abstractXOR: A global constraint dedicated to differential cryptanalysis

Loïc Rouquette, Christine Solnon

abstractXOR: A global constraint dedicated to differential cryptanalysis

Loïc Rouquette[1,2] and Christine Solnon[1]

(1) CITI, INRIA, INSA Lyon, F-69621 Villeurbanne
(2) LIRIS, UMR5201 CNRS, F-69621 Villeurbanne

Abstract. Constraint Programming models have been recently proposed to solve cryptanalysis problems for symmetric block ciphers such as AES. These models are more efficient than dedicated approaches but their design is difficult: straightforward models do not scale well and it is necessary to add advanced constraints derived from cryptographic properties. We introduce a global constraint which simplifies the modelling step and improves efficiency. We study its complexity, introduce propagators and experimentally evaluate them on two cryptanalysis problems (single-key and related-key) for two block ciphers (AES and Midori).

1 Motivations

Symmetric bloc ciphers use a secret key K to cipher an input text X_0 into a cipher text X_r in such a way that X_r can be deciphered back into X_0 with the same key K. Differential cryptanalysis aims at evaluating if we can guess K by studying difference propagation during ciphering [6]. These differences are obtained by applying a XOR (bitwise exclusive or, denoted $\oplus$) between two input texts. In the related-key attack [5], differences are also introduced in keys. For mounting these attacks, we must compute *Maximum Differential Characteristics (MDCs)*, *i.e.*, most probable differences.

A widely used symmetric block cipher is AES [10]. However, AES is rather time consuming, and lighter ciphers must be designed for devices with limited computational resources. Each time a new cipher is designed, we must compute MDCs to evaluate its robustness with respect to differential attacks. In this section, we illustrate MDCs on Midori128 [2], which is a lighweight cipher simpler to explain than AES. However, all models can be extended to AES and to other existing symmetric block ciphers, and we experimentally evaluate our approach on both Midori and AES.

Midori128. The ciphering iterates r *rounds* and each round is composed of four operations: *SubBytes* replaces every byte with another byte according to a given lookup table; *ShuffleCells* moves bytes; and *MixColumns* and *AddKey* perform XORs. For each round $i \in [0, r-1]$, X_i denotes the text state at the beginning of round i, and S_i, Y_i, and Z_i denote intermediate text states after each operation: S_i is the result of applying *SubBytes* on X_i; Y_i is the result of applying *ShuffleCells* on S_i; Z_i is the result of applying *MixColumns* on Y_i; and X_{i+1} is the result of applying *AddKey* on Z_i and K.

The goal of the MDC problem is to compute differences. We denote δK the differences in the key (*i.e.*, δK is the result of applying a XOR between two keys), and δX_i

Maximise $\sum_{i\in[0,r-1],b\in[0,15]} P_i[b]$ so that $\forall i \in [0,r-1], \forall b \in [0,15]$:
(C_1) $(\delta X_i[b], \delta S_i[b], P_i[b]) \in \textit{subBytesTable}_b$
(C_2) $\delta Y_i[f(b)] = \delta S_i[b]$ where f is a given permutation from $[0,15]$ to $[0,15]$
(C_3) $\delta Z_i[b] \oplus \delta Y_i[(b+4)\%16] \oplus \delta Y_i[(b+8)\%16] \oplus \delta Y_i[(b+12)\%16] = 0$
(C_4) $\delta Z_i[b] \oplus \delta K[b] \oplus \delta X_{i+1}[b] = 0$

Fig. 1: MDC problem for Midori128.

(A_0) $n = \sum_{i\in[0,r-1],b\in[0,15]} \Delta X_i[b]$
$\forall i \in [0,r-1], \forall b \in [0,15]$:
(A_1) $\Delta X_i[b] = \Delta S_i[b]$
(A_2) $\Delta Y_i[b] = \Delta S_i[f(b)]$ where f is a given permutation from $[0,15]$ to $[0,15]$
(A_3) $\Delta Z_i[b] \circ \Delta Y_i[(b+4)\%16] \circ \Delta Y_i[(b+8)\%16] \circ \Delta Y_i[(b+12)\%16] = 0$
(A_4) $\Delta Z_i[b] \circ \Delta K[b] \circ \Delta X_{i+1}[b] = 0$

Fig. 2: Step1 problem for Midori128

(resp. δS_i, δY_i, and δZ_i) the differences in the text at the beginning of round i (resp. after applying *SubBytes*, *ShuffleCells*, and *MixColumns*). For each $A \in \{K, X_i, S_i, Y_i, Z_i, X_r : i \in [0, r-1]\}$, δA is a sequence of 16 bytes (where each byte is a sequence of 8 bits) and, given a byte position $b \in [0,15]$, $\delta A[b]$ denotes the byte at position b in δA. $\delta A[b]$ is called a *differential variable*, and δ denotes the set of all differential variables. The domain of each differential variable $\delta A[b] \in \delta$ is $D(\delta A[b]) = [0,255]$.

The goal is to find the most probable assignment of differential variables. For all operations but *SubBytes*, differences are deterministically computed, *i.e.*, we can compute δX_{i+1} given δS_i and δK. In this case, the probability of observing δX_{i+1} given δS_i and δK is equal to 1. However, this is not the case for *SubBytes*: when $\delta X_i[b] \in [1,255]$, there are several possible values for $\delta S_i[b]$. The only case where we can deterministically compute $\delta S_i[b]$ given $\delta X_i[b]$ is when $\delta X_i[b] = 0$: in this case, $\delta S_i[b] = 0$. The table $\textit{subBytesTable}_b$ contains all triples $(\delta_{in}, \delta_{out}, p)$ such that p is the $\log_2$ probability that $\delta S_i[b] = \delta_{out}$ when $\delta X_i[b] = \delta_{in}$. This table depends on the position b of the byte in δS_i. We introduce a variable $P_i[b]$ which corresponds to this $\log_2$ probability and whose domain is $D(P_i[b]) = \{-6,-5,-4,-3,-2,0\}$.

Fig. 1 describes the MDC problem for Midori128. The goal is to maximise the sum of all $\log_2$ probabilities $P_i[b]$. Constraints (C_1) to (C_4) correspond to the 4 operations applied at each round: (C_1) is the table constraint corresponding to *SubBytes*; (C_2) corresponds to *ShuffleCells*, which moves bytes from position b in δS_i to position $f(b)$ in δY_i; (C_3) and (C_4) correspond to *MixColumns* and *AddKey*, respectively, and only involve XOR operations.

Two step solving process. Most differential variables are equal to 0 in MDCs. Indeed, when $\delta S_i[b] = \delta X_i[b] = 0$, the $\log_2$ probability $P_i[b]$ is equal to 0 whereas in all other cases it is smaller than or equal to -2 for Midori, and -6 for AES. Hence, the MDC problem is usually solved in two steps [16]: in Step1, we search for difference positions, whereas in Step2 we search for the exact values of the differential variables.

