The Algebraic FreeLunch Efficient Gröbner Basis Attacks Against Arithmetization-Oriented Primitives

Augustin Bariant[1,2], Aurélien Boeuf[2], Axel Lemoine[2,4], Irati Manterola Ayala[3] Morten Øygarden[3], Léo Perrin[2], and Håvard Raddum[3]

[1] ANSSI, Paris, France
[2] INRIA, Paris, France
[3] Simula UiB, Bergen, Norway
[4] DGA, France

{irati,morten.oygarden,haavardr}@simula.no
{augustin.bariant,aurelien.boeuf,axel.lemoine,leo.perrin}@inria.fr

Abstract. In this paper, we present a new type of algebraic attack that applies to many recent arithmetization-oriented families of permutations, such as those used in `Griffin`, `Anemoi`, `ArionHash`, and `XHash8`, whose security relies on the hardness of the constrained-input constrained-output (CICO) problem. We introduce the FreeLunch approach: the monomial ordering is chosen so that the natural polynomial system encoding the CICO problem already *is* a Gröbner basis. In addition, we present a new dedicated resolution algorithm for FreeLunch systems of complexity lower than applicable state-of-the-art FGLM algorithms.

We show that the FreeLunch approach challenges the security of full-round instances of `Anemoi`, `Arion` and `Griffin`. We confirm these theoretical results with experimental results on those three permutations. In particular, using the FreeLunch attack combined with a new technique to bypass 3 rounds of `Griffin`, we recover a CICO solution for 7 out of 10 rounds of `Griffin` in less than four hours on one core of AMD EPYC 7352 (2.3GHz).

Keywords: Algebraic attacks · Gröbner basis · FreeLunch · Symmetric cryptanalysis · `Griffin` · `Arion` · `Anemoi`

1 Introduction

Recent years have seen the emergence of new directions in symmetric cryptography. The aim of symmetric primitives has always been to provide strong security guarantees along with stringent performance requirements. For instance, modern AES [1] implementations are able to process gigabytes of data in seconds. This has not changed, but the nature of the performance constraints considered is evolving. While the focus has traditionally been on software and hardware footprints, new use cases bring an entirely different set of relevant metrics.

Numerous such new settings exist, such as Homomorphic Encryption-friendly ciphers [16,17,39,19,8,34], ciphers designed to run efficiently in Multi Party Computation protocols [4,3,22], block ciphers defined over modular rings enabling more efficient masking [38], and Arithmetization-Oriented [5] Permutations (AOP) operating on vectors over large field elements to better integrate with modern Zero-Knowledge proof systems. Despite their differences, there are general trends in these designs, which we will group under a broad umbrella: Symmetric Techniques for Advanced Protocols (STAP).

One noticeable change the STAPs have brought about is the underlying alphabet on which these primitives operate. Until recently, the overwhelming majority of symmetric primitives were designed based on operations either on the vector space $(\mathbb{F}_2)^n$, or using arithmetic over small binary fields of size 2^m, for $3 \leq m \leq 9$. For STAP, the mathematical structures used can be fields of characteristic 2, but with a much larger size (typically $m \geq 128$), large fields of prime characteristic (say, $p > 2^{128}$), or possibly not even a field (like Elisabeth [19] or Rubato [34]).

Moreover, the performance metrics now primarily involve the number of multiplications in the underlying structure. Some schemes are designed to have a low *multiplicative depth*, that is, few multiplications in sequence on any data path from input to output, while others are designed to merely have a low number of multiplications in total. These performance metrics have led designers to propose schemes that are light on multiplications but compensate by defining the multiplication operations over large fields. The hope is that schemes constructed this way may still be secure.

Resurgence of Algebraic Attacks. Despite its functionality, the constraints on multiplications significantly impact the security analysis of STAP constructions. Key-recovery attacks can often be simplified to the resolution of a (system of) non-linear equation(s). While this general approach has been applied successfully to $\mathbb{F}_2$-based stream ciphers, traditional block ciphers and hash functions have mostly been resistant to algebraic attacks.

However, since the constraints that STAP primitives must meet often affect their algebraic structure, attacks leveraging these algebraic properties become much more threatening compared to classical symmetric attacks, such as differential of linear attacks. For example, algebraic attacks are one of the main threats to AOPs – after their initial security analysis, most designers end up setting the number of rounds specifically so as to prevent these attacks. Unfortunately, this is not always sufficient: to attack Jarvis [5], Albrecht et al. managed to re-write the equations considered by the designers in a simpler way, which made the resolution step much more efficient than initially thought [2]. When considering FHE or MPC-oriented stream ciphers, FLIP [39] and its descendant Elisabeth [19] both fell [23,29] to linearization-based attacks: the system of equations to be solved ended up being simple enough that linear algebra-based approaches could be used. The attackers did not so much re-write the equations considered; rather, they identified previously overlooked simplifications.

However, despite their crucial impact (particularly on setting the number of rounds), algebraic attacks against symmetric primitives are not nearly as well understood as classical attacks, *e.g.* differential attacks. In particular, there is no consensus among designers on how to provide solid and convincing arguments for the security of primitives against algebraic attacks.

Principles of an Algebraic Attack. Let us describe the main phases involved in the setup of an algebraic attack against a symmetric cipher. We purposefully stay at a high level here and provide a more thorough presentation in Section 2.

First, the problem of interest to the cryptanalyst is modelled as a system of polynomial equations: state variables are chosen, and equations linking them in a way that captures the round function constraints are defined. While it may seem at first glance that this encoding step does not offer a lot of freedom, it turns out that `Griffin` [32] and `Anemoi` [13] both have some specifically interesting algebraic modelizations.

Second, this system of equations is solved, using one of a few existing strategies. If the system can be represented as a unique univariate equation, an efficient FFT-based algorithm can be used to retrieve its roots, as performed in [10,9]. If, on the other hand, the system consists of several non-linear equations, one may need to convert the system into a form that can be solved efficiently. An example of this approach is "linearization", whereby the attacker introduces a new variable for each non-linear term. The resulting system becomes linear in the introduced variables and can then be solved using linear algebra, assuming that there are enough equations, as was done for example in [29].

However, in most cases, the cryptanalyst may need to rely on more sophisticated techniques. A common strategy, adopted in this paper, consists in deriving a univariate polynomial equation that shares its solution with the original system, and in efficiently solving it afterwards. This is typically done via the computation a *Gröbner basis* [20] for the system, using an algorithm such as F_4 [27], or its follow-up F_5 [28]. The Gröbner basis is then modified using a *change of order* algorithm, such as FGLM [26] or its variants [24,25,11], yielding a univariate polynomial equation sharing its solutions with the original system.

In the past, cipher designers have adopted different approaches to prevent such an attack: the authors of `Griffin` bounded the complexity of an algebraic attack with an estimate of the theoretical cost of F_5, but the authors of `Arion` [41] chose instead to bound the complexity of the change of order step[5].

Our Contributions. In this paper, we present a new type of algebraic attack that, when applicable, significantly outperforms all known methods. It requires a complete overhaul of the general approach outlined above: the encoding, the Gröbner basis computation and the change of order steps are different. In fact, the Gröbner basis comes for free, and the "reordering" step is performed using a novel and more efficient algorithm of our own design[6]. Unfortunately, the

[5] This seems to be a choice of pragmatism, as it seems easier to get tight bounds; nothing in their experiments suggests that F_5 is faster.

[6] If the solving steps were meals of the day, the lunch would be free, hence FreeLunch.

precise complexity analysis of one of the steps has remained beyond our reach. Nevertheless, we performed thorough experiments, implementing the full attack against several round-reduced primitives—we provide code to verify their results in Appendix C. Extrapolating these results, we can confidently claim *e.g.* to shave off tens of bits of security from some full-round `Griffin` instances.

Outline of this Paper. We recall the necessary background on algebraic attacks based on Gröbner bases in Section 2. Our main contribution, the FreeLunch method, is introduced in a generic fashion in Section 3, and we show how it can be successfully applied to various primitives in Section 4. While it is a priori not possible to obtain a Gröbner basis for free in some cases, we show in Section 5 that it may still be possible to derive one at a negligible cost and apply this finding to the AOP `Anemoi`. We conclude in Section 6 with a discussion on the impact of the FreeLunch approach in terms *e.g.* of design.

2 Algebraic Background

We consider the case of an algebraic attack against a permutation intended for sponge use. In this section, we will walk through the detailed inner workings of such an attack: it will allow us to introduce all the necessary mathematical background and state-of-the-art methods and to set the stage for our attacks by both providing all the theoretical tools we need and allowing to highlight the advantages of our method. Throughout the paper, we denote the base field by $\mathbb{F}$. Depending on context, this notion includes both $\mathbb{F}_p$ for a prime p and/or $\mathbb{F}_{2^n}$. We will also use the notions of polynomials and polynomial mappings interchangeably.

2.1 From an Attack to a System of Equations

We first present the constrained-input constrained-output (CICO) problem, that we will focus on solving. It was initially proposed by the designers of Keccak [33] as a crucial problem for estimating the security of permutations used in sponge constructions. It can also be seen as a variant of the limited birthday problem [30]. A natural instance in the context of algebraic cryptanalysis may be stated in the following form.

Problem 1 (CICO problem). Let $F : \mathbb{F}^t \longrightarrow \mathbb{F}^t$ be a permutation and $1 \leq \ell < t$ an integer. The goal is to find $x \in \{0\}^\ell \times \mathbb{F}^{t-\ell}$ such that $F(x) \in \{0\}^\ell \times \mathbb{F}^{t-\ell}$.

In this paper we will focus on the case $\ell = 1$. An attacker able to solve the CICO problem has control over both the input and output of the permutation, which is precisely what a good permutation is supposed to prevent. Furthermore, in our case, the value 0 in the output could be replaced e.g., by a digest d to immediately obtain a preimage attack.

Encoding. To solve a CICO instance, we need to *model* or *encode* the problem into a system of polynomial equations. In symmetric cryptography, this is usually done iteratively by modelling one round after another, possibly adding new variables to keep the degree of the initial system low. The main challenge is encoding the cipher's non-linear operations in a form that may be amenable for cryptanalysis. For STAP ciphers, the existence of low-degree models is usually possible by design: while such systems can be leveraged for cryptanalysis, they are also required for a fast verification in many ZK protocols.

Consider a non-linear function $S : \mathbb{F}^t \to \mathbb{F}^t$, and let $S_0, ..., S_{t-1}$ be its coordinate functions. A trivial model of such a function would consist of t equations of the form $y_i = S_i(x_0, ..., x_{t-1})$. In this case, a tuple $(x_0, ..., x_{t-1}, y_0, ..., y_{t-1})$ is a solution of the system if and only if we indeed have $y = S(x)$.

This situation corresponds to the simplest case, where the model is simply the evaluation of the function. However, more sophisticated models can exist, as first pointed out by the authors of Rescue [5,42]. Indeed, the non-linear layer of this permutation involves both $x \mapsto x^\alpha$ and $x \mapsto x^{1/\alpha}$, where α is a small integer. In this case, even though a non-linear function has a very high degree ($1/\alpha$ being a dense integer of $\mathbb{Z}/(p-1)\mathbb{Z}$), it is possible to design a low-degree model by using the equation $x = y^\alpha$ rather than $y = x^{1/\alpha}$.

More generally, all that is needed from a model is that it describes the graph of S, i.e., the set $\mathit{\Gamma}_S = \left\{(x, y) \mid (x, y) \in (\mathbb{F}^t)^2, y = S(x)\right\}$. In what follows, we focus on models corresponding to a system of multivariate polynomials $P = \{p_0, \dots, p_{n-1}\} \subset \mathbb{F}[x_0, \dots, x_{n-1}]$, which is associated with the following system of polynomial equations:

$$p_i(x_0, \dots, x_{n-1}) = 0 \qquad 1 \le i \le n-1 \ . \tag{1}$$

The polynomial systems we consider have a finite number of solutions in the algebraic closure of $\mathbb{F}$, and are such that there exists a solution of P which directly leads to a solution of an associated CICO problem (or to a preimage of a given hash). In the remainder of this paper, we call `sysGen` the procedure used to generate a system of equation.

If $n = 1$, then P contains a unique equation of degree D in one variable, and we can use univariate techniques to solve the problem. In practice, such an attack needs a number of operations given by $\mathcal{O}\left((D(\log(D) + \log(p)) \log(\log(D))\right)$ (see [10] for more details). We call the function returning the root(s) of a univariate polynomial `uniSol`.

2.2 Finding Structures in a System of Equations

Once our attack is represented by a system of equations, we need to solve it. To this end, we need to better understand the structures implied by a system of polynomial equations. Indeed, in order to solve the system, we need to somehow derive equations sharing the solutions of the original system, and whose solutions can be computed in practice. We thus need to formally describe the set of multivariate polynomials that have the roots we are interested in.

This set of polynomials is in fact an ideal $I = \langle P \rangle$, contained in the ring $R = \mathbb{F}[x_0, \ldots, x_{n-1}]$. We denote d_i the degree of equation p_i, and D_I the ideal degree of I, *i.e.* the number of solutions of P in the algebraic closure of $\mathbb{F}^n$, counted with multiplicity. In order to study this ideal, we need the notion of *monomial order*.

Definition 1. *A* ***monomial order*** $\prec$ *is a total order on the set of monomials of R such that i) for any monomial $m \in R$ we have $1 \prec m$; and ii) for any three monomials $m_1, m_2, t \in R$ we have*

$$m_1 \prec m_2 \implies t \cdot m_1 \prec t \cdot m_2 \ .$$

The typical monomial orders used in computations are the *lexicographical (lex)* order and the *graded reverse lexicographical (grevlex)* order. We now define a particular *weighted* order which we will use throughout the paper.

Definition 2. *Consider a weight vector $\mathbf{w} = (w_0, \ldots, w_{n-1}) \in \mathbb{R}^n$, where $w_0 \neq 0$. We say that $\mathbf{w}$ is* ***associated with the monomial order*** $\prec$*, defined by:*

$$\prod_{i=0}^{n-1} x_i^{\alpha_i} \prec \prod_{i=0}^{n-1} x_i^{\beta_i} \iff \begin{cases} \displaystyle\sum_{i=0}^{n-1} w_i\alpha_i < \sum_{i=0}^{n-1} w_i\beta_i \\ \text{or} \\ \exists k, \quad \displaystyle\sum_{i=0}^{n-1} w_i\alpha_i = \sum_{i=0}^{n-1} w_i\beta_i, \ \forall j > k, \alpha_j = \beta_j \text{ and } \alpha_k \prec \beta_k. \end{cases}$$

Technically speaking, this defines a weighted lex order with $x_0 \prec x_1 \prec \ldots \prec x_n$. The particularities of this choice will be needed in Section 5.

Definition 3. *The* ***leading monomial*** *of a nonzero polynomial $f \in R$, relative to a monomial order $\prec$, is the largest monomial contained in f according to $\prec$. It is denoted $\mathrm{LM}(f)$. The* ***leading coefficient*** *of f, $\mathrm{LC}(f)$, is the coefficient associated with $\mathrm{LM}(f)$. Finally, the* ***leading term*** *of f, $\mathrm{LT}(f)$, is the product of its leading monomial and coefficient.*

If $S = \{f_1, f_2, \ldots\} \subseteq R$, then we can extend the above definitions to the set S, e.g., $LT(S) = \{LT(f_1), LT(f_2), \ldots\}$. We may now define the notion of a Gröbner basis of an ideal of R.

Definition 4 (Gröbner basis [14]). *Let I be an ideal of R. A finite set of polynomials $G \subset I$ is a* ***Gröbner basis*** *with respect to $\prec$ if the leading monomial of every polynomial in I is a multiple of the leading monomial of some polynomial in G. A Gröbner basis G is said to be* ***reduced*** *if for all $g \in G$, no monomial in g is divisible by an element of $\mathrm{LT}(G) \setminus \{\mathrm{LT}(g)\}$ and $\mathrm{LC}(G) = \{1\}$.*

An ideal I always contains a Gröbner basis. For a fixed $\prec$ there are usually many Gröbner bases, but only one reduced Gröbner bases. We will crucially rely on the following results throughout this paper.

Proposition 1 ([20, Chapter 2,§9, Prop 4 and Thm 3]). *Let G be a set of polynomials of R, $G = \{g_1, ..., g_m\}$. If the leading monomials of g_i and g_j are relatively prime for all $1 \leq i \neq j \leq m$, then G is a Gröbner basis for $\langle G \rangle$.*

Proposition 2 ([20, Chapter 2 §6 Prop 1]). *Let I be an ideal, $\prec$ a monomial order, G a Gröbner basis of I w.r.t. $\prec$, and $f \in R$. There exists a unique $r \in R$ such that:*

- $\mathrm{LT}(r)$ *is not divisible by any element of* $\mathrm{LT}(G)$.
- $\exists\ g \in I$, *such that* $f = g + r$.

The polynomial r is called the ***remainder*** *or* ***normal form*** *of f w.r.t. I and $\prec$.*

2.3 Exploiting a Gröbner Basis

The characteristics of a Gröbner basis can vary greatly depending on the underlying monomial order. For our goal, that is finding solutions of P, one typically wants to compute a Gröbner basis of $\langle P \rangle$ in the *lex* order. However, computing a Gröbner basis directly in this order tends to be computationally expensive. Instead, it is common to first compute a Gröbner basis in the *grevlex* order – which tends to be significantly faster – and then apply a dedicated order-changing algorithm, such as FGLM or its variants [24,25,26], to finally recover a basis in lex order. While we do not give a complete description of these algorithms, we nevertheless hightlight some of the principles underpinning them, as they will be needed later for our own resolution algorithm.

