CLAASP-MP: An Automated MILP Framework for Monomial Prediction

Emanuele Bellini, Mohamed Rachidi, and Sharwan K. Tiwari

Cryptography Research Center, Technology Innovation Institute, Abu Dhabi, UAE
emanuele.bellini@tii.ae, mohamed.rachidi@tii.ae, sharwan.tiwari@tii.ae

Abstract. As AI increasingly automates cryptanalytic programming, transparent and reproducible frameworks are essential for making generated results human-verifiable. We present CLAASP-MP, a monomial-prediction tool integrated into CLAASP and based on the three-subset division property without unknown subset (3SDP-woU). CLAASP-MP generates a Mixed Integer Linear Programming (MILP) model directly from the CLAASP component graph, supporting S-boxes, linear layers, Boolean operations, modular addition and multiplication, and linear and nonlinear feedback shift registers.

For selected output bits, CLAASP-MP computes the algebraic normal form (ANF) for small numbers of rounds, derives superpolies for a chosen cube, and bounds the algebraic degree with respect to plaintext, key, or IV variables. It detects absent monomials through vanishing ANF coefficients, zero superpolies, or degree bounds, thereby identifying balanced outputs and integral distinguishers.

We evaluate CLAASP-MP on block ciphers, permutations, and stream-cipher update functions. For block ciphers, we reproduce known distinguishers, find new cubes, and report the first integral distinguishers for SAECHAM. For Splight, we reach rounds 10 and 11, despite a previous two-subset division-property search reporting no 10-round distinguisher. On MSX-128, a 96-dimensional cube leaves 64 of 128 output bits balanced after 7 rounds. For permutations, we extend the best-known division-property distinguishers for ChaCha and Salsa from 6 and 5 rounds, respectively, to 6.75 rounds; give the first such distinguishers for Forró; and reduce the data complexity for Chaskey-Pi from 2^{64} to 2^{23} at round 3 and from 2^{127} to 2^{122} at round 4. For stream ciphers, we find new 36-dimensional cubes for Bivium, extend the tight degree bounds for Trivium to later initialization clocks, and reproduce published superpolies.

These results establish CLAASP-MP as a unified, practical, and human-verifiable framework for monomial prediction across symmetric primitives.

Keywords: Division property · Integral distinguishers · Superpoly · Cube attacks · Monomial prediction

1 Introduction

Despite the rapid progress of AI in automating cryptanalytic tasks, AI-generated results still require careful human verification, as plausible-looking models, implementations, and attacks may contain subtle errors. By providing standardized, transparent, and reproducible models, automated frameworks make such results easier to inspect, validate, and reproduce, thereby turning the speed and creativity of AI-assisted cryptanalysis into trustworthy scientific contributions. Beyond supporting AI-assisted research, these frameworks remain valuable for validating manually derived results and systematically discovering new cryptanalytic properties. In this work, we develop a framework to automate monomial prediction for symmetric primitives.

Symmetric cryptanalysis often exploits the algebraic structure of a cipher, for example, by studying which monomials appear in the algebraic normal form (ANF) of its output bits and how the algebraic degree evolves across rounds. Such information is central to integral and cube attacks, as well as degree-based distinguishers. However, for realistic primitives, directly expanding ANFs or tracking monomials quickly becomes infeasible.

The *division property*, introduced by Todo at Eurocrypt 2015 [33], provides a principled way to track monomial propagation. It generalizes the classical *integral property* of Daemen et al. [10], where one studies the XOR of ciphertext bits over a structured set of plaintexts (a *cube*).

When an output bit of a cipher is summed over a cube of input values, the sum vanishes whenever the algebraic degree of that bit is below the dimension of the cube. In formulas, if the cipher output bit $f : \{0,1\}^n \to \{0,1\}$ is evaluated over a cube $\mathcal{C}_I$, that is, the affine subspace obtained by letting the input coordinates indexed by $I \subseteq \{0,\ldots,n-1\}$ vary freely while the remaining coordinates are fixed to constants, then $\bigoplus_{x\in\mathcal{C}_I} f(x) = 0$ whenever $\deg(f) < \dim(\mathcal{C}_I) = |I|$. When the sum is non-constant, the result is a key-dependent polynomial, called *superpoly*, that can be exploited for key recovery attacks.

Todo's *two-subset division property* [33] formalizes this phenomenon by tracking sets of monomials across rounds. It enabled improved integral distinguishers and led to the first theoretical analysis of full-round MISTY1 [32]. Todo and Morii [35] extended this to the *bit-based* and *three-subset* formulation, offering finer monomial tracking and tighter algebraic degree bounds. Boura and Canteaut [6] provided an equivalent parity-set view.

Direct propagation of the bit-based or three-subset division property becomes infeasible as the number of rounds or the state size increases. Xiang et al. [37] addressed this by encoding these propagation rules as linear constraints in a MILP framework, enabling solver-driven searches for integral distinguishers.

A key simplification came from Hao et al. [14], who showed that when key bits are treated as inputs, the "unknown" subset in the three-subset formulation can be removed without loss for public functions, yielding the three-subset division property *without unknown subset* (3SDP-woU). Building on this, Hu et al. [19] introduced *monomial prediction* (MP), which determines whether an input monomial contributes to a target output monomial via the parity of monomial

trails. Recent works have further strengthened MP by improving the modeling of nonlinear components such as modular addition, as well as by developing nested monomial prediction and core monomial prediction techniques [18,11,16,15,21].

1.1 Related Works

Early attempts to automate division property propagation include the MILP formulation of Xiang et al. [37], later refined by simplified rules [20], pruning techniques [36], and the 3SDP-woU formulation [14].

A closely related automated framework is the SAT-based tool Solvatore by Eskandari et al. [13], which models division-property transitions as propositional constraints and searches for integral distinguishers. This approach is effective but focuses on integrals and constructs a new SAT instance for each cube. Section 5 compares it with our work.

Overall, existing tools either target integral distinguishers or rely on component-wise, cipher-specific encodings. This motivates the development of unified frameworks that support ANF extraction, degree analysis, and cube-based attacks within one modeling abstraction.

1.2 Our Contribution

In this work, we introduce a generic and fully automated MILP framework integrated into the CLAASP library [2] (`https://github.com/Crypto-TII/claasp`). The 3SDP-woU formulation [14] and monomial prediction [19] are known in theory, but prior implementations remain cipher-specific and require manual MILP modeling for every new primitive. To the best of our knowledge, CLAASP-MP is the first open-source framework that, starting from the cipher's internal directed acyclic graph (DAG) representation in CLAASP, automatically generates a single global MILP model and uniformly supports the full range of division property-based algebraic cryptanalytic tasks. Linear constraints are generated automatically for the core building blocks of modern symmetric primitives, including S-boxes, linear layers, Boolean word operations, modular addition and multiplication, and linear and nonlinear feedback shift register (FSR) components. Unlike existing general-purpose tools, such as Solvatore [13], which only check the feasibility of monomial trails to detect integral distinguishers, CLAASP-MP enumerates all feasible trails and counts their combined parity, which captures both forms of cancellation (no trail at all and an even number of trails). A standard divide-and-conquer strategy [18,15] on top of the global model accelerates the enumeration on instances where the global trail count is large. This unified model applies to fixed-input/output-length primitives: block ciphers, stream ciphers' update functions, and permutations.

For block ciphers, we reproduce previously known integral distinguishers and identify new cubes for several of them. For SAECHAM, no integral analysis appears in the literature, so to the best of our knowledge, ours are the first such results. For Splight, whose designers report a 9-round distinguisher and conclude from a two-subset bit-based division property search that no 10-round integral

distinguisher exists, CLAASP-MP reaches rounds 10 and 11. For MSX-128, we illustrate the modular multiplication model with a 96-dimensional cube that leaves 64 output bits balanced at round 7.

For permutations, we use CLAASP-MP to extend the best-known division-property integral distinguishers of ChaCha and Salsa core permutations from 6 rounds to 6.75 and from 5 to 6.75 rounds, respectively. For Forró, every published analysis is differential-linear, so to the best of our knowledge ours are the first integral results. For Chaskey-Pi, which has already been studied with automated integral tools, CLAASP-MP reproduces the published distinguishers exactly and lowers the data complexity from 2^{64} to 2^{23} at round 3 and from 2^{127} to 2^{122} at round 4. For stream ciphers, CLAASP-MP finds many new 36-dimensional integral cubes for Bivium and extends the tight degree bounds for Trivium to later initialization clocks while reproducing many known superpolies.

Finally, we compare our approach with the SAT-based tool Solvatore and provide practical implementation examples in the Appendix. All experiments reported in this paper were performed on a machine equipped with an Apple M1 Max processor and 32 GB of memory, where the main bottleneck in practice is solver time rather than memory.

Our contribution is thus both the framework and the new results obtained with it, and we hope the tool will serve as a basis for such follow-up work.

1.3 Outline

Preliminaries about division property, monomial prediction, algebraic attacks, and MILP modeling for propagation rules are provided in Section 2. Section 3 describes CLAASP-MP, and how the MILP model is constructed from the CLAASP component graph. Section 4 reports experimental results on block ciphers, permutations, and stream ciphers. Section 5 discusses differences between our MILP framework and a SAT-based approach, and implementation details and usage examples appear in the Appendix.

2 Preliminaries

Notations. Let $\mathbb{F}_2 = \{0, 1\}$ and let $\mathbb{F}_2^n$ denote the n-dimensional vector space over $\mathbb{F}_2$. For vectors $\boldsymbol{x} = (x_0, \ldots, x_{n-1})$ and $\boldsymbol{y} = (y_0, \ldots, y_{n-1})$ in $\mathbb{F}_2^n$, we write $\boldsymbol{x} \preceq \boldsymbol{y}$ if $x_i \leq y_i$ for all i, and $\boldsymbol{x} \succeq \boldsymbol{y}$ if $x_i \geq y_i$ for all i. The Hamming weight of $\boldsymbol{x}$, denoted by $\mathrm{wt}(\boldsymbol{x})$, is the number of ones in $\boldsymbol{x}$. For a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$, its algebraic normal form (ANF) is defined as

$$f(\boldsymbol{x}) = \bigoplus_{\boldsymbol{u} \in \mathbb{F}_2^n} a_{\boldsymbol{u}}\, \boldsymbol{x}^{\boldsymbol{u}}, \quad a_{\boldsymbol{u}} \in \mathbb{F}_2,$$

where $\boldsymbol{x}^{\boldsymbol{u}} = \prod_{i=0}^{n-1} x_i^{u_i}$ denotes the *monomial* corresponding to $\boldsymbol{u}$. The coefficient $a_{\boldsymbol{u}}$ can be expressed as $a_{\boldsymbol{u}} = \bigoplus_{\boldsymbol{x} \preceq \boldsymbol{u}} f(\boldsymbol{x})$, and the algebraic degree of f is $\deg(f) = \max_{\boldsymbol{u}: a_{\boldsymbol{u}}=1} \mathrm{wt}(\boldsymbol{u})$. It is convenient to denote a monomial function by $\pi_{\boldsymbol{u}}(\boldsymbol{x}) = \boldsymbol{x}^{\boldsymbol{u}}$, so that the ANF of f is the XOR of all $\pi_{\boldsymbol{u}}$ with $a_{\boldsymbol{u}} = 1$.