More precisely, at Step1, the set of variables is $\Delta = \{\Delta_j : \delta_j \in \delta\}$. Each variable $\Delta_j \in \Delta$ has a binary domain $D(\Delta_j) = \{0,1\}$ and indicates if there is a difference or not in δ_j, *i.e.*, $\Delta_j = 0 \Leftrightarrow \delta_j = 0$ and $\Delta_j = 1 \Leftrightarrow \delta_j \in [1,255]$. The Step1 problem

is defined in Fig. 2 for Midori128. It is very similar to the problem of Fig. 1. The main difference is that $\log_2$ probability variables ($P_i[b]$) are removed, and the objective function and constraint (C_1) are replaced with constraint (A_0) which ensures that the number of $\Delta X_i[b]$ variables assigned to 1 is equal to a given value n. Constraint (A_1) comes from the fact that $\delta X_i[b] = 0$ iff $\delta S_i[b] = 0$. Finally, XOR constraints (C_3) and (C_4) are replaced with abstract XOR constraints (A_3) and (A_4): an abstract XOR constraint $\Delta_1 \circ \ldots \circ \Delta_l = 0$ is satisfied iff, for each Δ_j assigned to 1 there exists an integer value in $[1, 255]$ such that the XOR of all these values is equal to 0. This constraint may be encoded by $\sum_{j=1}^{l} \Delta_j \neq 1$. Indeed, when all Δ_j are assigned to 0, the abstract XOR is trivially satisfied; when exactly one Δ_j is assigned to 1, it is trivially violated; otherwise, it is satisfied because it is always possible to find $k \geq 2$ values in $[1, 255]$ such that the result of XORing them is equal to 0. We refer to this encoding of an abstract XOR constraint as the $\text{sum}_{\neq 1}$ encoding.

Given a Step1 solution s, we define the Step2 model obtained from the model of Fig. 1 by adding the following constraint for each variable $\Delta_j \in \Delta$: if Δ_j is assigned to 0 in s then $\delta_j = 0$, else $\delta_j \neq 0$. This model is much easier to solve than the original one as many Δ_j variables are assigned to 0 in s. However, some Step1 solutions may lead to inconsistent Step2 problems. These Step1 solutions are said to be *Step2-inconsistent*. These inconsistencies mainly come from the fact that XORs are poorly abstracted at Step1: every abstract XOR constraint ensures that there exist integer values whose XOR is equal to 0, but this is ensured for each constraint separately so that several abstract XORs can be satisfied at Step1 while they are Step2-inconsistent when considering them all together. For example, the two abstract XOR constraints $\Delta_1 \circ \Delta_2 = 0$ and $\Delta_1 \circ \Delta_2 \circ \Delta_3 = 0$ are satisfied when Δ_1, Δ_2, and Δ_3 are assigned to 1, but this assignment is Step2-inconsistent because $(\delta_1 \oplus \delta_2 = 0) \Rightarrow (\delta_1 = \delta_2) \Rightarrow (\delta_3 = 0)$.

Finally, to compute MDCs, we iteratively search for all Step1 solutions with increasing values of n, and for each Step1 solution we solve the associated Step2 problem, until some conditions are reached (see [12] for details).

Existing approaches to compute MDCs. Two main dedicated approaches have been proposed to solve the Step1 problem for AES: a graph traversal approach [11], and a Branch & Bound approach [7]. Both approaches do not scale well and they are not able to solve all AES instances within a reasonable amount of time.

An appealing alternative to dedicated approaches is to use generic solvers such as Integer Linear Programming (ILP), Boolean satisfiability (SAT) or Constraint Programming (CP). ILP has been used to compute MDCs for block ciphers such as SIMON, PRESENT or LBlock [26]. However, it is difficult to model the *SubBytes* operation (modelled by constraint (C_1) in Fig. 1) by means of linear inequalities, and ILP does not scale well to solve Step2.

SAT has also been used to compute MDCs for ciphers such as ARX [21] or Simon [17]. CryptoMiniSat [23] introduces XOR-clauses and uses Gaussian elimination to efficiently propagate them. These XOR-clauses can be used to model XOR constraints (C_3) and (C_4) in Step2. However, they cannot be used to model abstract XOR constraints (A_3) and (A_4) in Step1. Indeed, if $1 \oplus 1 = 0$ at a bitwise level (during Step2), this is no longer true during Step1 because the XOR of two bytes different from 0 may be equal to 0. Similarly to ILP, non linear operations such as *SubBytes* are not straight-

forward to model by means of clauses. In [18], Lafitte shows how to encode a relation associated with a non linear operation into a set of clauses and, in [24], Sun et al. show how to reduce the number of clauses by using the same approach as in [1]. However, the resulting SAT model does not scale well and cannot solve Step2 for AES, for example.

CP has been used to compute MDCs for AES [14,15], Midori [13], and SKINNY [25]. These CP models are very efficient. However, if efficient Step2 models are easily derived from problem definitions (such as Fig. 1 for Midori128), efficient Step1 models are much harder to design. Indeed, a basic model is derived from Fig. 2 by replacing every abstract XOR with its $\text{sum}_{\neq 1}$ encoding. However, this model has a lot of Step2-inconsistent solutions. To reduce the number of Step2-inconsistent solutions, it is necessary to add constraints derived from advanced cryptographic properties.

Contributions and overview of the paper. Our goal is to ease the design of CP models for computing MDCs, while ensuring an efficient solving process. To this aim, we introduce a global constraint which propagates XORs in a global way in order to reduce the number of Step2-inconsistent solutions.

In Section 2, we introduce notations and preliminary definitions. In Section 3, we introduce the *abstractXOR* constraint which ensures that a set of abstract XOR constraints is Step2-consistent. We show that deciding of *abstractXOR* feasibility is $\mathcal{NP}$-complete when differential variables are constrained to belong to $[0, 255]$ whereas it is polynomial when the domain of differential variables is not upper bounded. Hence, we relax *abstractXOR* by removing this upper bound. In Section 4, we introduce two propagators for *abstractXOR*: the first one simply ensures feasibility, and the second one ensures Generalised Arc Consistency (GAC). In Section 5, we experimentally evaluate them on two MDC problems (related-key and single-key) for Midori and AES.

2 Notations and preliminary definitions

Given two integer values a and b, $[a, b]$ denotes the set of all integer values ranging from a to b. $\mathbb{N}^+$ denotes the set of all natural numbers (excluding 0).

Δ denotes a set of variables such that the domain of each variable $\Delta_j \in \Delta$ is $D(\Delta_j) \subseteq \{0, 1\}$. Δ_j is assigned iff $\#D(\Delta_j) = 1$, and an assignment is complete if all variables of Δ are assigned. Δ^0 denotes the set of variables assigned to 0 and $\Delta \setminus \Delta^0$ denotes the set of variables that are either assigned to 1 or not yet assigned.

C denotes a set of abstract XOR constraints defined on Δ. $C_{\downarrow \Delta^0}$ denotes the set of XOR constraints obtained from C by (i) replacing each $\Delta_j \in \Delta^0$ with 0, (ii) replacing each $\Delta_j \in \Delta \setminus \Delta^0$ with an integer variable δ_j whose domain is $D(\delta_j) = \mathbb{N}^+$, and (iii) replacing each abstract XOR $\circ$ with the bitwise XOR $\oplus$. Examples are displayed in Fig. 3 (equations of $C_{\downarrow \Delta^0}$ are simplified by replacing $\delta_j \oplus 0$ with δ_j).

$C_{\downarrow \Delta^0}$ is represented by a matrix M which contains one row for each equation and one column for each variable in $\Delta \setminus \Delta^0$: $M[i, j] = 1$ if δ_j occurs in the i^{th} equation of $C_{\downarrow \Delta^0}$; otherwise, $M[i, j] = 0$. We denote n and m the numbers of rows and columns of M. For each row $i \in [1, n]$, we define $nonZero_i = \{j \in [1, m] : M[i, j] = 1\}$, $pivot_i = \min nonZero_i$, and $nonPivot_i = nonZero_i \setminus \{pivot_i\}$. M is in *reduced row-echelon (RRE)* form iff, for every row $i \in [1, n]$, there is exactly one non-zero cell in column $pivot_i$, *i.e.*, $\sum_{i'=1}^{n} M[i', pivot_i] = 1$ (see examples in Fig. 3).