The Quotient Ring. Thanks to Proposition 2, we can define the quotient ring R/I, where each class has a unique representative r such that $\mathrm{LT}(r)$ is not divisible by any element of a Gröbner basis G. The monomial order $\prec$ does not affect the quotient ring R/I, but determines the representative of each class.

Definition 5. *Let I be an ideal of R. We say that I is* zero-dimensional *if* $\dim_{\mathbb{F}}(R/I)$ *is finite. In this case, its* ideal degree D_I *is* $\dim_{\mathbb{F}}(R/I)$.

The quotient ring R/I has a canonical basis with respect to $\prec$ denoted $\mathcal{B}_\prec(R/I)$, where the basis elements are given by all the monomials in R that are not in the ideal $\langle \mathrm{LM}(G) \rangle$, for G a Gröbner basis for I. If I is zero-dimensional, we have $|\mathcal{B}_\prec(R/I)| = D_I$. Each element r of R/I can then be written as a vector in the basis $\mathcal{B}_\prec(R/I)$, which we will call $\mathrm{NormalForm}(r)$. This allows us to define the linear matrix $T_j : R/I \to R/I$ corresponding to the multiplication by x_j.

Definition 6 (Multiplication matrix of x_j). *The* ***multiplication matrix*** *T_j of x_j relative to a zero-dimensional ideal I, a monomial order $\prec$, and the basis $\mathcal{B}_\prec(R/I) = (\epsilon_1, \ldots, \epsilon_{D_I})$ is defined as the square matrix which has each column defined as $C_i = \epsilon_i \times x_j$ represented in the basis $\mathcal{B}_\prec(R/I)$.*

Said differently, T_0 is the $D_I \times D_I$ matrix mapping the basis elements $\epsilon_i \mapsto \mathrm{NormalForm}(x_0\epsilon_i)$. The following result is well-known in the literature, but we have not been able to find a reference that holds for finite fields (e.g., it is derived as Corollary 4.6 in [21] over $\mathbb{C}$). For completeness, we provide a short proof that works over any field.

Proposition 3. *Let I be a zero-dimensional ideal of R, $\prec$ a monomial order, and T_0 the multiplication matrix of the variable x_0 with respect to $\prec$. We have $\det(x_0\boldsymbol{I} - T_0) \in I$, where $\boldsymbol{I}$ is the identity matrix.*

Proof. Let $C(x) = \det(x\boldsymbol{I} - T_0) = \sum_{i=0}^{D_I} c_i x^i$ be the characteristic polynomial of T_0. By the Cayley-Hamilton Theorem we have $C(T_0) = \sum_{i=0}^{D_I} c_i T_0^i = \mathbf{0}$, where $\mathbf{0}$ is the $D_I \times D_I$ zero-matrix. Letting ϵ denote the column vector representing the constant polynomial 1 in R/I, we then have $C(T_0)\epsilon = 0$. As $T_0^i\epsilon$ is the representation of $\mathrm{NormalForm}(x_0^i)$, this implies that $\mathrm{NormalForm}(C(x_0)) = 0$, hence $C(x_0) \in I$. □

The next definition is a very standard and common hypothesis for an ideal when implementing Gröbner basis polynomial solving algorithms.

Definition 7 ([24], Definition 3.1). *An ideal I of R is in* ***shape position*** *if its reduced Gröbner basis in the lexicographical order has the following form*

$$G = \{f_0(x_0),\ x_1 - f_1(x_0),\ \ldots,\ x_{n-1} - f_{n-1}(x_0)\}.$$

In this case, it follows straightforwardly that $D_I = \deg(f_0)$, and the cost of finding solutions for I, given its lexicographic Gröbner basis, reduces to the problem of finding the roots of f_0.

3 The Algebraic FreeLunch

In this section, we present our own custom approach to algebraic attacks, which is applicable to solve CICO instances for some arithmetization-oriented permutations (see Section 4). As with the algebraic attacks we presented in the previous section, we first need to describe our problem using a system of polynomial equations. We start with a description of the general form of system for which a Gröbner basis can be obtained for free (see Section 3.1). Then, we show how to deduce a univariate polynomial from this system in Section 3.2. Means to create these polynomial systems for various primitives are discussed in Sections 3.3 and 3.4, where we reuse and generalize an encoding technique introduced by the authors of `Griffin` in way that can be applied to iterated functions. The entire solving strategy is summarized in Section 3.5.

3.1 FreeLunch Systems

We saw in Proposition 1 that there is a class of polynomial systems that admits a simple Gröbner basis. This is the motivation for the following definition.

Definition 8 (FreeLunch System). *Let R be the ring $\mathbb{F}[x_0, \ldots, x_{n-1}]$ and $P = \{p_0, \ldots, p_{n-1}\}$ be a sequence of polynomials of R. We say that P is a* ***FreeLunch system*** *if there exists a monomial order $\prec$ and integers $(\alpha_0, \ldots, \alpha_{n-1})$ such that for all $i \in \{0, \ldots, n-1\}$, $\mathrm{LM}_{\prec}(p_i) = x_i^{\alpha_i}$. Any monomial order $\prec$ that verifies this property is said to be a FreeLunch order.*

Note that this is not the first time Proposition 1 has been used in cryptography. In [15], the authors describe a polynomial modeling for AES that can be said to be a FreeLunch system in a *graded lex* order. However, the ensuing change of order computation to a *lex* order is too costly to threaten the security of AES. The following properties are now easy to verify, and were also used in [15].

Proposition 4. *A FreeLunch system P is a Gröbner basis for the ideal $I = \langle P \rangle$ with respect to any of its FreeLunch orders. Moreover, I is zero-dimensional and of ideal degree $D_I = \prod_{i=0}^{n-1} \alpha_i$.*

Proof. The first statement follows directly from Proposition 1. For the latter statement, note that the canonical basis of R/I (w.r.t. a FreeLunch order $\prec$) is

$$\mathcal{B}_{\prec}(R/I) = \{x_0^{i_0} \cdots x_{n-1}^{i_{n-1}} \mid 0 \leq i_j < \alpha_j, \text{ for } 0 \leq j \leq n-1\}.$$

Counting all these basis elements yields D_I. □

We conceived a dedicated algorithm for the resolution of FreeLunch systems, which has a competitive time complexity. The following result will be proven in Section 3.2, where $2 \leq \omega \leq 3$ denotes the linear algebra exponent.

Theorem 1. *Given a FreeLunch system P, a FreeLunch order $\prec$, and the associated multiplication matrix T_0 of the variable x_0, there exists an algorithm to compute a solution for x_0 with time complexity*

$$\tilde{\mathcal{O}}\left(\alpha_0 \left(\prod_{i=1}^{n-1} \alpha_i\right)^{\omega}\right) .$$

3.2 Extracting a Univariate Equation from a FreeLunch System

We now turn to the problem of solving a FreeLunch system, with the aim of showing Theorem 1. While a FreeLunch system is already a Gröbner basis, it is typically only so under specially crafted monomial orders, as we will see in later sections. To easily retrieve the solutions of a FreeLunch system, we look for a univariate polynomial belonging to the ideal spawned by the system. To do so, a common approach is to compute a Gröbner basis in the *lex* order. Given an initial Gröbner basis, computing a *lex* Gröbner basis can be performed using a *change of order* algorithm

Exisiting change of order algorithms. The FGLM-algorithm [26] provides an efficient method for changing the monomial orders of Gröbner bases of zero-dimensional ideals, running in $\mathcal{O}(D_I^3)$ operations. Later variants, such as [24,25] focused on the particular case of changing between the *grevlex* and *lex* order for ideals in shape position. They improve the complexity to respectively $\mathcal{O}(D_I^\omega)$ and $\mathcal{O}(TD_I^2)$ for the fast FGLM and the sparse FGLM algorithms, where T is the sparsity indicator of the multiplication matrix T_0. Recently, Berthomieu *et al.* designed a new change of order algorithm from *grevlex* to *lex* based on Hermite Normal Forms [11], running in $\mathcal{O}(T^{\omega-1}D_I)$, which requires the stability property [11, Definition 2.1] and the shape position. The stability property is unfortunately not verified by FreeLunch systems, so this algorithm is not applicable directly. Moreover, in the case of [11], we note that the FreeLunch systems does not generally satisfy the stability property. We note that the authors of [11] state that when the base field is large enough, and the ideal under consideration is radical, the stability property can be ensured through a generic linear change of coordinates. The issue is that doing so might transform the FreeLunch system into a different type of system that is not a Gröbner basis.

A new approach. These reasons led us to design a new dedicated algorithm for finding a univariate polynomial $f_0(x_0)$ belonging to the ideal, exploiting the sparsity of the multiplication matrix T_0.

Let I be a zero-dimensional ideal of $R = \mathbb{F}[x_0, \ldots, x_{n-1}]$, $\prec$ a monomial order giving a FreeLunch system, $\mathcal{B}_\prec = (\epsilon_1, \ldots, \epsilon_{D_I})$ the canonical basis of R/I, and T_0 the multiplication matrix corresponding to the variable x_0. Let H be the subspace of R/I containing the classes h of R/I where the unique representative of h with respect to $\prec$ does not contain the variable x_0. Let D_1 be the dimension of H and $\mathcal{B}_\prec^H = [\phi_1, \ldots, \phi_{D_1}]$ be a canonical basis for the subspace H. It is clear that $\mathcal{B}_\prec^H$ exactly consists of the monomials m of $\mathcal{B}_\prec$ such that $x_0 \nmid m$. Thus, it holds that $D_1 = \displaystyle\prod_{i=1}^{n-1} \alpha_i = D_I/\alpha_0$. We order the basis $\mathcal{B}_\prec$ specifically as

$$[\phi_1, \ldots, \phi_{D_1},\ x_0\phi_1, \ldots, x_0\phi_{D_1},\ x_0^2\phi_1, \ldots, x_0^2\phi_{D_1},\ \ldots,\ x_0^{\alpha_0-1}\phi_1, \ldots, x_0^{\alpha_0-1}\phi_{D_1}]\ ,$$

and identify any polynomial $f \in R/I$ with its coefficient vector v_f of length D_I. The coefficient vector for the polynomial $x_0 f \in R/I$ can then be computed as a matrix/vector multiplication $T_0 v_f^\top$ for a fixed matrix T_0. The following lemma gives the structure of T_0.

Lemma 1. *Under the basis $\mathcal{B}_\prec$, the matrix T_0 is of the following form, represented as a block matrix with block sizes $D_1 \times D_1$:*

$$T_0 = \begin{pmatrix} 0 & 0 & \ldots & 0 & -M_0 \\ \boldsymbol{I} & 0 & \ldots & 0 & -M_1 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \ldots & 0 & \boldsymbol{I} & -M_{\alpha_0-1} \end{pmatrix}.$$

The block matrices $M_0, \ldots, M_{\alpha_0-1}$ are a representation of the reduction of $x_0^{\alpha_0}\mathcal{B}_{\prec}^H$ modulo I. The exact entries in the M_i matrices depend on the particular polynomials making up the Gröbner basis for the FreeLunch system. We call `matGen` the procedure which, given a basis $\mathcal{B}_{\prec}$, returns T_0.

From Proposition 3 it follows that $\det(x_0\boldsymbol{I}_{D_I} - T_0)$ is a univariate polynomial belonging to the ideal I. Computing this determinant and using a root-finding algorithm to solve for $\det(x_0\boldsymbol{I}_{D_I} - T_0) = 0$ will finally give us a value for x_0 that solves the CICO problem. The following lemma shows that computing this determinant of this particularly structured matrix T_0 can be done with much lower complexity than for a generic matrix of dimension D_I.

Lemma 2. *Let $M_0, \ldots, M_{\alpha_0-1}$ be the matrices defined in Lemma 1. We have*

$$\det(x_0\boldsymbol{I}_{D_I} - T_0) = \pm \det\left(x_0^{\alpha_0}\boldsymbol{I}_{D_1} + \sum_{i=0}^{\alpha_0-1} x_0^i M_i\right) .$$

Proof.

$$\det(x_0\boldsymbol{I}_{D_I} - T_0) = \det\begin{pmatrix} x_0\boldsymbol{I} & 0 & \ldots & 0 & M_0 \\ -\boldsymbol{I} & x_0\boldsymbol{I} & \ldots & 0 & M_1 \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & 0 & -\boldsymbol{I} & x_0\boldsymbol{I} & M_{\alpha_0-2} \\ 0 & 0 & 0 & -\boldsymbol{I} & x_0\boldsymbol{I} + M_{\alpha_0-1} \end{pmatrix} .$$

The rows of this matrix can be split into a set of α_0 blocks of D_1 rows each. Denote these blocks as $L_0, \ldots, L_{\alpha_0-1}$ from top to bottom. We now do elementary row operations block-wise, from bottom to the top, with $L_i = L_i + x_0 L_{i+1}$, for $i = \alpha_0 - 2, \ldots, 0$. This does not change the value of the determinant and after these row operations the resulting determinant to compute is:

$$\det\begin{pmatrix} 0 & 0 & \ldots & 0 & x_0^{\alpha_0}\boldsymbol{I} + \sum_{i=0}^{\alpha_0-1} x_0^i M_i \\ -\boldsymbol{I} & 0 & \ldots & 0 & x_0^{\alpha_0-1}\boldsymbol{I} + \sum_{i=0}^{\alpha_0-2} x_0^i M_{i+1} \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \ldots & -\boldsymbol{I} & 0 & x_0^2\boldsymbol{I} + \sum_{i=0}^{1} x_0^i M_{i+\alpha_0-2} \\ 0 & \ldots & 0 & -\boldsymbol{I} & x_0\boldsymbol{I} + M_{\alpha_0-1} \end{pmatrix} .$$

In this block matrix representation, the determinant of the full matrix is the determinant of the top right matrix, up to the sign $(-1)^{\alpha_0+1}$. □

Complexity Analysis. We call `polyDet` the procedure returning the polynomial $\det(x_0\boldsymbol{I}_{D_I} - T_0)$ using Lemma 2. This step has a complexity $\tilde{\mathcal{O}}(D_I D_1^{\omega-1})$ wwith the algorithm of Labahn *et al.* [37]. Note that this is precisely the complexity that was obtained with the algorithm of Berthomieu *et al.* for systems satisfying the *stability* and *shape position* properties. In order to estimate the logarithmic factors in the complexity formula, we bound the complexity with [31, Theorem 4.4], using a polynomial matrix multiplication algorithm of complexity $\mathcal{O}(D_1^\omega \log(\alpha_0) + D_1^2 \log(\alpha_0) \log(\log(\alpha_0)))$ [18]. This way, we bound the number of operations of `polyDet` with (when D_1 is large):

$$\mathcal{O}\left(\alpha_0 \log(\alpha_0)^2 D_1^\omega + \alpha_0 \log(\alpha_0)^2 \log(\log(\alpha_0)) D_1^2\right) \approx \mathcal{O}(\alpha_0 \log(\alpha_0)^2 D_1^\omega) \ . \qquad (2)$$

The remaining task to show for Theorem 1 is to recover the roots of a univariate polynomial of degree D_I, a step we refer to as `uniSol`. This costs $\tilde{\mathcal{O}}(D_I)$ operations and is thus negligible in comparison with the `polyDet` step.

We want to highlight that the complexity of the `matGen` step is hard to estimate precisely (recall that T_0 is assumed known in Theorem 1). As we will see in later experiments, this step can be costlier than `polyDet`.

3.3 Ordering a FreeLunch

Having seen how to efficiently find solutions for FreeLunch systems, we will focus in the next two subsections on the problem of actually finding them. Recall that FreeLunch systems rely on the existence of specific monomial orders. How can we figure out if such an order exists (and thus, if a system is a FreeLunch)? In general, answering this question is not trivial. However, the systems we will be concerned with in Sections 4 and 5 naturally have a deeper structural property that allows for a procedural approach to this problem.

Definition 9 (Triangular System). *Let $P = (p_1, \ldots, p_{n-1}, g)$ be a polynomial system in $\mathbb{F}[x_0, x_1, \ldots, x_{n-1}]$. We say that P is a* ***triangular system*** *if there exists polynomials $q_0, q_1, \ldots, q_{n-1}$, integers $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$, and $c_0, \ldots, c_{n-1} \in \mathbb{F} \setminus \{0\}$ such that*

$$\begin{cases} p_i &= c_i x_i^{\alpha_i} + q_i(x_0, \ldots, x_{i-1}) \qquad \textit{for} \ \ 1 \le i \le n-1 \ , \\ g &= c_0 x_0^{\alpha_0} + q_0(x_0, \ldots, x_{n-1}) \ . \end{cases}$$

A triangular system P can be assigned the following monomial order that is naturally motivated by the FreeLunch definition.

Construction 1 *For a triangular system P, we define its triangular order, $\prec_T$, as the monomial order from Def. 2 associated with the weight vector defined recursively by:*

$$\begin{cases} \mathrm{wt}(x_0) &= 1 \ , \\ \mathrm{wt}(x_i) &= \mathrm{wt}\left(\mathrm{LM}_{\prec_T}\left(q_i(x_0, x_1, \ldots, x_{i-1})\right)\right) / \alpha_i \qquad \textit{for} \ \ 1 \le i \le n-1 \ . \end{cases}$$

The recursion is well-defined since the leading monomial of q_i and its associated weight are only dependent on the weights of x_j for $j < i$. The definition ensures that $\mathrm{LM}_{\prec_T}(p_i) = x_i^{\alpha_i}$ for $1 \le i \le n-1$. Hence, a triangular system P is a FreeLunch system with respect to $\prec_T$ if the leading monomial of g is univariate in x_0, which gives the following Proposition:

Proposition 5 (Ordering a FreeLunch). *Let P be a triangular system, and $\prec_T$ be its triangular order. If $\alpha_0 > \mathrm{wt}(\mathrm{LM}_{\prec_T}(q_0(x_0, \ldots, x_{n-1})))$ then P is a FreeLunch system and $\prec_T$ is one of its FreeLunch orders.*

As we will see below, such systems naturally occur when investigating some cryptographic permutations.

