IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2022, No. 2, pp. 289–321. DOI:10.46586/tosc.v2022.i2.289-321

Fast MILP Models for Division Property

Patrick Derbez[1*] and Baptiste Lambin[2†]

[1] Univ Rennes, Centre National de la Recherche Scientifique (CNRS), Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), Rennes, France
patrick.derbez@irisa.fr
[2] Ruhr University Bochum, Bochum, Germany
baptiste.lambin@uni.lu

Abstract. Nowadays, MILP is a very popular tool to help cryptographers search for various distinguishers, in particular for integral distinguishers based on the division property. However, cryptographers tend to use MILP in a rather naive way, modeling problems in an exact manner and feeding them to a MILP solver. In this paper, we show that a proper use of some features of MILP solvers such as lazy constraints, along with using simpler but less accurate base models, can achieve much better solving times, while maintaining the precision of exact models. In particular, we describe several new modelization techniques for division property related models as well as a new variant of the Quine-McCluskey algorithm for this specific setting. Moreover, we positively answer a problem raised in [DF20] about handling the large sets of constraints describing valid transitions through Super S-boxes into a MILP model. As a result, we greatly improve the solving times to recover the distinguishers from several previous works ([DF20], [HWW20], [SWW17], [Udo21], [EY21]) and we were able to search for integral distinguishers on 5-round `ARIA` which was out of reach of previous modeling techniques.

Keywords: block cipher · integral distinguisher · MILP

1 Introduction

Nowadays, modeling an attack against a symmetric primitive as a Mixed Integer Linear Programming (MILP) problem and then solving it by some MILP solver is a popular method. It was applied for the first time by Mouha *et al.* [MWGP11] and by Wu and Wang [WW11] for studying the resistance of several ciphers (e.g. Enocoro) against both differential and linear cryptanalysis. Since then, the use of MILP models in our community has increased, the advantage being that many cryptanalytic problems are relatively easy to translate into linear constraints (typically on bits) and available solvers, in particular Gurobi [Gur21], are most often very efficient to solve them. Furthermore such generic solvers are used by a large number of people and companies, ensuring a decent level of confidence in their results. Currently, MILP solvers are mainly used for differential cryptanalysis [MWGP11], including the search of sophisticated boomerang distinguishers [LS19, DDV20], and for integral cryptanalysis by exhausting division trails on a cipher [XZBL16].

While generating a MILP model for a given problem is most often relatively easy, generating a *fast enough* model is harder. This is mainly because there are no tight bounds regarding the complexity of solving a system of n inequalities involving m variables as

*Patrick Derbez was supported by the French Agence Nationale de la Recherche through the CryptAudit project under Contract ANR-17-CE39-0003.
†Baptiste Lambin was supported by the project Analysis and Protection of Lightweight Cryptographic Algorithms (432878529).

Licensed under Creative Commons License CC-BY 4.0.
Received: 2021-11-23 Revised: 2022-03-01 Accepted: 2022-05-01 Published: 2022-06-10

it depends on too many parameters, including the heuristic used for the exploration of the underlying search tree. Still, it is commonly assumed that lowering the number of inequalities as well as the number of variables involved in the model does speed-up the solving process [ST17, BC20]. As an example, in [AST+17] Abdelkhalek *et al.* used the Quine-McCluskey (QM) algorithm [Qui52, Qui55, McC56] to derive a minimum set of linear inequalities on boolean variables to represent the DDT of an 8-bit S-box. Doing so they were able to prove that no differential characteristic against 13-round `Skinny-128` has probability greater than 2^{-128} while it would have been impossible to get this result in a reasonable time with a naive modelization. However, Sasaki and Todo showed that the most compact modelization is not always the fastest one to solve [ST17].

1.1 Division Property

In this paper we focus on modelization techniques for division property. For this cryptanalysis technique, the goal is to answer the question of the existence of a division trail through the cipher for particular inputs and outputs as the absence of trails implies the existence of an integral distinguisher. Typically the cipher is decomposed into several simple functions for which we know the valid transitions and a trail is said to be valid if and only if all the internal transitions are valid. The main issue with this technique is that the presence of a trail does not ensure the non-existence of integral distinguishers. Indeed, the validity of a trail does depend on the internal decomposition of the cipher, and a valid trail for one decomposition might be invalid for another one. In this case we say that the trail is a *false positive*. This is because the core of division property is to track the monomials through an iterated function and, to simplify, does not properly handle the XOR operation. For instance, let p and q be two polynomials and a binary variable x such that $p = xp_1 \oplus p_2$ and $q = xq_1 \oplus q_2$. Then both p and q depend on x but it is unclear for $p \oplus q$. To keep the process simple, in the 2-subset division property we do assume that $p \oplus q$ also depends on x but this obviously might be incorrect. Thus over the years, researchers tried to provide more accurate modelization techniques to avoid as much as possible false-positive trails. In [XZBL16], Xiang *et al.* showed how to use the convex hull to provide an exact representation of valid transitions through an S-box (the method takes a practical time up to a 6-bit S-box). Later in [ZR19], Zhang and Rijmen gave an exact modelization for any linear layer which is practical for binary matrices on a field extension. Then Hu *et al.* showed in [HWW20] that quadratic constraints could be used to modelize the propagation through any linear layer and thus switched to SAT/SMT solvers that seem better-suited for this kind of constraints. Derbez and Fouque described in [DF20] a new algorithm to compute the propagation table of any 16-bit Super S-box and in [Udo21] Udovenko described such big tables with hundreds of thousands logical constraints that he successfully solved with a SAT-solver.

We observe that last works related to 2-subset division property all abandon MILP solvers for either ad-hoc algorithms or SAT/SMT solvers. In this paper we show that MILP solvers, and more precisely Gurobi, are still very competitive and that they can actually solve faster all the problems from those previous works. Our main idea is that it is unnecessary to deal with large and complicated models since most of the constraints are not required to conclude on the existence of a division trail.

1.2 Exact Modelization vs. Lossy Modelization

Modelizing propagation rules through a cipher may require a large number of inequalities but not all of them are required to conclude on the existence of a division trail from a specific value u to a specific v. For instance it was shown in [XZBL16] that one may allow only *minimal* valid transitions through the different layers composing the cipher without affecting the result of the model. More generally, we can add constraints to the model or

remove some of them as long as it does not remove all valid division trails nor add any new one.

Thus in this work we propose to go further with the concept of *lossy modelization* to modelize possible transitions $u \xrightarrow{f} v$ through a function f. The main idea is to relax some of the constraints to simplify the model. This means that we do not remove any valid propagation but may allow invalid ones. So far this technique was only and unwillingly used for functions too complex to be modelized in an exact manner, typically using a Copy-Xor-And decomposition. But as shown by ElSheikh and Youssef, lossy modelizations may sometimes lead to faster models than exact modelizations [EY21]. Indeed, if the resulting model does not have any solution for a given input/output pair, we are still sure that there would not be any solution for the corresponding exact model, and so all distinguishers found using lossy modelizations are valid. On the other hand, a solution to the model might be a false-positive resulting from the lossy modelization. Thus, for each function for which we have access to a more accurate modelization than the one used in the model, we do need to check whether the corresponding constraints hold or not for the actual solution in order to conserve the same level of precision. To handle this, the most efficient way is to exploit specific functionalities of the Gurobi solver named *callbacks* and *lazy constraints*. A callback is a user-designed function that is called by the solver whenever some predetermined events happen during the solving process (e.g. a new solution is found), without fully interrupting it. This is very useful as it allows the user to have more control over the solving process. During a callback it is possible to add extra constraints to the model, called *lazy constraints*, mainly because of the way they are processed by the solver.[1]

1.3 Our Contribution

In this paper we thus propose several new modelization techniques to search for division trails on block ciphers, focusing on 2-subset (conventional) division property. In particular we provide the following results:

1. We describe a new algorithm which greatly improves the computation time for the Quine-McCluskey algorithm in the specific case of division property. We also propose a greedy approach to compact inequalities output by the Quine-McCluskey algorithm when inserted into a MILP model. Our approach is very fast and, as an example, was able to compute the Quine-McCluskey inequalities for a Super S-box involved in `LED` and to reduce the number of inequalities from 388134 to 108668 in few seconds.

2. We provide several new (lossy) modelization techniques including a new modelization of S-boxes based on piecewise linear constraints (PWLC).

3. We also show that the $\mathcal{ZR}$-technique [ZR19] for linear layers can be computed locally on the fly to discard false-positive trails. In particular, this helps modeling linear layers in an exact manner through callbacks and leads to models which are much faster to be solved than with the simple technique of ElSheikh and Youssef.

4. Finally, we were able to retrieve with MILP models all the previous results which required SAT/SMT solvers [HWW20, Udo21] or ad-hoc algorithms [DF20]. Furthermore, our models are consistently solved faster than with previous approaches. We also solved the open problem of modeling Super S-boxes in MILP, as they would require a very large amount of constraints. Contrary to Udovenko who solved it as a SAT problem with millions of constraints, we show that by the use of lazy constraints, this is easily handled by MILP. Surprisingly, our results suggest that

[1] Lazy constraints are slightly different than regular constraints, we refer the reader to the Gurobi documentation for more details

handling division property through Super S-boxes is not as promising as expected in [DF20] and seems to be only interesting to generate the input/output pairs to try for.

Some of the lossy modelizations we propose are quite close to the original word-based division property [Tod15], which are then refined and made more precise using lazy constraints, leading to bit-based division property [TM16]. Thus, one can see similarities with the typical approach used to search for differential characteristics (e.g. [GLMS20]) since the search is performed in two steps : a first model to search for best truncated differentials characteristics and a second model to find the best instantiations of them. However in our case everything is handled by the same model thanks to Gurobi's callback functionality, thus not having to solve 2 different models in a row.

In Table 1, we give an overview of the best times achieved to search for integral distinguishers over various block ciphers. We make the distinction between *conventional distinguishers* (Conv. Dist.) which are the usual distinguishers, looking for balanced bits resulting from a set of plaintexts with constant bits, and *extended distinguishers* (Ext. Dist.), which are the distinguishers searched in [LDF20, DF20], where one is looking for balanced *linear combinations* of bits coming from a set of plaintexts with constant *linear combinations* of bits. Note that currently the linear combinations are computationally limited to the size of an S-box or Super S-box (depending on the cases), which is given in the "Word Size" column of Table 1 for extended distinguishers. More details are given in Section 5 as well as in the Appendix.

Table 1: Comparison of our solving times with previous works.
* Time required to find the relevant trails, see Section B.5 for more details.
† Our implementation using [EY21] simple lazy constraints.
‡ With exact modeling for the linear layer, and all possible starting round for `Camellia`.

Cipher	Rounds	Type of Result	Word Size	Our Time	Previous Time
`AES`	4	Conv. Dist.	-	0.4s	-
	5	No Ext. Dist.	8-bit	13min	31min [EY21]†
`ARIA`	4	Conv. Dist.	-	0.8s	-
	5	No Ext. Dist.	8-bit	5h	$\geq$ 24h [EY21]†
`CRAFT`	13	Conv. Dist.	-	3.6s	-
	14	No Ext. Dist.	16-bit	11min	-
`HIGHT`	20	Ext. Dist.	16-bit	12min	13 days [DF20]
	21	No Ext. Dist.	16-bit	14min	-
`LED`	8	No Ext. Dist.	16-bit	3h*	16h [Udo21]
`Skinny64`	11	Ext. Dist.	16-bit	9min	22min [DF20]
	12	No Ext. Dist.	16-bit	80s	4min [DF20]
`Camellia`	7	Conv. Dist.	-	30s	99min [HWW20]
`Camellia`	8	No Conv. Dist.‡	-	18h	-
`CLEFIA`	10	Conv. Dist.	-	23min	82min [HWW20]
`CLEFIA`	11	No Conv. Dist.‡	-	4h	-
`LEA`	8	Conv. Dist.	-	20s	30min [SWW17]

Regarding `ARIA`, the previously best known integral distinguisher was given over 4 rounds with 24 balanced bits. We show that we can improve this to 128 balanced bits, and thanks to our new modeling strategies, we can also efficiently search for extended

distinguisher over 5 rounds, showing that there are no such distinguishers. Note that in the case of `ARIA`, using the simple lazy constraints from [EY21] was not enough as getting a result for a single input/output pair had to be interrupted after more than 3 hours of computations for this pair, while with our new techniques, all computations are done within about 5 hours.