$\Delta = \{\Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5, \Delta_6, \Delta_7\}$
$C = \{\Delta_1 \circ \Delta_4 \circ \Delta_6 \circ \Delta_7 = 0, \quad \Delta_2 \circ \Delta_4 \circ \Delta_5 \circ \Delta_7 = 0, \quad \Delta_3 \circ \Delta_5 \circ \Delta_6 = 0\}$

Example 1: $\Delta^0 = \{\Delta_7\}$

$C_{\downarrow \Delta^0}$:

$$\delta_1 \oplus \delta_4 \oplus \delta_6 = 0$$
$$\delta_2 \oplus \delta_4 \oplus \delta_5 = 0$$
$$\delta_3 \oplus \delta_5 \oplus \delta_6 = 0$$

M:

δ_1	δ_2	δ_3	δ_4	δ_5	δ_6
1	0	0	1	0	1
0	1	0	1	1	0
0	0	1	0	1	1

Example 2: $\Delta^0 = \{\Delta_2, \Delta_3, \Delta_7\}$

$C_{\downarrow \Delta^0}$:

$$\delta_1 \oplus \delta_4 \oplus \delta_6 = 0$$
$$\delta_4 \oplus \delta_5 = 0$$
$$\delta_5 \oplus \delta_6 = 0$$

M:

δ_1	δ_4	δ_5	δ_6
1	1	0	1
0	1	1	0
0	0	1	1

Fig. 3: Top: A set Δ of variables and a set C of abstract XOR constraints. Bottom: $C_{\downarrow \Delta^0}$ and M when Δ_7 is assigned to 0 (Ex. 1, on the left) and when Δ_2, Δ_3, and Δ_7 are assigned to 0 (Ex. 2, on the right). In Ex. 1, M is in RRE form and $nonZero_1 = \{1, 4, 6\}$, $pivot_1 = 1$, and $nonPivot_1 = \{4, 6\}$. In Ex. 2, M is not in RRE form because the pivot columns of rows 2 and 3 have two non-zero cells.

3 Definition and complexity of *abstractXOR*

When computing MDCs in a two-step process, we aim at minimising as much as possible the number of Step1 solutions which are Step2-inconsistent. As many Step2-inconsistencies come from the fact that XOR constraints are poorly abstracted at Step1, we introduce a global constraint to obtain a tighter Step1 model.

Definition 1. *Given an integer value $u > 0$, the constraint abstractXOR$_{u,C}(\Delta)$ is satisfied by a complete assignment iff $C_{\downarrow \Delta^0} \cup \{\delta_j \leq u : \Delta_j \in \Delta \setminus \Delta^0\}$ is consistent.*

Let us consider Ex. 1 of Fig. 3. If $u = 3$, then *abstractXOR*$_{u,C}(\Delta)$ is satisfied because there exists a solution of $C_{\downarrow \Delta^0}$ such that every δ_j belongs to $[1, 3]$ (*e.g.*, $\delta_1 = \delta_5 = 1$, $\delta_2 = \delta_6 = 2$, and $\delta_3 = \delta_4 = 3$). However, if $u = 2$, then *abstractXOR*$_{u,C}(\Delta)$ is not satisfied because $C_{\downarrow \Delta^0}$ has no solution when every δ_j must belong to $[1, 2]$.

In Ex. 2, *abstractXOR*$_{u,C}(\Delta)$ is not satisfied because $(\delta_4 \oplus \delta_5 = 0 \wedge \delta_5 \oplus \delta_6 = 0) \Rightarrow (\delta_4 = \delta_5 = \delta_6) \Rightarrow (\delta_4 \oplus \delta_6 = 0)$. Therefore, δ_1 must be equal to 0, which is impossible as δ_1 must belong to $[1, u]$.

abstractXOR allows us to easily model Step1 problems. For example, for Midori128, we replace constraints (A_3) and (A_4) with *abstractXOR*$_{255,C}(\Delta)$ where $C = \{(A_3), (A_4)\}$, and Δ contains all variables involved in (A_3) or (A_4). The resulting model has less Step2-inconsistent solutions than the basic model obtained by replacing (A_3) and (A_4) with sum$_{\neq 1}$ constraints: *abstractXOR* ensures the consistency of $(C_3) \wedge (C_4)$ at Step2, whereas the basic model only ensures the feasibility of each XOR separately.

However, checking the feasibility of *abstractXOR* is intractable.

Theorem 1. *Deciding if a complete assignment satisfies abstractXOR$_{u,C}(\Delta)$ is an $\mathcal{NP}$-complete problem.*

Proof. To decide whether *abstractXOR*$_{u,C}(\Delta)$ is satisfied by a complete assignment, we must decide whether $C_{\downarrow \Delta^0}$ is consistent when all δ_j variables occurring in $C_{\downarrow \Delta^0}$ are constrained to belong to $[1, u]$. This problem trivially belongs to $\mathcal{NP}$ as we can decide in polynomial time whether a given assignment of all δ_j variables satisfies $C_{\downarrow \Delta^0}$. To show that it is $\mathcal{NP}$-complete, we give the intuition of a reduction from the graph

Algorithm 1: RRE form of an $n \times m$ matrix M

```
1 i ← 1
2 while i ≤ n do
      /* every row i' ∈ [1, i − 1] is in RRE form, i.e., Σ_{i''=1}^{n} M[i'', pivot_{i'}] = 1 */
3     if nonZero_i = ∅ then remove row i and decrement n;
4     else
5         for each row i' ∈ [1, n] such that i' ≠ i and M[i', pivot_i] = 1 do
6             for each column j' ∈ [1, m] do M[i', j'] ← M[i', j'] ⊕ M[i, j'] ;
7         i ← i + 1
```

colouring problem, which aims at deciding if we can assign a colour $c_i \in [1, u]$ to each vertex i of a graph so that $c_i \neq c_j$ for each edge (i, j). Given a graph G, we associate a variable δ_i (resp. δ_{ij}) with every vertex i (resp. edge (i, j)) of G, and we define the XOR constraints: $C = \{\delta_i \oplus \delta_j \oplus \delta_{ij} = 0 : (i, j) \text{ is an edge of } G\}$. If each variable must belong to $[1, u]$, then each XOR constraint associated with an edge (i, j) ensures that $\delta_i \neq \delta_j$ (because $\delta_i = \delta_j \Leftrightarrow \delta_{ij} = 0$). Hence, we can show that every solution of C corresponds to a valid colouring of G, and vice-versa.

Now, let us show that we can decide if *abstractXOR* is satisfied in polynomial time when δ_j variables are not upper bounded. In this case, we have to decide if $C_{\downarrow \Delta^0}$ is consistent. We first show how to put the matrix M associated with $C_{\downarrow \Delta^0}$ in RRE form. This is done by Algo. 1, which uses a principle similar to Gaussian elimination of linear equations. Algo. 1 does not change the set of solutions because it only removes empty rows (line 3), or replaces a row i' with the result of XORing it with another row i (line 6). To show that Algo. 1 puts M in RRE form, we show that the comment after line 2 is an invariant property of the loop lines 2-7. This property is trivially satisfied at the first iteration when $i = 1$ and, if it is satisfied at some iteration, then it is satisfied at the next iteration: if row i is empty (line 3) then it is removed and i is not incremented so that the property is still satisfied; otherwise (lines 4-7), every row $i' \neq i$ which contains a non-zero cell on column $pivot_i$ is XORed with row i so that column $pivot_i$ only contains one non-zero cell on row i just after lines 5-6. The complexity of this algorithm is $\mathcal{O}(mn^2)$.

