Master Thesis

Constraint programming for symmetric cryptographic problems solving

Programmation par contraintes pour la résolution de problèmes de cryptographie symétrique

Ambroise Baudot

Academic Year 2019–2020

Final year internship done in partnership with
Loria
in preparation for the engineering diploma of TELECOM Nancy and Master's degree at
Université de Lorraine

Internship supervisor: Marine Minier
Academic supervisor: Dominique Méry

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),
Nom, prénom : Baudot, Ambroise
Élève-ingénieur régulièrement inscrit en 3e année à TELECOM Nancy et à Université de Lorraine
Numéro de carte de l'étudiant(e) : 31721544
Année universitaire : 2019–2020
Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Constraint programming for symmetric cryptographic problems solving - *Programmation par contraintes pour la résolution de problèmes de cryptographie symétrique*

Par la présente, je déclare m'être informé sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.
Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.
Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.
Je suis conscient que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Nancy, le September 13, 2020
Signature :

Master Thesis

Constraint programming for symmetric cryptographic problems solving

Programmation par contraintes pour la résolution de problèmes de cryptographie symétrique

Ambroise Baudot

Academic Year 2019–2020

Final year internship done in partnership with
Loria
in preparation for the engineering diploma of TELECOM Nancy and Master's degree at
Université de Lorraine

Ambroise Baudot
ambroise.baudot@inria.fr

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00
contact@telecomnancy.eu

Université de Lorraine
Campus, Boulevard des Aiguillettes,
54506, VANDŒVURE-LÈS-NANCY
+33 (0)3 72 74 50 00
fst-scol-contact@univ-lorraine.fr

Loria
Campus Scientifique, 615 Rue du Jardin-Botanique
54506, VANDŒVURE-LÈS-NANCY
+33 (0)3 83 59 20 00

Internship supervisor: Marine Minier
Academic supervisor: Dominique Méry

Acknowledgments

“ First of all, I would like to thank Ms Marine Minier, for giving me the opportunity to achieve this internship within CARAMBA team.
Her valuable help over the past few months allowed me to carry out my work which could not have led to concrete results without her helpful remarks.

Then, because of this exceptional context which has done the work more complex as usual, I would like to thank the administrative staff who accompanied us in our teleworking, as well as our academic administration. ”

– Ambroise Baudot

Contents

Part I

Introduction

1 Presentation

This introduction chapter aims to describe the context in which this document has be written and to present the outline of this one.

About this document

Context To conclude the scholar curriculum of the 3-rd year of *Télécom Nancy* school and Master's Degree at *Université de Lorraine*, a 6-months internship aims to put into practice theoretical skills and knowledge in a company or a laboratory. This internship has been realized in the LORIA (Laboratoire lOrrain de Recherche en Informatique et ses Applications) within the CARAMBA team.

This document This document is the report of this internship. It is to present an overview of the internship: motivations, work realized, future works... in order to apprehend the context and the added value of the work realized in this field.
After an overview of the context, we will first present the state-of-the-art, giving to the reader the best understanding to apprehend the motivations of the work realized, then the work already done on equivalent fields, then the contribution done in this internship, and finally a conclusion relating to the future works will put into perspective the work contributed with the global context of this internship.

Practical Notes This document follows the template (formatting, length, sections, etc.) imposed by *Télécom Nancy*, and is written in English. In addition to this report, an oral presentation will complete this presentation. Appendices are attached to this document to provide further information. To have a deep understanding of the context most precise than in the synthetic "state-of-the-art" part, please refer to the references provided in the bibliography. At last, to complete the anti-plagiarism attestation, with the internship supervisor agreement, some sections presenting either the scientific context or other works already done (not my own contributions) are from other publications. In these cases, they are explicitly mentioned.

About the laboratory

Loria The "Lorraine Research Laboratory in Computer Science and its Applications" (french acronym for "LORIA") was created in 1997 and is a research unit, common to CNRS, the University of Lorraine and INRIA. Loria's missions mainly deal with fundamental and applied research in computer sciences.
Its structure is organized in teams for the scientific part, in addition to the transversely services (HR, IS, etc.). Each teams deal with some fields, and a conductor, the "research director", leads this team. 400 people work in the lab, and 28 teams work on scientific fields ; LORIA in one of the biggest lab in Lorraine. Among these teams, 15 are common with the INRIA.

INRIA CARAMBA team is one of the 15 teams common with the INRIA. Inria is the French national research institute for digital science and technology, with 200 project-teams (often shared with universities or other institutions) and more than 3500 researchers. Inria has born in 1967 from a French government plan. At the birth of the Internet, INRIA leads many projects in different scientific fields, has its own or shared project-teams, its startups, and many shared works with other main actors of scientific research, industrial partners, etc.

CARAMBA team This team is included in the INRIA theme "Algorithmics, programming, software and architecture", and "Algorithmics, Computer Algebra and Cryptology" sub-theme. It is based in the LORIA (in the department "Algorithms, Computation, Image and Geometry") at Nancy. This team includes 8 permanent members (research directors, researchers, professors), 8 non-permanent members (doctorates, interns), and 2 administrative staffs.

The team deals with 3 main fields (excerpt from the official founding document):

- Extended NFS family. A common algorithmic framework, called the Number Field Sieve (NFS), addresses both the integer factorization problem as well as the discrete logarithm problem over finite fields.
- Algebraic curves and their Jacobians. This work consists in developing algorithms and software for computing essential properties of algebraic curves for cryptology, eventually enabling their widespread cryptographic use.
- Design and cryptanalysis of symmetric cryptographic primitives.

and 2 transverse fields:

- Arithmetic. This work relies crucially on efficient arithmetic, be it for small or large sizes. It consists in improving algorithms and implementations, for computations that are relevant to the application areas.
- Polynomial systems. It is rather natural with algebraic curves, and occurs also in NFS-related contexts, that many important challenges can be represented via polynomial systems, which have structural specificities. This research intends to develop algorithms and tools that, when possible,take advantage of these specificities.

In the context of this internship, I have worked on the 3^{rd} main field: cryptanalysis of symmetric cryptographic primitives.

2 Motivations

For centuries, cryptography has always been an important way to communicate without being spied on. Nowadays, it is indispensable, with the use of public channels, like communications on the Internet, for instance.

We can distinguish two main categories of cryptography used today : symmetric and asymmetric cryptography. Let's present the first one, and then the context and the motivations of the works presented in this report.

Symmetric cryptography

Principle The principle of the symmetric cryptography is to use a key for the ciphering process and the same key for the inverse process. In other words, to cipher a plaintext t with a key k, $c = f(t, k)$, and $t = f^{-1}(c, k)$ where c is the ciphertext.

Usage In practice, there is the constraint of the secret of the key, which is not needed in the asymmetric cryptography. That's why it is often more convenient to implement safe protocols thanks to asymmetric cryptography, ensuring security properties like the secret. However, symmetric ciphering processes are more efficient (depending on the rapidity criteria) than asymmetric ones. Using very simple mathematical operations directly defined at the bit or byte level (like XOR operations), symmetric ciphering (considering most commons ciphering systems) is significantly faster than the asymmetric ciphering. That's why this encryption way is very used everywhere on the Internet and in other fields. A common choice is to implement the initialization of the session thanks to an asymmetric protocol, to exchange keys, and then to implement all the whole main process in a symmetric way.

AES Since the XX^{th} century, the National Institute of Standards and Technology published cryptographic standards, depending of current calculus capacity and cryptanalysis knowledge. In 1997, in response to the need to find a new standard, the NIST organized a competition in order to find this new one. In this context, Rijndael algorithm (cf. 3) has been proposed. It is a block ciphering process.
The Advanced Encryption Standard (AES), which is the same algorithm as Rinjdael but with a fixed parameter value, has been chosen as the new standard.

Cryptanalysis

The cryptanalysis is the set of the methods used to break the security properties, here the secret, ensured by the ciphering process. Today, there is no efficient way to "break" the AES in all cases (not preliminary knowledge etc.), but in some configuration, the ciphering process is weaker. The differential cryptanalysis (4) is to study the impact of an input difference, to predict the output with some initial knowledge.

Overview

The related-key cryptanalysis is a kind of differential cryptanalysis, allowing to study a difference not only in the input plaintext but also in the key used for the ciphering process. This way has been studied on the AES, to conclude on the efficiency of these attacks [12]. In this work, we are interested to generalize this method to Rijndael. That was the goal of this internship.

Which results expected ? We will present later different ways to model to obtain some results. Results in fact are some differential characteristic, *i.e.*some combinations of a given input difference in the text, a fixed input difference in the initial key, which allows us to predict the output difference. Depending on the parameter chosen (size of key, blocks, cf. later), the number of solutions is huge, and a challenge is to optimize the exhaustive search. The ideal goal to reach is this exhaustive search in order to have an overview of the number of weak cases: the number of "solutions", where given an initial knowledge an attacker is able to predict easily (or reasonably) the output.

Part II

State of the art

3 Rijndael

Overview This chapter aims to describe the design of Rijndael, following the first steps presented in the specification [6]. Firstly, a short introduction introduces the subject, then the global principle is explained, describing the main idea and the processing, and a last section will end this presentation chapter.

3.1 Presentation

After presenting the context in which Rijndael was conceived, we will introduce the knowledge necessary to understand the explanation that will follow.

3.1.1 Context

In January 1997, the National Institute of Standards and Technology (NIST) started a challenge to select a new candidate for encryption standard. In September, 15 finalists are selected. The criteria on which they are selected are mainly the following: security (in particular a proof that there exist no weaknesses), cost (free and cost of implementation and execution on the aspects size in memory and time spent), and algorithm characteristics (cross-platform, "agility" of the key, simplicity). After a second round, in 2000 the NIST published the selection of Rijndael for the new standard, written by Belgian researchers.

3.1.2 Mathematical background

Since this document is only an overview of what is written and explained in the book, the aim is not to go deeply into the explanations and the details or to copy again the mathematical definitions, that's why the important notions will be mentioned but only the most important ones and without be really detailed.

Preliminaries The first notions to defined is the set on which we will work. We consider that Abelian groups $< G, + >$ and structures $< F, \boxplus, . >$ are given. Then, with an additional operator $\odot : F \times G \rightarrow F$, we are able to dispose of vector spaces, $< G, F, +, \boxplus, ., \odot >$, on which the following rules are verified in addition of rules already defined in structures and abelian groups : associativity of $\odot$ and ., distributivity of $\odot$ on + and $\boxplus$. and 1 as a neutral element of $\odot$ for any v in V.
This allows us to define the notion of *basis* and subsequently of coordinates of vectors over V. Now, we can introduce the notion of linear functions f and matrices associated to theses functions.
The last notion of this introduction is the notion of finite fields, where operations (addition, multiplication) are modulo p, an integer called the *characteristic* of this field, and $GF(p)$ allows to represent elements of this field by an unique way.

Polynomials One of the most important notions is the notion of polynomials over a field F defined as follows: $b(x) = \sum_{i=0}^{n-1} b_i x_i$, where $\forall i, b_i \in F$; these terms are called coefficients, $b_{n-1} \neq 0$, and n is called the degree of the polynomial.
Finally, a polynomial over $GF(2)$ can be represented by a sequence (c_i) where any c_i is in $\{0, 1\}$, and in particular $GF(2)|8$, which correspond to all polynomials with a degree less than 8, for each polynomial of this set there exist an unique byte. For instance, 00010001 corresponds to $x^4 + 1$ and *vice-versa*.

Columns In Rijndael specification, we deal with 4 bytes-columns, so that a column is a polynomial with a degree less than 4. It is possible to transform an input polynomial $c(x) = c_0 + c_1.x + c_2.x^2 + c_3.x^3$, respectively a column $[c_0, c_1, c_2, c_3]$, by multiplying by another polynomial $p(x) = p_0 + p_1.x + p_2.x^2 + p_3.x^3$, respectively a matrix well filled with the coefficient of b, in order to obtain a new polynomial $d(x) = d_0 + d_1.x + d_2.x^2 + d_3.x^3$, respectively a new column $[d_0, d_1, d_2, d_3]$.
Actually, as we can see, the coefficient of these polynomials are the values of the arrays ($d_i.x^i$ is the i^{th} element of the column, $C[i]$). In fact, each coefficient is in $GF(2)|8$. Indeed, we have said above that a column contains bytes, each element of the column is a byte, which can be written like a polynomial with a degree less than 8. Hence, $\forall i \in [\![0, 3]\!], b_i \in GF(2)|8$.

Additional knowledge... In the original book, other information is introduced about linear codes, partitions and boolean functions, modes of operation... which are not the most useful part to explain the following description of the algorithm. There is also another part about block cipher which will be dealt with in subsections of the next section.

3.2 Principle

After an introduction about the needs of the new standard, we will present the global process, and then the details of the steps.

3.2.1 Motivations and constraints

We can note the following aspects:

- Since the cryptosystem, as a standard, will be implemented on all kind of architectures, the operations are simple and memory efficient, and more generally, the process must satisfy the criteria detailed by the NIST (cf. 3.1.1).
- In particular, the encryption must be non-linear, in order to resist to the differential attacks.

