Programming Equation Systems of Arithmetization-Oriented Primitives with Constraints

Mengyu Chang, Kexin Qiao, Junjie Cheng, Changhai Ou and Liehuang Zhu

Abstract: Arithmetization-Oriented (AO) cryptographic algorithms operate on large finite fields. The most threatening attack on such designs is the Gröbner basis attack, which solves the equation system encoded from the cryptanalysis problem. However, encoding a primitive as a system of equations is not unique, and finding the optimal one with low solving complexity is a challenge. This paper introduces an automatic tool that converts the CICO problem into a Mixed-Integer Quadratic Constraint Programming (MIQCP) model, using integer variables and constraints to track degree propagation and determine variable introduction points. The optimal MIQCP solution provides the lowest solving complexity. We build models for Griffin, Anemoi, and Ciminion permutations to cover modules comprehensively. Experiments show reduced Gröbner basis attack complexity, lower than designers' bounds for small numbers of rounds, e.g. up to 8 rounds for Griffin.This tool can be used for security evaluation against Gröbner basis attack in new designs.

Key words: Gröbner basis , automatic cryptanalysis , CICO , MIQCP , Griffin;

1 Introduction

Advanced protocols such as homomorphic encryption, multi-party computation, and zero-knowledge proofs desire cryptographic primitives with new features. In such applications, traditional encryption, decryption, and hash calculation not work. Instead, an arithmetic equivalence of the primitives is calculated. Such designs are called arithmetization-oriented (AO) designs. The number of non-linear operations in the arithmetic path leverages the implementation efficiency of the protocols, so such primitives are designed to have a low multiplicative depth. Many of the current designs use operations on large finite fields with characteristic 2 or a large prime and use simple algebraic calculations such as power maps and inverse as non-linear components, such as the MPC-friendly MiMC [1], GMiMC [2], HADESMiMC [3], Ciminion [4], and ZK-friendly Vision/Rescue [5], Grendel [6, 7], POSEIDON [8], Griffin [9], Anemoi [10] etc. . The designs start from a permutation, then embedded in a construction like Sponge [11] or a dedicated one.

The security of AO primitives defined on large prime fields is of special interest since the traditional cryptanalysis based on statistically distinguishing properties (differential, linear, and their variants) is paralyzed. Thus, algebraic attacks pose significant threats. A common approach for algebraic attack is to build a system of equations on unknowns in the cryptographic primitive with constrained-input constrained-output (CICO), and then to solve the system with existing algorithms like Gröbner basis, XL, and Sylvester resultant algorithms. In designing AO primitives, designers estimate the lower bound of the complexity of solving the equation system built for a certain number of rounds and add some redundant rounds to ensure

• Mengyu Chang, Kexin Qiao, Junjie Cheng and Liehuang Zhu are with the Department of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China. E-mail:1361587350@qq.com (M.C.); qiaokexin0327@163.com (K.Q.); junjie.cheng@outlook.com (J.C.); liehuangz@bit.edu.cn (L.Z.)

• Changhai Ou is with the Department of Cyberspace Science and Technology, Wuhan University, Wuhan 430072, China. E-mail:ouchanghai@whu.ac.cn

∗ Corresponding author: Kexin Qiao, Email: qiaokexin0327@163.com.

Manuscript received: year-month-day; accepted: year-month-day

a security level. Over-estimating an attack's complexity may suggest an insufficient number of rounds. There are multiple ways of encoding a primitive as a system of equations, varying in number and positions of variables, number of equations, and degrees, resulting in different solving complexity. If the freedom in building equation systems was not fully explored, advanced algebraic techniques might break the security claims [12,13]. The motivation of this work is to fully explore the freedom in building equation systems to obtain the lowest algebraic attack complexity for AO primitives.

Another motivation for this work is to incorporate algebraic analysis into the automatic toolbox that has gained increasing popularity in recent years. Searching for statistical distinguishers for symmetric ciphers used to be tedious work until the problem was converted to mixed-integer linear programming (MILP) or boolean satisfactory (SAT/SMT) problem. Up to now, the MILP-based toolbox for cryptanalysis includes differential and linear cryptanalysis [14–17], impossible differential attack [18], boomerang attack [19–21], cube attack [22–24], Demirci-Selçuk MitM attack [25], differential-linear attack [26,27] and preimage attack on AES-like hash functions [28–30] etc.. And there are toolboxes[31] developed to make the security evaluation more convenient. Our goal is to model the search for the optimal way of constructing an equation system into a constraint programming problem, with the optimal solution indicating the equation system with the optimal solving complexity.

Our contribution. This paper proposes a Mixed-Integer Quadratic Constraint Programming (MIQCP)-based tool to find the optimal way to build equation systems with respect to Gröbner basis attack for AO primitives. The model can handle the under-determined CICO problems. We start by parsing the arithmetic path by labeling three types of branches called upstream/downstream/pending for arithmetic modules to reflect the computation direction that decides how the degree propagates. Intermediate branches with multiple labeled ends are also classified into modules. For intermediate positions, integer variables representing whether a variable and a constant should be set here, the degree of this position, and the degree of the equation if built here, etc. ., are set. Then constraints among these variables regarding each arithmetic module and branching module are introduced. The objective is to minimize the sum of the degree of equations. To show the wide applicability of the automatic tool, we build models for Griffin [9], Anemoi[10], Ciminion [4] that cover most module types for AO primitives. With the solving time of the MIQCP model limited to 12 hours, we proved that there can be equations systems with Gröbner basis attack complexity even lower than the loose lower bound given by designers for a small number of rounds. This happens for 8 rounds of Griffin.

In addition to automaticity, this tool has the advantage of handling under-determined CICO issues. Sponge structured hash function allows usage where the constrained degrees of freedom in input and output are strictly smaller than the total degrees of freedom of the state. For a preimage attack, only one preimage is required to be a valid attack, so the remaining degrees of freedom can be consumed at intermediate positions of the arithmetic process and still guarantee one solution in the average sense. More importantly, the technique of allowing the degrees of freedom to be consumed at intermediate positions can effectively reduce the degree of equations constructed and, hence, the attack complexity. However, little existing literature has been devoted to exploring the complexity of such under-determined CICO problems. This is mainly because it is difficult to manually explore the optimal constrained positions in addition to those in the input and output. The method proposed in this paper can handle this problem naturally.

The tool presented in this paper can be used to estimate the complexity of the Gröbner basis attack more accurately, especially in designing new AO primitives. The interface of our model to the Gröbner basis attack is simply the number of variables and the sum of degrees of equations. We treat the process of obtaining the complexity of the Gröbner basis attack given the number of variables and the sum of degrees of equations as a black-box call. Works [32,33] on how to accurately estimate the complexity of Gröbner basis attack promotes the black-box oracle and, therefore, the overall estimation of our approach.

Organization. Section 2 presents preliminaries on Sponge structured AO hash functions, CICO problems, and the Gröbner basis attack. Section 3 gives an overview of the model to link up the remaining sections. Section 4 describes how to parse a given AO primitive by dividing it into arithmetic modules and branching

modules, and the variables that will be set. This is a preparation for building the model. Section 5 and Section 6 present how to add constraints for arithmetic modules and branching modules respectively. Section 7 gives results on Griffin, Anemoi, and Ciminion algorithms. We conclude the paper in Section 8.

2 Preliminaries

Notations. q is an n-bit large prime. The width of the state of a cryptographic primitive is t branches, or say tn-bits. We use lowercase letters to represent variables in the prime field $\mathbb{F}_q$, e.g. $x_i \in \mathbb{F}_q$. We use bold letters to represent vectors on the prime field $\mathbb{F}_q$, e.g. $\mathbf{x} \in \mathbb{F}_q^t$. The operation units of an AO primitive are elements in $\mathbb{F}_q$. $\mathcal{M}$ is the MIQCP module that will be built. equSys is the equation system constructed for the AO primitive. $Condition1 \Rightarrow Condition2$ represents indication constraint in constraint programming, meaning that if *Condition* 1 is satisfied there must be *Condition* 2.

2.1 Sponge construction with AO permutation

To design AO hash functions, a permutation can be embedded in Sponge construction [11]. As it is shown in Figure 1, the state size is split into c inner branches referred to as capacity and r outer branches referred to as rate. Initial inner branches are set to a constant, and the message blocks after padding are absorbed into the outer branches after each permutation. A digest of size h-branch is squeezed from the outer part after all blocks are processed. It has been proved that a Sponge construction embedded with a random permutation is indistinguishable from a random oracle up to $2^{n\cdot\min\{c/2,h\}}$ queries [34].

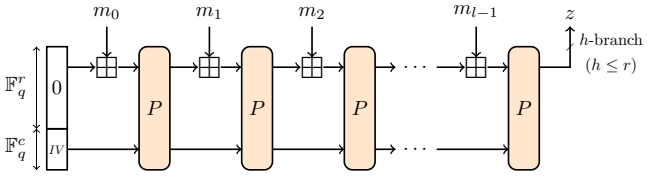


Fig. 1 Sponge construction with AO permutation P.

Round permutations of Griffin [9], Anemoi[10], and Ciminion [4] are shown in Figure 2. Components in these round permutations include linear operations defined by MDS matrixes, additions, multiplications, non-linear univariate polynomials (e.g. Sbox defined by power maps), and non-linear multivariate polynomials. The multivariate polynomial functions used in Griffin can be simplified to $G_1(x,y) = (c_0^1 x+y)^2 + c_1^1(c_0^1 x+y) + c_2^1$ and $G_2(x,y,z) = (c_0^2 x + y + z)^2 + c_1(c_0^2 x + y + z) + c_2^2$ where c_is are round related constants.

2.2 CICO problem

The CICO problem [35] oriented for AO primitives can be stated as follows.

Definition 1 (CICO Problem) Let $F : \mathbb{F}_q^t \to \mathbb{F}_q^t$ be a permutation on t n-bit branches. Solving the CICO problem is to find a couple (x, y) with $y = F(x)$ and t_0 branches of x fixed and t_1 branches of y fixed. When $t_0 + t_1 < t$, we say the CICO problem is under-determined. Otherwise, when $t_0 + t_1 = t$, we say the CICO problem is well-determined. When $t_0 + t_1 > t$, there may be no solutions.

There is one expected solution for a well-determined CICO problem. For an under-determined CICO problem, there are $(t - (t_0 + t_1))$ degrees of freedom in the system, so there are $2^{n(t-(t_0+t_1))}$ expected solutions.

