Kalai, Y., Kamara, S.F. (eds.) Advances in Cryptology – CRYPTO 2025. pp. 335–367. Springer Nature Switzerland, Cham (2025)
6. Bariant, A., Boeuf, A., Lemoine, A., Manterola Ayala, I., Øygarden, M., Perrin, L., Raddum, H.: The Algebraic FreeLunch: Efficient Gröbner Basis Attacks Against Arithmetization-Oriented Primitives. In: Annual International Cryptology Conference (Crypto). pp. 139–173. Springer (2024)
7. Bariant, A., Bouvier, C., Leurent, G., Perrin, L.: Algebraic Attacks against Some Arithmetization-Oriented Primitives. IACR Transactions on Symmetric Cryptology pp. 73–101 (2022)

8. Beckermann, B.: A reliable method for computing M-Padé approximants on arbitrary staircases. J. Comput. Appl. Math. **40**(1), 19–42 (1992), http://dx.doi.org/10.1016/0377-0427(92)90039-Z
9. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Keccak sponge function family. Tech. rep., Keccak Team (2008), https://keccak.team/files/CSF-0.1.pdf
10. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. In: ECRYPT hash workshop. vol. 2007. Citeseer (2007)
11. Bostan, A., Éric Schost: Polynomial evaluation and interpolation on special sets of points. Journal of Complexity **21**(4), 420–446 (2005). https://doi.org/https://doi.org/10.1016/j.jco.2004.09.009, https://www.sciencedirect.com/science/article/pii/S0885064X05000026, festschrift for the 70th Birthday of Arnold Schonhage
12. Bouvier, C., Briaud, P., Chaidos, P., Perrin, L., Salen, R., Velichkov, V., Willems, D.: New design techniques for efficient arithmetization-oriented hash functions: Anemoi permutations and jive compression mode. In: Annual International Cryptology Conference (Crypto). pp. 507–539. Springer (2023)
13. Campa, L., Roy, A.: Groebner basis cryptanalysis of anemoi. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 303–332. Springer (2025)
14. Cantor, D.G., Kaltofen, E.L.: On fast multiplication of polynomials over arbitrary algebras. Acta Informatica **28**(7), 693–701 (1991). https://doi.org/10.1007/BF01178683, https://doi.org/10.1007/BF01178683
15. Cox, D.A., Little, J., O'Shea, D.: Using Algebraic Geometry. Graduate Texts in Mathematics, Springer, Second edn. (2005)
16. Cox, D.A., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms. Springer International Publishing (2015)
17. von zur Gathen, J., Gerhard, J.: Modern computer algebra. Third edition 2013. Cambridge University Press (1999). https://doi.org/10.1017/CBO9781139856065
18. Giorgi, P., Jeannerod, C.P., Villard, G.: On the complexity of polynomial matrix computations. In: Proceedings ISSAC 2003. pp. 135–142. ACM Press (2003). https://doi.org/10.1145/860854.860889
19. Grassi, L., Hao, Y., Rechberger, C., Schofnegger, M., Walch, R., Wang, Q.: Horst Meets Fluid-SPN: Griffin for Zero-Knowledge Applications. In: Annual International Cryptology Conference (Crypto). pp. 573–606. Springer (2023)
20. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 519–535 (2021)
21. Harvey, D., van der Hoeven, J.: Polynomial multiplication over finite fields in time $o(n \log n)$. J. ACM **69**(2) (Mar 2022). https://doi.org/10.1145/3505584, https://doi.org/10.1145/3505584
22. Harvey, D., van der Hoeven, J., Lecerf, G.: Faster polynomial multiplication over finite fields. J. ACM **63**(6) (Jan 2017). https://doi.org/10.1145/3005344, https://doi.org/10.1145/3005344
23. van der Hoeven, J., Lecerf, G.: Fast multivariate multi-point evaluation revisited. Journal of Complexity **56**, 101405 (2020). https://doi.org/10.1016/j.jco.2019.04.001
24. van der Hoeven, J., Lecerf, G.: Fast computation of generic bivariate resultants. J. Complex. **62**, 101499 (2021). https://doi.org/10.1016/J.JCO.2020.101499, https://doi.org/10.1016/j.jco.2020.101499

