Solving a Symmetric Key Cryptographic Problem with Constraint Programming

Marine Minier, Christine Solnon, Julia Reboul

Solving a Symmetric Key Cryptographic Problem with Constraint Programming

Marine Minier[1,2], Christine Solnon[1,3], and Julia Reboul[1]

[1] Université de Lyon, INSA-Lyon, F-69621, France
[2] CITI, INRIA
[3] LIRIS, CNRS UMR5205
{marine.minier,christine.solnon,julia.reboul}@insa-lyon.fr

Abstract. This paper tries to sum up a starting work at the edge between Cryptography and Constraint Programming. Indeed, many cryptographic problems are solved using Branch & Bound approaches which are implemented from scratch using classical programming languages such as C. This implies quite a lot of programming work. Furthermore, these problems are NP-hard and solving them within a reasonable amount of time is still challenging.
The main goal of this paper is to investigate the capabilities of classical Constraint Programming tools for solving these problems. In this preliminary study, we focus on a particular problem coming from the symmetric key cryptography world. This particular problem could help cryptographers mount attacks called differential attacks against block ciphers. Branch & Bound approaches are not able to solve this problem within a reasonable amount of time. We thus introduce a CP model to solve it, and show that Choco is able to solve it to optimality in a few hours.
Keywords: Symmetric Key Cryptography, Block Cipher, Differential Attack, Constraint Programming, Choco.

1 Introduction

Cryptography is now a cornerstone for the security of communications. It guarantees properties such as confidentiality, integrity and signature. Whereas public key cryptography is usually built on problems well known for their hardnesses, symmetric key cryptography relies on simple operations that are iterated many times to speed up the encryption/decryption process.

The most important symmetric key primitives are hash functions that guarantee integrity and stream and block ciphers that guarantee confidentiality. Hash functions create a fixed size fingerprint from messages of arbitrary lengths. They are the most famous examples of symmetric key algorithms. During the last decade, however, many cryptanalytic results

appeared that completely break the standards MD5, SHA-0 and SHA-1 [15, 16, 14] by finding collisions, *i.e.*, two messages having the same reduction. Following these results, SAT solvers have been used to generate such collisions [9, 5, 8] and the future hash standard Keccak [10].

Concerning cryptanalysis of stream ciphers, several papers propose different approaches. In [13], the authors propose to solve algebraic systems generated by writing the equations linking together the keybits and the outputs of a stream cipher. In [11], the authors compute the bounds of differential and linear cryptanalysis using mixed-integer linear programming. They apply their results to a particular stream cipher named Enocove-128v2. However only a few results focus on the cryptanalysis of block ciphers.

In this paper, we propose to model the new attacks proposed in [3, 7] against block ciphers using Constraint Programming (CP). We have chosen CP because our goal in this starting work is to propose a first mathematical model by means of constraints, and have a first feedback on the hardness of the problem for classical CP tools. Other tools could be used such as, for example, Integer Programming, Satisfiability Modulo Theories, or SAT, and our goal for further work is to compare different models and approaches.

This paper is organized as follows: Section 2 describes the cryptanalytic problem on block ciphers; Section 3 describes the CP model; and Section 4 gives first experimental results obtained with Choco [12].

2 Problem Statement

In this Section, we detail the general structure of a block cipher focusing on the particular case of the Advanced Encryption Standard (AES) [6]. We then describe what a differential attack is and finally introduce the chosen key differential attack model.

2.1 Block Ciphers

A block cipher is a bijective function E that ciphers binary blocks (called plaintexts) of length n under binary keys of length k to output binary ciphertexts of length n:

$$\begin{aligned} E : \{0,1\}^n \times \{0,1\}^k &\to \{0,1\}^n \\ (x, K) &\mapsto y = E_K(x) \end{aligned}$$

such that $x = E_K^{-1}(E_K(x))$ for all keys K and plaintexts x. Most of today's block ciphers have an iterated structure. They apply what we call a round function f r times. The round function f is parametrized by a round subkey K_i which is generated from the master key K using a Key Schedule (KS) algorithm. The left part of Fig. 1 gives a general overview of a block cipher.

