# Using Constraint Programming to solve a Cryptanalytic Problem

David Gerault, Marine Minier, Christine Solnon

# Using Constraint Programming to solve a Cryptanalytic Problem*

**David Gerault**(1), **Marine Minier**(2), **and Christine Solnon**(3)
(1) LIMOS, Clermont-Ferrand, France
(2) Université de Lorraine, LORIA, UMR 7503, F-54506, France
(3) Université de Lyon, INSA-Lyon, F-69621, France, LIRIS, CNRS UMR5205

## Abstract

We describe Constraint Programming (CP) models to solve a cryptanalytic problem: the chosen key differential attack against the standard block cipher AES. We show that CP solvers are able to solve these problems quicker than dedicated cryptanalysis tools, and we prove that a solution claimed to be optimal in two recent cryptanalysis papers is not optimal by providing a better solution.

## 1 Introduction

Since 2001, AES (Advanced Encryption Standard) is the encryption standard for block ciphers [FIPS 197, 2001]. It guarantees communication confidentiality by using a secret key $K$ to encode an original plaintext $X$ into a ciphertext $AES_K(X)$, in such a way that the ciphertext can further be decoded into the original one using the same key, *i.e.*, $X = AES_K^{-1}(AES_K(X))$. Cryptanalysis aims at testing whether confidentiality is actually guaranteed. In particular, differential cryptanalysis [Biham and Shamir, 1991] evaluates whether it is possible to find the key within a reasonable number of trials by considering plaintext pairs $(X, X')$ and studying the propagation of the initial difference $X \oplus X'$ between $X$ and $X'$ while going through the ciphering process (where $\oplus$ is the xor operator). Today, differential cryptanalysis is public knowledge, and block ciphers such as AES have proven bounds against differential attacks. Hence, [Biham, 1993] proposed a new type of attack called related-key attack 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 mount related-key attacks, the cryptanalist must find optimal related-key differentials, *i.e.*, a plaintext difference $\delta X$ and a key difference $\delta K$ that maximize the probability that, for randomly chosen plaintext $X$ and key $K$, the difference $X \oplus \delta X$ in the input plaintexts becomes the difference $AES_K(X) \oplus AES_{K \oplus \delta K}(X')$ in the output ciphertexts. Finding the optimal related-key differentials for AES is a highly combinatorial problem that hardly scales. Two main approaches have been proposed to solve this problem: a graph traversal approach [Fouque *et al.*, 2013], and a Branch & Bound approach [Biryukov and Nikolic, 2010]. The approach of [Fouque *et al.*, 2013] requires about 60 GB of memory when the key has 128 bits, and it has not been extended to larger keys. The approach of [Biryukov and Nikolic, 2010] only takes several megabytes of memory, but it requires several days of computation when the key has 128 bits, and several weeks when the key has 192 bits.

During the process of designing new ciphers, this search generally needs to be performed several times, so it is desirable that it can be done rather quickly. Another point that should not be neglected is the time needed to design and implement these approaches: To ensure that the computation is completed within a "reasonable" amount of time, it is necessary to reduce the branching by introducing clever reasoning. Of course, this hard task is also likely to introduce bugs, and checking the correctness or the optimality of the computed solutions may not be so easy. Finally, reproducibility may also be an issue. Other researchers may want to adapt these algorithms to other problems, with some common features but also some differences, and this may again be very difficult and time-consuming.

In [Gerault *et al.*, 2016], we proposed to use Constraint Programming (CP) to solve this problem. When using CP to solve a problem, one simply has to model the problem by means of constraints. Then, this model is solved by generic solvers usually based on a *Branch & Propagate* approach: The search space is explored by building a search tree, and constraints are propagated at each tree node in order to prune branches. CP opens new perspectives for this kind of cryptanalysis problems. First, it is very competitive with dedicated approaches: When the key has 128 bits, Chuffed [Chu and Stuckey, 2014] is able to find optimal solutions in less than one hour. Second, the CP model is easier to check or re-use than a full program that not only describes the problem to solve, but also how to solve it. Actually, CP allowed us to prove that a solution claimed to be optimal in [Biryukov and Nikolic, 2010; Fouque *et al.*, 2013] is not optimal by providing a better solution.