2.1 Division property

The division property, introduced by Todo [33], traces how monomials propagate through the round functions of a cipher and generalizes integral cryptanalysis. It provides a structured way to reason about the presence or absence of monomials in the ANF of an output bit.

Let $\mathbb{X} \subseteq \mathbb{F}_2^n$ be a multiset of input vectors. For a binary vector $\boldsymbol{u} \in \mathbb{F}_2^n$, the parity of the monomial $\pi_{\boldsymbol{u}}(\boldsymbol{x})$ over $\mathbb{X}$ is

$$P_{\boldsymbol{u}}(\mathbb{X}) = \bigoplus_{\boldsymbol{x}\in\mathbb{X}} \pi_{\boldsymbol{u}}(\boldsymbol{x}) = \bigoplus_{\boldsymbol{x}\in\mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}}.$$

If $P_{\boldsymbol{u}} = 0$, the contribution of $\boldsymbol{x}^{\boldsymbol{u}}$ cancels in the sum; if $P_{\boldsymbol{u}} = 1$, the monomial $\boldsymbol{x}^{\boldsymbol{u}}$ is active. For any Boolean function $f(\boldsymbol{x}) = \sum_{\boldsymbol{u}} a_{\boldsymbol{u}}\, \boldsymbol{x}^{\boldsymbol{u}}$, we have

$$\bigoplus_{\boldsymbol{x}\in\mathbb{X}} f(\boldsymbol{x}) \;=\; \sum_{\boldsymbol{u}} a_{\boldsymbol{u}}^f\, P_{\boldsymbol{u}}(\mathbb{X}),$$

so the effect of each coefficient $a_{\boldsymbol{u}}^f$ on the XOR over $\mathbb{X}$ is fully determined by the parities $P_{\boldsymbol{u}}$. If the division property ensures that certain $P_{\boldsymbol{u}}$ are always zero, then the corresponding terms vanish from the XOR. Conversely, if $P_{\boldsymbol{u}}$ may be nonzero, the monomial $\boldsymbol{x}^{\boldsymbol{u}}$ can influence the output. This allows us to reason about which monomials can or cannot appear in the ANF of complex functions.

Two-subset bit-based division property. In the bit-based division property [35], each exponent vector $\boldsymbol{u}$ belongs either to a set of monomials with unknown parity or to the set of monomials whose parity is guaranteed to be zero. Let $\mathbb{K}$ be the set of exponent vectors with unknown parity. The multiset $\mathbb{X}$ is said to satisfy the two-subset division property $\mathrm{D}^{1^n}_{\mathbb{K}}$ if

$$P_{\boldsymbol{u}} = \bigoplus_{\boldsymbol{x}\in\mathbb{X}} \pi_{\boldsymbol{u}}(\boldsymbol{x}) = \begin{cases} \text{unknown}, & \text{if there exists } \boldsymbol{k} \in \mathbb{K} \text{ such that } \boldsymbol{u} \succeq \boldsymbol{k}, \\ 0, & \text{otherwise.} \end{cases}$$

For a multiset of chosen plaintexts (or IVs), one first determines its division property $\mathrm{D}^{1^n}_{\mathbb{K}}$ and then propagates $\mathbb{K}$ through each round function.

Integral distinguishers are detected by checking whether certain unit vectors are missing in the propagated division set. Let $\mathbb{K}$ propagate to $\mathbb{K}'$ and let $\mathbf{e}_i$ be the unit vector corresponding to an output bit y_i. If $\mathbf{e}_i \notin \mathbb{K}'$, then by definition no $\boldsymbol{k} \in \mathbb{K}'$ satisfies $\mathbf{e}_i \succeq \boldsymbol{k}$, so the parity $P_{\mathbf{e}_i}$ must be zero. Since $\pi_{\mathbf{e}_i}(\boldsymbol{y}) = y_i$, we obtain $\bigoplus_{\boldsymbol{x}\in\mathbb{X}} y_i = 0$, i.e., y_i is balanced over the chosen input structure. If all unit vectors appear in $\mathbb{K}'$, no output bit is guaranteed to be balanced.

Another important algebraic consequence [34] of the division property arises when analyzing a public function $f : \mathbb{F}_2^n \to \mathbb{F}_2$. Let $\mathbb{X}$ be a cube $\mathcal{C}_I$ (an affine subspace determined by a set of public coordinates I) with division property $\mathrm{D}^{1^n}_{\{\boldsymbol{k}\}}$. If no division trail from the exponent $\boldsymbol{k}$ reaches the unit vector $\mathbf{1}$, then $\bigoplus_{\boldsymbol{x}\in\mathbb{X}} f(\boldsymbol{x}) = 0$. Expanding this sum as $\sum_{\boldsymbol{u}} a_{\boldsymbol{u}} P_{\boldsymbol{u}}$ and splitting it at $\boldsymbol{k}$ leaves only $\sum_{\boldsymbol{u}\succeq\boldsymbol{k}} a_{\boldsymbol{u}} P_{\boldsymbol{u}} = 0$, since $P_{\boldsymbol{u}} = 0$ for all $\boldsymbol{u} \not\succeq \boldsymbol{k}$. For $\boldsymbol{u} \succeq \boldsymbol{k}$, $P_{\boldsymbol{u}}$ depends on the fixed non-cube variables, and the equality must hold for every choice of these constants. Therefore, $a_{\boldsymbol{u}} = 0$ for all $\boldsymbol{u} \succeq \boldsymbol{k}$.

Three-subset bit-based division property. The three-subset division property [35,36] refines this by distinguishing monomials of odd parity in addition to those of even parity. Instead of classifying monomials only as *unknown* or *always even*, it introduces a class for monomials whose parity is guaranteed to be *odd*. This extra information allows certain integral properties to be detected that may be missed under the two-subset formulation.

Let $\mathbb{K}, \mathbb{L} \subseteq \mathbb{F}_2^n$ be two sets of exponent vectors, where $\mathbb{K}$ contains the monomials whose parity is unknown and $\mathbb{L}$ contains those whose parity is one. A multiset $\mathbb{X} \subseteq \mathbb{F}_2^n$ is said to satisfy the three-subset division property $\mathrm{D}^{1^n}_{\mathbb{K},\mathbb{L}}$ if

$$P_{\boldsymbol{u}} = \bigoplus_{\boldsymbol{x}\in\mathbb{X}} \pi_{\boldsymbol{u}}(\boldsymbol{x}) = \begin{cases} \text{unknown}, & \text{if there exists } \boldsymbol{k} \in \mathbb{K} \text{ with } \boldsymbol{u} \succeq \boldsymbol{k}, \\ 1, & \text{if there exists } \boldsymbol{l} \in \mathbb{L} \text{ with } \boldsymbol{u} = \boldsymbol{l}, \\ 0, & \text{otherwise.} \end{cases}$$

This separates monomials that are always even, always odd, or undetermined, giving more precise information than the two-subset version. For example, the three-subset property was essential in proving that the experimentally observed 15-round integral property of Simon 32 holds for all keys.

For a public Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$, the unknown subset $\mathbb{K}$ and odd-parity subset $\mathbb{L}$ propagate independently. The authors of [36] showed that odd parity information alone can determine the presence of specific monomials in the ANF of f. A simplified statement (see [14]) is: let $f(\boldsymbol{x}) = \sum_{\boldsymbol{u}} a_{\boldsymbol{u}} \boldsymbol{x}^{\boldsymbol{u}}$ and let $\boldsymbol{\ell}$ be an n-dimensional bit vector. If the initial division property $D^{1^n}_{\emptyset,\{\boldsymbol{\ell}\}}$ propagates through f to $D^1_{\emptyset,\{1\}}$, then $a_{\boldsymbol{\ell}} = 1$. Thus, the three-subset property can determine the presence of specific ANF monomials.

Three-subset division property without unknown subset (3SDP-woU). In the original three-subset division property, the unknown subset $\mathbb{K}$ represents monomials whose parity depends on secret key bits, and key addition introduces new unknowns at every round.

This makes it difficult to model three-subset propagation in MILP or SMT frameworks. Hu and Wang [20] proposed a variant of the three-subset with simplified propagation rules, while Wang *et al.* [36] combined MILP modeling with a pruning technique to improve accuracy. However, both approaches become computationally demanding for large ciphers.

For public Boolean functions, however, the unknown subset is unnecessary. Hao *et al.* [14] observed that if the round function and the key schedule are both treated as part of the public mapping, then key bits can be regarded as input variables. Every monomial then has a fixed ANF coefficient, and no parity uncertainty arises. Thus, the unknown subset can be removed without loss of accuracy, leading to the *three-subset division property without unknown subset*.

Let $\mathbb{X} \subseteq \mathbb{F}_2^n$ be a multiset of input vectors and let $\widetilde{\mathbb{L}}$ be a multiset of n-bit exponent vectors. The set $\mathbb{X}$ is said to satisfy the 3SDP-woU, denoted $\mathrm{D}^{1^n}_{\widetilde{\mathbb{L}}}$, if

for each $\boldsymbol{u} \in \mathbb{F}_2^n$,

$$P_{\boldsymbol{u}} = \bigoplus_{\boldsymbol{x} \in \mathbb{X}} \pi_{\boldsymbol{u}}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{u} \text{ appears an odd number of times in } \widetilde{\mathbb{L}}, \\ 0, & \text{otherwise.} \end{cases}$$

This key-independent formulation allows the propagation of monomial parities to be modeled purely algebraically, enabling automatic analysis through MILP constraints in our monomial prediction framework.

2.2 Monomial prediction

Monomial prediction (MP) provides an algebraic formulation of 3SDP-woU by characterizing when an input monomial contributes to a given output monomial.

Let $\boldsymbol{f} : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a vectorial Boolean function, $\boldsymbol{f}(\boldsymbol{x}) = (f_0(\boldsymbol{x}), \ldots, f_{m-1}(\boldsymbol{x}))$. For $\boldsymbol{v} = (v_0, \ldots, v_{m-1}) \in \mathbb{F}_2^m$, $\boldsymbol{y}^{\boldsymbol{v}} = \prod_{i=0}^{m-1} f_i(\boldsymbol{x})^{v_i}$. Expanding $\boldsymbol{y}^{\boldsymbol{v}}$ in the input variables $\boldsymbol{x}$ gives $\boldsymbol{y}^{\boldsymbol{v}} = \bigoplus_{\boldsymbol{u} \in \mathbb{F}_2^n} a_{\boldsymbol{u},\boldsymbol{v}}\, \boldsymbol{x}^{\boldsymbol{u}}$ with $a_{\boldsymbol{u},\boldsymbol{v}} \in \mathbb{F}_2$, where $a_{\boldsymbol{u},\boldsymbol{v}} = 1$ indicates that $\boldsymbol{x}^{\boldsymbol{u}}$ appears in the ANF of $\boldsymbol{y}^{\boldsymbol{v}}$, denoted $\boldsymbol{x}^{\boldsymbol{u}} \to \boldsymbol{y}^{\boldsymbol{v}}$. The goal of MP is to decide, for given $\boldsymbol{u}$ and $\boldsymbol{v}$, whether $a_{\boldsymbol{u},\boldsymbol{v}} = 1$.