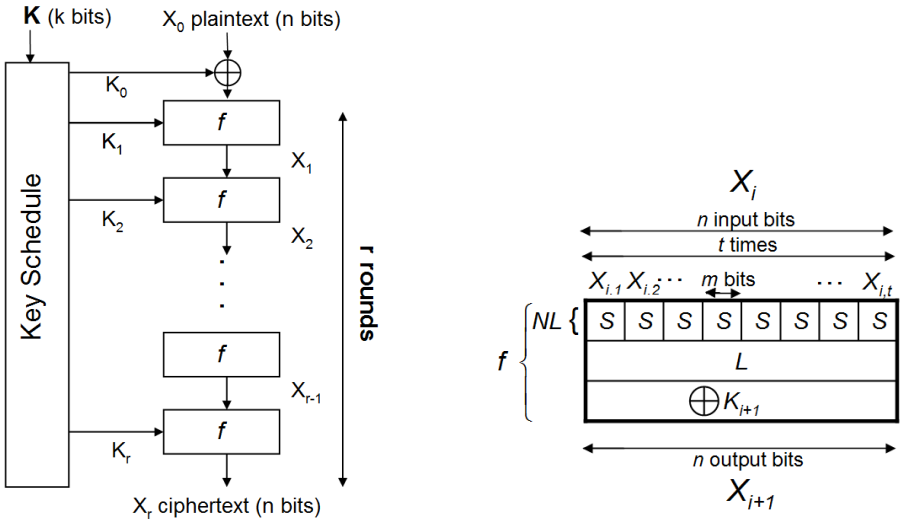

Fig. 1. On the left: General Overview of a modern block cipher. On the right: Description of the i^{th} round that computes X_{i+1} from X_i and K_{i+1} so that $X_{i+1} = f(X_i, K_{i+1}) = L(NL(X_i), K_{i+1})$.

Two famous examples of block ciphers are the Data Encryption Standard (DES) and the AES: DES was the encryption standard between 1977 and 2001, and it is an example of what we call a Feistel scheme. AES is the actual standard since 2001 and it uses an SPN (Substitution-Permutation Network) structure.

The round function f of an SPN is, as indicated by its name, composed of a non-linear layer NL (the substitution layer) and a linear layer L (the permutation layer followed by a xor with the subkey K_{i+1}) so that $X_{i+1} = f(X_i, K_{i+1}) = L(NL(X_i), K_{i+1})$.

The non-linear layer NL is usually composed by t parallel applications of a single non-linear permutation called an S-box which acts on blocks of size $m = n/t$ bits. More precisely, X_i is cut into t sub-sequences, denoted

$X_{i,1}, \ldots, X_{i,t}$, and $NL(X_i)$ is the concatenation of $S(X_{i,1}), \ldots, S(X_{i,t})$, as shown on the right part of Fig. 1. The linear layer L usually acts on the whole n-bit vector.

2.2 The AES

Since 2001, the standard of block ciphers is the AES [6] designed by V. Rijmen and J. Daemen. The AES ciphers blocks of length $n = 128$ bits, where each block is seen as a 4×4 matrix of bytes, under keys of length $k = 128, 192$ or 256 bits. The number of rounds depends on the key length: $r = 10$ (resp. 12 and 14) for $k = 128$ (resp. 192 and 256). Each byte is seen as an element of the set $\{0, \cdots, 255\}$.

As explained in Section 2.1, the AES is an SPN, and its round function f is composed of a non-linear layer NL and a linear layer L. The non-linear layer calls 16 times (one for each byte, *i.e.*, $t = 16$) a single S-box S that acts at byte level (i.e. $m = 128/16 = 8$ bits).