25. Hyun, S.G., Neiger, V., Schost, E.: Implementations of efficient univariate polynomial matrix algorithms and application to bivariate resultants. In: Proceedings ISSAC 2019. pp. 235—-242. ISSAC '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3326229.3326272
26. Kailath, T., Kung, S.Y., Morf, M.: Displacement ranks of matrices and linear equations. Journal of Mathematical Analysis and Applications **68**(2), 395–407 (1979). https://doi.org/https://doi.org/10.1016/0022-247X(79)90124-0, https://www.sciencedirect.com/science/article/pii/0022247X79901240
27. Kedlaya, K.S., Umans, C.: Fast polynomial factorization and modular composition. SIAM J. Comput. **40**(6), 1767–1802 (2011). https://doi.org/10.1137/08073408X, https://doi.org/10.1137/08073408X
28. Koschatko, K., Lüftenegger, R., Rechberger, C.: Exploring the six worlds of gröbner basis cryptanalysis: Application to anemoi. IACR Trans. Symmetric Cryptol. **2024**(4), 138–190 (2024). https://doi.org/10.46586/TOSC.V2024.I4.138-190, https://doi.org/10.46586/tosc.v2024.i4.138-190
29. Mahler, K.: Perfect systems. Composit. Math. **19**(2), 95–166 (1968)
30. Neiger, V., Pernet, C.: Deterministic computation of the characteristic polynomial in the time of matrix multiplication. Journal of Complexity **67**, 101572 (2021). https://doi.org/https://doi.org/10.1016/j.jco.2021.101572, https://www.sciencedirect.com/science/article/pii/S0885064X21000273
31. Pan, V.Y.: Structured matrices and polynomials: unified superfast algorithms. Springer-Verlag, Berlin, Heidelberg (2001)
32. Pernet, C., Signargout, H., Villard, G.: High-order lifting for polynomial sylvester matrices. Journal of Complexity **80**, 101803 (2024). https://doi.org/https://doi.org/10.1016/j.jco.2023.101803, https://www.sciencedirect.com/science/article/pii/S0885064X23000729
33. Roy, A., Steiner, M.J., Trevisani, S.: Arion: Arithmetization-oriented permutation and hashing from generalized triangular dynamical systems. arXiv preprint arXiv:2303.04639 (2023)
34. Szepieniec, A., Ashur, T., Dhooghe, S.: Rescue-prime: a standard specification (SoK). Cryptology ePrint Archive (2020)
35. The PML team: PML: Polynomial Matrix Library (2023), version 0.3, https://github.com/vneiger/pml
36. V. Shoup, et al.: NTL: A Library for Doing Number Theory. https://libntl.org/
37. Van Barel, M., Bultheel, A.: A general module theoretic framework for vector M-Padé and matrix rational interpolation. Numer. Algorithms **3**, 451–462 (1992), http://dx.doi.org/10.1007/BF02141952
38. Villard, G.: On computing the resultant of generic bivariate polynomials. In: Kauers, M., Ovchinnikov, A., Schost, É. (eds.) Proceedings of the 2018 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2018, New York, NY, USA, July 16-19, 2018. pp. 391–398. ACM (2018). https://doi.org/10.1145/3208976.3209020, https://doi.org/10.1145/3208976.3209020
39. Villard, G.: Elimination ideal and bivariate resultant over finite fields. In: Dickenstein, A., Tsigaridas, E.P., Jeronimo, G. (eds.) Proceedings of the 2023 International Symposium on Symbolic and Algebraic Computation, ISSAC 2023, Troms Ø, Norway, July 24-27, 2023. pp. 526–534. ACM (2023). https://doi.org/10.1145/3597066.3597100, https://doi.org/10.1145/3597066.3597100
40. Yang, H.S., Zheng, Q.X., Yang, J., Liu, Q.F., Tang, D.: A New Security Evaluation Method Based on Resultant for Arithmetic-Oriented Algorithms. In: International Conference on the Theory and Application of Cryptology and Information Security (AsiaCrypt). pp. 457–489. Springer (2024)

A Fast resultant for generic bivariate polynomials

In this section, we explain how we adapted Villard's bivariate resultant algorithm [38] to the context of our attack, so as to obtain an efficient solution to the step `bivSol`. After giving more context in Appendix A.1, we describe the modified algorithm and analyze its complexity in Appendix A.2, leading to the bound announced in Proposition 7.

A.1 Context

Problem and assumptions. Given two polynomials $P, Q \in \mathbb{F}[x, y]$, we focus on the task of computing their resultant with respect to y (see Definition 2), up to a scalar: we seek $\lambda \operatorname{Res}_y(P, Q)$, for any $\lambda \in \mathbb{F} \setminus \{0\}$. We use the following notation and assumptions, that are not restrictive in the context of our attack:

- We define $d_P = \operatorname{tdeg}(P)$ and $d_Q = \operatorname{tdeg}(Q)$, and we assume $d_P \geq d_Q$. One easily reduces to this using $\operatorname{Res}_y(P, Q) = \pm \operatorname{Res}_y(Q, P)$.
- The base field $\mathbb{F}$ is such that we can easily find an element $r \in \mathbb{F}$ of order at least $2d$, where d is a parameter computed in the algorithm, which is negligible compared to $\operatorname{tdeg}(P) \operatorname{tdeg}(Q)$. In the context of $\mathbb{F} = \mathbb{F}_q$ with a large cardinality $q \gg 2d$, such an element r can be found efficiently by random choice and verification. (If q is too small, one could work over an extension of $\mathbb{F}$ of degree $\lceil \log(d) \rceil$, which will only impact the logarithmic factors in the complexity bounds stated below; we did not have to resort to such extensions even in our largest experiments.)
- The polynomials P and Q are "sufficiently generic". This is a requirement for the correctness of Villard's approach [38]. We are particularly interested in the case where $Q = \lambda y^{d_Q} + \bar{Q}$ for a nonzero constant $\lambda \in \mathbb{F}$ and with $\deg_y(\bar{Q}) < d_Q$. This is satisfied for a generic Q of total degree d_Q.

