Constraint Programming Models for Chosen Key Differential Cryptanalysis

David Gerault, Marine Minier, Christine Solnon

Constraint Programming Models for Chosen Key Differential Cryptanalysis

David Gerault[4], Marine Minier[1,2], and Christine Solnon[1,3]

[1] Université de Lyon, INSA-Lyon, F-69621, France
[2] CITI, INRIA
[3] LIRIS, CNRS UMR5205
[4] LIMOS, Clermont-Ferrand, France
{marine.minier,christine.solnon}@insa-lyon.fr, dagerault@gmail.com

Abstract. In this paper, we introduce Constraint Programming (CP) models to solve a cryptanalytic problem: the chosen key differential attack against the standard block cipher AES. The problem is solved in two steps: In Step 1, bytes are abstracted by binary values; In Step 2, byte values are searched. We introduce two CP models for Step 1: Model 1 is derived from AES rules in a straightforward way; Model 2 contains new constraints that remove invalid solutions filtered out in Step 2. We also introduce a CP model for Step 2. We evaluate scale-up properties of two classical CP solvers (Gecode and Choco) and a hybrid SAT/CP solver (Chuffed). We show that Model 2 is much more efficient than Model 1, and that Chuffed is faster than Choco which is faster than Gecode on the hardest instances of this problem. Furthermore, we prove that a solution claimed to be optimal in two recent cryptanalysis papers is not optimal by providing a better solution.

1 Introduction

Cryptography ensures properties such as confidentiality, integrity and signature of communications. Cryptanalysis aims at testing whether these properties are actually guaranteed. Whereas public key cryptography relies on hard problems, symmetric key cryptography relies on simple operations that are iterated many times to speed up encryption/decryption. The most important symmetric key primitives are hash functions and ciphers.

Hash functions guarantee integrity by creating a fixed size fingerprint of messages. Many cryptanalytic results have completely broken the standards MD5, SHA-0 and SHA-1 [23, 24, 11] by finding collisions, *i.e.*, messages with a same fingerprint. SAT solvers have been used to find collisions [15, 6, 12] and also against the future hash standard Keccak [16].

Ciphers guarantee confidentiality by encoding the original message into a different message, using a key, in such a way that the encoded message can further be decoded into the original one. Stream ciphers encode streams "on the fly", whereas block ciphers split the text in blocks which are encoded separately. Different approaches have been proposed for applying CP to cryptanalysing stream ciphers: [20] proposes to solve algebraic systems of equations that link together

keys and encoded streams; [17] uses mixed integer linear programming to compute bounds for the Enocove-128v2 stream cipher. Since the seminal works of [3, 9], several results appeared on block cipher cryptanalysis [17, 21, 22], mostly based on Mixed-Integer Programming.

Overview of the paper. In this paper, we focus on the cryptanalysis against block ciphers proposed in [3, 9] and described in Section 2. The problem is usually solved in 2 steps: In Step 1, bytes are abstracted by binary values; In Step 2, byte values are searched. In Section 3, we describe a first CP model for Step 1, initially proposed in [14]. This model generates many invalid solutions that are filtered out in Step 2 (as initially proposed in [3, 9]). In Section 4, we introduce new constraints that remove most of these invalid solutions. In Section 5, we briefly describe a CP model for Step 2. In Section 6, we evaluate scale-up properties of two classical CP solvers (Choco and Gecode) and a hybrid CP/SAT solver (Chuffed). We show that the new model for Step 1 is much more efficient than the initial model, and that Chuffed is faster than Choco which is faster than Gecode on the hardest instances of the problem. Furthermore, we prove that a solution claimed to be optimal in [3, 9] is not optimal by providing a better solution. Actually, CP allows us not only to solve cryptanalysis problems more efficiently than the dedicated approaches of [3, 9], but also in a safer way as it is easier to check the correctness of a CP model than the correctness of a dedicated program.

2 Problem Statement

In this Section, we detail the general structure of the AES (Advanced Encryption Standard) block cipher [8]. We then describe what a differential attack is and finally introduce the chosen key differential attack model.

2.1 AES block cipher

A block cipher is a function $E : \{0,1\}^n \times \{0,1\}^l \rightarrow \{0,1\}^n$ which, given a binary block X (called plaintext) of length n and a binary key K of length l, outputs a binary ciphered text $E(X, K)$ of length n such that $X = E^{-1}(E(X, K), K)$.

Most of today's block ciphers have an iterated structure: They apply a round function f r times so that $E(X, K) = X_r$ with $X_0 = X$ and $X_{i+1} = f(X_i, K_{i+1})$ for all $i \in [0; r-1]$.

Two famous examples of block ciphers are DES (Data Encryption Standard), which was the encryption standard between 1977 and 2000, and AES [8] which is the actual standard since 2001. AES ciphers blocks of length $n = 128$ bits, where each block is seen as a 4×4 matrix of bytes, where a byte is a sequence of 8 bits.

Given a 4×4 matrix of bytes M, we note $M[j]$ the 4 bytes at column $j \in [0, 3]$, and $M[j][k]$ the byte at column $j \in [0, 3]$ and row $k \in [0, 3]$.

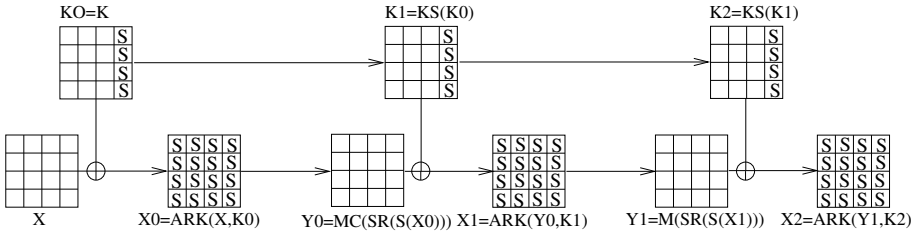


Fig. 1. AES ciphering process with $r = 2$ rounds. Each 4×4 array represents a group of 16 bytes. Before the first round, X_0 is obtained by applying ARK on the initial text X and the initial key $K = K_0$. Then, for each round $i \in [1, 2]$, S, SR and MC are applied on X_i to obtain Y_i, KS is applied on K_{i-1} to obtain K_i, and ARK is applied on K_i and Y_{i-1} to obtain X_i. Bytes that pass through the S-box are signaled by an S.

The length of keys is $l \in \{128, 192, 256\}$. The number of rounds depends on the key length: $r = 10$ (resp. 12 and 14) for $l = 128$ (resp. 192 and 256). In this Section, we describe AES for $l = 128$.