The linear part L of the AES is composed of the following 3 linear mappings:

- ShiftRows (SR): a linear mapping that rotates on the left all the rows except the first one of the current matrix as follows:

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

$\xrightarrow{\text{SR}}$

a	b	c	d
f	g	h	e
k	l	i	j
p	m	n	o

- MixColumns (MC): another linear mapping represented by a 4×4 matrix chosen for its good properties of diffusion (see [4]). Each column of the input matrix is multiplied by a fixed matrix.
- AddRoundKey (ARK): a simple x-or operation between the current block and the subkey of the round r denoted by K_r.

Those r rounds are surrounded at the top by an initial key addition with the subkey K_0. The last round does not contain the MixColumns operation. We omit here the details of the key schedule that may be found in [6]. Just note that each subkey K_i is directly computed from the subkey K_{i-1} and that most of the key schedule operations are linear and only 4 calls to the non linear S-box are made.

2.3 Differential Cryptanalysis

Differential cryptanalysis has been introduced by E. Biham et A. Shamir in 1991 [2]. The main principle of this technique is to consider plaintext pairs (X_0, X'_0) and to study the propagation of the initial difference between X_0 and X'_0 into the cipher, while going through the successive round functions f. We note δX_i as the difference between the two plaintexts X_i and X'_i obtained after the ith round of the ciphering of X_0 and X'_0. Usually, this difference is computed using the x-or operator (denoted $\oplus$), *i.e.*, $\delta X_i = X_i \oplus X'_i$.

Let us keep in mind that the round function f is composed of a linear part L and a non linear part NL, *i.e.*, $X_i = f(X_{i-1}, K_i) = L(NL(X_{i-1}), K_i)$. The linear part L only moves the differences to some other place. Indeed, for every linear operator l we have $l(A \oplus B) = l(A) \oplus l(B)$. So, we can easily predict how the differences are propagated from δX_i to δX_{i+1} by the linear layer L.

The non linear layer has to be studied more carefully. As said before, the non linear part NL of the f function is decomposed into t parallel calls to a given S-box S which operates on m bits such that each S-box computes $S(X_{i,j})$ for a different sub-sequence $X_{i,j}$ of X_i with $j \in \{1, \ldots, t\}$. Therefore, we first need to study how the S-box propagates the differences for a pair (A, B) of sequences of m bits. 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 sequences of m bits. This probability is denoted $D_{\alpha,\beta}$ and is defined by

$$D_{\alpha,\beta} = \frac{\#\{(A,B) \in \{0,1\}^m \times \{0,1\}^m \mid (A \oplus B = \alpha) \wedge (S(A) \oplus S(B) = \beta)\}}{2^m}$$

For example, let us consider the S-box of the AES that acts on bytes (so that $m = 8$), and let us consider an input difference $\alpha = 00000001$ and an output difference 00100000. The transition from 00000001 to 00100000 only occurs for 4 couples (A, B) of inputs (or 2 pairs due to the fact that $A \oplus B = B \oplus A$), among the 256 possible couples. Therefore, the transition probability $D_{00000001,00100000}$ is equal to $\frac{4}{256}$. For the AES S-box with $m = 8$, 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 in the case of the AES S-box, S is a bijection so that $A \oplus B = 0 \Leftrightarrow S(A) \oplus S(B) = 0$. As a consequence, $D_{00000000,00000000} = 1$. 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, to cross a complete non-linear layer NL at round i and study the propagation of the differences from $X_i \oplus X'_i$ to $NL(X_i) \oplus NL(X'_i)$ (where

X_i and X'_i are cut into t sub-sequences $X_{i,1}, \ldots, X_{i,t}$ and $X'_{i,1}, \ldots, X'_{i,t}$), we compute the probability of obtaining the output difference $NL(X_i) \oplus NL(X'_i)$ when the input difference is $X_i \oplus X'_i$. This probability is:

$$p_1(NL(X_i) \oplus NL(X'_i)|X_i \oplus X'_i) = \prod_{j=1}^{t} D_{\alpha_j,\beta_j} \tag{1}$$

where $\alpha_j = X_{i,j} \oplus X'_{i,j}$ and $\beta_j = S(X_{i,j}) \oplus S(X'_{i,j})$.