## 2 Problem Statement

**AES block cipher.** AES ciphers blocks of length $n = 128$ bits, and each block is a $4 \times 4$ matrix of bytes. The length of keys is $l \in \{128, 192, 256\}$. In this paper, we only consider

*This research was conducted with the support of the FEDER program of 2014-2020 and the region council of Auvergne

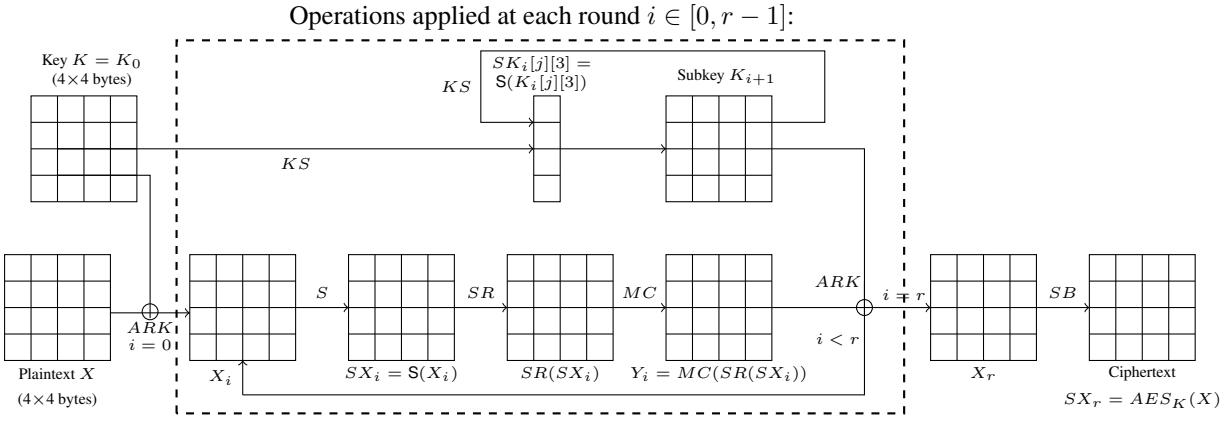


Figure 1: AES ciphering process. Each $4 \times 4$ array represents a group of 16 bytes. Before the first round, $ARK$ is applied on $X$ and $K$ to obtain $X_0$. Then, for each round $i \in [0, r-1]$, $\mathsf{S}$ is applied on $X_i$ to obtain $SX_i$, $SR$ and $MC$ are applied on $SX_i$ to obtain $Y_i$, KS is applied on $K_i$ to obtain $K_{i+1}$ (and during $KS$, $\mathsf{S}$ is applied on $K_i[j][3]$ to obtain $SK_i[j][3]$, $\forall j \in [0, 3]$), and $ARK$ is applied on $K_{i+1}$ and $Y_i$ to obtain $X_{i+1}$. The ciphertext $SX_r$ is obtained by applying $SB$ on $X_r$.

keys of length $l = 128$, and keys are $4 \times 4$ matrices of bytes. Given a $4 \times 4$ matrix of bytes $M$, we note $M[j][k]$ the byte at row $j \in [0, 3]$ and column $k \in [0, 3]$.

AES is an iterative process, and we note $X_i$ the ciphertext at the beginning of round $i \in [0, r]$. Each round is composed of the following operations, as displayed in Fig. 1:

- `SubBytes` $S$. $S$ is a non-linear permutation which is applied on each byte of $X_i$ separately, *i.e.*, $\forall j, k \in [0, 3]$, $X_i[j][k]$ is replaced by $\mathsf{S}(X_i[j][k])$, according to a lookup table. We note $SX_i = \mathsf{S}(X_i)$.
- `ShiftRows` $SR$. $SR$ is a linear mapping that rotates on the left by 1 byte position (resp. 2 and 3 byte positions) the second row (resp. third and fourth rows) of $SX_i$.
- `MixColumns` $MC$. $MC$ is a linear mapping that multiplies each column of $SR(SX_i)$ by a $4 \times 4$ fixed matrix chosen for its good properties of diffusion [Daemen and Rijmen, 2002]. In particular, it has the Maximum Distance Separable (MDS) property: For each column, the total number of bytes which are different from 0, before and after applying $MC$, is either equal to 0 or strictly greater than 4. We note $Y_i = MC(SR(SX_i))$.
- `KeySchedule` $KS$. The subkey at round 0 is the initial key, *i.e.*, $K_0 = K$. For each round $i \in [0, r-1]$, the subkey $K_{i+1}$ is generated from $K_i$ by applying $KS$. It first replaces each byte $K_i[j][3]$ of the last column by $\mathsf{S}(K_i[j][3])$ (where $\mathsf{S}$ is the `SubBytes` operator), and we note $SK_i[j][3] = \mathsf{S}(K_i[j][3])$. Then, each column of $K_{i+1}$ is obtained by performing a xor operation between bytes coming from $K_i$, $SK_i$, or $K_{i+1}$.
- `AddRoundKey` $ARK$. $ARK$ performs a xor operation between bytes of $Y_i$ and subkey $K_{i+1}$ to obtain $X_{i+1}$.