3.2.2 Main idea of processing

Rijndael encryption is a block cipher system. The input plaintext is split into 128, 160, 192, 224 or 256 bits-blocks which will be ciphered one after one. In the case of the AES, this value is fixed to 128. The key has also a length of 128, 192, or 256 bits (it does not depend on the block length). Input and output are two-dimensional arrays of 8-bit bytes. In fact, the process is a sequence of rounds (the last one is slightly different from the others), and after each round we consider the intermediate result, called *state*, which is the result of the execution on the data, associated to a current key called "round key". Indeed, the key changes at each round.

Now, let's study the "round" which is executed each time.

3.2.3 Steps

Let's define the five operations which compose the round, applied on an input state (data associated to the current key, "round key") to give us an output text. The first input state is the plaintext (with the good representation of data, cf. section *state* bellow) and the original key, and the output one is the ciphered text.

State

The state is actually the "current text", associated to the round key which will cipher this text.

Key The key is modified at each round with the operation "KeySchedule". In fact, all round keys are computed in advance, in order to obtain `ExpandedKey[]`, an array which contains all the subkeys for each round (so the length is the number of rounds, multiplied by the number of columns of one round key (constant), because subkeys are concatenated one after the other, more precise information are provided later).
The subkey is written in 8-bits bytes format, and then is split into blocks of 32 bits. Then, each block is a 4 bytes-column, so that the subkey is actually a bi-dimensional array, of shape $(4, N_k)$, where N_k is the length of the subkey divided by 32.

Text The input of the first round is the plaintext. As well as the subkey, the text is written in column of 4 bytes. As a column contains $4 * 8 = 32$ bits, there are $N_b = \text{length}/32$ columns in a state (cf. just bellow).

State So for each round we define an input state and an output state. Before the first round, the state is the plaintext, and for any round i, the text ciphered i times, associated to a round key, `ExpandedKey[i]`. They are two bi-dimensional arrays with 4 lines and where each cell is a byte, and a number of column depending on parameters chosen (block length and key length). That is illustrated in Fig. 3.1, with $N_b = 4$ and $N_k = 6$, *i.e.* a block length equals to $4 * 8 * 4 = 128$ bits and a key length equals to $32 * 6 = 192$ bits. In this figure, $a_{i,j}$ is the coefficient at the row i and the column j, so `c[i][j]` in the implementation.

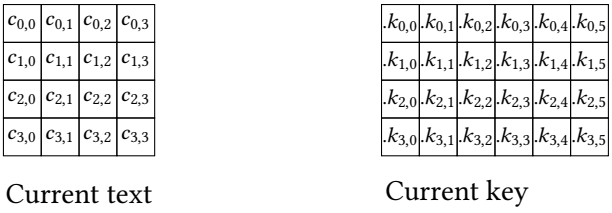


Figure 3.1: State

The main structure can be presented as follows:

```
Round(State, ExpandedKey[i]){
    SubBytes(State)
    ShiftRows(State)
    MixColumns(State)
    AddRoundKey(State, ExpandedKey[i])
}
```

Number of rounds In fact, as we have said, the first and the last round are slightly different (initial key addition `AddRoundKey` and no `MixColumns` in the last round). So the precise description is the following:

```
KeyExpansion //computation of ExpandedKey
AddRoundKey
for i in range(Nr − 1){
    Round(State, ExpandedKey[i]){
        SubBytes(State)
        ShiftRows(State)
        MixColumns(State)
        AddRoundKey(State, ExpandedKey[i])
    }
}
FinalRound(State, ExpandedKey[Nr]){
    SubBytes(State)
    ShiftRows(State)
    AddRoundKey(State, ExpandedKey[i])
}
```

What is the value of N_r?
It is chosen according to the criteria defined, in particular those relating to security and the prevention of attacks. For instance, it must satisfy the "full diffusion" (diffusion is the influence of the change of a bit value on the bits values of subsequent rounds), number of rounds to avoid differences tracking (or make it too difficult), with a security margin. It depends also of the key size, and number found are justified by existence and complexity of attacks with these parameters ([13], [15] information about these choices).

The number of rounds, depending on N_k and N_b, is given in Fig. 3.1.

$\mathbf{N_k}$	N_b				
	4	*5*	*6*	*7*	8
4	10	11	12	13	14
5	11	11	12	13	14
6	12	12	12	13	14
7	13	13	13	13	14
8	14	14	14	14	14

Table 3.1: Number of rounds $N_r(N_k, N_b)$

SubBytes (SB)

This operation is to prevent differential attacks, introducing non-linearity in the ciphering process. Actually it is a substitution process. It is defined according to the criteria needed: good non-linearity property, algebraic complexity (to avoid algebraic attacks), lowest correlation between input and output, lowest propagation of difference (to prevent tracking of differences in differential attacks). A substitution-box, called "S-Box" process this operation. The substitution table, called SRD, is given in the Appendix A.
In $GF(2^8)$, the multiplicative inverse satisfies these criteria [19], defined as follows: $f : a \rightarrow b = a^{-1}$ if $a \neq 0, 0$ otherwise. It can be implemented thanks to a lookup-table. The operation f is a

multiplication of the column by a matrix, followed by an addition (bitwise XOR) with a constant column, that is illustrated in Fig. 3.2.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix} \oplus \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}$$

Figure 3.2: SubBytes

ShiftRows (SR)

For each row, bytes are cyclically shifted. The offset is different for each row and is chosen to maximize the resistance against different kind of attacks. That is illustrated in Fig. 3.2.

$\mathbf{N_b}$ / offset at...	Row 0	Row 1	Row 2	Row 3
4	0	1	2	3
5	0	1	2	3
6	0	1	2	3
7	0	1	2	4
8	0	1	3	4

Table 3.2: ShiftRows

MixColumns (MC)

This step is essentially a linear permutation operating on columns. It has to maximize the diffusion, the influence of the bytes at this rounds on other bytes in the following rounds. This transformation is linear over $GF(2)$.
Each column can be seen as a polynomial with a degree less than 4, and coefficient which are actually the value of the bytes, in $[0\text{x}00,\ 0\text{x}08]$. For instance, $02.x^2 + 03$ is the column $[0_8, 00000010, 0_8, 00000011]$. So columns polynomials are multiplied by a fixed polynomial $c(x)$ of degree less than 4, and the result modulo $(x^4 + 1)$ gives us a new column.

What about the implementation? By the same way as ShiftRows, it can be easily implemented thanks to a fixed matrix. The fixed polynomial is $c(x) = 03.x^3 + 01.x^2 + 02.x + 02$. So, for a column $[a_i]$, we can compute the result $[b_i]$ as follows: $b(x) = c(x).a(x) \pmod{x^4 + 1}$. This is illustrated in the Fig. 3.3.
In practice, each coefficient is computed as follow, according to this definition:

$$\begin{cases} b_0 &= (02 \cdot a_0) \oplus (03 \cdot a_1) \oplus a_2 \oplus a_3 \\ b_1 &= a_0 \oplus (02 \cdot a_1) \oplus (03 \cdot a_2) \oplus a_3 \\ b_2 &= a_0 \oplus a_1 \oplus (02 \cdot a_2) \oplus (03 \cdot a_3) \\ b_3 &= (03 \cdot a_0) \oplus a_1 \oplus a_2 \oplus (02 \cdot a_3) \end{cases}$$

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Figure 3.3: MixColumns

How to compute $\mathbf{a} \cdot \mathbf{b}$*?* Firstly, bytes a and b are written like polynomials, and then after the multiplication is proceed, the result is divided to be *modulo* an irreducible polynomial: $x^8 + x^4 + x^3 + x + 1$ (`0x 01 1b`). Thus, the result, with a degree less than 8 (because of the *modulo* operation), is another byte.
Example $57 \cdot 83 =$ `c1`. Indeed, $(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) \pmod{x^8 + x^4 + x^3 + x + 1} = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \pmod{x^8 + x^4 + x^3 + x + 1} = x^7 + x^6 + 1$.

AddRoundKey (ARK)

This operation consists in a simple bitwise XOR between the round key (`ExpandedKey[i]`) and the state, to obtain the new state $[a_i] \oplus [k_i] = [b_i]$. That is illustrated in Fig. 3.4.

a_0	a_4	a_8	a_{12}
a_1	a_5	a_9	a_{13}
a_2	a_6	a_{10}	a_{14}
a_3	a_7	a_{11}	a_{15}

$\oplus$

k_0	k_4	k_8	k_{12}
k_1	k_5	k_9	k_{13}
k_2	k_6	k_{10}	k_{14}
k_3	k_7	k_{11}	k_{15}

$=$

b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

Figure 3.4: AddRoundKey

KeySchedule (KS)

Reminder: N_r denotes the number of rounds, N_b the number of columns in the state (N_b = block length/32).
As mentioned above, the key changes at each round thanks to the `KeySchedule` operation. Actually, this operation first computes the `ExpandedKey`, whose length is equal to the block length times the number of round plus one (because of the initial derivation at the beginning), and then the round key selection. Thus, the number of columns of `ExpandedKey` is $N_b * (N_r + 1)$. For the round i, the subkey to use is `ExpandedKey`$[\,(N_b * i) : (N_b * (i+1) - 1)\,]$.

As well as the text ciphering, this process must be non-linear to prevent differential attacks. What's more, the diffusion must be maximized and efficient, the symmetries must disappear, and the computation must be memory efficient.

Definition The computation of a byte depends of the column. N_k denotes the number of columns for a master key. To compute k_i, the column at the index i,

$$k_i = k_{i-N_k} \oplus \begin{cases} k_{i-1} & \text{if } i = 0 \text{mod.} N_k \\ f(k_{i-1}) & \text{otherwise, where } f \text{ is a non-linear function} \end{cases}$$

Actually, the non-linear function f is a sequences of bitwise XOR combining the S-box (non-linear) and round constants (RC) on the first column with a rotation (in order to break the symmetry, further details at the end of the KeySchedule description). RC, on $GF(2^8)$, are recursively defined as follows:

- $RC(0) = x^0 = \texttt{0x00}$
- $RC(1) = x^1 = \texttt{0x01}$
- $RC(i) = x.RC(i-1) = x^{i-1}, i > 2$

The precise definition of the main process of this operation and of the non-linear function is given in Fig. 3.5. The schema Fig. 3.6 illustrates the overall processing. `W` is a shorter notation for `ExpandedKey`, which denotes also the 4 rows of the round keys: `ExpandedKey[i]`, the round key at the round i, is denoted by `W[.][i]`. The dimensions of `W` are $(4, N_b * (N_r + 1))$.

3.3 Remarks

After the presentation of the encryption process, let's talk about decryption, and then let's add a few details about the justifications.

3.3.1 Decryption

All these operations are reversible, *i.e.* it is possible to find and implement the inverse operation. For instance, the inverse of the XOR operation is the XOR itself ($((a \oplus k) \oplus k) = a$), and the multiplication of a column by a matrix can by multiplied by the inverse matrix to find the original ($(A.M).M^{-1} = A$).

Finally, with the same complexity and memory cost, it is possible to implement the decryption in order to obtain the plaintext.
`Decryption(CipheredText, Key)`
The first operation is to obtain `ExpandedKey` from `Key` thanks to the same function.

3.3.2 Justification

As said above, some parameters are in fact fixed constants (coefficients in polynomials/matrices, round constants...). They have been chosen to optimize the compromise between the complexity and the security level. Indeed there are some constraints (for instance, avoiding patterns in functions definition, like fixed points or inverted fixed points), some elements to be maximized or minimized... All these details are provided in the corresponding sections of the original book [6].

3.4 Conclusion

After few word about the optimization, let's conclude and talk about the AES.

3.4.1 Optimization

The steps described above and the inverse process can be seen as the strict definition. However, it is possible to remark that operations are interchangeable, *i.e.* $f(g(a)) = g(f(a))$. Assuming we dispose of equivalents models, it appears that there are "patterns", sequences of functions, which are similar. In fact, interchanging operations in the decryption process (obtaining a strictly equivalent model), there is a sequence of functions which can be replaced by another one in order to simplify the implementation, regrouping operations.

3.4.2 AES

From Rinjdael, the AES has been defined [8]. The AES encryption is the Rijndael encryption where a parameter has been fixed: the block length is equal to 128, that is illustrated in 3.7. However, the key length can still be chosen among $\{129, 192, 256\}$.