The Sponge construction allows under-determined CICO usage even at the desired security level. For t_0 branches being the capacity part and t_1 branches being the extracted hash from the outer part of the state, solving the CICO problem is equivalent to a preimage attack. To ensure s-bit security, there should be no algorithm with an expected complexity smaller than 2^s to solve the CICO problem. The complexity of a general preimage attack is $2^{n\min\{t_0/2,t_1\}}$, and complexity of a general collision attack is $2^{n\cdot\min\{t_0/2,t_1/2\}}$. Some AO primitives claim security of $s = n/2$ bit (detailed in Section 7), so it is allowed that $t_0 = t_1 = 1$. Other concrete instances are the members of the SHA3 family [36] where $2d$-bit capacity and d-bit hash are constrained, where d can take the values 224, 256, 384, 512, and the state size is 1600-bit large ($2d + d < 1600$). To build a well-determined system of equations, i.e. systems with as many equations as variables, $(t - t_0 - t_1)$

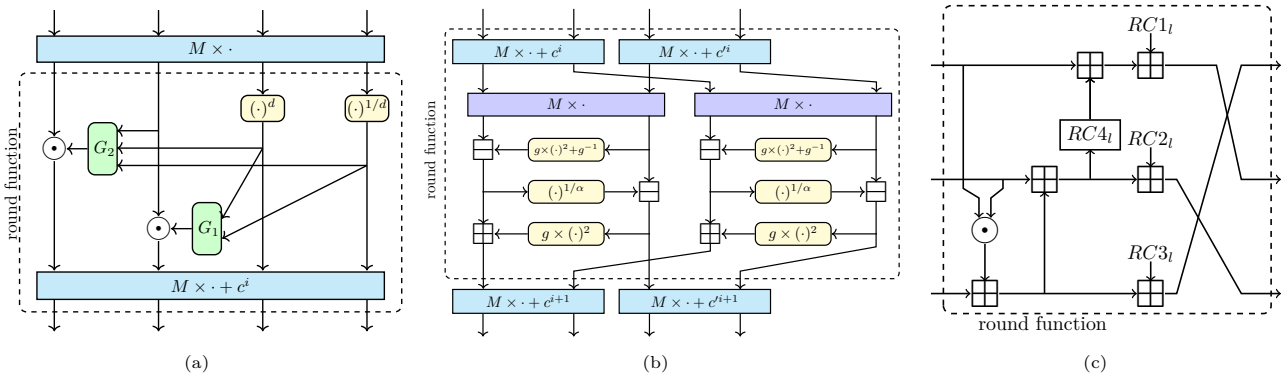


Fig. 2 Round functions of AO primitives. (a) Griffin. (b) Anemoi. (c) Ciminion.

intermediate positions can be set to constants to decrease the equation degrees.

When a hash function has only one message block to process, the CICO problem on the hash function is a CICO problem on a permutation, which is the case we focus on in this paper.

2.3 The Gröbner basis attack

The Gröbner basis method is used for solving well-determined multivariate systems. n_v is the number of variables and n_e is the number of equations, denote the system as $P_i(\mathbf{x}) = 0$, for $i = 0, \cdots, n_e - 1, \mathbf{x} = (x_0, x_1, \cdots x_{n_v-1}) \in \mathbb{F}_q^{n_v}$. The degree of polynomial P_i is d_i. The main technique is to calculate the Gröbner basis[37] of the ideal $\mathcal{I} = \langle P_0, \cdots, P_{n_e-1} \rangle$. The Gröbner basis G of $\mathcal{I}$ under the lexicographic order contains polynomials whose head monomial is a power of x_i for all variables. Thus the solution of the system is easy to obtain by solving the univariate polynomial on the last variable x_{n_v-1} in G, and then substituting it in the polynomial with the last two variables (x_{n_v-2}, x_{n_v-1}) and solve the equation for x_{n_v-2} and continue the process iteratively until all variables are found. However, computing the Gröbner basis under the lexicographic order is very expensive. A moderate way to do the calculation is to use the following three steps.

(1) Use F5 algorithm [38] to compute a the Gröbner basis under grevlex order. Under the hypothesis that the system is regular, the complexity of this step is estimated as

$$O\left(\binom{D_{reg} + n_v}{n_v}^{\omega}\right) \tag{1}$$

where the degree of regularity is

$$D_{deg} = 1 + \sum_{i=0}^{n_e-1} (d_i - 1), \tag{2}$$

and $2 \leq \omega < 3$ is a constant representing the complexity of solving linear equations systems. In this paper, we set ω to 2 in all complexity calculations.

(2) Convert the Gröbner basis deduced in the above step into lexicographic order using the FGLM algorithm [39]. The complexity of this step is estimated as

$$O(n_v d^3),$$

where d is the degree of the ideal $\mathcal{I}$.

(3) Solve the univariate polynomials iteratively deduced from the Gröbner basis in lexicongraphic order.

A lower bound of the complexity of the first step is used as a metric to guarantee the difficulty in solving the well-determined multivariate system. As long as the number of variables n_v and the number of equations n_e are known, the complexity can be estimated.

2.4 MIQCP

The Mixed-Integer Quadratic Constraint Program (MIQCP) is a paradigm that aims to optimize an objective function under quadratic constraints exerted on variables. Though the objective function is allowed to be quadratic, in our model it is linear. The quadratic constraints in our model come from two types of expressions. The first type is explicit - one variable is the product of the other two variables. The second type is the indication constraint in the form that "*Condition* 1 $\Rightarrow$ *Condition* 2". The indication constraints are converted to quadratic conditions implicitly by the off-the-shelf solver.

3 Overview

The key features in deciding the complexity of the Gröbner basis attack are the number of variables and the degree of equations in the equation system. To explore the freedom in constructing equations, every intermediate position along the arithmetic path is allowed to introduce new variables. Once a new variable is introduced, an equation that makes it equal to an existing expression should be added to consume the degree of freedom it brings in. If no variable is introduced at a position, then this position is expressed by existing variables.

The MIQCP model $\mathcal{M}$ captures where to introduce variables and how the degrees propagate through the arithmetic path that decides the degree of equations. The arithmetic path is parsed modularly round by round. We divide the path into arithmetic modules and the branching modules. The arithmetic modules are the operations used in the primitive. The branching modules are the branches that connect the arithmetic modules. Variables and constraints for each module are added to the model $\mathcal{M}$. The flow of building the model $\mathcal{M}$ for the Gröbner basis attack is shown in Figure 3. The right side is the process of parsing the round permutation, and the left side is the process of building the model accordingly. Details are illustrated in the corresponding sections.

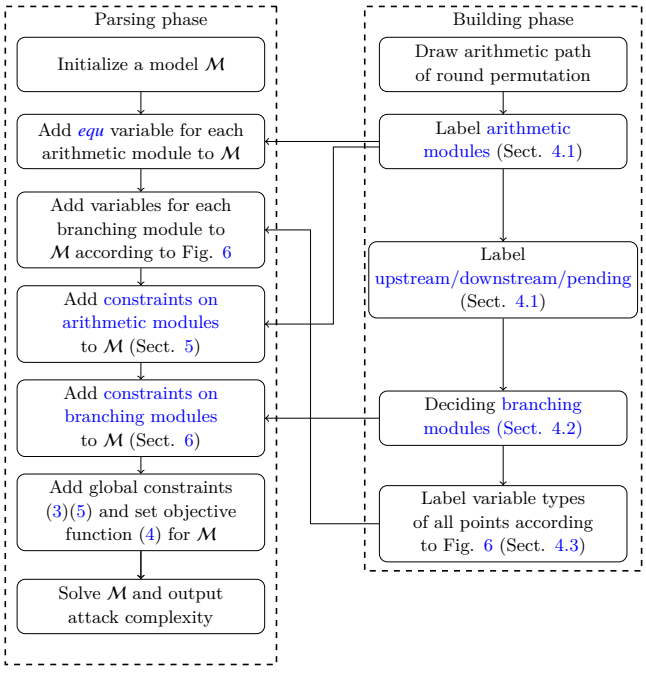


Fig. 3 Workflow for building the MIQCP model $\mathcal{M}$.

We mention the most critical variable types and the three global constraints to complete the modeling flowchart. The var_i^rs indicate whether a variable in equSys would be introduced in the i-th position in the r-th round. Similarly, the con_i^rs are the indicator variables for constants. The equ_j^r is the degree of the j-th equation that would be introduced in the r-th round. The complexity of the Gröbner basis attack estimated in Eq. (1) is determined by the number of variables and the sum of equation degrees. To avoid the hassle

of modeling combinatorial numbers, we fix the number of variables to a constant n_v by setting

$$\sum_{r,i} var_i^r = n_v, \tag{3}$$

and obtain the minimum sum of equation degrees by setting the objective function to

$$\min obj = \sum_{r,j} equ_j^r, \tag{4}$$

and then calculate combinatorial numbers offline. Then, compare the complexity under different numbers of variables to get the global minimum.

To solve a well-determined equation system, we require that the number of constants in the model should equal to the number of branches of the permutation:

$$\sum_{r,i} con_i^r = t. \tag{5}$$

4 Parsing the Arithmetic Path

Given an AO primitive, an automatic Gröbner basis attack begins by parsing the arithmetic path. We divide the path into arithmetic modules and the branching modules. The arithmetic modules are the operations used in the primitive. The branching modules are the branches that connect the arithmetic modules.

4.1 Arithmetic Modules

MDS module. To achieve full diffusion, the linear layer used in cryptographic algorithms usually applies MDS matrix multiplication together with constant addition. For a t-dimentional MDS matrix, its branch number is $(t + 1)$. Since adding a constant does not influence the degree, we include them in an MDS module.

t-addition module. Addition with $(t - 1)$ addends is called t-addition since there are t branches around the addition symbol and any $(t - 1)$ branches determine the other. It is a linear module and shares some similarities with the MDS module in generating constraints in $\mathcal{M}$.

The MDS module and the t-addition module are referred to as linear modules.

Non-linear univariate polynomial (NLUP) module. This is a more general case of power map-type Sboxes. The degree of the output of NLUP is multiple times that of the input depending on the degree of the polynomial.

Non-linear multivariate polynomial (NLMP) module. This is also a non-linear module. The simplest NLMP is the multiplication of two branches.

We redefine each arithmetic module's inputs and outputs from the perspective of low-degree computation. We call the arithmetic equivalent of inputs and outputs as upstreams and downstreams.

Definition 2 (Upstream/downstream/pending) For an arithmetic module with degree α being a small integer, the upstreams are the input branches and the downstreams are the output branches. In other words, computing from upstream to downstream is always a not larger degree of computation than the inverse computation, if the inverse computation is possible. If a branch can have multiple choices regarding the arithmetic module, it is called a pending branch.

We use the symbol $\rightarrow$ to represent the upstream input, $\succ$ the downstream output, and $\multimap$ the pending branch, connected to the four arithmetic modules, as is shown in Figure 4. Labeling NLUP and NLMP modules is deterministic since the low-degree calculation is only possible in one direction. For a t-addition module, any $(t - 1)$ branches can be upstream to determine the other branch. For a t-dimensional MDS affine module, any t branches will determine the other t branches. So all branches of MDS and t-addition modules are pending. The final optimal choice will be given by the automatic model.