We use Property *atLeast2* (defined below) to decide if $C_{\downarrow \Delta^0}$ is consistent.

Definition 2. *A matrix M in RRE form satisfies Prop.* atLeast2 *if each row has at least two non-zero cells,* i.e., $\forall i \in [1, n], \#nonZero_i \geq 2$.

Theorem 2. *$C_{\downarrow \Delta^0}$ is consistent iff its associated matrix M in RRE form satisfies Prop.* atLeast2.

Proof. If M does not satisfy Prop. *atLeast2*, then it contains a row with exactly one non-zero cell, *i.e.*, there exists an equation of the form $\delta_j = 0$. In this case $C_{\downarrow \Delta^0}$ is inconsistent as $D(\delta_j) = \mathbb{N}^+$.

If M satisfies Prop. *atLeast2*, then we can always build a solution for $C_{\downarrow \Delta^0}$. The idea is to first assign values to variables associated with non-pivot columns, and then

Algorithm 2: Check Prop. *atLeast2* of an $n \times m$ matrix M in RRE form

```
1 for each row i ∈ [1, n] such that nonPivot_i = ∅ do
2     if D(Δ_{pivot_i}) = {1} then return failure;
3     else
4         remove 1 from D(Δ_{pivot_i})
5         remove row i and decrement n
6         remove column pivot_i and decrement m
7 return success
```

compute values of variables associated with pivot columns by XORing the corresponding non-pivot variables. To ensure that values computed for pivot variables are always different from 0, we have to choose carefully the values of non-pivot variables. More precisely, non-pivot variables are assigned one after the other. When choosing a value for a non-pivot variable δ_j, for each row i such that $j \in nonPivot_i$, if all variables of $nonPivot_i$ but δ_j are already assigned, then we must choose a value different from the result of the XOR of these assigned variables. As the domains of δ_j variables are not upper bounded, we can always build a solution.

A consequence of Theorem 2 is that we can decide in polynomial time if a complete assignment satisfies *abstractXOR*$_{\infty,C}(\Delta)$. Indeed, this amounts to deciding whether $C_{\downarrow\Delta^0}$ is consistent. This can be done by using Algo. 1 to put the matrix M associated with $C_{\downarrow\Delta^0}$ in RRE form, and then checking that Prop. *atLeast2* is satisfied.

4 Propagation of *abstractXOR*

As deciding of the satisfaction of $abstractXOR_{u,C}(\Delta)$ is polynomial when $u = \infty$, we consider that $u = \infty$ from now on. In this section, we introduce an algorithm that checks feasibility (Section 4.1), an algorithm that ensures Generalised Arc Consistency (Section 4.2), and we discuss implementation and complexity issues (Section 4.3).

4.1 Checking feasibility of *abstractXOR*

Before starting the search, we build the matrix M associated with $C_{\downarrow\Delta^0}$ and use Algo. 1 to put it in RRE form. During the search, we maintain M in RRE form: each time a variable $\Delta_j \in \Delta$ is assigned to 0, we remove column j from M and, if j is the pivot column of a row i, we execute lines 3-6 of Algo. 1.

Once M is in RRE form, we check feasibility by exploiting Theo. 2, as shown in Algo. 2: for each row i with only one non-zero cell, if Δ_{pivot_i} is assigned to 1 we trigger failure, otherwise we assign 0 to Δ_{pivot_i} and remove row i and column $pivot_i$ from M.

Theorem 3. *Algo. 2 returns success iff* abstractXOR$_{\infty,C}(\Delta)$ *can be satisfied.*

Proof. Algo. 2 returns failure (line 2) when there is a row i with a single non-zero cell and the corresponding variable Δ_{pivot_i} is assigned to 1. This row corresponds to

the equation $\delta_{pivot_i} = 0$ which cannot be satisfied when $\Delta_{pivot_i} = 1$. When Algo. 2 returns success (line 7), M satisfies Prop. *atLeast2* and, for each column $j \in [1, m]$, the variable Δ_j can still be assigned to 1. In this case, Theo. 2 ensures that the constraint can be satisfied by assigning 1 to each non-assigned variable.

Algo. 2 does not only check feasibility, but also filters domains: it removes 1 from the domain of every variable Δ_{pivot_i} associated with a row i such that $nonPivot_i = \emptyset$ (line 4). This does not remove solutions as this row corresponds to the equation $\delta_{pivot_i} = 0$ which is satisfied iff $\Delta_{pivot_i} = 0$.

4.2 Ensuring the Generalized Arc Consistency of *abstractXOR*

To ensure GAC, we must ensure that for each variable $\Delta_j \in \Delta$ and each value $v \in D(\Delta_j)$, the couple (Δ_j, v) has a support, *i.e.*, there exists a consistent assignment which assigns v to Δ_j and a value $v' \in D(\Delta_{j'})$ to every other variable $\Delta_{j'} \in \Delta \setminus \{\Delta_j\}$.

By maintaining M in RRE form and ensuring Prop. *atLeast2*, we ensure that $(\Delta_j, 1)$ has a support for each variable $\Delta_j \in \Delta$ such that $1 \in D(\Delta_j)$. Also $(\Delta_j, 0)$ has a support for every variable $\Delta_j \in \Delta$ assigned to 0. However, when Δ_j is not assigned, $(\Delta_j, 0)$ may not have a support. This occurs when there exist $\Delta_j, \Delta_{j'} \in \Delta \setminus \Delta^0$ such that $D(\Delta_j) = \{0, 1\} \wedge D(\Delta_{j'}) = \{1\} \wedge C_{\downarrow \Delta^0} \Rightarrow (\delta_j = \delta_{j'})$. In this case, the couple $(\Delta_j, 0)$ has no support because $C_{\downarrow \Delta^0} \wedge (\delta_j = 0) \wedge (\delta_{j'} = 1)$ is inconsistent.

Hence, to ensure GAC we need to identify cases where the equality of two variables is a logical consequence of $C_{\downarrow \Delta^0}$. This is done by the following theorem.

Theorem 4. *For each pair of variables $\{\Delta_j, \Delta_{j'}\} \subseteq \Delta \setminus \Delta^0$, $C_{\downarrow \Delta^0} \Rightarrow (\delta_j = \delta_{j'})$ iff one of the following cases holds in the matrix M in RRE form associated with $C_{\downarrow \Delta^0}$:*
Case 1: $\exists i \in [1, n], nonZero_i = \{j, j'\}$
Case 2: $\exists i, i' \in [1, n], (pivot_i = j) \wedge (pivot_{i'} = j') \wedge (nonPivot_i = nonPivot_{i'})$

Proof. Case 1 occurs when M contains a row i with exactly two non-zero cells, and this row corresponds to the equation $\delta_j = \delta_{j'}$. Case 2 occurs when M contains 2 rows i and i' such that $nonPivot_i = nonPivot_{i'}$. These rows imply that $\delta_{pivot_i} = \delta_{pivot_{i'}}$ because both δ_{pivot_i} and $\delta_{pivot_{i'}}$ are equal to the result of XORing a same set of variables.

There is no other case where $C_{\downarrow \Delta^0} \Rightarrow (\delta_j = \delta_{j'})$ because, when M is in RRE form, every row i has a different pivot column $pivot_i$. Therefore, every equation in $C_{\downarrow \Delta^0}$ contains a different pivot variable δ_{pivot_i}. Hence, δ_j and $\delta_{j'}$ are constrained to be equal either because they occur in a same equation without any other variable, or because they are the pivot variables of two different equations which share the same non-pivot variables.