Finally, all of our timings are given as real time, using a standard laptop with a 4-core/8-thread CPU[2], without any parallelization outside of the native multithreading from Gurobi. As such, when comparing to timings given in [DF20] where they used a 128-core server, one should remember that we could improve our timing further by conducting the search in parallel, allocating 8 threads for each instance of Gurobi, which would be very close to dividing our solving times by n if using $8n$ threads in the cases considered. Our code is available at

https://github.com/FastMILPDivisionProperty/FastMILPDivision

2 Division Property

In this section, we give the notations and definitions we use in this paper. We also succinctly introduce division property based distinguishers on block ciphers and refer interested readers to [TM16, XZBL16] for more details.

2.1 Notations and Definitions

We denote by $\mathbf{x} = (x_0, \dots, x_{n-1}) \in \mathbb{F}_2^n$ an n-bit vector, where x_0 is the least significant bit and will often write $x_0 x_1 \dots x_{n-1}$ instead of $(x_0, \dots, x_{n-1})$. There is a trivial mapping from n-bit vectors to monomials in variables $(X_0, \dots, X_{n-1})$ and we will often refer to $\mathbf{x}$ as a monomial.

Definition 1. We say that a monomial $\mathbf{m_0}$ contains a monomial $\mathbf{m_1}$ if and only if all the variables of $\mathbf{m_1}$ belong to $\mathbf{m_0}$, i.e. if and only if $\mathbf{m_1}$ is a divisor of $\mathbf{m_0}$. In that case we will write $\mathbf{m_1} \preceq \mathbf{m_0}$. For instance, $x_0 \preceq x_0 x_1$ but $x_2 \not\preceq x_0 x_1$.

Definition 2 (Bit-product)**.** For $\mathbf{x}, \mathbf{u} \in \mathbb{F}_2^n$, we denote by $\mathbf{x^u}$ the bit product

$$\mathbf{x^u} = \prod_{i=0}^{n-1} x_i^{u_i}.$$

Definition 3 (Bit-based Division Property [TM16])**.** A set $\mathbb{X} \subset \mathbb{F}_2^n$ has the division property $D_{\mathbb{K}}^n$, where $\mathbb{K} \subset \mathbb{F}_2^n$ is a set, if for all $\mathbf{u} \in \mathbb{F}_2^n$, we have

$$\bigoplus_{\mathbf{x} \in \mathbb{X}} \mathbf{x^u} = \begin{cases} \mathit{unknown} & \text{if there is } \mathbf{k} \in \mathbb{K} \text{ s.t. } \mathbf{u} \succeq \mathbf{k} \\ 0 & \text{otherwise} \end{cases}$$

2.2 Integral Distinguishers

Given a block cipher, let $P_b(X_0, \dots X_{n-1}, K_0, \dots, K_{m-1})$ be the polynomial describing the b-th bit of the ciphertext as a function of the plaintext (X) and the master key (K). If no monomial greater than or equal to $X_0 X_1 \dots X_{i-1}$ appears in P_b then for any value $\mathbf{y}$ of $(X_i, \dots, X_{n-1}, K_0, \dots, K_{m-1})$ we have that

$$\bigoplus_{\mathbf{x} \in \{0,1\}^i} P_b(\mathbf{x}, \mathbf{y}) = 0,$$

[2]Intel(R) Core(TM) i7-8665U CPU @ 1.90GHz

which is a property a random function should not have. However, in practice we cannot computationally obtain the polynomial expression of all the bits of the ciphertext because the number of terms is too large. This is where division property is useful. Let f and g be two n-bit functions and let $y_i = f_i(x_0, \ldots, x_{n-1})$ and $z_i = g_i(y_0, \ldots, y_{n-1}) = g_i \circ f(x_0, \ldots, x_{n-1})$ be the intermediate and final expressions of the coordinate functions of f and g respectively. Division property captures that if for all monomials $\mathbf{y}^{\mathbf{v}}$ appearing in $\mathbf{z}^{\mathbf{u}}$, $\mathbf{y}^{\mathbf{v}}$ does not involve a monomial greater than $\mathbf{x}^{\mathbf{w}}$ then $\mathbf{z}^{\mathbf{u}}$ (now seen as a function of the x_i's) does not either. Hence, a common way to search for integral distinguishers is to study the *division trails* of this cipher, which show the propagation of the division property through the basic operations composing the block cipher.

Definition 4 (Division Trails [XZBL16])**.** Let f denote the round function of an iterated block cipher. Assume the input set to the block cipher has initial division property $D^n_{\{\mathbf{k}\}}$, and denote the division property after propagating through i rounds of the block cipher (i.e. i applications of f) by $D^n_{\mathbb{K}_i}$. Thus, we have the following chain of division property propagations :

$$\{\mathbf{k}\} \triangleq \mathbb{K}_0 \xrightarrow{f} \mathbb{K}_1 \xrightarrow{f} \mathbb{K}_2 \xrightarrow{f} \ldots \xrightarrow{f} \mathbb{K}_r.$$

Moreover, for any vector $\mathbf{k}_i$ in $\mathbb{K}_i$ ($i \geq 1$), there must exist a vector $\mathbf{k}_{i-1}$ in $\mathbb{K}_{i-1}$ such that $\mathbf{k}_{i-1}$ can propagate to $\mathbf{k}_i$ by the division property propagation rules, i.e. $f^{\mathbf{k}_i}$ contains a monomial $\mathbf{m}$ such that $\mathbf{m} \succeq \mathbf{k}_{i-1}$. Furthermore, for $(\mathbf{k}_0, \mathbf{k}_1, \ldots, \mathbf{k}_r) \in \mathbb{K}_0 \times \mathbb{K}_1 \times \cdots \times \mathbb{K}_r$, if $\mathbf{k}_{i-1}$ can propagate to $\mathbf{k}_i$ for all $i \in \{1, 2, \ldots, r\}$, $(\mathbf{k}_0, \mathbf{k}_1, \ldots, \mathbf{k}_r)$ is called an r-round division trail.

In the rest of the paper, we will denote $\mathbf{k} \xrightarrow{f} \mathbf{k}'$ if the vector $\mathbf{k} \in \mathbb{F}_2^n$ can propagate to a vector $\mathbf{k}' \in \mathbb{F}_2^m$ through the n-bit to m-bit function f.

2.3 Tools

Searching for division trails through a cipher is an interesting problem and many tools and models have been developed to tackle it.

- **Ad-hoc algorithms.** The first ad-hoc algorithms dedicated to division property were proposed by Todo and Morii in [TM16] but were limited to 32-bit block ciphers. More recently in [DF20], Derbez and Fouque proposed a simple branch-and-bound algorithm, which can handle a large amount of constraints[3] and in particular the propagation tables of 16-bit Super S-boxes.

- **MILP models.** Xiang et al. [XZBL16] were the first to show how to use MILP models to search for division trails. Sasaki and Todo [ST17] proposed a way to minimize (also using MILP) the number of inequalities required to modelize S-boxes. Later on, MILP was also used to search for extended distinguishers [LDF20] taking into account the addition of an extra linear mapping at both the input and output of the cipher. Finally, ElSheikh and Youssef recently proposed to combine lossy modelization and lazy constraints for linear layers [EY21].

- **SAT/SMT models.** In [SWW17], Sun *et al.* described the first SAT and SMT models to search for division trails. The main argument to use such models instead of MILP ones is that all propagation rules can be described as logical equations. Following this work, Eskandari *et al.* developed Solvatore, a SAT-based tool they used to find integral distinguishers against 30 primitives [EKKT18]. Later, Hu *et al.* [HWW20] went further by giving an exact modelization of the propagation through a linear layer using only a quadratic constraint. Finally, very recently Udovenko

[3] While technically not using constraints as their approach is based on propagation tables, these tables could be easily be translated by constraints.

showed how to modelize the propagation table of 16-bit functions using hundreds of thousands logical constraints that he succesfully solved using a SAT solver [Udo21].

3 Previous Modelization Techniques

In this section we recall the main modelization techniques dedicated to cipher components and used for MILP models.

3.1 Modelizing Basic Binary Operations

We first begin by the binary operations AND, XOR and Copy which are extensively used to describe division trails. Indeed, any cipher can be decomposed using only those 3 operations which are thus of fundamental importance. Linear constraints for those operations have been originally proposed by Xiang *et al.* [XZBL16] and later generalized and improved by Sun *et al.* [SWW20].

Operation	XOR	$COPY$	AND
Trail	$(a_1, a_2, \ldots, a_m) \to b$	$a \to (b_1, b_2, \ldots, b_m)$	$(a_1, a_2, \ldots, a_m) \to b$
Constraints	$a_1 + \ldots + a_m = b$	$b_1 + \ldots + b_m = a$	$a_1 + \ldots + a_m \geq b$ $a_1 + \ldots + a_m \leq mb$

Note that *constant* a_i's can be removed from the inequalities (corresponding to key variables in the cipher) related to both the XOR and AND operations.

3.2 Modelizing S-boxes

In the following we are considering an S-box $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$, and aim at modelizing the propagations $u \xrightarrow{S} v$ for $u, v \in \mathbb{F}_2^n$. A naive approach would be to modelize an S-box by simply using its computational circuit (essentially its ANF), modelizing each basic operation (XOR, AND, etc.) at the bit level. However this is largely inefficient, both in term of number of constraints/variables and overall precision. Hence, it is better to first construct a table containing the valid propagations, and then to modelize this table as a whole. This means that we generate the set

$$\mathbb{S} = \{(u_1, \ldots, u_n, v_1, \ldots, v_n) \in \mathbb{F}_2^{2n} \text{ s.t. } u \xrightarrow{S} v\},$$

possibly removing redundant propagations, and aim at finding a set of linear inequalities modelizing this set.

3.2.1 Exact Modelization : Convex Hull

The first technique used to modelize the propagation through an S-box is using the *Convex Hull* of $\mathbb{S}$, the smallest convex set containing all points of $\mathbb{S}$. It can be represented by its H-representation, a set of linear inequalities $\mathcal{L}$ such that the only feasible solutions of $\mathcal{L}$ are exactly the points of $\mathbb{S}$. This H-representation is typically computed using the SageMath software [The20], which in backend is using the Parma Polyhedra Library [BHZ08]. Such approach was first proposed in the context of differential cryptanalysis [SHW+14], and later used to actually search for division trails [XZBL16]. Note that the number of inequalities in the H-representation can be rather large. But hopefully, since in our case all coefficients u_i and v_i belong to $\{0, 1\}$, one can significantly reduce the size of $\mathcal{L}$ while still maintaining its correctness [SHW+14] (i.e. the feasible solutions of the reduced set are still exactly the points of $\mathbb{S}$).

Unfortunately this technique is essentially limited to S-boxes of size $n \leq 6$ since for larger S-boxes the H-representation becomes quite hard to compute. For instance, it takes few days to compute the inequalities for `AES` S-Box.

3.2.2 Exact Modelization : Quine-McCluskey Algorithm

In [AST+17], Abdelkhalek *et al.* proposed to use the Quine-McCluskey (QM) algorithm to reduce the number of inequalities required to describe the *difference distribution table* of an S-box. This approach was efficient enough so they were able to modelize the 8-bit S-box of `SKINNY`-128 with few hundreds inequalities only while its DDT contains several thousands of entries. Such approach was also used in the context of division property as for instance to modelize division trails through the `AES` S-box to show lower bounds on the degree [HLLT20].