4.2 Branching Modules

An intermediate branch has multiple ends. Each end has a status decided from the arithmetic module it connects. We call the intermediate branch with labeled status on each end the branching module. After

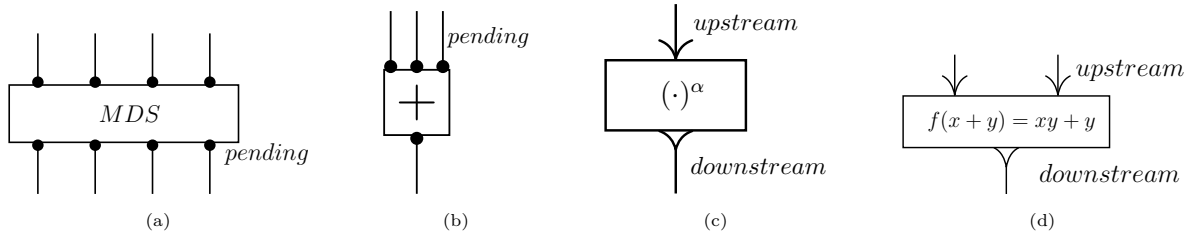


Fig. 4 Upstream, downstream, and pending branches for arithmetic modules. (a) MDS. (b) 4-addition. (c) NLUP. (d) NLMP.

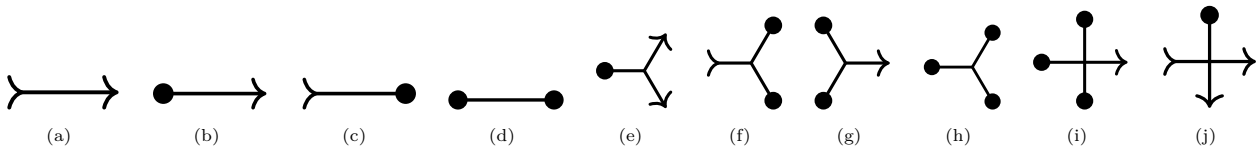


Fig. 5 Branching modules. (a) down-up. (b) pending-up. (c) down-pending. (d) pending-pending. (e) pending-ups. (f) down-pending-pending. (g) pending-pending-up. (h) pending-pending-pending. (i) pending-pending-pending-up. (j) down-pending-ups.

parsing several AO cryptographic algorithms, we find that there are only a few types of branching modules, as shown in Figure 5. The branching module is named by the status of its ends in a "down-pending-up" order. Two-end branching modules include the down-up module, the pending-up module, the down-pending module, and the pending-pending module. Three-end branching modules include the pending-ups module, the down-pending-pending module, the pending-pending-up module, and the pending-pending-pending module. Four-end branching modules include the pending-pending-pending-up module and the down-pending-ups module. Modeling for branching modules with more ends can be deduced from that on these modules. Together with the arithmetic modules, they can cover the arithmetic paths of most AO primitives.

4.3 Types of variables Set in the Model

Degree propagation will be captured by constraints on the following types of variables.

- *var*: binary variable. Indicate whether a variable is introduced at this position.
- *con*: binary variable. Indicate whether this position is set to a constant and thus consumes a degree of freedom.
- *deg*: integer variable. The degree of the expression of this position.
- *g*: binary variable set for pending branches of a linear module. Indicate whether this branch has been determined externally.
- *bse*: binary variable set for pending branches of a linear module. Indicate whether this branch is selected into the basis of the linear module it belongs to.
- *h*: binary variable set for pending branches of a linear module. Indicate that the branch is determined externally but is not chosen into the basis. It equals to $g - bse$.
- *equ*: integer variable. The sum of degrees of equations that are built regarding a module, since only the sum of degrees of equations matters in the Gröbner basis attack complexity.

We add to the variables superscripts to indicate rounds and subscripts to indicate intermediate positions. For example, var_i^r indicates if a variable would be introduced at the i-th position in the r-th round.

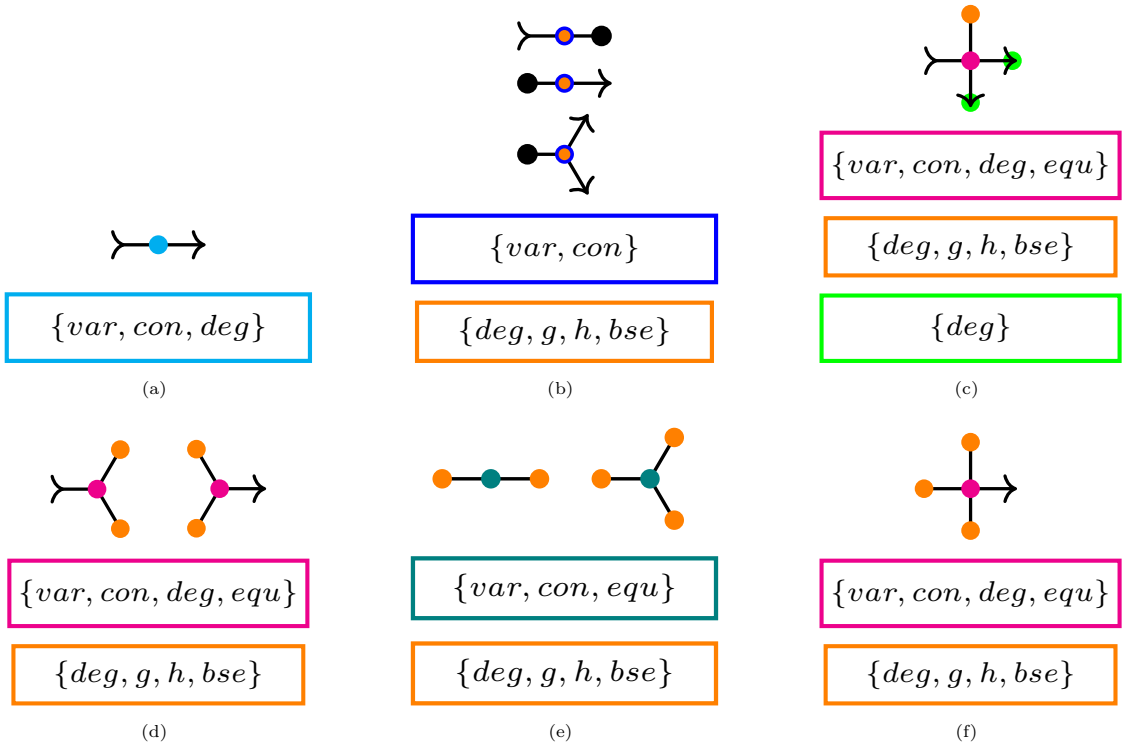
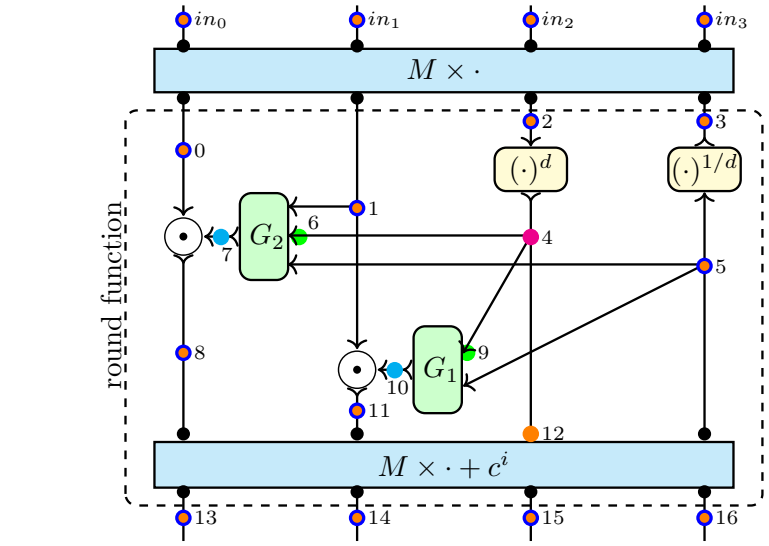


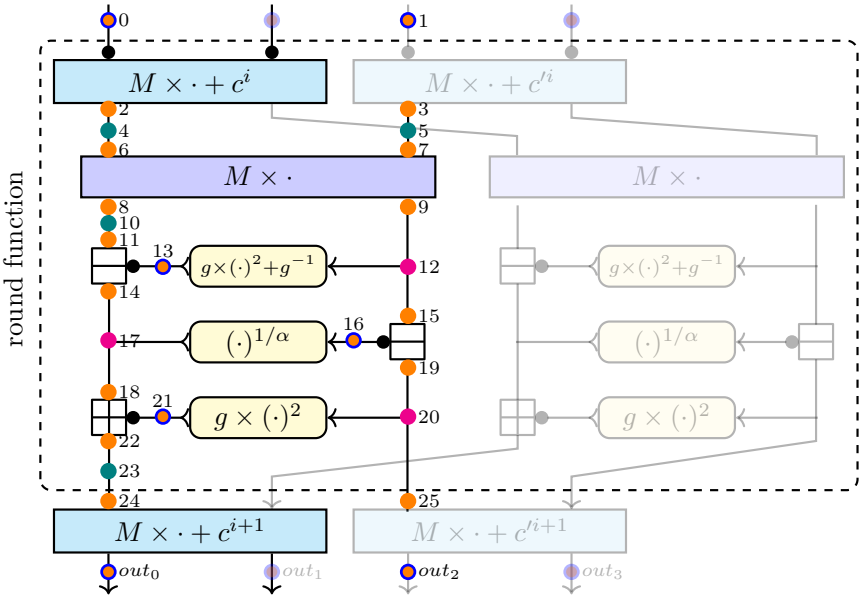
Fig. 6 Variables introduced for branching modules. (a) down-up. (b) down-pending and pending-up(s). (c) down-pending-ups. (d) down-pending-pending and pending-pending-up. (e) pending-pending and pending-pending-pending. (f) pending-pending-pending-up.

Not all types of variables need to be set for all modules in the MIQCP model. For each arithmetic module, an equation variable *equ* is set. For branching modules, what types of variables are set are given in Figure 6. We classify the types of variables introduced into groups and labeled by boxes in different colors. Each group corresponds to a point in a branching module. Note that some of the points labeled for introducing variables are the ends of the branching module, but some are extra added at the intermediate position. For the down-up module, $\{var, con, deg\}$ variables for the intermediate position are set. For the down-pending and the pending-up(s) module, basic variable $\{var, con\}$ and variable quadruple for pending end $\{deg, g, h, bse\}$ are set. For the down-pending-up(s) module, $\{var, con, deg, equ\}$ for the intermediate position, $\{deg, g, h, bse\}$ for the pending end, and $\{deg\}$ for each upstream end are set. For the down-pending-pending and pending-pending-up modules, $\{var, con, deg, equ\}$ for the intermediate position, and $\{deg, g, h, bse\}$s for the pending ends are set. For the pending-pending and pending-pending-pending modules, $\{var, con, equ\}$ for the intermediate position, and $\{deg, g, h, bse\}$s for the pending ends are set. For the pending-pending-pending-up module, $\{var, con, deg, equ\}$ for the intermediate position, $\{deg, g, h, bse\}$ for each pending end are set.