Let us illustrate these two cases on the example displayed in Fig. 3:

- If $\Delta^0 = \{\Delta_5, \Delta_7\}$ then $C_{\downarrow \Delta^0}$ contains the equation $\delta_2 \oplus \delta_4 = 0$, and if $D(\Delta_4) = \{1\}$ and $D(\Delta_2) = \{0, 1\}$, then $(\Delta_2, 0)$ has no support.
- If $\Delta^0 = \{\Delta_3, \Delta_5, \Delta_6\}$, then
  $C_{\downarrow \Delta^0}$ is equal to: $\delta_1 \oplus \delta_4 \oplus \delta_7 = 0$ and $\delta_2 \oplus \delta_4 \oplus \delta_7 = 0$
  and M is equal to:

δ_1	δ_2	δ_4	δ_7
1	0	1	1
0	1	1	1

  This implies that the pivot variables δ_1 and δ_2 are both equal to $\delta_4 \oplus \delta_7$, and if $D(\Delta_1) = \{1\}$ and $D(\Delta_2) = \{0, 1\}$, then $(\Delta_2, 0)$ has no support.

Algorithm 3: Propagation of the assignment of 1 to a variable Δ_j

```
1  let Q be an empty queue; enqueue Δ_j in Q
2  while Q is not empty do
3      dequeue a variable Δ_j from Q and remove 0 from D(Δ_j)
4      if j is the pivot column of a row i then
5          if nonPivot_i = {j'} and D(Δ_j') = {0,1} then enqueue Δ_j' in Q ;
6          else
7              for each i' ∈ [1,n] such that nonPivot_i = nonPivot_i' do
8                  if D(Δ_pivot_i') = {0,1} then enqueue Δ_pivot_i' in Q;
9      else
10         for each i ∈ [1,n] such that nonPivot_i = {j} do
11             if D(Δ_pivot_i') = {0,1} then enqueue Δ_pivot_i' in Q;
```

To maintain GAC during the search, we call Algo. 3 each time a variable must be assigned to 1. This algorithm uses a queue Q of variables that must be assigned to 1. At each iteration of the loop lines 2-11, a variable Δ_j is dequeued from Q, and it is assigned to 1. This assignment is propagated on every variable $\Delta_{j'}$ such that $C_{\downarrow \Delta^0} \Rightarrow (\delta_j = \delta_{j'})$. We exploit Theorem 4 to identify these variables:

- Case 1 has two sub-cases: if j is the pivot column of a row i, we simply check if $nonPivot_i$ is reduced to a singleton (line 5); otherwise, we have to search for every row i such that $nonPivot_i$ only contains j (lines 10-11).
- Case 2 only holds when j is the pivot column of a row i, and we have to search for every row i' such that $nonPivot_i = nonPivot_{i'}$ (lines 7-8).

Also, each time a variable is assigned to 0, we proceed as explained in Section 4.1 to check feasibility. Then, for each line which has been modified when executing lines 3-6 of Algo. 1, we check if cases 1 or 2 of Theo. 4 hold and imply that $\delta_j = \delta_{j'}$ with $D(\Delta_j) = \{0, 1\}$ and $D(\Delta_{j'}) = \{1\}$: in this case, we call Algo. 3 to propagate the assignment of 1 to Δ_j.

4.3 Implementation and complexities

Sparse Sets. Our propagators mainly involve traversing non-zero cells of rows and columns of M. As M is very sparse, we represent its rows and columns with sparse sets [19]: each sparse set contains the non-zero cells of a row or a column. This allows us to visit every non-zero cell of a column (resp. row) in linear time with respect to the number of non-zero cells instead of $\mathcal{O}(m)$ (resp. $\mathcal{O}(n)$), and to decide in constant time if an element belongs to a set. Sparse sets also allow to restore sets in constant time when backtracking, provided that we only remove elements at each choice point. Unfortunately, this is not the case here as new non-zero cells may appear when XORing lines. Hence, when backtracking, we undo all operations done before the recursive call.

Time complexity of the propagators. Let n_1 (resp. m_1) be the maximum number of non-zero cells in a row (resp. a column) of M. When using sparse sets, the complexity of putting M in RRE form, as described by Algo. 1, becomes $\mathcal{O}(nn_1m_1)$.

The complexity of the propagation of the assignment of a variable to 0 is $\mathcal{O}(n_1 m_1)$. Indeed, when a variable Δ_j is assigned to 0, we have to (i) remove column j, (ii) execute lines 3-7 of Algo. 1 if j is a pivot column, and (iii) run Algo. 2. The complexity of this depends on whether j is a pivot column or not:

- if j is a pivot column, then (i) is achieved in $\mathcal{O}(1)$ as column j only contains one non-zero cell; (ii) is achieved in $\mathcal{O}(n_1 m_1)$; and (iii) is achieved in $\mathcal{O}(n_1)$ provided that we keep track of the rows that have been modified at step (ii);
- if j is not a pivot column, then (i) is achieved in $\mathcal{O}(n_1)$ and (iii) is achieved in $\mathcal{O}(n_1)$ provided that we keep track of the rows that have been modified at step (i).

The complexity of the propagation of the assignment of a variable to 1 by Algo. 3 is $\mathcal{O}(mn_1 m_1)$. Indeed, in the worst case, this implies to assign 1 to every other variable. Hence, the loop lines 2-11 is performed $\mathcal{O}(m)$ times. The loop lines 7-8 is iterated $\mathcal{O}(n_1)$ times (we traverse non-zero cells of the column of a variable in $nonPivot_i$ to identify the rows i' for which we have to check if $nonPivot_i = nonPivot_{i'}$), and we decide if $nonPivot_i = nonPivot_{i'}$ in $\mathcal{O}(m_1)$. The loop lines 10-11 is iterated $\mathcal{O}(n_1)$ times as we only have to consider the non-zero cells of column j.

Implementation. Our global constraint has been implemented in Java and integrated in Choco 4 [22]. As its propagators are rather expensive, we give a low priority to *abstractXOR* so that, at each node of the search tree, Choco propagates all other constraints before propagating *abstractXOR*.

5 Experimental evaluation

In this section, we experimentally evaluate the interest of *abstractXOR*. We first consider the related-key MDC problem, where differences can be injected both in the key and the input text, and we report results for Midori in Section 5.1 and for AES in Section 5.2. In Section 5.3, we consider the single-key MDC problem, where differences are injected only in the input text. All experiments have been done on a single core of an Intel Xeon E3-1270v3 (3.50 GHz) with 32 GB RAM.

5.1 Related-key MDC for Midori

Description of the problem. The related-key MDC for Midori is described in Section 1 for the case where the input text X_0 is a sequence of 128 bits (denoted Midori128). Midori is also defined for 64 bit texts (denoted Midori64). In this case, *SubBytes* and *subBytesTable* are defined for 4 bit sequences instead of 8 bit sequences. Also, a key schedule is used to compute a new subkey at each round (see [2] for details).

We consider different values for r, ranging from 3 to the number of rounds defined in [2], *i.e.*, 16 (resp. 20) for Midori64 (resp. Midori128). For each value of r, the constant n used in constraint (A_0) of Fig. 2 is set to the smallest value for which there exists a solution, as this is the most difficult instance: instances with smaller values of n are often trivially inconsistent, whereas instances with larger values are useless.

We report results on two problems: $\textit{Enum}_1$ aims at enumerating all solutions of the Step1 problem described in Fig. 2 for Midori128; $\textit{Opt}_{1+2}$ aims at finding the MDC whose probability is maximal as described in Fig. 1 for Midori128.

