Combining Solvers to Solve a Cryptanalytic Problem*

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

[1] Université Clermont Auvergne, LIMOS, France
[2] Université de Lorraine, LORIA, UMR 7503, F-54506, France
[3] Université de Lyon, INSA-Lyon, LIRIS, CNRS UMR5205, F-69621, France

Abstract. We describe Constraint Programming models to solve a cryptanalytic problem: the chosen key differential attack against the standard block cipher AES. We more particularly show how combining two solvers, namely Picat_Sat and Chuffed, greatly speeds-up the solution process and allows us to solve all instances in less than twelve hours while dedicated cryptanalysis tools need weeks.

1 Introduction

Since 2001, AES (Advanced Encryption Standard) is the encryption standard for block ciphers [7]. It guarantees communication confidentiality by using a secret key K to cipher an original plaintext X into a ciphertext $AES_K(X)$, in such a way that the ciphertext can further be deciphered into the original one using the same key, *i.e.*, $X = AES_K^{-1}(AES_K(X))$. It uses either 128-, 192-, or 256-bit keys. Cryptanalysis aims at testing whether confidentiality is actually guaranteed. In particular, differential cryptanalysis [2] 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 $\delta X = X \oplus X'$ 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, [1] proposed a new type of attack called related-key attack that allows an attacker to also inject differences between the keys K and K' (even if the secret key K remains unknown from the attacker).

To mount related-key attacks, the cryptanalyst must find related-key differentials, defined as a plaintext difference δX, a key difference δK, and a ciphertext difference δC. The optimal ones maximize the probability that, for random plaintexts X and key K, an input difference $(\delta X, \delta K)$ leads to the output difference δC, *i.e.*, it holds that $AES_K(X) \oplus AES_{K \oplus \delta K}(X \oplus \delta X) = \delta C$. 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 [8], and a Branch & Bound approach [4]. The approach of [8] requires about 60 GB of memory when the key has 128 bits, and it has not been extended to larger keys. The approach of [4] 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.

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

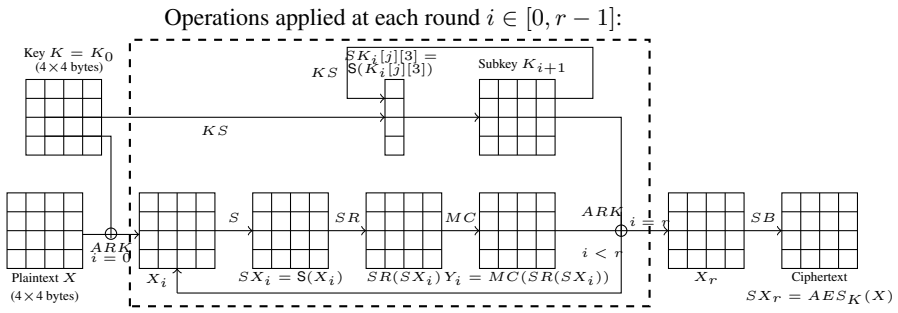

Fig. 1. AES-128 cipher. Each 4×4 array represents a group of 16 bytes. Before the first round, ARK is applied on X and K to obtain X_0. Then, for each round $i \in [0, r-1]$, S is applied on X_i to obtain SX_i, SR and MC are applied on SX_i to obtain Y_i, KS is applied on K_i to obtain K_{i+1} (and during KS, 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.

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.

In [11], we introduced a first Constraint Programming (CP) model to solve this problem when the key has 128 bits. In [10], we extended this model to larger keys of 192 and 256 bits, and we improved it by introducing a new way for modeling byte equivalence classes. We also showed how to speed-up the solution process by combining two solvers: Picat_Sat [16] and Chuffed [5]. In this paper, we first describe the problem in Section 2, and the CP model of [10] in Section 3. In these two sections, we only consider the case where keys have 128 bits, as this simplifies the presentation. The extension to the case where keys have 192 or 256 bits is quite straightforward and we refer the reader to [10] for more details. Then, we study the solver combination more thoroughly in Section 4.

2 Problem Statement

AES block cipher. AES ciphers blocks of length $n = 128$ bits, and each block is a 4×4 matrix of bytes. The length of keys is $l \in \{128, 192, 256\}$. In this section, we only consider keys of length $l = 128$, and keys are 4×4 matrices of bytes. Given a 4×4 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) 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) 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) is a linear mapping that multiplies each column of $SR(SX_i)$ by a 4×4 fixed matrix chosen for its good properties of diffusion [6]. 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) is the operation that generates subkeys. 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 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) performs a xor between bytes of Y_i and subkey K_{i+1} to obtain X_{i+1}.