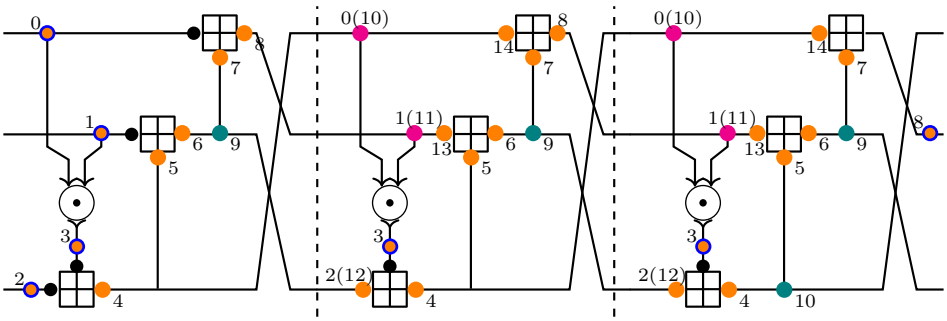
Figure 7 presents examples of parsing the Griffin, Anemoi, and Ciminion round permutations. For a permutation algorithm, MDS matrix is surrounded by pending types, and we can easily determine the upstream and downstream of the S-boxes, Multiplication and G-functions. Based on this, we look for which module they form. For example, in the Griffin permutation, the top of node 4 is the downstream of the s-box, the left are nodes 6 and 9 respectively, which are the upstream of the G_i, while node 12 is the pending of the MDS, so it constitutes a down-pending-ups model. The round permutation for Griffin involves 17 points where variable groups are set. The Anemoi round function involves 26 points for a subroutine on a pair of input branches. The points on parallel subroutines are the same. For Ciminion, the branching modules of the first round, the middle rounds, and the last round are different, involving 13, 15, and 13 points, respectively.



(a)



(b)



(c)

Fig. 7 Parsing of the Griffin, the Anemoi and Ciminion round functions. (a) Griffin. (b) Anemoi. (c) Ciminion.

5 Constraints on Arithmetic Modules

In this section, we explain how to set constraints for arithmetic modules in the MIQCP model. The process of building equations is the process of adding variables to the system equSys and consuming the degrees of freedom they introduce by imposing the arithmetic rules.

5.1 MDS Module

For an MDS module, the input and output are denoted by $\mathbf{x} = (x_0, x_1, \cdots, x_{t-1}) \in \mathbb{F}_p^t$ and $\mathbf{x}' = (x_t, x_{t+1}, \cdots, x_{2t-1}) \in \mathbb{F}_p^t$, and in the arithmetic path we have $\mathbf{x}' = A\mathbf{x} + \mathbf{b}$, where A is an MDS matrix, b is a vector of round constant. The expressions of the input branches and output branches, and the possible equations built within the MDS module form an arithmetic equivalence of the transformation.

Mechanism of building equations within an MDS module. The branches related to the MDS module may be predetermined by external modules, for example, when it is the downstream branch of an NLUP module. Concentrating on the MDS module, there are t degrees of freedom. Any predetermined t branches will deduce the other t branches. Once more than t branches are predetermined, say s, to ensure arithmetic equivalence, $(s - t)$ equations should be added to equSys. Meanwhile, the other $(2t - s)$ branches can be expressed in multiple ways. To keep a low degree for both the $(s - t)$ equations and the $(2t - s)$ expressions to be determined, we need to choose t predetermined branches with the lowest degrees to express other branches, formally defined as the basis of the MDS module.

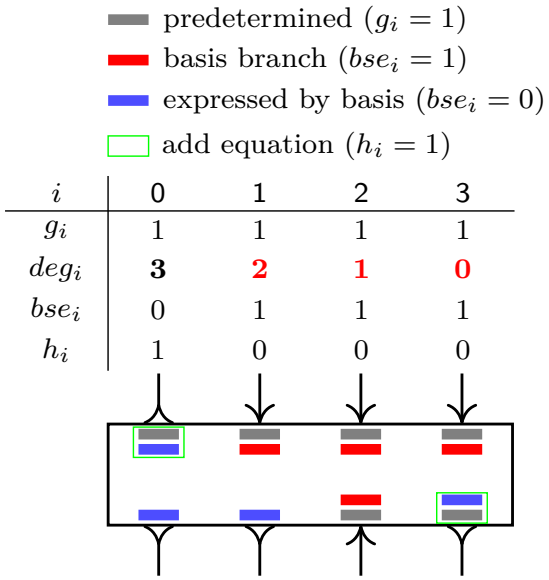
Definition 3 (Basis of MDS module) A basis of a t-dimentional MDS module is a set of t branches with the lowest degrees out of the s predetermined branches of the module.

For example, for the MDS module with $t = 4$ in Figure 8, six branches are predetermined with degrees 3, 2, 1, 0, 1, and 5 respectively. We select the four branches with the lowest degrees 0, 1, 1, and 2 as a basis. Then the other four branches can be expressed by the basis. An equation of degree 3 regarding the 0-th branch and an equation of degree 5 regarding the 7-th branch are added. The 4-th and 5-th branches now get their expression by the basis of the MDS module and will propagate on. Therefore the arithmetic equivalence is guaranteed in the equSys.

The above mechanism should be expressed by variables and constraints. Wlog, suppose the branches with indexes i $(0 \le i \le 2t - 1)$ are connected to the MDS module in the r-th round. Recall Section 4.3, the variables relating to the MDS module include $var_i^r, con_i^r, g_i^r, h_i^r, bse_i^r, deg_i^r, 0 \le i \le 2t-1$, and equ_{MDS}^r associated to this MDS module.

Constraints. The above variables should abide by the following constraints to reflect the mechanism of building equations within the MDS module.

- If a variable is introduced, i.e. $var_i^r = 1$, the branch is determined:
$$var_i^r = 1 \Rightarrow g_i^r = 1, 0 \le i \le 2t - 1.$$
- If a constant is introduced in the branch, i.e. $con_i^r = 1$, the branch is determined:
$$con_i^r = 1 \Rightarrow g_i^r = 1, 0 \le i \le 2t - 1.$$
- The number of introduced constants around an MDS module is at most t:
$$\sum_{i=0}^{2t-1} con_i^r \le t.$$
- Basis can only select from determined branches (redundant due to the definition and the binary feature of h_i^r):
$$g_i^r \ge bse_i^r, 0 \le i \le 2t - 1,$$
- Definition of h_i^r:
$$h_i^r = g_i^r - bse_i^r, 0 \le i \le 2t - 1.$$



i	0	1	2	3
g_i	1	1	1	1
deg_i	**3**	**2**	**1**	**0**
bse_i	0	1	1	1
h_i	1	0	0	0

i	4	5	6	7
g_i	0	0	1	1
deg_i	**2**	**2**	**1**	**5**
bse_i	0	0	1	0
h_i	0	0	0	1

Fig. 8 Basis of MDS module

- The basis contains t branches:

$$\sum_{i=0}^{2t-1} bse_i^r = t.$$

- The basis should include the t determined branches with the lowest degrees. In other words, the degrees of determined branches that are not selected into the basis should be no lower than those in the basis:

$$deg_B = \underbrace{\max\{bse_j^r \times deg_j^r | 0 \le j \le 2t-1\}}_{\text{the maximum degree in the basis}}$$
$$h_i^r = 1 \Rightarrow deg_B \le deg_i^r, 0 \le i \le 2t-1.$$

- The sum of degrees of the equations

$$equ_{MDS}^r = \sum_{i=0}^{2t-1} (h_i \times deg_i^r)$$

- If a branch is not determined, i.e. $g_i^r = 0$, it should be expressed by the basis branches, so the degree is determined by that of the base:

$$g_i^r = 0 \Rightarrow deg_i^r = deg_B, 0 \le i \le 2t-1.$$

Besides, to avoid trivial solutions, when the base degree is zero, the *con* variables of the basis branch should be set to 1. This can be ensured by

$$deg_B = 0 \Rightarrow \sum_{i=0}^{2t-1} con_i^r = t.$$

5.2 t-Addition Module

The t addition module is similar to the MDS module. Any $(t-1)$ branches will determine the other branches, so the number of branches in a basis is $(t-1)$. We give the details of constraints on the t-addition module in the Appendix 9.1.

5.3 Non-linear Univariate Polynomial (NLUP) Module

A univariate polynomial operates on a branch. The downstream degree is α times the upstream degree, where α is the degree of the univariate polynomial. Suppose the upstream position is with index j and the downstream position is with index j'. Variables related to a non-linear univariate polynomial module are $var^r_i, con^r_i, deg^r_i, i \in \{j, j'\}$, and equ^r_{NLUP}.

Constraints:

- If a constant is at one side of the NLUP module, the other side is a constant:
$$con^r_j = 1 \Leftrightarrow con^r_{j'} = 1.$$
- Regarding the downstream position,
  - if a new variable is introduced at the downstream position, an equation will be introduced to equSys with degree α times the upstream degree:
  $$var^r_{j'} = 1 \Rightarrow equ^r_{NLUP} = \alpha \times deg^r_j.$$
  - if no variable is introduced, the degree of downstream position is α times the upstream degree:
  $$var^r_{j'} = 0 \Rightarrow \begin{cases} deg^r_{j'} = \alpha \times deg^r_j, \\ equ^r_{NLUP} = 0. \end{cases}$$
- Regarding the upstream position, the cases will be covered by constraints for branching modules.

If there are multiple NLUP modules in one round, apply the same rules to all.

5.4 Non-linear Multivariate Polynomial (NLMP) Module

Multivariate polynomials used in AO primitives are simple, so the downstream degree can be determined easily by the upstream degrees. We instantiate the model for the G_i function with degree 2 and the multiplication ($\times$) used in Griffin, and other polynomials can be modeled accordingly. Suppose the downstream position is with index j, and the upstream positions are with index $i_0, i_1, \cdots$. Recall Section 4.3, the variables relating to an NLMP modules include $var^r_j, con^r_j, deg^r_j, var^r_{i_k}, con^r_{i_k}, deg^r_{i_k}, k = 0, 1, \cdots$, and equ^r_{NLMP}.

Constraints:

- Regarding the downstream position,
  - If a variable or a constant is introduced in the downstream position, an equation is introduced:
  $$\begin{aligned} \text{for } G\text{: } & var^r_j + con^r_j \geq 1 \Rightarrow equ^r_{NLMP} \\ & = 2 \times \max\{deg^r_{i_0}, deg^r_{i_1}, \cdots\}, \\ \text{for } \times\text{: } & var^r_j + con^r_j \geq 1 \Rightarrow equ^r_{NLMP} \\ & = deg^r_{i_0} + deg^r_{i_1}. \end{aligned}$$
  - If no variable nor constant is introduced in the downstream position, the output is expressed by

the upstream positions:

$$\text{for } G\text{: } var_j^r = 0 \text{ and } con_j^r = 0$$
$$\Rightarrow \begin{cases} deg_j^r = 2 \times \max\{deg_{i_0}^r, deg_{i_1}^r, \cdots\}, \\ equ_{NLMP}^r = 0. \end{cases}$$
$$\text{for } \times\text{: } var_j^r = 0 \text{ and } con_j^r = 0$$
$$\Rightarrow \begin{cases} deg_j^r = deg_{i_0}^r + deg_{i_1}^r, \\ equ_{NLMP}^r = 0. \end{cases}$$

- Regarding the upstream position, the cases are covered by constraints on branching modules.