The main idea used in the QM algorithm is to search for cosets of bit-aligned vector spaces of impossible values. For instance, assume the following values for (x, y, z, t) are impossible:

$$(0,0,1,1) \quad (0,1,1,1) \quad (1,0,1,1) \quad (0,0,0,1) \quad (0,0,1,0) \quad (0,0,0,0)$$

Discarding those 6 values from a MILP model can be done with the 6 inequalities:

$$\begin{aligned}
&x + y + (1-z) + (1-t) \geq 1\\
&x + (1-y) + (1-z) + (1-t) \geq 1\\
&(1-x) + y + (1-z) + (1-t) \geq 1\\
&x + y + z + (1-t) \geq 1\\
&x + y + (1-z) + t \geq 1\\
&x + y + z + t \geq 1
\end{aligned}$$

In order to decrease the number of inequalities, the QM algorithm aims at identifying pairs of impossible values that differ in only one bit. For instance $\{(0,0,1,1),(0,1,1,1)\}$ is such a pair and leads to the new impossible value $(0,\star,1,1)$. Once all pairs have been proceeded, the goal is to find a *minimal* cover of the impossible values. For our example it could be:

$$(0,0,\star,\star) \quad (\star,0,1,1) \quad (0,\star,1,1)$$

However, this algorithm is inherently exponential in the number of variables and most of the public implementations handle systems containing no more than 16 variables. For more complex systems it is recommended to use an heuristic version of the QM algorithm (e.g. Espresso). It is also worth mentioning here that in [BC20], Boura and Coggia proposed some alternative methods regarding this problem which most often lead to less inequalities than Espresso.

3.2.3 Lossy Modelization: Modular Addition

Describing propagation of division property through the addition modulo 2^n can be done directly using the convex hull technique or Quine-McCluskey algorithm. But both the time complexity and the number of inequalities increase so fast that it should be reserved for very small values of n (i.e. $n \approx 4$). For higher values, the best known technique is to use a lossy modelization based on a Copy-Xor-And decomposition of the modular addition. For instance in [SWW17], Sun *et al.* proposed a decomposition using $3n-1$ **xor**, $3n-1$ **copy** and $2n-1$ **and** operations, leading to $10n-4$ linear constraints and $12n-19$ intermediate variables.

Note that, as for the **and** operation, if one of the two branches is constant (typically in the case of a modular addition between a state variable and a key variable), then the linear constraints required to describe valid transitions can be simplified [SWLW16].

In [BBdS+20], Beierle *et al.* proposed a much simpler modelization of the modular addition requiring only 2 inequalities per bit. Consider an addition of two n-bit words $a, b \in 0, 1^n$ and let $y = a + b \mod 2^n$. This operation can be computed from successive applications of the function $f(a_i, b_i, c_i) = (\text{Maj}(a_i, b_i, c_i), a_i \oplus b_i \oplus c_i)$ where Maj is the majority function and c_i the i-th carry, using that $(c_{i+1}, y_i) = f(a_i, b_i, c_i)$. The division table of the function f can be modelized with the two following inequalities:

$$\begin{cases} -a_i - b_i - c_i + 2c_{i+1} + y_i & \geq & 0 \\ a_i + b_i + c_i - 2c_{i+1} - 2y_i & \geq & -1 \end{cases}.$$

3.3 Modelizing Linear Mappings

In this section we will consider linear mappings represented by a matrix $M \in \mathbb{F}_{2^n}^{m \times m}$. Note that by choosing a basis of $\mathbb{F}_{2^n}$ one can derive an equivalent matrix working at bit-level $M_b \in \mathbb{F}_2^{nm \times nm}$. Depending on the size of M and on which coefficients appear in it, several modelizations exist in the literature.

3.3.1 Exact Modelization : As an S-box

An obvious way to modelize such a linear mapping is simply to ignore its linear nature and see it as an S-box. While most often the size of linear mappings forbids the use of the exact modelizations described in the previous section, there are still cases where considering it as an S-box can be useful. The first one is when nm is small enough to be handled by either the convex hull method or the QM algorithm. This is for example the case in the `HIGHT` block cipher, for which the round function involves two linear mappings from $\mathbb{F}_2^{8 \times 8}$ that can be handled by the QM algorithm. Another notable case is when $M \in \mathbb{F}_{2^n}^{m \times m}$ is a binary matrix as in the block cipher `CRAFT`. In this case, the linear layer can actually be seen as the application of n (linear) m-bit S-boxes in parallel. Indeed, let us see this through an example with the MixColumns matrix M from `CRAFT`, for which it is easy to derive an equivalent matrix M_b over $\mathbb{F}_2$ with

$$M = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \in \mathbb{F}_{2^4}^{4 \times 4}, \quad M_b = \begin{pmatrix} I_4 & 0 & I_4 & I_4 \\ 0 & I_4 & 0 & I_4 \\ 0 & 0 & I_4 & 0 \\ 0 & 0 & 0 & I_4 \end{pmatrix} \in \mathbb{F}_2^{16 \times 16},$$

where I_4 is the identity matrix over $\mathbb{F}_2$ of dimension 4. Thus when applying this matrix to a vector $x = (x_1, \ldots, x_{16}) \in \mathbb{F}_2^{16}$, one can observe that it is equivalent to performing the following matrix multiplications in parallel:

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_i \\ x_{i+4} \\ x_{i+8} \\ x_{i+12} \end{pmatrix}, i \in \{1, 2, 3, 4\}.$$

In other words, we apply the same (linear) 4-bit S-box in parallel over x (up to reordering the variables), where the S-box has ANF:

$$S(x_1, x_2, x_3, x_4) = (x_1 + x_3 + x_4, x_2 + x_4, x_3, x_4).$$

3.3.2 Lossy Modelization : The Copy-XOR Technique

The Copy-XOR technique is the first one that was proposed to modelize division property over linear layers [XZBL16]. The idea is very simple. To modelize the propagations $u \xrightarrow{M} v$ with corresponding variables $\mathbf{u}, \mathbf{v}$, one introduces temporary variables $\mathbf{t}_{i,j}$ for each non-zero coefficient at index (i, j) of the matrix M (over $\mathbb{F}_2$). Then the propagation is modelized with the constraints:

$$\begin{aligned} (\mathbf{t}_{i,j})_{i \in C_j} &= \mathrm{COPY}(\mathbf{u}_j), & 1 \leq j \leq nm, \\ \mathbf{v}_i &= \mathrm{XOR}(\mathbf{t}_{i,j})_{j \in R_i}, & 1 \leq i \leq nm, \end{aligned}$$

where $C_j = \{i \text{ s.t. } M_{i,j} = 1\}$ and $R_i = \{j \text{ s.t. } M_{i,j} = 1\}$.

Zhang and Rijmen showed in [ZR19] that this modelization is actually lossy as it may introduce some false-positive trails. Furthermore it relies on adding a possibly large amount of additional variables to the model, namely as many as the number of non-zero coefficients in the matrix over $\mathbb{F}_2$, and thus is hardly scalable.

3.3.3 Exact Modelization : The $\mathcal{ZR}$ Technique

To overcome weaknesses of the Copy-XOR modelization of linear layers, Zhang and Rijmen proposed a new method relying on the following theorem:

Theorem 1. *Let $M \in \mathbb{F}_2^{n \times n}$ be an invertible matrix and $u, v \in \mathbb{F}_2^n$. Let $I_u = \{i \text{ s.t. } u_i = 1\}$ and $I_v = \{i \text{ s.t. } v_i = 1\}$ and assume $|I_u| = |I_v|$. Then $u \xrightarrow{M} v$ is a valid propagation if and only if the minor of M built with the columns with indexes in I_u and rows with indexes in I_v is invertible.*

Note that, because each of its components is of degree 1, a linear mapping M cannot decrease the weight of division property vectors: if we have a valid propagation $u \to v$, we necessarily have $\mathrm{wt}(u) \leq \mathrm{wt}(v)$. Furthermore, if $u \to v$ is a valid transition through M then there exists $v' \preceq v$ such that $\mathrm{wt}(v') = \mathrm{wt}(u)$ and $u \to v'$ is a valid transition as well.

From this theorem and careful observations, they thus give a way to generate a set of inequalities that exactly modelize the valid propagations through a linear mapping. The main idea of Zhang and Rijmen is to compute all possible linear combinations of rows and for each of them they add a constraint to ensure that if the rows are selected then at least one column with a non-zero coefficient (regarding the linear combination) should be also selected. As a consequence, for an $n \times n$ binary matrix over $\mathbb{F}_2$, $2^n - 1$ inequalities are required, one for each non-zero linear combination of rows.

Note that if $M \in \mathbb{F}_{2^n}^{m \times m}$ is a binary matrix (as for instance in `SKINNY` and `Midori`) the number of inequalities can be lowered to $n(2^m - 1)$ inequalities since the application of M can be seen as the parallel application of n binary matrices of size $m \times m$ (see Section 3.3.1).

3.4 Lazy Constraints

As mentioned in Section 1.2, using a lossy modelization might lead to false-positive trails that can be removed with lazy constraints. One obvious way to get those extra constraints is simply to use the constraints generated from an exact modelization technique described in this section. Typically, when a trail is found by the MILP solver, all its transitions are checked against a list of constraints. If one constraint is violated, it is added to the model as a *lazy constraint*, discarding the current solution and each one that would involve the same invalid transition. The main advantage of lazy constraints is to add into the model only a fraction of the constraints that would be required to obtain an exact model. The intuition is that given a trail $u_0 \xrightarrow{f_0} u_1 \xrightarrow{f_1} \dots \xrightarrow{f_{r-1}} u_r$ for which both u_0 and u_r are set to

specific values (which is exactly the framework of division property), the possible values for the u_i's are restricted and thus it is not necessary to fully describe all the valid transitions through each layer.

In [EY21], ElSheikh and Youssef proposed to combine the Copy-XOR modelization for linear layers with lazy constraints to ensure the validity of a division trail. Given a MILP solution to a model, for each transition $u \xrightarrow{M} v$ through a linear layer, they check whether the minor $M_{v,u}$ is invertible or not. If it is not invertible then they propose to add a lazy constraint to remove this transition. However this is a rather inefficient way of handling false-positive for the linear layer, and we show in Section 4.5 how to better improve this. Moreover, they used this idea for the linear layers only, while we show in next sections that using it for both S-boxes and Super S-boxes can be very efficient as well.

4 New Modelization Techniques

In this section we describe our new proposal to modelize division trails for MILP solvers such as Gurobi.

4.1 An Important Property

Regarding 2-subset division property, a transition $u \xrightarrow{f} v$ through a function f is valid if and only if x^u divides at least one monomial of $f(x)^v$. In particular, if transition $u \xrightarrow{f} v$ is valid then for all $u' \preceq u$, transition $u' \xrightarrow{f} v$ is also valid. A direct consequence for the search of division trails through a cipher is that for all $v' \succ v$, the transition $u \xrightarrow{f} v'$ can be safely added to or removed from the model. In other words, allowing or not the transition $u \xrightarrow{f} v'$ does not affect the result of the model [XZBL16]. This property was originally used to reduce the propagation tables and to focus on *minimal* valid transitions. But we found that, in several cases, it might be faster to allow non-minimal transitions, even though they are invalid. A simple example is related to the modelization of division property through the **and** operator. If we allow the transition $(0,\ldots,0) \xrightarrow{AND} (1)$ (which is possible since $(0,\ldots,0) \xrightarrow{AND} (0)$ is valid) then the operator can be modelized using only one inequality instead of two:

$$a_1 + \ldots + a_m \leq mb$$

4.2 Exact Modelization : Quine-McCluskey Algorithm

The property given Section 4.1 is very powerful when combined with QM. Let $\mathcal{S}$ be the set of all transitions $u' \xrightarrow{f} v'$ for which there exists a valid transition $u \xrightarrow{f} v$ such that $u \succeq u'$ and $v \preceq v'$. The only transitions that must be removed from the model are thus the ones that do not belong to $\mathcal{S}$. Let denote by $\mathcal{I}$ the set of such transitions. Given $u \xrightarrow{f} v \in \mathcal{I}$, any transition $u' \xrightarrow{f} v'$ such that $u' \succeq u$ and $v' \preceq v$ is also impossible because otherwise $u \xrightarrow{f} v$ would belong to $\mathcal{S}$. In other words, given $a = (a_u|a_v)$ describing the transition $u \xrightarrow{f} v$, we know that all 0's of a_u as well as all 1's of a_v can be replaced by a $\star$. Furthermore if two such vectors differ at only one position then one of the vectors has a $\star$ at that position and thus covers the second vector. As a consequence there is no need to perform the *saturation* phase of QM algorithm but only to extract *maximal* vectors. Let $\mathcal{I}_{\max}$ be the set of maximal impossible vectors. This set of vectors covers all impossible transitions and is actually minimal, meaning that removing any element from it would lead to allowing an impossible transition. Indeed, let $a \in \mathcal{I}_{\max}$ and replace all $\star$'s of a_u and a_v by 0 and 1 respectively. This vector corresponds to an impossible transition and

should be removed by the constraints. But all vectors removing it must cover a and since a is maximal it must belong to the set of impossible vectors.