3.4 FreeLunch Systems From Iterated Functions

The permutations we target share the same structure: a composition of a number of round functions. The input and output of every round is a state of t elements[7] from $\mathbb{F}$ and the round functions typically consist of a limited number of multiplications and α-th roots in $\mathbb{F}$. Writing them out directly as polynomial functions yields polynomials of high degree, owing to the α-th root operations. A natural modeling strategy introduces a new variable for each of them to keep the degree growth manageable, as $x = y^\alpha$ is of much lower degree than $y = x^{1/\alpha}$ when $\alpha \in \{3, 5, ..., 257\}$ and $|\mathbb{F}|$ is large. In this section, we take inspiration from an encoding suggested by the authors of `Griffin` [32] and show how to model this class of primitives as polynomials that form a low degree FreeLunch system.

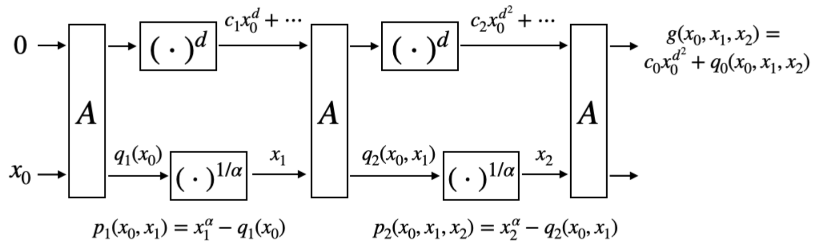


Fig. 1: Triangular system for a simple SPN with two branches and two rounds.

Toy Example. Let us start with a toy SPN of two rounds, where the round function F is given by $F = S \circ A : \mathbb{F}^2 \to \mathbb{F}^2$, for an invertible affine layer A and a non-linear layer S. Moreover, we write $S = (S_1, S_2)$ where $S_1(y) = y^d$, for a small integer d, and $S_2(y) = y^{1/\alpha}$. This simple construction is shown in Figure 1, where we also label the branches with variables and polynomials at different points. We consider a CICO problem with input $(0, x_0)$.

As we assume $d \ll p$, we note that the round function F can only achieve a high degree as a polynomial function $\mathbb{F}^2 \to \mathbb{F}^2$ due to the map S_2. Thus, we introduce new variables x_1 and x_2 for the output of S_2 in the first and second rounds, respectively. The polynomials p_1, p_2 relate the symbolic input and output

[7] We say that the permutation has t *branches*, or as we like to think of them, *brunches*.

of the two S_2-functions, where q_1 is an affine polynomial in x_0 that is input to S_2 in the first round, and $q_2(x_0, x_1)$ is the input to S_2 in the second round and has degree d in the x_0-variable and degree 1 in the x_1-variable. Finally, we let $g(x_0, x_1, x_2)$ represent the first output of the construction that is required to be 0 by the CICO-problem. We can now write g as

$$g(x_0, x_1, x_2) = c_0 x_0^{d^2} + q_0(x_0, x_1, x_2),$$

for a suitable constant $c_0 \in \mathbb{F}$, and where $q_0(x_0, x_1, x_2)$ has degree $< d^2$ in x_0, degree d in x_1 and degree 1 in x_2. If $c_0 \neq 0$ we observe that $P = \{p_1, p_2, g\}$ forms a triangular system (Definition 9), whose solutions yield a solution to the specified CICO-problem. The weight vector of $\prec_T$ from Construction 1 is $(1, 1/\alpha, d/\alpha)$, and it is straightforward to verify that P satisfies the condition of Proposition 5. Hence, P is a FreeLunch system.

General Case. The above example shows the core idea for how a FreeLunch system can be made from a round function that relies on the functional inverse of low degree function to achieve a high degree. Let us generalize this insight. Let $F_i : \mathbb{F}^t \to \mathbb{F}^t$, $\mathbf{z}_{i-1} \mapsto \mathbf{z}_i$, denote the i-th round of a primitive, where $\mathbf{z}_{i-1} = (z_{i-1,0}, \ldots, z_{i-1,t-1})$ is the state after $i-1$ rounds. Recall that F_i may itself have a high degree (in $\mathbf{z}_{i-1}$), but suppose there exists a set of variables $\mathbf{x}_i = \{x_{i,0}, \ldots, x_{i,\ell_i-1}\}$ satisfying

$$x_{i,j}^{\alpha_{i,j}} = \mathcal{L}_{i,j}(\mathbf{z}_{i-1}), \text{ for } 0 \leq j < \ell_i, \tag{3}$$

where $\alpha_{i,j}$ is an integer and $\mathcal{L}_{i,j}$ an affine function. Moreover, suppose that there exists a polynomial function $G_i : \mathbb{F}^{t+\ell_i} \to \mathbb{F}^t$ of low degree d_i, satisfying

$$F_i(\mathbf{z}_{i-1}) = \{G_i(\mathbf{z}_{i-1}, \mathbf{x}_i) \mid \mathbf{x}_i \text{ satisfies } (3)\}. \tag{4}$$

In other words, while F_i and G_i are different as polynomial functions, they yield the same output when $\mathbf{x}_i$ is restricted by (3). For instance, in the toy example above, we used

$$G_1(\mathbf{z}_0, x_1) = \left((A_1(\mathbf{z}_0))^d, x_1\right), \qquad G_2(\mathbf{z}_1, x_2) = A_1\left((A_1(\mathbf{z}_1))^d, x_2\right),$$

where A_1 denotes the first output of A.

Polynomial Modeling. We now have an iterated function of t branches where each round can be described using the functions $\mathcal{G} = \{G_1, \ldots, G_r\}$ satisfying (3) and (4), and where G_i is of degree d_i. We introduce the shorthand $d_{\leq i} = d_1 d_2 \cdots d_i$, and we require that the exponents d_i are small enough to ensure that their composition will not exceed the maximal degree determined by the finite field, *i.e.* $d_{\leq r} < |\mathbb{F}| - 1$.

With this in place, we give the following blueprint for constructing a polynomial system. Recall that we focus on the variant of the CICO-problem where a single input in $\mathbb{F}$ is unknown, which we will symbolically denote by x_0, and the output of the first branch should be 0. The initial state is written as $\mathbf{z}_0(x_0)$, which consists of t affine polynomials in x_0. The proceeding state is defined as $\mathbf{z}_1(x_0, \mathbf{x}_1) = G_1(\mathbf{z}_0, \mathbf{x}_1)$, where we note that $\mathbf{z}_1$ is now t polynomials of at most degree d_1 in the variables $x_0, \mathbf{x}_1$. Furthermore, we create functions $\mathbf{p}_1 = \{p_{1,0}, \ldots, p_{1,\ell_1-1}\}$ to encode the relations (3) that we encounter in this step. That is, for $\mathbf{x}_1 = \{x_{1,0}, \ldots, x_{1,\ell_1-1}\}$, we construct the polynomials

$$p_{1,j} = x_{1,j}^{\alpha_{1,j}} - \mathcal{L}_{1,j}(\mathbf{z}_0), \text{ for } 0 \leq j < \ell_1 .$$

This process of updating the state $\mathbf{z}_i$ and constructing polynomials[8] $\mathbf{p}_i$ is repeated for all rounds up to $r-1$. In the last round, we generate polynomials $\mathbf{p}_r$ as before, but instead of updating the state, we compute the final polynomial

$$g(x_0, \mathbf{x}_1, \ldots, \mathbf{x}_r) = [G_r(\mathbf{z}_{r-1}, \mathbf{x}_r)]_1,$$

where $[\cdot]_1$ means the first polynomial of $G_r(\mathbf{z}_{r-1}, \mathbf{x}_r)$. This construction yields the polynomial system $P_\mathcal{G} = \{\mathbf{p}_1, \ldots, \mathbf{p}_r, g\}$ over the ring $\mathbb{F}[x_0, \mathbf{x}_1, \ldots, \mathbf{x}_r]$.

$P_\mathcal{G}$ as a FreeLunch system. It is easy to verify that $P_\mathcal{G}$ is a triangular system if g contains a univariate monomial in x_0. In fact, this is a stronger case than the generic triangular systems considered in Section 3.3, since we are also able to bound the degrees of the polynomials in $P_\mathcal{G}$ by round degrees $d_1, \ldots, d_r$. This allows us to give an analogous variant of Proposition 5 for $P_\mathcal{G}$. Instead of a condition on the entire system that could be computationally expensive to verify, we reduce the assumption to the condition of a single monomial in g.

Proposition 6. *Let $P_\mathcal{G}$ be a polynomial system as constructed above, where all $\alpha_{i,j}$ from (3) are at least 2, and the functions $\mathcal{G} = \{G_1, \ldots, G_r\}$ are of degrees $d_1, \ldots, d_r \geq 2$. Then $P_\mathcal{G}$ is a FreeLunch system if g contains the monomial $x_0^{d_{\leq r}}$.*

Before proving the proposition, we start by defining $\prec_\mathcal{G}$, which is the monomial order from Definition 2 whose weight vector is given by

$$\begin{cases} \mathrm{wt}(x_0) &= 1 , \\ \mathrm{wt}(x_{i,j}) &= d_{\leq i-1}/\alpha_{i,j} \qquad \text{for } \ 1 \leq i \leq r \ \text{ and } \ 1 \leq j \leq \ell_i , \end{cases}$$

where we define $d_{\leq 0} = 1$. Recall that $\mathbf{z}_i$ denotes the i-th state represented by t polynomials in $x_0, \mathbf{x}_1, \ldots, \mathbf{x}_i$. We will write $\mathrm{wt}(\mathrm{LM}(\mathbf{z}_i))$ for the maximal weight among the monomials of these t polynomials.

Lemma 3. *Let $\mathbf{z}_i$ be the i-th state associated with a system $\mathcal{G}$ that satisfies the conditions of Proposition 6. Then the following inequality holds for $\prec_\mathcal{G}$*

$$\mathrm{wt}\left(\mathrm{LM}(\mathbf{z}_i)\right) \leq d_{\leq i} .$$

[8] If a single variable is introduced in a round, we will ease notation by writing $\mathbf{x}_i = x_i$, $\mathbf{p}_i = p_i$ and α_i.

Proof. We proceed by induction. The base case of $i = 0$ is immediate since $\mathbf{z}_0$ is affine in x_0, and $d_{\leq 0} = 1$ by definition. For the induction step, we recall that $\mathbf{z}_i = G_i(\mathbf{z}_{i-1}, \mathbf{x}_i)$, where G_i has degree d_i. Thus we have

$$\mathrm{wt}(\mathrm{LM}(\mathbf{z}_i)) \leq d_i \cdot \max\{\mathrm{wt}(\mathrm{LM}(\mathbf{z}_{i-1})), \mathrm{wt}(x_{i,1}), \ldots, \mathrm{wt}(x_{i,\ell_i})\} \ .$$

Now we have $\mathrm{wt}(x_{i,j}) < d_{\leq i-1}$, and $\mathrm{wt}(\mathrm{LM}(\mathbf{z}_{i-1})) \leq d_{\leq i-1}$ by the induction hypothesis. Hence

$$\mathrm{wt}(\mathrm{LM}(\mathbf{z}_i)) \leq d_i d_{\leq i-1} = d_{\leq i} \ . \qquad \square$$

The proof of this lemma also implies that $\prec_{\mathcal{G}}$ coincides with $\prec_T$ from Construction 1 if all functions $\mathcal{L}_{i,j}(\mathbf{z}_{i-1})$ achieve their maximal weight $d_{\leq i-1}$. We now have all we need to show Proposition 6.

Proof. (Proposition 6). From Lemma 3 we observe

$$\begin{aligned} \mathrm{wt}\left(x_{i,j}^{\alpha_{i,j}}\right) &= \alpha_{i,j} \mathrm{wt}\left(x_{i,j}\right) = d_{\leq i-1} \\ &\geq \mathrm{wt}(\mathrm{LM}(\mathbf{z}_{i-1})) \geq \mathrm{wt}(\mathrm{LM}(\mathcal{L}_{i,j}\left(\mathbf{z}_{i-1}\right))). \end{aligned}$$

Hence $\mathrm{LM}\left(f_{i,j}\right) = x_{i,j}^{\alpha_{i,j}}$. Moreover, Lemma 3 also guarantees that

$$\mathrm{wt}\left(\mathrm{LM}(g)\right) \leq \mathrm{wt}(\mathrm{LM}(\mathbf{z}_r)) \leq d_{\leq r}.$$

Due to the fact that $\alpha_{i,j} \geq 2$, the factor $1/\alpha_{i,j}$ that appears in the weight of all variables $\mathbf{x}_i$, $i \geq 1$, the above equality can only be achieved by the monomial $x_0^{d_{\leq r}}$. It then follows from the assumption that $\mathrm{LM}(g) = x_0^{d_{\leq r}}$, which makes $P_{\mathcal{G}}$ a FreeLunch system. $\square$

Computing a reduced Gröbner Basis for $\langle P_{\mathcal{G}} \rangle$ (`sysGen`). We have just seen that computing a Gröbner basis for a given FreeLunch system $P_{\mathcal{G}}$ is – as the name suggests – free. There are, however, two practical concerns worth addressing. Firstly, while $P_{\mathcal{G}}$ is itself a Gröbner basis, it is generally not the unique reduced Gröbner basis w.r.t. any of its FreeLunch orders. Secondly, generating the polynomials in $P_{\mathcal{G}}$ may itself be hard.

In practice we do not generate the polynomials in $P_{\mathcal{G}}$ in the direct manner outlined earlier. Rather, we will use the fact that $\mathbf{p}_i$ and g are constructed by composing certain round functions. This allows us to reduce by the polynomials $\mathbf{p}_1, \ldots, \mathbf{p}_{i-1}$ introduced earlier in the process in order to suppress the growth of the number of monomials. This is detailed in Appendix A, where we show that when $P_{\mathcal{G}}$ satisfies the conditions of Proposition 6, the polynomial system generated in this manner is also a FreeLunch system that is the unique reduced Gröbner basis for $\langle P_{\mathcal{G}} \rangle$ w.r.t. $\prec_{\mathcal{G}}$. Appendix A also provides a complexity estimate for generating this latter polynomial system, under the assumption that reductions following every multiplication of multivariate polynomials can be done efficiently.

3.5 Summary of the FreeLunch Attack

The strategy of the attack presented in this section is summarized in Algorithm 1. The initial condition is that there exists a FreeLunch system associated with

1. `sysGen`: Generate a FreeLunch system (Section 3.4).
2. `matGen`: Compute the multiplication matrix T_0 (Section 3.2).
3. `polyDet`: Compute $f(x_0) = \det\left(x_0^{\alpha_0}\boldsymbol{I}_{D_1} + \sum_{i=0}^{\alpha_0-1} x_0^i M_i\right)$ (Section 3.2).
4. `uniSol`: Solve $f(x_0) = 0$.

Algorithm 1: Overview of the FreeLunch Attack.

the target primitives. Methods for constructing this FreeLunch system were presented in Section 3.3 and 3.4, and the complexities for `sysGen` using these methods are discussed in Appendix A. A different way of generating a FreeLunch system will also be shown in Section 5. We will estimate the complexity of `polyDet` by (2), but we do not have a clear estimate for `matGen`. The final step `uniSol` recovers the roots of a univariate polynomial of degree D_I. This costs $\tilde{\mathcal{O}}(D_I)$ operations and is thus negligible in comparison with the earlier steps. We expect the complexity of the attack as a whole to be dominated by either `matGen` or `polyDet` for the primitives we have investigated. This is in line with our experiments (see Section 6.1), where `matGen` seems to be the dominating step for larger instances.

The numbers for the complexity of the `polyDet` step in our attacks against several AO permutations are shown are in[9] Table 1. Details of how we obtained them will be provided further in the paper.

4 Using FreeLunch Systems Directly

Experimental Verification In this section and in Section 5, we support theoretical attacks with practical experiments on reduced-round versions. All experiments are performed on 1 core of AMD EPYC 7352 (2.3GHz) with 250 GB of memory, and on $\mathbb{F}_p$ with $p =$ `0x64ec6dd0392073`. The `sysGen` step is performed with `SageMath` [44], Magma [12] or the NTL [45] and Flint [35] libraries, the `matGen` step is performed with Flint, and the `polyDet` step is perfomed with the Polynomial Matrix Library [43,36].

[9] The complexities correspond to the number of basic $\mathbb{F}_p$ operations; writing them as number of calls to the primitive would yield lower but hard to compute numbers.

Name	α/e	Number of branches 2	3	4	5	6	8	≥ 12
`Griffin`	3	$\emptyset$	120 (16)	112 (15)	$\emptyset$	$\emptyset$	76 (11)	64 (10)
	5	$\emptyset$	141 (14)	110 (11)	$\emptyset$	$\emptyset$	81 (9)	74 (9)
`Arion`	3	$\emptyset$	128 (6)	134 (6)	114 (5)	119 (5)	98 (4)	$\emptyset$
	5	$\emptyset$	132 (6)	113 (5)	118 (5)	122 (5)	101 (4)	$\emptyset$
α-`Arion`	3	$\emptyset$	104 (5)	84 (4)	88 (4)	92 (4)	98 (4)	$\emptyset$
	5	$\emptyset$	83 (4)	87 (4)	91 (4)	94 (4)	101 (4)	$\emptyset$
`Anemoi`	3	118 (21)	$\emptyset$	-	$\emptyset$	-	-	-
	5	156 (21)	$\emptyset$	-	$\emptyset$	-	-	-
	7	174 (20)	$\emptyset$	-	$\emptyset$	-	-	-
	11	198 (19)	$\emptyset$	-	$\emptyset$	-	-	-

Table 1: Time complexity ($\log_2$) of `polyDet` in FreeLunch-based attacks against some full-round algorithms (aiming at 128-bit security). Number of rounds in parentheses, $\emptyset$ corresponds to undefined algorithms. The α/e column reports α for `Griffin` and `Anemoi`; and e for the `Arion` variants.