The round function f of AES uses an SPN (Substitution-Permutation Network) structure and is described in Fig. 1 for $r = 2$ rounds. Before the first round, AddRoundKey is applied on the original plaintext X and the initial key $K_0 = K$ to obtain $X_0 = ARK(X, K_0)$. Then, for each round $i \in [0, r-1]$:

- SubBytes, ShiftRows and MixColumns are applied on X_i to obtain $Y_i = MC(SR(S(X_i)))$,
- KeySchedule is applied on K_i to obtain $K_{i+1} = KS(K_i)$,
- AddRoundKey is applied on Y_i and K_{i+1} to obtain $X_{i+1} = ARK(Y_i, K_{i+1})$.

These different operations are described below.

SubBytes S. The S operation, also called S-box, is a non-linear permutation which is applied on each byte of X_i separately, *i.e.*, for each $j, k \in [0, 3]$, S substitutes $X_i[j][k]$ by $S(X_i[j][k])$, according to a lookup table.

ShiftRows SR. SR is a linear mapping that rotates on the left by one byte position (resp. 2 and 3 byte positions) the second row (resp. third and fourth rows) of the current matrix $S(X_i)$, *i.e.*, for each $j, k \in [0, 3]$:

$$SR(S(X_i))[j][k] = S(X_i)[(k + j)\%4][k]$$

MixColumns MC. MC is a linear mapping that multiplies each column of the input matrix $SR(S(X_i))$ by a 4×4 fixed matrix chosen for its good properties of diffusion (see [5]). In particular, it has the Maximum Distance Separable (MDS) property: For each column $j \in [0, 3]$, it ensures:

$$w(SR(S(X_i))[j]) + w(MC(SR(S(X_i)))[j]) \in \{0, 5, 6, 7, 8\}$$

where w is a function which returns the number of bytes different from 0^8 (we note 0^8 the byte composed of 8 bits equal to 0).

AddRoundKey ARK. ARK performs a xor operation (noted $\oplus$) between Y_i and subkey K_{i+1}, *i.e.*, for each column and row $j, k \in [0, 3]$,

$$ARK(Y_i[j][k], K_{i+1}[j][k]) = Y_i[j][k] \oplus K_{i+1}[j][k]$$

KeySchedule KS. The subkey at round 0 is the initial key, *i.e.*, $K_0 = K$. For each round $i \in [1, r]$, the subkey K_i is generated from the previous subkey K_{i-1} by applying the key schedule, *i.e.*, $K_i = KS(K_{i-1})$. For keys of length $l = 128$ bits, each subkey K_i is a 4×4 byte matrix. KS operates on columns:

- It first computes the first column $K_i[0]$ from K_{i-1} as follows:

  $$\forall k \in [0; 3], K_i[0][k] = K_{i-1}[0][k] \oplus S(K_{i-1}[3][(k + 1)\%4])$$

  where S is the SubBytes operation. Moreover, $r+1$ predefined constants are added to $K_i[0][0]$.
- For the last 3 columns $j \in \{1, 2, 3\}$, we have: $K_i[j] = K_i[j - 1] \oplus K_{i-1}[j]$

2.2 Differential Cryptanalysis

Differential cryptanalysis was introduced in 1991 [2], and aims at evaluating confidentiality by testing whether it is possible to find the secret key within a reasonable number of trials. The idea is to consider plaintext pairs (X, X') and to study the propagation of the initial difference between X and X' while going through the successive rounds. We note δX_i the xor difference between the two plaintexts X_i and X'_i obtained after the ith round of the ciphering of X and X', *i.e.*, $\delta X_i = X_i \oplus X'_i$, and we say that $\delta X_i[j][k] = X_i[j][k] \oplus X'_i[j][k]$ is a *differential byte* (for each column and row $j, k \in [0, 3]$).

Let us keep in mind that the round function f is composed of a set of linear operations (SR, MC, ARK) and a non linear operation (S). The linear operations only move differences to other places. Indeed, for every linear operation $l \in \{SR, MC, ARK\}$, we have $l(A \oplus B) = l(A) \oplus l(B)$. So, we can easily predict how differences are propagated from δX_i to δX_{i+1} by these operations.

The non linear operation S has to be studied more carefully. As said before, S operates on each byte $X_i[j][k]$ separately. Therefore, we need to study how the S-box propagates differences for a pair (A, B) of bytes. To this aim, we evaluate the probability that the output difference $S(A) \oplus S(B)$ is equal to β when the input difference $A \oplus B$ is equal to α, where α and β are bytes. This probability is denoted $D_{\alpha,\beta}$ and is defined by

$$D_{\alpha,\beta} = \frac{\#\{(A, B) \in \{0, 1\}^8 \times \{0, 1\}^8 \mid (A \oplus B = \alpha) \wedge (S(A) \oplus S(B) = \beta)\}}{256}$$

For example, let us consider an input difference $\alpha = 00000001$ and an output difference $\beta = 00100000$. For the AES S-box, the transition from 00000001 to 00100000 only occurs for 4 couples of inputs, among the 256 possible couples so that $D_{00000001,00100000} = \frac{4}{256}$. For the AES S-box, most of the times the transition

probability is equal to $\frac{0}{256}$ or $\frac{2}{256}$, and rarely to $\frac{4}{256}$. Note that S is a bijection so that $A \oplus B = 0^8 \Leftrightarrow S(A) \oplus S(B) = 0^8$. As a consequence, $D_{0^8,0^8} = 1$. In other words, if there is no difference in the input $A \oplus B$, then there is no difference in the output $S(A) \oplus S(B)$.

Then, for each round $i \in [0; r]$, we study difference propagation when the 16 bytes of X_i and X'_i pass through the S-box. For each column $j \in [0; 3]$ and each row $k \in [0; 3]$, we note $\delta X_i[j][k] = X_i[j][k] \oplus X'_i[j][k]$ and $\delta SX_i[j][k] = S(X_i[j][k]) \oplus S(X'_i[j][k])$ the difference for the byte at column j and row k before and after passing the S-box, respectively. The probability of obtaining the output difference $\delta SX_i[j][k]$ when the input difference is $\delta X_i[j][k]$ is given by $D_{\delta X_i[j][k],\delta SX_i[j][k]}$. Hence, the probability of obtaining the output difference $\delta X_r = X_r \oplus X'_r$ after r rounds given an input difference $\delta X = X \oplus X'$ is:

$$p_1(\delta X_r|\delta X) = \prod_{i=0}^{r} \prod_{j=0}^{3} \prod_{k=0}^{3} D_{\delta X_i[j][k],\delta SX_i[j][k]} \tag{1}$$

We refer the reader to [2] for more details.

A first goal of the attacker is to find the values of δX_i for $i \in \{0, \dots r\}$ which maximize the probability p_1. Once done, the attacker retrieves some partial information on the secret key K from the next subkey $K_{r+1} = KS(K_r)$. To do so, the attacker has to cipher M chosen plaintext pairs (X, X') to obtain M ciphered pairs (C, C'). From those pairs (C, C'), the attacker deciphers the last round to partially retrieve δX_r according to all possible values of some bits of K_{r+1}. The correct key will be the one for which the optimal value of δX_r (that maximizes p_1) appears the most frequently. The number M of plaintext pairs required for the success of the attack may be directly computed from p_1 and is equal to c/p_1 for c a small constant as shown in [2].