Monomial trails. In practice, $\boldsymbol{f}$ is given as the composition of simpler round (or layer) functions: $\boldsymbol{f} = \boldsymbol{f}^{(r-1)} \circ \boldsymbol{f}^{(r-2)} \circ \cdots \circ \boldsymbol{f}^{(0)}$, where each $\boldsymbol{f}^{(i)} : \mathbb{F}_2^{n_i} \to \mathbb{F}_2^{n_{i+1}}$ is a simple (round) operation of the cipher. Let $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(i+1)}$ denote the input and output of $\boldsymbol{f}^{(i)}$ so that $\boldsymbol{x}^{(i+1)} = \boldsymbol{f}^{(i)}(\boldsymbol{x}^{(i)})$.

Definition 1. *A sequence of monomials*

$$(\boldsymbol{x}^{(0)})^{\boldsymbol{u}^{(0)}} \to (\boldsymbol{x}^{(1)})^{\boldsymbol{u}^{(1)}} \to \cdots \to (\boldsymbol{x}^{(r)})^{\boldsymbol{u}^{(r)}}$$

is called an r-round monomial trail *from* $(\boldsymbol{x}^{(0)})^{\boldsymbol{u}^{(0)}}$ *to* $(\boldsymbol{x}^{(r)})^{\boldsymbol{u}^{(r)}}$ *if for each* $i = 0, \ldots, r-1$ *the propagation* $(\boldsymbol{x}^{(i)})^{\boldsymbol{u}^{(i)}} \to (\boldsymbol{x}^{(i+1)})^{\boldsymbol{u}^{(i+1)}}$ *is valid according to the algebraic expansion of* $\boldsymbol{f}^{(i)}$*. We write* $(\boldsymbol{x}^{(0)})^{\boldsymbol{u}^{(0)}} \rightsquigarrow (\boldsymbol{x}^{(r)})^{\boldsymbol{u}^{(r)}}$ *if at least one such trail exists.*

A valid trail indicates that the input monomial may contribute to the given output monomial, and since several trails may lead to the same one, it is the *parity* that decides whether the monomial actually appears in the ANF.

Monomial hull and parity. Let $\mathsf{Hull}(\boldsymbol{u}^{(0)}, \boldsymbol{u}^{(r)})$ denote the set of all trails connecting $\boldsymbol{x}^{\boldsymbol{u}^{(0)}}$ and $\boldsymbol{x}^{\boldsymbol{u}^{(r)}}$: $|\mathsf{Hull}(\boldsymbol{u}^{(0)}, \boldsymbol{u}^{(r)})| = \#\{\text{valid trails } \boldsymbol{x}^{\boldsymbol{u}^{(0)}} \rightsquigarrow \boldsymbol{x}^{\boldsymbol{u}^{(r)}}\}$. The parity determines the ANF coefficient: $a_{\boldsymbol{u}^{(0)},\boldsymbol{u}^{(r)}} = |\mathsf{Hull}(\boldsymbol{u}^{(0)}, \boldsymbol{u}^{(r)})| \bmod 2$, so $\boldsymbol{x}^{\boldsymbol{u}^{(0)}}$ appears in the ANF of $\boldsymbol{x}^{\boldsymbol{u}^{(r)}}$ precisely when the number of trails is odd. The number of trails satisfies the recursion

$$|\mathsf{Hull}(\boldsymbol{u}^{(0)}, \boldsymbol{u}^{(i)})| = \sum_{\substack{\boldsymbol{u}^{(i-1)}: \\ (\boldsymbol{x}^{(i-1)})^{\boldsymbol{u}^{(i-1)}} \to (\boldsymbol{x}^{(i)})^{\boldsymbol{u}^{(i)}}}} |\mathsf{Hull}(\boldsymbol{u}^{(0)}, \boldsymbol{u}^{(i-1)})|.$$

Equivalently, for $\boldsymbol{f} = \boldsymbol{f}^{(r-1)} \circ \cdots \circ \boldsymbol{f}^{(0)}$ the monomial prediction criterion of Hu et al. [19] states that $(\boldsymbol{x}^{(0)})^{\boldsymbol{u}^{(0)}} \to (\boldsymbol{x}^{(r)})^{\boldsymbol{u}^{(r)}}$ if and only if $|\mathsf{Hull}(\boldsymbol{u}^{(0)}, \boldsymbol{u}^{(r)})| \equiv 1 \pmod 2$. In the 3SDP-woU framework, this parity propagation is expressed algebraically and captured by the MILP constraints used in our implementation.

2.3 Cube attacks and integral distinguishers

Let an output bit of a symmetric cipher be represented as a Boolean function $f : \mathbb{F}_2^n \times \mathbb{F}_2^m \to \mathbb{F}_2$, $(\boldsymbol{x}, \boldsymbol{k}) \mapsto f(\boldsymbol{x}, \boldsymbol{k})$, where $\boldsymbol{x}$ denotes public inputs (plaintext or IV) and $\boldsymbol{k}$ the key bits. Its ANF can be written as $f(\boldsymbol{x}, \boldsymbol{k}) = \sum_{u \in \{0,1\}^n,\, v \in \{0,1\}^m} a_{u,v}\, \boldsymbol{x}^{\boldsymbol{u}} \boldsymbol{k}^{\boldsymbol{v}}$. Fix a cube $\mathcal{C}_I \subseteq \mathbb{F}_2^n$, an affine subspace determined by a set of public coordinates I. Split $\boldsymbol{x} = (\boldsymbol{x}_I, \boldsymbol{x}_{\overline{I}})$, and fix $\boldsymbol{x}_{\overline{I}} = \boldsymbol{c}$. Then f can be uniquely written as $f(\boldsymbol{x}_I, \boldsymbol{c}, \boldsymbol{k}) = S_{\mathcal{C}_I}(\boldsymbol{k})\, \boldsymbol{x}_I + Q(\boldsymbol{x}_I, \boldsymbol{k})$, where $\boldsymbol{x}_I = \prod_{i \in I} x_i$ and Q is not divisible by $\boldsymbol{x}_I$. Summing over the cube eliminates all terms of Q and yields

$$\bigoplus_{\boldsymbol{x}_I \in \mathcal{C}_I} f(\boldsymbol{x}_I, \boldsymbol{c}, \boldsymbol{k}) = S_{\mathcal{C}_I}(\boldsymbol{k}).$$

Three cases arise. If $S_{\mathcal{C}_I}(\boldsymbol{k}) \equiv 0$, the superpoly is identically zero and the output is balanced over the cube, giving an integral distinguisher. If $S_{\mathcal{C}_I}(\boldsymbol{k}) \equiv 1$, the cube sum is constant in the key and equals one for every key, which also yields an integral distinguisher. Otherwise, $S_{\mathcal{C}_I}(\boldsymbol{k})$ is non-constant and the cube sum is key-dependent. In this last case, if $S_{\mathcal{C}_I}(\boldsymbol{k})$ is low degree or sparse and $\boldsymbol{c}$ is fixed (commonly $\boldsymbol{c} = 0$), then evaluating the cipher on $\mathcal{C}_I$ provides relations in the key variables.

When $S_{\mathcal{C}_I}$ is linear, these relations can directly reveal key bits [12]. A main practical task is to choose cubes for which $S_{\mathcal{C}_I}(\boldsymbol{k})$ is simple, or to locate outputs whose degree in $\boldsymbol{x}$ remains low after several rounds; this cube and target selection problem is precisely what the division property and monomial prediction are designed to address. When analyzing a new primitive, the choice is guided by CLAASP-MP itself: its degree bounds indicate which output bits saturate and at what dimension. Specific cubes are then found by searching candidate cubes and testing each with the tool, exploiting any word or symmetry alignment the primitive exhibits.

If the algebraic degree of $f(\boldsymbol{x}, \boldsymbol{k})$ with respect to $\boldsymbol{x}$ is at most d then the XOR of f over any affine subspace of dimension $d+1$ vanishes: $\bigoplus_{x \in \mathcal{A}} f(x, k) = 0$. This principle underlies integral and higher-order differential attacks [10,22]. Degree-based distinguishers use it to estimate the number of secure rounds and to detect degree drops caused by cancellations. Division property based degree bounds are typically tighter than classical ones [5], since they track the evolution of individual monomials rather than only the global degree. If at some intermediate state of a block cipher, many output bits are balanced over a low-dimensional affine space $\mathcal{A}$, one can often append extra rounds and extend the distinguisher into a partial key recovery attack. For stream ciphers, balanced keystream bits yield distinguishers and cannot be used for key recovery.

All integral distinguishers reported by CLAASP-MP are deterministic, holding for every key, since they come either from a cube sum that is constant over all keys or from a degree bound. Probabilistic integral properties, which hold only for a fraction of keys or inputs, are statistical rather than algebraic, and lie outside the monomial-prediction framework.

An alternative approach within algebraic cryptanalysis models the cipher as a system of equations in plaintext and ciphertext bits and attempts to recover key variables using Gröbner basis, XL, or SAT-solving techniques [29,28,27]. These methods usually require only a few known plaintext–ciphertext pairs but are computationally demanding.

The division property and its monomial prediction provide algebraic rules to determine, without expanding the full ANF, which input monomials can reach a chosen output or contribute to the parity of outputs over a cube. The MILP framework CLAASP-MP implements these rules under 3SDP-woU, enabling automated algebraic analysis across different primitives.

2.4 Propagation Rules and MILP Modeling of Division Property

The monomial prediction framework requires a precise description of how basic Boolean operations affect the propagation of monomials between their input and output variables. Under the 3SDP-woU framework, the propagation rules for the basic Boolean operations [37,14] form the basis of the MILP model used in CLAASP. Each rule specifies which pairs of input and output exponent vectors are valid, allowing monomial trails to be encoded by linear constraints.

For each operation, a pair $(\boldsymbol{u}, \boldsymbol{v})$ is *valid* if the monomial $\boldsymbol{x}^{\boldsymbol{u}}$ contributes to $\boldsymbol{y}^{\boldsymbol{v}}$ in the algebraic expansion; the set of all such pairs constitutes its propagation rule. Because any Boolean function can be decomposed into `XOR`, `AND`, `NOT`, and `COPY`, the propagation rules of these basic operations determine all monomial trails. We use binary MILP variables $a_1, a_2, \ldots$ for the input exponents and $b, b_1, b_2, \ldots$ for the output exponents.