4.1 A Detailed Example: `Griffin`

Specification of `Griffin`. `Griffin` [32] is a family of sponge hash and compression functions proposed by Grassi *et al.* at Crypto 2023 designed to be used in Zero-Knowledge applications. As such, it makes use of the internal permutation `Griffin`-π, which is defined over the finite field $\mathbb{F}$.

Each round function of `Griffin`-π is composed of a non-linear layer, the addition of a round constant, and a linear layer defined by multiplication by an MDS matrix.The specific features of `Griffin` impose that the primitive is only suitable for $\mathbb{F}^t$ where $t = 3$ or t is a multiple of four.

Definition 10 (Non-linear layer of `Griffin`-π). *Let $\alpha \in \{3, 5, 7, 11\}$ be the smallest integer such that $\gcd(\alpha, p-1) = 1$, $p > 2^{63}$ and let t be the number of branches. For $0 \leq i \leq t-1$, let $(\delta_i, \mu_i) \in \mathbb{F}^2 \setminus \{(0,0)\}$ be pairwise distinct such that $\delta_i^2 - 4\mu_i$ is a quadratic nonresidue modulo p. Then, the non-linear layer of `Griffin`-π is $S(x_0, \ldots, x_{t-1}) = (y_0, \ldots, y_{t-1})$, where each y_i is defined by the equations:*

$$y_i := \begin{cases} x_0^{1/\alpha} & \text{if } i = 0 \\ x_1^{\alpha} & \text{if } i = 1 \\ x_2 \cdot (L_2(y_0, y_1, 0)^2 + \delta_2 \cdot L_2(y_0, y_1, 0) + \mu_2) & \text{if } i = 2 \\ x_i \cdot (L_i(y_0, y_1, x_{i-1})^2 + \delta_i \cdot L_i(y_0, y_1, x_{i-1}) + \mu_i) & \text{otherwise,} \end{cases}$$

for $L_i(z_0, z_1, z_2) = (i-1) \cdot z_0 + z_1 + z_2$.

Definition 11 (`Griffin`-π). *Let r be the number of rounds, and for $1 \leq i \leq r-1$ let $\mathbf{c}^{(i)} \in \mathbb{F}^t$ be a constant vector (we assume $\mathbf{c}^{(r)} = 0$). Then `Griffin`-π $\mathcal{G}^\pi : \mathbb{F}^t \to \mathbb{F}^t$ is defined as*

$$\mathcal{G}^\pi(\cdot) := \mathcal{F}_r \circ \cdots \circ \mathcal{F}_2 \circ \mathcal{F}_1(M \times \cdot),$$

where for $1 \leq i \leq r$, the i-th round function $\mathcal{F}_i$ is defined as

$$\mathcal{F}_i(\cdot) = \mathbf{c}^{(i)} + M \times S(\cdot),$$

for $M \in \mathbb{F}^{t \times t}$ a matrix, and S the non-linear layer of ***`Griffin`**-π.*

The first round function of `Griffin`-π for $t = 4$ is depicted in Fig. 2 where, to simplify the construction, we denote by F_i the last two equations of Definition 10. The authors proposed various instances with a 128-bit security claim. The number of branches varies from 3 to 24 (though not all values are possible), and the number of rounds is computed for different degrees α based on the complexity of finding a Gröbner basis using the basic encoding as it was the most efficient attack they could find.

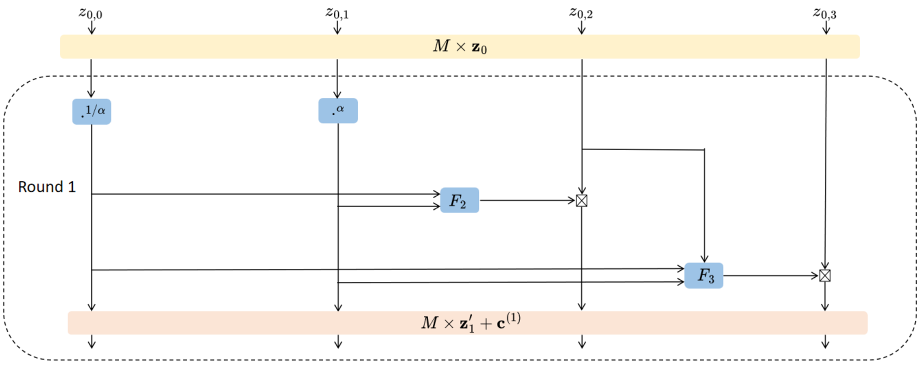


Fig. 2: First round function of `Griffin`-π with $t = 4$.

FreeLunch system for `Griffin`. We observe that the round function of `Griffin` readily lends itself to a naïve construction of the system $P_{\mathcal{G}}$, as described in Section 3.4. Indeed, for each round i we can simply define $\mathbf{z}_i = G_i(\mathbf{z}_{i-1}, x_i)$ by $\mathbf{z}_i = \mathbf{c}^{(i)} + M \times \mathbf{z}_i'$, where $z_{i,0}' = x_i$ and $z_{i,j}'$ given as y_j, for $1 \leq j < t$, in Definition 10 of the i-th round. Note that G_i will be of degree at most $d_i = 2\alpha + 1$. Under the assumption that the polynomial g in $P_{\mathcal{G}}$ satisfies the monomial property of Proposition 6, we get an associated ideal degree of $(\alpha\,(2\alpha + 1))^r$.

Remark 1. Note that the naïve modeling $P_{\mathcal{G}}$ given above for `Griffin` is not new; in fact, it was invented by the authors of this algorithm for their initial security analysis [32, Section 6.2]. However, the authors did not attempt to compute a Gröbner basis for $\langle P_{\mathcal{G}} \rangle$ in a FreeLunch order, but rather in the usual grevlex order. They estimate that computing a Gröbner basis in this latter monomial order will well exceed the security level for the suggested number of rounds.

Bypassing Several Rounds. A further improvement is constructing an affine input in x_0 for the CICO problem that is tailored to bypass the inversion operation for a few initial rounds. This effectively means that fewer variables x_i are necessary, which in turn has a significant impact on the resulting ideal degree. Observe that bypassing inversions fits seamlessly with the machinery introduced in Section 3.4. The only difference is that we choose a different sequence of polynomial functions $\mathcal{G}^*$, where G_1^* effectively spans several rounds but only depends on $\mathbf{z}_0$. The ensuing functions G_i^*, $i \geq 2$, can still be constructed as described for the naïve method above (though there will now be fewer of them). The exact number of initial rounds we can bypass will depend on t, where a larger t generally allows us to bypass more rounds[10]. All underlying details are given in Appendix B.

For $t = 3, 4$, we can bypass one round with linear functions in $\mathbf{z}_0$. For $t = 8$, we are able to bypass two rounds with cubic functions in $\mathbf{z}_0$, and three rounds can be bypassed with $\deg(\mathbf{z}_0) = 6\alpha + 3$ for $t \geq 12$. Assuming all systems satisfy Proposition 6, we get the following ideal degrees.

$$D_{I,t} = \begin{cases} (\alpha\,(2\alpha+1))^{r-1}\,, & \text{for } t = 3, 4, \\ 3\,(\alpha\,(2\alpha+1))^{r-2}\,, & \text{for } t = 8, \\ (6\alpha+3)\,(\alpha\,(2\alpha+1))^{r-3}\,, & \text{for } t \geq 12\ . \end{cases} \tag{5}$$

Complexity Analysis and Experimental Results. We can now use the machinery learned in Section 3.2 to solve the above-described FreeLunch system for `Griffin`. As noted in Section 3.2, it is hard to theoretically estimate the complexity of `matGen` where one computes the multiplication matrix T_0. On the other hand, based on previous analysis, we estimate the complexity of `polyDet` by computing $D_{I,t}D_{1,t}^{\omega-1} = D_{I,t}(D_{I,t}/\alpha_0)^{\omega-1}$ for the different values of $D_{I,t}$. As a consequence, the running time for `polyDet` becomes

$$\tilde{\mathcal{O}}(D_{I,t}D_{1,t}^{\omega-1}) = \begin{cases} \tilde{\mathcal{O}}((\alpha^{\omega}\,(2\alpha+1))^{r-1}), & \text{for } t = 3, 4, \\ \tilde{\mathcal{O}}(3\,(\alpha^{\omega}\,(2\alpha+1))^{r-2}), & \text{for } t = 8, \\ \tilde{\mathcal{O}}((6\alpha+3)\,(\alpha^{\omega}\,(2\alpha+1))^{r-3}), & \text{for } t \geq 12\ . \end{cases} \tag{6}$$

The resulting estimated time complexities of running `polyDet` for the proposed instances of `Griffin` are listed in Table 2. Experimental results are presented in Table 3 and discussed in Section 6.1.

4.2 Applicability Beyond `Griffin`: the Example of `ArionHash`

Specification of `ArionHash`. `ArionHash` [41] is an arithmetization-oriented hash function proposed by Roy *et al.* that, much like `Griffin`, uses a permutation

[10] A similar observation of bypassing rounds was already considered in [32, Section 6.2]. However, the authors only describe a method for bypassing a single round for $t = 3$ and do not consider the effect of having a larger t.

Table 2: Estimated time complexities of `polyDet` for the different proposed instances of full-round `Griffin`, where $\omega = 2.81$. Number of rounds in parenthesis.

Branches	Complexity ($\log_2$)	
	$\alpha = 3$	$\alpha = 5$
3	120 (16)	141 (14)
4	112 (15)	110 (11)
8	76 (11)	81 (9)
12,16,20,24	64 (10)	74 (9)

Table 3: Experimental results on `Griffin` with $(t, \alpha) = (12, 3)$. `sysGen` uses Flint and NTL with the fast multivariate multiplication algorithm of Appendix A.

Number of rounds	Complexity of `polyDet`	Time (s)			Memory (MB)
		`sysGen`	`matGen`	`polyDet`	
5	26	0.17	0.02	0.53	14
6	34	4.0	6.67	50.78	471
7	41	$2,558$	$3,361$	$5,727$	$27,600$

as its core primitive. Called `Arion`-π, this permutation utilizes in each round a polynomial of very high degree in one branch and low degree polynomials in the remaining branches to significantly decrease the number of necessary rounds to achieve the desired security.

Definition 12 (Non-linear layer of `Arion`-π). *Let $p \geq 5$ be a prime, t the number of branches, e the smallest positive integer be such that* $\gcd(e, p-1) = 1$, *and* $121 \leq \alpha \leq 257$ *an integer such that* $\gcd(\alpha, p-1) = 1$.

For $0 \leq i \leq t-2$, let $\delta_{i,1}, \delta_{i,2}, \mu_i \in \mathbb{F}$ be such that $g_i(x) = x^2 + \delta_{i,1} \cdot x + \delta_{i,2}$ is a quadratic function without zeroes in $\mathbb{F}$ and define $h_i(x) = x^2 + \mu_i \cdot x$. Then the non-linear layer of `Arion`-π *is $\mathcal{S} = \{f_0, \ldots, f_{t-1}\}$, where each f_i is defined "from-right-to-left" by the equations:*

$$\begin{aligned} f_{t-1}(x_0, \ldots, x_{t-1}) &= x_{t-1}^{1/\alpha}, \\ f_i(x_0, \ldots, x_{t-1}) &= x_i^e \cdot g_i(\sigma_{i,t}) + h_i(\sigma_{i,t}), \qquad t-2 \geq i \geq 0 \end{aligned}$$

where $\sigma_{i,t}$ represents the sum of all previously computed inputs and outputs

$$\sigma_{i,t} = \sum_{j=i+1}^{t-1} x_j + f_j(x_0, \ldots, x_{t-1}) \ .$$

Definition 13 (`Arion`-π). *Let r be the number of rounds, and for $1 \leq i \leq r$ let $\mathbf{c}_i \in \mathbb{F}^t$ be a constant vector. Then* ***`Arion`***-π *is defined as the following composition over $\mathbb{F}^t$:*

$$\textit{Arion-}\pi : (x_0, \ldots, x_{t-1}) \mapsto (L_{\mathbf{c}_r} \circ \mathcal{S}_r) \circ \cdots \circ (L_{\mathbf{c}_1} \circ \mathcal{S}_1) \circ L_0(x_0, \ldots, x_{t-1}),$$

where $L_{\mathbf{c}_i}$ is the affine map of [41, Definition 3] and $\mathcal{S}_i$ is the non-linear layer of ***`Arion`***-π, *for $1 \leq i \leq r$.*

We illustrate the construction of the first round of `Arion`-π in Fig. 3 for $t = 4$ where, for the sake of clarity, we only represent the function f_i of the GTDS without the details of g_i and h_i.

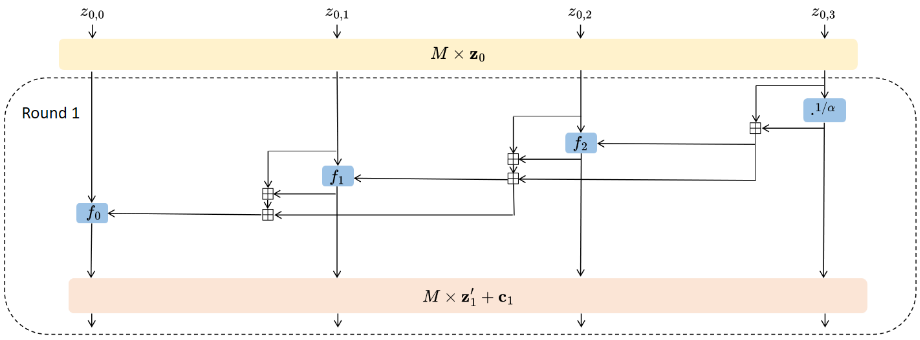


Fig. 3: First round function of `Arion`-π with $t = 4$.

We provide the parameters for `Arion`-π and `ArionHash` as well as for their additionally proposed aggressive versions α-`Arion` and α-`ArionHash` with $e = 3, 5$ and $\alpha = 121$ in Table 4 (number of rounds are in parenthesis). The authors claim 128-bit security for each parameter set.

FreeLunch system for `ArionHash`. Due to the similarities in construction between `Arion`-π and `Griffin`-π, it comes as no surprise that the round function of `Arion`-π also fits the naïve construction of the system $P_{\mathcal{G}}$ described in Section 3.4. In this case, for each round i we define $\mathbf{z}_i = G_i(\mathbf{z}_{i-1}, x_i)$ by $\mathbf{z}_i = L_{c_i}(\mathbf{z}'_i)$, where $z'_{i,t-1} = x_i$ and $z'_{i,j} = f_j(\mathbf{z}_{i-1})$ for $0 \leq j \leq t-2$. Note that each component of $\mathbf{z}'_i$ (and thus of $\mathbf{z}_i$) will have degree at most $d_i = (2^{t-1}(e+1) - e)^i$. Assuming that the polynomial g in $P_{\mathcal{G}}$ satisfies the monomial property of Proposition 6, we get an associated ideal degree of $\left(\alpha \left(2^{t-1}(e+1) - e\right)\right)^r$.

In addition, one can further improve this technique by generating a set of input states constructed so that the inversion operation for the first round is bypassed, reducing the number of necessary variables and, consequently, the associated ideal degree. This is done analogously to `Griffin`, and all underlying details can be found in Appendix D. For `Arion` we are only able to bypass a single round with $\deg(\mathbf{z}_0) = 3e$, independent of t. Assuming all systems satisfy Proposition 6, we get the ideal degree

$$D_I = 3e \left(\alpha \left(2^{t-1}(e+1) - e\right)\right)^{r-1} .$$

Complexity Analysis and Experimental Results. We can now apply the new methods introduced in Section 3.2 to solve the FreeLunch system for

	Arion-π & ArionHash		α-Arion& α-ArionHash	
Branches	Complexity ($\log_2$)		Complexity ($\log_2$)	
	$e = 3$	$e = 5$	$e = 3$	$e = 5$
3	128 (6)	132 (6)	104 (5)	83 (4)
4	134 (6)	113 (5)	84 (4)	87 (4)
5	114 (5)	118 (5)	88 (4)	91 (4)
6	119 (5)	122 (5)	92 (4)	94 (4)
8	98 (4)	101 (4)	98 (4)	101 (4)

Table 4: Estimated time complexity ($\log_2$) of `polyDet` for the different full-round instances of `ArionHash`, where $\alpha = 121$ and $\omega = 2.81$. Number of rounds in parenthesis.

Number of branches	Complexity of `polyDet`	Time (s) `sysGen`	`matGen`	`polyDet`	Memory (MB)
3	32	1.31	< 0.01	6.8	$3,387$
4	33	1.46	0.07	18.7	$7,551$
5	35	9.54	0.08	64.5	$15,903$
6	36	247	0.31	215	$32,626$
8	39	$24,872$	4.86	$2,545$	$134,165$

Table 5: Experimental results on 2-round `Arion`, with $(e, \alpha) = (3, 121)$. `sysGen` is performed using `SageMath`. `polyDet` uses an evaluation/interpolation algorithm of pml [43] since the algorithm of [37] implemented in pml does not work for the non-generic polynomial matrix in input of `polyDet`.

`ArionHash`. Based on the general complexity analysis of the attack, we list the estimated time complexities of `polyDet` for the different proposed `ArionHash` parameters in Table 4. Note that here $D_1 = D_I/\alpha_0 = \alpha^{r-1}$, so that the running time for `polyDet` becomes

$$\tilde{\mathcal{O}}(D_I D_1^{\omega-1}) = \tilde{\mathcal{O}}\left(3e\left(\alpha^\omega\left(2^{t-1}(e+1)-e\right)\right)^{r-1}\right) .$$

Experimental results are presented in Table 5 and discussed in Section 6.1.