2.3 Chosen Key Differential Cryptanalysis

Today, differential cryptanalysis is public knowledge, so modern block ciphers such as AES have been designed to have proven bounds against differential attacks. However, in 1993, E. Biham proposed a new type of attack called related key attack [1] that allows an attacker to inject differences not only between the plaintexts X and X' but also between the keys K and K' (even if the secret key K stays unknown from the attacker) to try to mount more powerful attacks. The goal of the attack stays the same as previously, *i.e.*, try to find some partial information on the secret key K by testing some bits of the last subkey K_r on the veracity of the differential relation which happens with probability p_1.

We note $K_i[j][k]$ and $K'_i[j][k]$ the bytes at column j and row k in the subkeys of K and K' at round i, $\delta K_i[j][k]$ the difference between $K_i[j][k]$ and $K'_i[j][k]$, i.e., $\delta K_i[j][k] = K_i[j][k] \oplus K'_i[j][k]$, and $\delta SK_i[j][k]$ the difference between $S(K_i[j][k])$ and $S(K'_i[j][k])$, i.e., $\delta SK_i[j][k] = S(K_i[j][k]) \oplus S(K'_i[j][k])$. As the only bytes of the subkeys that pass through the S-box are those that are at column $j = 3$, equation (1) is modified by multiplying it by $D_{\delta K_i[3][k],\delta SK_i[3][k]}$, for each round

i and each line k, *i.e.*, the goal of the attacker is to find the values of δX_i and δK_i for $i \in \{0, \dots r\}$ which maximize the probability p_2 defined by equation (2).

$$p_2(\delta X_r, \delta K_r | \delta X, \delta K) = p_1(\delta X_r | \delta X) * \prod_{i=0}^{r} \prod_{k=0}^{3} D_{\delta K_i[3][k], \delta SK_i[3][k]} \quad (2)$$

2.4 Two step solving process for chosen key differential cryptanalysis

Two main papers [3, 9] describe results for the chosen key differential cryptanalysis of the AES and propose algorithms for finding initial pairs of plain texts and keys which maximize the probability p_2. In both papers, the problem is solved in two steps.

First step: Search of binary solutions. In the first step, each unknown δX_i is modeled with a 4×4 byte matrix, and a binary variable $\Delta X_i[j][k]$ is associated with every differential byte $\delta X_i[j][k]$. These binary variables are equal to 0 if their associated differential bytes are equal to 0^8, *i.e.*,

$$\Delta X_i[j][k] = 0 \Leftrightarrow X_i[j][k] = X'_i[j][k] \Leftrightarrow \delta X_i[j][k] = 0^8$$

and they are equal to 1 otherwise. We also associate binary variables $\Delta K_i[j][k]$ and $\Delta Y_i[j][k]$ with every differential byte $\delta K_i[j][k] = K_i[j][k] \oplus K'_i[j][k]$ and $\delta Y[j][k] = Y_i[j][k] \oplus Y'_i[j][k]$, respectively.

The operations that transform δX into δX_r (described in Section 2.1 and Fig. 1), are translated into constraints between these binary variables. In this first step, the goal is to find solutions which satisfy these constraints. Solutions of this first step are called *binary solutions*. Note that during this first step, the SubBytes operation S is not considered. Indeed, the S-box does not introduce nor remove differences, i.e., for all bytes A and B, $(A \oplus B = 0^8) \Leftrightarrow (S(A) \oplus S(B) = 0^8)$.

Second step: Search of byte solutions. In the second step, we try to transform binary solutions found in the first step into *byte solutions*. More precisely, for each binary variable $\Delta X_i[j][k]$, $\Delta Y_i[j][k]$ or $\Delta K_i[j][k]$ set to 1 in the binary solution, we search for a byte value $\delta X_i[j][k]$, $\delta Y_i[j][k]$ or $\delta K_i[j][k]$ different from 0^8 so that the AES transformation rules are satisfied. Note that some binary solutions may not be transformable into byte solutions. These binary solutions are called *byte-inconsistent binary solution*, whereas binary solutions that can be transformed into byte solutions are called *byte-consistent binary solutions*. Note also that a byte-consistent binary solution may be transformable into more than one byte solution.

Objective function. The goal is to find a byte solution that maximizes probability p_2 of equation (2), while being strictly lower than 1 (*i.e.*, there must be at least one difference between the initial plain texts and keys). It has been shown that a byte solution that maximizes probability p_2 also maximizes the number of

factors $D_{\alpha,\beta}$ of equation (2) for which $\alpha = \beta = 0$ (because $D_{0^8,0^8} = 1$ whereas $D_{\alpha,\beta} \leq \frac{4}{256}$ if $(\alpha, \beta) \neq (0^8, 0^8)$). Therefore, we introduce a variable obj which is equal to the number of $\Delta X_i[j][k]$ and $\Delta K_i[3][k]$ variables of equation (2) which are set to 1:

$$obj = \sum_{i=0}^{r} \sum_{j=0}^{3} \sum_{k=0}^{3} \Delta X_i[j][k] + \sum_{i=0}^{r} \sum_{k=0}^{3} \Delta K_i[3][k]$$

We add the constraint $obj \geq 1$, to ensure that probability p_2 is strictly lower than 1. To find a byte solution that maximizes p_2, we first have to find byte-consistent binary solutions that minimize the value of obj. Note that there may exist byte-inconsistent binary solutions that have a smaller obj value. However, these binary solutions must be discarded as it is not possible to transform them into byte solutions. Finally, among all byte-consistent binary solutions that minimize obj, we have to search for the one that maximizes the actual probability p_2 associated with its best byte solution.

Existing approaches to solve the problem. In [3], step 1 is solved with a dedicated Branch & Bound approach. In a preliminary study, we have implemented this approach in C programming language. For $r = 3$ (resp. $r = 4$), we found the optimal binary solution in about one hour (resp. 24 hours) on a single core PC. In [9], step 1 is solved by performing a breadth-first traversal of a state-transition graph that has about $2^{33.6}$ nodes for a 128 bit key length. The graph needs 60GB of memory and it is pre-computed in 30 minutes on a 12-core computer for $r = 5$. Using this graph, binary solutions are found in a few seconds. Both in [3] and [9], it is claimed that the optimal solution for $r = 4$ rounds has an objective value $obj = 13$. We shall see in Section 6 that there exists a better solution.

3 First CP Model for Step 1

In this section, we describe a CP model for the first step described in Section 2.4. This model was initially introduced in [14].