Models for Enum$_1$. We consider two models. The first one, denoted *Enum$_1$ Global*, is derived from Fig. 2 in a straightforward way, by replacing (A_3) and (A_4) with $abstractXOR_{\infty,C}(\Delta)$ where $C = \{(A_3), (A_4)\}$ and Δ contains all variables occurring in (A_3) or (A_4). It is implemented in Java with Choco 4 [22], and we consider two propagators: *Global$_{Feas}$* only checks feasibility, as described in Section 4.1, and *Global$_{GAC}$* ensures GAC, as described in Section 4.2. In both cases, we use the minDom/wdeg variable ordering heuristic [8].

The second model, denoted *Enum$_1$ Advanced*, is introduced in [12] (and is more efficient than the model of [13]). It is obtained from Fig. 2 by replacing (A_3) and (A_4) with their sum$_{\neq 1}$ encodings, and by adding a constraint derived from a property of *MixColumns* called the *Maximum Distance Separable (MDS)* property. It further adds new variables and constraints to remove Step2-inconsistent solutions by reasoning on equality relations between Δ_j variables. This model is much more difficult to design than *Global*. For this model, we report results obtained by Picat-SAT [28], which encodes the problem into a SAT formula and then uses the SAT solver Lingeling [4]. We made experiments with other CP solvers (such as Choco, Gecode or Chuffed, for example), and we only report results obtained with Picat-SAT because it scales much better.

Models for Opt$_{1+2}$. The problem described in Fig. 1 cannot be solved within a reasonable amount of time (even for the smallest value of r) without decomposing it into two steps, as described in Section 1. We consider two models for this two step process. *Opt$_{1+2}$ Global* simply merges *Enum$_1$ Global* with the model of Fig. 1, and adds a constraint which relates δ_j and Δ_j variables, *i.e.*, $\delta_j = 0 \Leftrightarrow \Delta_j = 0$. Also, we add a variable ordering heuristic to assign Δ_j variables before δ_j variables. This model is implemented in Choco 4.

Opt$_{1+2}$ Advanced uses *Enum$_1$ Advanced* to search for Step1 solutions. However, we do not merge this model with the Step2 model of Fig. 1 and use a single solver to solve the two steps because CP solvers like Choco cannot efficiently solve *Enum$_1$ Advanced* whereas SAT solvers like Lingeling cannot efficiently solve Step2 [14]. Hence, *Opt$_{1+2}$ Advanced* uses Picat-SAT to solve *Enum$_1$ Advanced*, and each time a Step1 solution s is found, it uses Choco with the model of Fig. 1 to search for the best Step2 solution associated with s. This process is stopped either when there is no more Step1 solution, or when an optimal Step2 solution is found (*i.e.*, a solution such that all $P_i[b]$ variables are assigned to -2 as this is the largest possible value).

Results. On the top row of Fig. 4, we display the number of choice points needed to enumerate all Step1 solutions. *Global$_{GAC}$* explores less choice points than *Global$_{Feas}$*, though the difference is very small for Midori64 when $r \geq 12$.

In the middle row of Fig. 4, we display the CPU time spent to enumerate all Step1 solutions. For Midori64, the two *Global* variants have very similar times whereas, for Midori128, *Global$_{GAC}$* is faster than *Global$_{Feas}$*. *Advanced* is much slower than *Global*.

In the bottom row of Fig. 4, we display the CPU time needed to solve the full MDC problem. For Midori64, *Global$_{Feas}$* and *Global$_{GAC}$* have very similar results, and are much faster than *Advanced*. For Midori128, *Global$_{GAC}$* is faster than *Global$_{Feas}$*, which is faster than *Advanced*, especially when r increases.

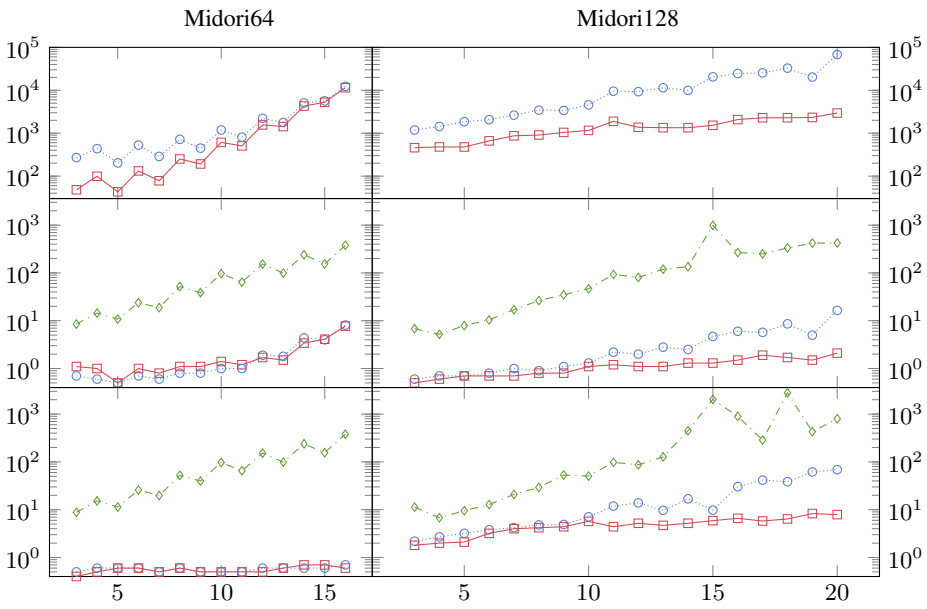

Fig. 4: Comparison of $Global_{Feas}$ (⋯○⋯), $Global_{GAC}$ (—□—), and Advanced (- ◇ -) for Midori. The x-axis gives the number of rounds r, and the y-axis the number of choice points for $Enum_1$ (up), and the run time for $Enum_1$ (Middle) and for Opt_{1+2} (bottom). Times are in seconds.

5.2 Related-key MDC for AES

Description of the problem. Like Midori, AES iterates r rounds, and each round is composed of four operations. However, AES computes a new sub-key at each round according to a key schedule which combines XOR and *SubBytes* operations. Also, the *MixColumns* operation is different and it combines XORs with a finite field multiplication by constant coefficients. This multiplication is easily modelled at Step2 using table constraints. However, it cannot be modelled at Step1 and constraint (A_3) is replaced with the following constraints which are derived from the MDS property of *MixColumns* (see [14] for more details):

$$\forall i \in [0, r-2], \forall k \in [0,3], \sum_{j=0}^{3} (\Delta Z_i[k+4j] + \Delta Y_i[k+4j]) \in \{0,5,6,7,8\}$$

$$\forall i_1, i_2 \in [0, r-2], \forall k_1, k_2 \in [0,3], \sum_{j=0}^{3} (x_{i_1 i_2 k_1 k_2 j} + y_{i_1 i_2 k_1 k_2 j}) \in \{0,5,6,7,8\}$$

where $x_{i_1 i_2 k_1 k_2 j}$ and $y_{i_1 i_2 k_1 k_2 j}$ are binary variables which are constrained as follows:

$$x_{i_1 i_2 k_1 k_2 j} = 1 \Leftrightarrow \delta Z_{i_1}[k_1 + 4j] \oplus \delta Z_{i_2}[k_2 + 4j] \neq 0$$
$$y_{i_1 i_2 k_1 k_2 j} = 1 \Leftrightarrow \delta Y_{i_1}[k_1 + 4j] \oplus \delta Y_{i_2}[k_2 + 4j] \neq 0$$

There exist three variants of AES, denoted AESl, where $l \in \{128, 192, 256\}$ corresponds to the number of bits in the key. The key schedule depends on l whereas all