Inputs:

- Initial key, `K[4][`N_k`]`
- Empty ExpandedKey key, `W[4][`$N_b(N_r+1)$`]`

Global process: `KeyExpansion`

```
KeyExpansion(...){
    for (j=0; j<Nk; j++)
        //initialization
    for (j=Nk; j<Nb(Nr+1); j++)
        //key schedule
}
```

Initialization

```
for (i=0; i<4; i++)
    W[i][j]=K[i][j];
```

In fact, the KeySchedule operation depends on the size of the state. The conditions to process some operations are slightly different.

if $N_b \leq 6$: KeySchedule:

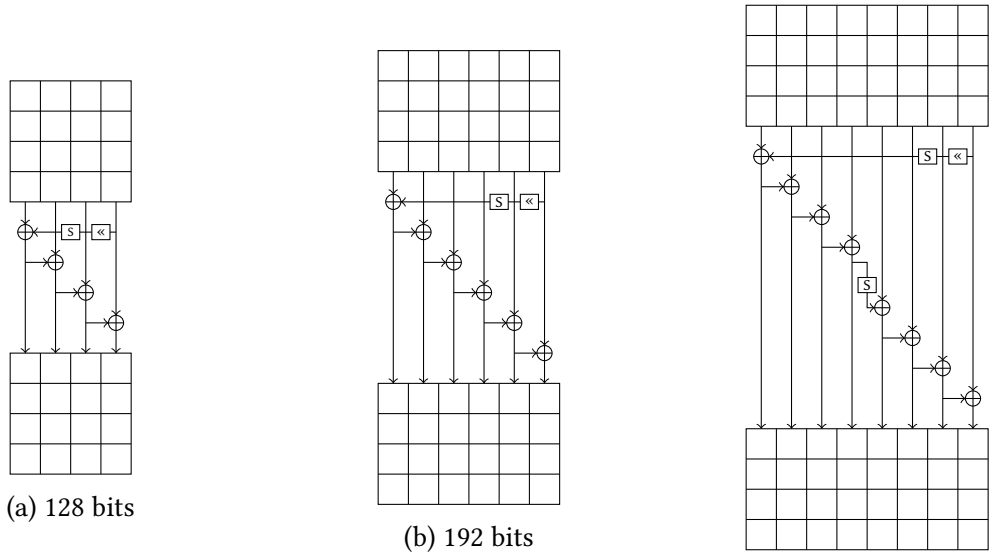
```
if (j mod Nk == 0) { //non-linear function
    W[0][j] = W[0][j-Nk] ⊕ S[W[1][j-1]] ⊕ RC[j/Nk];
    for (i=1; i<4; i++)
        W[i][j] = W[i][j-Nk] ⊕ S[W[i+1 mod 4][j-1]];
} else {
    for (i=0; i<4; i++)
        W[i][j] = W[i][j-Nk] ⊕ W[i][j-1];
}
```

if $N_b > 6$: KeySchedule:

```
if (j mod Nk == 0) { //non-linear function
    W[0][j] = W[0][j-Nk] ⊕ S[W[1][j-1]] ⊕ RC[j/Nk];
    for (i=1; i<4; i++)
        W[i][j] = W[i][j-Nk] ⊕ S[W[i+1 mod 4][j-1]];
} else if (j mod Nk == 4) { //non-linear function
    for (i=0; i<4; i++) W[i][j] = W[i][j-Nk] ⊕ S[W[i][j-1]];
} else {
    for (i=0; i<4; i++)
        W[i][j] = W[i][j-Nk] ⊕ W[i][j-1];
}
```

- The bold elements highlight the cyclic rotation of the non-linear function (2^{nd} and 4^{th} line of `KeySchedule`), a rotation in the column all the N_k columns (when $j \pmod{N_k}$ is equal to 0).
- The RC at the 2^{nd} line of `KeySchedule` aims to break the symmetry.

Figure 3.5: KeySchedule (official specification [6])

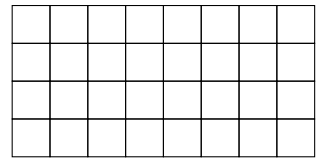


(a) 128 bits

(b) 192 bits

(c) 256 bits

Figure 3.6: Key Schedule schema for each key length



AES, Rijndael-128 (4 columns)

Rijndael-160 (5 columns)

Rijndael-192 (6 columns)

Rijndael-224 (7 columns)

Rijndael-256 (8 columns)

Figure 3.7: Differences between the AES and Rijndael

4 Differential cryptanalysis

In this section we present the differential cryptanalysis. Then, the related-key cryptanalysis, which is, as said in the introduction, a particular usage of cryptanalysis.

4.1 Main principle

4.1.1 Distinguisher

The idea is to find a relation, R, between an input state and an output one. Given an input, the output has a probability p of happening. The further the value is from the uniform probability, more one has this property of distinguishability which allows us to distinguish a random text from a cipher text.

4.1.2 Differential cryptanalysis

Idea In our case, we are interested in block ciphering: each block of the plaintext is ciphered, passing through a sequence of boxes, operations. Let t be a plaintext, known, and $c(t)$, the ciphering of t. The idea of the differential cryptanalysis is to guess $c(t')$, with the probability as high as possible, where $t \oplus t'$ is the difference between the two plaintexts.

Principle In fact, we have to track the evolution of the difference, $t \oplus t'$, called δt, during the ciphering process. The initial difference, $\delta t = \alpha$, is known, and we have to predict with the best probability $c(t) \oplus c(t') = \beta$.

In fact, tracking the difference over the ciphering process is deterministic when the operations are linear, with a probability equal to 1. However, it is not the case for non-linear operation, and we have to compute the probability to obtain an output given an input.

4.1.3 The S-Box in practice

As said above, for this non-linear operation, we have to compute the best probability to pass through the S-box. In fact, we count the number of inputs which satisfies a given output, as follows:
$D(\alpha, \beta) = \#\{S(t) \oplus S(t') = \beta | t \oplus t' = \alpha\}$.
Now, we search the best values of t, *i.e.*the values of t where we obtain $S(t) \oplus S(t') = \beta_{max}$, where β_{max} is the value of β which appears the most times. Actually, β_{max} is the output which has the highest probability of happening, that's why we keep the values of t which correspond to this case, with this probability which is maximum (number of occurrences of β_{max}).

After this step, we know the "best" choice for the S-box, and depending on this value, we can choose the best linear way to maximize the probability (*i.e.*as a consequence minimize the number of S-boxes on one hand, and choosing the best S-boxes on the other hand). Indeed, the final probability is the product of the probability at each step, with the constraint that the output of a step is the input of the following one.

4.2 Related key cryptanalysis

4.2.1 Overview

In this paragraph, $f_K(X)$ denotes the output ciphering process of the plaintext X with the key K.

Actually, this principle can be applied not only to the plaintext but also to the key. We can follow the same reasoning, given a text ciphered with a key K, we can have a difference not only in the plaintext but also in the key. In fact, we have introduced $\delta K = K \oplus K'$, and we search for an output difference $C \oplus C'$, where $C' = f_{K'}(X')$, in order to maximize the probability to obtain a certain output.

In other words, for any given X, K, we are looking for differential characteristics δX and δK such that we can predict with the best probability p a difference output $\delta C = C \oplus C'$.

The following subsection presents the related-key differential characteristics introduced in the case of the AES in [12].

4.2.2 Rijndael related-key differential characteristics

To mount related-key attacks, we track differences through the ciphering process, where differences are obtained by applying the XOR operator. We note δA the *differential matrix* obtained by applying the XOR operator on two matrices A and A', and for every round i, row j, and column k, $\delta A[i][j][k] = A[i][j][k] \oplus A'[i][j][k]$ is called a *differential byte*. The set of all differential bytes is denoted $diffBytes_l$. Among these differential bytes, some of them pass through S-boxes, and we note $Sboxes_l$ this set. When the key length is $l = 128$, these two sets are defined by:

$$\begin{aligned} diffBytes_{128} &= \{\delta X[i][j][k], \delta X_r[i][j][k], \delta K_r[i][j][k] : i \in [0, r], j, k \in [0, 3]\} \\ &\cup \{\delta K_i[i][j][k], \delta SK_i[i][j][3], \delta X_i[i][j][k], \delta SX_i[i][j][k], \\ &\quad \delta Y_i[i][j][k], \delta Z_i[i][j][k] : i \in [0, r-1], j, k \in [0, 3]\} \\ Sboxes_{128} &= \{\delta K_i[i][j][3], \delta X_i[i][j][k] : i \in [0, r-1], j, k \in [0, 3]\} \end{aligned}$$

This definition is in the case of the AES-128 (key length = 128), we will precise this reasoning in all cases (other cases of the AES and Rijndael) in 6.
The goal is to find an optimal related-key differential characteristics, *i.e.*, a byte value for every differential byte in $diffBytes_l$ such that the probability of observing δX_r given δX and δK_0 is maximal.
This problem would be easy to solve (and the probability would be equal to 1) if every operation applied during the ciphering process were linear. However, AES is composed of three linear operations (SR, MC, and ARK), and one non linear operation (SB). Moreover, the key schedule KS combines linear XOR operations with the non linear SB operation.

Propagation of differences by the linear operations The linear operators only move differences to other places, and we can deterministically compute the output difference of a linear operator given its input difference. For SR and MC, given two matrices A_1 and A_2, we have $SR(A_1) \oplus SR(A_2) = SR(A_1 \oplus A_2)$ and $MC(A_1) \oplus MC(A_2) = MC(A_1 \oplus A_2)$. This implies that the output difference of SR is $\delta Y_i = SR(\delta SX_i)$ when the input difference is δSX_i, and the output difference of MC is $\delta Z_i = MC(\delta Y_i)$ when the input difference is δY_i. Similarly, given four matrices A_1, A_2, A_3, and A_4, we have $ARK(A_1, A_2) \oplus ARK(A_3, A_4) = ARK(A_1 \oplus A_3, A_2 \oplus A_4)$. Therefore, the output difference of ARK is $\delta X_{i+1} = ARK(\delta Z_i, \delta K_{i+1})$ when the input differences are δZ_i and δK_{i+1}.

Propagation of differences by the non-linear operation SB This property no longer holds for SB which applies to a byte B an S-box transformation $S(B)$ such that, given two bytes B and B', $S(B \oplus B')$ is not necessarily equal to $S(B) \oplus S(B')$. Therefore, given an input differential byte δB_{in}, we cannot deterministically compute the output difference after passing through S-boxes. We can only compute probabilities. More precisely, for every couple of differential bytes $(\delta B_{in}, \delta B_{out}) \in [0, 255]^2$, we can compute the probability that the input difference δB_{in} becomes the output difference δB_{out}, which is the proportion of byte couples (B, B') such that $\delta B_{in} = B \oplus B'$ and $\delta B_{out} = S(B) \oplus S(B')$. More precisely, this probability is denoted $p_S(\delta B_{out} | \delta B_{in})$ and is defined by

$$p_S(\delta B_{out} | \delta B_{in}) = \frac{\#\{(B, B') \in [0, 255]^2 \mid (B \oplus B' = \delta B_{in}) \wedge (S(B) \oplus S(B') = \delta B_{out})\}}{256}$$

For the AES S-box, most of the times the probability p_S is equal to $\frac{0}{256}$ or $\frac{2}{256}$, and rarely to $\frac{4}{256}$ [6]. The only case where p_S is equal to 1 is when there is no difference, *i.e.*, $p_S(0|0) = 1$: There is no difference in the output ($\delta B_{out} = 0$) if and only if there is no difference in the input ($\delta B_{in} = 0$), since the S-Box is bijective.

Probability of a valuation of differential bytes To compute the probability of a given valuation of all differential bytes, we first have to check that all linear operators are satisfied by the valuation. If this is not the case, then the probability is equal to 0. Otherwise it is equal to the product of the transition probabilities of all differential bytes that pass through S-boxes, *i.e.*,

$$p = \prod_{\delta B \in Sboxes_l} p_S(\delta SB | \delta B) \tag{4.1}$$

We refer the reader to [3] for more details on differential characteristics.

5 Constraints programming

As said above, the way used to explore and find the solutions is constraints programming. Indeed, main approaches used in the state of the art are Integer Linear Programming (ILP), Boolean SATisfiability (SAT), and Constraints Programming (CP).
For some efficiency reasons after experience, the last one is the more convenient in our case, and assuming this conclusion, let's present the basic principles of CP, as reminded in [12]. We refer the reader to [22] for more details.

5.1 Principle

CP is used to Constraint Satisfaction Problems (CSPs). A CSP is defined by a triple (X, D, C) such that X is a finite set of variables, D is a function that maps every variable $x_i \in X$ to its domain $D(x_i)$ (that is, the finite set of values that may be assigned to x_i), and C is a set of constraints (that is, relations between some variables which restrict the set of values that may be assigned simultaneously to these variables).
Constraints may be defined in extension, by listing all allowed (or forbidden) tuples of the relation, or in intention, by using mathematical operators. Let us consider for example a CSP with $X = \{x_1, x_2, x_3\}$ such that $D(x_1) = D(x_2) = D(x_3) = \{0, 1\}$, and let us consider a constraint that ensures that the sum of the variables in X is different from 1. This constraint may be defined by a table constraint that enumerates all allowed tuples:

$$(x_1, x_2, x_3) \in \{(0, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 0), (1, 1, 1)\}.$$

Conversely, it may be defined by enumerating all forbidden tuples:

$$(x_1, x_2, x_3) \notin \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}.$$

Finally, it may be defined by using arithmetic operators: $x_1 + x_2 + x_3 \neq 1$.
Solving a CSP involves assigning values to variables such that all constraints are satisfied. More formally, an *assignment* $\mathcal{A}$ is a function which maps each variable $x_i \in X$ to a value $\mathcal{A}(x_i) \in D(x_i)$. An assignment $\mathcal{A}$ *satisfies* (resp. *violates*) a constraint $c \in C$ if the tuple defined by the values assigned to the variables of c in $\mathcal{A}$ belongs (resp. does not belong) to the relation defined by c. An assignment is *consistent* (resp. *inconsistent*) if it satisfies all the constraints (resp. violates some constraints) of the CSP. A *solution of a CSP* is a consistent assignment.
An objective function may be added to a CSP, thus defining a Constrained Optimization Problem (COP). This objective function is defined on some variables of X and the goal is to find the solution that optimizes (minimizes or maximizes) the objective function.