3.1 Variables

Let r be the number of rounds, and let $l = 128$ be the length of the key. We define the following binary variables (see Fig. 1 for an overview of the bytes associated with these variables):

- For each column and row $j, k \in [0; 3]$, $\Delta X[j][k]$ is the variable associated with the differential byte $\delta X[j][k] = X[j][k] \oplus X'[j][k]$.
- For each round $i \in [0; r]$ and for each column and row $j, k \in [0; 3]$, $\Delta X_i[j][k]$ and $\Delta K_i[j][k]$ are variables associated with differential bytes $\delta X_i[j][k] = X_i[j][k] \oplus X'_i[j][k]$ and $\delta K_i[j][k] = K_i[j][k] \oplus K'_i[j][k]$, respectively.

– For each round $i \in [0; r-1]$ and for each column and row $j, k \in [0; 3]$, $\Delta Y_i[j][k]$ is the variable associated with the differential byte $\delta Y_i[j][k] = Y_i[j][k] \oplus Y'_i[j][k]$.

All these variables are binary variables, which are set to 0 when the associated differential byte is 0^8 and to 1 otherwise.

3.2 Constraints

Constraints correspond to the propagation of differences by operations of the round function f. As said before, the non linear operation S does not imply any constraint as it neither introduces nor removes differences. The linear ARK and KS operations involve xor operations. Therefore, we first define a xor constraint. Then, we define constraints implied by the SR, MC, ARK and KS operations.

xor. Let us consider three differential bytes δA, δB and δC such that $\delta A \oplus \delta B = \delta C$. If $\delta A = \delta B = 0^8$, then $\delta C = 0^8$. If ($\delta A = 0^8$ and $\delta B \neq 0^8$) or ($\delta A \neq 0^8$ and $\delta B = 0^8$) then $\delta C \neq 0^8$. However, if $\delta A \neq 0^8$ and $\delta B \neq 0^8$, then we cannot know if δC is equal to 0^8 or not. When abstracting differential bytes δA, δB and δC with binary variables ΔA, ΔB and ΔC (which only model the fact that there is a difference or not), we obtain the following definition of the xor constraint:

$$xor(\Delta A, \Delta B, \Delta C) \Leftrightarrow \Delta A + \Delta B + \Delta C \neq 1$$

AddRoundKey. At the beginning of the ciphering process, ARK performs xor operations on ΔX and ΔK_0 to obtain ΔX_0, *i.e.*,

$$\forall (j, k) \in [0; 3]^2, xor(\Delta X[j][k], \Delta K_0[j][k], \Delta X_0[j][k])$$

Then, for each round $i \in [1, r]$, ARK performs xor operations on ΔY_{i-1} and ΔK_i to obtain ΔX_i, *i.e.*,

$$\forall i \in [1, r], \forall (j, k) \in [0; 3]^2, xor(\Delta Y_{i-1}[j][k], \Delta K_i[j][k], \Delta X_i[j][k])$$

ShiftRows and MixColumns. For each round $i \in [0, r-1]$, SR and MC are applied on ΔX_i to obtain ΔY_i, and MC ensures MDS (see Section 2.1), *i.e.*,

$$\forall i \in [0, r-1], \forall j \in [0; 3], \sum_{k=0}^{3} \Delta X_i[(k+j)\%4][k] + \Delta Y_i[j][k] \in \{0, 5, 6, 7, 8\}$$

KeySchedule. KS is applied at each round i to compute ΔK_i from ΔK_{i-1}, and it is composed of xor operations between some columns of the key, *i.e.*,

$$\forall i \in [1, r], \forall k \in [0; 3], xor(\Delta K_{i-1}[0][k], \Delta K_{i-1}[3][(k+1)\%4], \Delta K_i[0][k])$$
$$\forall i \in [1, r], \forall j \in [1; 3], \forall k \in [0; 3], xor(\Delta K_i[j-1][k], \Delta K_{i-1}[j][k], \Delta K_i[j][k])$$

3.3 Objective Variable

Finally, we introduce an integer variable obj, whose domain is $[1, \frac{l}{6}]$[5], and we define obj as the number of differential variables on which a non linear S operation is performed, *i.e.*,

$$obj = \sum_{i=0}^{r} \sum_{j=0}^{3} \sum_{k=0}^{3} \Delta X_i[j][k] + \sum_{i=0}^{r} \sum_{k=0}^{3} \Delta K_i[3][k]$$

3.4 Ordering heuristics

As we want to minimize the number of $\Delta X_i[j][k]$ and $\Delta K_i[j][3]$ variables set to 1, we add a variable ordering heuristic that first assigns these variables, and a value ordering heuristic that first tries to assign them to 0.

3.5 Limitations of the first CP model for Step 1

In [14], we evaluated the CP model described in Section 3 (implemented with Choco 3 [19]) on two problems: the optimization problem, the goal of which is to find a binary solution that minimizes the value of obj, and the enumeration problem, the goal of which is to find all binary solutions for a given value of obj (corresponding to the optimal one). These very first experimental results showed us that Choco is able to solve these problems up to $r = 5$ rounds in a reasonable amount of time. Note that it has been shown in [3] that it is useless to try to solve these problems for more than 5 rounds because no valid characteristics exist beyond this limit. However, solutions for low values of r are used as a basis to build attacks with larger values of r. For example, [3] shows how to build an attack for $r = 12$ and $l = 192$ by combining 2 solutions with $r = 4$. Hence, it is very useful to find solutions with lower values of r.

For these two problems, binary solutions are not necessarily byte-consistent. In particular, it may happen that the binary solution of the optimization problem is byte-inconsistent. For instance, for $r = 3$ rounds, the optimal binary solution has a cost of $obj = 3$ and there exist 512 binary solutions with this cost. However, none of these solutions are byte-consistent: The optimal byte-consistent binary solution has a cost of $obj = 5$. When solving the enumeration problem with this cost, we find $21,504$ solutions, among which only 2 are byte-consistent. This means that we spend most of the time at generating useless binary solutions which are discarded in the second step because they are byte-inconsistent. Note that approaches proposed by [3, 9] also suffer from the same problem.

[5] The upper bound $\frac{l}{6}$ comes from the fact that $D_{\alpha,\beta} \leq 2^{-6}, \forall(\alpha, \beta) \neq (0^8, 0^8)$, and probability p_2 must be larger than 2^{-l} which corresponds to a probability with uniform distribution of the 2^l possible keys.

4 Additional constraints for step 1

In this section, we introduce new variables and constraints that are added to the first CP model described in Section 3. They are used to infer equality relations between differential bytes, and these relations are used to propagate the MDS property of MixColumns at the byte level. They remove most binary solutions that cannot be transformed into byte solutions, thus speeding up the solution process.