Finally, the probability of obtaining the output difference $\delta X_r = X_r \oplus X'_r$ (after r rounds) when the input difference $\delta X_0 = X_0 \oplus X'_0$ is:

$$p_2(\delta X_r|\delta X_0) = \prod_{i=0}^{r} p_1(NL(X_i) \oplus NL(X'_i)|X_i \oplus X'_i) \tag{2}$$

where $X_i = L(NL(X_{i-1}), K_i)$ and $X'_i = L(NL(X'_{i-1}), K_i)$ for all rounds $i \in \{1, \ldots r\}$. We refer the reader to [2] for more details.

The goal of the attacker is to find the values of δX_i for $i \in \{0, \ldots r\}$ which maximize this probability p_2.

2.4 Chosen Key Differential Cryptanalysis

Today, differential cryptanalysis is public knowledge, so modern block ciphers such as the 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_0 and X'_0 but also between the keys K and K' to try to mount more powerful attacks. The main idea behind this is to be able to derive from those attacks particular cryptanalysis in the more classical settings: the unknown key model.

We note K_i and K'_i the subkeys of K and K' at round i and we note δK_i the difference between K_i and K'_i, i.e., $\delta K_i = K_i \oplus K'_i$. The goal of the attacker is to find the values of δX_i and δK_i which maximize the probability p_2 defined by equation (2) while satisfying $X_i = L(NL(X_{i-1}), K_i)$ and $X'_i = L(NL(X'_{i-1}), K'_i)$ for every round $i \in \{1, \ldots r\}$.

Two main papers [3, 7] describe results for the chosen key differential cryptanalysis of the AES and propose algorithms for finding the values of δX_i and δK_i which maximize the probability p_2 in the case of the AES. In both papers, the problem is solved in two steps. In the first step, each unknown δX_i (resp. δK_i) is decomposed into a 4×4 matrix of bytes, and a binary variable $\Delta X_i[j][k]$ (resp. $\Delta K_i[j][k]$) is associated with every byte $\delta X_i[j][k]$ (resp. $\delta K_i[j][k]$) at row j and column k of δX_i (resp. δK_i). These

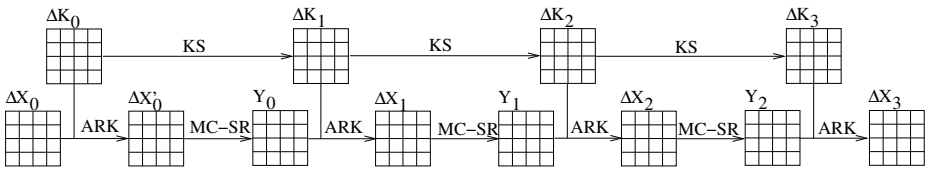

Fig. 2. First step of chosen key differential cryptanalysis with 3 rounds. Each 4×4 array represents a group of 16 binary variables, associated with a bit-vector of 16 bytes. For each round $1 \leq i \leq 3$, ΔX_i is obtained by applying AddRoundKey (ARK) on Y_{i-1} and ΔK_i; then Y_i is obtained by applying ShiftRows and MixColumns (MC-SR) on ΔX_i. Round 0 is a special case, where ARK is applied on ΔX_0 and ΔK_0 before applying MC-SR to obtain Y_0. Each subkey ΔK_i is obtained by applying KeySchedule (KS) on ΔK_{i-1}.

binary variables are equal to 0 if their associated bytes are equal to 0^8, *i.e.*,