Existing algorithms. For such generic P and Q with total degrees d_P and d_Q, the sought resultant has degree $D = d_P d_Q$; in general, this is an upper bound, which might not be reached. The customary approach to compute $\operatorname{Res}_y(P, Q)$ is based on an evaluation-interpolation paradigm (see [17, Cor. 11.20 and 11.21]). One first evaluates the variable x in P and Q at $D+1$ distinct points from $\mathbb{F}$, in time $\tilde{O}(nD)$ where $n = \deg_y(P) + \deg_y(Q)$. Then, for each point, one relies on the fast univariate resultant with complexity quasi-linear in n [17, Cor. 11.19], and finally one interpolates $\operatorname{Res}_y(P, Q)$ from the obtained $D+1$ values, in $\tilde{O}(D)$. In total, this costs $\tilde{O}(nD)$ operations in $\mathbb{F}$.

In 2018, a faster algorithm was proposed by Villard [38] when $\deg_x(P) = \deg_x(Q)$ and $\deg_y(P) = \deg_y(Q)$, reaching complexity $\tilde{O}(n^{1-1/\omega} D_{\max})$ for n as above and $D_{\max} = 2 \deg_x(P) \deg_y(P)$ the resultant degree bound for this type of degree profile. This algorithm has a genericity requirement, and can be sketched as follows. Let $\mathsf{S} \in \mathbb{F}[x]^{n \times n}$ be the Sylvester matrix of P and Q and define $R = \det(\mathsf{S}) = \operatorname{Res}_y(P, Q)$. The algorithm chooses a parameter $m \in \{1, \ldots, n\}$ and considers an $m \times m$ submatrix $\mathsf{H} \in \mathbb{F}(x)^{m \times m}$ of the inverse S^{-1}. Thus $R\mathsf{H}$ is

a matrix with polynomial entries, as it is a submatrix of the adjugate of S. So, to recover R, one could compute H and seek the least common denominator of its entries, which is a divisor of R and is in fact equal to R in generic situations. Yet, computing all entries of H as individual fractions is too expensive; the number of coefficients from $\mathbb{F}$ this involves is already in $\Theta(m^2 D_{\max})$, and we will see that the above complexity bound is in $\tilde{O}(m^{\omega-1} D_{\max})$, with $\omega - 1 < 2$. Instead, one seeks a representation with a fraction of matrices, as $\mathsf{H} = \mathsf{B}^{-1}\mathsf{C}$, for some denominator and numerator matrices $\mathsf{B}, \mathsf{C} \in \mathbb{F}[x]^{m \times m}$ whose degree d is as small as possible. Generically, they have degree $d \in O(D_{\max}/m)$, and they yield a representation of H with only $O(mD_{\max})$ coefficients. The resultant is recovered via $R = \det(\mathsf{B})$. The algorithm has two main components: first manipulate S as a structured matrix to obtain some implicit representation of H, which costs $\tilde{O}(nD_{\max}/m)$; then reconstruct B explicitly and compute its determinant, which costs $\tilde{O}(m^{\omega-1} D_{\max})$. The total complexity is thus $\tilde{O}((n/m + m^{\omega-1})D_{\max})$, and optimizing on the parameter m leads to $m = \Theta(n^{1/\omega})$, yielding the announced complexity bound.

The properties behind the correctness of the approach is that the minimal degree d of the numerator and denominator is in $O(D_{\max}/m)$, and that any such denominator satisfies $\det(\mathsf{B}) = \det(\mathsf{S}) = R$. This is valid for generic P and Q [38, Sec. 4], but may fail for a given specific pair (P, Q). In the latter case, the produced polynomial can be completely unrelated to the sought resultant R. Note that in our context, such unlucky cases are detected via the subsequent computations in `bivSol`, and one can then restart the resultant computation with the more general (and costlier) evaluation-interpolation approach mentioned above. In our numerous experimental runs of our implementation of the modified Villard's algorithm (Algorithm 3), we did not encounter any genericity failure.

Recently in 2024, this was further improved by Pernet, Signargout, and Villard in the case of unbalanced x- and y-degrees [32]. In particular, they obtained the complexity bound $\tilde{O}(n^{(\omega-1)/3} D_{\max})$ when $\deg_x(P)$ is in $O(\deg_y(P)^{1/3})$, and keeping the assumption $\deg_x(P) = \deg_x(Q)$ and $\deg_y(P) = \deg_y(Q)$. We do not expand on this approach since this type of degree unbalancedness does not appear in our context; however, another type does.