4.1 Propagation of MDS at the byte level

For each round $i \in [0, r-1]$ and each column $j \in [0, 3]$, the MDS property of MixColumns (introduced in Section 2.1) ensures:

$$\sum_{k=0}^{3} w(X_i[(k+j)\%4][k]) + w(Y_i[j][k]) \in \{0, 5, 6, 7, 8\}$$

At differential byte level, this property still holds:

$$\sum_{k=0}^{3} w(\delta X_i[(k+j)\%4][k]) + w(\delta Y_i[j][k]) \in \{0, 5, 6, 7, 8\}$$

In the first model, this property is ensured by the constraint:

$$\sum_{k=0}^{3} \Delta X_i[(k+j)\%4][k] + \Delta Y_i[j][k] \in \{0, 5, 6, 7, 8\}$$

However, the MDS property also holds for any xor difference between two different columns in two different rounds of the differential byte model: $\forall i_1, i_2 \in [0, r-1], \forall j_1, j_2 \in [0, 3]$,

$$\begin{aligned}\sum_{k=0}^{3} & w(\delta X_{i_1}[(k+j_1)\%4][k] \oplus \delta X_{i_2}[(k+j_2)\%4][k]) \\ + \; & w(\delta Y_{i_1}[j_1][k] \oplus \delta Y_{i_2}[j_2][k]) \in \{0, 5, 6, 7, 8\}\end{aligned}$$

To ensure this property (that removes most byte-inconsistent boolean solutions), we introduce new boolean variables, called equality variables: For each pair of differential bytes δA and δB (in δX_i, δY_i, and δK_i matrices), we introduce the boolean equality variable $EQ_{\delta A, \delta B}$ which is equal to 1 if $\delta A = \delta B$, and to 0 otherwise. Using these differential byte equality variables, the MDS property between different columns is ensured by the following constraint:
$\forall j_1, j_2 \in [0, 3], \forall i_1, i_2 \in [0, r-1]$,

$$\sum_{k=0}^{3} EQ_{\delta X_{i_1}[(k+j_1)\%4][k], \delta X_{i_2}[(k+j_2)\%4][k]} + EQ_{\delta Y_{i_1}[j_1][k], \delta Y_{i_2}[j_2][k]} \in \{0, 1, 2, 3, 8\}$$

4.2 Constraints on equality variables

In this section, we define constraints that hold on equality variables.

Constraints derived from xor constraints. As pointed out in Section 3.2 when defining the constraint $xor(\Delta A, \Delta B, \Delta C)$ (where ΔA, ΔB and ΔC are binary variables associated with differential bytes δA, δB and δC, respectively), if $\Delta A = \Delta B = 1$, then we cannot know if ΔC is equal to 0 or 1. However, whenever $\Delta C = 0$ (resp. $\Delta C = 1$), we know for sure that the corresponding byte δC is equal to 0^8 (resp. different from 0^8), meaning that the two bytes δA and δB are equal (resp. different), *i.e.*, that $EQ_{\delta A,\delta B} = 1$ (resp. $EQ_{\delta A,\delta B} = 0$). The same reasoning may be done for ΔA and ΔB because $(\delta A \oplus \delta B = \delta C) \Leftrightarrow (\delta B \oplus \delta C = \delta A) \Leftrightarrow (\delta A \oplus \delta C = \delta B)$. Therefore, we redefine the xor constraint as follows:

$$\begin{aligned} xor(\Delta A, \Delta B, \Delta C) \Leftrightarrow ((\Delta A + \Delta B + \Delta C \neq 1) \\ \wedge \ \ (EQ_{\delta A,\delta B} = 1 - \Delta C) \\ \wedge \ \ (EQ_{\delta A,\delta C} = 1 - \Delta B) \\ \wedge \ \ (EQ_{\delta B,\delta C} = 1 - \Delta A)) \end{aligned}$$

Constraints to ensure that equality variables define an equivalence relation. Symmetry is ensured by

$$\forall \delta A, \delta B, EQ_{\delta A,\delta B} = EQ_{\delta_B,\delta_A}$$

and transitivity by

$$\forall \delta A, \delta B, \delta C, (EQ_{\delta A,\delta B} = EQ_{\delta B,\delta C} = 1) \Rightarrow (EQ_{\delta_A,\delta_C} = 1)$$

Constraints that relate equality variables with binary differential variables. For each pair of differential bytes $\delta A, \delta B$ such that the corresponding binary variables are ΔA and ΔB, respectively, we have:

$$\begin{aligned} (EQ_{\delta A,\delta B} = 1) \Rightarrow (\Delta A = \Delta B) \\ EQ_{\delta A,\delta B} + \Delta A + \Delta B \neq 0 \end{aligned}$$

4.3 Constraints derived from KS

The KeySchedule (described in Section 2.1) mainly performs xor operations: At each round i, the first column $K_i[0]$ is obtained by performing a xor between bytes of $K_{i-1}[0]$ and $K_{i-1}[3]$; for the last three columns $j \in \{1, 2, 3\}$, $K_i[j]$ is obtained by performing a xor between $K_{i-1}[j]$ and $K_i[j-1]$. Besides these xor operations, all bytes of $K_{i-1}[3]$ pass through the S-box before xoring them with $K_{i-1}[0]$ to obtain $K_i[0]$. Therefore, each byte of K_i, for each round $i \in [1, r]$ may be expressed as a combination of xor operations between bytes of the initial key K_0, and bytes obtained by applying the S operation on column 3 of rounds $j < i$. For example (recall that $A \oplus A = 0^8$ and $0^8 \oplus A = A$):

$$\begin{aligned} K_2[1][1] &= K_2[0][1] \oplus K_1[1][1] \\ &= K_1[0][1] \oplus S(K_1[3][2]) \oplus K_1[0][1] \oplus K_0[1][1] \\ &= S(K_1[3][2]) \oplus K_0[1][1] \end{aligned}$$

When reasoning at the differential byte levels, we have

$$\delta K_2[1][1] = \delta SK_1[3][2] \oplus \delta K_0[1][1]$$

where $\delta SK_1[3][2] = S(K_1[3][2]) \oplus S(K'_1[3][2])$. As S is a non linear operation, we cannot assume that $\delta SK_1[3][2] = S(\delta K_1[3][2])$. Therefore, $\delta SK_1[3][2]$ is a new differential byte. However, there is a finite number of such new differential bytes: for each round $i \in [0, r]$ and each line $k \in [0, 3]$, we introduce a new differential byte

$$\delta SK_i[3][k] = S(K_i[3][k]) \oplus S(K'_i[3][k])$$

and a new binary variable $\Delta SK_i[3][k]$ which is equal to 0 if $\delta SK_i[3][k] = 0^8$, and to 1 otherwise. Note that $\Delta SK_i[3][k]$ is a redundant variable which is equal to $\Delta K_i[3][k]$. So, we add the constraint

$$\forall i \in [1, r], \forall k \in [0, 3], \Delta K_i[3][k] = \Delta SK_i[3][k]$$

We introduce this redundant variable because at the byte level this equality no longer holds, *i.e.*, $\delta K_i[3][k] = A \not\Rightarrow \delta SK_i[3][k] = S(A)$ (because S is a non linear operator such that $S(A \oplus B) \neq S(A) \oplus S(B)$ except when $A = B$), and for the V sets defined below we reason at the byte level.