4.3 Last Example: `XHash8`

`XHash8` is a permutation proposed by Ashur, Kindi and Mahzoun in [6]. Along with `XHash12`, it is a follow-up of RPO [7], itself a follow-up of Rescue-Prime [42]. `XHash8` features a layer with inversion operations in eight out of twelve branches. Thus we need to introduce polynomials $\mathbf{p}_i = (p_{i,0}, \ldots, p_{i,7})$ and variables $\mathbf{x}_i = (x_{i,0}, \ldots, x_{i,7})$ in these layers. We can, by adjusting for this minor difference, directly define a FreeLunch system as we have seen in the previous two subsections. Since this is very similar to what we saw for `Griffin` and `Arion`, we will give the details for this analysis in Appendix F, and only limit ourselves to a short discussion of the highlights in the immediate following. We note that the related constructions XHash12, RPO and Rescue-Prime all contain a layer of inversion operations in all branches, and hence we cannot directly obtain a FreeLunch from them.

Complexity and Impact on Security Analysis. In contrast to the flexible constructions of `Griffin`-π and `Arion`-π, `XHash8` is only defined for a fixed prime $p \approx 2^{64}$ and a fixed sponge setting with state size $t = 12$, where the rate is 8 and capacity 4. Although this does not directly lend itself to a CICO problem with a single zero in input and output, we applied the FreeLunch approach to this setting. We generate a FreeLunch system with $\alpha_0 = 7^6$ and $D_1 = 7^{24}$, whose time complexity of `polyDet` is approximately 2^{240} for $\omega = 2.81$. As this is significantly higher than brute force for the chosen p, we conclude that `XHash8` seems very secure against the techniques presented in this paper.

That said, we note that the current security estimates for `XHash8` are (conservatively) extrapolated from scaled-down experiments with $t = 3$ using a single unknown input [6, Appendix B]. While the FreeLunch framework cannot currently be extrapolated in a similar manner for the full construction, we still hope it could provide a basis for future insights into the security of `XHash8`.

5 Forcing the Presence of a FreeLunch for `Anemoi`

We have just seen three examples where the FreeLunch machinery of Section 3.4 could be readily applied. `Anemoi` is another class of permutations that rely on the inverse of low degree monomials in a finite field to achieve a high degree and so it would, a-priori, seem like another candidate where we can apply the FreeLunch techniques. However, we will see that this is not as straightforward as it may appear because a direct application of the technique creates a polynomial system $P_{\mathcal{G}}$ where g does not satisfy the assumption of Proposition 6. Instead, we will show how to compute a modified polynomial system $P_{\mathcal{G}^*}$ that retains the valid solution to the CICO problem, which will turn out to be a FreeLunch system. This comes at the cost of a somewhat larger, yet still comparable, ideal degree than what was given in Conjecture 2 of [13]. We start by describing `Anemoi`.

Description of `Anemoi`. The `Anemoi` permutations [13] operate on $\mathbb{F}_q^{2\ell}$ for $\ell \geq 1$, and either $q = 2^n$ with n odd, or $q = p$ for any prime $p \geq 3$. There are differences between the operations for the odd and even characteristic cases that will impact our later modeling. Thus, we focus on the setting of $\ell = 1$ and p prime, leaving the even case as future work. In odd characteristic, `Anemoi` takes a parameter α such that $x \mapsto x^\alpha$ is a permutation of $\mathbb{F}$, usually $\alpha = 3$, 5, 7 or 11. The original paper gives two specific hash function instances based on `Anemoi` with $\ell = 1$: `AnemoiSponge-BN-254`, with a 254-bit prime p, `AnemoiSponge-BLS12-381`, with a 381-bit prime p. 127 bits of security are claimed for both of these.

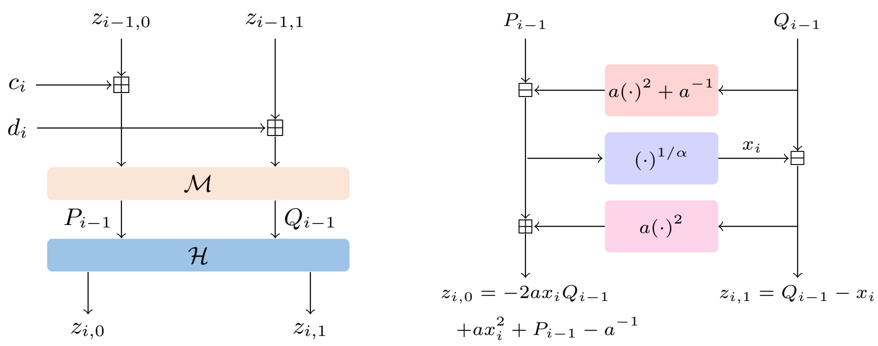
Definition 14 (Odd Anemoi with $\ell = 1$). *For a given p, α and number of rounds r, Anemoi is a permutation of $\mathbb{F}^2$ defined as*

$$\texttt{Anemoi}_{p,\alpha,r}(x, y) = \mathcal{M} \circ \mathsf{R}_r \circ \cdots \circ \mathsf{R}_1(x, y).$$

For $1 \leq i \leq r$, the i-th round function R_i is defined as

$$\mathsf{R}_i(x, y) = \mathcal{H} \circ \mathcal{M}(x + c_i, y + d_i) \qquad \text{and} \qquad \mathcal{M}(x, y) = (2x + y, x + y),$$

for constants $c_i, d_i \in \mathbb{F}$. $\mathcal{H}$ is the nonlinear operation over $\mathbb{F}^2$ that is described in Figure 4b for a non-zero constant $a \in \mathbb{F}$.



(a) Anemoi round function.

(b) $\mathcal{H}$ in odd characteristic.

Fig. 4: Description of Anemoi over prime fields with $\ell = 1$.

Failure of the Direct FreeLunch Approach. As a starting point, we consider the following slight modification[11] of the polynomial system $P_{\mathcal{G}}$ for Anemoi, for $1 \leq i \leq r$.

$$p_i(x_0, \ldots, x_i) = x_i^\alpha + aQ_{i-1}(x_0, \ldots, x_{i-1})^2 - P_{i-1}(x_0, \ldots, x_{i-1}) + a^{-1}, \quad (7)$$
$$g(x_0, \ldots, x_r) = P_r(x_0, \ldots, x_r) \ . \quad (8)$$

A first observation is that $z_{i,0}$ must have a larger leading term than $z_{i,1}$ under any monomial order. Since this leading term gets distributed to both branches under $\mathcal{M}$ (without the possibility of cancelling the leading term), we have $\mathrm{LM}(Q_i) = \mathrm{LM}(P_i)$. Now note from the output shown in Figure 4b that in the computation of $z_{i,0}$, the terms aQ_{i-1}^2 and $-aQ_{i-1}^2$ will both occur and cancel each other. Hence, the leading monomial of g must be either $x_r\mathrm{LM}(Q_{r-1})$ or x_r^2, so there is

[11] The only difference from the description in Section 3.4 is that we allow $\mathcal{L}_{i,j}$ from (3) to be quadratic, due to the term Q_{i-1}^2.

no possible choice of monomial order where g will have a leading monomial in only x_0.

In order to circumvent this issue, we will multiply g by suitable monomials in $x_1, \dots, x_r$ that leads to a reduction by the polynomials $p_1, \dots, p_r$. This process will ultimately lead to a new polynomial g^*, whose leading monomial will be univariate in x_0. To briefly illustrate the idea, we consider the first step of this procedure. Writing out g in terms of P_{r-1}, Q_{r-1} and x_r, we have

$$g(P_{r-1}, Q_{r-1}, x_r) = (1 - 4ax_r)Q_{r-1} + (2ax_r - 1)x_r + 2(P_{r-1} - a^{-1}) \ ,$$

taking into account the final $\mathcal{M}$-transformation. In order to cancel out the product $x_r Q_{r-1}$ using p_r, we construct the following polynomial:

$$\begin{aligned} g' &= x_r^{\alpha-1} g + (4aQ_{r-1} - 2ax_r + 1)p_r \\ &= 4a^2 Q_{r-1}^3 + aQ_{r-1}^2 + 4Q_{r-1}(1 - aP_{r-1}) - P_{r-1} + a^{-1} + \\ & x_r^{\alpha-1}(Q_{r-1} + 2P_{r-1} - 2a^{-1}) - 2x_r(a^2 Q_{r-1}^2 - aP_{r-1} + 1) \ . \end{aligned}$$

Hence, we have successfully eliminated x_r from the leading monomial of g' under any monomial order that satisfies

$$\mathrm{wt}\left(\mathrm{LM}\left(Q_{r-1}^3\right)\right) > \ \mathrm{wt}\left(\mathrm{LM}\left(x_r^{\alpha-1}(Q_{r-1} + P_{r-1}) + x_r(Q_{r-1}^2 - P_{r-1})\right)\right) \ ,$$

which, since $\alpha \geq 3$, can be simplified further to

$$\mathrm{wt}\left(\mathrm{LM}\left(Q_{r-1}^2\right)\right) > \mathrm{wt}\left(x_r^{\alpha-1}\right) = (\alpha - 1)\mathrm{wt}(x_r) \ .$$

Constructing FreeLunch Systems From `Anemoi`. We now turn our attention to the general construction of g^* that will allow us to apply the FreeLunch machinery for solving the CICO problem for `Anemoi`. Here, we will not only be interested in the leading monomials of the intermediate states and p_i, but also in the second and third monomials. To this end, we define $\prec_A$ to be the monomial order associated with the weight vector defined recursively by

$$\begin{cases} \mathrm{wt}(x_0) = 1, \\ \mathrm{wt}(x_i) = \dfrac{2}{\alpha}\mathrm{wt}(x_0 \cdots x_{i-1}), \text{ for } 1 \leq i \leq r \ . \end{cases}$$

Indeed, this choice of monomial order allows us to prove the following two lemmas. For a polynomial h, we let $\mathrm{Mon}_j(h)$ denote the j-th monomial of h according to $\prec_A$. As usual, we write $\mathrm{LM}(h) = \mathrm{Mon}_1(h)$. To avoid pathological cases, we always consider an affine input in x_0 for the CICO-problem such that x_0 is not eliminated after the initial linear operation $\mathcal{M}$. Finally, remember that two monomials may have equal weight, and only get sorted by their lexicographic order.

Lemma 4. *Let $Q_i(x_0, \dots, x_i)$ be as defined in Figure 4a and ordered according to $\prec_A$. Then the following holds for $\alpha \geq 3$.*

$$\mathrm{LM}(Q_i) = x_0 \cdots x_i, \qquad \textit{and} \qquad \mathrm{wt}(\mathrm{LM}(Q_i)) > \mathrm{wt}(\mathit{Mon}(Q_i)) \ .$$

Proof. We proceed by induction. The statements are clearly true for $i = 0$, as Q_0 is an affine polynomial in x_0 by our CICO setting. Now assume it holds for $i-1$. As mentioned above, leading terms cannot be canceled under $\mathcal{M}$. Thus, we can restrict ourselves to the two largest monomials in the first output from $\mathcal{H}$, that is $-ax_iQ_{i-1} + ax_i^2 - P_{i-1} - a^{-1}$. From the induction hypothesis we have $\mathrm{LM}(x_iQ_{i-1}) = x_0 \cdots x_i$, and it follows from the definition of $\prec_A$ that this has a strictly higher weight than x_i^2 when $\alpha \geq 3$. $\square$

Lemma 5. *Let $p_i(x_0, \ldots, x_i)$ be as defined in (7) and ordered according to $\prec_A$, and let $\alpha \geq 3$. Then for all $1 \leq i \leq r$ the following holds.*

1. $\mathrm{LM}(p_i) = x_i^\alpha$.
2. $Mon_2(p_i) = (x_0 \cdots x_{i-1})^2$.
3. $\mathrm{wt}(\mathrm{LM}(p_i)) = \mathrm{wt}(Mon_2(p_i)) > \mathrm{wt}(Mon_3(p_i))$.

Proof. We see from the definition of p_i that $\mathrm{LM}(p_i)$ must be either x_i^α or $\mathrm{LM}(Q_i^2)$. From Lemma 4, we have $\mathrm{wt}(\mathrm{LM}(Q_i^2)) = 2\mathrm{wt}(x_0 \cdots x_{i-1})$, so these two monomials have the same weight by definition of $\prec_A$. $\mathrm{LM}(p_i) = x_i^\alpha$ then follows from Definition 2. Finally, $\mathrm{Mon}_3(p_i) = \mathrm{Mon}_2(Q_i^2)$ and thus has a strictly smaller weight than the initial two monomials (Lemma 4). $\square$

Before we can define g^*, we also need a way to predict the powers of x_i we will use in the multiplication of g prior to the reductions by $p_1, \ldots, p_r$. This is handled by the following integer sequences.

Definition 15. *We define two integer sequences $\{u_i\}_{0 \leq i \leq r}$ and $\{k_j\}_{1 \leq j \leq r}$, where $u_r = 1$, and the remaining sequences are recursively defined as follows:*

k_i is the unique integer $0 \leq k_i < \alpha$ such that $k_i \equiv -u_i \mod \alpha$;
$u_{i-1} = u_i + 2(u_i + k_i)/\alpha$.

In the following, we will denote $u = u_0$.

Note that in the above definition, $u_i + k_i$ is always a multiple of α; hence u_{i-1} is indeed an integer.

For a polynomial h and sequence of polynomials H, we write $\mathrm{Red}(h, H)$ to denote the reduction of h by H w.r.t. $\prec_A$.More specifically, $\mathrm{Red}(h, H)$ is the remainder after performing multivariate division of h by H (see [20, Ch. 2, §3]). We are now in a position to define g^*. Let $g'_r = g$, and recursively define

$$g'_{i-1} = \mathrm{Red}(x_i^{k_i} g'_i, \{p_i, p_{i+1}, \ldots, p_r\}), \text{ for } i = r, r-1, \ldots, 1 \ .$$

We set $g^* = g'_0$, and $I_A = \langle P_{\mathcal{G}^*} \rangle$, where $P_{\mathcal{G}^*} = \{p_1, \ldots, p_r, g^*\}$. It is clear from the construction that I_A is a subideal of $\langle P_{\mathcal{G}} \rangle$. The following result guarantees that $P_{\mathcal{G}^*}$ is a FreeLunch system generating this subideal.

Proposition 7. *The polynomial system $P_{\mathcal{G}^*}$ is a FreeLunch system, where $\mathrm{LM}_{\prec_A}(g^*) = x_0^u$. Moreover, the variety of the associated ideal I_A contains all valid solutions of the underlying instance of `Anemoi`.*

Proof. By Lemma 5 we have $\mathrm{LM}(p_i) = x_i^\alpha$, so we need only show that $\mathrm{LM}(g^*)$ is a univariate monomial in x_0 to guarantee that $P_{\mathcal{G}^*}$ is a FreeLunch system. To this end, we will show by induction on descending i that $\mathrm{LM}(g_i') = (x_0 \cdots x_i)^{u_i}$, and $\mathrm{wt}(\mathrm{LM}(g_i')) > \mathrm{LM}(\mathrm{Mon}_2(g_i'))$.

For $i = r$, we have $g_r' = g = Q_r$, and the statement holds by Lemma 4. Suppose the hypothesis holds for a given i and consider $i-1$. If we denote $s_i = (u_i + k_i)/\alpha$, we have $\mathrm{LM}(x_i^{k_i} g_i') = (x_0 \cdots x_{i-1})^{u_i} x_i^{\alpha s_i}$. We now reduce this monomial by p_i. Write c for the leading coefficient of $x_i^{k_i} g_i'$. From Lemma 5, we have that the first two monomials in p_i have the same weight, while all other monomials have smaller weights. Moreover, the induction hypothesis assures that $\mathrm{Mon}_j(x_i^{k_i} g_i')$ will a have smaller weight than $\mathrm{LM}(x_i^{k_i} g_i')$ for $j \geq 2$. Hence,

$$\mathrm{LM}\left(x_i^{k_i} g_i' - c(x_0 \cdots x_{i-1})^{u_i} x_i^{\alpha(s_i-1)}) p_i\right) = (x_0 \cdots x_{i-1})^{u_i+2} x_i^{\alpha(s_i-1)},$$

following from the fact that $\mathrm{wt}((x_0 \cdots x_{i-1})^{u_i+2} x_i^{\alpha(s_i-1)}) = \mathrm{wt}(\mathrm{LM}(x_i^{k_i} g_i')) = \mathrm{wt}\left(\mathrm{LM}\left((x_0 \cdots x_{i-1})^{u_i} p_i\right)\right)$, and the weight of all other monomials are guaranteed to be strictly smaller. Repeating this process $s_i - 1$ times, we get $\mathrm{LM}(g_i') = (x_0 \cdots x_i)^{u_i + 2s_i}$, which proves the induction statement. Since this implies that $\mathrm{LM}(g^*) = \mathrm{LM}(g_0') = x_0^u$, it also concludes the proof of the first part of the proposition. The second part holds since I_A is a subideal of $\langle P_{\mathcal{G}} \rangle$, where the variety of the latter ideal will contain all solutions of `Anemoi`. $\square$

Ideal Degree. Based on experiments, the authors of `Anemoi` conjectured a tight upper bound on the ideal degree of one modeling of the CICO-problem to be $(\alpha+2)^r$ [13, Conjecture 2]. As I_A is a FreeLunch system, we have $D_{I_A} = \alpha^r u$, where we recall that u is an integer depending on r and α. As I_A is a subideal of $\langle P_{\mathcal{G}} \rangle$, we generally expect D_{I_A} to be strictly larger than $(\alpha+2)^r$. The following result proved in Appendix E, guarantees that D_I can at most differ by a factor close to α, which in practical instances will be a small constant.