Let us note *Bytes* the set of all bytes (for all rounds), *i.e.*,

$$\begin{aligned} \textit{Bytes} = \quad & \{X[j][k], X_i[j][k], SX_i[j][k], Y_i[j][k], \\ & K_i[j][k], SK_i[j][3] \mid i \in [0, r], j, k \in [0, 3]\} \end{aligned}$$

**Optimal related-key differentials.** Let us note $\delta B = B \oplus B'$ the difference between two bytes $B$ and $B'$, and $\textit{diffBytes} = \{\delta B \mid B \in \textit{Bytes}\}$ the set of all differential bytes.

Mounting attacks requires finding a related-key differential characteristic, *i.e.* a plaintext difference $\delta X = X \oplus X'$ and a key difference $\delta K = K \oplus K'$, such that $\delta X$ becomes $\delta SX_r$ after $r$ rounds with a probability as hight as possible. This can be achieved by tracking the propagation of the initial differences through the cipher. Once such an optimal differential characteristic is found, the cryptanalist asks for the encryption of messages satisfying the input difference. When this difference propagates as expected, he can infer informations leading to a recovery of the secret key K with less workload than exhaustive search. The difficulty of recovering K decreases as the probability of the used differential characteristic increases. The AES operators $SR, MC, ARK$, and $KS$ are linear, *i.e.*, they propagate differences in a deterministic way (with probability 1). However, the $\mathsf{S}$ operator is not linear: Given a byte difference $B \oplus B' = \delta B$, the probability that $\delta B$ becomes $\mathsf{S}(B) \oplus \mathsf{S}(B') = \delta SB$ is $Pr(\delta B \rightarrow \delta SB)$. It is equal to 1 if $\delta B = 0$ (*i.e.*, $B = B'$). However, if $\delta B \neq 0$, then $Pr(\delta B \rightarrow \delta SB) \in \{\frac{2}{256}, \frac{4}{256}\}$.

The probability that $\delta X$ becomes $\delta SX_r$ is equal to the product of all $Pr(\delta B \rightarrow \delta SB)$ such that $\delta B$ (resp. $\delta SB$) is a byte difference before (resp. after) passing through the $\mathsf{S}$ operator when ciphering $X$ with $K$ and $X'$ with $K'$, *i.e.*, the product of all $Pr(\delta X_i[j][k] \rightarrow \delta SX_i[j][k])$ and all $Pr(\delta K_i[j][3] \rightarrow \delta SK_i[j][3])$ with $i \in [0, r]$ and $j, k \in [0, 3]$. The goal is to find $\delta X$ and $\delta K$ that maximize this probability.

**Two step solving process.** To find $\delta X$ and $\delta K$, we search for the values of all differential bytes in *diffBytes*. Both [Biryukov and Nikolic, 2010] and [Fouque *et al.*, 2013] propose to solve this problem in two steps. In Step 1, a Boolean variable $\Delta B$ is associated with every differential byte $\delta B \in \textit{diffBytes}$ such that $\Delta B = 0 \Leftrightarrow \delta B = 0$ and

$\Delta B = 1 \Leftrightarrow \delta B \in [1, 255]$. The goal of Step 1 is to find a *Boolean solution* that assigns values to Boolean variables such that the AES transformation rules are satisfied. During this first step, the `SubBytes` operation $S$ is not considered. Indeed, it does not introduce nor remove differences. Therefore, we have $\Delta X_i[j][k] = \Delta SX_i[j][k]$ and $\Delta K_i[j][3] = \Delta SK_i[j][3]$. As we search for a solution with maximal probability, the goal of Step 1 is to search for a Boolean solution which minimizes the number of variables $\Delta X_i[j][k]$ and $\Delta K_i[j][3]$ which are set to 1.

In Step 2, the Boolean solution is transformed into a *byte solution*: For each differential byte $\delta B \in \mathit{diffBytes}$, if the corresponding Boolean variable $\Delta B$ is assigned to 0, then $\delta B$ is also assigned to 0; otherwise, we search for a byte value in $[1, 255]$ to be assigned to $\delta B$ such that the AES transformation rules are satisfied and the probability is maximized. Note that some Boolean solutions may not be transformable into byte solutions. These Boolean solutions are said to be *byte-inconsistent*.