We propose to exploit the fact that each differential byte of K_i is the result of a xor between a finite set of bytes. We first use the KS rules defined in Section 2.1 to build, for each $i \in [1, r]$, and $j, k \in [0, 3]$, the set $V(i, j, k)$ of all differential bytes (coming either from δK_0 or from the set of new differential bytes δSK_i), such that:

$$\delta K_i[j][k] = \bigoplus_{\delta A \in V(i,j,k)} \delta A$$

For example, $V(1, 0, 0) = \{\delta K_0[0][0], \delta SK_0[3][1]\}$.

Note that these sets are computed before the search and do not depend on the initial values of plaintexts and keys.

For each of these sets, we introduce a set variable which contains the corresponding binary differential variables which are equal to 1:

$$V_1(i, j, k) = \{\Delta A \mid \delta A \in V(i, j, k) \wedge \Delta A = 1\}$$

For example, if $\Delta K_0[1][1] = 1$ and $\Delta SK_1[3][2] = 0$, then $V_1(2, 1, 1) = \{\Delta K_0[1][1]\}$.

Whenever two differential key bytes $\delta K_{i1}[j1][k1]$ and $\delta K_{i2}[j2][k2]$ have the same V_1 sets, then we may infer that $\delta K_{i1}[j1][k1] = \delta K_{i2}[j2][k2]$. More precisely, we define the constraint: $\forall i_1, i_2 \in [1, r], \forall j_1, j_2, k_1, k_2 \in [0, 3]$,

$$(V_1(i_1, j_1, k_1) = V_1(i_2, j_2, k_2)) \Rightarrow (EQ_{\delta K_{i_1}[j_1][k_1], \delta K_{i_2}[j_2][k_2]} = 1)$$

Also, if $V_1(i, j, k)$ is empty (resp. contains one or two elements), we infer that $\Delta K_i[j][k]$ is equal to 0 (resp. a variable, or a xor between 2 variables). More precisely, we define the constraints: $\forall i \in [1, r], \forall j, k \in [0, 3]$,

$$\begin{aligned} V_1(i, j, k) = \emptyset &\Rightarrow \Delta K_i[j][k] = 0 \\ V_1(i, j, k) = \{\Delta A\} &\Rightarrow \Delta K_i[j][k] = 1 \wedge EQ_{\delta K_i[j][k], \delta A} = 1 \\ V_1(i, j, k) = \{\Delta A, \Delta B\} &\Rightarrow xor(\Delta A, \Delta B, \Delta K_i[j][k]) \end{aligned}$$

From a practical point of view, V_1 variables are not modeled with set variables, but with vectors of boolean variables. The dimension of these vectors is equal to the number of possible elements in these sets, *i.e.*, $16 + 4(r + 1)$ (the 16 bytes of K_0 plus the four bytes that pass through an S-box at each round). Each boolean variable $V[p]$ is equal to 1 if the p^{th} element belongs to V_1 (*i.e.*, if the variable associated with the p^{th} element is equal to 1), and to 0 otherwise. For each of these vectors, we introduce an integer variable which is constrained to be equal to the sum of the variables of the vector.

5 CP model for Step 2

We have implemented in Choco 3 [19] the second step that, given a binary solution, searches for the byte-consistent solution with the highest p_2 value (or prove that there is no byte-consistent solution). The CP model for this second step is rather straightforward and mainly uses table constraints to define relations between the input and the output of the S-box function. The key point is to use a variable ordering that first chooses variables associated with the matrix ΔX_i such that $\sum_{j,k} \Delta X_i[j][k]$ is minimal.

The second step is not a bottleneck and is rather quickly solved by Choco. For example, when $l = 128$, it is solved in 0.41 (resp. 0.42 and 1.26) seconds, on average, when the number of rounds is $r = 3$ (resp. $r = 4$ and $r = 5$), whereas it is solved in 2.26 seconds when $l = 192$ and $r = 8$. Therefore, we have not tried to use other solvers for this step.

6 Experimental evaluation

In this section, we experimentally compare our two CP models for Step 1: Model 1 refers to the first model introduced in Section 3; Model 2 refers to the first model plus the additional constraints introduced in Section 4. These two models are defined with the MiniZinc modelling language [18]. Model 2 is available at $http://gerault.net/resources/CP\_AES.tar.gz$.

We compare three solvers on these models: Gecode [10] and Choco 4 [19], which are classical CP solvers, and Chuffed [4], which is a lazy clause hybrid solver that combines features of finite domain propagation and Boolean satisfiability. All solvers are run on a single core and with default parameters[6], except option `-f` for Choco 4 (to break ties of the heuristic described in Section 3.4 with the last conflict heuristic). All runs are limited to one hour of CPU time on a 2.5 - 3.5 GHz i7-4710MQ processor with 8 GB of memory.

Table 1 sums up the results for a number of rounds $r \in \{3, 4, 5\}$[7]. For each round, the objective value *obj* ranges from the largest value such that Model 1 finds no solution to the smallest value such that there exists a byte-consistent

[6] We tried other parameter settings. The best results were obtained with default ones.

[7] Let us recall that it has been shown in [3] that it is useless to try to solve the problem for more than 5 rounds when the key length is $l = 128$.