5.2 Solving CP problems

CP languages provide high-level features to define CSPs and COPs in a declarative way. These languages are either integrated to CP libraries (such as, for example, Choco [21], or Gecode [11]),

or independent from any library (such as, for example, MiniZinc [18]). In this last case, Then, these problems are solved by generic constraint solvers which are usually based on a systematic exploration of the search space: Starting from an empty assignment, they incrementally extend a partial consistent assignment by choosing a non-assigned variable and a consistent value for it until either the current assignment is complete (a solution has been found) or the current assignment cannot be extended without violating constraints (the search must backtrack to a previous choice point and try another extension). To reduce the search space, this exhaustive exploration of the search space is combined with constraint propagation techniques: Each time a variable is assigned to a value, constraints are propagated to filter the domains of the variables that are not yet assigned, *i.e.*, to remove values that are not consistent with respect to the current assignment. When constraint propagation removes all values from a domain, the search must backtrack.
Different levels of constraint propagation may be considered; some enforce stronger consistencies than others, *i.e.*, they remove more values from the domains. A widely used propagation level enforces Generalized Arc Consistency (GAC): a constraint c is GAC if for each variable x_i of c and each value $v_i \in D(x_i)$, there exists a tuple t in the cartesian product of the domains of the variables of c such that t satisfies c and x_i is assigned to v_i in t (t is called a *support* of $x_i = v_i$).
Let us consider for example the constraint $x_1 + x_2 + x_3 \neq 1$. If $D(x_1) = D(x_2) = \{0\}$ and $D(x_3) = \{0, 1\}$, then the constraint is not GAC because there is no support of $x_3 = 1$ (the only tuple in $D(x_1) \times D(x_2) \times D(x_3)$ that satisfies the constraint is $(0, 0, 0)$). Hence, to enforce GAC, we must remove 1 from $D(x_3)$.
A key point to speed up the solving process of CSPs is to define the order in which variables are assigned, and the order in which values are assigned to these variables, when building the search tree. CP languages allow the user to specify this through variable and value ordering heuristics.

Remark on XOR constraint Actually, the XOR constraint is implemented thanks to the the following constraint: $\text{XOR}(a_0, ..., a_n)$, *i.e.*$a \oplus ... \oplus a_n = 0$, where a_i are boolean variables. It is equivalent to $a_0 + ... + a_1 \neq 1$. Indeed, that is trivial in the case $n = 2$ or $n = 3$, and recursively it can be proved pour any $n > 3$.

Part III

Contributions

6 Modeling

In this chapter, the following considerations have been studied in the case of the AES, but the reasoning is independent of the value of the N_b parameter. That's why we present first the work already done on the AES, because we dispose of the conclusions before trying to generalize to Rijndael. Then, later, another chapter will present the work done during the internship, the same reasoning on Rijdnael, given this work on the AES. For this part on the AES, we cite the work presented in [12].

6.1 Two-step model

The three approaches described in [4], [10], and [14] compute optimal differential characteristics in two steps.

Step 1: Computation of truncated differentials Abstraction of differential bytes with Boolean values

In a first step, each differential byte $\delta B \in \mathit{diffBytes}_l$ is abstracted with a Boolean variable ΔB such that $\Delta B = 0 \Leftrightarrow \delta B = 0$ and $\Delta B = 1 \Leftrightarrow \delta B \in [1, 255]$. In other words, each Boolean variable assigned to 1 gives the position of a difference. The goal is to find all truncated differential characteristics (*i.e.*, all Boolean solutions) that minimize the number of differences passing through S-boxes (*i.e.*, that minimize $\sum_{\delta B \in \mathit{Sboxes}_l} \Delta B$). We say that an S-box $\delta B \in \mathit{Sboxes}_l$ is *active* whenever there is a difference passing through it, *i.e.*, $\Delta B = 1$.

As linear operators are mostly defined by means of XOR operations, condition (i) mainly involves checking a boolean abstraction of XOR: given two boolean abstractions ΔB_1 and ΔB_2 of two differential bytes δB_1 and δB_2, respectively, we check that whenever $\Delta B_1 = 0$ (resp. $\Delta B_2 = 0$), then $\Delta B_1 \oplus \Delta B_2 = \Delta B_2$ (resp. $\Delta B_1 \oplus \Delta B_2 = \Delta B_2$). However, whenever $\Delta B_1 = \Delta B_2 = 1$, we cannot know whether $\Delta B_1 \oplus \Delta B_2$ is equal to 0 or 1 (as $\delta B_1 \oplus \delta B_2$ is equal to 0 if $\delta B_1 = \delta B_2$, and different from 0 otherwise).

In a second step, for each truncated differential characteristic, we search for actual byte values that satisfy the AES operations and that maximize the probability p defined in Eq.(4.1). When a Boolean variable ΔB is equal to 0 in the truncated differential characteristic, there is only one possible value for δB, which is 0. However, when $\Delta B = 1$, there are 255 possible values for δB.

Note that some truncated differential characteristics are not valid and cannot be transformed into byte solutions because truncated differential characteristics only satisfy Boolean abstractions of the actual AES operations. These characteristics are said to be *byte-inconsistent*. It may also happen that the maximal probability p is such that it is possible to have a greater probability with a larger number of active S-boxes (*i.e.*, when the probability p is smaller than $2^{-6(v+1)}$ where v is

the number of active S-boxes in the truncated differential). In this case, we need to search for new truncated differentials, with a larger number of active S-boxes.

Algorithm 1: Computation of optimal differential characteristics

Input: The size l of the key and the number r of rounds
Output: An optimal differential characteristic c^*

```
1  begin
2      v* ← Step1-opt(l, r)
3      v ← v*
4      c* ← null
5      repeat
6          T ← Step1-enum(l, r, v)
7          for each truncated differential characteristic t ∈ T do
8              c ← Step2(l, r, t)
9              if c ≠ null and (c* = null or p(c) > p(c*)) then c* ← c;
10         v ← v + 1
11     until c* ≠ null and p(c*) ≥ 2^{-6v};
12     return c*
```

The complete procedure to find optimal differential characteristics is described in Algorithm 1. It first calls function *Step1-opt* to compute the minimal number v^* of active S-boxes in a truncated differential characteristic (line 2), and it initializes v to this minimal number of active S-boxes. Then, it calls function *Step1-enum* to compute the set T of all truncated differential characteristics such that the number of active S-boxes is equal to v (line 6). For each truncated differential characteristic $t \in T$, it calls function *Step2* (line 8): if t is not byte-consistent, *Step2* returns *null*; otherwise, it returns the optimal differential characteristic c associated with t. If this optimal differential characteristic has a greater probability than c^*, then it updates c^* (line 9). Hence, when exiting from the loop lines 7-9, if c^* is equal to *null* (because all truncated differential characteristics in T are byte-inconsistent), we have to increment v and iterate again on lines 6 to 10; otherwise, $p(c^*)$ is the largest probability with v active S-boxes and we have $2^{-7v} \leq p(c^*) \leq 2^{-6v}$ (because each S-box has a probability which is equal to 2^{-7} or 2^{-6}). However, it may be possible that $p(c^*)$ is not maximal: there may exist a solution with higher probability with $v + 1$ active S-boxes if $p(c^*) < 2^{-6(v+1)}$. In this case, we have to increment v and iterate again on lines 6 to 10 to check whether there exists a higher probability with one more active S-box. Hence, lines 6 to 10 are repeated with increasing values of v until a differential characteristics has been found, and it is not possible to obtain a better differential characteristics with a higher value of v.

In this paper, we describe CP models for implementing functions *Step1-opt*, *Step1-enum*, and *Step2*. *Step1-opt* and *Step1-enum* use the same model which is described in Sections 6.2 and 6.3.

From differential characteristics to related-key attacks Let us first clarify the relation between the differential characteristic $c*$ computed by Algo. 1 and an optimal differential. What cryptanalysts would like to have is an optimal differential, *i.e.*, input and output differences that maximize the probability of observing the output difference given the input difference. Differential characteristics have been introduced to simplify the problem by specifying all intermediate differences (after each step of the ciphering process). Several differential characteristics may have the same input and output differences and only differ on intermediate differences. Hence, the probability of a differential is the sum of the probabilities of all differential characteristics that share its input and output differences. Thus, finding the differential characteristic with the

best probability gives us a lower bound on the expected differential probability. This notion of expected differential probability evaluates the average behavior of the differential taken on average on all the key space, assuming the independence of intermediate probabilities, the stochastic equivalence of the keys and other simplifying assumptions such as those coming from the related-key setting, for example.

Algo. 1 computes a differential characteristic which maximizes the probability p of observing the output difference δX_r on the ciphertexts given the input difference δX in the plaintexts (and all the intermediate difference values δX_i for i from 0 to $r-1$) and the input difference δK in keys. In this section, we give the intuition of how this optimal differential characteristic can be used in a related-key attack or to distinguish the AES from a random permutation.

As stated in the introduction, finding the exact probability of a differential with input difference δX and output difference δX_r (in the single key setting or in the related-key setting) is a hard task that does not scale well. However, finding the best differential characteristic is easily done by analyzing the transitions of simpler steps (and assuming their independence). It gives us a lower bound on the expected differential probability. This notion of expected differential probability evaluates the average behavior of the differential taken on average on all the key space, assuming the independence of intermediate probabilities, the stochastic equivalence of the keys and other simplifying assumptions coming from the related-key setting for example.

In summary, the probability of the differential characteristic computed by Algo. 1 (denoted p) is an approximation for a lower bound of the expected differential probability.

After this clarification of the p definition, let us now explain how we could use this differential characteristic to construct first a so-called distinguisher and from it a related-key differential attack. Suppose that an attacker could query a particular machine (the so-called oracle) that implements whether r rounds of the *AES* parametrized by an unkonwn and randomly chosen key K or second a random permutation. Then the attacker randomly chooses a set $S = \{X_1, \ldots, X_m\}$ of m plaintexts. She asks the oracle to cipher those m plaintexts under the key K: $C_i = Enc_K(X_i)$ and also the associated plaintexts $X'_i = X_i \oplus \delta X$ for each plaintext $X_i \in S$ under the key $K' = K \oplus \delta K$, ($C'_i = Enc_{K'}(X'_i)$ the result of ciphering X'_i with key K'). As the attacker has found a differential characteristic $(\delta X, \cdots, \delta X_r)$ of expected probability equal to p with $p \gg 2^{-n}$ for r rounds of the AES, she can expect, if the oracle implements the AES as the *Enc* function, to find a pair with the right output difference (given the input difference) after trying roughly $m = 1/p \ll 2^n$ random input pairs, which is much sooner than expected for a randomly selected permutation. Note that if the *Enc* function is a random permutation, this property will not hold. Thus with the differential characteristic of probability p the attacker is able to know if the *Enc* function of the oracle is r rounds of the AES or a random permutation.

As a motivating example, consider a device used by spies to encrypt messages. This device, given a message m, encrypts it with a secret key K and returns the corresponding ciphertext. The key K is protected: it cannot be read, even by disassembling the device. However, a flaw in the device permits to input, in addition to m, a key mask δK, to obtain the encryption of m with the key $K' = K \oplus \delta K$. Now assume such a device is captured. We would like to extract the key K in order to decrypt previously captured messages encrypted using this device. To this end, we can apply a related-key differential attack. However, the computational effort required to recover K might be too high. In this case, alternatively, we might be interested in at least knowing which encryption algorithm is used by this device: we can decides whether it uses a r round version of AES, by applying a related-key distinguishing attack.

The intuition of these attacks is as follows: if we use the device to encrypt enough pairs of messages with input differences $\delta X, \delta K$, and the encryption scheme is AES, we will observe the output difference δX_r more often than any other output difference, thus successfully distinguishing the cipher. Additionally, the difference at round $r-1$ will be more likely to be δX_{r-1}. Therefore, we can validate a partial subkey guess by partially deciphering one round of obtained ciphertext with it: a correct guess will result in the correct difference more often than an incorrect guess, and the rest of the key can be determined by exhaustive search.