## 3 Basic CP Model for Step 1

A first CP model for Step 1 may be derived from the AES transformation rules in a rather straightforward way. A CP model is defined by a set of variables, such that each variable $x$ has a domain $D(x)$, and a set of constraints, *i.e.*, relations that restrict the values that may be simultaneously assigned to the variables.

**Variables.** For each differential byte $\delta B \in \mathit{diffBytes}$, we define a Boolean variable $\Delta B$ whose domain is $D(\Delta B) = \{0, 1\}$: it is assigned to 0 if $\delta B = 0$, and to 1 otherwise.

**`XOR` constraint.** We first define a `XOR` constraint for $ARK$ and $KS$. Let us consider three differential bytes $\delta A$, $\delta B$ and $\delta C$ such that $\delta A \oplus \delta B = \delta C$. If $\delta A = \delta B = 0$, then $\delta C = 0$. If ($\delta A = 0$ and $\delta B \neq 0$) or ($\delta A \neq 0$ and $\delta B = 0$) then $\delta C \neq 0$. However, if $\delta A \neq 0$ and $\delta B \neq 0$, then we cannot know if $\delta C$ is equal to 0 or not: This depends on whether $\delta A = \delta B$ or not. When abstracting differential bytes $\delta A$, $\delta B$ and $\delta C$ with Boolean variables $\Delta A$, $\Delta B$ and $\Delta C$ (which only model the fact that there is a difference or not), we obtain the following definition of the `XOR` constraint:
$\texttt{XOR}(\Delta A, \Delta B, \Delta C) \Leftrightarrow \Delta A + \Delta B + \Delta C \neq 1$.

**`AddRoundKey` and `KeySchedule` constraints.** Both $ARK$ and $KS$ are modeled with `XOR` constraints between $\Delta X_i$, $\Delta Y_i$, and $\Delta K_i$ variables (for $ARK$), and between $\Delta K_i$ and $\Delta SK_i$ variables (for $KS$).

**`ShiftRows` and `MixColumns`.** $SR$ simply shift variables. The MDS property of $MC$ is ensured by posting a constraint on the sum of some variables of $\Delta X_i$ and $\Delta Y_i$ variables, which must belong to the set $\{0, 5, 6, 7, 8\}$.

**Objective function.** We introduce an integer variable $obj_{Step1}$ that must be minimized, and we post an equality constraint between $obj_{Step1}$ and the sum of Boolean variables on which a non linear S operation is performed (all variables $\Delta X_i[j][k]$ and $\Delta K_i[j][3]$ with $i \in [0, r]$ and $j, k \in [0, 3]$).

Let $v$ be the optimal value of $obj_{Step1}$. It may happen that none of the Boolean solutions with $obj_{Step1} = v$ is byte-consistent, or that the maximal probability $p$ of Boolean solutions with $obj_{Step1} = v$ is such that it is possible to have a better probability with a larger value for $obj_{Step1}$ (*i.e.*, $p < (\frac{4}{256})^{v+1}$). In this case, we need to search for new Boolean solutions, such that $obj_{Step1}$ is minimal while being strictly greater than $v$. This is done by adding the constraint $obj_{Step1} > v$ before solving again Step 1.

**Limitations of the basic CP model.** This basic CP model is complete, *i.e.*, for any solution at the byte level (on $\delta$ variables), there exists a solution at the Boolean level (on $\Delta$ variables). However, preliminary experiments have shown us that there is a huge number of Boolean solutions which are byte inconsistent. For example, when the number of rounds is $r = 4$, the optimal cost is $obj_{Step1} = 11$, and there are more than 90 millions of Boolean solutions with $obj_{Step1} = 11$. However, none of these solutions is byte-consistent. In this case, most of the Step 1 solving time is spent at generating useless Boolean solutions which are discarded in Step 2.

## 4 Additional constraints for step 1

When reasoning at the Boolean level, many solutions are not byte-consistent because constraints at the byte level have been ignored. To propagate some properties at the byte level, we introduce new variables and constraints that model equality relations between differential bytes.