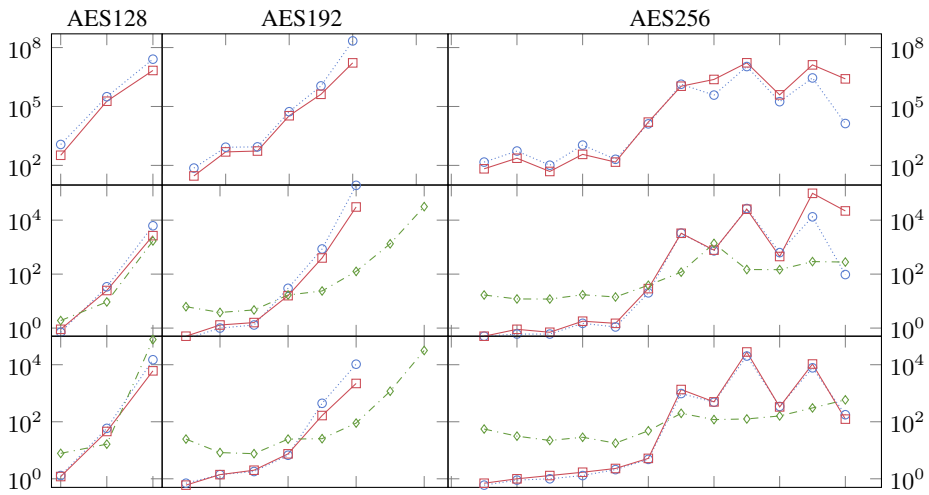

Fig. 5: Comparison of $Global_{Feas}$ (⋯○⋯), $Global_{GAC}$ (—□—), and *Advanced* (- ◇ -) for AES. The x-axis gives the number of rounds r, and the y-axis the number of choice points for $Enum_1$ (up), and the run time for $Enum_1$ (Middle) and Opt_{1+2} (bottom). Times are in seconds.

other operations do not depend on l. For each key size l, we consider different values for the number of rounds r, ranging from 3 to an upper bound which depends on MDC probabilities: when increasing r, the probability decreases and it is useless to compute MDCs whenever the $\log_2$ probability becomes smaller than -128.

Like in Section 5.1, we consider two problems: $Enum_1$ aims at enumerating all Step1 solutions, and Opt_{1+2} aims at finding the optimal MDC.

Models for $Enum_1$. We consider two CP models. *Global* is derived in a straightforward way from the definition of AES and the MDS property by replacing all XOR equations with an *abstractXOR* global constraint. It is implemented with Choco 4, and we consider two propagators (ensuring feasibility and GAC, respectively).

Advanced is the model introduced in [14] (which is more efficient than the ones of [15] and [20]). It uses a preprocessing step to infer new XOR equations from the key schedule, and it adds new variables and constraints to remove Step2-inconsistent solutions by reasoning on equality relations between Δ_j variables. This model is much more difficult to design than *Global*. It is implemented with Picat-SAT.

Models for Opt_{1+2}. Like in Section 5.1, *Global* solves the two steps with a single model implemented with Choco 4 whereas *Advanced* enumerates Step1 solutions with Picat-SAT and searches for optimal MDCs with Choco 4.

Results. On the top row of Fig. 5, we display the number of choice points needed to enumerate all Step1 solutions. In most cases, $Global_{Feas}$ explores slightly more choice points than $Global_{GAC}$. However, for 5 instances of AES256, $Global_{Feas}$ explores slightly less choice points than $Global_{GAC}$. This is a bit surprising (as ensuring GAC is stronger

than ensuring feasibility) but not impossible as filtering has an impact on the variable ordering heuristic.

In the middle row of Fig. 5, we display the time spent to enumerate all Step1 solutions. In many cases, $Global_{GAC}$ is faster than $Global_{Feas}$, but the difference is often rather small. *Advanced* is slower than *Global* when $r \leq 3$ (resp. 6 and 8) for AES128 (resp. 192 and 256), but it has better scale-up properties and it becomes faster for larger values of r. In particular, *Advanced* is able to solve AES192 when $r = 9$ (resp. $r = 10$) in 1,326s (resp. 31,611s) whereas *Global* is not able to complete the run within a time limit of 200,000s.

In the bottom row of Fig. 5, we display the time needed to solve the full MDC problem. The performance of the three approaches are rather similar to the one in the middle row. However, for many instances the fact that the two steps are solved within a single model improves the solution process. This is the case, for example, for AES128 when $r = 5$. In this case, there are 103 Step1 solutions. If *Advanced* is more efficient than $Global_{GAC}$ to enumerate these solutions (1,694s for *Advanced* instead of 2,656s for $Global_{GAC}$), *Advanced* needs much more time to find the optimal MDC (76,103s instead of 6,096s).

5.3 Experimental results for the single-key problem

In the single-key differential attack, differences are introduced only in the initial text X_0, and no difference is introduced in the key, *i.e.*, $\delta K = 0$. Like for related-key, we consider two problems: $Enum_1$ (to enumerate all Step1 solutions), and Opt_{1+2} (to find the optimal MDC). We also consider two block ciphers, *i.e.*, Midori and AES. In all cases, we consider *Global* and *Advanced* models, and these models are obtained from related-key models by assigning 0 to all variables associated with the key.

CPU times are reported in Table 1. For AES, the problem is the same whatever the length of the key (128, 192, or 256), as there is no difference in the key. For Midori, $Enum_1$ is the same whatever the length of the initial text (64 or 128) as bit sequences are abstracted by Boolean values. However, Opt_{1+2} is different for Midori64 and Midori128. Surprisingly, single-key problems are much harder to solve than related-key ones, though the size of the search space is smaller (as all variables associated with the key are assigned to 0). This comes from the fact that the number of differences (defined by the constant n in Fig. 2) is strongly increased: n is increased from 3 (resp. 4 and 5) to 7 (resp. 16 and 23) when $r = 3$ (resp. 4 and 5) for Midori, and from 5 (resp. 12) to 9 (resp. 25) when $r = 3$ (resp. 4) for AES.

Results for Midori. *Advanced* finds much more Step1 solutions than *Global*: it finds 64 (resp. 4,908) solutions when $r = 3$ (resp. 4), whereas *Global* finds 16 (resp. 68) solutions. Every solution found by *Advanced* and not by *Global* is Step2 inconsistent and *Advanced* spends a lot of time to enumerate these useless solutions. Hence, *Advanced* is not able to solve Midori within one hour when $r > 3$. When $r = 4$, *Advanced* is able to solve $Enum_1$ in 59,036s, but it is not able to solve Opt_{1+2} within a reasonable amount of time because most Step1 solutions are Step2 inconsistent.

Global is able to solve up to $r = 5$ (resp. $r = 4$) for Midori64 (resp. Midori128). Step2 is much harder for Midori128 than for Midori64 because differential variables

	Enum$_1$			*Opt*$_{1+2}$					
	Midori 64 and 128			Midori64			Midori128		
r	G_{Feas}	G_{GAC}	*Adv*	G_{Feas}	G_{GAC}	*Adv*	G_{Feas}	G_{GAC}	*Adv*
3	0.6	0.6	8.7	0.7	0.6	11.1	1.3	8.1	12.5
4	22.4	8.0	-	17.6	11.1	-	434.9	290.5	-
5	2897.8	686.8	-	2608.1	689.7	-	-	-	-

	AES 128, 192, and 256					
	Enum$_1$			*Opt*$_{1+2}$		
	G_{Feas}	G_{GAC}	*Adv*	G_{Feas}	G_{GAC}	*Adv*
3	1.3	0.7	7.5	1.1	1.0	55.0
4	-	-	-	1.2	1.4	-

Table 1: Single-Key results: Time (in seconds) needed by *Global*$_{Feas}$ (G_{Feas}), *Global*$_{GAC}$ (G_{GAC}), and *Advanced* (*Adv*) for Midori (left) and AES (right). We report '-' when time exceeds 3600s.

associated with the text take their values in $[0, 255]$ for Midori128 and in $[0, 16]$ for Midori64. Ensuring GAC often pays off and *Global*$_{GAC}$ is faster than *Global*$_{Feas}$, except for *Opt*$_{1+2}$/Midori128/$r = 3$.