XOR Let $y = x_1 \oplus x_2 = x_1 + x_2$ over $\mathbb{F}_2$. The algebraic expression of y contains the monomials $\{x_1, x_2\}$. A monomial $x_1^{u_1} x_2^{u_2}$ contributes to y^v if $v = u_1 \oplus u_2$ with $(u_1, u_2) \neq (1, 1)$. Hence, the valid monomial propagations are: $x_1^0 x_2^0 \to y^0$, $x_1^1 x_2^0 \to y^1$, $x_1^0 x_2^1 \to y^1$. The following corresponding exponent transitions define the propagation rules: $(0,0) \to 0$, $(1,0) \to 1$, $(0,1) \to 1$, and that can be captured by the linear equality $b = a_1 + a_2$, which holds for all valid exponent transitions of the XOR operation.

AND Let $y = x_1 x_2$ over $\mathbb{F}_2$. A monomial $x_1^{u_1} x_2^{u_2}$ contributes to y^v if and only if $(u_1, u_2, v) \in \{(0,0,0), (1,1,1)\}$, giving the transitions: $(0,0) \to 0$, $(1,1) \to 1$. These are modeled by the linear equalities $b = a_1, \quad b = a_2$, ensuring that $b = 1$ exactly when both input exponents are 1.

NOT Let $y = \neg x = 1 + x$ over $\mathbb{F}_2$. The monomials of y are $\{1, x\}$, giving transitions: $0 \to 0$, $0 \to 1$, $1 \to 1$. These are represented by the inequality $b \geq a$, so an active input exponent always activates the output exponent.

COPY Let $(y_1, y_2) = (x, x)$. The valid exponent transitions are $(0) \to (0, 0)$, $(1) \to (1, 0)$, $(1) \to (0, 1)$, $(1) \to (1, 1)$. These are modeled by $a \geq b_1, \quad a \geq b_2, \quad b_1 + b_2 \geq a$, expressing that an active input exponent may activate one or both outputs.

3 Description of CLAASP-MP

This section describes how the 3SDP-woU monomial prediction framework is instantiated inside the CLAASP library. CLAASP-MP is a new analysis module integrated into CLAASP [2]. Once a cipher is implemented in CLAASP, CLAASP-MP automatically constructs the corresponding MILP model by encoding the propagation rules of basic components described in Section 2.4 along the DAG representation of a symmetric primitive. Using this model, CLAASP-MP can compute, for any selected output bit, the ANF, and determine tight upper bounds on the algebraic degree. The same formulation also enables the automated detection of monomial absence and balancedness, providing integral distinguishers directly from the model.

3.1 Cipher Representation in CLAASP

In CLAASP, each primitive is a DAG whose nodes are components (S-boxes, linear transformations, Boolean word operations, modular addition, FSR updates, etc.) and whose edges represent bit connections. Each component has a unique identifier, a type, fixed input and output bit sizes, and a description of how its inputs are taken from the outputs of preceding components. This DAG is sufficient for CLAASP-MP to build its MILP model; every edge is assigned an exponent variable. An illustration is given in Appendix D. CLAASP includes more than 100 symmetric primitives, all immediately compatible with CLAASP-MP.

3.2 MILP Modeling of Components

CLAASP-MP builds a single MILP model $\mathcal{M}$ by processing the DAG components in topological order.

Boolean word operations. Boolean word operations such as `XOR`, `AND`, `NOT`, and `COPY` are modeled at the bit level in CLAASP-MP. Let the component $\mathcal{C}$ have wordsize w, and let (a_i), (b_i), (c_i) denote the exponent variables associated with its input and output words. These variables are created once by the MILP model $\mathcal{M}$ according to bit-level connections defined by the DAG. For these

operations, the 3SDP-woU propagation rules of Section 2.4 translate into the linear constraints per bit. CLAASP-MP inserts these constraints into a MILP model $\mathcal{M}$ exactly as shown in algorithm 1. For simplicity, we illustrate the two-input case (or the two-output case for `COPY`); the extension to additional inputs or outputs is straightforward.

Algorithm 1: MILP constraints for Boolean word operations

`XOR` component: two input words (a_i), (b_i), one output word (c_i)
for $i = 0$ **to** $w - 1$ **do**
    $\mathcal{M}$.addConstr$\big(c_i = a_i + b_i\big)$
`AND` component: two input words (a_i), (b_i), one output word (c_i)
for $i = 0$ **to** $w - 1$ **do**
    $\mathcal{M}$.addConstr$(c_i = a_i)$
    $\mathcal{M}$.addConstr$(c_i = b_i)$
`NOT` component: one input word (a_i), one output word (b_i)
for $i = 0$ **to** $w - 1$ **do**
    $\mathcal{M}$.addConstr$(b_i \geq a_i)$
`COPY` component: one input word (a_i), two output words (b_i), (c_i)
for $i = 0$ **to** $w - 1$ **do**
    $\mathcal{M}$.addConstr$(a_i \geq b_i)$
    $\mathcal{M}$.addConstr$(a_i \geq c_i)$
    $\mathcal{M}$.addConstr$(b_i + c_i \geq a_i)$

S-box modeling For an n-bit S-box $F(\boldsymbol{x}) = (f_0, \ldots, f_{n-1})$, CLAASP-MP builds a tight MILP model of its 3SDP-woU monomial transitions [37]. For a fixed output exponent $\boldsymbol{v}$, the product $\prod_{k:\, v_k=1} f_k(\boldsymbol{x})$ expands over $\mathbb{F}_2$ into a sum of input monomials $\boldsymbol{x}^{\boldsymbol{u}}$, and every such $\boldsymbol{u}$ is a valid transition $\boldsymbol{u} \to \boldsymbol{v}$. Taking all 2^n output exponents $\boldsymbol{v}$ gives the exact transition set $T = \{(\boldsymbol{u}, \boldsymbol{v})\}$.

CLAASP-MP then describes T by linear inequalities whose 0/1 solutions are exactly the points of T, so the model is tight. For S-boxes of up to five bits it uses the convex-hull facets of T [37]. Every 0/1 vector is a vertex of the unit cube, so the hull of T contains no other 0/1 point. For six-bit and larger S-boxes the convex hull is expensive to compute, so CLAASP-MP minimizes the description with the Espresso logic minimizer [7]. The points of T are marked as 1 in a truth table, and Espresso returns an equivalent product of sums, in which each clause gives one inequality; the steps are given in algorithm 2.

Linear layer modeling Linear layers $L(\boldsymbol{x}) = M\boldsymbol{x}$ with $M \in \mathbb{F}_2^{m \times n}$ are stored in CLAASP's component `description` field. For permutations and rotations, exponent propagation is structural. Each output bit is identified with the input bit it is wired from, giving $b_i = a_{\pi(i)}$ in the MILP model.

Algorithm 2: Tight MILP constraints for an S-box component

Input: MILP model $\mathcal{M}$, an n-bit S-box $F = (f_0, \ldots, f_{n-1})$

Compute the ANFs $f_0, \ldots, f_{n-1}$ and set $T \leftarrow \emptyset$
for *each output exponent* $\boldsymbol{v} \in \mathbb{F}_2^n$ **do**
  expand $\prod_{k:\, v_k=1} f_k(\boldsymbol{x})$ over $\mathbb{F}_2$
  for *each input monomial* $\boldsymbol{x}^{\boldsymbol{u}}$ *in this product* **do**
    $T \leftarrow T \cup \{(\boldsymbol{u}, \boldsymbol{v})\}$
if *F has at most five bits* **then**
  $I \leftarrow$ convex-hull facets of the points of T
else
  $I \leftarrow$ inequalities from the Espresso product-of-sums of T
for *each inequality in I* **do**
  add its linear constraint on the exponent variables of F to $\mathcal{M}$

For general linear layers, an output bit $y_j = \sum_{i:M_{j,i}=1} x_i$ (over $\mathbb{F}_2$) depends on all inputs x_i with a 1 in row j. CLAASP-MP creates for each input exponent a_i as many copies as the number of ones in column i of M (the number of outputs that input i feeds), constraining them exactly as in the `COPY` operation. Each occurrence of a_i in an output equation uses a distinct copy (algorithm 3). This yields a uniform MILP encoding for all linear layers.

Algorithm 3: MILP constraints for a linear layer

Input: MILP model $\mathcal{M}$, linear component L with matrix M

for *each input bit i* **do**
  let w_i = number of ones in column i of M
  create copies $a_i^1, \ldots, a_i^{w_i}$: $\mathcal{M}$.addConstr$(a_i \geq a_i^\ell)$ for all ℓ,
  $\mathcal{M}$.addConstr$\left(\sum_{\ell=1}^{w_i} a_i^\ell \geq a_i\right)$
for *each output bit j* **do**
  initialize an empty linear expression L_j
  for *each input bit i with $M_{j,i} = 1$* **do**
    take the next unused copy of a_i, say a_i^ℓ, and set $L_j \leftarrow L_j + a_i^\ell$
  $\mathcal{M}$.addConstr$(b_j = L_j)$

Modular addition modeling Modular addition is the main nonlinear component of ARX designs such as ChaCha, Salsa, Speck and Forró, so its model determines how precise the analysis of these primitives can be. For $z = (x + y) \bmod 2^n$, CLAASP-MP uses the perfect monomial prediction model of Hu and Yap [21], which is exact under 3SDP-woU.

Let a_i, b_i and z_i denote the exponent variables of the two inputs and of the output at bit position i, counted from the least significant bit, and let c_i be one carry variable per position. The propagation is captured by

$$a_i + b_i + c_{i-1} = z_i + 2\,c_i, \qquad 0 \le i \le n-1,$$

with $c_{-1} = 0$, since no carry enters the least significant bit, and $c_{n-1} = 0$, which performs the reduction modulo 2^n. CLAASP-MP adds these constraints bit by bit to $\mathcal{M}$, in the same way as for the Boolean word operations.

Modular multiplication modeling Modular multiplication is rarer in symmetric designs but critical to ciphers such as the recent MSX [25], and in a different form to IDEA [24], which multiplies modulo $2^{16} + 1$. Prior division-property modeling is due to Xing et al. [38], who encode it as an S-box under the two-subset division property; CLAASP-MP instead decomposes it bit by bit into `AND` and modular addition constraints, which needs no truth table and scales to the 32-bit operands used in real designs.

Writing $x = \sum_{i=0}^{n-1} x_i 2^i$ and $y = \sum_{j=0}^{n-1} y_j 2^j$, the n-bit product $z = (x \cdot y) \bmod 2^n$ expands as

$$z = \sum_{j=0}^{n-1} \left(\sum_{i=0}^{n-1} (x_i \cdot y_j) 2^{i+j} \right) \bmod 2^n.$$

Each partial product $x_i y_j$ is an exact bitwise `AND` on exponent variables, and `COPY` constraints handle the repeated use of x_i and y_j. The factor 2^{i+j} shifts row j left by j positions and truncates the bits with $i + j > n - 1$; the shifted rows are accumulated into z, initialized to 0, by the modular addition model of section 9, as detailed in algorithm 4. Every step uses its exact propagation rule, so the composed model contains exactly the monomial trails of the multiplication, and the coefficient of a monomial is recovered from their parity as elsewhere in CLAASP-MP.