Proposition 8. *Let u be as defined in Definition 15 for integers $r, \alpha \geq 1$. Then*

$$\left(\frac{\alpha+2}{\alpha}\right)^r \;\leq\; u \;\leq\; (\alpha+1)\left(\frac{\alpha+2}{\alpha}\right)^r - \alpha\ .$$

From experiments for $\alpha = 3$ and large r, we find $u \approx 2.1\,(5/3)^r$.

Complexity Analysis and Experimental Results. The FreeLunch system $P_{\mathcal{G}^*}$ consists of r polynomials of degree α and one polynomial of degree u. The algorithm of Section 3.2 has a complexity of $\tilde{\mathcal{O}}(\alpha^{r\omega} u)$. We plugged in the numbers for odd `Anemoi` ($\ell = 1$), see Table 6. We also ran experiments for `Anemoi` with $(\ell, \alpha) = (1, 3)$ and different number of rounds to verify the theory presented above. The results are presented in Table 7.

Security claim	$\alpha = 3$	$\alpha = 5$	$\alpha = 7$	$\alpha = 11$
128	118 (21)	156 (21)	174 (20)	198 (19)
256	203 (37)	270 (37)	307 (36)	358 (35)

Table 6: Expected complexity of `polyDet`. of the FreeLunch attack against full `Anemoi` instances over $\mathbb{F}_p$ and $\ell = 1$. Number of rounds in parentheses.

Number of rounds	Complexity of `polyDet`	Time (s) `sysGen`	`matGen`	`polyDet`	Memory (MB)
3	20	< 0.01	< 0.01	0.02	< 400
4	26	< 0.01	0.34	0.24	< 400
5	32	0.07	23.3	7.6	< 400
6	37	2.52	$2,127$	292	$2,863$
7	43	128	$156,348$	$10,725$	$42,337$

Table 7: Experimental results on `Anemoi` with $(\ell, \alpha) = (1, 3)$. `sysGen` is performed with Magma and refers to the generation of the polynomial system $P_{\mathcal{G}^*}$ from scratch, including the computation of $P_{\mathcal{G}}$.

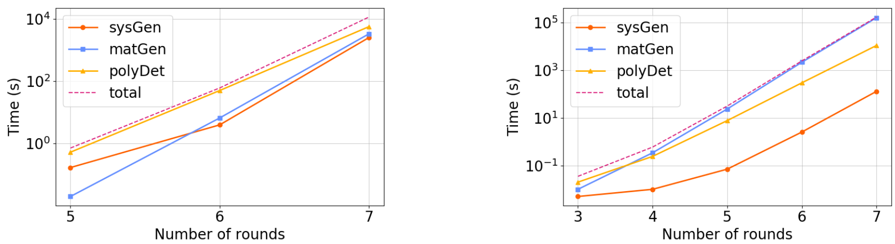
6 Conclusions

We have presented the FreeLunch approach, an algebraic attack particularly efficient against arithmetization-oriented permutations. We conclude this paper with some comments regarding our experiments as well the consequences of our results, in particular regarding the areas we believe worth investigating further.

6.1 Discussion on Experimental Results

Figure 5 depicts the runtimes of each step of our attack that we obtained experimentally when targeting `Griffin` and `Anemoi`. A first observation is that the running time of a full FreeLunch-based attack is hard to predict: there are three steps (`sysGen`, `matGen`, and `polyDet`), and we experimentally found situations where each of them was the slowest. The case of `sysGen` is a bit peculiar: using `SageMath`, Magma or Flint/NTL yields very different results and a deeper understanding seems out of our grasp. We nevertheless would argue (see Appendix A) that, should their implementations use similar tools, `sysGen` will always be of lower complexity than that of the rest of the attack.

Assuming that the dominating step is either `matGen` or `polyDet`, it then seems easy to extrapolate: as we can see in Figure 5, their logarithm increases linearly with the number of rounds. Even better: for `Griffin`, Equation (6) predicts that adding a round multiplies the complexity of `polyDet` by $\alpha^{\omega}(2\alpha + 1) \approx 109.5$, which closely matches our observations as $5727/50.79 \approx 112.7$. For `matGen`, we see that adding a round multiplies the time complexity by about 500. Extrapolating from this, an attack against full-round `Griffin` should take about $4.2 \cdot 10^{11}$s on a single CPU (around $13,000$ years), or around 2^{70} clock cycles at 2.3 GHz. Similarly, for `Anemoi` with $\mathbb{F}_p$ and $(l, \alpha) = (1, 3)$, adding a round multiplies the time complexity of `matGen` by about 75. Extrapolating gives respectively 2^{104} seconds (or 2^{135} clock cycles) and 2^{204} seconds (2^{235} clock cycles) for full-round `Anemoi` with 128 and 256 bits of security.



(a) `Griffin` complexity (from table 3).

(b) `Anemoi` complexity (from table 7).

Fig. 5: Experimental time complexity of our attacks on `Griffin` and `Anemoi`.

6.2 Preventing the FreeLunch Attack

Our attack breaks full-round instances of symmetric primitives built using state-of-the-art security arguments, which consequently must be revisited: one must learn how to prevent the relevant applicability of the FreeLunch approach.

At the Primitive Level. An obvious but perhaps costly countermeasure consists of simply adding more rounds. This is particularly tempting as we are able to tightly estimate the complexity of `polyDet`, a step which we have found to often be the most expensive in practice. Choosing a number of rounds high enough to prevent it would be a simple yet convincing argument. Primitive designers must also be mindful of "classical" tricks, i.e., symmetric cryptanalysis techniques (a priori) unrelated to root finding that can be used to enhance its efficiency. In the case of 12-branch `Griffin`, the fact that we can bypass 3 out of 10 rounds using some kind of subspace trail is a problem we deem worth studying.

At the Mode of Operation Level. The FreeLunch systems are multivariate, but a single variable (x_0) plays an inherently different role, meaning that they have a univariate *flair*. This makes them particularly well suited to CICO instances whereby a single output word has to be set to 0, but they will not work if more 0 are needed in the output. Thus, a simple countermeasure against the FreeLunch approach (and univariate ones) consists of forcing the capacity of the sponge to have at least two words set to 0, even if one word would a priori be enough.

6.3 Open Problems for Future Work

Time Taken by Polynomial Reductions. A roadblock in our complexity estimates is the number of operations needed to perform certain reductions of a polynomial modulo an ideal. This is crucial for understanding the complexity of the `matGen` step and, to a lesser degree, `sysGen` (c.f. Appendix A). A tighter estimate for these computations would greatly benefit our analysis: we would be able to figure out which step of our attack is the actual bottleneck without the need for experiments or assumptions, and designers could then be able to use fewer rounds to achieve a given security level against FreeLunch-based attacks. For instance, estimating the complexity of a reduction by a FreeLunch triangular system (Definition 9) would be a big step forward.

Other Custom Approaches. The FreeLunch approach is, to the best of our knowledge, the first "custom" root finding method designed specifically for use in symmetric cryptanalysis. There is, of course, no reason to believe that it is the only one possible, and we consider it a direction worth pursuing. As a first step, a multivariate variant of FreeLunch where several variables play the role of x_0, and where several words need to be set to 0, would be an interesting target.

Acknowledgements This work has been facilitated through the COSINUS associate team. The authors would like to thank Gaëtan Leurent for the helpful insights on the new dedicated Gröbner basis solving algorithm, and Pierre Briaud and Carlos Cid for the good discussions in the early stages of this work. The authors would also like to thank Vincent Neiger for the proof-reading and the discussions on the algorithmic aspects of the Gröbner basis theory. The research in this paper was supported in part by the French DGA. The work of Aurélien Bœuf, Axel Lemoine, and Léo Perrin was supported by the European Research Council (ERC, grant agreement no. 101041545 "ReSCALE"). Morten Øygarden has been supported by the Norwegian Research Council through the project qsIo2.

References

1. Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce (Nov 2001)
2. Albrecht, M.R., Cid, C., Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schofnegger, M.: Algebraic cryptanalysis of stark-friendly designs: Application to marvellous and mimc. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11923, pp. 371–397. Springer (2019). https://doi.org/10.1007/978-3-030-34618-8_13
3. Albrecht, M.R., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 191–219 (2016). https://doi.org/10.1007/978-3-662-53887-6_7

4. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 430–454. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_17
5. 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 **2020**(3), 1–45 (Sep 2020), https://tosc.iacr.org/index.php/ToSC/article/view/8695
6. 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
7. Ashur, T., Kindi, A., Meier, W., Szepieniec, A., Threadbare, B.: Rescue-prime optimized. Cryptology ePrint Archive, Paper 2022/1577 (2022), https://eprint.iacr.org/2022/1577
8. Ashur, T., Mahzoun, M., Toprakhisar, D.: Chaghri - A fhe-friendly block cipher. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022. pp. 139–150. ACM (2022). https://doi.org/10.1145/3548606.3559364
9. Bariant, A.: Algebraic cryptanalysis of full ciminion. Cryptology ePrint Archive, Paper 2023/1283 (2023), https://eprint.iacr.org/2023/1283
10. Bariant, A., Bouvier, C., Leurent, G., Perrin, L.: Algebraic attacks against some arithmetization-oriented primitives. IACR Transactions on Symmetric Cryptology **2022**(3), 73–101 (Sep 2022), https://tosc.iacr.org/index.php/ToSC/article/view/9850
11. Berthomieu, J., Neiger, V., Safey El Din, M.: Faster change of order algorithm for gröbner bases under shape and stability assumptions. In: Proceedings of the 2022 International Symposium on Symbolic and Algebraic Computation. pp. 409–418 (2022)
12. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. J. Symbolic Comput. **24**(3-4), 235–265 (1997), http://dx.doi.org/10.1006/jsco.1996.0125, computational algebra and number theory (London, 1993)
13. 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: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023. Lecture Notes in Computer Science, vol. 14083, pp. 507–539. Springer Nature Switzerland, Cham (2023)
14. Buchberger, B.: A theoretical basis for the reduction of polynomials to canonical forms. ACM SIGSAM Bulletin **10**(3), 19–29 (1976)
15. Buchmann, J., Pyshkin, A., Weinmann, R.P.: A Zero-Dimensional Gröbner Basis for AES-128. In: Robshaw, M. (ed.) Fast Software Encryption. Lecture Notes in Computer Science, vol. 4047, pp. 78–88. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
16. Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In: Peyrin, T. (ed.) Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9783, pp. 313–333. Springer (2016). https://doi.org/10.1007/978-3-662-52993-5_16

17. Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. J. Cryptol. **31**(3), 885–916 (2018). https://doi.org/10.1007/S00145-017-9273-9
18. Cantor, D.G., Kaltofen, E.: On fast multiplication of polynomials over arbitrary algebras. Acta Informatica **28**(7), 693–701 (1991)
19. Cosseron, O., Hoffmann, C., Méaux, P., Standaert, F.: Towards Case-Optimized Hybrid Homomorphic Encryption - Featuring the Elisabeth Stream Cipher. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13793, pp. 32–67. Springer (2022). https://doi.org/10.1007/978-3-031-22969-5_2
20. Cox, D., Little, J., OShea, D.: Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra. Springer Science & Business Media (2013)
21. Cox, D.A., Little, J.B., O'Shea, D.: Using Algebraic Geometry, Graduate Texts in Mathematics, vol. 185. Springer (1998). https://doi.org/10.1007/978-1-4757-6911-1
22. Dobraunig, C., Eichlseder, M., Grassi, L., Lallemand, V., Leander, G., List, E., Mendel, F., Rechberger, C.: Rasta: A cipher with low anddepth and few ands per bit. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10991, pp. 662–692. Springer (2018). https://doi.org/10.1007/978-3-319-96884-1_22
23. Duval, S., Lallemand, V., Rotella, Y.: Cryptanalysis of the FLIP Family of Stream Ciphers. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9814, pp. 457–475. Springer (2016). https://doi.org/10.1007/978-3-662-53018-4_17
24. Faugère, J.C., Mou, C.: Sparse FGLM algorithms. Journal of Symbolic Computation **80**, 538–569 (2017)
25. Faugère, Jean-Charles and Gaudry, Pierrick and Huot, Louise and Renault, Guénaël: Sub-cubic change of ordering for Gröbner basis: a probabilistic approach. In: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation. pp. 170–177 (2014)
26. Faugère, Jean-Charles and Gianni, Patrizia and Lazard, Daniel and Mora, Teo: Efficient computation of zero-dimensional Gröbner bases by change of ordering. Journal of Symbolic Computation **16**(4), 329–344 (1993)
27. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases (F_4). Journal of pure and applied algebra **139**(1-3), 61–88 (1999)
28. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In: Proceedings of the 2002 international symposium on Symbolic and algebraic computation. pp. 75–83 (2002)
29. Gilbert, H., Boissier, R.H., Jean, J., Reinhard, J.: Cryptanalysis of elisabeth-4. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part

III. Lecture Notes in Computer Science, vol. 14440, pp. 256–284. Springer (2023). https://doi.org/10.1007/978-981-99-8727-6_9
30. Gilbert, H., Peyrin, T.: Super-sbox cryptanalysis: Improved attacks for aes-like permutations. In: Hong, S., Iwata, T. (eds.) Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6147, pp. 365–383. Springer (2010). https://doi.org/10.1007/978-3-642-13858-4_21
31. Giorgi, P., Jeannerod, C.P., Villard, G.: On the complexity of polynomial matrix computations. In: Proceedings of the 2003 international symposium on Symbolic and algebraic computation. pp. 135–142 (2003)
32. Grassi, L., Hao, Y., Rechberger, C., Schofnegger, M., Walch, R., Wang, Q.: Horst meets fluid-spn: Griffin for zero-knowledge applications. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III. Lecture Notes in Computer Science, vol. 14083, pp. 573–606. Springer (2023). https://doi.org/10.1007/978-3-031-38548-3_19
33. Guido, B., Joan, D., Michaël, P., Gilles, V.: Cryptographic sponge functions (2011), https://keccak.team/files/CSF-0.1.pdf
34. Ha, J., Kim, S., Lee, B., Lee, J., Son, M.: Rubato: Noisy ciphers for approximate homomorphic encryption. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13275, pp. 581–610. Springer (2022). https://doi.org/10.1007/978-3-031-06944-4_20
35. Hart, W.B.: Flint: Fast library for number theory. Computeralgebra-Rundbrief: Vol. 49 (2011)
36. Hyun, S.G., Neiger, V., Schost, É.: Implementations of efficient univariate polynomial matrix algorithms and application to bivariate resultants. In: Proceedings ISSAC 2019. pp. 235–242. ACM (2019). https://doi.org/10.1145/3326229.3326272, https://github.com/vneiger/pml
37. Labahn, G., Neiger, V., Zhou, W.: Fast, deterministic computation of the hermite normal form and determinant of a polynomial matrix. Journal of Complexity **42**, 44–71 (2017)
38. Masure, L., Méaux, P., Moos, T., Standaert, F.: Effective and efficient masking with low noise using small-mersenne-prime ciphers. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 14007, pp. 596–627. Springer (2023). https://doi.org/10.1007/978-3-031-30634-1_20
39. Méaux, P., Journault, A., Standaert, F., Carlet, C.: Towards stream ciphers for efficient FHE with low-noise ciphertexts. In: Fischlin, M., Coron, J. (eds.) Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9665, pp. 311–343. Springer (2016). https://doi.org/10.1007/978-3-662-49890-3_13
40. Moenck, R.T.: Practical fast polynomial multiplication. In: Proceedings of the third ACM symposium on Symbolic and algebraic computation. pp. 136–148 (1976)

41. Roy, A., Steiner, M.J., Trevisani, S.: Arion: Arithmetization-oriented permutation and hashing from generalized triangular dynamical systems (2023), https://arxiv.org/abs/2303.04639
42. Szepieniec, A., Ashur, T., Dhooghe, S.: Rescue-prime: a standard specification (sok). Cryptology ePrint Archive, Paper 2020/1143 (2020), https://eprint.iacr.org/2020/1143
43. The PML team: PML: Polynomial Matrix Library (2023), version 0.3, https://github.com/vneiger/pml
44. The Sage Developers: SageMath, the Sage Mathematics Software System (2022), https://www.sagemath.org
45. V. Shoup, et al.: NTL: A library for doing number theory. https://libntl.org/
46. Von Zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge university press (2013)

A Computing a Reduced Gröbner Basis for $\langle P_{\mathcal{G}} \rangle$

As noted at the end of Section 3.4, we do not generate the polynomial system $P_{\mathcal{G}}$ directly in practice. Rather, we construct a related polynomial system iteratively while reducing as many monomials as possible along the way. More formally, for a polynomial h and an ordered sequence of polynomials H, we let $\mathrm{Red}(h, H)$ denote the operation of reducing h by H (according to a specified monomial order). That is, $\mathrm{Red}(h, H)$ is the remainder after performing multivariate division of h by H (see [20, Ch. 2, §3]). For a tuple of polynomials $\mathbf{h} = (h_1, \ldots, h_t)$, we write $\mathrm{Red}(\mathbf{h}, H) = (\mathrm{Red}(h_1, H), \ldots, \mathrm{Red}(h_t, H))$. Now fix a monomial order, and define $\mathbf{z}'_0 = \mathbf{z}_0$. We generate $\mathbf{p}'_i = (p'_{i,1}, \ldots, p'_{i,l_i})$ and the reduced states $\mathbf{z}'_i$ recursively as follows for $1 \le i \le r$ and $1 \le j \le l_i$.

$$p'_{i,j} = \mathrm{Red}\left(x_{i,j}^{\alpha_{i,j}} - \mathcal{L}_{i,j}(\mathbf{z}'_{i-1}), \{\mathbf{p}'_1 \ldots, \mathbf{p}'_{i-1}\}\right),$$
$$\mathbf{z}'_i = \mathrm{Red}\left(G_i(\mathbf{z}'_{i-1}, \mathbf{x}_i), \{\mathbf{p}'_1, \ldots, \mathbf{p}'_i\}\right),$$

where $\mathcal{L}_{i,j}$ is the polynomial from (3). Finally, we define

$$g' = \mathrm{Red}\left([G_r(\mathbf{z}'_{r-1}, \mathbf{x}_r)]_1, \{\mathbf{p}_1, \ldots, \mathbf{p}_r\}\right),$$

and write $P'_{\mathcal{G}} = \{\mathbf{p}'_1, \ldots, \mathbf{p}'_r, g'\}$. Since the construction of $P'_{\mathcal{G}}$ only differs from that of $P_{\mathcal{G}}$ by reductions with generators in the ideal $I_{\mathcal{G}} = \langle P_{\mathcal{G}} \rangle$ their ideals should, intuitively speaking, be identical. This intuition is confirmed by the following lemma.