Thus, given the propagation table of an n-bit to m-bit function f, we can construct a minimal set of impossible transitions using Algorithm 1. The complexity of this algorithm is upper bounded by $(n+m)2^{n+m}$ and in practice takes only few seconds for the values of $n+m$ considered in this paper.

Algorithm 1: Computing a minimal set of vectors describing all impossible transitions

Data: T, the propagation table associated to an n-bit to m-bit function f
Result: $\mathcal{I}$, a minimal set of vectors describing all impossible transitions
foreach $(u,v) \in \{0,1\}^{n+m}$ **do** $H[u|v] \leftarrow 0$
foreach $u \to v \in T$ **do** `// construct` $\mathcal{S}$
  $S_1 \leftarrow u|v$
  while $S_1 \neq \emptyset$ **do**
    $u'|v' \leftarrow$ an element of S_1 such that $|v'| - |u'|$ is minimal
    remove $u'|v'$ from S_1
    if $H[u'|v'] = 0$ **then**
      $H[u'|v'] \leftarrow 1$
      foreach $u'' \preceq u'$, $|u''| = |u'| - 1$ **do** $S_1 \leftarrow u''|v'$
      foreach $v'' \succeq v'$, $|v''| = |v'| + 1$ **do** $S_1 \leftarrow u'|v''$
    end
  end
end
$\mathcal{I} \leftarrow \emptyset$
foreach $(u,v) \in \{0,1\}^{n+m}$ **do** `// construct` $\mathcal{I}_{\max}$
  if $H[u|v] = 0$ **then**
    $flag \leftarrow 1$
    foreach $u' \preceq u$, $|u'| = |u| - 1$ **do** $flag \leftarrow flag \;\&\; H[u'|v]$
    foreach $v' \succeq v$, $|v'| = |v| + 1$ **do** $flag \leftarrow flag \;\&\; H[u|v']$
    if $flag = 1$ **then** $\mathcal{I} \leftarrow u|v$
  end
end
return $\mathcal{I}$

Comparison with Convex Hull. While the convex hull method is currently limited to 5 or 6-bit S-boxes, we were able to generate the inequalities for much larger S-boxes, including for the 16-bit Super S-boxes described in [DF20]. However, there is somehow a limitation to our QM algorithm. The inequalities resulting from the convex hull computation may allow only the minimal valid transitions (and more generally any particular sets of transitions) but our method necessarily allows all valid transitions.

Decreasing number of inequalities. Because MILP models can include inequalities with non-binary coefficients, it is possible to decrease the number of inequalities obtained by a direct application of the Quine-McCluskey algorithm. Indeed, all the inequalities generated from the algorithm have the form:

$$\sum_a a + \sum_b (1-b) \geq 1,$$

where all involved variables are binary ones. In particular, they can be rewritten as $A + b \geq 1$ where A is an expression taking positive integer values and b is a binary variable

(or 1 minus a binary variable). Such inequalities can be compacted since a collection of k inequalities of the form $A + b_i \geq 1$ for $i \in \{1, \ldots, k\}$ can be replaced by the inequality:

$$kA + b_1 + b_2 + \ldots + b_k \geq k.$$

We emphasize that this new inequality is fully equivalent to the system of k inequalities $\{A + b_1 \geq 1, \ldots, A + b_k \geq 1\}$, meaning that solutions are exactly the same in both cases.[4] Indeed, if $A \geq 1$ then all inequalities are trivially satisfied and if $A = 0$ then all inequalities are satisfied if and only if $b_1 = b_2 = \ldots = b_k = 1$. Finally, if $A = 0$ and for instance $b_1 = 0$ then both the inequalities $kA + b_1 + b_2 + \ldots + b_k \geq k$ and $A + b_1 \geq k$ do not hold.

There are many ways to *compact* inequalities and finding the best one is a hard problem. This is because there are several possibilities for writing an inequality into the required form, one for each involved variable. However we would like to solve this problem on large instances, such as the system of 387070 inequalities describing the propagation table of the 16-bit Super S-box of LED. Thus we propose a very fast greedy approach giving good enough results for our needs. Let $\mathcal{I}$ be the set of all inequalities outputted by QM algorithm. Then, and until $\mathcal{I}$ is empty, we apply the following procedure:

1. For each inequality from $\mathcal{I}$, we generate all the possible A as described above.
2. If for one inequality, all the A's but one appear in no other inequality, we pick the remaining A. Otherwise we pick the A occurring in the most inequalities.
3. We generate the corresponding MILP inequality.
4. We remove from $\mathcal{I}$ all the inequalities covered by the inequality just created.

The results of our new approach are given in Table 2. We were able to compute sets of inequalities for very large S-boxes with the Quine-McCluskey algorithm and then to reduce them for MILP models. All those results were obtained in few seconds on a laptop while omitting the time required to compute the corresponding propagation tables. An example of the process is described in Appendix A.

4.3 Lossy Modelization : Piecewise Linear Constraints

Piecewise linear functions are functions defined such that there exists a set of intervals where the function restricted to each interval is an affine function. Such a function is defined by a list of points defining the start and end of each segment. We give an example of such a function in dimension 2 in Figure 1, where the list of points is

$$L = [(0, 0), (1, 1), (2, 1), (3, 2), (4, 4)].$$

These functions can be modelized in MILP so that a pair of variables is constrained to take the values of a piecewise linear function. A nice thing about the Gurobi solver is that it has a direct way to add such a constraints (called *PWL constraints*) without having to generate the corresponding set of linear inequalities (e.g. using `addGenConstrPWL` in the Python interface). We will use such a constraint to get a lossy modelization of S-boxes as follows.

Given the list of all valid propagations $u \xrightarrow{S} v$, we first compute

$$w_i = \min_{\mathrm{wt}(u)=i} (\mathrm{wt}(v) \text{ s.t. } u \xrightarrow{S} v), 0 \leq i \leq n$$

that is, w_i is the minimum weight of any v such that $u \xrightarrow{S} v$ where the hamming weight of u is i. Our goal is to add a constraint so that if $\mathrm{wt}(u) = i$, then $\mathrm{wt}(v) \geq w_i$. While

[4] But high values of k might lead to numerical issues in Gurobi.

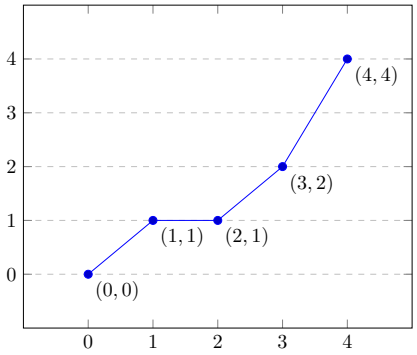

Figure 1: Piecewise linear function defined by L.

this does not look very restrictive, it was enough to prove some results (see Section 5). Actually, it is very close to asking Gurobi to first search for word-based division property distinguishers before searching for bit-based division property. We can easily do so by using the following constraints

$$\begin{aligned}
\mathbf{s}_1 &= \sum_{i=1}^{n} \mathbf{u}_i \\
\mathbf{s}_2 &\leq \sum_{i=1}^{n} \mathbf{v}_i \\
\texttt{addGenConstrPWL}(\mathbf{s}_1, \mathbf{s}_2, [0, 1, \ldots, n], [w_0, w_1, \ldots, w_n]) & \\
\mathbf{u}_i, \mathbf{v}_i \text{ binary variables }, 1 \leq i \leq n & \\
0 \leq \mathbf{s}_1 \leq n & \\
0 \leq \mathbf{s}_2 \leq n &
\end{aligned}$$

where $\texttt{addGenConstrPWL}(\mathbf{x}, \mathbf{y}, T_x, T_y)$ defines a PWL constraints where the piecewise linear function $f(\mathbf{x}) = \mathbf{y}$ is defined by the pairs $(T_x[i], T_y[i])$.

A particular case. For most of the S-boxes used in common block ciphers, the piecewise linear constraint does only ensure the following properties on transitions $u \xrightarrow{S} v$ through a n-bit S-box:

$$\mathrm{wt}(v) = 0 \implies \mathrm{wt}(u) = 0 \text{ and } \mathrm{wt}(u) = n \implies \mathrm{wt}(v) = n.$$

For this particular case, using a piecewise constraint is unnecessary and it is more efficient to only add the two following inequalities to the model:

$$u_0 + \ldots + u_{n-1} \leq (n-1)(v_0 + \ldots + v_{n-1}) \text{ and } n - v_0 - \ldots - v_{n-1} \leq (n-1)(n - u_0 - \ldots - u_{n-1}).$$

4.4 Lossy Modelization : Weight Equality

As mentioned in Section 3.3.3, for any linear mapping M from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ and a valid propagation $u \xrightarrow{M} v$, we necessarily have $\mathrm{wt}(u) = \mathrm{wt}(v)$. This directly provides a very simple lossy modelization for linear mappings :

$$\sum_{1 \leq i \leq n} \mathbf{u}_i = \sum_{1 \leq i \leq m} \mathbf{v}_i.$$

While very simple, this constraint is sometimes sufficient to prove the existence of an integral distinguisher (i.e. to prove that there are no valid division trails through a cipher).

To estimate the complexity of such approach we need to evaluate the probability that a transition $u \xrightarrow{M} v$ is valid knowing that $\mathrm{wt}(u) = \mathrm{wt}(v)$. Because it is valid if and only if the corresponding minor is invertible, the question becomes what is the probability that a minor of a matrix is invertible. For a random binary matrix, the probability goes from 50% for a matrix of size 1 and decreases to $\approx 30\%$ when the size increases. But in practice matrices involved in block ciphers are far from random ones and this drastically affects the probabilities. For instance, sampling 1000 minors of each size for the `AES` MixColumns matrix we obtained the following results:

1 : 15.9%	5 : 1.4%	9 : 1.9%	13 : 3.5%	17 : 5.1%	21 : 12.3%	25 : 18.6%	29 : 26.3%
2 : 6.4%	6 : 0.9%	10 : 1.3%	14 : 3.6%	18 : 6.0%	22 : 13.0%	26 : 18.7%	30 : 31.5%
3 : 2.9%	7 : 0.6%	11 : 1.6%	15 : 3.4%	19 : 7.5%	23 : 15.3%	27 : 20.5%	31 : 44.5%
4 : 1.9%	8 : 1.8%	12 : 1.9%	16 : 3.8%	20 : 11.4%	24 : 17.5%	28 : 22.5%	32 : 100%

As we can observe, those results are very different from what we would expect for random matrices. In particular it seems unlikely to get anything but a false-positive using only the weight equality.