We first use the device to encrypt a given number m of message pairs X, X', where $X' = X \oplus deltaX$: X is encrypted with K, and X' is encrypted with $K' = K \oplus \delta K$. Consider an oracle that performs AES encryptions on a given number of rounds with a key K unknown to the attacker. Given a plaintext X, this oracle returns $AES_K(X)$. Given a plaintext X and a differential matrix δK, it returns $AES_{K'}(X)$ where $K' = K \oplus \delta K$ is the key obtained by XORing the secret key K with δK. In a related-key differential attack, we are interested in recovering K. Alternatively, as a weaker goal, we can try to determine whether the oracle correctly performs AES encryptions under a fixed (unknown) key K, or applies a random permutation instead. The latter attack is called a distinguishing attack.

More formally, the first step of the attack is to randomly pick a set $S = \{X_1, \ldots, X_m\}$ of m plaintexts, and use the device to cipher them. We note $C_i = AES_K(X_i)$ the ciphertext obtained for $X_i \in S$. For each plaintext $X_i \in S$, we also compute the plaintext $X'_i = X_i \oplus \delta X$, and use the device to cipher X'_i under the key $K' = K \oplus \delta K$. We note $C'_i = AES_{K'}(X'_i)$ the obtained ciphertext.

First, consider that you have access to an oracle that implements first, the *AES* parametrized by an unkonwn and randomly chosen key K and second a random permutation. Then the attacker choose a set $S = \{X_1, \ldots, X_m\}$ of m plaintexts. Then, he/she asks the oracle to cipher those m plaintexts under the key K: $C_i = AES_K(X_i)$ and the associated plaintexts $X'_i = X_i \oplus \delta X$ for each plaintext $X_i \in S$ under the key $K' = K \oplus \delta K$, ($C'_i = AES_{K'}(X'_i)$ the result of ciphering X'_i with key K'). The main goal of the attacker is thus to be able with those data to distinguish the *AES* from the random permutation.

By definition, the probability p associated with the differential characteristic $(\delta X, \delta K, \delta X_r)$ is the probability that $C_i \oplus C'_i = \delta X_r$, for any $X_i \in S$. Therefore, the expected number of plaintexts $X_i \in S$ for which $C_i \oplus C'_i = \delta X_r$ is equal to mp. If m is in $O(1/p)$, this expected number is in $O(1)$ [3, 2]. If p is greater than 2^{-l}, then the complexity of a distinguishing attack is $O(1/p)$.

If the number of rounds used by the device is $r+1$, a r round distinguisher can be used to recover some bits of the subkey K_{r+1} (usually, the recovered bits are those implied in subkey active S-boxes). This is done by testing all possible values for those bits of K_{r+1} and by deciphering the round $r+1$ according to these values: The good values are those for which the expected output difference δX_r appears the most often.

From a theoretical point of view, it means that if $O(1/p)$ is lower than the cost of the key exhaustive search, we are able to distinguish the cipher (here the AES) from a random permutation using the exhibited property. Thus, if we cipher a sufficient number m of plaintext pairs $(X, X') = (X, X \oplus \delta X)$ under the two different keys K and $K' = K \oplus \delta K$ such that the corresponding pairs of ciphertexts are $C = AES_K(X)$ and $C' = AES_{K'}(X')$, we expect than at least one output pair will have a difference of the form $C \oplus C' = \delta X_r$. To be sure that such a willing output difference occurs, m must be of the form $O(1/p)$ [3, 2].

Thus, we have built a so-called distinguisher on r rounds, *i.e.*, we have exhibited a particular property that distinguishes a random permutation from r AES rounds with a complexity equal to $\mathcal{O}(1/p)$. From this distinguisher, we are able to mount a related-key differential attack on $r+1$ rounds adding a round at the end by recovering some key bits of the subkey K_{r+1}. Indeed, by testing all the possible values for some bits of the subkey K_{r+1} and by deciphering the round $r+1$ according the hypothesis of the subkey K_{r+1} value, the right key value will correspond to the key for which the expected output difference δX_r will appear the most often. The wrong subkey values will behave as random. Given this differential characteristic, an attacker can mount a related-key attack by ciphering M plaintext pairs (T, T') to obtain M ciphered pairs (C, C'), where M is equal to c/p_1 for c a small constant. 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 secret key is the one for which the optimal value of δX_r (that maximizes the probability defined in Eq. 4.1) appears the most frequently.

We refer the reader to [3] for more details on related-key attacks.

Let us now explain how the differences δX, δK, and δX_r of the differential characteristic $c*$ computed by Algo. 1 may be used to construct a so-called distinguisher. To this aim, let us assume that we can query an oracle: given a plaintext X, this oracle returns $C = Enc_K(X)$ such that C is either the result of ciphering X by r rounds of the AES with an unknown and randomly chosen key K, or it is a random permutation.
The attacker randomly chooses a set $S = \{X^1, \ldots, X^m\}$ of m plaintexts. For each plaintext $X^i \in S$, she asks the oracle to compute $C^i = Enc_K(X^i)$ and $C'^i = Enc_{K'}(X'^i)$ where $X'^i = X^i \oplus \delta X$, and $K' = K \oplus \delta K$.
If the oracle implements the AES, then she can expect to find a pair (X^i, X'^i) such that $C^i \oplus C'^i = \delta X_r$ after trying roughly $1/p$ random input pairs. As p is much larger than 2^{-n}, this is much sooner than expected for a randomly selected permutation. In other words, the differential characteristic $c*$ allows the attacker to know if the *Enc* function of the oracle is r rounds of the AES or a random permutation: we have built a distinguisher on r rounds, *i.e.*, we have exhibited a particular property that distinguishes a random permutation from the AES with a complexity equal to $\mathcal{O}(1/p)$.

Finally, this distinguisher may be used to mount a related-key differential attack on $r+1$ rounds. This last step is difficult to explain in a few lines. The basic idea is to add an extra round at the end and to recover some key bits of the subkey K_{r+1} by exploiting the differences in the ciphertexts. Indeed, by testing all the possible values for some bits of the subkey K_{r+1} and by deciphering the round $r+1$ according to the hypothesis of the subkey K_{r+1} value, the right key value will correspond to the key for which the expected output difference δX_r will appear the most often. The wrong subkey values will behave as random. We refer the reader to [3] for more details on related-key attacks.

Notations In the two following sections (sec. 6.2 and sec. 6.3), i denotes the round number, j the row number, and k the column number. Let be δA a variable, $A \in \{X, Y, Z, K\}$, $\delta A_i[j][k]$ is the value of δA at the i^{th} round, at row j and column k. $\delta A[i][j][k]$ denotes the same value. For state values $(\delta X, \delta Y, \delta Z)$, the dimensions are: $i \in [0, N_r]$, $j \in [0, 3]$, $k \in [0, N_b]$. In the case of the AES, $N_b = 3$. For the key (δK), $k \in [N_k]$.
However, in practical implementation, depending on models used, an other way can be used to represent these variables, developing the variable, expanding it for all rounds: $\delta A[j][k], k \in$

$[0, N_r * N_b]$ (or $\delta K[j][k], k \in [0, N_r * N_k]$ in the case of the key). Nevertheless, the first notation will be used in this document.

6.2 Basic CP model for Step 1

Functions *Step1-opt* and *Step1-enum* used in Algorithm 1 for computing optimal differential characteristics share the same CP model. The only difference is in the goal of the solving process: in *Step1-opt* the goal is to search for a solution that optimizes a given variable called obj_{Step1}, whereas in *Step1-enum* the goal is to enumerate all solutions when the variable obj_{Step1} is assigned to a given value.
In this section, we describe a first CP model for *Step1-opt* and *Step1-enum*, called CP_{Basic}, which has been introduced in [16] and is derived in a straightforward way from the definition of the AES operations.

6.2.1 Variables of CP_{Basic}

CP_{Basic} does not associate a Boolean variable with every differential byte $\delta B \in diffBytes_l$. Indeed, during Step 2, the initial differential plaintext δX, the last round differential subkey δK_r, and the final differential ciphertext δX_r can be deterministically computed given the values of all other differential bytes:

- δX is obtained by XORing δX_0 and δK_0.
- δK_r is obtained by applying the key schedule to δK_{r-1} for AES-128, and to δK_{r-1} and δK_{r-2} for AES-192 and AES-256. Note that for the bytes δB that pass through S-boxes during this last round, we deterministically choose for δSB the value that maximizes $p_S(\delta SB|\delta B)$.
- δX_r is obtained by XORing δZ_{r-1} and δK_r.

Hence, CP_{Basic} associates a Boolean variable ΔB with every differential byte δB, $\delta B \in diffBytes_l \setminus \{\delta X[j][k], \delta K_r[j][k], \delta X_r[j][k] : j, k \in [0, 3]\}$. Each Boolean variable ΔB is assigned to 0 if $\delta B = 0$, and to 1 otherwise.
We also define an integer variable obj_{Step1} which corresponds to the number of active S-boxes. The domain of this variable is $D(obj_{Step1}) = [1, \frac{l}{6}]$. Indeed, the smallest possible value is 1 because we need to have at least one active S-box to have a differential characteristic (we forbid the obvious solution such that δX and δK only contain bytes set to 0, meaning that there is no difference in the initial plaintext and key). The largest possible value is $\frac{l}{6}$ because the highest probability $p_S(\delta_{out}|\delta_{in})$ to pass through the AES S-box is $2^{-6} = \frac{4}{256}$ when $\delta_{in} \neq 0$. Therefore, the greatest probability of a differential characteristic with obj_{Step1} active S-boxes is $2^{-6*obj_{Step1}}$. As we want a differential characteristic which is more efficient than the key exhaustive search, its probability must be greater than 2^{-l} (as explained in Section 12) and, therefore, obj_{Step1} must not be larger than $\frac{l}{6}$.

6.2.2 Definition of a XOR constraint

As many AES transformations use the XOR operator, we define a constraint to model it at the Boolean level.
During Step 2, when reasoning at the byte level, the XOR operator is applied on each bit of each

(C_1) $obj_{Step1} = \sum_{\delta B \in Sboxes_l} \Delta B$

(C_2) $\forall \delta B \in Sboxes_l, \Delta SB = \Delta B$

(C_3) $\forall i \in [0, r-2], \forall j, k \in [0,3], XOR(\Delta Z_i[j][k], \Delta K_{i+1}[j][k], \Delta X_{i+1}[j][k])$

(C_4) $\forall i \in [0, r-1], \forall j, k \in [0,3], \Delta Y_i[j][k] = \Delta SX_i[j][(j+k)\%4]$

(C_5) $\forall i \in [0, r-2], \forall k \in [0,3], \left(\sum_{j=0}^{3} \Delta Y_i[j][k] + \Delta Z_i[j][k]\right) \in \{0,5,6,7,8\}$

(C_6) $\forall j, k \in [0,3], \Delta Z_{r-1}[j][k] = \Delta Y_{r-1}[j][k]$

(C_7) $\forall i \in [0, r-1], \forall j \in [0,3],$
$XOR(\Delta K_{i+1}[j][0], \Delta K_i[j][0], \Delta SK_i[(j+1)\%4][3])$

(C_8) $\forall i \in [0, r-1], \forall j \in [0,3], \forall k \in [1,3],$
$XOR(\Delta K_{i+1}[j][k], \Delta K_{i+1}[j][k-1], \Delta K_i[j][k])$

Figure 6.1: Constraints of the CP_{Basic} model for Step 1 of AES-128

byte using the following table:

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

However, during Step 1, each byte is abstracted by a Boolean value indicating whether the byte is equal to 0 or not, and we must consider an abstraction of this operator. More precisely, let us consider three differential bytes δB_1, δB_2 and δB_3 such that $\delta B_1 \oplus \delta B_2 = \delta B_3$ (or equivalently, $\delta B_1 \oplus \delta B_2 \oplus \delta B_3 = 0$). If two of these bytes are equal to 0, then we know for sure that the third one is also equal to 0 (because $0 \oplus 0 = 0$). Also, if one differential byte is equal to 0 and a second one is different from 0, then we know for sure that the third one is different from 0 (because $0 \oplus \delta B_i = \delta B_i$). However, when two differential bytes are different from 0, then we cannot know if the third one is equal to 0 or not (because if the two bytes have the same value then the third one is equal to 0, whereas if they have different values the third one is different from 0). Hence, when abstracting differential bytes δB_1, δB_2 and δB_3 with Boolean variables ΔB_1, ΔB_2 and ΔB_3 (which only model the fact that there is a difference or not), we obtain the following definition of the XOR constraint:

$$XOR(\Delta B_1, \Delta B_2, \Delta B_3) \Leftrightarrow \Delta B_1 + \Delta B_2 + \Delta B_3 \neq 1$$

where + stands for the integer addition. In other words, either the three variables are false (no difference), or at least two variables are true (at least two differences).

6.2.3 Constraints of CP_{Basic}

The constraints of CP_{Basic} are listed in Fig. 6.1 and are described below.

Number of active S-boxes Constraint (C_1) ensures that obj_{Step1} is equal to the number of active S-boxes, *i.e.*, the number of ΔB variables that are associated with differential bytes in $Sboxes_l$ and that are assigned to 1.