Lemma 6. *For any fixed monomial order we have*

$$I_{\mathcal{G}} = \langle P_{\mathcal{G}} \rangle = \langle P'_{\mathcal{G}} \rangle \ .$$

Proof. For any polynomial h and polynomial sequence H, we can write the reduction operation as $\mathrm{Red}(h, H) = h + W$, for some polynomial $W \in \langle H \rangle$. Since the G_i's used in the construction of $\mathbf{p}'_i$ and $\mathbf{z}'_i$ are polynomial functions, one can show by induction that

$$p'_{i,j} = p_{i,j} + \langle \{\mathbf{p}_1 \ldots, \mathbf{p}_{i-1}\} \rangle, \qquad z'_{i,j} = z_{i,j} + \langle \{\mathbf{p}_1, \ldots, \mathbf{p}_i\} \rangle \tag{9}$$

holds for all $1 \le i \le r$ and $1 \le j \le l_i$. In particular, we have $g' = g + \langle\{\mathbf{p}_1, \ldots, \mathbf{p}_r\}\rangle$. Thus it is clear that $P_\mathcal{G}$ and $P'_\mathcal{G}$ generate the same polynomial ideal. $\square$

The following result relates $P'_\mathcal{G}$ and $P_\mathcal{G}$ when Proposition 6 holds. Recall that we write $d_{\le i} = d_1 \cdots d_i$, where $d_i = \deg(G_i)$.

Proposition 9. *Let $P_\mathcal{G}$ satisfy the condition of Proposition 6. Then constructing $P'_\mathcal{G}$ w.r.t. $\prec_\mathcal{G}$ is also a FreeLunch system. Moreover, replacing g' in $P'_\mathcal{G}$ with $g'/LC(g')$ yields the unique reduced Gröbner basis for $I_\mathcal{G}$ w.r.t. $\prec_\mathcal{G}$.*

Proof. By definition of polynomial division, we have $\mathrm{wt}(\mathrm{LM}(\mathrm{Red}(h, H))) \le \mathrm{wt}(\mathrm{LM}(h))$. Since $\mathrm{LM}(p_{i,j})'$ cannot be reduced by $\{\mathbf{p}'_1 \ldots, \mathbf{p}'_{i-1}\}$ under $\prec_\mathcal{G}$, it follows from (9) and the prior discussion that $\mathrm{LM}(p'_{i,j}) = \mathrm{LM}(p_{i,j})$. For the similar statement on $\mathrm{LM}(g')$, we note that the condition $\mathrm{LM}(g) = x_0^{d_{\le r}}$ can only hold if for every i there exist a j_i such that $\mathrm{LM}(z_{i,j_i}) = x_0^{d_{\le i}}$. By construction of $\prec_\mathcal{G}$, this monomial will not be reduced by $\{\mathbf{p}'_1 \ldots, \mathbf{p}'_{i-1}\}$. Again, it follows from (9) that $\mathrm{LM}(z'_{i,j_i}) = x_0^{d_{\le i}}$. In particular, $\mathrm{LM}(g') = \mathrm{LM}(g)$, hence $P'_\mathcal{G}$ is also a FreeLunch system.

For the last assertion, one observes from the way $p_{i,j}$ only depends on the variables $x_0, \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, x_{i,j}$ that

$$\mathrm{Red}(p'_{i,j}, P'_\mathcal{G} \setminus \{p'_{i,j}\}) = \mathrm{Red}(p'_{i,j}, \{\mathbf{p}_1 \ldots, \mathbf{p}_{i-1}\}) \ ,$$

holds for $\prec_\mathcal{G}$. Hence $P'_\mathcal{G}$ is already fully reduced, and replacing g' with $g'/\mathrm{LC}(g')$ makes all polynomials monic. $\square$

Remark 2. Recall from Proposition 2 that if H is a Gröbner basis for $\langle H\rangle$, then $\mathrm{Red}(h, H)$ does not depend on the order of the sequence H. It follows from Proposition 9 that if $P_\mathcal{G}$ satisfies the condition of Proposition 6, then the reductions in the construction of $p'_{i,j}$, $\mathbf{z}'_i$ and g' are independent of the order of the sequence $\{\mathbf{p}_1, \ldots \mathbf{p}_i\}$, w.r.t. $\prec_\mathcal{G}$.

Complexity of computing $P'_\mathcal{G}$. We are left with bounding the complexity of computing $P'_\mathcal{G}$, which will yield our estimate for the `sysGen` step. In the setting we will be interested in, this is expected to be dominated by the cost of applying the last round function G_r to compute g', and its reduction by $\{\mathbf{p}'_1 \ldots, \mathbf{p}'_r\}$. Our insight is that the reductions involved in the `sysGen` process are cheaper than the reductions required in `matGen`, since the reductions are performed on a smaller Gröbner basis; but we do not have a proof for such a statement. However, it is possible to bound the cost of the multiplications performed on the state $\mathbf{z}'_{r-1}$ when applying G_r. Let m denote the number of these multiplication, where we recall that m is typically small by design. We reduce by $\{\mathbf{p}'_1 \ldots, \mathbf{p}'_r\}$ after each multiplication, and will assume that this reduction is negligible compared to the cost of the multiplications themselves. Thus we have m multiplications of multivariate polynomials of maximal degree $d_{\le r}$ in x_0 and $\alpha_{i,j} - 1$ in $x_{i,j}$, for $1 \leq i \leq r$, $1 \leq j \leq l_i$. We can then use the Kronecker trick presented by Moenck [40, Section 3.4] to perform these multiplications in an efficient manner. In short, the Kronecker trick starts by transforming the multivariate polynomials to univariate polynomials. This allows us to perform the multiplication using an efficient univariate multiplication algorithm, before converting the result back to a multivariate polynomial. Moenck describes the algorithm and proves its correctness for any bound on the degree of each variable in both polynomials in the input of multiplication, but only gives a complexity estimate when all bounds are equal. It is, however, easy to verify that the complexity formula for the multivariate multiplication algorithm in our setting will be:

$$\tilde{\mathcal{O}}(d_{\leq r} \prod_{\substack{1\leq i\leq r\\ 1\leq j\leq l_i}} 2\alpha_{i,j}) \ ,$$

when applying either the Fast Fourier Transform, or Schönhage & Strassen's algorithm to perform the univariate multiplication [46, Chapter 8]. Repeating this m times yields our estimate for cost of multiplications in the `sysGen` step:

$$\tilde{\mathcal{O}}(md_{\leq r} \prod_{\substack{1\leq i\leq r\\ 1\leq j\leq l_i}} 2\alpha_{i,j}) \ .$$

In comparison, recall that the `polyDet` step of our analysis is expected by Theorem 1 to require

$$\tilde{O}(d_{\leq r}(\prod_{\substack{1\leq i\leq r\\ 1\leq j\leq l_i}} \alpha_{i,j})^{\omega})$$

operations in $\mathbb{F}$. Thus, when m remains small, we do not expect the multiplications in `sysGen` to be the bottleneck of the overall attack.

Use in Experiments. We implemented the Kronecker trick for the experiments we ran with the Flint library [35], using the NTL library [45] for the univariate multiplication; the mapping between flint and NTL polynomial representations was performed by hand. The multivariate multiplications performed for experiments with Magma and `SageMath` used their own built-in functionalities.

B Bypassing the first rounds of `Griffin`

The number of rounds that can be bypassed before we need to introduce x_1 depends on the number of branches. For $t \geq 12$ branches we can find an easily computable set of input states that allows to bypass the first three rounds of `Griffin`, so x_1 only appears in the fourth round. We explain in detail how this can be done for $t = 12$. After that it will become clear that three rounds can also be bypassed for $t \in \{16, 20, 24\}$, and how to determine how many rounds can be bypassed for $t < 12$.

Denote the input state to `Griffin` as

$$(a_0x_0 + b_0, a_1x_0 + b_1, a_2x_0 + b_2, \ldots, a_{10}x_0 + b_{10}, 0).$$

The a_i and b_j are constants in $\mathbb{F}$ that we now proceed to determine. Once the a_i and b_j are fixed the variable x_0 can be varied freely over $\mathbb{F}$, generating a set of input states for the CICO problem that all have constant input to the $x^{1/\alpha}$ function in the three first rounds. Figure 6 illustrates the evolution of one of the chosen input states up to the start of round 3.

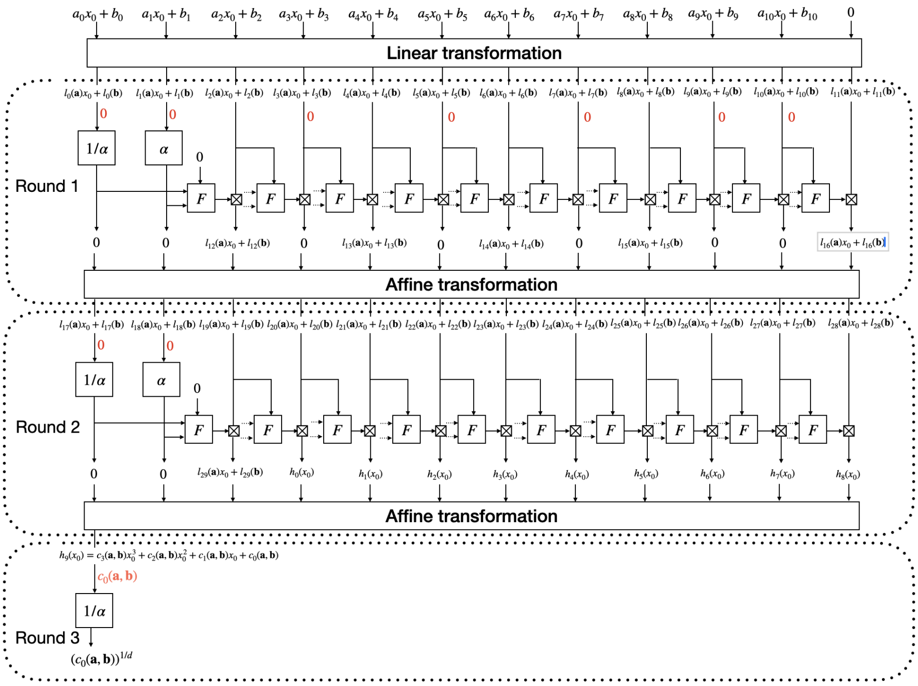


Fig. 6: Evolution of chosen set of input states to `Griffin` with 12 branches. Red values give conditions on the a_i and b_j such that the input of $x^{1/\alpha}$ in the third round becomes a known constant independent of x_0.

The values of a_i and b_j in Figure 6 can be determined as follows. After the initial linear transformation before the first round, all branches can be expressed as $l_i(\mathbf{a})x_0 + l_i(\mathbf{b})$ for $0 \leq i \leq 11$, where $l_i(\cdot)$ is a known linear combination. To get 0 on the branches indicated in Figure 6, the a_i's and b_j's need to satisfy the following linear equations

$$\begin{array}{ll} l_0(\mathbf{a}) = 0 & l_0(\mathbf{b}) = 0 \\ l_1(\mathbf{a}) = 0 & l_1(\mathbf{b}) = 0 \\ l_3(\mathbf{a}) = 0 & l_3(\mathbf{b}) = 0 \\ l_5(\mathbf{a}) = 0 & l_5(\mathbf{b}) = 0 \\ l_7(\mathbf{a}) = 0 & l_7(\mathbf{b}) = 0 \\ l_9(\mathbf{a}) = 0 & l_9(\mathbf{b}) = 0 \\ l_{10}(\mathbf{a}) = 0 & l_{10}(\mathbf{b}) = 0. \end{array}$$

With 0 on any two adjacent branches, the input to F will either be all 0, with $F(0,0,0)$ being equal to a constant, or the output of F will be multiplied with 0, making sure the value on the branch remains 0. This ensures that the algebraic expressions on the branches stay linear in $x_0, \mathbf{a}$ and $\mathbf{b}$ after the affine transformation at the start of the second round. The need to have input 0 to $x^{1/\alpha}$ and x^α in the second round gives four more linear constraints

$$\begin{array}{ll} l_{17}(\mathbf{a}) = 0 & l_{17}(\mathbf{b}) = \gamma_{17} \\ l_{18}(\mathbf{a}) = 0 & l_{18}(\mathbf{b}) = \gamma_{18}, \end{array}$$

where the γ_i are known constants.

Before the affine transformation in the second round, most branches will have cubic polynomials in x_0 as their values (the $h_i(x_0)$ in Figure 6). These are again linearly mixed in the affine transformation at the end of round two, producing the cubic polynomial

$$h_9(x_0) = c_3(\mathbf{a},\mathbf{b})x_0^3 + c_2(\mathbf{a},\mathbf{b})x_0^2 + c_1(\mathbf{a},\mathbf{b})x_0 + c_0(\mathbf{a},\mathbf{b})$$

on the first branch. We want to enforce that $c_3(\mathbf{a},\mathbf{b}) = c_2(\mathbf{a},\mathbf{b}) = c_1(\mathbf{a},\mathbf{b}) = 0$ such that the input to the $x^{1/\alpha}$ function in round three becomes a known constant independent from x_0. The expressions for the coefficients are cubic in the a_i and b_j, but note that all the polynomials $h_i(x_0)$ for $0 \le i \le 8$ are made as products of linear factors as

$$(l_i(\mathbf{a})x_0 + l_i(\mathbf{b}))(l_j(\mathbf{a})x_0 + l_j(\mathbf{b}))(l_k(\mathbf{a})x_0 + l_k(\mathbf{b})),$$

and that $h_9(x_0)$ is a sum of these. By calculating the coefficients for the x_0^3, x_0^2, and x_0 terms, we see that $c_3(\mathbf{a},\mathbf{b})$ is cubic in $\mathbf{a}$, but does not contain $\mathbf{b}$ at all. Similarly, $c_2(\mathbf{a},\mathbf{b})$ is quadratic in $\mathbf{a}$ and linear in $\mathbf{b}$ and $c_1(\mathbf{a},\mathbf{b})$ is linear in $\mathbf{a}$ and quadratic in $\mathbf{b}$.

We can now use the 9 linear equations in $\mathbf{a}$ introduced above to eliminate $a_2, \ldots, a_{10}$ from $c_3(\mathbf{a})$. This leaves c_3 as $c_3(a_0, a_1)$, a cubic expression in a_0 and a_1. Next we fix a_1 to an arbitrary non-zero value (to avoid the trivial solution $a_0 = \ldots = a_{10} = 0$) and solve for $c_3(a_0) = 0$ using a root-finding algorithm for univariate polynomials. With a_0 and a_1 fixed, all the other a_i gets fixed as well from the linear constraints from rounds 1 and 2.

Once all a_i have been found, $c_2(\mathbf{a},\mathbf{b}) = 0$ just becomes a linear equation in $\mathbf{b}$. Using this linear equation together with the 9 from above, we can eliminate $b_1, \ldots, b_{10}$ from the last coefficient $c_1(\mathbf{a},\mathbf{b})$. With all the a_i fixed, c_1 then just becomes $c_1(b_0)$, a quadratic expression in b_0 and we easily solve $c_1(b_0) = 0$. This determines all the values for the b_i.

With the a_i and b_j now fixed, we known that the input state from our chosen set will generate polynomials in x_0 of degree $6\alpha + 3$ on the branches at the start of round 4. We can then start the basic attack from there, adapting the weighted order of the variables accordingly. When the number of x_i-variables is reduced by 3 and with the degree of x_0 bounded to $6\alpha + 3$ until the fourth round, the dimension of the Gröbner basis ideal becomes much smaller, which again reduces the overall attack complexity significantly.

When there are more than 12 branches we can do the exact same trick as explained above. The only difference is that there will be more values of a_i and b_j that can be chosen arbitrarily when solving for $c_3(\mathbf{a}, \mathbf{b}) = c_2(\mathbf{a}, \mathbf{b}) = c_1(\mathbf{a}, \mathbf{b}) = 0$. When there are less than 12 branches, there is not enough degrees of freedom to make it through the third round. For $t = 8$ we can bypass the two first rounds, so x_1 only needs to be introduced in round 3, and for $t = 3, 4$ it is possible to bypass the first round and introduce x_1 in round 2.

C Solutions to the CICO problem with respect to `Griffin`

In this section we give explicit solutions to the CICO problem related to a `Griffin` permutation whose characteristics are specified below. Everything discussed here can be checked by the reviewers using the supplementary material provided in verify.zip.