If multiple arithmetic modules exist in one round, apply the same rules to all.

6 Constraints on Branching Modules

Intermediate branching modules connect arithmetic modules. Equations may be constructed within the branching modules. The arithmetic flow from one end to another end should also be captured by the constraints. Besides the intermediate branching modules, constraints on the border branches should also be considered.

If a variable is introduced at an intermediate position, i.e. $var = 1$, the degrees of this branch's ends are 1. Similarly, if a constant is set in the position, i.e. $con = 1$, the degrees of this branch's ends are 0. As detailed in this section, the following common constraints in Eq. (6) will be added to all branches.

$$C_{common}(var, con, deg) = \begin{cases} var = 1 \Rightarrow deg = 1, \\ con = 1 \Rightarrow deg = 0. \end{cases} \tag{6}$$

6.1 Border branch constraints

Constant indicator variables on constrained input branches and constrained output branches should be set to 1:

$$con_i^0 = 1, con_{i'}^R = 1,$$

for i, i' that belong to the capacity part and output part respectively.

For other border branches, if it is a branch connecting to a linear module, and no variable or constant would be introduced at this position in equSys (i.e. $var = 0, con = 0$), we can ignore it, and the branch would not participate in the arithmetic flow. This is ensured by constraint

$$var_{i'}^0 + con_{i'}^0 = 0 \Rightarrow g_{i'} = 0,$$
$$var_{j'}^R + con_{j'}^R = 0 \Rightarrow g_{j'} = 0,$$

for i', j' being the indexes of unconstrained border branches that are connected to an MDS module.

For all border branches, common constraints should be added.

$$C_{common}(var_i, con_i, deg_i),$$

for i being the border branch indexes.

6.2 Intermediate branch constraints

In the following, we will specify the constraints for all types of branching modules. For each branching module, the related variables are presented with suffixes u, d, p representing the upstream, downstream, and pending ends when necessary. Since the cases of introducing a new variable or a constant are already covered in the common constraints, the specific constraints just need to cover the case that the position is neither a new variable nor a constant, i.e. $var + con = 0$. The principle for generating these constraints is

that for an upstream end, we should associate its degree variable with another end. For pending ends, we should decide whether it is determined externally, and if so, associate its degree variable with another end.

6.2.1 down-up

Related variables are var, con, deg. Only common constraints in Eq. (6) are needed.

6.2.2 down-pending

Relative variables are var, con, g, deg. Since $var + con = 0$, this position has an expression from the arithmetic module connected with the downstream end. From the perspective of the linear arithmetic module, the pending branch is determined externally.

$$var + con = 0 \Rightarrow g = 1,$$
$$C_{common}(var, con, deg).$$

6.2.3 pending-up(s)

Relative variables are var, con, g, deg. Since $var + con = 0$, the position can only be expressed by the linear arithmetic module connected with the unique pending end. So, from the perspective of the linear arithmetic module, the pending branch is not externally determined.

$$var + con = 0 \Rightarrow g = 0,$$
$$C_{common}(var, con, deg).$$

6.2.4 down-pending-ups

Variables related to this module are $var, con, deg, equ, deg_p, g_p, deg_i, i = 1, 2, \cdots$ where i corresponds to indexes of upstream ends, p stands for pending. When $var+con = 0$, this position has an expression from the downstream branch. There may be another expression from the basis of the linear modular connected with the pending end. We want the upstream branch to take the one with a lower degree. So if the downstream end was not to determine the pending end ($g = 0$), an equation would be added to connect the downstream end and the pending end, so

$$g = 0 \Rightarrow \begin{cases} deg_i = \min\{deg, deg_p\}, i = 1, 2, \cdots, \\ equ = \max\{deg, deg_p\}. \end{cases}$$

Otherwise, if the downstream determines the pending end ($g = 1$ and thus $deg_p = deg$), this pending end can be chosen as a basis. The connection between the downstream and the pending ends has already been ensured inside the linear module. The upstream takes the expression from the downstream end with degree deg. It is possible that the degree of the downstream is larger than that of the basis of the linear module (which happens when the pending end is not chosen as a basis), so it would be better for the upstream to take the expression from the pending end. In this case, the model will automatically go to the above case where $g = 0$. So we set

$$g = 1 \Rightarrow \begin{cases} deg_p = deg, \\ deg_i = deg, i = 1, 2, \cdots, \\ equ = 0. \end{cases}$$

Actually, $equ = 0$ constraint is not necessary as it will be forced automatically. Besides, common constraints should be added.

$$C_{common}(var, con, deg),$$
$$C_{common}(var, con, deg_p),$$
$$C_{common}(var, con, deg_i), i = 1, 2, \cdots.$$

6.2.5 down-pending-pending

Relative variables are g_{p1},deg_{p1},g_{p2},deg_{p2}, var, con, deg, equ. When $var + con = 0$, there are three cases where the values of g_{p1} and g_{p2} are taken differently. The first case is that both pending ends are determined

by the downstream end. So the degrees of the pending ends are equal to that of the downstream end.

$$g_1 + g_2 = 2 \Rightarrow \begin{cases} deg_{p1} = deg, \\ deg_{p2} = deg, \\ equ = 0. \end{cases}$$

Though the lefthand condition is also valid when $var+con = 1$, the common constraints under this condition are compatible with the righthand constraints.

The second case is that the downstream end determines none of the pending ends, which will be expressed by the linear modules. In this case, two equations should be added for this branching module to connect the arithmetic flow. The sum of the equation degrees is the sum of the largest two degrees of the three.

$$\begin{aligned} &g_{p1} + g_{p2} = 0 \Rightarrow \\ &equ = deg_{p1} + deg_{p2} + deg - \min\{deg_{p1}, deg_{p2}, deg\}. \end{aligned}$$

Note that the lefthand side only happens when $var + con = 0$.

The third case is that only one of the pending ends is determined, i.e. $g_{p1} + g_{p2} = 1$. The determination can be from the downstream end or the other pending end. In any case, an equation should be added to connect the arithmetic flow with a degree the largest one of the three branches.

$$\begin{aligned} &g_{p1} + g_{p2} = 1 \Rightarrow \\ &\begin{cases} deg + deg_{p1} + deg_{p2} - \max\{deg, deg_{p1}, deg_{p2}\} \\ = 2 \times \min\{deg, deg_{p1}, deg_{p2}\}, \\ equ = \max\{deg, deg_{p1}, deg_{p2}\}. \end{cases} \end{aligned}$$

Note that the lefthand side only happens when $var + con = 0$.

Besides, common constraints should be added.

$$\begin{aligned} &C_{common}(var, con, deg_p), \\ &C_{common}(var, con, deg_{p1}), \\ &C_{common}(var, con, deg_{p2}). \end{aligned}$$

6.2.6 pending-pending

Variables related to this module are $deg_{p1}, deg_{p1}, g_{p1}, g_{p2}, var, con, equ$. When $var + con = 0$, constraint $g_{p1} = 1$ means the 1st pending end is determined externally by the linear module connected with the 2nd pending end, and vice versa. So, at most one is determined.

$$var + con = 0 \Rightarrow g_{p1} + g_{p2} \leq 1.$$

When one is determined and the other is not, the degrees on the two ends are equal, and no equation is added.

$$g_{p1} + g_{p2} = 1 \Rightarrow \begin{cases} deg_{p1} = deg_{p2}, \\ equ = 0. \end{cases}$$

When both branches are expressed by the linear module, an equation should be added to connect them.

$$g_{p1} + g_{p2} = 0 \Rightarrow equ = \max\{deg_{p1}, deg_{p2}\}.$$

Note that deg_{p1} and deg_{p2} will be set in the linear module.

Besides, common constraints should be added.

$$\begin{aligned} &C_{common}(var, con, deg_{p1}), \\ &C_{common}(var, con, deg_{p2}). \end{aligned}$$

6.2.7 pending-pending-up

When the pending-pending module meets another upstream branch, besides the constraints built for the pending-pending module, add additional constraints regarding the degree variable deg_u of the upstream

end:

$$\begin{aligned}&\text{pending-pending module},\\ &deg_u = \min\{deg_{p1}, deg_{p2}\},\\ &C_{common}(var, con, deg_u).\end{aligned}$$

6.2.8 pending-pending-pending

Variables related to this module are $deg_{p1}, deg_{p2}, deg_{p3}, g_{p1}, g_{p2}, g_{p3}, var, con, equ$. When $var + con = 0$, at most two branches are determined externally. So,

$$var + con = 0 \Rightarrow g_{p1} + g_{p2} + g_{p3} \leq 2.$$

If one branch determines the other, the computation flow is naturally established, and the degrees of the three ends are equal.

$$g_{p1} + g_{p2} + g_{p3} = 2 \Rightarrow \begin{cases} deg_{p1} = deg_{p2} = deg_{p3}, \\ equ = 0. \end{cases}$$

If only one branch is determined, the degree of the determined one is the smaller of the other degrees. To ensure computation flow, one equation needs to be added with the degree being the maximum of the three.

$$\begin{aligned}&g_{p1} + g_{p2} + g_{p3} = 1 \Rightarrow\\ &\begin{cases} deg_{p1} + deg_{p1} + deg_{p2} - \max\{deg_{p1}, deg_{p2}, deg_{p3}\} \\ = 2 \times \min\{deg_{p1}, deg_{p2}, deg_{p3}\}, \\ equ = \max\{deg_{p1}, deg_{p2}, deg_{p3}\}. \end{cases}\end{aligned}$$

If no branch is determined, two equations need to be added to ensure the computation flow. The sum of the degrees of the equations is the sum of the two largest degrees.

$$\begin{aligned}&g_{p1} + g_{p2} + g_{p3} = 0 \Rightarrow\\ &equ = deg_{p1} + deg_{p2} + deg_{p3} - \min\{deg_{p1}, deg_{p2}, deg_{p3}\}.\end{aligned}$$

Besides, common constraints should be added.

$$\begin{aligned}&C_{common}(var, con, deg_{p1}),\\ &C_{common}(var, con, deg_{p2}),\\ &C_{common}(var, con, deg_{p3}).\end{aligned}$$

6.2.9 pending-pending-pending-up

When the pending-pending-pending module meets another upstream branch, besides the constraints built for the pending-pending-pending module, add additional constraints regarding the degree variable deg_u of the upstream end:

$$\begin{aligned}&\text{pending-pending-pending module},\\ &deg_u = \min\{deg_{p1}, deg_{p2}, deg_{p3}\},\\ &C_{common}(var, con, deg_u).\end{aligned}$$

This completes the full model of the CICO problem on a given AO primitive.