3.3 MILP Model of the Full Primitive

Once all components of a primitive have been translated into constraints, $\mathcal{M}$ describes how exponent variables propagate from the cipher inputs to all cipher outputs. The exponent variables attached to each wire of the DAG are reused across components, so $\mathcal{M}$ captures the complete algebraic structure of the primitive. All analyses performed by CLAASP-MP are obtained by adding a small number of task-dependent constraints and objectives to this fixed model. To obtain the ANF of an output bit, CLAASP-MP fixes the corresponding output exponent variable to the value one and enumerates every feasible assignment of the input exponent variables. A monomial appears in the ANF when the number of assignments reaching it is odd, even number cancels.

Degree evaluation is performed in the same manner. After fixing an output exponent bit to one, the model maximizes the sum of the selected input exponent

Algorithm 4: MILP constraints for modular multiplication

Input: MILP model $\mathcal{M}$, inputs $x = (x_i)_{i=0}^{n-1}$ and $y = (y_j)_{j=0}^{n-1}$

Create output word $z = (z_0, \ldots, z_{n-1})$ and initialize to 0
Create products $p_{i,j} = x_i y_j$ for all i, j with $i + j < n$
`// AND on each` $p_{i,j}$`; COPY distributes` x_i `and` y_j
for $i = 0$ **to** $n - 1$ **do**
    for each j with $i + j < n$ **do**
        $\mathcal{M}$.addConstr($x_i \geq p_{i,j}$)
    $\mathcal{M}$.addConstr$\left(\sum_{j:\, i+j<n} p_{i,j} \geq x_i\right)$
for $j = 0$ **to** $n - 1$ **do**
    for each i with $i + j < n$ **do**
        $\mathcal{M}$.addConstr($y_j \geq p_{i,j}$)
    $\mathcal{M}$.addConstr$\left(\sum_{i:\, i+j<n} p_{i,j} \geq y_j\right)$
for $j = 0$ **to** $n - 1$ **do**
    Create shifted word $s = (s_0, \ldots, s_{n-1})$
    for $k = 0$ **to** $n - 1$ **do**
        if $k < j$ **then**
            $\mathcal{M}$.addConstr($s_k = 0$)
        else
            $\mathcal{M}$.addConstr($s_k = p_{k-j,\, j}$)
    $z \leftarrow$ modular_addition($\mathcal{M}, z, s$)

variables (the plaintext, key, or IV variables). This optimum d is an upper bound on the algebraic degree in those variables, and CLAASP-MP can return it directly. CLAASP-MP also computes a tighter bound by parity: it enumerates all optimal trails of degree d and checks their combined parity. The remaining input variables are kept as symbolic coefficients, so the parity is reduced over each monomial. If some monomial of degree d survives with odd parity, the degree is exactly d. Otherwise the whole degree-d layer cancels, and CLAASP-MP reports $d - 1$ as a tight upper bound. A cube is defined by selecting a set of public inputs as cube variables, while all other public inputs are fixed to zero. The model determines whether the cube monomial reaches the chosen output bit. With the non-cube public inputs fixed, the coefficient of the cube monomial is a polynomial in the key variables, which is the superpoly.

4 Experimental Results with CLAASP-MP

We evaluate CLAASP-MP on block ciphers, permutations, and stream ciphers, instantiating the primitives in CLAASP and applying the routines of CLAASP-MP to the common MILP model described in Section 3.3. Usage examples based on SIMON-32, covering ANF extraction, degree computation, cube-based testing, and superpoly computation, are given in Appendix A.

All MILP instances are solved with Gurobi, with its default settings except where stated below. To ensure precision when computing the upper bound on the degree in the cube variables, we enforce a strict optimality tolerance by setting `OptimalityTol = 1e-9`. In the absence of this adjusted tolerance, the solver may occasionally return false optimal solutions due to floating-point inaccuracies.

For tasks that require enumerating all optimal exponent assignments (ANFs, superpolies, key coefficients, and parity-based degrees), CLAASP-MP sets `PoolSearchMode = 2` with `PoolGap = 0` and a large `PoolSolutions` cap (2×10^9), so the solver enumerates the optimal objective solutions. `PoolSearchMode` alone does not guarantee that every trail is returned, so before trusting any parity CLAASP-MP checks the outcome. It requires an `OPTIMAL` status, and it rejects the result if the pool reached the `PoolSolutions` cap or the solver stopped at `SUBOPTIMAL`. A single missing trail can flip a parity, so these checks are needed for correctness. Tasks that only need a single optimal solution, such as plain degree upper bounds, use `PoolSearchMode = 0`, which stops after the first optimal solution and is substantially faster.

Divide-and-conquer strategy for large enumerations. For the high round superpoly computations, the number of feasible monomial trails can be very large, and a single global enumeration can approach the solver's practical limit. To keep this enumeration tractable, CLAASP-MP automates a standard divide-and-conquer strategy [18,15] on top of the global MILP model. The cipher rounds are split at a chosen intermediate round into two parts, each part is encoded as a separate MILP model and solved independently, and the resulting partial trails are then combined to recover the full superpoly, or key coefficient. Because each half has a much smaller trail count, the enumeration can be substantially faster when the global model is large. As an illustrative example, on Speck-32 we computed the superpoly for a cube with inactive bits $\{24, 26\}$: a single global enumeration did not complete within a day, whereas the divide-and-conquer version returned the same superpoly in approximately 13 seconds on the same machine. The strategy does not always help, since the split carries the cost of enumerating the feasible solutions at the intermediate round, and for small instances a direct global enumeration can be preferable.

4.1 Integral Distinguishers on Block Ciphers

We apply CLAASP-MP to search for integral distinguishers across 15 families of block ciphers in CLAASP with different design structures. Table 8 in the Appendix summarizes the distinguishers on block ciphers obtained with CLAASP-MP and compares them with the literature. Our tool also reproduces the results for integral distinguishers and superpolies based on parity counting, for example, by simple counting for SIMON and by using the divide-and-conquer strategy described above for Speck. Some distinguishers obtained by CLAASP-MP that we claimed to match those reported in the literature may appear different at first glance; however, they are in fact

equivalent, up to minor differences in the interpretation of the specifications (e.g., endianness conventions).

The new results are reported in the following tables, where **Dim** is the cube dimension and **Deg.** is the degree upper bound in the cube variables of the output bit, so a bound strictly below the dimension means that the cube monomial is absent and the output bit is balanced.

SAECHAM. SAECHAM [30] is a lightweight ARX block cipher with a 64-bit block, a 128-bit key and 88 rounds, a variant of CHAM-64/128 [23]. Its design paper evaluates only differential and linear cryptanalysis, and to the best of our knowledge no integral analysis has been published. Table 1 reaches round 20.

Cipher	Rds	Cube Bits (active)	Dim	Deg.	Integral Output Bits
SAECHAM	13	{9, 24, 25}	3	2	{31}
SAECHAM	13	{32–47}	16	15	{47}
SAECHAM	14	{0, 16, 17, 24, 40, 41, 51, 63}	8	7	{47}
SAECHAM	14	{32–47}	16	15	{47}
SAECHAM	15	{13, 24, 40, 41, 51, 56, 57, 63}	8	7	{63}
SAECHAM	16	{0–15}	16	15	{15}
SAECHAM	16	{0–16, 23–31, 56}	27	26	{31}
SAECHAM	18	{0–16, 23–33, 40–47, 56}	37	36	{47}
SAECHAM	18	{0–47}	48	47	{47}
SAECHAM	18	{0–62}	63	62	{15, 47, 63}
SAECHAM	19	{0–47, 55–61, 63}	56	55	{63}
SAECHAM	19	{0–62}	63	62	{15, 63}
SAECHAM	20	{0–47, 55–63}	57	56	{15}
SAECHAM	20	{0–62}	63	62	{15}

Table 1. Integral distinguishers for SAECHAM, reaching round 20.

Splight. Splight [39] is a lightweight block cipher with a 64-bit block, a 128-bit key and 32 rounds, built from 4-bit S-boxes and a nibble-wise linear layer acting on two 32-bit branches. Its designers analyze its integral security with the two-subset bit-based division property, reporting a 9-round distinguisher and concluding that no 10-round one exists [39]; CLAASP-MP reaches rounds 10 and 11. Table 2 also records a 6-round distinguisher whose cube is only four bits wide and which leaves eight output bits balanced.

Cipher	Rds	Cube Bits (active)	Dim	Deg.	Integral Output Bits
Splight	6	{60–63}	4	3	{12–15, 28–31}
Splight	6	{44–47}	4	3	{12–15, 28–31}
Splight	7	{28–31, 48–51}	8	7	{12–15, 28–31}
Splight	8	{0–3, 32–39, 44–47, 52–55}	20	19	{12–15, 28–31}
Splight	9	{4–7, 16–23, 28–31, 40–43, 48–63}	36	35	{12}
Splight	**10**	**{8–11, 16–63}**	**52**	**51**	**{12}**
Splight	**11**	**{0–3, 5–63}**	**63**	**62**	**{8–15, 20–23, 28–31}**

Table 2. Integral distinguishers. Rounds 10 and 11 lie beyond the 9 reported in [39].

MSX-128. MSX [25] is a family of lightweight block ciphers for microcontrollers whose round function combines modular additions with a 32-bit multiplication, and whose 128-bit member has four 32-bit branches and 18 rounds. Its analysis uses the modular multiplication model of section 9. Table 3 reaches round 7, where 64 of the 128 output bits are balanced at data complexity 2^{96}.

Cipher	Rds	Cube Bits (active)	Dim	Deg.	Integral Output Bits
MSX-128	6	{0–63}	64	63	{32–63}
MSX-128	7	{0–31, 64–127}	96	95	{32–63, 96–127}

Table 3. Integral distinguishers for MSX-128, reaching round 7.

uKNIT-BC. uKNIT-BC [17] is a lightweight block cipher with a 64-bit block, a 128-bit key and 12 rounds, whose rounds are deliberately not identical. Using the degree routine of Section 3.3, CLAASP-MP bounds the degree of every output bit in the 64 plaintext bits by 62 at round 6, while at round 7 the bound reaches 63, so the degree saturates between rounds 6 and 7 of 12.

4.2 Integral Distinguishers on Permutations

We also search for integral distinguishers on CLAASP permutations.