Parameters of `Griffin` instances

- Prime number: $p = 28407454060060787$ (55 bits);
- Exponent: $\alpha = 3$;
- Number of rounds: $r \in \{5, 6, 7\}$;
- Number of branches: $t = 12$ (corresponds to 10 rounds in the real version);
- Parameters of the quadratic functions: $\delta_i = 4(i + 1),\ \mu_i = 7(i + 1)^2,\ i = 0, \ldots, r - 1$;
- Round constants[12] (up to 7 rounds):

[12] Our `Griffin` with $r \leq 7$ rounds will use constants $(\boldsymbol{c}_0, \ldots, \boldsymbol{c}_{r-2}, \boldsymbol{c}_6)$.

$$
\begin{aligned}
c_0 = (&24948861045225956, 21203603017242449, 5137804740880040,\\
&17203989140901077, 15884693750499599, 2024260346950 61,\\
&21925627314327927, 14915791625715646, 2270637844838412,\\
&19287066862534400, 23053628619630528, 20205234482325465)\\
c_1 = (&1953994697959081, 13694436956212586, 2244645965787647,\\
&20803493439220167, 13296675195272853, 18296898451764242,\\
&20376008308269607, 10239947264048958, 1116873941458788,\\
&19425600591729552, 20854422412323996, 10085561279368253)\\
c_2 = (&16905640598099854, 25230133406843513, 8957962046730991,\\
&14294289436907403, 10949906559535418, 28179662462119909,\\
&20848690834284278, 5962920227944130, 15129107418293752,\\
&6002695762195648, 6114627516150292, 22521669951514122)\\
c_3 = (&13008050022403386, 28091350245684079, 23189230572909585,\\
&8101795236077784, 3593606052472638, 11330866710107896,\\
&9840541134611106, 13915746912957553, 19822110644988410,\\
&24750875289653592, 25496607366081073, 2269647499797729)\\
c_4 = (&15770582149454036, 4472996328290429, 8197094411507273,\\
&14151116175893923, 19977244056516294, 22071066831282832,\\
&10912395968228633, 28293903648852908, 15600461636809584,\\
&16248565278833955, 23850742575902912, 12384888390231181)\\
c_5 = (&24731271392756981, 4234794164011219, 5709721189329773,\\
&23115678305163655, 11185048660199721, 21406367947811616,\\
&14929808901464726, 14209993563715217, 19373914823111461,\\
&12307896526346864, 16319890415782340, 19440350754040851)\\
c_6 = (&0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
\end{aligned}
$$

The round constants listed above were generated randomly using the following code in `SageMath`:

```
p = 28407454060060787
Fp = FiniteField(p)
R = 6
b = 12
set_random_seed(int.from_bytes(b"Griffin","little"))
RC = [random_vector(Fp, b) for _ in range(R)]
```

Solutions to the CICO problem

Solutions of the CICO problem for $r \in \{5, 6, 7\}$ are presented here. $\texttt{Griffin}_r$ denotes r-round $\texttt{Griffin}$.

$$
\begin{aligned}
\texttt{Griffin}_5(&0, 3490692521816093, 23601145558450866,\\
&12269607430774150, 9967688977125539, 6082447726448232,\\
&5654276540748670, 3202643372242143, 14009612926527540,\\
&18297056060918841, 3219981769736554, 1403954519004962)\\
= (&0, 5621630451433068, 19970363721022544,\\
&26741918912648639, 19340983417234439, 703450676999922,\\
&28208520610521445, 25436703150515389, 27364572020087999,\\
&24554342355067488, 12423823785589327, 9583319713824981)\\
\texttt{Griffin}_5(&0, 20508905520120247, 23936168820965785,\\
&23455122418677455, 17553236564762600, 3639182016432478,\\
&17868020430519088, 3713757452078792, 5123533774106823,\\
&15978370877576178, 18526890154199809, 7146019784219766)\\
= (&0, 398957666284762, 14158444455164092,\\
&23271855932022231, 21234778660794826, 254005063154066,\\
&20580021469733731, 20943402370356289, 5582293336098692,\\
&4611071270038675, 18302255515509522, 26476200309584297)\\
\texttt{Griffin}_6(&0, 15940424764849354, 15734551202904841,\\
&2252716448242183, 24464738475171520, 22965208929786740,\\
&9160692692879124, 10856186368261439, 8227155735266026,\\
&27520010287112407, 2695459349798501, 26535488466949249)\\
= (&0, 28084819548111304, 27440009447766981,\\
&18719426192325148, 15867588640423583, 21846560125606713,\\
&6096085299560336, 3230042720230543, 8933226700213982,\\
&16946922076875842, 4787108825372316, 24658818061833687)\\
\texttt{Griffin}_6(&0, 13453267118668680, 16821847582014700,\\
&16088365946216368, 5399506335051336, 27388226484505332,\\
&23347730487451583, 17984517745797377, 8860239602618916,\\
&21421104166559501, 2200106791585633, 4277034158946419)\\
= (&0, 15056970863769267, 16790578228924700,\\
&8672008243997393, 19887571512638539, 26391726423500753,\\
&20727079056917053, 20813391870109600, 20843251064205404,\\
&18441611384455618, 7490737291933204, 20355381094708976)
\end{aligned}
$$

$$
\begin{aligned}
\texttt{Griffin}_7(&0, 471901030567494, 19368389705049758,\\
&6207658171606065, 13611402711597287, 12429039749894833,\\
&22884233714242756, 18540641300363820, 25230426657954964,\\
&17547513396056919, 15871260254068404, 5181585218256522)\\
= (&0, 24311659110177349, 24245222836079936,\\
&1360999973748497, 25535756342488541, 10834064085568113,\\
&2487598456547217, 22567275155120838, 2042666706826108,\\
&694695024982032, 10782435475712749, 20264250160050251)
\end{aligned}
$$

D Bypassing the first round of Arion-π

We can use a trick similar to that used for `Griffin` to bypass the first round of `Arion`-π such that the variable x_1 is only first introduced in the second round. Unlike `Griffin`, this method can be applied to any number of branches in `Arion`-π. However, we can only bypass a single round.

Denote the input state to `Arion` as

$$(a_0x_0 + b_0, a_1x_0 + b_1, a_2x_0 + b_2, \dots, a_{t-2}x_0 + b_{t-2}, 0).$$

The a_i and b_j are constants in $\mathbb{F}$ that will be determined. Once the a_i and b_j are fixed, the variable x_0 can be varied freely over $\mathbb{F}$, generating a set of input states for the CICO problem that all have input 0 to the $x^{1/\alpha}$ function in the first round. Figure 7 illustrates the evolution of one of the chosen input states up to the start of round 2.

The values of a_i and b_j can, in general, be determined as follows. After the initial matrix multiplication, all branches can be expressed as $l_i(\mathbf{a})x_0 + l_i(\mathbf{b})$ for $0 \le i \le t-1$, where $l_i(\mathbf{a})$ and $l_i(\mathbf{b})$ are known linear combinations. To get 0 on the last $t-2$ branches, the a_i's and b_j's need to satisfy the following linear equations

$$
\begin{aligned}
l_2(\mathbf{a}) &= 0 \qquad & l_2(\mathbf{b}) &= 0\\
&\vdots & &\vdots\\
l_{t-1}(\mathbf{a}) &= 0 & l_{t-1}(\mathbf{b}) &= 0.
\end{aligned}
$$

With $2t-4$ equations on $2t-2$ variables, these constraints leave two degrees of freedom for the variables in $\mathbf{a}$ and $\mathbf{b}$. Naively, one could think of additionally imposing the constraints $l_1(\mathbf{a}) = 0$ and $l_1(\mathbf{b}) = 0$ such that only the first branch is nonzero and the degree on x_0 is further reduced. However, the unique solution to this system is the trivial solution $(\mathbf{a}, \mathbf{b}) = (0, 0)$, which is not of interest. Thus, we avoid this by instead imposing arbitrary conditions for two variables a_k and b_l (as long as a_k is set to be a non-zero value to avoid the trivial solution). With a_k and b_l fixed, all the other variables get fixed, too, from the previous linear constraints. For simplicity, one could fix the values $a_0 = 1$ and $b_0 = 0$, leading to an input state of the form $(x_0, a_1x_0, a_2x_0, \dots, a_{t-2}x_0, 0)$, where all a_i's are fixed.

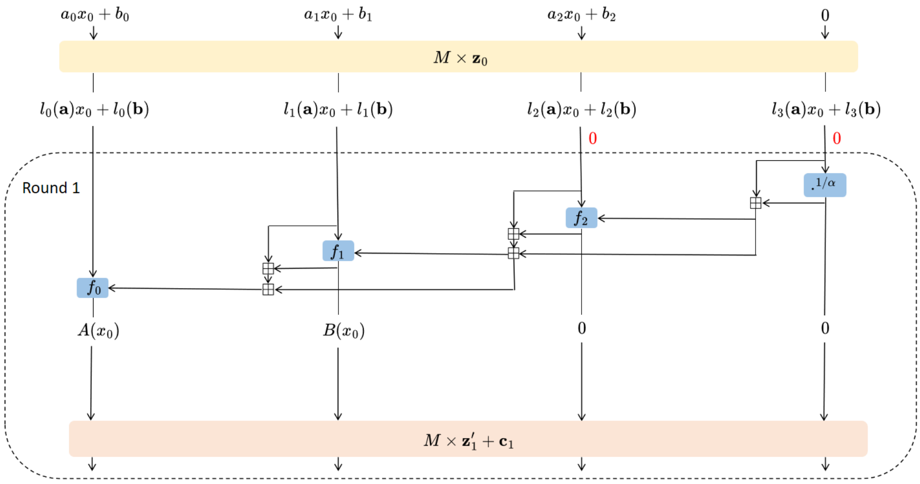


Fig. 7: Evolution of chosen set of input states to `Arion`-π with 4 branches. Red values give conditions on the a_i and b_j.

With 0 on the last $t-2$ input branches, the output of the non-linear layer of `Arion`-π will be of the form $(A(x_0), B(x_0), 0, \ldots, 0)$, where A and B are polynomials in x_0 of degree $3e$ and e, respectively. Thus, the input state from our chosen set will generate polynomials in x_0 of degree $3de$ on the branches after the affine transformation in round 1. We can then start the basic attack from there, adapting the weighted order of the variables accordingly. When the number of x_i-variables is reduced by 1 and with the degree of x_0 bounded to $3e$ until the second round, the dimension of the Gröbner basis ideal becomes smaller, which again reduces the overall attack complexity.

E Proof of Proposition 8

Proof. Recall from Definition 15 that u is defined as $u = u_0$ through the sequence $\{u_i\}_{0\le i\le r}$. To simplify the exposition, we will work with the sequence $\{v_i\}_{0\le i\le r}$ defined by $v_0 = 1$, and

$$v_{i+1} = v_i + 2\left\lceil \frac{v_i}{\alpha} \right\rceil, \text{ for } 0 \le i < r.$$

Note that $v_i = u_{r-i}$ and, in particular, $v_r = u$. Define two more integer sequences $\{a_i\}_{0\le i\le r}$ and $\{b_i\}_{0\le i\le r}$ defined by $a_0 = b_0 = 1$ and for $0 \le i < r$

$$a_{i+1} = \frac{\alpha+2}{\alpha} a_i, \qquad b_{i+1} = \frac{\alpha+2}{\alpha} b_i + 2.$$

As a first step, we will prove $a_i \le v_i \le b_i$. This is clearly true for $i = 0$. Supposing it holds up to some i, then using the identity $x \le \lceil x \rceil < x + 1$, we have

$$\frac{\alpha+2}{\alpha} v_i \le v_i + 2\left\lceil \frac{v_i}{\alpha} \right\rceil < \frac{\alpha+2}{\alpha} v_i + 2,$$

Thus, using the induction hypothesis and the definitions of $\{a_i\}$ and $\{b_i\}$,

$$a_{i+1} \leq v_{i+1} \leq b_{i+1}.$$

Observe that $a_i = \left(\frac{\alpha+2}{\alpha}\right)^i$, which proves the left-hand side of the inequality in the proposition. For the right-hand side we note that $\{b_i\}$ can be written as $b_i = (\alpha+1)\left(\frac{\alpha+2}{\alpha}\right)^i - \alpha$, when $i \geq 1$. Indeed, this can be verified for $i = 1$. Supposing it holds up to some i, then

$$\begin{aligned} b_{i+1} &= \frac{\alpha+2}{\alpha} b_i + 2 \\ &= (\alpha+1)\left(\frac{\alpha+2}{\alpha}\right)^{i+1} - \frac{\alpha(\alpha+2)}{\alpha} + 2 \\ &= (\alpha+1)\left(\frac{\alpha+2}{\alpha}\right)^{i+1} - \alpha, \end{aligned}$$

and the bounds on u stated in Proposition 8 follows. □

F FreeLunch Systems for `XHash8`

Description of `XHash8`. `XHash8` is an SPN with nonlinear S-boxes, multiplication by a fixed MDS matrix M, and addition by round constants C_i. Its state contains $t = 12$ elements in $\mathbb{F}_p$ where $p = 2^{64} - 2^{32} + 1$. The rate is fixed to 8 and capacity 4. There are 3 rounds in total, and each round consists of 3 steps, for a total of 9 steps (plus the initial affine layer (I)). With the cipher state denoted as $\mathbf{z} = (z_0, \ldots, z_{11})$, one round of `XHash8` is constructed from the following functions (excluding $(P3)^{(k)}$ which is specified below):

$$\begin{aligned} (I) &: \mathbf{z} \mapsto M \times (C_0 + \mathbf{z}), \\ (F)^{(k)} &: \mathbf{z} \mapsto C_{3k} + M \times (z_0^7, \ldots, z_{11}^7), \\ (B')^{(k)} &: \mathbf{z} \mapsto C_{3k+1} + (z_0^{\frac{1}{7}}, z_1, z_2^{\frac{1}{7}}, z_3^{\frac{1}{7}}, z_4, z_5^{\frac{1}{7}}, z_6^{\frac{1}{7}}, z_7, z_8^{\frac{1}{7}}, z_9^{\frac{1}{7}}, z_{10}, z_{11}^{\frac{1}{7}}). \end{aligned}$$

The last step of a round, $(P3)^{(k)}$, consists of naturally mapping $\mathbf{z}$ to a state of four elements in a cubic expansion $\mathbb{F}_{p^3}$, denoted $(S_{0,1,2}, S_{3,4,5}, S_{6,7,8}, S_{9,10,11})$, and then computing $S_{i,i+1,i+2}^7$ and mapping the result back to $\mathbb{F}_p$. After that, like with $(F)^{(k)}$, an MDS layer is applied, and the round constant C_{3k+2} is added. Effectively, $(P3)^{(k)}$ is equivalent to mapping each z_{3q+r} to a multivariate polynomial of degree 7 in $z_{3q}, z_{3q+1}, z_{3q+2}$ (see also the detailed description in [6, Appendix A]), which is the way we modelize it.

The steps are applied in the following order, from left to right:

$$(I)\ (F)^{(1)}(B')^{(1)}(P3)^{(1)}(F)^{(2)}(B')^{(2)}(P3)^{(2)}(F)^{(3)}(B')^{(3)}(P3)^{(3)}.$$

One round preceded by (I) is shown in figure 8, taken from [6].

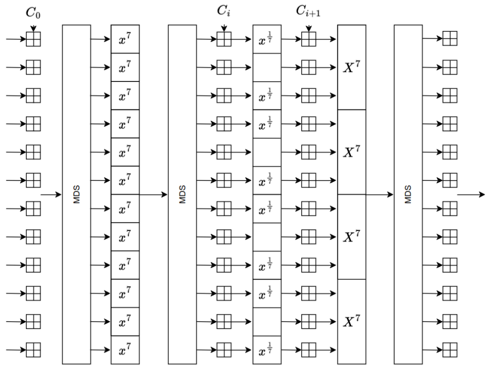


Fig. 8: Round i of `XHash8` preceded by an (I) step: $(I)(F)^{(i)}(B')^{(i)}(P3)^{(i)}$.

FreeLunch System for `XHash8` Our resolution allows us to solve the CICO problem on one branch. However, since the size of one branch is roughly 64 bits, this CICO problem could simply be solved by making 2^{64} queries to the permutation, for which our solving algorithm does not give us an advantage. On top of that, the real capacity of `XHash8` is $c = 4$ for a security claim of 128 bits. Rather than claiming a full attack on `XHash8`, we show a special case where a FreeLunch system can be easily extracted. However, the later solving steps, in particular the `polyDet` step, will still have a very high complexity.

Following the construction of FreeLunch systems from Section 3 we define the initial state as $\mathbf{z}_0 = (x_0, 0, \ldots, 0)$ and add a new variable $x_{i,j}$ for $0 \leq i \leq 2$ and $j \in \{0, 2, 3, 5, 6, 8, 9, 11\}$ after every $(\cdot)^{1/7}$. All other nonlinear operations can be represented as polynomials of degree 7, fixing the weights of the introduced variables to

$$\mathrm{wt}(x_0) = 1, \qquad \mathrm{wt}(x_{i,j}) = 7^{2i} + 1 \ .$$

We end up with 25 polynomials in 25 variables; 24 of these polynomials have $x_{i,j}^7$ as leading monomials and the last polynomial has $x_0^{7^6}$ as a leading monomial. The coefficient of the $x_0^{7^6}$-term in the last polynomial will be non-zero with a very high probability, ensuring we get a FreeLunch system, with $D_1 = 7^{24}$ and $\alpha_0 = 7^6$.

Complexity of solving the system. We can solve the system using the algorithm described in Section 3. The complexity of `matGen` is hard to estimate precisely. The complexity of the `polyDet` step is:

$$\mathcal{O}(D_1^{\omega} \alpha_0 \log(\alpha_0)^2) \approx 2^{240}$$

when $\omega = 2.81$. Note that this is significantly higher than 2^{64}, the brute force complexity for solving this CICO problem.