4.5 Lossy Modelization : The local $\mathcal{ZR}$ Technique

Since most of the probabilities given above are very low, it would be too slow to rely on lazy constraints that only discard one transition at a time as it was done in [EY21]. Thus we propose to perform a local $\mathcal{ZR}$ in order to generate more efficient lazy constraints. More precisely, if we have to test a transition $u \xrightarrow{M} v$ through a linear layer M, we can apply the $\mathcal{ZR}$ technique only locally. The idea is to extract the corresponding minor and to check whether it is invertible or not. If not, there is at least one linear combination of the rows equals to the null vector. Going back to the full matrix, we can compute the same linear combination of rows and look at the columns with non-zero coefficients. Then we add the constraint describing that if the rows involved in the linear combination are all selected then at least one of the columns with a non-zero coefficient should be selected as well. In other words, we compute and add constraints from the $\mathcal{ZR}$ technique if and only if they are required. This is very powerful for SPN block ciphers because in that case u (resp. v) is the concatenation of several outputs (resp. inputs) of m-bit S-boxes and such outputs (resp. inputs) have almost always a weight of 0, 1 or m (resp. 0, $m-1$ or m).

Improving performance. Because generating on the fly the constraints from the $\mathcal{ZR}$ technique mainly makes sense for *large* matrices, the cost of checking the rank of minors has to be taken into account. For instance on `AES`, we may have to compute rank of minors up to dimension 31, leading to approximately $31^3 \approx 2^{14.9}$ extra computations per trail found by Gurobi. Hopefully, based on a special property about the inverse of a block matrix, it is possible to reduce the complexity from k^3 (for a minor of size k) to $\min(k, n-k)^3$ for a matrix of size n. This is based on a particular case of the Nullity theorem:

Theorem 2. *An invertible matrix M and its inverse M^{-1} can be partitioned into 2×2 block matrices as*

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \text{ and } M^{-1} = \begin{bmatrix} E & F \\ G & H \end{bmatrix},$$

where A, D, E and H are square matrices such that A and E (resp. D and H) have the same size. The dimension of the kernel of A is equal to the dimension of the kernel of H. In particular, A is invertible if and only if H is invertible as well.

This means that the transition $u \to v$ through M is valid if and only if the transition $\bar{v} \to \bar{u}$ through M^{-1} is valid as well.

4.6 Lossy Modelization : Modular Addition

The modelization of Beierle *et al.* only requires two inequalities per bit:

$$\begin{cases} -a_i - b_i - c_i + 2c_{i+1} + y_i & \geq & 0 \\ a_i + b_i + c_i - 2c_{i+1} - 2y_i & \geq & -1 \end{cases}.$$

The second inequality is used to remove the transitions $(a, b, c) \to (1, 1)$ for $(a, b, c) \neq (1, 1, 1)$ as well as both the transitions $(0, 0, 0) \to (1, 0)$ and $(0, 0, 0) \to (0, 1)$. All of them are non-minimal transitions which can be safely added to the model according to Section 4.1. As a consequence this inequality can be removed from the model without affecting its validity.

5 Results

In this section we discuss the results we obtained using our new modelization techniques. As explained in the introduction, we make a distinction between *conventional* distinguishers and *extended* distinguishers, and we only consider distinguishers that are not key-dependent. Extended distinguishers are the ones considered by [LDF20, DF20] by adding a linear mapping at the input and/or output of the cipher, and built by fixing a *linear combination* of bits to be constant at the input while checking if some *linear combination* of bits at the output are balanced. This makes extended distinguishers cover a larger scope but harder to search, as, for a given input, conventional distinguishers only need to check whether there is a division trail for n vectors of weight 1 (n being the block size), whereas extended distinguishers need to do so for many input/output pairs.

An important trick to search for extended distinguishers is to use some memoization on whether or not a given input/output pair leads to a division trail. Indeed, considering that there is usually a large amount of sets, it is very likely that several pairs will appear in multiple sets, meaning that a naive algorithm would do a large number of redundant computations. Instead, we keep a hash table mapping input/output pairs to whether or not it leads to a division trail. By doing so, whenever we start to examine a new set, we first check if any pair in the set is already present in the hash table. If even a single pair is present in the table *and* leads to a division trail, then there is no need to check any other pair in this specific set as it cannot lead to a distinguisher.

5.1 Notations for Modeling Strategies

In the following, we compare how different modeling strategies affect the solving time for several ciphers. To be more concise, we use several abbreviations for each modeling strategy.

- CH will denote using the Convex Hull modeling from Section 3.2.1,
- QM will denote using the Quine-McCluskey algorithm from Section 4.2,
- PWL will denote using piecewise linear constraints from Section 4.3,
- ZR will denote using the modeling from Zhang and Rijmen from Section 3.3.3,
- CX will denote using the Copy-XOR modeling from Section 3.3.2,
- WE will denote using the Weight Equality modeling from Section 4.4.

As most ciphers we considered have both an S-box and a linear layer, we commonly use an expression of the form $M + M'$ to denote the use of a modeling M for the S-box layer and a modeling M' for the linear layer, where M and M' are taken from the list above. For example, QM+CX denotes using the Quine-McCluskey algorithm for the S-box layer and the Copy-XOR modeling for the linear layer. Note that when using a lossy modelization (which are PWL, CX and WE), we will add lazy constraints to the model so that we obtain a correct result, as described in Section 3.4.

5.2 Number of Inequalities

Using the approach described in Section 4.2, we were able to provide relatively small sets of inequalities to describe various S-boxes and linear layers used in common ciphers. Compared to the Convex Hull method, our QM-MILP greedy algorithm always led to smaller sets of inequalities and takes only few seconds to run even for 16-bit Super S-boxes. Thus it provides a simple way to generate inequalities for any S-box for which we know the propagation table.

Table 2: Number of inequalities required to describe valid transitions. Note that the Copy-Xor technique is lossy and does add extra variables.

Cipher	Function	CH/CX/ZR	QM	QM-MILP
Skinny	4-bit S-box	12/-/-	18	10
	8-bit S-box	-	193	105
	16-bit Super S-box (col. 0)	-	172812	48859
	16-bit Super S-box (col. 1, 2, 3)	-	173106	48964
	4-bit MixColumn	6/8/16	10	6
AES	8-bit S-box	-	87	33
LED	4-bit S-box	10/-/-	20	7
	16-bit Super S-box (col. 0)	-	387070	108199
	16-bit Super S-box (col. 1)	-	321797	78647
	16-bit Super S-box (col. 2, 3)	-	388134	108668
	16-bit MixColumn	-/32/65536	33412	33412
HIGHT	8-bit linear F_0	-/16/256	162	162
	8-bit linear F_1	-/16/256	162	162
	16-bit Super S-box (F_0)	-	15968	5493
	16-bit Super S-box (F_1)	-	69455	23671
Midori	16-bit Super S-box	-	1912088	520340

5.3 Different Strategies

We made experiments on several ciphers to observe how different modeling strategies influence the solving time of the models. We give a summary of the modeling strategies leading to the best solving time as well as the average number of lazy constraints added for each cipher considered in Table 3. Our main observation is that while hundreds of thousands inequalities are necessary to provide an exact modelization, only few hundreds are actually required to conclude on the existence of a division trail. Note that when the Super S-box (SSB) is available, it is used with an exact modelization for the first and last rounds to derive the extended distinguishers to look for.

Table 3: Fastest modeling strategy and average number of lazy constraints added per input/output pair. LC-S-box, LC-Lin and LC-SSB denote respectively the average number of lazy constraints coming from the S-box, the linear layer and the Super S-box, when relevant.

Cipher	Rounds	Modeling	LC-S-box	LC-Lin	LC-SSB
AES	4	PWL + WE	0	0	-
	5	QM + WE	-	60	-
ARIA	4	PWL + WE	0	0	-
	5	QM + CX	-	91	-
CRAFT	13	QM/CH + QM/CH	-	-	0
	14	QM/CH + CX	-	0	314
HIGHT	20	CX	-	6	0
	21	CX	-	21	0
LED	8	QM/CH + WE	-	107	54
Skinny64	11	QM/CH + QM/CH	-	-	7
	12	QM/CH + QM/CH	-	-	93
Camellia	7	PWL + WE	0	0	-
	8	QM + CX	-	31	-
CLEFIA	10	PWL + WE	0	0	-
	11	QM + CX	-	9	-

5.3.1 Strategy PWL + WE

At the beginning of this work, our bet was that the strategy PWL + WE together with lazy constraints to ensure an exact modelization, would lead to the best solving times. This strategy is quite equivalent to first searching for word-based division trails before confirming or invalidating them at the bit level. This strategy was the fastest to retrieve the integral distinguisher against both AES and ARIA reduced to 4 rounds as well as against 10-round CLEFIA. However, while being very fast in finding word-based distinguishers, it was rarely the best strategy regarding all the experiments we made. We believe there are two main reasons. The first one is that such strategy does necessarily lead to more lazy constraints being added to the model than other strategies. But lazy constraints are typically *slower* than regular ones and are not preprocessed by Gurobi. The second reason is more generic and related to the way Gurobi works. When some internal transitions are valid it would make sense to modify the current *wrong* solution as little as possible to converge to a solution. Unfortunately there is no easy way to do that and the best option is to divide ourselves the model into two or more submodels.

5.3.2 Strategy CH/QM + WE

With this strategy the S-boxes are described in an exact manner into the model while the linear layer is only modelized using the weight equality. Along our experiments QM was a bit slower than CH and QM-MILP which are equivalent. But overall the difference is not significant. In general this model is faster than the previous one and is probably the first strategy to try against a block cipher. Its main advantage is its scalability. Indeed, WE can be applied whatever the size of the matrix behind the linear layer and thus it is the best-suited to handle ciphers with very large internal states including for instance underlying block ciphers of hash functions (e.g. Whirlpool).

5.3.3 Strategy CH/QM + CH/QM

When the linear layer operates on very few bits (≈ 4), as for instance in `Skinny` or `Craft`, it is better to directly use an exact modelization for it. Actually the previous strategy will quickly lead to the same constraints but it seems Gurobi does a better job if it has access to those inequalities at the preprocessing phase.

5.3.4 Strategy CH/QM + CX

This strategy is in-between the two previous ones. With CX for the linear layer, we ensure that, when we extract the minor to check its inversibility, it does contain at least a non-zero coefficient per row and per column. This strategy typically gives faster models than WE when the linear layer is sparse. For instance, it was the fastest one 5-round `ARIA`, 11-round `CLEFIA` and for `HIGHT`. However CX requires an extra variable per non-zero entry of the matrix, wherever it is used. As a consequence, this technique hardly scales for large block ciphers and/or large matrices.

A hybrid approach. While we mainly tried models with the same modelization of a particular component through the whole cipher it may be interesting to mix them. For instance, as showed Section 4.4, the probability that WE leads to a valid transition increases with the weight of the input to reach reasonable values. Thus it is possible that ahybrid approach for which WE is used for the first rounds of the cipher and CX for the last ones, leads to a faster model.

5.3.5 Super S-boxes

The number of inequalities required to modelize a Super S-box is too high to be handled efficiently by the solver and it was always faster to use lazy constraints. We also always described internal components of a Super S-boxes into our models, somehow providing a lossy modelization of them. Indeed, if the internal transitions are all valid, the probability that the transition pass the Super S-box check is quite high, around 50% for both `Midori-64` and `Skinny-64`. Furthermore, the actual probability is much higher in practice because most of the problems occur when the input weight is high (between 12 and 15 for 16-bit Super S-boxes) and we rarely encountered more than 4 such transitions. As a consequence only few inequalities from the Super S-box propagation table are added to the model as it can be observed in Table 3. Furthermore, while constraints from Super S-boxes did invalidate some internal transitions, we did not find any input/output pair for which all trails were invalidated. In other words, we never encountered a case for which constraints from Super S-boxes led to a new distinguisher. However, handling Super S-boxes is still very important to identify the best linear maps to add at the front and at the end of a cipher to look for extended distinguishers.

5.3.6 Lazy Constraints

Many times a false-positive solution does contain several invalid transitions. In that case an interesting question is which constraints should we add to the model. Only one is necessary to discard a specific false-positive trail but it might be more efficient to add many of them at once. There is no universal answer to this question as it really depends on the models under study. In our work we found that adding several constraints at once was overall faster as long as we do not add too many of them (heuristically upper bounded by 5).