Handling unbalanced degrees. It has proved beneficial for our attack to target polynomials P and Q that have different degrees, and to start the resultant computation by a preliminary reduction which does not change the output resultant but does significantly reduce the y-degree of P, and therefore the size of the Sylvester matrix. This is based on the following basic result:

Lemma 1. *Let $P, Q \in \mathbb{F}[x, y]$ have total degrees d_P and d_Q, with Q nonzero and $y^{d_Q} \in Q$. Then, there is a unique pair $(\bar{Q}, \bar{P}) \in \mathbb{F}[x, y]^2$ such that $\deg_y(\bar{P}) < d_Q$ and $P = \bar{Q}Q + \bar{P}$. One can compute $\bar{P}$ using $O(\mathsf{M}(d_P^2))$ operations in $\mathbb{F}$, and it satisfies $\operatorname{tdeg}(\bar{P}) \leq d_P$ and $\operatorname{Res}_y(P, Q) = \mu \operatorname{Res}_y(\bar{P}, Q)$ for some $\mu \in \mathbb{F} \setminus \{0\}$.*

Proof. Seeing bivariate polynomials as univariate polynomials in $\mathbb{F}[x][y]$, the existence and uniqueness of $(\bar{Q}, \bar{P})$ follows from Euclidean division for univariate

polynomials in y, noticing that our assumption on Q means that its y-leading coefficient is the nonzero constant λ, which is invertible in $\mathbb{F}[x]$. The fact that $\mathrm{Res}_y(P,Q) = \mu\,\mathrm{Res}_y(\bar{P},Q)$ is then a classical property of resultants.

Assume by contradiction that $\mathrm{tdeg}(\bar{P}) > d_P$. Then, the identity $P = \bar{Q}Q + \bar{P}$ implies $\bar{Q} \neq 0$ and $\mathrm{tdeg}(\bar{P}) = \mathrm{tdeg}(\bar{Q}) + d_Q$. Since $y^{d_Q} \in Q$, there is a monomial of the form $x^i y^{j+d_Q}$ in $\bar{Q}Q$ with $i+j = \mathrm{tdeg}(\bar{Q})$. This monomial has total degree $\mathrm{tdeg}(\bar{Q}) + d_Q > d_P$ and thus $x^i y^{j+d_Q} \in \bar{P}$. This is absurd since $\deg_y(\bar{P}) < d_Q$.

Since the polynomials Q, P, and $\bar{P}$ all have x-degree at most d_P, we can substitute y with x^{d_P+1} in the identity $P = \bar{Q}Q + \bar{P}$. That is $P(x, x^{d_P+1}) = \bar{Q}(x, x^{d_P+1})Q(x, x^{d_P+1}) + \bar{P}(x, x^{d_P+1})$. Now we prove that $\bar{P}(x, x^{d_P+1})$ is the remainder in the Euclidean division of $P(x, x^{d_P+1})$ by $Q(x, x^{d_P+1})$ in $\mathbb{F}[x]$: as $y^{d_Q} \in Q$, we have that $\deg(Q(x, x^{d_P+1})) = d_Q(d_P+1)$. Now, as $\bar{P}$ is the remainder in a Euclidean division by Q in $\mathbb{F}[x][y]$, it verifies $\deg_y(\bar{P}) \leq d_Q - 1$, and thus $\deg(\bar{P}(x, x^{d_P+1})) \leq (d_Q - 1)(d_P + 1) + d_P < d_Q(d_P + 1) = \deg(Q(x, x^{d_P+1}))$, which proves our result.

We compute $\bar{P}$ as follows: the polynomials $P(x, x^{d_P+1}), Q(x, x^{d_P+1})$ are computed by copying the coefficients of P and Q. Then we perform a Euclidean division $R(x) = P(x, x^{d_P+1}) \pmod{Q(x, x^{d_P+1})}$. As we have seen, $R(x) = \bar{P}(x, x^{d_P+1})$. Now as $\deg_x(\bar{P}) \leq d_P$, the coefficient of $x^i y^j$ in $\bar{P}$ can be recovered as the coefficient of $x^{i+(d_P+1)j}$ in R. The most involved step is the Euclidean division of $P(x, x^{d_P+1})$ by $Q(x, x^{d_P+1})$, which can be performed in time $O(\mathsf{M}(\deg(P(x, x^{d_P+1})))) = O(\mathsf{M}(d_P^2))$ operations in $\mathbb{F}$.