Results for AES. When $r = 3$, both *Enum*$_1$ and *Opt*$_{1+2}$ are quickly solved, and *Global* is an order faster than *Advanced*. When $r = 4$, there is a huge number of Step1 solutions (we have enumerated 1,715,652 solutions within a 24 hour time limit with *Global*$_{GAC}$, and all these solutions are Step2 consistent). Hence, *Global* fails at enumerating all Step1 solutions within a reasonable amount of time. However, when merging Step1 and Step2 models to solve *Opt*$_{1+2}$, we find an optimal solution in less than 2s (the optimality proof is trivial because all $P_i[b]$ variables are assigned to the largest possible value).

When $r = 4$, the probability of the optimal MDC is equal to 2^{-150}, which is smaller than 2^{-128}. Hence, this MDC is useless to mount attacks. However, the fact that *Global* is able to enumerate a huge number of Step1 solutions in a reasonable amount of time opens new perspectives: we can search for a set of MDCs that share the same values in the initial text δX_0 and in the cipher text δX_r, and combine these MDCs to find better differentials.

6 Conclusion

We have introduced a new global constraint which eases the design of models for computing MDCs: these models are straightforwardly derived from problem definitions. This global constraint allows us to compute MDCs much faster than advanced models (which are much more difficult to design and which combine SAT and CP solvers) for single-key and related-key Midori, and for single-key AES. However, for related-key AES, it fails at solving the two largest instances of AES192 within a reasonable amount of time, and SAT has better scale-up properties for enumerating Step1 solutions. As pointed out in [14], clause learning is a key ingredient for solving this problem, and further work will aim at improving scale-up properties of Choco on this problem by adding clause learning to Choco.

We believe our new global constraint opens promising perspectives for cryptographs, and we aim at using it to solve new differential cryptanalysis problems such as those studied in [9] or [27], and new symmetric block ciphers such as Skinny [3].

Acknowledgement: This work has been funded by ANR DeCrypt (ANR-18-CE39-0007). We thank Charles Prud'homme for answering our numerous questions on Choco.

References

1. Abdelkhalek, A., Sasaki, Y., Todo, Y., Tolba, M., Youssef, A.: MILP modeling for (large) s-boxes to optimize probability of differential characteristics. IACR Trans. Symmetric Cryptol. 2017(4), 99–129 (2017)
2. Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A block cipher for low energy. In: ASIACRYPT. LNCS, vol. 9453, pp. 411–436. Springer (2015)
3. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: CRYPTO 2016 - 36th Annual International Cryptology Conference,. Lecture Notes in Computer Science, vol. 9815, pp. 123–153. Springer (2016)
4. Biere, A.: Yet another local search solver and lingeling and friends entering the sat competition 2014 pp. 39–40 (01 2014)
5. Biham, E.: New types of cryptoanalytic attacks using related keys (extended abstract). In: EUROCRYPT, LNCS 765. pp. 398–409. Springer (1993)
6. Biham, E., Shamir, A.: Differential cryptoanalysis of feal and n-hash. In: EUROCRYPT. LNCS, vol. 547, pp. 1–16. Springer (1991)
7. Biryukov, A., Nikolic, I.: Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to AES, camellia, khazad and others. In: Advances in Cryptology, LNCS 6110. pp. 322–344. Springer (2010)
8. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004. pp. 146–150. IOS Press (2004)
9. Cid, C., Huang, T., Peyrin, T., Sasaki, Y., Song, L.: Boomerang connectivity table: A new cryptanalysis tool. In: EUROCRYPT. LNCS, vol. 10821, pp. 683–714. Springer (2018)
10. FIPS 197: Advanced Encryption Standard. Federal Information Processing Standards Publication 197 (2001), u.S. Department of Commerce/N.I.S.T.
11. Fouque, P., Jean, J., Peyrin, T.: Structural evaluation of AES and chosen-key distinguisher of 9-round AES-128. In: Advances in Cryptology - CRYPTO 2013 - Part I. LNCS, vol. 8042, pp. 183–203. Springer (2013)
12. Gérault, D.: Security Analysis of Contactless Communication Protocols. Ph.D. thesis, Université Clermont Auvergne (2018)
13. Gérault, D., Lafourcade, P.: Related-key cryptanalysis of midori. In: Progress in Cryptology - INDOCRYPT 2016. LNCS, vol. 10095, pp. 287–304 (2016)
14. Gerault, D., Lafourcade, P., Minier, M., Solnon, C.: Computing AES related-key differential characteristics with constraint programming. Artif. Intell. 278 (2020)
15. Gérault, D., Minier, M., Solnon, C.: Constraint programming models for chosen key differential cryptanalysis. In: CP. LNCS, vol. 9892, pp. 584–601. Springer (2016)
16. Knudsen, L.: Truncated and higher order differentials. In: Fast Software Encryption. pp. 196–211. Springer (1995)
17. Kölbl, S., Leander, G., Tiessen, T.: Observations on the SIMON block cipher family. In: CRYPTO - 35th Annual Cryptology Conference. LNCS, vol. 9215, pp. 161–185. Springer (2015)
18. Lafitte, F.: Cryptosat: a tool for sat-based cryptanalysis. IET Information Security 12(6), 463–474 (2018)
19. Le clément de saint Marcq, V., Schaus, P., Solnon, C., Lecoutre, C.: Sparse-Sets for Domain Implementation. In: CP workshop on Techniques foR Implementing Constraint programming Systems (TRICS) (2013), https://hal.archives-ouvertes.fr/hal-01339250

20. Minier, M., Solnon, C., Reboul, J.: Solving a Symmetric Key Cryptographic Problem with Constraint Programming. In: Workshop on Constraint Modelling and Reformulation (ModRef). pp. 1–13 (2014)
21. Mouha, N., Preneel, B.: A proof that the ARX cipher salsa20 is secure against differential cryptanalysis. IACR Cryptology ePrint Archive 2013, 328 (2013)
22. Prud'homme, C., Fages, J.G., Lorca, X.: Choco Documentation. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. (2016), http://www.choco-solver.org
23. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: SAT. LNCS, vol. 5584, pp. 244–257. Springer (2009)
24. Sun, L., Wang, W., Wang, M.: More accurate differential properties of led64 and midori64. IACR Transactions on Symmetric Cryptology 2018(3), 93–123 (2018)
25. Sun, S., Gérault, D., Lafourcade, P., Yang, Q., Todo, Y., Qiao, K., Hu, L.: Analysis of aes, skinny, and others with constraint programming. In: 24th International Conference on Fast Software Encryption (2017)
26. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (related-key) differential characteristic search: Application to simon, present, lblock, DES(L) and other bit-oriented block ciphers. In: ASIACRYPT. LNCS, vol. 8873, pp. 158–178. Springer (2014)
27. Todo, Y., Isobe, T., Hao, Y., Meier, W.: Cube attacks on non-blackbox polynomials based on division property. In: CRYPTO. LNCS, vol. 10403, pp. 250–279. Springer (2017)
28. Zhou, N.F., Kjellerstrand, H., Fruhman, J.: Constraint Solving and Planning with Picat. Springer (2015)