The order in which constraints are added seems to play an important role as well. In practice we found better to add constraints related to S-boxes before constraints related to linear layers. We also observed that handling rounds according to their distance to

the middle of the cipher was overall faster (i.e. the middle round is the last one to be processed).

5.4 Observations on Some Ciphers

We give the detailed results of our experiments in Appendix B, however in this section we would like to make few observations on some of these experiments.

5.4.1 HIGHT

An interesting observation for HIGHT is the number of solutions to the base model that were examined and the amount of constraints added. To conduct the full search, we needed to examine 1296 input/output pairs and determine whether or not there exists a division trail for each, possibly finding *wrong* solutions due to the lossy modeling of CX and thus needing to add lazy constraints. In total, 9626 solutions were found (including the *final* one when a division trail existed, meaning no lazy constraints were added on those) and 8334 lazy constraints were added overall. This means that on average, for a given input/output pair, the lossy model using CX is around $8331/1296 \simeq 6$ constraints *away* from a model that is complete enough for this specific input/output pair. Meanwhile with the lossy model using the WE modeling on the subset we tested (containing 81 input/output pairs), the total number of lazy constraints added was 15645, leading to an average of 193 lazy constraints added per input/output pair. To give a comparison when modeling the F_i functions with an exact modeling, using QM requires 162 inequalities *for each* application of F_0/F_1, thus $20 \times 4 \times 162 = 12960$ inequalities for a full model over 20 rounds, while ZR requires 255 inequalities for *each* application of F_0/F_1, leading to a total of 20400 inequalities for the same model. Thus HIGHT is a very good example to show that for a given input/output pair, a very large portion of the inequalities given by an exact modeling is essentially *useless* and only makes the model more complicated.

5.4.2 LED

The main problem we encountered with LED is that, in the context of the search for extended distinguisher, there is a *large* number of sets of pairs to examine, and each contains a rather large amount of pairs themselves. In total, there are 1040384 different sets to examine, each containing about 7000 pairs on average.

In addition to that, another problem we encountered was that some input/output pairs took an very large amount of time to be solved. Considering the large amount of pairs to examine, it was not reasonable to try to solve the full problem without some additional tweaking. Our solution was simply to add a time limit for each input/output pair, which we arbitrarily set to 30s. If a given set contains pairs that reached the time limit (and thus for which the existence of a division trail could not be decided), then 2 cases happen : either the set also contains a pair leading to a division trail, meaning the set cannot lead to a distinguisher and thus we can stop the search for this set, or all pairs that haven't reached the time limit lead to no division trails. In the latter case, one would have to increase the time limit to actually conclude anything, but in our case, this thankfully never happened. Note that thanks to this and the use of memoization, we only needed to examine 1409 pairs to solve all of the different input/output sets.

While not necessary in the other experiments we made, using such a time limit is again another thing to consider when searching for extended distinguishers, as it can drastically improve the total solving time.

Remark. The Super S-Boxes of LED actually do not depend on the key. While for columns 0, 2 and 3 the round constants involved do not depend on the round, this is not the case for column 1. To simplify the analysis we thus considered the round constant as a key for

the Super S-Box on column 1, and computed its propagation table using the algorithm from [DF20]. As a consequence, we only show that there is no integral distinguisher on 8 rounds that would hold independently of the round constants used in column 1 along the cipher. However, it might be possible that there exists one for specific values of the round constants (but this seems quite unlikely).

6 Conclusion

In this paper, we showed that proper use of Gurobi's functionalities to solve MILP models can greatly improve the solving times to search for distinguishers based on the division property. Especially, we showed that in a lot of cases, using a lossy model along with lazy constraints not only results in better solving times, but also allows to handle more cases than previously, such as large (Super) S-boxes or complicated linear layers without loss of accuracy.

Moreover, this strategy of lossy modeling leads to overall much simpler (i.e. less constraints) model, even when considering the lazy constraints added to remove false positives. As such, a possible direction for future work would be to provide a human readable proof of why a given case leads to no division trail, or at the very least a form of easily verifiable certificate. We also believe the research direction pointed in this paper can be extended to other cryptanalysis techniques as the search of cube attacks and differential distinguishers. In particular, it would be interesting to see whether lossy modeling and lazy constraints could be used with other variants of the division property, including the variant used to provide lower bounds on the degree of block ciphers [HLLT20] as well as the recent framework to prove resistance against integral attacks [HLLT21]. Finally, while we highlighted how the modeling step can be improved by using lazy constraints, another very common feature when solving MILP problems is the use of custom heuristics to help the solver converge (either toward a solution, or to prove infeasibility). As far as we know, no such heuristics exist for MILP problems derived from division property, or from cryptanalysis in general, while it is very common when solving *classical* problems, for example the 2-opt heuristic for the travelling salesman problem.

References

[AIK+00] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita. Camellia: A 128-bit block cipher suitable for multiple platforms - design and analysis. In Douglas R. Stinson and Stafford E. Tavares, editors, *Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, Waterloo, Ontario, Canada, August 14-15, 2000, Proceedings*, volume 2012 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2000.

[AST+17] Ahmed Abdelkhalek, Yu Sasaki, Yosuke Todo, Mohamed Tolba, and Amr M. Youssef. MILP modeling for (large) s-boxes to optimize probability of differential characteristics. *IACR Trans. Symmetric Cryptol.*, 2017(4):99–129, 2017.

[BBdS+20] Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, and Qingju Wang. Alzette: A 64-bit arx-box - (feat. CRAX and TRAX). In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara,*

CA, USA, August 17-21, 2020, Proceedings, Part III, volume 12172 of *Lecture Notes in Computer Science*, pages 419–448. Springer, 2020.

[BC20] Christina Boura and Daniel Coggia. Efficient MILP modelings for sboxes and linear layers of SPN ciphers. *IACR Trans. Symmetric Cryptol.*, 2020(3):327–361, 2020.

[BHZ08] R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

[BLMR19] Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. CRAFT: lightweight tweakable block cipher with efficient protection against DFA attacks. *IACR Trans. Symmetric Cryptol.*, 2019(1):5–45, 2019.

[DDV20] Stéphanie Delaune, Patrick Derbez, and Mathieu Vavrille. Catching the fastest boomerangs application to SKINNY. *IACR Trans. Symmetric Cryptol.*, 2020(4):104–129, 2020.

[DF20] Patrick Derbez and Pierre-Alain Fouque. Increasing precision of division property. *IACR Trans. Symmetric Cryptol.*, 2020(4):173–194, 2020.

[DR99] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[EKKT18] Zahra Eskandari, Andreas Brasen Kidmose, Stefan Kölbl, and Tyge Tiessen. Finding integral distinguishers with ease. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 115–138. Springer, 2018.

[EY21] Muhammad ElSheikh and Amr M. Youssef. On milp-based automatic search for bit-based division property for ciphers with (large) linear layers. In Joonsang Baek and Sushmita Ruj, editors, *Information Security and Privacy - 26th Australasian Conference, ACISP 2021, Virtual Event, December 1-3, 2021, Proceedings*, volume 13083 of *Lecture Notes in Computer Science*, pages 111–131. Springer, 2021.

[GLMS20] David Gérault, Pascal Lafourcade, Marine Minier, and Christine Solnon. Computing AES related-key differential characteristics with constraint programming. *Artif. Intell.*, 278, 2020.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

[Gur21] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021.

[HLK+13] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, and Donggeon Lee. LEA: A 128-bit block cipher for fast encryption on common processors. In Yongdae Kim, Heejo Lee, and Adrian Perrig, editors, *Information Security Applications - 14th International Workshop, WISA 2013, Jeju Island, Korea, August 19-21, 2013, Revised Selected Papers*, volume 8267 of *Lecture Notes in Computer Science*, pages 3–27. Springer, 2013.

[HLLT20] Phil Hebborn, Baptiste Lambin, Gregor Leander, and Yosuke Todo. Lower bounds on the degree of block ciphers. In Shiho Moriai and Huaxiong Wang, editors, *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*, volume 12491 of *Lecture Notes in Computer Science*, pages 537–566. Springer, 2020.

[HLLT21] Phil Hebborn, Baptiste Lambin, Gregor Leander, and Yosuke Todo. Strong and tight security guarantees against integral distinguishers. In Mehdi Tibouchi and Huaxiong Wang, editors, *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*, volume 13090 of *Lecture Notes in Computer Science*, pages 362–391. Springer, 2021.

[HSH+06] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A new block cipher suitable for low-resource device. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 2006.

[HWW20] Kai Hu, Qingju Wang, and Meiqin Wang. Finding bit-based division property for ciphers with complex linear layers. *IACR Trans. Symmetric Cryptol.*, 2020(1):396–424, 2020.

[KKP+03] Daesung Kwon, Jaesung Kim, Sangwoo Park, Soo Hak Sung, Yaekwon Sohn, Jung Hwan Song, Yongjin Yeom, E-Joong Yoon, Sangjin Lee, Jaewon Lee, Seongtaek Chee, Daewan Han, and Jin Hong. New block cipher: ARIA. In Jong In Lim and Dong Hoon Lee, editors, *Information Security and Cryptology - ICISC 2003, 6th International Conference, Seoul, Korea, November 27-28, 2003, Revised Papers*, volume 2971 of *Lecture Notes in Computer Science*, pages 432–445. Springer, 2003.

[LDF20] Baptiste Lambin, Patrick Derbez, and Pierre-Alain Fouque. Linearly equivalent s-boxes and the division property. *Des. Codes Cryptogr.*, 88(10):2207–2231, 2020.

[LS19] Yunwen Liu and Yu Sasaki. Related-key boomerang attacks on GIFT with automated trail search including BCT effect. In Julian Jang-Jaccard and Fuchun Guo, editors, *Information Security and Privacy - 24th Australasian Conference, ACISP 2019, Christchurch, New Zealand, July 3-5, 2019, Proceedings*, volume 11547 of *Lecture Notes in Computer Science*, pages 555–572. Springer, 2019.

[LWZ10] Yanjun Li, Wenling Wu, and Lei Zhang. Integral attacks on reduced-round ARIA block cipher. In Jin Kwak, Robert H. Deng, Yoojae Won, and Guilin Wang, editors, *Information Security, Practice and Experience, 6th International Conference, ISPEC 2010, Seoul, Korea, May 12-13, 2010. Proceedings*, volume 6047 of *Lecture Notes in Computer Science*, pages 19–29. Springer, 2010.

[McC56] Edward J McCluskey. Minimization of boolean functions. *The Bell System Technical Journal*, 35(6):1417–1444, 1956.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, volume 7537 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011.

[Qui52] Willard V Quine. The problem of simplifying truth functions. *The American mathematical monthly*, 59(8):521–531, 1952.

[Qui55] Willard V Quine. A way to simplify truth functions. *The American mathematical monthly*, 62(9):627–631, 1955.

[SHW+14] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to simon, present, lblock, DES(L) and other bit-oriented block ciphers. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 158–178. Springer, 2014.

[SSA+07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher CLEFIA (extended abstract). In Alex Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2007.

[ST17] Yu Sasaki and Yosuke Todo. New algorithm for modeling s-box in MILP based differential and division trail search. In Pooya Farshim and Emil Simion, editors, *Innovative Security Solutions for Information Technology and Communications - 10th International Conference, SecITC 2017, Bucharest, Romania, June 8-9, 2017, Revised Selected Papers*, volume 10543 of *Lecture Notes in Computer Science*, pages 150–165. Springer, 2017.

[SWLW16] Ling Sun, Wei Wang, Ru Liu, and Meiqin Wang. Milp-aided bit-based division property for arx-based block cipher. *IACR Cryptol. ePrint Arch.*, page 1101, 2016.

[SWW17] Ling Sun, Wei Wang, and Meiqin Wang. Automatic search of bit-based division property for ARX ciphers and word-based division property. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 128–157. Springer, 2017.