□

A.2 Modified Villard's algorithm, and complexity analysis

For use in `bivSol`, we need three main modifications of Villard's resultant algorithm, resulting in Algorithm BivariateResultant. First, to efficiently handle cases where our input polynomials P and Q have unbalanced degrees, we start by a preliminary division with remainder step (Line 3), as discussed above. Say for example $\mathrm{tdeg}(P) \gg \mathrm{tdeg}(Q)$: this step allows us to reduce P into a polynomial $\bar{P}$ with monomial support closer to that of Q, in time that is quasi-linear in the size of P and without changing the sought resultant, i.e. $\mathrm{Res}_y(P,Q) = \mathrm{Res}_y(\bar{P},Q)$ up to a constant. Second, mostly for practical reasons, we work with interpolation bases instead of approximant bases (definitions are below). Roughly, this means that we will reconstruct a fraction $\mathsf{C}(x)\mathsf{B}(x)^{-1}$ of small-degree polynomial matrices from its evaluations at $2d$ points in the field, rather than from its power series expansion truncated at x^{2d}. Finally, some parameters have to be modified due to the fact that our input polynomials are dense in their total degree, whereas the original reference considers the maximum degree in each variable. This influences the number d of interpolation points we use, and the choice of submatrix to be extracted from the interpolation basis at Line 15.

For a given list of pairwise distinct field elements $\mathcal{P} = (r_0, \ldots, r_{2d-1})$, and a given sequence of matrices $\mathcal{H} = (H_0, \ldots, H_{2d-1})$ with each H_k in $\mathbb{F}^{m \times m}$, the

$\mathbb{F}[x]$-module of matrix rational interpolants [29,8,37] is defined as

$$\begin{aligned}\mathcal{I}_{\mathcal{P},\mathcal{H}} &= \left\{(u,v) \in \mathbb{F}[x]^m \times \mathbb{F}[x]^m \mid u(r^k)H_k = v(r^k) \text{ for } 0 \le k < 2d\right\} \\ &= \left\{(u,v) \in \mathbb{F}[x]^m \times \mathbb{F}[x]^m \mid uH = v \bmod \textstyle\prod_{k<2d}(x - r_k)\right\}, \end{aligned} \tag{1}$$

where $H \in \mathbb{F}[x]^{m\times m}$ is the unique matrix of degree less than $2d$ such that $H(r_k) = H_k$ for $0 \le k < 2d$. Given $\mathcal{P}$ and $\mathcal{H}$, one can compute a module basis of $\mathcal{I}_{\mathcal{P},\mathcal{H}}$ with good properties, as summarized in the next lemma. We will not enter details beyond those that are needed for our presentation: the computed basis is a $2m \times 2m$ matrix over $\mathbb{F}[x]$ of degree at most $2d$, is nonsingular (its determinant is nonzero), and each row corresponds to an element (u, v) of $\mathcal{I}_{\mathcal{P},\mathcal{H}}$.

Lemma 2 ([18] and [25, Prop. 3.1]). *Let $d \in \mathbb{Z}_p$ and assume $\mathbb{F}$ has cardinality at least $2d$. Given a list of pairwise distinct points $\mathcal{P} = (r_0, \ldots, r_{2d-1}) \in \mathbb{F}^{2d}$ and a matrix $H \in \mathbb{F}[x]^{m\times m}_{<2d}$, a basis of $\mathcal{I}_{\mathcal{P},\mathcal{H}}$ in weak Popov form can be computed using $O(m^\omega d \log(d) + m^2\,\mathsf{M}(d)\log(d)^2)$ operations in $\mathbb{F}$. If the points are in geometric progression, meaning $r_k = r_0^k$ for $1 \le k < 2d$, then the factor $\log(d)^2$ is replaced by a factor $\log(d)$.*

Algorithm 3 $\mathsf{BivariateResultant}(P(x,y), Q(x,y))$

Input: polynomials $P, Q \in \mathbb{F}[x,y]$ with $\operatorname{tdeg}(P) \ge \operatorname{tdeg}(Q) > 0$
Output: a polynomial $R \in \mathbb{F}[x]$

▷ *handling unbalanced degrees* ◁
if $\deg_y(P) > \deg_y(Q)$ and $y^{\operatorname{tdeg}(Q)}$ belongs to Q **then**
  $\bar{P} \leftarrow$ remainder in the division of P by Q ▷ *see Lemma 1*
else
  $\bar{P} \leftarrow P$
▷ *parameter selection* ◁
$n \leftarrow \deg_y(\bar{P}) + \deg_y(Q)$; $m \leftarrow \lceil n^{1/\omega} \rceil$; $d \leftarrow \lceil (\operatorname{tdeg}(P)\operatorname{tdeg}(Q) + n)/m \rceil$
▷ *multipoint evaluation at $\mathcal{P} = (1, r, r^2, \ldots, r^{2d-1})$* ◁
$r \in \mathbb{F} \leftarrow$ some element of order $\ge 2d$
$P_k^{\mathrm{ev}}(y) \in \mathbb{F}[y] \leftarrow \bar{P}(r^k, y)$ and $Q_k^{\mathrm{ev}}(y) \in \mathbb{F}[y] \leftarrow Q(r^k, y)$, for $0 \le k < 2d$
▷ *submatrix of $\mathsf{S}^{-1}(r^k), 0 \le k < 2d$, with S the Sylvester matrix of P, Q* ◁
$\mathsf{H}_k \in \mathbb{F}^{m\times m} \leftarrow$ top-right $m \times m$ submatrix of $\mathsf{S}(r^k)^{-1}$, computed efficiently via structured linear algebra ▷ *see proof of Proposition 7 for details*
▷ *determinant of denominator of polynomial matrix fraction $\mathsf{H} = \mathsf{C}_1\mathsf{B}_1^{-1}$* ◁
$\mathcal{P} \leftarrow (1, r, r^2, \ldots, r^{2d-1})$; $\mathcal{H} \leftarrow (\mathsf{H}_0, \ldots, \mathsf{H}_{2d-1})$
$\left[\begin{smallmatrix}\mathsf{B}_1 & \mathsf{C}_1 \\ \mathsf{B}_2 & \mathsf{C}_2\end{smallmatrix}\right] \in \mathbb{F}[x]^{2m\times 2m}_{\le 2d} \leftarrow$ weak Popov basis of $\mathcal{I}_{\mathcal{P},\mathcal{H}}$ (see Lemma 2), where submatrices have size $m \times m$
$R \in \mathbb{F}[x] \leftarrow \mathsf{DeterminantOfWeakPopov}(\mathsf{B}_1)$ ▷ *[30, Algo. 2]*
return R