7 Applications

We build MIQCP model for Griffin, Anemoi, and Ciminion to show the wide applicability of our model. The first two primitives allow for an under-determined CICO problem, while Ciminion allows only for the well-determined. The lower bound of the Gröbner basis attack complexity given by the designers is presented in the last line in the results tables, which may be exerted on the degree of regularity (shown in the second column) or the overall complexity (shown in the third column). The MIQCP models are solved by Gurobi[40], and the solving time limit is set to 12-hour. The source codes are provided in repository https://github.com/qiaokexin/Groebner_MIQCP.git.

7.1 Application to Griffin

Under-determined CICO. For Griffin [9], to achieve an s-bit security level, the number of branches in the capacity part is required to be $c \geq \lceil 2s/\log_2(q) \rceil$, and the output should be at least $\lceil 2s/\log_2(q) \rceil$ branches. For $q \approx 2^{256}, s = 128$ [9, Table 2], it is allowed that $c = r = 1$, forming an under-determined CICO problem for all choices of state width $t = 3, 4, 8, 12, 16, 20, 24$.

Estimation by designers. The designers of Griffin used two strategies to estimate the lower bound of the complexity of Gröbner basis attack and focus on the attacker's easiest case where the number of outer branches in the Sponge construction is 1. The first strategy introduces variables on all output branches in each round, so there will be two equations with degree α (the degree of Sbox) and $(t-2)$ equations with degree 3 (due to G_i's and multiplication) for each round function. In this case, the number of variables is $(tR+1)$, and the degree of regularity used to calculate the overall complexity is reduced to a lower bound $D^{(1)}_{\text{est}} = \alpha R$, where R is the number of rounds. The second strategy is to introduce one variable for each round, so the number of variables is $(1+R)$, and the degree of regularity is reduced to a lower bound $D^{(2)}_{\text{est}} = \alpha^R$. The second strategy's complexity is higher overall than the first one, so we present the lower bound of the first strategy in the comparison table. So the complexity lower bound used by the designers is $\binom{\alpha R + 1 + tR}{1 + tR}^{\omega}$.

Our results. We apply our proposed automatic Gröbner basis attack model to Griffin-π permutation. We focus on the under-determined CICO problem where one input branch and one output branch are constrained. The solver can automatically set $(t-2)$ additional constants in the equation system. Such a setting reflects the advantage of our automatic method to handle the globally under-determined CICO problem with the potential to reduce overall complexity. The degree of regularity is calculated by Equation (2). The complexity estimated by our results and the lower bound of complexity estimated by the designers is shown in Table 1. From the designers' point of view, it is reasonable to underestimate an attack's complexity with a lower bound of the degree of regularity. Even so, for up to 8 rounds, the complexity estimated by our method without reducing D_{reg} still falls below the underestimated complexity the designers gave. To test for correctness, we implemented equation systems for 2-round versions and obtained effective preimages.

The optimal results of 8 branches can be extended to 12, 16, 20, and 24 branches. Since the number of constants is equal to the number of branches, in order to extend the 8-branch case to 12, 16, 20, or 24 branches without increasing the number of variables or the equation degrees, it is only necessary to rationalize the additional constants. For example, for the optimal 4-round equation system with 8 variables suggested by the model shown in Figure 9, to extend it to a 12-branch case with the same number of variables and equation degrees, 4 constant branches can be added at the left of a constant output branch of the first MDS layer. This way, the downward propagation does not affect the degree of MDS basis and, thus, the equation degrees. The sum of equation degrees with 8 variables of 12-branch width case remains to be 72. Such extension exists for all the other branches.

7.2 Application to Anemoi

Under-determined CICO. Anemoi[10] follows a flat sponge claim. Provided that $h \geq c, h\log_2 q \geq 2s, c\log_2 q \geq 2s$, a flat sponge claim states that a sponge-based hash function provides $c\lfloor \log_2 q \rfloor/2$ bits of security. A specific instance recommends $\lceil \log_2 q \rceil = 255$ for 127-bit security. So, it is allowed that $h = c = 1$, forming an under-determined CICO problem with state width larger than 2.

Estimation by designers. The equation system is built by introducing variables at every round, so the number of variables is $2lR$, where l is half the number of state branches, and R is the number of rounds. The lower bound of the degree of regularity conjectured by the designers is $2lR + \kappa_\alpha$, where κ_α is a constant related to Sbox degree α, and takes $\kappa_3 = 1, \kappa_5 = 2, \kappa_7 = 4, \kappa_9 = 7$, and $\kappa_\alpha = 9$ for $\alpha \geq 11$. So the complexity lower bound used by the designers is $\binom{2lR + \kappa_\alpha + 2lR}{2lR}^{\omega}$.

Table 1 Complexity of Gröbner basis attack on Griffin-π with Sbox degree $\alpha = 5$

r	2			3			4			5		
	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.
	2	**42**	**19.64**	**3**	**97**	**34.43**	4	272	55.51	5	347	70.53
$t = 3$	3	23	21.97	4	58	37.59	**5**	**93**	**51.42**	**6**	**148**	**67.35**
(R=12)	4	20	25.09	5	43	40.10	6	78	56.1	7	113	70.52
Griffin [9]	7	27	28.49	10	40	43.28	13	53	58.19	16	66	73.17

r	6			7			8			9		
	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.
	7	227	84.79	8	348	104.32	10	278	118.43	11	357	135.70
$t = 3$	**8**	**148**	**84.36**	**9**	**203**	**100.65**	**11**	**203**	**117.50**	**12**	**258**	**133.99**
(R=12)	9	133	89.46	10	168	103.65	12	188	122.79	13	223	136.90
Griffin [9]	19	79	88.20	22	92	103.26	25	105	118.35	28	118	133.46

r	2			3			4			5			6		
	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.
	2	**38**	**19.07**	4	76	40.74	6	122	63.97	8	156	85.59	10	186	106.65
$t = 4$	3	19	20.31	**5**	**41**	**39.40**	**7**	**71**	**60.91**	**9**	**101**	**82.12**	**11**	**131**	**103.24**
(R=11)	4	16	22.43	6	34	41.26	8	60	62.91	10	98	87.65	12	128	109.08
Griffin [9]	9	33	32.99	13	49	50.32	17	65	67.78	21	81	85.32	25	97	102.91
	2	24	16.46	**5**	**35**	**37.05**	7	107	69.40	9	323	112.86	11	971	167.68
$t = 8$	**3**	**11**	**15.56**	6	28	37.72	**8**	**72**	**67.29**	**10**	**180**	**105.68**	**12**	**504**	**157.47**
(R=9)	4	10	16.73	7	28	41.15	9	64	69.79	11	173	112.31	13	726	181.77
Griffin [9]	17	57	46.02	25	85	70.45	33	113	95.05	41	141	119.73	49	169	144.46

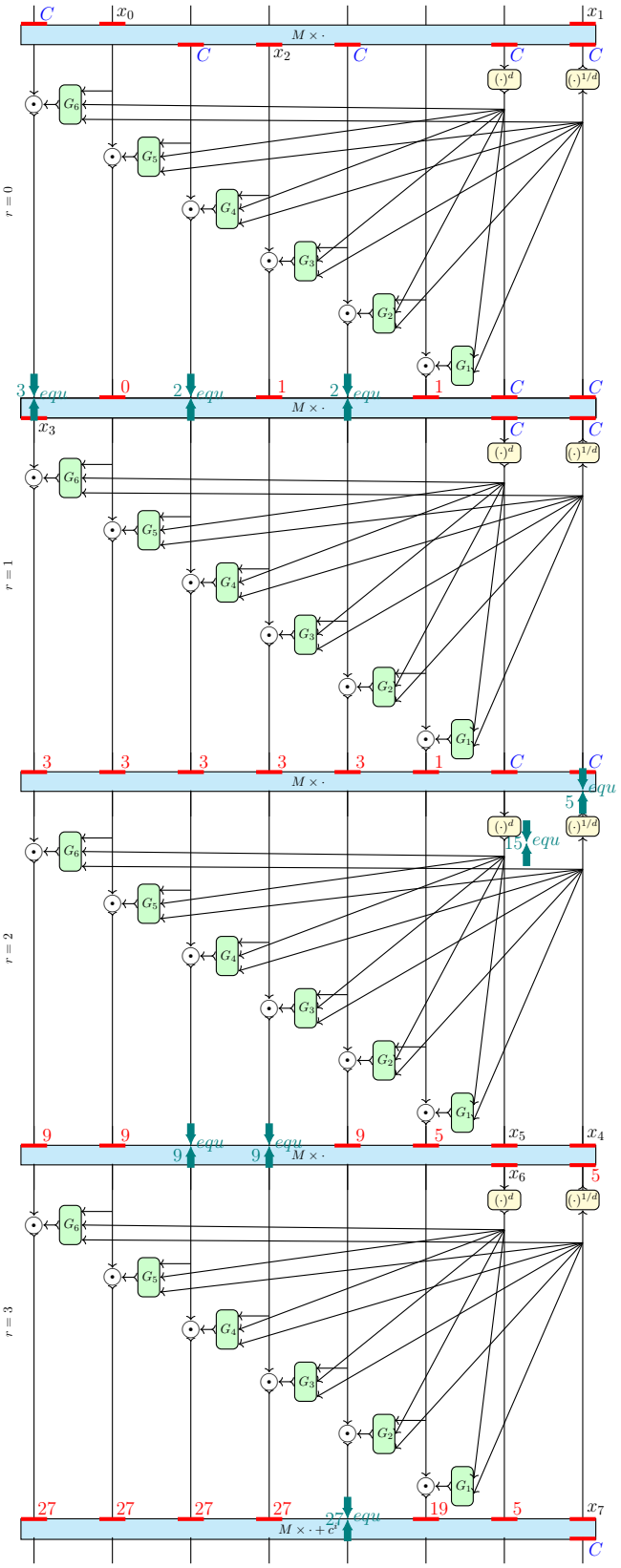


Fig. 9 4-round 8-branch Griffin equations construction. The basis, degrees, and where to introduce variables, constants and equations are labeled.

Our results. The complexity of Gröbner basis attack on Anemoi with Sbox degree $\alpha = 3$ is shown in Table 2. The results are obtained by setting the time limit for solving the MIQCP model to 12-hour and the dash - in the table indicates that there is no feasible solution for the corresponding variable within the 12-hour limit. For rounds up to 4, the complexity given by our model is lower than the lower bound estimated by the designers. We implemented the 2-round equation system suggested by the model shown in Figure 10. Four variables and two constants are set at intermediate positions. The degree of intermediate positions and the basis branches of MDS modules are labeled. We obtain an effective preimage by constructing four equations indicated by the meeting arrows in the figure with degrees 2, 4, 4, and 4.