[SWW20] Ling Sun, Wei Wang, and Meiqin Wang. Milp-aided bit-based division property for primitives with non-bit-permutation linear layers. *IET Inf. Secur.*, 14(1):12–20, 2020.

[The20] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.0)*, 2020. `https://www.sagemath.org`.

[TM16] Yosuke Todo and Masakatu Morii. Bit-based division property and application to simon family. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 357–377. Springer, 2016.

[Tod15] Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *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*, volume 9056 of *Lecture Notes in Computer Science*, pages 287–314. Springer, 2015.

[Udo21] Aleksei Udovenko. Convexity of division property transitions: theory, algorithms and compact models. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security*, volume 13090 of *Lecture Notes in Computer Science*. Springer, 2021.

[WW11] Shengbao Wu and Mingsheng Wang. Security evaluation against differential cryptanalysis for block cipher structures. *IACR Cryptol. ePrint Arch.*, page 551, 2011.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *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*, volume 10031 of *Lecture Notes in Computer Science*, pages 648–678, 2016.

[ZR19] Wenying Zhang and Vincent Rijmen. Division cryptanalysis of block ciphers with a binary diffusion layer. *IET Inf. Secur.*, 13(2):87–95, 2019.

A Example of the Greedy Algorithm

For instance, regarding the 4-bit S-box of `Skinny`, we can construct its propagation table using the algorithm described in [DF20] and we obtain:

input	*minimal outputs*
0000	0000
1000	$1000 - 0100 - 0001$
0100	$1000 - 0100 - 0010$
0010	$1000 - 0100 - 0010 - 0001$
0001	$1000 - 0100 - 0010 - 0001$
1100	$1000 - 0100 - 0011$
1010	$1000 - 0110 - 0101 - 0011$
0110	$1000 - 0100 - 0010$

input	*minimal outputs*
1001	$1000 - 0110 - 0011$
0101	$1000 - 0100 - 0011$
0011	$1000 - 0100 - 0001$
1110	$1000 - 0110 - 0011$
1101	$1100 - 0110 - 1011$
1011	$1100 - 0111$
0111	$1000 - 0100 - 0011$
1111	1111

Applying the Quine-McCluskey algorithm leads to the following impossible transitions:

$$
\begin{array}{llll}
(\star,\star,1,\star \mid 0,0,0,0) & (\star,\star,\star,1 \mid 0,0,0,0) & (1,\star,1,\star \mid 0,\star,0,0) & (1,\star,\star,\star \mid 0,0,\star,0) \\
(\star,1,\star,1 \mid 0,0,\star,0) & (\star,\star,1,1 \mid 0,0,\star,0) & (1,1,\star,1 \mid \star,0,\star,0) & (1,\star,1,1 \mid 0,\star,\star,0) \\
(1,1,1,1 \mid \star,\star,\star,0) & (\star,1,\star,\star \mid 0,0,0,\star) & (1,\star,1,\star \mid 0,0,0,\star) & (1,1,\star,1 \mid \star,0,0,\star) \\
(1,1,1,\star \mid 0,\star,0,\star) & (1,\star,\star,1 \mid 0,\star,0,\star) & (1,1,1,1 \mid \star,\star,0,\star) & (1,1,\star,1 \mid 0,0,\star,\star) \\
 & (1,1,1,1 \mid 0,\star,\star,\star) & (1,\star,1,1 \mid \star,0,\star,\star) &
\end{array}
$$

Let see how our algorithm does effectively compact the corresponding inequalities. For instance, the impossible transition $(1,\star,\star,\star \mid 0,0,\star,0)$ can be rewritten into 4 different ways:

1. $(1 - u_0 + v_0 + v_1) + v_3 \geq 1$
2. $(1 - u_0 + v_0 + v_3) + v_1 \geq 1$
3. $(1 - u_0 + v_1 + v_3) + v_0 \geq 1$
4. $(v_0 + v_1 + v_3) + 1 - u_0 \geq 1$

None of those forms can be combined with another inequality to be compacted and thus we remove the transition and add the corresponding inequality to the system. Transitions $(\star,1,\star,\star \mid 0,0,0,\star)$, $(1,\star,\star,1 \mid 0,\star,0,\star)$ and $(1,\star,1,1 \mid \star,0,\star,\star)$ are handled in the same way. At this step, each inequality can be combined with another one. We thus take one of them that can be combined with most inequalities. For instance $(\star,1,\star,1 \mid 0,0,\star,0)$, $(\star,\star,1,1 \mid 0,0,\star,0)$ and $(\star,\star,\star,1 \mid 0,0,0,0)$ can be combined to form the inequality:

$$3(1 - u_3 + v_0 + v_1 + v_3) + 1 - u_1 + 1 - u_2 + v_2 \geq 3.$$

Again, each remaining inequality can be combined with another one and we thus pick for instance $(1,1,1,\star \mid 0,\star,0,\star)$, $(1,\star,1,\star \mid 0,\star,0,0)$ and $(1,\star,1,\star \mid 0,0,0,\star)$ which lead to the inequality:

$$3(1 - u_0 + 1 - u_2 + v_0 + v_2) + 1 - u_1 + v_1 + v_3 \geq 3.$$

At this step, $(\star,\star,1,\star \mid 0,0,0,0)$ cannot be combined with any of the remaining inequalities and we thus should add it to the system. However it can still be combined with previously processed inequalities as for instance with $(1,\star,1,\star \mid 0,0,0,\star)$. We can thus add instead the inequality:

$$2(1 - u_2 + v_0 + v_1 + v_2) + 1 - u_0 + v_3 \geq 2.$$

We continue this process, taking in priority inequalities that cannot be combined with remaining inequalities or in exactly one way. At the end we can obtain for instance the following system of 10 inequalities, 2 less than the CH method:

$$\begin{aligned}
(1 - u_0 + v_0 + v_1) + v_3 &\geq 1\\
(1 - u_1 + v_0 + v_1) + v_2 &\geq 1\\
(1 - u_0 + 1 - u_3 + v_0) + v_2 &\geq 1\\
(1 - u_0 + 1 - u_2 + 1 - u_3) + v_1 &\geq 1\\
3(1 - u_3 + v_0 + v_1 + v_3) + 1 - u_1 + 1 - u_2 + v_2 &\geq 3\\
3(1 - u_0 + 1 - u_2 + v_0 + v_2) + 1 - u_1 + v_1 + v_3 &\geq 3\\
2(1 - u_2 + v_0 + v_1 + v_2) + 1 - u_0 + v_3 &\geq 2\\
3(1 - u_0 + 1 - u_1 + 1 - u_3 + v_1) + v_0 + v_2 + v_3 &\geq 3\\
3(1 - u_0 + 1 - u_1 + 1 - u_2 + 1 - u_3) + v_0 + v_2 + v_3 &\geq 3\\
2(1 - u_0 + 1 - u_2 + 1 - u_3 + v_0) + 1 - u_1 + v_3 &\geq 2
\end{aligned}$$

B Experiments

B.1 AES

AES [DR99] is the international block cipher standard and probably the most studied block cipher. It is an SPN with block size 128 bits, using an 8-bit S-box and a linear layer built using a byte-wise permutation as well as a multiplication by a matrix over $\mathbb{F}_{2^8}$ which is applied in parallel over each 32-bit blocks of the state. The best known integral distinguisher on AES has been known since its design, covering 4 rounds and requiring 2^{32} data. We give a short summary of the timings of our experiments in Table 4.

Table 4: Solving times for AES over 4 and 5 rounds depending on the modeling. Timings are given in seconds, for the full solve for 4 rounds and over a subset of all cases for 5 rounds.

Rounds \ Model	QM + CX	QM + WE	PWL + CX	PWL + WE
4	7.7	2.7	2	**0.4**
5	14	**10**	55	22

We re-did the search for this (conventional) distinguisher, and obtained the following solving times. Using QM+CX, we needed a total of about 7.7s to show that all 128 bits at the output are balanced. Using QM+WE, the solving time comes down to about 2.7s, and to about 2s when using PWL+CX. The best solving time was obtained by using PWL+WE, requiring less than 0.4s to show that each of the 128 output bits are balanced. Moreover, note that even in this later case, using PWL+WE on its own is enough, meaning that we did not need to add lazy constraints as there was already no valid division trails.

We then searched for extended distinguishers over 5 rounds. As the Super S-box in AES is of size 32 bits, we were not able to use it to filter out trails that could possibly be invalid, and as such the precomputation was done for a single round at the start/end of the cipher, i.e. while considering a linear mapping covering a single S-box at the input/output. Testing over a subset of all sets of pairs, the shortest solving time was about 10s using QM+WE. For reference over the same subset, QM+CX requires about 14s, PWL+CX requires about 55s and finally PWL+WE requires about 22s.

In the end, using the parameters for which we obtained the fastest solving time above, the full search over all sets of pairs was done in about 13min, resulting in no distinguisher

found over 5 rounds. Note that this is technically a new result, as CX is a lossy modelization and thus, using the lazy constraints as described in Section 4.2, we are able to show that even when being exact on both the S-box and the linear layer, we still cannot find a distinguisher over 5 rounds.

B.2 ARIA

ARIA [KKP+03] is a block cipher standardized in South Korea, built as an 128-bit SPN using an 8-bit S-box. Its design differs from most recent SPNs as it's linear layer is a single matrix multiplication with a binary matrix over $\mathbb{F}_{2^8}$ applied over the whole state instead of the usual "column-wise" matrix multiplication.

The best conventional integral distinguisher [LWZ10] requires 2^{120} data, and the authors were only able to show that 24 bits at the output are balanced. We conducted the search using the same input and checking all output bits, and were able to show that *all* 128 bits at the output are balanced. Using QM+CX leads to a solving time of about 35s. Using QM+WE reduces it down to 13.5s and using PWL+CX makes it even faster, down to 3.2s. As for AES, the fastest solving time was obtained using PWL+WE, with a total solving time to check each of the 128 bits of about 0.8s. Again, as for AES, PWL+WE is enough on its own to show that all bits are balanced, without having to add any additional lazy constraints, and thus no need for an exact model over all operations.

As a new result, we were able to conduct the search for extended distinguishers over 5 rounds of ARIA. Since there is no Super S-box structure in ARIA, we did the same as for AES and do the precomputation over 1 round at the start/end of the cipher, thus considering a linear mapping covering a single S-box at the input/output. Moreover, as the linear layer is a 16×16 matrix over $\mathbb{F}_{2^8}$, we can use the strategy described in Section 3.3.1 and see this as the parallel application of 8 (linear) S-boxes, thus using QM to generate inequalities. This allows us to generate an exact model for the search using QM for both the S-box and linear layer, which, over a subset of all input/output pairs to consider, requires about 1084s (∼18min) to be solved. However, the best time over this subset was obtained using QM+CX, down to 141s, which is more than 7 times faster than directly using an exact model. Conducting the full search with these parameters, we only needed about 5 hours to solve all cases, though this resulted in no (extended) distinguisher found. A summary of all timings is given in Table 5.

Table 5: Solving times for ARIA over 4 and 5 rounds depending on the modeling. Timings are given in seconds, for the full solve for 4 rounds and over a subset of all cases for 5 rounds.

Rounds \ Model	QM+CX	QM+WE	PWL+CX	PWL+WE	QM+QM
4	35	13.5	3.2	**0.8**	-
5	**141**	284	1674	1278	1084

B.3 CRAFT

CRAFT [BLMR19] is a lightweight 64-bit block cipher mainly designed to resist Differential Fault Attacks. It uses an SPN structure, with a 4-bit S-box, a nibble wise permutation and a column-wise MixColumn operation defined with a binary matrix over $\mathbb{F}_2^4$. The best known integral distinguisher is given in the cipher's specifications, covering 13 rounds, requiring 2^{60} data and leading to 12 balanced bits. Moreover in [HLLT21], it was showed that 14 rounds of CRAFT should be safe from *any* key-independent integral distinguisher,

including extended distinguishers. Nevertheless, we also tried to search for such an extended distinguisher for the sake of completeness.