We state and prove the complexity aspects of this modified version in Proposition 7. For conciseness, we do not give details concerning genericity aspects: these are close to those developed for the original version of the algorithm [38].

We emphasize again that the genericity condition was always satisfied in our practical runs of the algorithm, reported on in Section 5.

Proposition 8. *Let $P, Q \in \mathbb{F}[x, y]$ have total degree d_P, d_Q with $d_P \geq d_Q > 0$, and assume that y^{d_Q} belongs to Q. Algorithm 3 uses*

$$O\left(\mathsf{M}(d_P^2) + d_P d_Q^{2-1/\omega} \log(d_P)^2 \log\log(d_P)\right)$$

operations in $\mathbb{F}$, and the memory used during the run of the algorithm does not exceed $\tilde{O}(d_P^2 + d_P d_Q^{1+1/\omega})$ elements from $\mathbb{F}$. For generic P and Q of total degrees d_P and d_Q, with $d_P \geq d_Q > 0$, the output polynomial is $R = \mathrm{Res}_y(P, Q)$.

Proof (time and space complexity bounds). We use notation from the algorithm, as well as $d_P = \mathrm{tdeg}(P)$, $d_Q = \mathrm{tdeg}(Q)$, $n = \deg_y(\bar{P}) + \deg_y(Q)$ and $D = d_P d_Q + n$; note that $d = \lceil D/m \rceil$.

The y-degree reduction step is based on Lemma 1 and uses $O(\mathsf{M}(d_P^2))$ operations in $\mathbb{F}$. After this, the polynomial $\bar{P}$ has y-degree less than d_Q and total degree at most d_P. The rest of the algorithm computes $\mathrm{Res}_y(\bar{P}, Q)$ which, according to Lemma 1, is equal to $\mathrm{Res}_y(P, Q)$ up to a constant. Observe that a generic Q of total degree d_Q does contain y^{d_Q}.

For the moment, we let m be any parameter with $1 \leq m < d_Q$. We derive complexity bounds in terms of m eventually replace m by the choice at Line 7. From the inequality $m < d_Q$, we obtain that $d = \lceil D/m \rceil \geq \lceil d_P d_Q/m \rceil \geq d_P$, and this allows us to use the bounds $d \in O(D/m)$ and $\lceil d/d_P \rceil \in O(d/d_P)$.

Write $\bar{P} = \sum_{0 \leq i \leq \deg_y(P)} P_i(x) y^i$ and $Q = \sum_{0 \leq i \leq \deg_y(Q)} Q_i(x) y^i$, where $p_i, q_i \in \mathbb{F}[x]$. The main task at Line 10 is to apply multipoint evaluation at the geometric progression $\mathcal{P} = (1, r, r^2, \ldots, r^{2d-1})$ on all P_i's and Q_i's. For a single i, evaluating P_i can be done by splitting the list of points into $\lceil 2d/d_P \rceil$ sublists, each still in geometric progression, and calling the algorithm of [11] which costs $O(\mathsf{M}(d_P))$ for each sublist, since $\deg(P_i) \leq d_P$. This yields a total of $O(d\,\mathsf{M}(d_P)/d_P)$ operations in $\mathbb{F}$ for computing all evaluations of P_i, and thus $O(nd\,\mathsf{M}(d_P)/d_P)$ for computing the polynomial $P_k^{\mathrm{ev}}(y)$ at Line 10. Using the same approach on the Q_i's, which satisfy $\deg(Q_i) \leq d_Q \leq d_P$, the cost of computing the polynomial $Q_k^{\mathrm{ev}}(y)$ fits within the same bound.