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

and they are equal to 1 otherwise, *i.e.*,

$$\Delta X_i[j][k] = 1 \Leftrightarrow X_i[j][k] \neq X'_i[j][k] \Leftrightarrow \delta X_i[j][k] \neq 0^8$$
$$\Delta K_i[j][k] = 1 \Leftrightarrow K_i[j][k] \neq K'_i[j][k] \Leftrightarrow \delta K_i[j][k] \neq 0^8$$

The operations that transform δX_0 into δX_r (described in Section 2.1), are translated into constraints between these binary variables, and new variables are associated with intermediate steps (between ShiftRows and AddRoundKey operations), as illustrated in Fig. 2. In this first step, the goal is to find all solutions which satisfy these constraints while maximizing the sum of a subset of the variables (see the next section for more details on the constraints and the objective function).

Note that during this first step, the non linear part NL of the function f is not considered. Indeed, the S-box does not introduce nor remove differences, i.e., given 2 bytes A and B, $(A \oplus B = 0^8) \Leftrightarrow (S(A) \oplus S(B) = 0^8)$.

In the second step, each solution found in the first step is transformed into a solution of the initial problem, i.e., for each binary variable $\Delta X_i[j][k]$ or $\Delta K_i[j][k]$ set to 1 in the solution, we search for a byte value $\delta X_i[j][k]$ or $\delta K_i[j][k]$ different from 0^8 so that the AES transformation rules are satisfied and the probability p_2 is maximized.

3 CP Model of the first step

In this paper, we propose to use Constraint Programming (CP) to solve the first step of the solution process described in Section 2.4. In this section, we describe the CP model used to solve this problem: variables, constraints, objective function, and ordering heuristics.

3.1 Variables

Let r be the number of rounds. We define the following variables:

- For all $i \in [0; r]$ and for all $j, k \in [0; 3]$, $\Delta X_i[j][k]$ and $\Delta K_i[j][k]$ are the variables which are associated with the bytes $\delta X_i[j][k]$ and $\delta K_i[j][k]$, respectively.
- For all $j, k \in [0; 3]$, $\Delta X'_0[j][k]$ is the variable which is associated with the byte at row j and column k of the result of ARK operation on δX_0 and δK_0.
- For all $i \in [1; r]$ and for all $j, k \in [0; 3]$, $Y_i[j][k]$ is the variable which is associated with the byte at row j and column k of the result of SR and MC operations on δX_i (except when $i = 0$, for which SR and MC are applied on $\Delta X'_0$).

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

3.2 Constraints

The constraints correspond to the propagation of differences by the different operations of the round function f. As said before, the non linear part NL does not imply any constraint as it neither introduces nor removes differences. The linear part L implies the following constraints.

AddRoundKey. ARK is applied on ΔX_0 and ΔK_0 to obtain $\Delta X'_0$, during the first round, i.e.,

$$ARK(\Delta X_0, \Delta K_0, \Delta X'_0)$$

Then, for each next round i, it is applied on Y_{i-1} and ΔK_i to obtain ΔX_i, i.e.,

$$\forall i \in [1, r], ARK(Y_{i-1}, \Delta K_i, \Delta X_i)$$

ARK performs a xor operation. Let B_1 and B_2 be two bytes. If $B_1 = B_2 = 0^8$, then $B_1 \oplus B_2 = 0^8$. If $B_1 = 0^8$ and $B_2 \neq 0^8$, then $B_1 \oplus B_2 \neq 0^8$. However, if $B_1 \neq 0^8$ and $B_2 \neq 0^8$, then we cannot know if $B_1 \oplus B_2$ is equal

to 0^8 or not. When abstracting the byte domain with a binary domain which only models the fact that there is a difference or not, we obtain the following definition of the constraint $ARK(A, B, C)$:

$$\forall (j,k) \in [0;3]^2, (A[j][k] + B[j][k] \neq 2) \Rightarrow (A[j][k] + B[j][k] = C[j][k])$$

where A, B and C are 4×4 binary matrices.