SubBytes Constraint (C_2) ensures that there is a difference in the output differential byte of an S-box iff there is a difference in the input differential byte. Indeed, SB has no effect on the presence/absence of differences during Step 1 because the S-box is bijective and $(B \oplus B' \neq 0) \Leftrightarrow (S(B) \oplus S(B') \neq 0)$.

AddRoundKey ARK is modeled by the XOR constraint (C_3).

ShiftRows SR is modeled by the equality constraint (C_4) that simply links the shifted bytes.

MixColumns MC cannot be modeled at the Boolean level, as knowing where differences hold in Y_i is not enough to determine where they hold in Z_i. However, the MDS property is modeled by constraint (C_5): It ensures that the number of differences in a same column of Y_i and Z_i is equal to 0 or greater than 4. Constraint (C_6) models the fact that MC is not applied during the last round.

KeySchedule KS is modeled by XOR constraints (combined with a rotation of the bytes of SK for some columns). When $l = 128$, this corresponds to constraints (C_7) and (C_8). The constant c_i which is XORed with $SK_i[1][3]$ and $K_i[0][0]$ when defining $K_{i+1}[0][0]$ does not appear in ($C7$): it is canceled when XORing $K_{i+1}[0][0]$ with $K'_{i+1}[0][0]$ to define $\delta K_{i+1}[0][0]$. Indeed, we have:

$$\begin{aligned} \delta K_{i+1}[0][0] &= K_{i+1}[0][0] \oplus K'_{i+1}[0][0] \\ &= SK_i[1][3] \oplus K_i[0][0] \oplus c_i \oplus SK'_i[1][3] \oplus K'_i[0][0] \oplus c_i \\ &= \delta SK_i[1][3] \oplus \delta K_i[0][0] \end{aligned}$$

6.2.4 Goal

The same model is used to solve two problems, *i.e.*, *Step1-opt* and *Step1-enum*. The difference between these problems is in their goals:

- For *Step1-opt*, the goal is to find the minimum number of active S-boxes, *i.e.*, minimize obj_{Step1};
- For *Step1-enum*, the goal is to enumerate all solutions when obj_{Step1} is assigned to a given value v. Each solution corresponds to a truncated differential characteristic with v active S-boxes.

6.2.5 Ordering heuristics

As the goal is to minimize the number of active S-boxes (for *Step1-opt*) or enumerate all solutions with a small number of active S-boxes (for *Step1-enum*), we define ordering heuristics as follows: first assign variables associated with bytes that pass through S-boxes (those in $Sboxes_l$), and first try to assign them to 0.

6.2.6 Performance of CP_{Basic}

CP_{Basic} is complete in the sense that for any solution at the byte level (on δ variables), there exists a solution of CP_{Basic} at the Boolean level (on Δ variables). However, experiments reported in [14] have shown us that there is a huge number of solutions of CP_{Basic} which are byte inconsistent and do not correspond to solutions at the byte level. For example, for AES-128, when the number of rounds is $r = 3$, the optimal solution of *Step1-opt* has $obj_{Step1} = 3$ active S-boxes, and *Step1-enum*

enumerates more than five hundred truncated differential characteristics with three active S-boxes. However, none of them is byte-consistent. Actually, the optimal byte-consistent truncated differential characteristic has five active S-boxes. In this case, most of the solving time is spent at generating useless truncated differential characteristics which are discarded in Step 2.

6.3 The CP-XOR model

A weakness of CP_{Basic} comes from the fact that the XOR constraint between Boolean variables is a poor abstraction of the XOR relation at the byte level, because whenever two XORed bytes are different from zero, we cannot know whether the result is equal to zero or not. In Section 6.3.1, we show how to tighten this abstraction by generating new XOR equations, obtained by combining initial equations coming from the key schedule. In Section 6.3.2, we describe the variables of CP_{XOR}, and we introduce new Boolean variables that model differences at the byte level. In Section 6.3.3, we describe the constraints of CP_{XOR}, and we show how to tighten the definition of the AES operations at the Boolean level by reasoning on differences at the byte level. In Section 6.3.4 we discuss some implementation issues.

6.3.1 Generation of new XORs

Every subkey differential byte $\delta K_i[j][k]$ either comes from the initial differential key δK, or is obtained by XORing two differential bytes according to the key schedule rules. Hence, the whole key schedule implies a set of $16 * (r-1)$ (resp. $16 * (r-1) - 8$ and $16 * (r-2)$) XOR equations for AES-128 (resp. AES-192 and AES-256), where each of these equations involves three differential bytes. We propose to combine these initial equations to infer new equations.
Let us consider, for example, the three equations that define $\delta K_1[0][3]$, $\delta K_2[0][2]$ and $\delta K_2[0][3]$, respectively, for AES-128:

$$\delta K_0[0][3] \oplus \delta K_1[0][2] \oplus \delta K_1[0][3] = 0 \quad (6.1)$$
$$\delta K_1[0][2] \oplus \delta K_2[0][1] \oplus \delta K_2[0][2] = 0 \quad (6.2)$$
$$\delta K_1[0][3] \oplus \delta K_2[0][2] \oplus \delta K_2[0][3] = 0 \quad (6.3)$$

These equations share bytes: $\delta K_1[0][2]$ for Eq. (6.1) and (6.2), $\delta K_1[0][3]$ for Eq. (6.1) and (6.3), and $\delta K_2[0][2]$ for Eq. (6.2) and (6.3). We may combine Eq. (6.1), (6.2), and (6.3) by XORing them, and exploit the fact that $B \oplus B = 0$ for any byte B, to generate the following equation:

$$\delta K_0[0][3] \oplus \delta K_2[0][1] \oplus \delta K_2[0][3] = 0 \quad (6.4)$$

This new equation is redundant at the byte level, as $(6.1) \wedge (6.2) \wedge (6.3) \Rightarrow (6.4)$. However, at the Boolean level, the propagation of the XOR constraint corresponding to Eq. (6.4) detects inconsistencies which are not detected when only considering the XOR constraints corresponding to Eq. (6.1), (6.2), and (6.3). Let us consider, for example, the case where $\Delta K_0[0][3] = 1$, $\Delta K_2[0][1] = \Delta K_2[0][3] = 0$, and all other Boolean variables still have 0 and 1 in their domains. In this case, the propagation of XOR constraints associated with Eq. (6.1), (6.2), and (6.3) does not detect an inconsistency as only one variable is assigned for each constraint, and for the two other variables, we can always choose values such that the sum is different from 1. However, at the byte level, $\delta K_0[0][3]$ cannot be assigned to a value different from 0 when $\delta K_2[0][1] = \delta K_2[0][3] = 0$. Indeed, in this case, Eq. (6.2) and (6.3) imply:

$$\delta K_1[0][2] \oplus \delta K_2[0][2] = \delta K_1[0][3] \oplus \delta K_2[0][2] = 0$$
$$\Rightarrow \quad \delta K_1[0][2] = \delta K_2[0][2] = \delta K_1[0][3]$$
$$\Rightarrow \quad \delta K_0[0][3] \oplus \delta K_1[0][2] \oplus \delta K_1[0][3] = \delta K_0[0][3]$$

As a consequence, if $\delta K_0[0][3] \neq 0$, we cannot satisfy Eq. (6.1). This inconsistency is detected when propagating the XOR constraint associated with Eq. (6.4), as it ensures that $\Delta K_0[0][3] + \Delta K_2[0][1] + \Delta K_2[0][3] \neq 1$.
Hence, we propose to combine XOR equations of the key schedule to generate new equations. More precisely, given two equations $\delta B_1 \oplus \ldots \oplus \delta B_n = 0$ and $\delta B'_1 \oplus \ldots \oplus \delta B'_m = 0$ such that $\{B_1, \ldots, B_n\} \cap \{B'_1, \ldots, B'_m\} \neq \emptyset$, we generate the equation:

$$\bigoplus_{B \in \{B_1,\ldots,B_n\} \cup \{B'_1,\ldots,B'_m\} \setminus \{B_1,\ldots,B_n\} \cap \{B'_1,\ldots,B'_m\}} \delta B = 0.$$

This new equation is recursively combined with existing ones to generate other equations until no more equation can be generated.
For example, from Eq. (6.1), (6.2), and (6.3), we generate the following equations:

$$\text{From (6.1) and (6.2):} \quad \delta K_0[0][3] \oplus \delta K_1[0][3] \oplus \delta K_2[0][1] \oplus \delta K_2[0][2] = 0 \tag{6.5}$$

$$\text{From (6.1) and (6.3):} \quad \delta K_0[0][3] \oplus \delta K_1[0][2] \oplus \delta K_2[0][2] \oplus \delta K_2[0][3] = 0 \tag{6.6}$$

$$\text{From (6.2) and (6.3):} \quad \delta K_1[0][2] \oplus \delta K_1[0][3] \oplus \delta K_2[0][1] \oplus \delta K_2[0][3] = 0 \tag{6.7}$$

Then, from Eq. (6.1) and (6.7) (or, equivalently, from Eq. (6.2) and (6.6) or from Eq. (6.3) and (6.5)), we generate Eq. (6.4). As no new equation can be generated from Eq. (6.1), (6.2), (6.3), (6.4), (6.5), (6.6), and, (6.7), the process stops.
The number of new equations that may be generated grows exponentially with respect to the number r of rounds. For example, for AES-128, when $r = 3$ (resp. $r = 4$), the total number of new equations that may be generated is 988 (resp. 16332). When further increasing r to 5, the number of new equations becomes so large that we cannot generate them within a time limit of one hour. To avoid this combinatorial explosion, we only generate equations that involve at most four differential bytes. Indeed, the basic Boolean XOR constraint associated with an equation simply states that the sum of the Boolean variables must be different from 1. In Section 6.3.3, we show how to strengthen this constraint by reasoning on differences at the byte level when the XOR constraint involves no more than four variables. In this case, each XOR constraint is very efficiently handled by simply channeling three pairs of Boolean variables. Preliminary experiments have shown us that these strengthened XOR constraints both reduce the number of choice points and speed-up the solution process, and that further adding XOR constraints for equations that involve more than four variables (to forbid that their sum is equal to one) does not significantly reduce the number of choice points and often increases time.
For AES-128 (resp. AES-192 and AES-256) with $r = 10$ (resp. $r = 12$ and $r = 14$) rounds, the number of initial equations coming from the key schedule is 144 (resp. 168 and 192). From these initial equations, we generate 122 (resp. 168 and 144) new equations that involve three differential bytes, and 1104 (resp. 1696 and 1256) new equations that involve four differential bytes.
The algorithm that generates equations has been implemented in Picat [25], and the time spent by Picat to search for all equations of length smaller than or equal to four is always smaller than 0.1 seconds. This time is negligible compared to the time needed to solve Step 1.
We note $xorEq_l$ the set of all equations (both initial and generated equations) coming from the key schedule when the key length is l.
Note that $xorEq_l$ actually contains all possible equations with at most four differential bytes implied by the key schedule algorithm. In other words, our procedure does not miss implied equations even if it does not allow the generation of intermediate equations of length greater than 4. We have checked this by exhaustively generating all possible equations with at most four differential key bytes and, for each equation that does not belong to $xorEq_l$, we have proven that it is not a logical consequence of the initial set of equations of the key schedule (by demonstrating

that $xorEq_l$ is consistent with the negation of the equation). The whole checking procedure (performed by a program written in C) for all possible equations, is done in a few minutes for the three possible key lengths.

6.3.2 Variables of CP_{XOR}

All variables of CP_{Basic} are also variables of CP_{XOR}.
We introduce new Boolean variables that are used to tighten the Boolean abstraction by reasoning on differences at the byte level: given two differential bytes δB_1 and δB_2, the Boolean variable $diff_{\delta B_1, \delta B_2}$ is equal to 1 if $\delta B_1 \neq \delta B_2$, and to 0 otherwise. We do not define a *diff* variable for every couple of differential bytes in $diffBytes_l$, but restrict our attention to couples of bytes for which it is useful to know whether they are equal or not.
More precisely, we consider three separate sets of differential bytes and we only compare differential bytes that belong to a same set.

- The first set, called DK, contains bytes coming from δK and δSK matrices. The *diff* variables associated with these bytes are used to tighten the constraints associated with the equations of $xorEq_l$, as explained in Section 6.3.3.

- The second and third sets, called DY and DZ, contain bytes coming from δY and δZ matrices, respectively. The *diff* variables associated with these bytes are used to propagate the MDS property of MixColumns at the byte level, as explained in Section 6.3.3.

For each of these three sets, we consider a separate subset for each row $j \in [0, 3]$:

- For DK, we know that every initial XOR equation due to the key schedule either involves three bytes on a same row j (*i.e.*, $\Delta K_{i+1}[j][k]$, $\Delta K_{i+1}[j][k-1]$, and $\Delta K_i[j][k]$), or it involves two bytes on a same row j (*i.e.*, $\Delta K_i[j][0]$, and $\Delta K_{i+1}[j][0]$), and a byte that has just passed through an S-box on the next row ($(j+1)\%4$) (*i.e.*, $\Delta SK_i[(j+1)\%4][3]$). As we cannot know if the input and output differences of S-boxes are equal or not (*i.e.*, if $\delta K_i[(j+1)\%4][3]$ is equal to $\delta SK_i[(j+1)\%4][3]$ or not), we can limit *diff* variables to couples of differential bytes that occur on a same row of δK matrices, or on two consecutive rows of δK and δSK matrices.