For the computation of the H_k's at Line 12, we follow Villard's approach [38], based on tools from matrices with displacement structure [31]. It has two main steps. The first one is to compute displacement generators for the inverse of Sylvester matrices $\{S(r^k), 0 \leq k < 2d\}$. For a given k, $S(r^k) \in \mathbb{F}^{n \times n}$ is a block-Toeplitz matrix of displacement rank 2 with known generators, so that generators of its inverse $S(r^k)^{-1} = S^{-1}(r^k)$ can be computed using standard structured linear algebra results ([31, Thm. 1.5.3], [38, Sec. 5]). Specifically, this is done by solving two Sylvester linear systems (with matrix $S(r^k)$) and two transposed Sylvester linear systems (with matrix $S(r^k)^{\mathsf{T}}$), for an overall cost of $O(\mathsf{M}(n)\log(n))$ operations in $\mathbb{F}$. The second step is to use a formula that expresses $S(r^k)^{-1}$ as a function of its computed generators, like the so-called ΣLU formula [26], to compute the top-right $m \times m$ submatrix of $S(r^k)^{-1}$

through essentially $2m$ multiplications of univariate polynomials of degree m, in time $O(m\,\mathsf{M}(m))$. Both steps being done $2d$ times, Line 12 costs a total of $O((\mathsf{M}(n)\log(n) + m\,\mathsf{M}(m))d)$ operations in $\mathbb{F}$.

According to Lemma 2, the minimal interpolant basis at Line 15 can be computed within $O(m^\omega d\log(d) + m^2\,\mathsf{M}(d)\log(d))$ operations in $\mathbb{F}$; it is in weak Popov form and has degree $\leq 2d$. It follows that the leading principal submatrix B_1 is also in weak Popov form, and in particular row reduced: its determinant can be computed in $O(m^\omega\,\mathsf{M}(d)\log(d))$, according to [30, Thm. 1.2]. This also bounds the above complexity of the interpolation basis step.

Gathering all terms, we get the overall bound

$$O\left(\mathsf{M}\big(d_P^2\big) + nd\frac{\mathsf{M}(d_P)}{d_P} + (\mathsf{M}(n)\log(n) + m\,\mathsf{M}(m))d + m^\omega\,\mathsf{M}(d)\log(d)\right). \quad (2)$$

Omitting logarithmic factors, this is in $\tilde{O}(d_P^2 + (n + m^\omega)d)$, which explains the choice $m \sim n^{1/\omega}$ at Line 7. In particular, $m \in O(n^{1/2})$ and thus $m\,\mathsf{M}(m)$ is in $O(\mathsf{M}(n)\log(n))$. Equation (2) becomes

$$O\left(\mathsf{M}\big(d_P^2\big) + nd\left(\frac{\mathsf{M}(d_P)}{d_P} + \frac{\mathsf{M}(n)\log(n)}{n} + \frac{\mathsf{M}(d)\log(d)}{d}\right)\right).$$

From $d \in O(d_P d_Q/n^{1/\omega})$ we get $\log(d) \in O(\log(d_P))$, and if we assume $y^{d_Q} \in Q$, we get $n < 2d_Q \leq 2d_P$. Thus, from the general bound on $\mathsf{M}(\cdot)$ (see Section 2.2), we get the bound in the statement.

Concerning memory, each main step carried out in Algorithm 3 operates in an amount of memory that is softly linear with respect to the number of field elements in the dense representation of the input/output objects of this step. The overall memory bound is however not softly linear, since the approach itself introduces some redundancy: for example the matrix B_1 has about $m^2 d \sim md_P d_Q$ coefficients, whereas the input and output of Algorithm 3 only have about $d_P^2 + d_P d_Q$ coefficients.

An obvious bound for the maximum amount of memory used during the algorithm is $\tilde{O}(d_P^2 + (n+m^2)d)$. This can be refined by considering the fact that, as mentioned above, at Line 10 we perform multipoint evaluation on $\lceil 2d/d_P \rceil$ sublists of $\leq d_P$ points. Then, one can compute the H_k's by corresponding batches of about d_P values of k since H_k only depends on the evaluation of $\bar{P}$ and Q at the point r^k. The advantage is that each batch of evaluations can be discarded once the corresponding H_k's are obtained, since they are not needed in the rest of the algorithm. With this approach, for each batch of d_P points, one transforms the d_P^2 coefficients of $\bar{P}$ into d_P univariate polynomials of degree n (hence with size nd_P), and then into d_P constant matrices of dimensions $m \times m$ (hence with size $m^2 d_P$); everything goes similarly for Q. Thus the memory used at each sublist does not exceed $\tilde{O}(d_P^2 + nd_P + m^2 d_P)$. Over the different batches of points, the initial coefficients of $\bar{P}$ and Q do not change, and the intermediate nd_P coefficients for evaluated polynomials need not be kept after use; however we do need to accumulate the results into $\mathcal{H}$. So, in total, the maximum memory used in Lines 10 and 12 is in $\tilde{O}(d_P^2 + nd_P + m^2 d)$. Then, Lines 15 and 16 take $\tilde{O}(m^2 d)$

memory. Under the assumption $y^{d_Q} \in Q$, we have $n < 2d_Q$, so that the term nd_P is in $O(d_P^2)$. Furthermore, above-mentioned bounds show $m^2 d \in O(d_P d_Q^{1+1/\omega})$, hence the overall memory bound for Algorithm 3 announced in the statement. □

B Analysis of the big-O complexity of the attacks

In this appendix, we prove the big-O bounds for Proposition 5 and Proposition 6, used for the numerical applications in Section 5.