ShiftRows and MixColumns. MC-SR is applied on $\Delta X'_0$ to obtain Y_0, during the first round, i.e.,

$$\textit{MC-SR}(\Delta X'_0, Y_0)$$

Then, for each next round i, it is applied on ΔX_i to obtain Y_i, i.e.,

$$\forall i \in [1, r-1], \textit{MC-SR}(\Delta X_i, Y_i)$$

MC is a particular error correcting code that acts on separated columns while SR moves the differences at some other places. Given two 4×4 binary matrices A and B, the constraint $\textit{MC-SR}(A, B)$ is defined by

$$\forall k \in [0;3], (\sum_{j=0}^{3} A[j][(k+j)\%4] = 0) \Rightarrow (\sum_{j=0}^{3} B[j][k] = 0)$$

$$\forall k \in [0;3], (\sum_{j=0}^{3} A[j][(k+j)\%4] > 0) \Rightarrow (\sum_{j=0}^{3} (A[j][(k+j)\%4] + B[j][k]) \geq 5)$$

KeySchedule. KS is applied at each round i to compute ΔK_i from ΔK_{i-1}, i.e.,

$$\forall i \in [1, r], KS(\Delta K_{i-1}, \Delta K_i)$$

KS is most of the times composed of xor operations between some columns of the key. Given two 4×4 binary matrices A and B, the constraint $KS(A, B)$ is defined by

$$\begin{aligned} \forall j \in [0;3], \quad & (A[j][0] + A[(j+1)\%4][3] \neq 2) \\ & \Rightarrow (B[j][0] = A[j][0] + A[(j+1)\%4][3]) \\ \forall j \in [0;3], \forall k \in [1;3], \quad & (B[j][k-1] + A[j][k] \neq 2) \\ & \Rightarrow \quad (B[j][k] = B[j][k-1] + A[j][k]) \end{aligned}$$

Finally, we add the constraint that the initial plaintexts X_0 and X'_0 must be different, *i.e.*,

$$\sum_{j=0}^{3} \sum_{k=0}^{3} \Delta X_0[j][k] \neq 0$$

and that the initial subkeys K_0 and K'_0 must be different, *i.e.*,

$$\sum_{j=0}^{3}\sum_{k=0}^{3} \Delta K_0[j][k] \neq 0$$

3.3 Objective function

In order to maximize the probability p_2 of equation (2), we have to maximize its different factors. To this aim, we have to maximize the number of factors of equation (1) for which $\alpha = \beta = 0$ (because $D_{0,0} = 1$). This amounts to minimizing the number of $\Delta X_i[j][k]$ and $\Delta K_i[j][3]$ variables which are set to 1. Therefore the objective function to minimize is

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

3.4 Ordering heuristics

As we want to minimize the number of $\Delta X_i[j][k]$ and $\Delta K_i[j][3]$ variables which are set to 1, we add a variable ordering heuristic which first assign these variables, and a value ordering heuristic which first tries to assign them to 0. Of course, this heuristic does not eliminate the need for an explicit objective function, as heuristically setting a variable x to 0 may force some other variables to be set to 1, thus leading to a worse solution than setting x to 1. This ordering heuristically drastically improves the solution process. For example, when the number of rounds is $r = 3$, the time needed to find the optimal solution is increased from 1 second to 29 seconds when removing these ordering heuristics.

4 Experimental Results

We have implemented the CP model described in the previous section in Choco 3 [12]. Table 1 displays experimental results obtained with this very first CP model. It shows us that CP is able to solve the problem up to $r = 5$ in a reasonable amount of time. As the probability defined by equation (2) with $r = 5$ rounds reaches $p = 2^{-105}$, it is useless to try to solve the case for $r = 6$ because p becomes greater than the maximal authorized probability $p = 2^{-128}$. Indeed, in this last case, the attack requires all the code book, *i.e.*, all possible plaintexts. This could be no more consider as an attack because we obtain all possible ciphertexts.