			Model 1							Model 2						
r	obj	S	bin	Gecode		Choco 4		Chuffed		bin	Gecode		Choco 4		Chuffed	
				Time	CP	Time	CP	Time	CP		Time	CP	Time	CP	Time	CP
3	2	0	0	**0.0**	9E1	**0.0**	4E1	**0.0**	5E1	0	**0.0**	9E1	0.1	4E1	**0.0**	5E1
3	3	0	5E2	0.1	2E3	0.4	2E3	**0.0**	7E2	0	**0.0**	3E2	0.3	2E2	0.1	2E2
3	4	0	5E3	1.3	2E4	1.8	1E4	**0.2**	5E3	0	**0.2**	9E2	0.5	4E2	**0.2**	4E2
3	5	2	2E4	6.0	6E4	5.1	5E4	**0.9**	2E4	4	**0.4**	2E3	0.6	1E3	0.6	1E3
4	8	0	0	**0.2**	2E4	0.6	1E4	0.3	8E3	0	**4.6**	1E4	4.9	5E3	6.2	4E3
4	9	0	2E4	7.1	1E5	5.4	7E4	**1.4**	4E4	0	8.1	2E4	**7.8**	8E3	10.7	7E3
4	10	0	6E6	-	-	1161.2	2E7	**113.5**	6E6	0	14.2	3E4	**12.8**	1E4	16.2	1E4
4	11	0	9E7	-	-	-	-	**1974.5**	9E7	0	24.4	5E4	**15.5**	2E4	25.2	2E4
4	12	2	-	-	-	-	-	-	-	8	44.7	1E5	**28.4**	5E4	35.7	3E4
5	10	0	0	**1.1**	1E5	1.4	5E4	2.3	4E4	0	39.2	3E4	**26.8**	2E4	37.3	1E4
5	11	0	3E1	**2.0**	2E5	2.4	1E5	5.0	7E4	0	63.0	5E4	**46.4**	3E4	61.5	2E4
5	12	0	5E5	998.0	4E6	98.3	2E6	**48.4**	7E5	0	110.0	9E4	**74.6**	5E4	97.9	3E4
5	13	0	4E7	-	-	-	-	**1246.5**	5E7	0	187.4	2E5	**142.1**	9E4	157.6	5E4
5	14	0	-	-	-	-	-	-	-	0	321.7	3E5	247.4	2E5	**246.5**	8E4
5	15	0	-	-	-	-	-	-	-	10	586.7	5E5	448.2	3E5	**408.1**	1E5
5	16	0	-	-	-	-	-	-	-	35	1175.8	1E6	770.1	6E5	**593.5**	2E5
5	17	6	-	-	-	-	-	-	-	50	2879.0	5E6	1524.9	1E6	**885.1**	4E5

Table 1. Comparison of models and solvers, on the enumeration problem. Each line displays: The number of rounds r, the objective function value obj, the number of byte-consistent binary solutions (S), and the results with models 1 and 2 (number of binary solutions (bin), CPU time in seconds (Time) and number of choice points (CP) for Gecode, Choco 4 and Chuffed). We report '-' when Time is greater than 3600.

binary solution. Table 1 shows us that Model 2 drastically reduces the number of byte-inconsistent solutions: For example, there are more than $9 * 10^7$ byte-inconsistent solutions with Model 1 when r=4 and obj=11, whereas there is no solution with Model 2. Hence, Model 2 is much more efficient than Model 1.

For both models, the number of choice points is greater for Gecode than for Choco, and for Choco than for Chuffed. However, choices points are handled faster by Gecode than by Choco (probably because Choco is implemented in Java and Gecode in C++), and faster by Choco than by Chuffed (probably due to lazy clause generation overhead). Therefore, Choco is not faster than Gecode on small instances, and Chuffed is not faster than Choco on small or medium-size instances. For the hardest instance (r=5; obj=17), Chuffed is nearly twice as fast as Choco, which is nearly twice as fast as Gecode.

All solvers are much faster than the Branch & Bound approach of [3]: Our C implementation of this approach needs 24 hours to find an optimal binary solution when $r = 4$. They are also faster and much less memory consuming than the approach of [9], that needs 60GB and 30 minutes on a 12-core computer to pre-compute the graph. For example, for $r = 5$ and $obj = 17$, Choco and Chuffed need 400MB and 88MB, respectively.

New results for differential cryptanalysis. We have found two byte-consistent binary solutions with $obj = 12$ for $r = 4$ rounds, and we have proven the optimality of these solutions by showing that there does not exist another byte-consistent

binary solution with an *obj* value strictly lower than 12. The optimal byte solution (computed in Step 2) when $obj = 12$ has a probability $p_2 = 2^{-79}$. The optimal byte solution and its associated binary solution are given in Appendix A. This solution is better than the solution claimed to be optimal in [3] and [9]: In these papers, authors say that the best byte-consistent binary solution for $r = 4$ has an *obj* value equal to 13, and that the optimal byte solution has a probability $p_2 = 2^{-81}$.

7 Discussion and Conclusion

We have introduced a CP model for solving a problem related to the chosen key differential cryptanalysis of AES with keys of length $l = 128$. This model follows the classical two step solving process of [3, 9]. In Step 1, we abstract bytes with binary values that indicate whether the byte is equal to 0^8. In Step 2, we search for non null byte values, for each binary value equal to 1. We have defined new constraints (not used in [3, 9]) which allow us to dramatically reduce the number of binary solutions that cannot be transformed into byte solutions. The idea is to keep track of equalities at the byte level to remove byte-inconsistent solutions at the binary level.

In this paper, we have described models for AES-128, with key length l=128. We have also defined MiniZinc models for AES-192, with l=192. At this time, the best solution we obtained for AES-192 concerns 8 rounds and has $obj = 19$ active S-boxes. We also plan to extend this work to other families of block ciphers, such as Rijndael [5] for which the approach of [9] cannot be used because of its exponential memory complexity.

In our model, we use boolean variables to represent equivalence classes defined by byte equalities: For each pair of bytes, we introduce a boolean variable which is set to 1 if the two bytes are equal, and we explicitly add constraints to ensure symmetry and transitivity of the equality relation. Another possibility would have been to use a graph variable (whose nodes are differential bytes, and edges are byte equality relations), and to post an n-clique global constraint on it, as proposed by Fages [7]: This constraint ensures that the graph is composed of n disjoint cliques, where each clique corresponds to an equivalence class. We have not used this constraint in our model, as it is not available in MiniZinc. We plan to investigate the interest of this constraint using Choco.

Finally, the CP model for Step 2 mainly uses table constraints. Some AES operations operate at the bit level (mostly xor operations), and we plan to improve our model by using bit-vector variables and channeling them with integer variables used to model bytes, as proposed in [13].

Acknowledgements. Many thanks to Jean-Guillaume Fages, for sending us Choco 4 before the official public release, and to Yves Deville, Pierre Schaus and François-Xavier Standaert for enriching discussions on this work.