ChaCha and Salsa. CLAASP-MP reproduces the known 6-round distinguishers. For Salsa, the 6-round distinguisher of [13] used a non-standard column round, applying the four quarter-rounds to $(0,4,8,12)$,$(1,5,9,13)$,$(2,6,10,14)$,$(3,7,11,15)$ instead of Salsa's offset tuples $(0,4,8,12),(5,9,13,1),(10,14,2,6),(15,3,7,11)$; on the standard specification their cube gives only a 5-round distinguisher. Using the same cube patterns, CLAASP-MP reaches up to 6.75 rounds for both ChaCha and Salsa. A round here applies the quarter-round to the four columns (or diagonals), so the permutation has 20 rounds; by 6.75 rounds we mean six full rounds plus three of the four quarter-rounds of the seventh round. Table 4 summarizes the maximum-round (6.75) distinguishers; a detailed list for rounds 6 to 6.75 is in the Appendix (Table 9). The 6.75-round results are on the core ChaCha and Salsa permutations, without the key schedule, and cannot be used for key recovery; they are an algebraic-resistance bound on the permutation.

Cipher	Rds	Inactive Bits ($\bar{I}$)	Integral Output Bits
ChaCha	6.75	{256}, {288}, {320}, {352}	{96–127, 224–255, 352–383, 480–511}
Salsa	6.75	{18}, {45}, {178}, {205}, {365}, {397}	{359}

Table 4. Maximum round (6.75) integral distinguishers for ChaCha and Salsa.

Forró. Forró [8] is a 512-bit keyless permutation of sixteen 32-bit words over 14 rounds, in the same ARX family as ChaCha and Salsa. Its design paper evaluates only linear approximations, differential-linear distinguishers and attacks using probabilistic neutral bits, so to the best of our knowledge Table 5 holds the first integral results. As with ChaCha and Salsa, they are an algebraic-resistance bound rather than an attack: 2^{511} data for 4 of 14 rounds, a margin of one degree. Some words are not updated between rounds 4 and 4.75, so they stay balanced at 4.75 also.

Cipher	Rds	Inactive Bits ($\bar{I}$)	Deg.	Integral Output Bits
Forró	4	{224}, {225}, {320}, {321}, {352}, {353}	510	{0–63, 160–287, 320–447, 480–511}
Forró	4	{192}, {193}	510	{0–63, 160–223, 238–255, 270–287, 320–420, 430–447, 480–511}
Forró	4	{257}	510	{7–31, 160–191, 204–223, 320–351, 364–388, 396–415, 480–511}
Forró	4	{256}	510	{8–31, 160–191, 205–223, 320–351, 365–388, 397–415, 480–511}
Forró	4	{288}, {289}	510	{22–31, 160–191, 219–223, 320–351, 379–388, 411–415, 480–511}
Forró	4	{160}, {161}	510	{29–31, 160–191, 320–351, 386–388, 480–511}
Forró	4	{128}, {129}	510	{187–191, 347–351, 480–484, 507–511}

Table 5. First integral distinguishers for Forró, with 511-dimensional cubes.

Chaskey-Pi. Chaskey-Pi is the 128-bit ARX permutation underlying the Chaskey message authentication code [26], four 32-bit words over 12 rounds. Unlike the primitives above it has already been studied with automated integral tools, so Table 6 reports improvements. CLAASP-MP reproduces the published distinguishers, then lowers the data from 2^{64} to 2^{23} and from 2^{127} to 2^{122} at rounds 3 and 4.

Cipher	Rds	Inactive Bits ($\bar{I}$)	Dim	Deg	Integral Output Bits	Ref.
Chaskey-Pi	3	{64–127}	64	63	{74–79}	[13,21]
Chaskey-Pi	3	{22–52, 54–127}	**23**	**22**	{75–79}	Ours
Chaskey-Pi	3	{27–52, 59–127}	33	32	{74}	Ours
Chaskey-Pi	4	{127}	127	126	{78, 79}	[13,21]
Chaskey-Pi	4	{52}	127	126	{30, 31, 77, 78, 79}	[13]
Chaskey-Pi	4	{111–116}	**122**	**121**	{79}	Ours

Table 6. Integral distinguishers on Chaskey-Pi, published results alongside ours.

4.3 Integral and Algebraic Analysis on Stream Ciphers

We next apply CLAASP-MP to the stream ciphers Bivium and Trivium.

Bivium. For Bivium (708 full initialization clocks), CLAASP-MP identifies several 36-dimensional cubes whose superpolies for the first keystream bit are identically zero, yielding integral distinguishers; for example $C_1 = \{i_{14}-i_{30}, i_{35}, i_{43}, i_{54}-i_{70}\}$ and $C_2 = \{i_{15}-i_{31}, i_{36}, i_{44}, i_{55}-i_{71}\}$. Many cubes of larger dimension (37–54) also exhibit integral behavior, and any cube of dimension at least 55 does so since the degree reaches 54 after full initialization.

Trivium. For Trivium, we extend the algebraic degree analysis of the first keystream bit beyond the range reported in [19]. CLAASP-MP computes a *tight* degree bound using the routine described in Section 3.3. Table 7 reports the bounds obtained beyond the previously analyzed range. We also recomputed

Clocks	**835**	**836**	**837**	**838**
Degree bound	78	79	79	79

Table 7. Tight degree bound of Trivium beyond 834 clocks.

superpolies for cubes from [12] at clocks 672, 735, and 767; all match the published results exactly, confirming correctness.

5 Comparison with Existing Tools

The closest framework to CLAASP-MP is Solvatore (https://github.com/kste/solvatore), a SAT-based tool introduced by Eskandari et al. [13] in 2018, which encodes division-property propagation into a Boolean satisfiability (CNF) instance. A cipher is represented as a sequence of basic bit-level operations (bit-copy, XOR, AND and the constant 1), which are functionally complete, together with high-level methods for S-boxes, bit permutations, modular addition and MixColumn. Each operation has a predefined SAT encoding, and a new primitive is analyzed by composing them in sequence. Compared to CLAASP, Solvatore is implemented entirely in Python and uses very compact cipher descriptions. Since it targets a specific type of integral distinguisher search based on the bit-based division property, it ignores the key schedule and round-key XOR operations, modeling only the evolution of the internal state. By contrast, CLAASP provides full cipher models including key scheduling, since its division-property tools also serve key-dependent applications. At the time of writing, Solvatore features 40 ciphers spanning SPN, ARX and Feistel families.

Solvatore is designed solely for detecting integral distinguishers. Given a candidate cube, it tests whether the division-property vector can reach the target bit. Under the two-subset division property this can only *exclude* the cube monomial: an unreachable (infeasible) target certifies that the monomial is absent, which is an integral distinguisher. A reachable target leaves the monomial potentially present with unknown parity, so Solvatore cannot certify presence.

In contrast, from the same MILP model CLAASP-MP can extract the full ANF of an output bit in the early rounds, compute degree bounds, determine whether a cube monomial reaches the output, and derive superpolies together with their key coefficients, each by adding a small set of query-specific constraints.

Solvatore builds a separate CNF instance for every candidate cube. CLAASP-MP instead builds one global MILP model from the CLAASP component graph, independent of the cube, and testing a cube only adds a few linear constraints to it. This has a direct impact on scalability: while the SAT-based framework reports integral distinguishers for ChaCha and Salsa up to 6 and 5 rounds, respectively, CLAASP-MP extends these to 6.75 rounds.

Beyond Solvatore, several automated tools search the division property through MILP or SAT models. Xiang et al. [37] first phrased this search as a MILP problem for six lightweight block ciphers, Sun et al. [31] extended it to ARX ciphers and to the word-based division property, and Amiry et al. [1] scaled it to large block ciphers. Like Solvatore, these work under the two-subset division property, so they can only exclude monomials, and each is aimed at a particular family or size regime. Wang et al. [36] instead use the three-subset division property, but their models are still written separately for each cipher.

6 Conclusion

We presented CLAASP-MP, a unified MILP-based framework for monomial prediction built on the 3SDP-woU formulation and integrated into the CLAASP library. CLAASP-MP supports ANF extraction, algebraic degree computation, superpoly derivation, key-coefficient recovery, and integral analysis across block ciphers, permutations, and stream ciphers. We reproduced several known results and showed how CLAASP-MP produces new ones. For block ciphers, we identified new integral cubes for several primitives, obtained the first integral distinguishers for SAECHAM, and reached rounds 10 and 11 of Splight, where its designers, using a two-subset division property search, report that no 10-round integral distinguisher exists [39]. For permutations, we extended the published distinguishers for the ChaCha and Salsa permutations, obtained the first integral distinguishers for Forró, and lowered the data complexity of the known Chaskey-Pi distinguishers. For stream ciphers, we extended the tight degree bounds for Trivium to later initialization clocks. The framework also includes a MILP model for modular multiplication, which we illustrate on MSX-128, where a 96-dimensional cube leaves 64 of the 128 output bits balanced at round 7.

Future work includes a more systematic application of CLAASP-MP to existing and new ciphers for tighter bounds and deeper algebraic insights. Another direction is to incorporate recent advances such as nested MP, core MP, and refined propagation for nonlinear components [18,11,16,15], to further improve the computation of large superpolies. Overall, CLAASP-MP provides a practical automated algebraic analysis tool and a basis for more precise and scalable monomial-based cryptanalysis.

References

1. Amiry, A., Nyberg, K., Mahdavifar, E.: Automatic division property tool for large block ciphers. IACR Transactions on Symmetric Cryptology **2022**(4), 150–180 (2022), https://tosc.iacr.org/index.php/ToSC/article/view/8703
2. Bellini, E., Gérault, D., Grados, J., Huang, Y.J., Makarim, R.H., Rachidi, M., Tiwari, S.K.: CLAASP: A Cryptographic Library for the Automated Analysis of Symmetric Primitives. In: Carlet, C., Mandal, K., Rijmen, V. (eds.) Selected Areas in Cryptography - SAC 2023 - 30th International Conference, Fredericton, Canada, August 14-18, 2023, Revised Selected Papers. Lecture Notes in Computer Science, vol. 14201, pp. 387–408. Springer (2023). https://doi.org/10.1007/978-3-031-53368-6_19
3. Bellini, E., Rachidi, M., Rohit, R., Tiwari, S.K.: Mind the composition of Toffoli gates: Structural algebraic distinguishers of ARADI. IACR Cryptol. ePrint Arch. p. 1559 (2024), https://eprint.iacr.org/2024/1559
4. Bellini, E., Rachidi, M., Rohit, R., Tiwari, S.K.: On the structural properties of Toffoli gate composition in ARADI: implications for algebraic distinguishers. In: Fischlin, M., Moonsamy, V. (eds.) Applied Cryptography and Network Security - 23rd International Conference, ACNS 2025, Munich, Germany, June 23-26, 2025, Proceedings, Part II. Lecture Notes in Computer Science, vol. 15826, pp. 400–425. Springer (2025). https://doi.org/10.1007/978-3-031-95764-2_16, https://doi.org/10.1007/978-3-031-95764-2_16
5. Boura, C., Canteaut, A.: On the influence of the algebraic degree of F^{-1} on the algebraic degree of G ∘ F. IEEE Trans. Inf. Theory **59**(1), 691–702 (2013). https://doi.org/10.1109/TIT.2012.2214203, https://doi.org/10.1109/TIT.2012.2214203
6. Boura, C., Canteaut, A.: Another view of the division property. 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. 654–682. Springer (2016). https://doi.org/10.1007/978-3-662-53018-4_24, https://doi.org/10.1007/978-3-662-53018-4_24
7. Brayton, R.K., Hachtel, G.D., McMullen, C.T., Sangiovanni-Vincentelli, A.L.: Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publishers (1984)
8. Coutinho, M., Passos, I., Vásquez, J.C.G., Sarkar, S., de Mendonça, F.L.L., de Sousa Jr., R.T., Borges, F.: Latin dances reloaded: Improved cryptanalysis against Salsa and ChaCha, and the proposal of Forró. Journal of Cryptology **36**, 18 (2023). https://doi.org/10.1007/s00145-023-09455-5
9. Cui, T., Wang, M., Fan, Y., Fu, Y., Huang, L.: Ballet. https://sfjs.cacrnet.org.cn/ (2019), https://sfjs.cacrnet.org.cn/site/term/list_76_1.html
10. Daemen, J., Knudsen, L.R., Rijmen, V.: The block cipher Square. In: Biham, E. (ed.) Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1267, pp. 149–165. Springer (1997). https://doi.org/10.1007/BFB0052343, https://doi.org/10.1007/BFb0052343
11. Derbez, P., Lambin, B.: Fast MILP models for division property. IACR Trans. Symmetric Cryptol. **2022**(2), 289–321 (2022). https://doi.org/10.46586/TOSC.V2022.I2.289-321, https://doi.org/10.46586/tosc.v2022.i2.289-321

12. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Joux, A. (ed.) Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5479, pp. 278–299. Springer (2009). https://doi.org/10.1007/978-3-642-01001-9_16, https://doi.org/10.1007/978-3-642-01001-9_16
13. Eskandari, Z., Kidmose, A.B., Kölbl, S., Tiessen, T.: Finding integral distinguishers with ease. In: Cid, C., Jr., M.J.J. (eds.) Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11349, pp. 115–138. Springer (2018). https://doi.org/10.1007/978-3-030-10970-7_6, https://doi.org/10.1007/978-3-030-10970-7_6
14. Hao, Y., Leander, G., Meier, W., Todo, Y., Wang, Q.: Modeling for three-subset division property without unknown subset - improved cube attacks against Trivium and Grain-128AEAD. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12105, pp. 466–495. Springer (2020). https://doi.org/10.1007/978-3-030-45721-1_17, https://doi.org/10.1007/978-3-030-45721-1_17
15. He, J., Hu, K., Lei, H., Wang, M.: Massive superpoly recovery with a meet-in-the-middle framework - improved cube attacks on Trivium and Kreyvium. In: Joye, M., Leander, G. (eds.) Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14651, pp. 368–397. Springer (2024). https://doi.org/10.1007/978-3-031-58716-0_13, https://doi.org/10.1007/978-3-031-58716-0_13
16. He, J., Hu, K., Preneel, B., Wang, M.: Stretching cube attacks: Improved methods to recover massive superpolies. 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 IV. Lecture Notes in Computer Science, vol. 13794, pp. 537–566. Springer (2022). https://doi.org/10.1007/978-3-031-22972-5_19, https://doi.org/10.1007/978-3-031-22972-5_19
17. Hu, K., Khairallah, M., Peyrin, T., Tan, Q.Q.: uKNIT: Breaking round-alignment for cipher design. IACR Transactions on Symmetric Cryptology (2026), featuring uKNIT-BC, an Ultra Low-Latency Block Cipher
18. Hu, K., Sun, S., Todo, Y., Wang, M., Wang, Q.: Massive superpoly recovery with nested monomial predictions. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13090, pp. 392–421. Springer (2021). https://doi.org/10.1007/978-3-030-92062-3_14, https://doi.org/10.1007/978-3-030-92062-3_14
19. Hu, K., Sun, S., Wang, M., Wang, Q.: An algebraic formulation of the division property: Revisiting degree evaluations, cube attacks, and key-independent sums. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings,

Part I. Lecture Notes in Computer Science, vol. 12491, pp. 446–476. Springer (2020). https://doi.org/10.1007/978-3-030-64837-4_15, https://doi.org/10.1007/978-3-030-64837-4_15

20. Hu, K., Wang, M.: Automatic search for a variant of division property using three subsets. In: Matsui, M. (ed.) Topics in Cryptology - CT-RSA 2019 - The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4-8, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11405, pp. 412–432. Springer (2019). https://doi.org/10.1007/978-3-030-12612-4_21, https://doi.org/10.1007/978-3-030-12612-4_21
21. Hu, K., Yap, T.: Perfect monomial prediction for modular addition. IACR Trans. Symmetric Cryptol. **2024**(3), 177–199 (2024). https://doi.org/10.46586/TOSC.V2024.I3.177-199, https://doi.org/10.46586/tosc.v2024.i3.177-199
22. Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings. Lecture Notes in Computer Science, vol. 1008, pp. 196–211. Springer (1994). https://doi.org/10.1007/3-540-60590-8_16, https://doi.org/10.1007/3-540-60590-8_16
23. Koo, B., Roh, D., Kim, H., Jung, Y., Lee, D., Kwon, D.: CHAM: A family of lightweight block ciphers for resource-constrained devices. In: Information Security and Cryptology – ICISC. pp. 3–25. Lecture Notes in Computer Science, Springer (2018). https://doi.org/10.1007/978-3-319-78556-1_1
24. Lai, X., Massey, J.L.: A proposal for a new block encryption standard. In: Damgård, I.B. (ed.) Advances in Cryptology — EUROCRYPT '90. pp. 389–404. Springer Berlin Heidelberg, Berlin, Heidelberg (1991)
25. Minematsu, K., Suzaki, T., Rahman, M., Suryawanshi, S., Isobe, T.: MSX: Lightweight block ciphers for microcontrollers with high-assurance against differential and linear attacks. IACR Communications in Cryptology **2**(4) (2026). https://doi.org/10.62056/av11c3c2h
26. Mouha, N., Mennink, B., Herrewege, A.V., Watanabe, D., Preneel, B., Verbauwhede, I.: Chaskey: An efficient MAC algorithm for 32-bit microcontrollers. In: Selected Areas in Cryptography – SAC 2014. pp. 306–323. Lecture Notes in Computer Science, Springer (2014). https://doi.org/10.1007/978-3-319-13051-4_19
27. Scala, R.L., Pintore, F., Tiwari, S.K., Visconti, A.: A multistep strategy for polynomial system solving over finite fields and a new algebraic attack on the stream cipher Trivium. Finite Fields Their Appl. **98**, 102452 (2024). https://doi.org/10.1016/J.FFA.2024.102452, https://doi.org/10.1016/j.ffa.2024.102452
28. Scala, R.L., Polese, S., Tiwari, S.K., Visconti, A.: An algebraic attack to the bluetooth stream cipher E0. Finite Fields Their Appl. **84**, 102102 (2022). https://doi.org/10.1016/J.FFA.2022.102102, https://doi.org/10.1016/j.ffa.2022.102102
29. Scala, R.L., Tiwari, S.K.: Stream/block ciphers, difference equations and algebraic attacks. J. Symb. Comput. **109**, 177–198 (2022). https://doi.org/10.1016/J.JSC.2021.09.001, https://doi.org/10.1016/j.jsc.2021.09.001
30. Shin, M., Shin, H., Kim, I., Kim, S., Lee, D., Hong, D., Sung, J., Hong, S.: SAECHAM: Secure and efficient lightweight block cipher CHAM variant. IEEE Internet of Things Journal **12**(15) (2025). https://doi.org/10.1109/JIOT.2025.3569746

31. Sun, L., Wang, W., Wang, M.: Automatic search of bit-based division property for ARX ciphers and word-based division property. In: Advances in Cryptology – ASIACRYPT 2017. Lecture Notes in Computer Science, vol. 10624, pp. 128–157. Springer (2017). https://doi.org/10.1007/978-3-319-70694-8_5
32. Todo, Y.: Integral cryptanalysis on full MISTY1. In: Gennaro, R., Robshaw, M. (eds.) Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 413–432. Springer (2015). https://doi.org/10.1007/978-3-662-47989-6_20, https://doi.org/10.1007/978-3-662-47989-6_20
33. Todo, Y.: Structural Evaluation by Generalized Integral Property. 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. 287–314. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_12
34. Todo, Y., Isobe, T., Hao, Y., Meier, W.: Cube attacks on non-blackbox polynomials based on division property. IEEE Trans. Computers **67**(12), 1720–1736 (2018). https://doi.org/10.1109/TC.2018.2835480, https://doi.org/10.1109/TC.2018.2835480
35. Todo, Y., Morii, M.: Bit-Based Division Property and Application to Simon Family. 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. 357–377. Springer (2016). https://doi.org/10.1007/978-3-662-52993-5_18
36. Wang, S., Hu, B., Guan, J., Zhang, K., Shi, T.: Milp-aided method of searching division property using three subsets and applications. 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. 398–427. Springer (2019). https://doi.org/10.1007/978-3-030-34618-8_14, https://doi.org/10.1007/978-3-030-34618-8_14
37. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. 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. 648–678 (2016). https://doi.org/10.1007/978-3-662-53887-6_24, https://doi.org/10.1007/978-3-662-53887-6_24
38. Xing, Y., Yan, H., Lai, X.: New observation on division property: Simplifying models of basic operations and modeling modular multiplication operation. In: Emrouznejad, A., Qian, Z. (eds.) The 2nd International Conference on Computer Science and Application Engineering, CSAE 2018, Hohhot, China, October 22-24, 2018. pp. 8:1–8:6. ACM (2018). https://doi.org/10.1145/3207677.3278003, https://doi.org/10.1145/3207677.3278003
39. Zhou, Q., Ye, Q., Lai, C., Yu, Y., Zhang, M., Liu, X.: Splight: A lightweight block cipher for resource-constrained embedded devices. IEEE Transactions

on Computers **75**(7), 2618–2631 (2026). https://doi.org/10.1109/TC.2026.3689069

Supplementary Material

A Practical Usage Examples for CLAASP-MP

The examples below illustrate the main routines of CLAASP-MP on the SIMON-32 block cipher as implemented in CLAASP.

Available Functionalities: Given any primitive implemented in CLAASP, the monomial prediction model supports the following operations:

- Compute the ANF of a given output bit.
- Determine the algebraic degree (upper bound or tight bound) of a given output bit.
- Compute the superpoly of a given output bit under a specified cube, and its tight degree bound.
- Compute an upper bound on the algebraic degree of a cube monomial of a given output bit, and the corresponding key coefficient.