Table 2 Complexity of Gröbner basis attack on Anemoi with Sbox degree $\alpha = 3$

r	1			2			3			4			5		
	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.
$t = 4$	**2**	**4**	**6.64**	**4**	**14**	**20.83**	6	31	39.58	11	36	59.34	16	-	-
	3	5	8.64	5	15	24.19	7	25	38.65	**12**	**33**	**58.06**	17	-	-
	4	6	10.26	6	13	23.10	**9**	**21**	**37.85**	14	32	59.22	**18**	**42**	**78.29**
	5	7	11.61	7	14	25.30	10	22	40.25	15	33	61.58	19	-	-
Anemoi[10]	4	5	13.95	8	9	29.14	12	13	44.62	16	17	60.24	20	21	75.94
$t = 6$	**2**	**4**	**6.64**	6	18	29.46	14	-	-	20	-	-			
	3	**4**	**6.64**	**8**	**16**	**29.14**	15	-	-	21	-	-			
	4	5	7.81	9	17	31.14	**16**	**32**	**60.24**	22	-	-			
	5	6	8.78	10	18	33.00	17	-	-	23	-	-			
Anemoi[10]	6	7	21.49	12	13	44.62	18	19	68.08	24	25	91.69			

7.3 Application to Ciminion

In Ciminion [4], permutations with the same round permutation p_c and p_E are applied in a nonce-based stream-encryption mode. The CICO problem is well-determined, with the constrained input a nonce branch and the constrained outputs two key stream branches. The designers estimated the complexity of attacking a weaker scheme by introducing two variables in the input. The lower bound of the degree of regularity is 2^{R+1}, and the attack complexity is lower bounded by $\binom{2+2^{R+1}}{2}^{\omega} \geq (\frac{(1+2^{R+1})^2}{2})^{\omega} \geq 2^{2R+1}$.

Our results. The results on well-determined CICO are not very interesting for rounds above 3, so we relax the problem to an under-determined CICO problem where only the first input branch and the first output branch are fixed. The complexity of Gröbner basis attack on Ciminion is shown in Table 3. The complexity for solving univariate polynomials is estimated by $O(d \cdot \log(d) \cdot (\log(q) + \log(d)) \cdot \log(\log(d)))$ [12, §3.1], where d is the degree of the equation. To test for correctness, we implemented the 7-round equation system suggested by the model shown in Figure 11. Instead of introducing variables at the input branches, introducing the two variables in the intermediate positions will reduce the solving complexity. An equation of degree 12 and an equation of degree 4 are constructed. We obtain an effective preimage by solving the equation system.

8 Conclusions

We present an automatic tool to obtain the equation system with low solving complexity in Gröbner basis attacks on arithmetization-oriented hash functions. Due to the large scale of the MIQCP model built to describe the AO primitive (which is a common drawback of automatic cryptanalysis tools), we limited all experiments to run for at most 12-hour and proved that the overestimated complexity of Gröbner basis attack occurs for about half the full rounds for Griffin. Though the results in Anemoi and Ciminion are not very interesting in that we only proved the complexity overestimation for a small number of rounds with limited time, the validity of these AO primitives models shows the wide applicability of the proposed

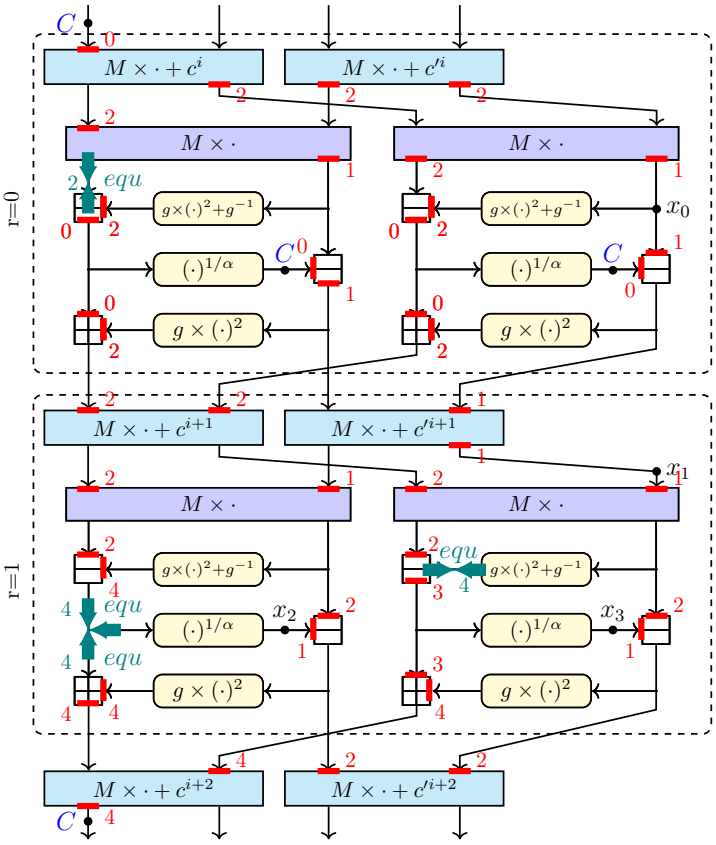


Fig. 10 2-round Anemoi equations construction. The basis, degrees, and where to introduce variables, constants and equations are labeled.

Table 3 Complexity of Gröbner basis attack on Ciminion

r	3			4			5			6		
	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.
$t = 3$	**2**	**3**	**5.17**	1	4	9.04	1	8	11.32	1	16	13.09
	3	4	6.64	**2**	**4**	**6.64**	**2**	**6**	**8.78**	**2**	**10**	**11.56**
	4	5	7.81	3	5	8.64	3	6	10.26	3	8	12.78
	5	6	8.78	4	6	10.26	4	7	12.26	4	8	13.95
Ciminion [4]	2	8	7	2	16	9	2	32	11	2	64	13
r	7			8			9					
	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.	n_v	$\sum equ$	cmp.			
$t = 3$	1	32	14.64	**1**	**64**	**16.08**	1	-	-			
	2	**16**	**14.17**	2	28	17.33	**2**	**48**	**20.40**			
	3	11	15.56	3	15	18.26	3	24	22.33			
	4	10	16.73	4	13	19.93	4	16	22.43			
Ciminion [4]	2	128	15	2	256	17	2	512	19			

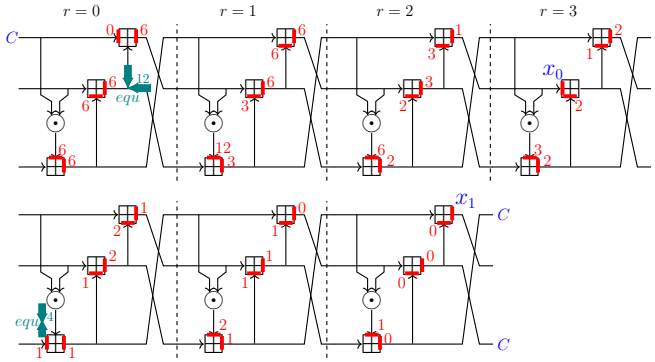


Fig. 11 7-round Ciminion equations construction. The basis, degrees, and where to introduce variables, constants and equations are labeled.

model. Therefore, with the tool presented in this paper, a more accurate estimation of the security against Gröbner basis attack can be obtained, which contributes to the AO primitives design for advanced protocols like zero-knowledge proof, homomorphic encryption, and multi-party computation.

The current model focuses on the straightforward Gröbner basis attack, involving only the number of variables and the sum of degrees. Advanced techniques in Gröbner basis attack will be taken into account in the future.

References

[1] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity," in Advances in Cryptology – ASIACRYPT 2016, J. H. Cheon and T. Takagi, Eds. Springer Berlin Heidelberg, 2016, pp. 191–219.

[2] M. R. Albrecht, L. Grassi, L. Perrin, S. Ramacher, C. Rechberger, D. Rotaru, A. Roy, and M. Schofnegger, "Feistel Structures for MPC, and More," in Computer Security – ESORICS 2019, K. Sako, S. Schneider, and P. Y. A. Ryan, Eds. Springer International Publishing, 2019, pp. 151–171.

[3] L. Grassi, R. Lüftenegger, C. Rechberger, D. Rotaru, and M. Schofnegger, "On a Generalization of Substitution-Permutation Networks: The HADES Design Strategy," in Advances in Cryptology – EUROCRYPT 2020, A. Canteaut and Y. Ishai, Eds. Springer International Publishing, 2020, pp. 674–704.

[4] C. Dobraunig, L. Grassi, A. Guinet, and D. Kuijsters, "Ciminion: Symmetric Encryption Based on Toffoli-Gates over Large Finite Fields," in Advances in Cryptology – EUROCRYPT 2021, A. Canteaut and F.-X. Standaert, Eds. Springer International Publishing, 2021, pp. 3–34.

[5] A. Aly, T. Ashur, E. Ben-Sasson, S. Dhooghe, and A. Szepieniec, "Design of Symmetric-key Primitives for Advanced Cryptographic Protocols," IACR Transactions on Symmetric Cryptology, pp. 1–45, 2020.

[6] A. Szepieniec, "On the Use of the Legendre Symbol in Symmetric Cipher Design," IACR Cryptol. ePrint Arch., vol. 2021, 2021. [Online]. Available: https://eprint.iacr.org/2021/984

[7] W. QIAO, S. SUN, and L. HU, "New Algebraic Attacks on Grendel with the Strategy of Bypassing SPN Steps," Chinese Journal of Electronics, vol. 33, pp. 1–9, 2023.

[8] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A New Hash Function for Zero-Knowledge Proof Systems," in 30th USENIX Security Symposium (USENIX Security 21). USENIX Association, Aug. 2021, pp. 519–535.

[9] L. Grassi, Y. Hao, C. Rechberger, M. Schofnegger, R. Walch, and Q. Wang, "Horst Meets Fluid-SPN: Griffin for Zero-Knowledge Applications," in Advances in Cryptology – CRYPTO 2023, H. Handschuh and A. Lysyanskaya, Eds. Springer Nature Switzerland, 2023, pp. 573–606.

[10] C. Bouvier, P. Briaud, P. Chaidos, L. Perrin, R. Salen, V. Velichkov, and D. Willems, "New Design Techniques for Efficient Arithmetization-Oriented Hash Functions: Anemoi Permutations and Jive Compression Mode," in Advances in Cryptology – CRYPTO 2023, H. Handschuh and A. Lysyanskaya, Eds. Cham: Springer Nature Switzerland, 2023, pp. 507–539.

[11] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Sponge Functions," in ECRYPT hash workshop, vol. 2007, no. 9, 2007.

[12] A. Bariant, C. Bouvier, G. Leurent, and L. Perrin, "Algebraic Attacks against Some Arithmetization-Oriented Primitives," IACR Transactions on Symmetric Cryptology, vol. 2022, no. 3, pp. 73–101, Sep. 2022.