The set of all bytes (for all rounds) is denoted *Bytes* (*i.e.*, $Bytes = \{X[j][k], X_i[j][k], SX_i[j][k], Y_i[j][k], K_i[j][k], SK_i[j][3] \mid i \in [0, r], j, k \in [0, 3]\}$). The difference between two bytes B and B' is denoted δB (*i.e.*, $\delta B = B \oplus B'$), and the set of all differential bytes is denoted *diffBytes* (*i.e.*, $diffBytes = \{\delta B \mid B \in Bytes\}$).

Optimal related-key differentials. Mounting attacks to recover the key K 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 δX becomes δSX_r after r rounds with a probability $Pr(\delta X \rightarrow \delta SX_r)$ as high as possible. This can be achieved by tracking the propagation of the initial differences through the cipher using known propagation rules. Once such an optimal differential characteristic is found, the cryptanalyst asks for the encryption of plaintext pairs (X, X') satisfying the input difference (*i.e.*, $X \oplus X' = \delta X$), with key K for X and key $K' = K \oplus \delta K$ for $\delta X'$. When this difference propagates as expected, he can infer informations leading to a recovery of the secret key K in $\mathcal{O}(\frac{1}{Pr(\delta X \rightarrow \delta SX_r)})$ trials.

The AES operators SR, MC, ARK, and KS are linear, *i.e.*, they propagate differences in a deterministic way (with probability 1). However, the S operator is not linear: Given a byte difference $B \oplus B' = \delta B$, the probability that δB 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 δX becomes δSX_r is equal to the product of all $Pr(\delta B \rightarrow \delta SB)$ such that δB (resp. δSB) is a byte difference before (resp. after) passing through the 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 δX and δK that maximize this probability.

Two step solving process. To find δX and δK, we search for the values of all differential bytes in *diffBytes*. Both [4] and [8] propose to solve this problem in two steps. In Step

1, a Boolean variable ΔB is associated with every differential byte $\delta B \in \mathit{diffBytes}$ 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 ΔB is assigned to 0, then δB is also assigned to 0; otherwise, we search for a byte value in $[1, 255]$ to be assigned to δB 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*. In this paper, we focus on Step 1 which is the most challenging step: the CP model for Step 2 is rather straightforward when using table constraints, and all instances are efficiently solved by Choco [15] (see [10] for more details).

3 CP Model

In this section, we briefly describe the CP model used to solve Step 1, and refer the reader to [11, 10] for more details. We first introduce basic variables and constraints, that model the AES transformation rules in a straightforward way. Then, we show how to tighten this model by introducing new variables and constraints that model equality relations at the byte level.

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

`XOR` constraint. As ARK and KS mainly perform `XOR` operations, we first define a `XOR` constraint. Let us consider three differential bytes δA, δB and δC such that $\delta A \oplus \delta B = \delta C$. If $\delta A = \delta B = 0$, 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 δC is equal to 0 or not: This depends on whether $\delta A = \delta B$ or not. When abstracting differential bytes δA, δB and δC with Boolean 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: $\texttt{XOR}(\Delta A, \Delta B, \Delta C) \Leftrightarrow \Delta A + \Delta B + \Delta C \neq 1$.

`AddRoundKey` and `KeySchedule` constraints. Both ARK and KS are modeled with `XOR` constraints between ΔX_i, ΔY_i, and ΔK_i variables (for ARK), and between ΔK_i and ΔSK_i variables (for KS).

`ShiftRows` and `MixColumns`. SR simply shifts variables. The MDS property of MC is ensured by posting a constraint on the sum of some variables of ΔX_i and ΔY_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. This basic CP model is complete, *i.e.*, for any solution at the byte level (on δ variables), there exists a solution at the Boolean level (on Δ 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.

Class variables. When reasoning at the Boolean level, many solutions are not byte-consistent because constraints at the byte level have been ignored. However, the output of the XOR operations allow us to infer some information about the equalty or difference between the byte values represented by some of the differential bits. Thus, we introduce new variables that model equality relations between differential bytes. More precisely, for each differential byte $\delta A \in \mathit{diffBytes}$, we introduce a variable $\mathit{Class}_{\delta A}$ that models the equivalence class of δA: two differential bytes δA and δB belong to the same equivalence class (*i.e.*, $\mathit{Class}_{\delta A} = \mathit{Class}_{\delta B}$) iff $\delta A = \delta B$. The domain of $\mathit{Class}_{\delta A}$ is $[0; 255]$, as there are 256 byte values.