Example Script (SIMON-32)

```
from claasp.ciphers.block_ciphers.simon_block_cipher import SimonBlockCipher
from claasp.cipher_modules.models.milp.milp_models.Gurobi.monomial_prediction \
    import MilpMonomialPredictionModel

# Initialize the cipher and the MILP model:
cipher = SimonBlockCipher(number_of_rounds=3)
milp = MilpMonomialPredictionModel(cipher)

# Compute ANF of the output bit at position 0:
milp.find_anf_of_specific_output_bit(0)
Output = p1*p2*p3*p10*p11 + p1*p2*p3*p10 + ... + k52

# Compute upper bound and tight (parity based) degree of output bits
milp.find_upper_bound_degree_of_specific_output_bit(0)
Output = 5
milp.find_upper_bound_degree_of_all_output_bits()
Output = [5, 5, ..., 5, 5, 3, 3, ... , 3, 3]
milp.find_tight_upper_bound_degree_via_parity_of_specific_output_bit(0)
Output = 5
milp.find_tight_upper_bound_degree_via_parity_of_all_output_bits()
Output = [5, 5, ... , 5, 5, 3, 3, ... , 3, 3]

# Define a cube and compute the superpoly of the output bit at position 0
cube = ["p16"]
milp.find_superpoly_of_specific_output_bit(0, cube)
Output = k33*k57 + k50*k57 + k51*k57 + 1
milp.find_tight_upper_bound_degree_via_parity_of_superpoly_of_specific_output_bit(0, cube)
Output = 2
milp.find_tight_upper_bound_degree_via_parity_of_superpoly_of_all_output_bits(cube)
Output = [2, -1, -1, -1, -1, -1, 2, 2, -1, ... , 0, 1]
-1 marks a bit whose superpoly is identically zero, bit is balanced over the cube

# Compute the upper bound on the degree of a cube monomial
milp.find_upper_bound_degree_of_cube_monomial_of_specific_output_bit(0, cube)
Output = 1
```

Listing 1.1. Example usage of the monomial prediction model on SIMON-32

B Extended Integral Distinguisher Tables

This appendix provides the comprehensive tables of integral distinguishers obtained by CLAASP-MP across various primitives. Table 8 details the distinguishers identified for block ciphers comparing with established literature results. Table 9 provides the distinguishers found for ChaCha and Salsa permutations, including new higher round distinguishers.

Cipher	**Rds**	**Inactive Bits ($\overline{I}$)**	**Integral Output Bits**	**Ref.**
Aradi	8	$\{31, 63, 95, 127\}$	{32–63, 96–127}	[3,4]
Ballet	6	{32}	{127}	[9]
HIGHT	18	{63}	{39}	[13]
LBlock	17	{30}	{62,63,34,35}	[13]
LEA	8	{31}	{32–36,55–63,66,118}	[13]
PRESENT	9	{60,61,62,63}	{63}	[37]
PRESENT	9	{63}	{3,7,11-15,19,23,27-31,35,39, 43-47, 51,55,59,60-63}	[13]
SIMON32	13	{0}	{16–31}	[37]
SIMON48	15	{0}	{24–47}	[37]
SIMON64	17	{0}	{32–42,48,54–63}	[37]
SIMON96	21	{0}	{48,50,55,89,94}	[37]
SIMON128	25	{0}	{64,66,126}	[37]
SIMECK32	14	{0}	{16,17,21,22,26,27,31}	[37]
SIMECK48	17	{0}	{24,28,29,43,44}	[37]
SIMECK64	20	{0}	{32,33,37,59,63}	[37]
SPECK32	5	{25,26}	{8-15,25-31}	[13]
SPECK32	6	{26}	{15}	[13]
PrinceV2	7	{63}	{0}	Ours
Twine	16	{59}	{4}	Ours

Table 8. Integral distinguishers on block ciphers.

Cipher	Rds	Inactive Bits ($\overline{I}$)	Integral Output Bits	Ref.
ChaCha	6	{0}	{32–68, 192–223, 352–415, 424–428}	[13]
ChaCha	6	{32}, {160}	{64–100, 224–287, 416–447, 456–460}	Ours
		{64}, {192}	{0–4, 96–159, 288–319, 448–492}	Ours
		{96}, {224}	{0–36, 160–191, 320–396, 480–511}	Ours
		{128}	{32–68, 192–223, 352–415, 424–428}	Ours
		{129}, {289}	{64–66, 424–426}	Ours
		{161}, {321}	{96–98, 456–458}	Ours
		{193}, {353}	{0–2, 488–490}	Ours
		{225}, {257}	{32–34, 392–394}	Ours
		{256}, {288}, {320}, {352}	{0–511}	Ours
ChaCha	6.5	{0}, {128}	{64–68, 192–223, 352–383}	Ours
		{32}	{64–100, 224–255, 456–460}	Ours
		{64}, {192}	{96–127, 448–479, 488–492}	Ours
		{96}	{320–351, 480–511}	Ours
		{129}, {289}	{64–66}	Ours
		{160}	{64–100, 224–255, 456–460}	Ours
		{161}, {321}	{96–98, 456–458}	Ours
		{193}, {353}	{488–490}	Ours
		{224}	{320–351, 480–511}	Ours
		{256}, {288}, {320}, {352}	{64–127, 192–255, 320–383, 448–511}	Ours
ChaCha	6.75	{0}, {128}	{352–383}	Ours
		{32}	{96–100, 224–252}	Ours
		{64}, {192}	{96–127, 488–492}	Ours
		{96}, {224}	{480–511}	Ours
		{160}	{96–100, 224–255}	Ours
		{161}, {321}	{96–98}	Ours
		{193}, {353}	{488–490}	Ours
		{256}, {288}, {320}, {352}	{96–127, 224–255, 352–383, 480–511}	Ours
Salsa	5	{0–31}	{73–74, 96–127, 135, 233, 295–304, 448–479}	[13]
Salsa	6	{18}	{199, 359}	Ours
		{45}, {205}, {365}, {397}	{39, 199, 359, 391}	Ours
		{178}	{359, 391}	Ours
		{338}	{39, 391}	Ours
		{498}	{39, 199}	Ours
Salsa	6.5	{18}, {45}, {205}, {365}, {397}	{199, 359}	Ours
		{178}	{359}	Ours
		{498}	{199}	Ours
Salsa	6.75	{18}, {45}, {178}, {205}, {365}, {397}	{359}	Ours

Table 9. State-of-the-art integral distinguishers for ChaCha and Salsa underlying permutations.

C Solvatore vs CLAASP-MP

Table 10 summarizes the main differences between Solvatore [13] and our tool.

Table 10. Comparison between Solvatore and CLAASP-MP

Feature	**Solvatore**	**CLAASP-MP**
Modeling approach	SAT, 2SDP [35] (2016)	MILP, 3SDP-woU [14] (2020)
Available ciphers	40	more than 100
Language	Python	Sagemath/Python
Cipher API	**Low-level**: Bit-Copy, AND, XOR, Constant bit 1. **High-level**: S-box, Bit Permutation, MixColumn, ShuffleWords, ModAdd, ModAdd with Constant	S-box, FSR, Constant, Linear Layer, MixColumn, XOR, AND, NOT, OR, ModAdd, ModMul, Rotate, Shift
Cipher Representation	Does not require a key schedule	Needs full cipher implementation
Supported Cryptanalysis Technique	Integral Distinguisher	ANF, Algebraic Degree, Superpoly, Integral Distinguisher,
Integral Distinguisher Approach	Separate CNF instance for every candidate cube	Global MILP model, independent of the chosen cube

D Implementation Examples in CLAASP

This appendix complements Section 3.1 by illustrating CLAASP primitives and their component-based structure used for MILP generation.

D.1 Component Representation: S-box Example

The listing below shows a 4-bit S-box's internal dictionary, with a unique ID, type, bit sizes, predecessor links, and operation description.

```
# round = 0 - round component = 4
'id': 'sbox_0_4',
'type': 'sbox',
'input_bit_size': 4,
'input_id_link': ['xor_0_1'],
'input_bit_positions': [[0, 1, 2, 3]],
'output_bit_size': 4,
'description': [12, 10, 13, 3, 14, 11, 15, 7, 8, 9, 1, 5, 0, 2, 4, 6],
```

By traversing the DAG whose nodes are dictionaries like this one, and applying the rules of Section 2.4, CLAASP-MP generates the MILP constraints, here from the S-box ANF.

D.2 Illustration: A Small ToySPN Primitive

Figure 1 and Figure 2 show the CLAASP code and structure of a two-round 6-bit SPN with two 3-bit S-boxes per round, XOR key injection, and 1-bit left rotation.

```
from claasp.cipher import Cipher

class ToySPN(Cipher):
  def __init__(self):
    super().__init__(family_name="toyspn",
      cipher_type="block_cipher",
      cipher_inputs=["plaintext", "key"],
      cipher_inputs_bit_size=[6, 6],
      cipher_output_bit_size=6)

    sbox = [0, 5, 3, 2, 6, 1, 4, 7]
    self.add_round()
    xor = self.add_XOR_component(
        ["plaintext", "key"],
        [[0,1,2,3,4,5],[0,1,2,3,4,5]],
        6
    )
    sbox1 = self.add_SBOX_component(
        [xor.id],
        [[0, 1, 2]],
        3,
        sbox
    )
    sbox2 = self.add_SBOX_component(
        [xor.id],
        [[3, 4, 5]],
        3,
        sbox
    )
    rotate = self.add_rotate_component(
        [sbox1.id, sbox2.id],
        [[0,1,2],[0,1,2]],
        6,
        1
    )
    self.add_round_output_component(
        [rotate.id],
        [[0,1,2,3,4,5]],
        6
    )
    self.add_round()
    xor = self.add_XOR_component(
        [rotate.id, "key"],
        [[0,1,2,3,4,5],[0,1,2,3,4,5]],
        6
    )
    sbox1 = self.add_SBOX_component(
        [xor.id],
        [[0, 1, 2]],
        3,
        sbox
    )
    sbox2 = self.add_SBOX_component(
        [xor.id],
        [[3, 4, 5]],
        3,
        sbox
    )
    rotate = self.add_rotate_component(
        [sbox1.id, sbox2.id],
        [[0,1,2],[0,1,2]],
        6,
        1
    )
    self.add_cipher_output_component(
        [rotate.id],
        [[0,1,2,3,4,5]],
        6
    )
toyspn = ToySPN()
hex(toyspn.evaluate([0x3F,0x3F]))
```

Fig. 1. ToySPN class definition in CLAASP.

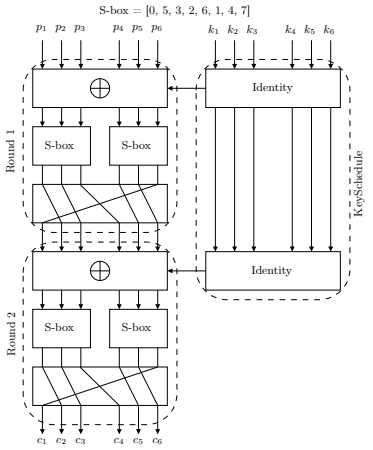

Fig. 2. ToySPN structure and implementation.