	Search for one solution		Search for all optimal solutions	
r	Time	#Choice points	Time	#Choice points
3	1	3,157	2	5,477
4	377	8,733,900	861	18,476,800
5	4,036	66,534,689	46,835,164	236,253,352

Table 1. Experimental results. Each line successively displays: the number r of rounds, the time (in seconds) and number of choice points needed to find an optimal solution and prove its optimality, the time and number of choice points needed to find all optimal solutions and the number of optimal solutions.

Our Choco program only solves the first step, *i.e.*, it assigns binary variables to 0 if the associated byte is 0^8, and to 1 if the associated byte is different from 0^8. In the second step, for each binary variable $\Delta X_0[j][k]$ or $\Delta K_0[j][k]$ set to 1 in the first step solution, we search for a byte value $\delta X_0[j][k]$ or $\delta K_0[j][k]$ different from 0^8 so that the different AES transformation rules are satisfied, for each round, and the probability p_2 is maximized. Note that some solutions of the first step cannot be transformed into solutions of the initial problem. At this time, this second step is solved by a program written in C. This second step basically involves the exploration of 2^{16} combinations, and it is rather quickly performed. Therefore, each time Choco finds a solution to the first step, we use our C program to search for byte values. If the C program succeeds, then we ask Choco to search for a better solution with respect to the objective function, otherwise we ask Choco to search for another solution without modifying the bound on the objective function. With this method, we retrieve the results given in [7].

5 Discussion

These first results obtained with a simple CP model encoded in Choco in a straightforward way are rather encouraging. Indeed, we have also implemented in C the Branch & Bound approach proposed in [3]. For $r = 3$, this approach is able to find the optimal solution in about one hour and for $r = 4$ in about 24 hours. We did not try to run it for $r = 5$.

The approach proposed in [7] is rather different. It is based on a breadth-first traversal of a graph which contains 2^{32} nodes, corresponding to the 2^{32} possible states considering that all plaintext and key bytes of δX_0 and δK_0 may contain a difference or not. This approach has an exponential memory complexity which may not scale well for other block cipher families. In particular, for the family of block ciphers called Rijndael,

the block and key size vary between 160-bit and 256-bit. In this case, the approach proposed in [7] would no longer be practical due to the size of the graph to store that becomes at least equal to 2^{36} and at most equal to 2^{64}.

As future work, we plan to complete the CP model by integrating byte variables and constraints related to the second step. Indeed, many solutions found during the first step cannot be transformed into solutions during the second step because we cannot find byte values which satisfy the constraints defined by the non linear S-box. Adding these constraints should allow us to reduce the search space and improve the solution process.

We also plan to extend this preliminary work to other families of block ciphers, such as the Rijndael family. Indeed, for this family, the approach proposed in [7] cannot be used because of its exponential memory complexity.

In this preliminary work, we have used Choco to solve the first step of our problem. Our goal was to validate our CSP model, and evaluate the hardness of this problem for a classical CP library. As pointed out in the introduction, other approaches could be used and some of them should be more suited to solve this optimization problem. Therefore, we plan to compare CP with Integer Programming, Satisfiability Modulo Theories and SAT solvers. Also, as pointed out by one reviewer, one way to optimize the objective function indirectly in a single run of a CP solver is to use Limited Discrepancy Search (LDS) on the variables of the objective function (and branch on these variables first, before other variables).

Acknowledgement. Many thanks to our reviewers for their careful reading and valuable comments.

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

5. 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.
6. FIPS 197. Advanced Encryption Standard. Federal Information Processing Standards Publication 197, 2001. U.S. Department of Commerce/N.I.S.T.
7. 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.
8. 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.
9. 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.
10. Pawel Morawiecki and Marian Srebrny. A sat-based preimage analysis of reduced keccak hash functions. *Inf. Process. Lett.*, 113(10-11):392–397, 2013.
11. 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.
12. 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.
13. 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.
14. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full sha-1. In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.
15. 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.
16. 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.