We add a constraint to assign class 0 to differential bytes with value 0, *i.e.*, $\Delta A = 0 \Leftrightarrow \mathit{Class}_{\delta A} = 0$. Also, we break symmetries due to the fact that equivalence classes may be swapped by enforcing an order on them with a precede constraint [12].

Revisiting `XOR` *with Class variables.* When defining the $\texttt{XOR}(\Delta A, \Delta B, \Delta C)$ constraint, if $\Delta A = \Delta B = 1$, then we cannot know whether 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 (resp. different from 0), meaning that the two bytes δA and δB are equal (resp. different), *i.e.*, that $\mathit{Class}_{\delta A} = \mathit{Class}_{\delta B}$ (resp. $\mathit{Class}_{\delta A} \neq \mathit{Class}_{\delta B}$). 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:

$$
\begin{aligned}
\texttt{XOR}(\Delta A, \Delta B, \Delta C) \Leftrightarrow \quad & \Delta A + \Delta B + \Delta C \neq 1 \\
& \wedge\ (\mathit{Class}_{\delta A} = \mathit{Class}_{\delta B}) \Leftrightarrow (\Delta C = 0) \\
& \wedge\ (\mathit{Class}_{\delta A} = \mathit{Class}_{\delta C}) \Leftrightarrow (\Delta B = 0) \\
& \wedge\ (\mathit{Class}_{\delta B} = \mathit{Class}_{\delta C}) \Leftrightarrow (\Delta A = 0)
\end{aligned}
$$

Propagation of MDS at 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 number of bytes of these columns that are equal (*i.e.*, that have the same $Class$ values).

New 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 δK_0, and bytes of δSK_{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 δK_0 and δSK_{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).

4 Decomposition of Step 1 into two sub-steps

Our cryptanalytic problem must be solved for three key lengths $l \in \{128, 192, 256\}$, and for each key length, it must be solved for different round numbers r: when $l = 128$ (resp. 192 and 256), r ranges from 3 to 5 (resp. 10 and 14). Hence, we have 3, 8, and 12 instances for AES-128, AES-192, and AES-256, respectively. For each instance, the goal is to find all solutions for a given value of obj_{Step1}. In this paper, we consider only one value for obj_{Step1}, *i.e.*, the smallest value for which at least one of the Step1 solutions is byte-consistent as this is the most challenging problem (see [10]).

The CP model for Step 1 was implemented with the MiniZinc modeling language [14], and we compared four CP solvers: Gecode [9], Choco 4 [15], Picat_Sat [16] and Chuffed [5]. The best results were obtained with Picat_Sat and Chuffed. When $l = 128$, the best performing solver is Chuffed, which is able to solve all instances within an hour. On the other hand, for larger keys ($l \in \{192, 256\}$), Picat_Sat gradually becomes more efficient. However, for the most difficult instance of the problem ($l = 192$ and $r = 10$), neither Chuffed nor Picat_Sat can enumerate all solutions within a week.

However, empirical evaluation shows us that Picat_Sat finds the first solution rather quickly, but is not efficient at enumerating all possible solutions when there are too many solutions. Furthermore, when looking at solutions, we observe some kind of structure in the repartition of the Δ variables that are set to 1 and that pass through the SubBytes operator S. When $l = 128$, these variables are, for each round $i \in [0, r-1]$: $SX_i = \{\Delta X_i[j][k], j, k \in [0,3]\}$ and $SK_i = \{\Delta K_i[j][3], j \in [0,3]\}$[4].

To take advantage of the complementarity of Chuffed and Picat_Sat, we decompose Step 1 into two sub-problems. To this aim, we introduce two new integer variables for

[4] When $l \in \{192, 256\}$, the definition of SK_i is different.

each round $i \in [0, r-1]$, called $SumX_i$ and $SumK_i$, and we post the constraints: $SumX_i = \sum_{\Delta X_i[j][k] \in SX_i} \Delta X_i[j][k]$ and $SumK_i = \sum_{\Delta K_i[j][3] \in SK_i} \Delta K_i[j][3]$.