- For DY and DZ, the MDS property implies relations between columns of DY and DZ and, in Section 6.3.3, we use a generalization of this property that XORs bytes of a same row for different columns. As these XORs are only performed on bytes that occur in a same row, we can limit *diff* variables to couples of differential bytes that occur in a same row of δY matrices (for DY) and δZ matrices (for DZ).

Hence, for each row $j \in [0, 3]$, we define the three following sets:

$$\begin{aligned} DK_j &= \{\delta K_i[j][k], \delta SK_i[(j+1)\%4][3] : i \in [1, r], k \in [0, 3]\} \\ DY_j &= \{\delta Y_i[j][k] : i \in [0, r-2], k \in [0, 3]\} \\ DZ_j &= \{\delta Z_i[j][k] : i \in [0, r-2], k \in [0, 3]\}. \end{aligned}$$

Given these sets, we define the *diff* variables as follows: For each set $D \in \{DK_j, DY_j, DZ_j : j \in [0, 3]\}$, and for each pair of differential bytes $\{\delta B_1, \delta B_2\} \subseteq D$, we define a Boolean variable $diff_{\delta B_1, \delta B_2}$.

(C'_1) $obj_{Step1} = \sum_{\delta B \in Sboxes_l} \Delta B$

(C'_2) $\forall \delta B \in Sboxes_l, \Delta SB = \Delta B$

(C'_3) $\forall i \in [0, r-2], \forall j, k \in [0,3], XOR(\Delta Z_i[j][k], \Delta K_{i+1}[j][k], \Delta X_{i+1}[j][k])$

(C'_4) $\forall i \in [0, r-1], \forall j, k \in [0,3], \Delta Y_i[j][k] = \Delta SX_i[j][(j+k)\%4]$

(C'_5) $\forall i \in [0, r-2], \forall k \in [0,3], \left(\sum_{j=0}^{3} \Delta Y_i[j][k] + \Delta Z_i[j][k]\right) \in \{0,5,6,7,8\}$

(C'_6) $\forall j, k \in [0,3], \Delta Z_{r-1}[j][k] = \Delta Y_{r-1}[j][k]$

(C'_7) $\forall D \in \{DK_j, DY_j, DZ_j : j \in [0,3]\}, \forall \{\delta B_1, \delta B_2\} \subseteq D,$
$diff_{\delta B_1, \delta B_2} = diff_{\delta B_2, \delta B_1}$

(C'_8) $\forall D \in \{DK_j, DY_j, DZ_j : j \in [0,3]\}, \forall \{\delta B_1, \delta B_2, \delta B_3\} \subseteq D,$
$diff_{\delta B_1, \delta B_2} + diff_{\delta B_2, \delta B_3} + diff_{\delta B_1, \delta B_3} \neq 1$

(C'_9) $\forall D \in \{DK_j, DY_j, DZ_j : j \in [0,3]\}, \forall \{\delta B_1, \delta B_2\} \subseteq D,$
$diff_{\delta B_1, \delta B_2} + \Delta B_1 + \Delta B_2 \neq 1$

(C'_{10}) $\forall (\delta B_1 \oplus \delta B_2 \oplus \delta B_3 = 0) \in xorEq_l,$
$(diff_{\delta B_1, \delta B_2} = \Delta B_3) \wedge (diff_{\delta B_1, \delta B_3} = \Delta B_2) \wedge (diff_{\delta B_2, \delta B_3} = \Delta B_1)$

(C'_{11}) $\forall (\delta B_1 \oplus \delta B_2 \oplus \delta B_3 \oplus \delta B_4 = 0) \in xorEq_l,$
$(diff_{\delta B_1, \delta B_2} = diff_{\delta B_3, \delta B_4}) \wedge (diff_{\delta B_1, \delta B_3} = diff_{\delta B_2, \delta B_4}) \wedge (diff_{\delta B_1, \delta B_4} = diff_{\delta B_2, \delta B_3})$

(C'_{12}) $\forall i_1, i_2 \in [0, r-2], \forall k_1, k_2 \in [0,3],$
$\sum_{j=0}^{3} diff_{\delta Y_{i_1}[j][k_1], \delta Y_{i_2}[j][k_2]} + diff_{\delta Z_{i_1}[j][k_1], \delta Z_{i_2}[j][k_2]} \in \{0,5,6,7,8\}$

(C'_{13}) $\forall i_1, i_2 \in [0, r-2], \forall j, k_1, k_2 \in [0,3],$
$diff_{\delta K_{i_1+1}[j][k_1], \delta K_{i_2+1}[j][k_2]} + diff_{\delta Z_{i_1}[j][k_1], \delta Z_{i_2}[j][k_2]} + \Delta X_{i_1+1}[j][k_1] + \Delta X_{i_2+1}[j][k_2] \neq 1$

Figure 6.2: Constraints of the CP_{XOR} model for Step 1

6.3.3 Constraints of CP_{XOR}

The constraints of CP_{XOR} are listed in Fig. 6.2. Constraints (C'_1) to (C'_6) are identical to Constraints (C_1) to (C_6) of CP_{Basic}. Constraints (C'_2), (C'_4), (C'_6), (C'_7), (C'_{10}), and (C'_{11}) are equalities of the form $\Delta_i = \Delta_j$: they allow us to rename variables in order to simplify the description of the model without changing the performance of the solvers.
Constraints (C'_7) and (C'_8) ensure symmetry and transitivity on *diff* variables. For each set $D \in \{DK_j, DY_j, DZ_j : j \in [0,3]\}$, they ensure that the *diff* variables associated with couples of bytes in D define an equivalence relation.
Constraint (C'_9) relates *diff* and Δ variables. Indeed, whenever two Boolean variables ΔB_1 and ΔB_2 are equal to 0, then we know for sure that there is no difference at the byte level, *i.e.*, $\delta B_1 = \delta B_2$. Similarly, whenever $\Delta B_1 \neq \Delta B_2$, we know for sure that there is a difference at the byte level, *i.e.*, $\delta B_1 \neq \delta B_2$.
Constraints (C'_{10}) and (C'_{11}) propagate the XOR equations of $xorEq_l$ and are described in Section 6.3.3. Constraint (C'_{12}) propagates the MDS property at the byte level and is described in Section 6.3.3. Constraint (C'_{13}) propagates *ARK* at the byte level and is described in Section 6.3.3.

Constraints associated with the XOR equations of $xorEq_l$

In the basic model, every XOR constraint involves exactly three Boolean variables, and ensures that the sum of these variables is different from 1.
In Section 6.3.1, we have shown how to generate new XOR equations from initial key schedule equations, and these new equations either involve three or four differential bytes. Hence, we need to extend the XOR constraint for the case where we have four Boolean variables. A straightforward extension is to simply state that the sum of the Boolean variables must be different from 1. Indeed, a XOR equation that involves four differential bytes cannot be satisfied if exactly one of these four differential bytes is different from zero. However, this is a poor abstraction of the relation at the byte level. We propose to tighten Boolean XOR constraints associated with the Key Schedule by exploiting *diff* variables.
Constraint (C'_{10}) corresponds to the case of equations that involve three differential bytes. For each equation $\delta B_1 \oplus \delta B_2 \oplus \delta B_3 = 0$ in $xorEq_l$, it ensures that whenever two differential bytes of $\{\delta B_1, \delta B_2, \delta B_3\}$ have different values, then the third one is different from 0 and therefore its associated Boolean variable is equal to 1. Note that this constraint combined with Constraint (C'_9) ensures that $\Delta B_1 + \Delta B_2 + \Delta B_3 \neq 1$. Indeed, if $\Delta B_1 = 1$ and $\Delta B_2 = \Delta B_3 = 0$, then Constraint (C'_9) implies that $diff_{\delta B_2, \delta B_3} = 0$, which is inconsistent with the fact that $diff_{\delta B_2, \delta B_3}$ must be equal to ΔB_1.
Constraint (C'_{11}) corresponds to the case of equations that involve four differential bytes. For each equation $\delta B_1 \oplus \delta B_2 \oplus \delta B_3 \oplus \delta B_4 = 0$ in $xorEq_l$, it ensures that whenever two differential bytes of $\{\delta B_1, \delta B_2, \delta B_3, \delta B_4\}$ are equal then the two other ones must also be equal. Again, this constraint combined with Constraint (C'_9) ensures that $\Delta B_1 + \Delta B_2 + \Delta B_3 + \Delta B_4 \neq 1$. Indeed, if $\Delta B_1 = 1$ and $\Delta B_2 = \Delta B_3 = \Delta B_4 = 0$, then Constraint (C'_9) implies that $diff_{\delta B_2, \delta B_3} = 0$ and $diff_{\delta B_1, \delta B_4} = 1$, which is inconsistent with the fact that $diff_{\delta B_2, \delta B_3}$ must be equal to $diff_{\delta B_1, \delta B_4}$.

Propagation of MDS at the byte level

As seen in Section 2.1 (paragraph "Maximum Distance Separable property"), the MDS property also holds for the result of the XOR of two different columns of δY and δZ. Hence, for all i_1, i_2 in $[0, r-2]$, and for all k_1, k_2 in $[0, 3]$, we have:

$$\sum_{j=0}^{3} (\delta Y_{i_1}[j][k_1] \oplus \delta Y_{i_2}[j][k_2] \neq 0) + (\delta Z_{i_1}[j][k_1] \oplus \delta Z_{i_2}[j][k_2] \neq 0) \in \{0, 5, 6, 7, 8\}.$$

This property is modelled by constraint (C'_{12}).

Propagation of *ARK* at the byte level

ARK implies the following equations: $\forall i_1, i_2 \in [0, r-2], \forall j, k_1, k_2 \in [0, 3]$,

$$\begin{aligned} \delta K_{i_1+1}[j][k_1] \oplus \delta Z_{i_1}[j][k_1] &= \delta X_{i_1+1}[j][k_1] \\ \delta K_{i_2+1}[j][k_2] \oplus \delta Z_{i_2}[j][k_2] &= \delta X_{i_2+1}[j][k_2]. \end{aligned}$$

By XORing these two equations, we obtain:

$$\begin{aligned} & \delta K_{i_1+1}[j][k_1] \oplus \delta K_{i_2+1}[j][k_2] \\ \oplus \quad & \delta Z_{i_1}[j][k_1] \oplus \delta Z_{i_2}[j][k_2] \\ = \quad & \delta X_{i_1+1}[j][k_1] \oplus \delta X_{i_2+1}[j][k_2]. \end{aligned}$$

From this equation, we infer that it is not possible to have exactly one of the three following inequalities which is true:

$$\begin{aligned} \delta K_{i_1+1}[j][k_1] &\neq \delta K_{i_2+1}[j][k_2] \\ \delta Z_{i_1}[j][k_1] &\neq \delta Z_{i_2}[j][k_2] \\ \delta X_{i_1+1}[j][k_1] &\neq \delta X_{i_2+1}[j][k_2] \end{aligned}$$

Constraint (C'_{13}) ensures this property: The first two inequalities are modeled by $\mathit{diff}_{\delta K_{i_1+1}[j][k_1],\delta K_{i_2+1}[j][k_2]}$ and $\mathit{diff}_{\delta Z_{i_1}[j][k_1],\delta Z_{i_2}[j][k_2]}$; For the third inequality, we exploit the fact that if $\Delta X_{i_1+1}[j][k_1] + \Delta X_{i_2+1}[j][k_2] = 1$ then $\delta X_{i_1+1}[j][k_1] \neq \delta X_{i_2+1}[j][k_2]$.