Proof of the big-O complexity bounds for Proposition 5. Let $\tilde{I}_j$ be the ideal defined in the proof of Proposition 5:

$$\tilde{I}_j = \langle z_1^\alpha - P_1, \ldots, z_{j-1}^\alpha - P_{j-1}, \theta^\alpha - P_j, \chi^{\alpha-1} + \cdots + \chi + 1 \rangle \ .$$

Using Proposition 3, we can multiply two polynomials P, Q in $\mathbb{F}[\mathbf{x}, z_1, \ldots, z_{j-1}, \theta, \chi]/\tilde{I}_j$ in time:

$$O\left(j \cdot \mathsf{M}\big((2\alpha-1)^{j+1}(d_P + d_Q)^k\big)\right).$$

Using the product tree described in the proof of Proposition 5, we get the following bound for the cost of the products at height i in the product tree:

$$O\left(\lfloor \alpha/2^{i+1} \rfloor \cdot j \cdot M((2\alpha-1)^{j+1})(2^i \mathrm{wdeg}(Q))^k)\right) \ .$$

Summing over $i = 1$ to $\lceil \log_2(\alpha) \rceil$, we get a complexity

$$\begin{aligned}
&O\left(j \cdot \sum_{i=1}^{\lceil \log_2(\alpha) \rceil} \lfloor \alpha/2^{i+1} \rfloor \cdot M((2\alpha-1)^{j+1}(2^i \mathrm{wdeg}(Q))^k)\right) \\
&= O\left(j \cdot M\left((2\alpha-1)^{j+1}\mathrm{wdeg}(Q)^k \sum_{i=1}^{\lceil \log_2(\alpha) \rceil} \lfloor \alpha/2^{i+1} \rfloor \cdot 2^{ik}\right)\right) \\
&= O\left(j \cdot M\left((2\alpha-1)^{j+1}\alpha^k \mathrm{wdeg}(Q)^k\right)\right) \ ,
\end{aligned}$$

since $\sum_{i=1}^{\lceil \log_2(\alpha) \rceil} \lfloor \alpha/2^{i+1} \rfloor \cdot 2^{ik} = O(\alpha^k)$ as long as $k \geq 2$.

Proof of the big-O complexity bounds for Proposition 6. Following the proof of Proposition 6, the $(n-j)$th step has complexity:

$$O\left(j \cdot \mathsf{M}\Big(\alpha^{(n-j)k}(2\alpha-1)^{j+2}\mathrm{wdeg}(Q)^k\Big)\right) \ ,$$

giving a total complexity of:

$$O\left(\sum_{j=1}^{n}(n-j)\cdot M\left(\alpha^{jk}(2\alpha-1)^{n-j+2}\mathrm{wdeg}(Q)^k\right)\right) .$$

Using the fact that $M(a)+M(b)\leq M(a+b)$, we get that the complexity is upper bounded by

$$O\left(M\left(\sum_{j=1}^{n}(n-j)\cdot\alpha^{jk}(2\alpha-1)^{n-j+2}\mathrm{wdeg}(Q)^k\right)\right) .$$

In particular:

$$\begin{aligned}&\sum_{j=1}^{n}(n-j)\cdot\alpha^{jk}(2\alpha-1)^{n-j+2}\\ =&\sum_{j=0}^{n-1}j\cdot\alpha^{(n-j)k}(2\alpha-1)^{j+2}\\ =&(2\alpha-1)^2\alpha^{nk}\sum_{j=0}^{n-1}j\cdot\left(\frac{2\alpha-1}{\alpha^k}\right)^j .\end{aligned}$$

When $k\geq 2,\alpha\geq 3$, we have that:

$$\begin{aligned}&\frac{2\alpha-1}{\alpha^k}\leq\frac{2\alpha-1}{\alpha^2}=1-\frac{\alpha^2-2\alpha+1}{\alpha^2}\\ =&1-\left(1-\frac{1}{\alpha}\right)^2\leq\frac{5}{9} .\end{aligned}$$

Thus, the sum is upper bounded by:

$$\begin{aligned}&\sum_{j=1}^{n}j\cdot\left(\frac{2\alpha-1}{\alpha^k}\right)^j\\ \leq&\sum_{j=1}^{n}j\cdot\left(\frac{5}{9}\right)^j\leq C .\end{aligned}$$

where C is a constant independent of α and k. Finally, we get complexity:

$$O\left(M\left(\alpha^2(\alpha^n\cdot\mathrm{wdeg}(Q))^k\right)\right) .$$