Step 1a. We use Picat_Sat to search for all possible consistent assignments for $SumX_i$ and $SumK_i$ variables: these assignments must satisfy all Step1 constraints. This search is done incrementally, by adding constraints each time a new solution is found. Let the values of $SumX_i$ and $SumK_i$ in the solution be x_i and k_i: we add the constraint $\bigvee_{i=0}^{r-1} (SumX_i \neq x_i \vee SumK_i \neq k_i)$ before searching for a new solution.

Step 1b. For each solution of Step 1a, we use Chuffed to search for all boolean solutions, given the values of $SumX_i$ and $SumK_i$ variables in the Step1a solution.

Experimental results. Figure 2 details the solving times for Step 1 without decomposing the problem, both with Chuffed and Picat_Sat, and when decomposing Step 1 into Step1a and Step1b. For very small instances, Chuffed alone is better for all versions of the AES. For larger instances Picat_Sat performs better than Chuffed and can solve almost all of them within the time limit of 48 hours. However, one instance is out of reach for Picat_Sat even after a week of computation. On the other hand, it is solved within 12 hours after decomposing the problem in Step 1a and Step 1b. We do not give the detailed results of each step: Step 1b is always very quickly solved by Chuffed (in 107 seconds for the hardest instance), and most of the solving time is spent for Step 1a by Picat_Sat. Table 1 gives the number of solutions of Steps 1a and 1b for each instance, showing that Step 1a has much less solutions than Step 1b. For example, for AES-192 with $r = 10$ rounds, Step 1a only has 5 solutions whereas Step 1b has 27548 solutions.

5 Conclusion

We have shown that we can solve all instances of our cryptanalysis problem in less than twelve hours when decomposing the problem into two steps and taking advantage of the complementarity of Chuffed and Picat_Sat. This is much faster than the Branch & Bound approach of [4], which needs several weeks to solve some of these instances. It is also faster and much less memory consuming than the approach of [8], that needs 60GB and 30 minutes on a 12-core computer to pre-compute the graph for AES-128.

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

	AES-128			AES-192								AES-256										
r	3	4	5	3	4	5	6	7	8	9	10	3	4	5	6	7	8	9	10	11	12	13	14
Step 1a	1	1	8	3	1	1	2	1	1	1	5	3	2	1	2	1	2	1	1	1	1	1	1
Step 1b	4	8	1113	15	4	2	6	4	8	240	27548	33	14	4	3	1	3	16	4	4	4	4	4

Table 1. Number of solutions of Steps 1a and 1b, for AES-128, 192 and 256, and for different number of rounds r.

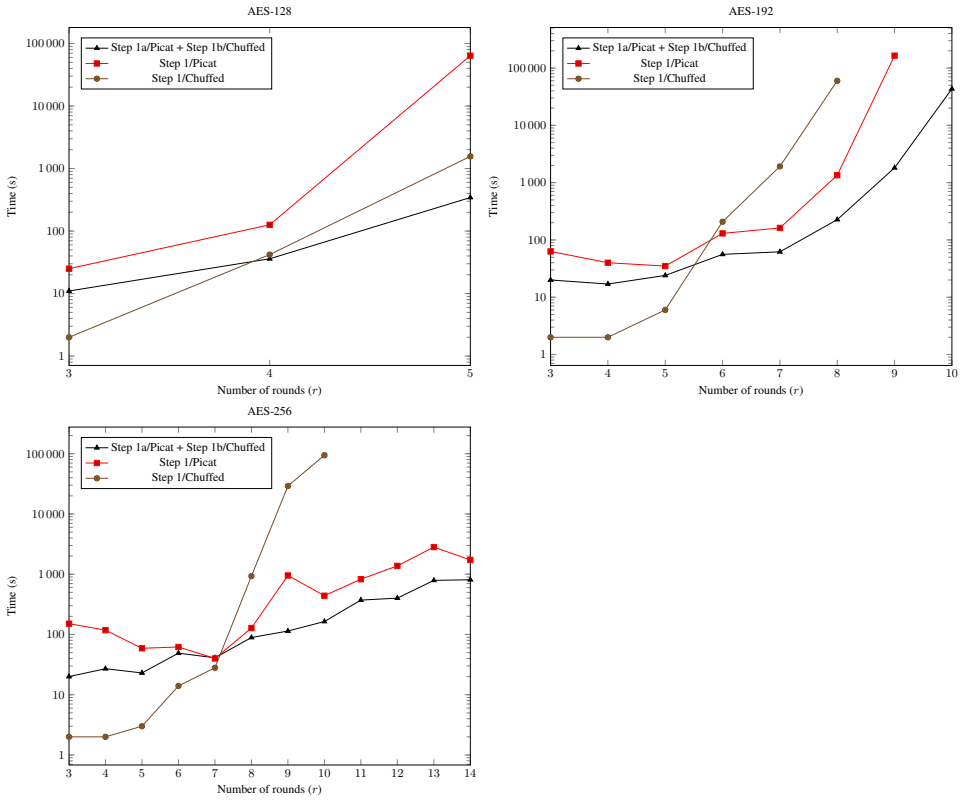

Fig. 2. Results for AES-128 (up left), AES-192 (up right), and AES-256 (down left). Each point gives the time needed to enumerate all solutions (no point if time exceeds 48 hours).