**New equality variables.** For each couple of differential bytes $(\delta A, \delta B) \in \mathit{diffBytes}^2$, we introduce a Boolean equality variable $EQ_{\delta A, \delta B}$ which is equal to 1 if $\delta A = \delta B$, and to 0 otherwise. These variables are constrained to define an equivalence relation by adding a symmetry constraint ($EQ_{\delta A, \delta B} = EQ_{\delta_B, \delta_A}$) and a transitivity constraint (if $EQ_{\delta A, \delta B} = EQ_{\delta B, \delta C} = 1$ then $EQ_{\delta_A, \delta_C} = 1$). Also, $EQ$ variables are related to $\Delta$ variables by adding the constraints:

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

**Revisiting the `XOR` constraint.** When defining the constraint $\texttt{XOR}(\Delta A, \Delta B, \Delta C)$, if $\Delta A = \Delta B = 1$, then we cannot know if $\Delta C$ is equal to 0 or 1. However, whenever $\Delta C = 0$ (resp. $\Delta C = 1$), we know for sure that the corresponding byte $\delta C$ is equal to 0 (resp. different from 0), meaning that the two bytes $\delta A$ and $\delta B$ 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 $\Delta A$ and $\Delta B$ 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:

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

**Propagation of MDS at the byte level.** The MDS property ensures that, for each column, the total number of bytes which are different from 0, before and after applying $MC$, is either equal to 0 or strictly greater than 4. This property also holds for any xor difference between two different columns of $X$ and $Y$ matrices. To propagate this property, for each pair of columns in $X$ and $Y$ matrices, we add a constraint on the sum of equality variables between bytes of these columns.

**Constraints derived from KS.** The `KeySchedule` mainly performs xor and S operations. As a consequence, each byte $\delta K_i[j][k]$ may be expressed as a xor between bytes of the original key difference $\delta K_0$, and bytes of $\delta SK_{i-1}$ (which are differences of key bytes that have passed through S during the previous round). Hence, for each byte $\delta K_i[j][k]$, we precompute the set $V(i,j,k)$ such that $V(i,j,k)$ only contains bytes of $\delta K_0$ and $\delta SK_{i-1}$ and $\delta K_i[j][k] = \bigoplus_{\delta A \in V(i,j,k)} \delta A$. For each set $V(i,j,k)$, we introduce a set variable $V_1(i,j,k)$ which is constrained to contain the subset of $V(i,j,k)$ corresponding to the Boolean variables equal to 1. We use these set variables to infer that two differential key bytes that have the same $V_1$ set are equal. 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).

## 5 CP model for Step 2

Given a Boolean solution for Step 1, Step 2 aims at searching for the byte-consistent solution with the highest probability (or proving that there is no byte-consistent solution). Hence, for each Boolean variable $\Delta A$ of Step 1, we define an integer variable $\delta A$ whose domain depends on the value of $\Delta A$: If $\Delta A = 0$, then $D(\delta A) = \{0\}$ (*i.e.*, $\delta A$ is also assigned to 0); otherwise, $D(\delta A) = [1, 255]$. As we look for a byte-consistent solution with maximal probability, we also add an integer variable $P_A$ for each byte $A$ that passes through the S operator, *i.e.*, $X_i[j][k]$ and $K_i[j][3]$: This variable corresponds to the base 2 logarithm of the probability $\Pr(\delta A \rightarrow \delta S_A)$ of obtaining the S output difference $\delta S_A$ when the S input difference is $\delta A$. The domain of these variables depends on the value of $\Delta A$ in the Step 1 solution: If $\Delta A = 0$, then $\Pr(0 \rightarrow 0) = 1$ and therefore $D(P_A) = \{0\}$; otherwise, $\Pr(\delta A \rightarrow \delta S_A) \in \{\frac{2}{256}, \frac{4}{256}\}$ and $D(P_A) = \{-7, -6\}$.

The constraints basically follow the AES operations to relate variables, as described for Step 1, but consider the definition of the operations at the byte level, instead of the Boolean level. A main difference is that the `SubBytes` operation, which has no effect at the Boolean level, must be modeled at the byte level. This is done thanks to a ternary table constraint which extensively lists all triples $(X, Y, P)$ such that there exist two bytes $B_1$ and $B_2$ whose difference before and after passing through S is equal to $X$ and $Y$, respectively, and such that $P$ is the base 2 logarithm of the probability of this transformation. The goal is to find a byte-consistent solution with maximal differential probability. As we consider logarithms, this amounts to searching for a solution that maximizes the sum of all $P_A$ variables.