[13] A. Bariant, A. Boeuf, A. Lemoine, I. M. Ayala, M. Øygarden, L. Perrin, and H. Raddum, "The Algebraic Freelunch: Efficient Gröbner Basis Attacks Against Arithmetization-Oriented Primitives," Cryptology ePrint Archive, Paper 2024/347, 2024, accepted to CRYPTO 2024. [Online]. Available: https://eprint.iacr.org/2024/347

[14] N. Mouha, Q. Wang, D. Gu, and B. Preneel, "Differential and linear cryptanalysis using Mixed-Integer linear programming," in Conference on Information Security and Cryptology–Inscrypt 2012. Springer, 2012, pp. 57–76.

[15] S. Sun, L. Hu, P. Wang, K. Qiao, X. Ma, and L. Song, "Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-oriented Block Ciphers," in Advances in Cryptology–ASIACRYPT 2014. Springer, 2014, pp. 158–178.

[16] K. Qiao, Z. Zhang, Z. Niu, and et al., "The exchange attack and the mixture differential attack revisited: From the perspective of automatic evaluation," Chinese Journal of Electronics, vol. 33, no. 1, pp. 19–29, 2024.

[17] Y. Cui, H. Xu, and W. Qi, "Milp-based linear attacks on round-reduced gift," Chinese Journal of Electronics, vol. 31, no. 1, pp. 89–98, 2022.

[18] Y. Sasaki and Y. Todo, "New Impossible Differential Search Tool from Design and Cryptanalysis Aspects - Revealing Structural Properties of Several Ciphers," in Advances in Cryptology - EUROCRYPT 2017, ser. Lecture Notes in Computer Science, vol. 10212, 2017, pp. 185–215. [Online]. Available: https://doi.org/10.1007/978-3-319-56617-7_7

[19] C. Cid, T. Huang, T. Peyrin, Y. Sasaki, and L. Song, "A Security Analysis of Deoxys and Its Internal Tweakable Block Ciphers," IACR Transactions on Symmetric Cryptology, pp. 73–107, 2017.

[20] H. Hadipour, N. Bagheri, and L. Song, "Improved rectangle attacks on skinny and craft," IACR Transactions on Symmetric Cryptology, pp. 140–198, 2021.

[21] A. Bariant and G. Leurent, "Truncated Boomerang Attacks and Application to AES-Based Ciphers," in Advances in Cryptology – EUROCRYPT 2023, C. Hazay and M. Stam, Eds. Cham: Springer Nature Switzerland, 2023, pp. 3–35.

[22] Z. Xiang, W. Zhang, Z. Bao, and D. Lin, "Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers," in Advances in Cryptology – ASIACRYPT 2016, J. H. Cheon and T. Takagi, Eds. Springer Berlin Heidelberg, 2016, pp. 648–678.

[23] Q. Wang, Y. Hao, Y. Todo, C. Li, T. Isobe, and W. Meier, "Improved Division Property Based Cube Attacks Exploiting Algebraic Properties of Superpoly," in Advances in Cryptology – CRYPTO 2018, H. Shacham and A. Boldyreva, Eds. Cham: Springer International Publishing, 2018, pp. 275–305.

[24] J. He, K. Hu, B. Preneel, and M. Wang, "Stretching Cube Attacks: Improved Methods to Recover Massive Superpolies," in Advances in Cryptology – ASIACRYPT 2022, S. Agrawal and D. Lin, Eds. Cham: Springer Nature Switzerland, 2022, pp. 537–566.

[25] D. Shi, S. Sun, P. Derbez, Y. Todo, B. Sun, and L. Hu, "Programming the Demirci-Selçuk Meet-in-the-Middle Attack with Constraints," in Advances in Cryptology – ASIACRYPT 2018, T. Peyrin and S. Galbraith, Eds. Springer International Publishing, 2018, pp. 3–34.

[26] E. Bellini, D. Gerault, J. Grados, R. H. Makarim, and T. Peyrin, "Fully Automated Differential-Linear Attacks Against ARX Ciphers," in Topics in Cryptology – CT-RSA 2023, M. Rosulek, Ed. Cham: Springer International Publishing, 2023, pp. 252–276.

[27] Y. Han, C. Wang, Z. Niu, and et al., "Sat-based automatic searching for differential and linear trails: Applying to crax," Chinese Journal of Electronics, vol. 33, no. 1, pp. 72–79, 2024.

[28] Z. Bao, X. Dong, J. Guo, Z. Li, D. Shi, S. Sun, and X. Wang, "Automatic Search of Meet-in-the-Middle Preimage Attacks on AES-like Hashing," in Advances in Cryptology – EUROCRYPT 2021, A. Canteaut and F.-X. Standaert, Eds. Springer International Publishing, 2021, pp. 771–804.

[29] X. Dong, J. Hua, S. Sun, Z. Li, X. Wang, and L. Hu, "Meet-in-the-Middle Attacks Revisited: Key-Recovery, Collision, and Preimage Attacks," in Advances in Cryptology – CRYPTO 2021, T. Malkin and C. Peikert, Eds. Springer International Publishing, 2021, pp. 278–308.

[30] X. Dong, J. Guo, S. Li, and P. Pham, "Triangulating Rebound Attack on AES-like Hashing," in Advances in Cryptology – CRYPTO 2022, Y. Dodis and T. Shrimpton, Eds. Cham: Springer Nature Switzerland, 2022, pp. 94–124.

[31] E. Bellini, D. Gerault, J. Grados, Y. J. Huang, R. Makarim, M. Rachidi, and S. Tiwari, "CLAASP: a cryptographic library for the automated analysis of symmetric primitives," in International Conference on Selected Areas in Cryptography. Springer, 2023, pp. 387–408.

[32] K. Koschatko, R. Lüftenegger, and C. Rechberger, "Exploring the Six Worlds of Gröbner Basis Cryptanalysis: Application to Anemoi," Cryptology ePrint Archive, Paper 2024/250, 2024, https://eprint.iacr.org/2024/250. [Online]. Available: https://eprint.iacr.org/2024/250

[33] M. J. Steiner, "Solving Degree Bounds for Iterated Polynomial Systems," IACR Transactions on Symmetric Cryptology, vol. 2024, no. 1, p. 357–411, Mar. 2024.

[34] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "On the Indifferentiability of the Sponge Construction," in Advances in Cryptology – EUROCRYPT 2008, N. Smart, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 181–197.

[35] ——, "Keccak sponge function family main document," Submission to NIST (Round 2), vol. 3, no. 30, pp. 320–337, 2009.

[36] M. Dworkin, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," 2015-08-04 2015.

[37] B. Buchberger, "A theoretical basis for the reduction of polynomials to canonical forms," SIGSAM Bull., vol. 10, no. 3, p. 19–29, aug 1976.

[38] J. C. Faugère, "A new efficient algorithm for computing Gröbner bases without reduction to zero (F5)," in Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, ser. ISSAC '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 75–83. [Online]. Available: https://doi.org/10.1145/780506.780516

[39] J. Faugère, P. Gianni, D. Lazard, and T. Mora, "Efficient Computation of Zero-dimensional Gröbner Bases by Change of Ordering," J. Symb. Comput., vol. 16, no. 4, p. 329–344, oct 1993. [Online]. Available: https://doi.org/10.1006/jsco.1993.1051

[40] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024, https://www.gurobi.com.

9 appendix

9.1 Constraints on t-addition module

Suppose the branches with indexes $i(0 \le i \le t-1)$ are connected to a t-addition module. The variables relating to a t-addition modules include $var_i^r, con_i^r, g_i^r, h_i^r, bse_i^r, deg_i^r, 0 \le i \le t-1$, and equ_{add}^r. These variables should abide by the following constraints to reflect the mechanism of building equations within the t-addition module.

- If a variable is introduced, i.e. $var_i^r = 1$, the branch is determined:
$$var_i^r = 1 \Rightarrow g_i^r = 1, 0 \le i \le t-1.$$
- If a constant is introduced in the branch, i.e. $con_i^r = 1$, the branch is determined:
$$con_i^r = 1 \Rightarrow g_i^r = 1, 0 \le i \le t-1.$$
- The number of introduced constants around a t-addition module is at most $t-1$:
$$\sum_{i=0}^{t-1} con_i^r \le t-1.$$
- Basis can only select from determined branches:
$$g_i^r \ge bse_i^r, 0 \le i \le t-1$$
(redundant due to definition of h).
- Definition of h_i^r:
$$h_i^r = g_i^r - bse_i^r, 0 \le i \le t-1.$$
- The basis contains $t-1$ branches:
$$\sum_{i=0}^{t-1} bse_i^r = t-1.$$
- The basis should include the $t-1$ determined branches with the lowest degrees. In other words, the degrees of determined branches that are not selected into the basis should be no lower than those in the basis:
$$deg_B = \underbrace{\max\{bse_j^r \times deg_j^r | 0 \le j \le t-1\}}_{\text{the maximum degree in the basis}}$$
$$h_i^r = 1 \Rightarrow deg_B \le deg_i^r, 0 \le i \le t-1.$$
- The sum of degrees of the equations
$$equ_{add}^r = \sum_{i=0}^{t-1} (h_i^r \times deg_i^r)$$

Mengyu Chang is currently a PG student at School of Cyberspace Science and Technology, Beijing Institute of Technology. Her research interest is cryptanalysis on symmetric ciphers.
(Email: 1361587350@qq.com)

Kexin Qiao is currently an Assistant Professor at School of Cyberspace Science and Technology, Beijing Institute of Technology. She received the Ph.D. degree from the University of Chinese Academy of Sciences in 2017, then worked at Bank Card Test Center until 2020. Her research interest includes cryptanalysis on symmetric ciphers and side-channel attacks. (Email: qiaokexin0327@163.com)

Junjie Cheng received the PG degree from School of Cyberspace Science and Technology, Beijing Institute of Technology. His research interest is cryptanalysis on symmetric ciphers.
(Email: junjie.cheng@outlook.com)

Changhai Ou was born in Tongren, Guizhou Province of China, in 1989. In 2013, he received his bachelor's degree in Computer Science and Technology from School of Computer and Information Technology, Beijing Jiaotong University, China. He received his Ph.D degree in Cyber Security from Institute of Information Engineering, Chinese Academy of Sciences (i.e. School of Cyber Security, University of Chinese Academy of Sciences) in July, 2018. He is now a Research Fellow in Hardware & Embedded Systems Lab (HESL), School of Computer Science and Engineering, Nanyang Technological University, Singapore. His current research interests include cryptographic hardware and embedded system security, side channel attacks, machine learning and privacy preserving.
(E-mail:ouchanghai@whu.ac.cn)

Liehuang Zhu is currently a Professor at the Department of Cyberspace Science, Beijing Institute of Technology. He is selected into the Program for New Century Excellent Talents in University from the Ministry of Education,China. His research interests include the Internet of things, cloud computing security, Internet, and mobile security. (Email: liehuangz@bit.edu.cn)