better than the solution claimed to be optimal in [4] and [8]: In these papers, the authors say that the best byte-consistent solution has $obj_{Step1} = 13$, and a probability equal to 2^{-81}. Furthermore, we have shown that the solution proposed in [4] 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 [3] (see [10] for more details).

Further work. These cryptanalysis problems open new challenges for the CP community. In particular, these problems are not easy to model. Naive CP models (such as the first model we introduced in [13]) do not scale well. The introduction of equality constraints at the byte level (as proposed in [11]) and the decomposition of Step 1 into two sub-steps solved by two different solvers allow us to solve the hardest instances within twelve hours but this kind of modeling tricks are not straightforward to design. Hence, a challenge is to define new CP frameworks, dedicated to cryptanalysis problems, in order to ease the development of efficient CP models for these problems.

References

1. Biham, E.: New types of cryptoanalytic attacks using related keys (extended abstract). In: Advances in Cryptology - EUROCRYPT '93. Lecture Notes in Computer Science, vol. 765, pp. 398–409. Springer (1993)
2. Biham, E., Shamir, A.: Differential cryptoanalysis of feal and n-hash. In: Advances in Cryptology - EUROCRYPT '91. Lecture Notes in Computer Science, vol. 547, pp. 1–16. Springer (1991)
3. Biryukov, A., Khovratovich, D., Nikolic, I.: Distinguisher and related-key attack on the full AES-256. In: Advances in Cryptology - CRYPTO 2009. Lecture Notes in Computer Science, vol. 5677, pp. 231–249. Springer (2009)
4. 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 - EUROCRYPT 2010. Lecture Notes in Computer Science, vol. 6110, pp. 322–344. Springer (2010)
5. Chu, G., Stuckey, P.J.: Chuffed solver description (2014), available at `http://www.minizinc.org/challenge2014/description\_chuffed.txt`
6. Daemen, J., Rijmen, V.: The Design of Rijndael. Springer-Verlag (2002)
7. FIPS 197: Advanced Encryption Standard. Federal Information Processing Standards Publication 197 (2001), u.S. Department of Commerce/N.I.S.T.
8. Fouque, P.A., Jean, J., Peyrin, T.: Structural evaluation of aes and chosen-key distinguisher of 9-round aes-128. In: Advances in Cryptology - CRYPTO 2013. Lecture Notes in Computer Science, vol. 8042, pp. 183–203. Springer (2013)
9. Gecode Team: Gecode: Generic constraint development environment (2006), available from `http://www.gecode.org`
10. Gérault, D., Lafourcade, P., Minier, M., Solnon, C.: Revisiting aes related-key differential attacks with constraint programming. Cryptology ePrint Archive, Report 2017/139 (2017), `http://eprint.iacr.org/2017/139`
11. Gerault, D., Minier, M., Solnon, C.: Constraint programming models for chosen key differential cryptanalysis. In: Principles and Practice of Constraint Programming - CP 2016. Lecture Notes in Computer Science, vol. 9892, pp. 584–601. Springer (2016)
12. Law, Y.C., Lee, J.H.M.: Global Constraints for Integer and Set Value Precedence, pp. 362–376. Springer Berlin Heidelberg (2004)
13. Minier, M., Solnon, C., Reboul, J.: Solving a Symmetric Key Cryptographic Problemwith Constraint Programming (Jul 2014), `https://hal.inria.fr/hal-01092574`, modRef 2014, Workshop of the CP 2014 Conference, September 2014, Lyon, France
14. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard CP modelling language. In: Principles and Practice of Constraint Programming - CP 2007. Lecture Notes in Computer Science, vol. 4741, pp. 529–543. Springer (2007)
15. Prudhomme, C., Fages, J.G.: 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)
16. Zhou, N.F., Kjellerstrand, H., Fruhman, J.: Constraint Solving and Planning with Picat. Springer (2015)