For the conventional distinguisher over 13 rounds, the fastest solving time was obtained using CH+CH, with a total time of 3.6s. Very similar times were obtained when using CH+CX, which is not very surprising as the distinguisher could already be proved using CX despite it being lossy. A more surprising result however is that using ZR for the linear layer (still with CH for the S-box) leads to about a 10 time slower solving process, requiring 38s of which 25s were dedicated to proving the balanced bits. This was rather unexpected as both CH and ZR lead to an exact modeling of the linear layer, yet lead to very different timings. Using either PWL the S-box or WE for the linear layer showed to not be very successful for 13 rounds of `CRAFT`, as the best time was obtained with PWL+CH (39s) up to 494s using PWL+WE.

For the search of extended distinguishers over 14 rounds (thus also including checks on the Super S-box), the solving times follow about the same trend, with a few variations. As in previous cases, we only examined the timings for different modeling and parameters for a subset of all sets of input/output pairs to check. The fastest solving time turned out to be using CH+CX (31s), but CH+CH and CH+ZR are close (41s and 65s respectively). Using either PWL the S-box or WE for the linear layer showed again to be overall slower, from 111s for CH+WE to 440s using PWL+WE. In the end, doing the full search with the fastest parameters lead to a total time of about 11min, leading to no distinguisher, which is in agreement with the results from [HLLT21].

Table 6: Solving times for `CRAFT` over 13 and 14 rounds depending on the modeling. Timings are given in seconds, for the full solve for 13 rounds and over a subset of all cases for 14 rounds.

Rounds \ Model	CH+CH	CH+CX	CH+ZR	PWL+CH	PWL+WE
13	**3.6**	3.8	38	39	494
14	41	**31**	65	39	440

B.4 `HIGHT`

`HIGHT` [HSH+06] is a 64-bit block cipher using a variant of the Generalized Feistel Structure with 8 branches each of size 8 bits, where some XORs in the branch merging are replaced by an addition modulo 2^8. In [DF20], Derbez and Fouque found 4 different 20-round extended distinguishers using their ad-hoc algorithm to search for such distinguishers. However, it required more than 13 days (real time) to compute on their 128-core server. We thus focused only on these extended distinguisher, in the hope of improving the search time.

As usual, we did some timings on a subset of all the sets of pairs using different modeling strategies. It turns out that the two exact modeling (QM and ZR) both give about the same timings of about 70s, while the WE modeling was the slowest, requiring about 100s. However, the fastest modeling ended up being CX, requiring only 30s to solve the given subset of cases. In the end with these parameters, the complete search took us only 12min on a standard 4-core/8-thread CPU, which is widely faster than the 13 days time over a 128-core server in [DF20] (and our algorithm could still be sped up using more parallelization).

We also ran the search for extended distinguisher over 21 rounds. Again, CX turned out to be the fastest choice with about 40s on a subset of all sets to consider, with both QM and ZR leading to about 95s and WE about 220s. Using CX, we were able to conduct the full search in about 14min, which resulted in no extended distinguisher found.

Table 7: Solving times for `HIGHT` over 20rounds depending on the modeling. Timings are given in seconds, over a subset of all cases.

Rounds \ Model	QM	ZR	CX	WE
20	70	70	**30**	100

B.5 `LED`

`LED` [GPPR11] is another 64-bit SPN block cipher, which has the particularity that, unlike a lot of other 64-bit lightweight block ciphers, is using a non-binary `MixColumn` matrix over $\mathbb{F}_{2^4}$. Conventional distinguishers were obtained over up to 7 rounds in [], however the ad-hoc algorithm from [DF20] to search for extended distinguisher was not able to finish in reasonable time over 8 rounds. We thus focused on the problem of searching for extended distinguishers over 8 rounds of `LED`.

To determine the best model parameters, we did the timings with different modeling on a rather large sample of 100 sets, totaling 700427 pairs to examine. The fastest parameters were using CH+WE and leading to 146s to solve these 100 sets, which when normalized by deducing the time spent on pairs reaching the 30s time limit (3 of them, thus 90s), gives a 56s of "effective" computation. On the other hand, the slowest solving time was 328s using PWL+CX, which gets normalized to 178s of effective computation (5 pairs reached the time limit). When comparing the normalized times, the difference is quite impressive as there is a factor of almost 10 between the fastest and slowest modelization. In the end, the full search required about 6.3h of computation on our standard laptop, which only required to examine 1409 pairs thanks to memoization. However, 392 pairs reached the 30s time limit, thus *wasting* 3.3h of computation, but resulting in only 3h of effective computation which is quite reasonable. To compare, [Udo21] was able to do the same search in about 16h using a SAT solver on twice more cores than us. Note that the total time could possibly be improved by setting a lower time limit, however due to time constraints, we were not able to experiment more as to what the lowest "good" time limit would be.

Table 8: Solving times for `LED` over 8 rounds depending on the modeling. Timings are given in seconds, over a subset of all cases, and normalized by deducing the time spent on the pairs reaching the time limit.

Rounds \ Model	CH + CX	CH + WE	PWL + CX	PWL + WE
8	73	**56**	178	79

B.6 `Skinny64`

`Skinny64` [BJK+16] is a 64-bit lightweight block cipher following an SPN structure with a 4-bit S-box and a binary `MixColumn` matrix over $\mathbb{F}_{2^4}$. Extended distinguishers were found over 11 rounds in [DF20] in 683 CPU-minutes (22 minutes on their 128-core server), and they were also able to conduct the search over 12 rounds in 152 CPU-minutes (4 minutes on their server), leading to no (extended) distinguisher. As for `HIGHT`, we looked at whether we could still improve these solving times. For 11 rounds, the best choice for the model was to use CH+CH, and we were able to find back the same distinguishers as in [DF20] in about 530s (a bit less than 9min). For 12 rounds, the best times were also obtained using CH+CH, and we were able to conduct the full search in about 80 seconds real time on a standard laptop. Both of these show a rather large improvement

over the solving time, even more so since, as mentioned previously, the only parallelization used is the one done by Gurobi internally (with 8 threads available in our case), while we treat each input/output pair one by one. As such, one would be able to improve the solving times even more so with more core available by conducting the search over multiple input/output pairs at the same time.

Table 9: Solving times for `Skinny64` over 11 and 12 rounds depending on the modeling. Timings are given in seconds over a subset of all cases.

Rounds \ Model	CH + CH	CH + WE	PWL + CH	PWL + WE
11	**238**	> 3600	> 3600	> 3600
12	**16**	109	90	289

B.7 `Camellia`

`Camellia` [AIK+00] is another 128-bit block cipher which has especially been approved by ISO/IEC. Its general structure is a Feistel network, however every 6 rounds, an additional linear layer (the FL-layer) is applied to both branches of the Feistel, making the starting round another variable to take into account. In [HWW20], Hu et al. showed that, one can find a distinguisher over 7 rounds, starting from the round of index 0 (meaning 6 rounds + FL-layer + 1 round), requiring 2^{127} data and leading to 64 balanced bits. Note that they only provided results when starting from the round of index 0, and their algorithm needed about 99min to show this distinguisher.

We ran the search for the same distinguisher, while also considering the starting round as another variable (ranging from 0 to 5, after which everything becomes symmetrical). As a result, we were able to show that this distinguisher holds for any of the possible starting rounds, each time leading to 64 balanced bits. We give the timings for each starting round and modeling in Table 10, where the columns indicate the modeling of the S-box (QM or PWL) and the modeling of the linear layer (CX or WE). Note that again, even when using a lossy model, we do not need to add lazy constraints to prove these distinguishers. In the table, we also give the time spent on proving the balanced bits, meaning the total time spent to prove that there was no division trails for 64 of the 128 output vectors. This shows an interesting behavior for the case of Camellia, as when using QM+CX the majority of the time is spent on proving the balanced bits, while when using a simpler (lossy) modeling, proving the balanced bits is only a small fraction of the time spent. Moreover, we can see that for a given starting round, there is a factor of about 4 up to 10 times between the slowest and fastest times. While all timings given here are rather short in general, this is still a non-negligible factor to consider if one were to have to run many of these searches, for example when designing a new block cipher and trying to evaluate various choices, or in the case of searching for extended distinguishers where one needs to examine a lot of different input/output pairs.

We also attempted to search for distinguishers over 8 rounds to estimate the time that would be required to solve the MILP problems while considering the use of lazy constraints for the linear layer. For a specific input division property and starting from round 0, using QM+CX requires about 240s to check that there exists a division trail for each possible output bit, while the other modeling strategies would require more than one hour at the very least. Overall, to go through each possible input division property of weight 127 as well as for each possible starting round, we needed about 18 hours in total, resulting in no distinguisher found, even while taking into account an exact modeling for the linear through the use of lazy constraints.

Table 10: Solving times for `Camellia` over 7 rounds depending on the starting round and modeling. Timings are given in seconds, where the number in parenthesis is the time spent to prove only the balanced bits. Best total time for each starting round is in bold.

Start \ Model	QM + CX	QM + WE	PWL + CX	PWL + WE
0	140 (107)	73 (19)	**30** (3.4)	35 (1.6)
1	97 (63)	43 (21)	48 (2)	**23** (1.2)
2	104 (80)	30 (20)	27 (1.9)	**11** (1.4)
3	114 (93)	35 (27)	25 (2.3)	**10** (1)
4	117 (89)	33 (20)	25 (2)	**19** (1.1)
5	128 (66)	44 (19)	**27** (2.6)	53 (1.6)

B.8 `CLEFIA`

`CLEFIA` [SSA+07] is another ISO standard, with block size 128 bits and using a Generalized Feistel structure. A distinguisher over 10 rounds requiring 2^{127} data and leading to 64 balanced bits was first given in [SWW17] using word-based division property, and [HWW20] verified this same distinguisher using bit-based division property, requiring about 82min of computations.

We searched for the same distinguisher to evaluate different modeling strategies, which led to some interesting observations. Using QM for the S-box (and either CX or WE for the linear layer), we were able to solve the 64 cases where a division trial exists (meaning those bits cannot be proved to be balanced) very fast, however we had to interrupt the search after more than 3 days of computation to try to prove the *first* balanced bit (meaning the very first case where there is no division trail). However, when switching to PWL for the S-box, using CX for the linear layer allowed us to finish all computations (including proving the 64 balanced bits) in about 4.5h, with almost all the time was spent proving the cases where there is no division trail (about 50s were spent on proving the 64 cases where there was a trail). However this can be further improved using the PWL+WE, with the total solving time coming down to only 23min, again with almost all time spent on proving the 64 balanced bits (only 15s spent on the other 64 bits).

As for `Camellia`, we attempted to search for distinguishers over one more round while taking into account the effect of having an exact modeling for the linear layer (through the use of lazy constraints). For a specific input division property of weight 127, the fastest model to check that each output bit admits a division trail required us about 176s using QM+CX, and using this modeling, we needed about 4 hours to check every input division property of weight 127, showing that none of them lead to a distinguisher even when modeling the linear layer exactly.

Table 11: Solving times for `CLEFIA` over 10 and 11 rounds depending on the modeling.

Rounds \ Model	QM + CX	QM + WE	PWL + CX	PWL + WE
10	$>$ 3 days	$>$ 3 days	4.5h	23min
11	176s	441s	$>$ 1h	$>$ 1h

B.9 `LEA`

`LEA` [HLK+13] is 128-bit block cipher developed to provide a high-speed software encryption on general-purpose processors. `LEA` is one of the cryptographic algorithms approved by

the Korean Cryptographic Module Validation Program and is the national standard of Republic of Korea. This algorithm has simple ARX structure and works on 32-bit words.

We searched for integral distinguishers using the same framework as [SWW17], where the data complexity is lowered until no distinguisher is found. The only difference with the work of Sun *et al.* is that we used a MILP model based on our new modelization of modular addition described Section 4.6. As a result, we were able to retrieve the 8-round integral distinguisher with data complexity of 2^{118} chosen plaintexts in less than 20 seconds while it took 30 minutes with their previous modelization technique.