## 6 Results and Conclusion

The CP model for Step 1 was implemented with the MiniZinc modeling language [Nethercote *et al.*, 2007], and we compared three CP solvers: Gecode [Gecode Team, 2006], Choco 4 [Prudhomme and Fages, 2013], and Chuffed [Chu and Stuckey, 2014]. The best results were obtained with Chuffed. The basic CP model (described in Section 3) does not scale well because it generates a huge number of Boolean solutions which are not byte-consistent. When adding the additional constraints described in Section 4, most of these byte-inconsistent solutions are filtered out, and Chuffed is able to solve all instances in less than one hour. The CP model for Step 2 was implemented with Choco 3. It solves all instances (*i.e.*, finds an optimal byte-consistent solution for each Boolean solution of Step 1 when it exists) in a few seconds. As a conclusion, CP solvers are much faster than the Branch & Bound approach of [Biryukov and Nikolic, 2010], which needs several days to solve these instances. It is also faster and much less memory consuming than the approach of [Fouque *et al.*, 2013], that needs 60GB and 30 minutes on a 12-core computer to pre-compute the graph.

**New results for differential cryptanalysis.** For $r = 4$ rounds, we have found a byte-consistent solution with $obj_{Step1} = 12$ and a probability equal to $2^{-79}$. This solution is better than the solution claimed to be optimal in [Biryukov and Nikolic, 2010] and [Fouque *et al.*, 2013]: In these papers, the authors say that the best byte-consistent solution has $obj_{Step1} = 13$, and a probability equal to $2^{-81}$.

We have shown how to extend our CP models to AES with longer keys (such that $l \in \{192, 256\}$) in [Gérault *et al.*, 2017]. These models allowed us to find optimal solutions for all possible instances of AES with $l \in \{128, 192, 256\}$ in less than 35 hours for Step 1, and in less than 6 minutes for Step 2. This is a clear improvement with respect to existing work as the approach of [Fouque *et al.*, 2013] cannot be extended to $l > 128$ due to its memory complexity, and the approach of [Biryukov and Nikolic, 2010] needs several weeks to solve instances with $l = 192$. Furthermore, we have shown that the solution proposed in [Biryukov and Nikolic, 2010] for $l = 192$ and $r = 11$ is inconsistent. We have also found better solutions when $l = 256$, and we have computed the actual optimal solution for AES with $l = 256$. Its probability is $2^{-146}$ (instead of $2^{-154}$ for the solution of [Biryukov *et al.*, 2009]). Using this solution, we improved the related-key distinguisher and the basic related-key differential attack on the full AES-256 by a factor $2^6$ and the $q$-multicollisions by a factor 2 (see [Gérault *et al.*, 2017] for more details).

These cryptanalysis problems open new and exciting challenges for the CP community. In particular, these problems are not easy to model. More precisely, naive CP models such as the one described in Section 3 may not scale well. The introduction of equality constraints at the byte level drastically improves the solving process, but these constraints are not straightforward to find and implement. Hence, a challenge is to define new CP frameworks, dedicated to this kind of cryptanalysis problems, in order to ease the development of efficient CP models for these problems.

## References


[Biham and Shamir, 1991] 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.

[Biham, 1993] 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.

[Biryukov and Nikolic, 2010] 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.

[Biryukov *et al.*, 2009] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. Distinguisher and related-key attack on the full AES-256. In *Advances in Cryptology - CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 231–249. Springer, 2009.

[Chu and Stuckey, 2014] Geoffrey Chu and Peter J. Stuckey. Chuffed solver description, 2014. Available at `http://www.minizinc.org/challenge2014/description\_chuffed.txt`.

[Daemen and Rijmen, 2002] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer-Verlag, 2002.

[FIPS 197, 2001] FIPS 197. Advanced Encryption Standard. Federal Information Processing Standards Publication 197, 2001. U.S. Department of Commerce/N.I.S.T.

[Fouque *et al.*, 2013] 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.

[Gecode Team, 2006] Gecode Team. Gecode: Generic constraint development environment, 2006. Available from `http://www.gecode.org`.

[Gerault *et al.*, 2016] David Gerault, Marine Minier, and Christine Solnon. Constraint programming models for chosen key differential cryptanalysis. In *Principles and Practice of Constraint Programming - CP 2016*, volume 9892 of *Lecture Notes in Computer Science*, pages 584–601. Springer, 2016.

[Gérault *et al.*, 2017] David Gérault, Pascal Lafourcade, Marine Minier, and Christine Solnon. Revisiting aes related-key differential attacks with constraint programming. Cryptology ePrint Archive, Report 2017/139, 2017. `http://eprint.iacr.org/2017/139`.

[Nethercote *et al.*, 2007] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming - CP 2007*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007.

[Prudhomme and Fages, 2013] 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.