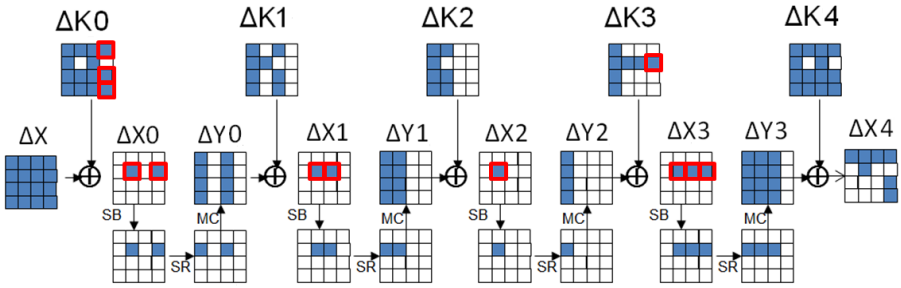
References

1. Eli Biham. New types of cryptoanalytic attacks using related keys (extended abstract). In *Advances in Cryptology - EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 398–409. Springer, 1993.
2. Eli Biham and Adi Shamir. Differential cryptoanalysis of feal and n-hash. In *Advances in Cryptology - EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1991.
3. Alex Biryukov and Ivica Nikolic. Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to aes, camellia, khazad and others. In *Advances in Cryptology - EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 322–344. Springer, 2010.
4. Geoffrey Chu and Peter J. Stuckey. Chuffed solver description, 2014. Available at http://www.minizinc.org/challenge2014/description_chuffed.txt.
5. J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer-Verlag, 2002.
6. Debapratim De, Abishek Kumarasubramanian, and Ramarathnam Venkatesan. Inversion attacks on secure hash functions using satsolvers. In *Theory and Applications of Satisfiability Testing - SAT 2007*, volume 4501 of *Lecture Notes in Computer Science*, pages 377–382. Springer, 2007.
7. Jean-Guillaume Fages. On the use of graphs within constraint-programming. *Constraints*, 20(4):498–499, 2015.
8. FIPS 197. Advanced Encryption Standard. Federal Information Processing Standards Publication 197, 2001. U.S. Department of Commerce/N.I.S.T.
9. Pierre-Alain Fouque, Jérémy Jean, and Thomas Peyrin. Structural evaluation of aes and chosen-key distinguisher of 9-round aes-128. In *Advances in Cryptology - CRYPTO 2013*, volume 8042 of *Lecture Notes in Computer Science*, pages 183–203. Springer, 2013.
10. Gecode Team. Gecode: Generic constraint development environment, 2006. Available from `http://www.gecode.org`.
11. Pierre Karpman, Thomas Peyrin, and Marc Stevens. Practical free-start collision attacks on 76-step SHA-1. *IACR Cryptology ePrint Archive*, 2015:530, 2015.
12. Florian Legendre, Gilles Dequen, and Michaël Krajecki. Encoding hash functions as a sat problem. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012*, pages 916–921. IEEE, 2012.
13. Laurent D. Michel and Pascal Van Hentenryck. Constraint satisfaction over bit-vectors. In *Principles and Practice of Constraint Programming - 18th International Conference (CP 2012)*, volume 7514 of *Lecture Notes in Computer Science*, pages 527–543. Springer, 2012.
14. Marine Minier, Christine Solnon, and Julia Reboul. Solving a Symmetric Key Cryptographic Problemwith Constraint Programming, July 2014. ModRef 2014, Workshop of the CP 2014 Conference, September 2014, Lyon, France.
15. Ilya Mironov and Lintao Zhang. Applications of sat solvers to cryptanalysis of hash functions. In *Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121 of *Lecture Notes in Computer Science*, pages 102–115. Springer, 2006.
16. Pawel Morawiecki and Marian Srebrny. A sat-based preimage analysis of reduced keccak hash functions. *Inf. Process. Lett.*, 113(10-11):392–397, 2013.
17. Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In *Information Security and Cryptology - 7th International Conference, Inscrypt 2011*, volume 7537 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011.

18. Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, CP'07, pages 529–543. Springer-Verlag, 2007.
19. Charles Prudhomme and Jean-Guillaume Fages. An introduction to choco 3.0: an open source java constraint programming library. In *CP Workshop on "CP Solvers: Modeling, Applications, Integration, and Standardization"*, 2013.
20. Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending sat solvers to cryptographic problems. In *Theory and Applications of Satisfiability Testing - SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.
21. Siwei Sun, Lei Hu, Meiqin Wang, Qianqian Yang, Kexin Qiao, Xiaoshuang Ma, Ling Song, and Jinyong Shan. Extending the applicability of the mixed-integer programming technique in automatic differential cryptanalysis. In *Information Security - 18th International Conference, ISC 2015*, volume 9290 of *Lecture Notes in Computer Science*, pages 141–157. Springer, 2015.
22. 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 *Advances in Cryptology - ASIACRYPT 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 158–178. Springer, 2014.
23. Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.
24. Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on sha-0. In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005.

A Solution with $obj = 12$ active S-boxes for AES with $r = 4$ rounds and $l = 128$ bits

The byte-consistent binary solution is displayed below. Each binary variable assigned to 1 is colored in blue, and is surrounded in red when it belongs to the objective function (*i.e.*, it passes through an S-box).



The optimal byte solution is displayed below, in hexadecimal notation.

Round	$\delta X = X \oplus X'$	$\delta K = K \oplus K'$
Init.	0d151846 0dacf2f2 0dfff2f2 0dacf2f2	
0	00000000 00ac0000 00000000 00ac0000	0d151846 0d00f2f2 0dfff2f2 0d00f2f2
1	00000000 00ff0000 00ff0000 00000000	0dfff2f2 00ff0000 0dfff2f2 00000000
2	00000000 00ff0000 00000000 00000000	0dfff2f2 0d00f2f2 00000000 00000000
3	00000000 00ff0000 00ff0000 00ff0000	0dfff2f2 00ff0000 00ff0000 00ff0000
End/4	fa000000 faff0000 fa000000 f700f2f2	f7fff2f2 f700f2f2 f7fff2f2 f700f2f2

The corresponding plaintexts X and X' and keys K and K' are displayed below, in hexadecimal notation. The probability p_2 associated with these plaintexts and keys is $p_2 = 2^{-79}$ whereas it is equal to $p_2 = 2^{-81}$ in the solution given in [3] and [9] (solution with $obj = 13$ active S-boxes).

Round	K	K'
0	00000000 00000000 00000000 00000000	0d151846 0d00f2f2 0dfff2f2 0d00f2f2
1	62636363 62636363 62636363 62636363	6f9c9191 629C6363 6f639191 62636363
2	9b9898c9 f9fbfbaa 9b9898c9 f9fbfbaa	96676a3b f4fb0958 9b9898c9 f9fbfbaa
3	90973450 696ccffa f2f45733 0b0fac99	9d68c6a2 6993cffa f2b5733 0bf0ac99
4	ee60da7b 876a1581 759e42b2 7e91ee2b	19f92889 706ae773 8261b040 89911cd9

Round	X	X'
Init.	6b291f8d a800d3d7 f239d5a4 510035ef	663c07cb a5ac2125 ffc62756 5cacc71d
0	6b291f8d a800d3d7 f239d5a4 510035ef	6b291f8d a8acd3d7 f239d5a4 51ac35ef
1	e5000327 00796300 0079005c 0000005a	e5000327 00866300 0086005c 0000005a
2	2e2de80b 5186a759 e0d3cbb2 2b02c803	2e2de80b 5179a759 e0d3cbb2 2b02c803
3	5a74f2ae b979ce4a e286aa6a ea86647b	5a74f2ae b986ce4a e279aa6a ea79647b
End	c501f2fa 4095b6af cdd8f67b 4fadf0a4	3f01f2fa ba6ab6af 37d8f67b b8ad0256