6.3.4 Implementation

CP_{XOR} only uses common and widely implemented constraints. Therefore, it may be easily implemented with any existing CP libraries. In order to ease the comparison between different solvers, we have implemented CP_{XOR} with a high-level modeling language called MiniZinc [18]: MiniZinc models are translated into a simple subset of MiniZinc called FlatZinc, using a compiler provided by MiniZinc, and most existing CP solvers have developed FlatZinc interfaces (currently, there are fifteen CP solvers which have FlatZinc interfaces).
We ran experiments with Gecode [11], Choco [21], Chuffed [5], and Picat-SAT [25]: Gecode and Choco are CP libraries which solve CSPs by combining search with constraint propagation; Chuffed is a lazy clause hybrid solver that combines features of finite domain propagation and Boolean satisfiability; Picat-SAT translates CSPs into Boolean satisfiability formulae, and then uses the SAT solver Lingeling [1] to solve it.
Gecode and Choco are clearly outperformed by Chuffed and Picat-SAT for both *Step1-opt* and *Step1-enum*. This shows us that clause learning is a key ingredient for solving these problems. Picat-SAT and Chuffed have complementary performance. For the optimisation problem *Step1-opt*, Picat-SAT is always the fastest solver. For the enumeration problem *Step1-enum*, Chuffed is faster than Picat-SAT on small instances, *i.e.*, all AES-128 instances, and AES-192 (resp. AES-256) instances when the number of rounds r is lower than 5 (resp. 7). However, Picat-SAT is faster than Chuffed on larger instances and Chuffed is not able to solve AES-192 (resp. AES-256) instances within a 24 hour time limit when $r \geq 9$ (resp. $r \geq 11$).
The good performance of Picat-SAT is not surprising given that most variables are Boolean variables. If we exclude equality constraints, there are only two different kinds of constraints:

- (C'_3), (C'_8), (C'_9), and (C'_{13}) are of the form: $\sum_{\Delta_i \in S} \Delta_i \neq 1$;
- (C'_1), (C'_5) and (C'_{12}) are of the form: $\sum_{\Delta_i \in S} \Delta_i = v$ where v is the integer variable obj_{Step1} for (C'_1) and an integer variable whose domain is $\{0, 5, 6, 7, 8\}$ for (C'_5) and (C'_{12});

where S is a set of Boolean variables. These two kinds of constraints are easily translated into Boolean formulae by combining at-most and at-least encodings which are widely studied in the SAT community [24]. Note that we cannot use the XOR-clauses introduced in CryptoMiniSat [23] for modeling the XOR constraint in Step 1 because the semantics of the XOR operation is changed when abstracting every byte by a Boolean value.
As CP_{XOR} only uses sum constraints, we can also translate it into an Integer Linear Programming (ILP) model in a very straightforward way, by using an encoding similar to the one introduced in [17], for example. Each constraint $\sum_{\Delta_i \in S} \Delta_i \neq 1$ is translated into $\#S$ inequalities: for each $\Delta_j \in S$, we add the inequality $-\Delta_j + \sum_{\Delta_i \in S \setminus \{\Delta_j\}} \Delta_i \geq 0$. Each constraint $\sum_{\Delta_i \in S} \Delta_i = v$ such that v is an integer variable whose domain is $\{0, 5, 6, 7, 8\}$ is encoded by introducing a new Boolean variable

$x \in \{0, 1\}$ and adding the following inequalities:

$$\sum_{\Delta_i \in S} \Delta_i \geq 5x$$
$$\forall \Delta_i \in S, x \geq \Delta_i$$

When $x = 0$, every Δ_i is constrained to be equal to 0 by $x \geq \Delta_i$; Otherwise, the first inequality ensures that at least 5 variables of S are set to 1. Gurobi [20] has been used to solve this ILP model, and experiments have shown that it is not competitive with Picat-SAT.

7 Adaptation on Rijndael and results

Overview Let's present the approach we have put in place to adapt this reasoning on Rijndael. We will deal first with the adaptation of CP_{XOR}, then the adaptation of Step2, and finally the results obtained.

7.1 CP_{XOR} on Rijndael

Let's present first the main principle and basic model, then the reduction of space search thanks to the new equations, then the models finally used for running the Step 1, and at last the results of this Step 1.

7.1.1 Definition

We first obtain the basic XOR equations, in order to then deduce new equations.

The general case is split into two cases, depending on the value of N_k.

Case $N_k \leq 6$

As a reminder, here is the key expansion of this case.

```
KeyExpansion(K[4][Nk], W[4][Nb(Nr + 1)]){
    for (j=0; j<Nk; j++)
        for (i=0; i<4; i++) W[i][j]=K[i][j];
    for (j=Nk; j<Nb(Nr + 1); j++){
        if (j mod Nk == 0) { //non-linear function
            W[0][j] = W[0][j − Nk] ⊕ S[W[1][j − 1]] ⊕ RC[J/Nk];
            for (i=1; i<4; i++)
                W[i][j] = W[i][j − Nk] ⊕ S[W[i + 1 mod 4][j − 1]];
        } else {
            for (i=0; i<4; i++) W[i][j] = W[i][j − Nk] ⊕ W[i][
j − 1];
        }
    }
}
```

Notations:

- i denotes the round number. i is from 0 to r-1, where r is the number of rounds. Indeed, we have seen that in the final round, there is not `KeySchedule` operation. In fact i is not used in this function, it is used only for `ExpandedKey` (and the other operations).

- j denotes the row number (in 0..4)
- k denotes the column number
- KC is the number of columns in the key (in 4, 6, 8)
- The XOR function must be defined (depending on the number of arguments).
  Here, $\mathrm{XOR}(a_0, a_1, \ldots, a_n)$, $n \geq 2$ is equivalent to $a_0 = a_1 \oplus ... \oplus a_n$ (or $a_0 \oplus a_1 \oplus ... \oplus a_n = 0$).

Case $N_k > 6$

```
KeyExpansion(K[4][Nk], W[4][Nb(Nr + 1)]){
    for (j=0; j<Nk; j++)
        for (i=0; i<4; i++) W[i][j]=K[i][j];
    for (j=Nk; j<Nb(Nr + 1); j++){
        if (j mod Nk == 0) { //non-linear function
            W[0][j] = W[0][j − Nk] ⊕ S[W[1][j − 1]] ⊕ RC[J/Nk];
            for (i=1; i<4; i++)
                W[i][j] = W[i][j − Nk] ⊕ S[W[i + 1 mod 4][j − 1]];
        } else if (j mod Nk == 4) { //non-linear function
            for (i=0; i<4; i++) W[i][j] = W[i][j − Nk] ⊕ S[W[i
][j − 1]];
        } else {
            for (i=0; i<4; i++) W[i][j] = W[i][j − Nk] ⊕ W[i][
j − 1];
        }
    }
}
```

Remark on RC: Actually, the `RoundConstant` is useless. Indeed, we are interested in tracking the differences. In other terms, we are interested in the evolution of $a \oplus b$ where $b = \delta a$, *i.e.*the evolution of δa, and $a \oplus b = a \oplus \mathrm{RC} \oplus b \oplus \mathrm{RC}$.

Python scripts have been made to generate all these XOR equations.

***Nota Bene*:** For implementation reasons, we denote in the following XOR3 (and respectively XOR4) equations XOR equations of length 3 (respectively 4).

7.1.2 New constraints

XOR Constraints For the same reasons as the AES, these equations are not enough restrictive to solve instances in a "reasonable" time (the notion of "reasonable" has been precised in the case of the AES). To find new equations, we can combine the existing ones.
Given two equations $a_i \oplus ... \oplus a_j = 0$ and $a_{i'} \oplus ... \oplus a_{j'} = 0$ ($\{a_i, ..., a_j\} \cap \{a_{i'}, ..., a_{j'}\} \neq \emptyset$), we can combine them obtaining $\bigoplus_{k \in \{i,...,j\} \cup \{i',...,j'\} \setminus \{i,...,j\} \cap \{i',...,j'\}} a_k = 0$.

Even if the number of constraints decreases the number of solutions, a bigger number of equations implies more time spent for solving. That's why we have to limit this number, in order to

avoid a time explosion. As a consequence, only XOR equations implying 3 or 4 variables are taken into account. That appears as a good compromise between the number of solutions obtained and the time spent.

As precised above, even if the new equations are deduced from the definition ones, there are not redundant, because there can be some inconsistencies satisfying definition equations, forbidden by these new ones.

From these new ones, it is possible to combine them again, either themselves or with the original ones, keeping only XOR3 and XOR4 constraints, until the equations set is stable by combination for each equation from this set with any other one, *i.e.*$\forall e_1, e_2 \in XOR_{eq}, e \in XOR_{eq}$, where e is the combination of e_1 and e_2 after reduction and XOR_{eq} the set of XOR equations.

Diff constraints With the same reasoning, we can easily generalize the generation of Differential Constraints, with the same model (cf. Fig. 6.2).

7.1.3 New models

S-Box in practice

As seen above, during the Step1, bytes are abstracted by a Δ value, $0 \mapsto 0$, $[1, 255] \mapsto 1$.
That's why "computing" the operations which imply S-Box is not directly possible at byte level. What's more, we need to count the number of S-Boxes, in order to satisfy the constraint of $S1_{enum}$, constraining the number of S-Boxes to a fixed value: $objS1$.

In fact, what matters is the number of S-Boxes, and tracking the differences through the S-Boxes. In other terms, through the S-Box, at byte level, $S(0) = 0$ and $S(1) = 1$, and in the case where we have to compute $S(1) = 1$, we have to count this occurrence.

AES In the case of the AES, an additional column was this "counter". A 5^{th} column was added to the state, representing the number of S-Boxes in the state. In fact, as the block length was fixed to 128 (AES), the position of the S-Box was deterministic: indeed, the row number is known, and in a state, only one column could be concerned by a S-Box. Indeed, S-Boxes are present only in the case $k \mod N_k = 0$, and at column $k - 1$. If $N_k == 128$, there is exactly one column by state. In other cases, there is less S-Box by state (state size is fixed, N_k increases).

Thus, it is easy to constraint the number of S-Boxes, constraining the addition of all 5^{th} column values for each state, equal to $objS1$.

DkFlat In the case of Rijndael, we can have several S-Boxes by state, in particular in the case where N_b is greater than N_k. Indeed, the condition $k \mod N_k$ can occur several times by round (several k values in $[N_b * r, N_b * (r + 1)]$). That's why the model presented above does not correspond to this case.

A solution is to count the number of S-Boxes not only with one column by state, but to track the eventual S-Box for each value. In other words, if the key is computed in advance, in an array with all values, we can duplicate it in order to have a dimension more, only to count the number

of S-Boxes. That is called $DkFlat$, because it represent this notion of key "expanded" before processing Rijndael. So the dimensions of DkFlat are $(r * N_k, 4, 2)$. For each column $k \in [0, r * N_k - 1]$ and each row $j \in [0, 3]$, $DkFlat[i, j, 1] = DkFlat[i, j, 0]$ if this value is going through a S-Box, and $DkFlat[i, j, 1] = 0$ otherwise.

Conclusion After the implementation of this solution (adaptation of equations etc.), several solutions have been found in specific cases. However, some issues occurred in adaptation of constraints, in particular XOR and DIFF constraints, being too much restrictive at Step1.
Many other models have been proposed and tested. In particular, an other one which has worked but for a low number of rounds only ($r = 3$) is to adapt CP_{Basic}, adding new XOR equations generated from KeySchedule definition.

7.1.4 Step1 Results

Implementation The generation of the new equations has been proceed with `Python` ; that does not represent an important part of the calculus time (a few seconds) in regard of the solving time. For the reasons explained in 6.3.4, the same tools have been used for solving problems.

Impact The number of equations increases (in the case of the AES-128 with 3 rounds: 116 XOR equations, 331 DIFF equations), but the computation time is several times[1] more efficient than with CP_{Basic} model. Concerning the number of equations, we recall that the better compromise between increasing the number of new equations (more complex to solve) and keeping a model as simple to solve as possible (to optimize the rapidity of solving) implied, as said above, to generate as many equations as possible but to keep only XOR3 and XOR4. The studies carried out that led to this conclusion are detailed in Section 6.3.1.
For instance, with the $DkFlat$ modeling, the case of the AES-128, 3 rounds, $objS1 = 5$, in solved within less than 5 seconds (4 solutions), while CP_{Basic} does not terminate within 240 minutes (more than 4500 solutions). Runtimes are detailed in section 7.3.

7.2 New Step 2

Unlike the adaptation of Step1, Step2 is easier to adapt. On the one hand, there are "practical" details of implementation, and on the other hand, the implementation of specific operations which differ between the AES and Rijndael.

Implementation The adaptation of Step1 required to change of way for modeling. *A contrario*, in the Step2 processing, there is only operations to generalize, but the modeling strategy chosen for the work already done on the AES could be generalized. About implementation, there were only implementation details to handle correctly parameters (like the block length for instance) and other variables to satisfy the specification of Rijndael.

[1] depending on parameters. Runtimes are detailed in the next paragraph, in table 7.2. For a number of round greater than 4, computation with CP_{Basic} cannot be computed in a "reasonable" time, *i.e.*several hours on a 64 bits PC, 8GB RAM, Core i7 4*2.67GHz

Specific operations However, depending on the block length, some operations or variables change. That is the case for the number of rounds, or the ShiftRows processing. So, some parts had to be re-implemented, to correspond to the definition of Rijndael. Let's precise that *a fortiori*, as for all particular cases of Rijndael, that still works in the case of the AES.

7.3 Results

Practical remarks concerning runtimes In this section, the runtimes mentioned are the results of tests process on a personal computer; the technical specifications of which are as follows: 64 bits PC, 8GB RAM, Core i7 4*2.67GHz.

Implementation issues The main part of the work done on this adaptation was to propose and test models to implement efficiently this reasoning on all cases of Rijndael. Actually, "efficiently" means which can be ran within a few hours or days for most "complicated" cases, *i.e.*cases with the greatest number of rounds. Indeed, model used on the AES (way to model the S-Box, for instance) could not be applied on Rijndael due to the possible values of the state size (block length).

In fact, several models have been proposed to describe the main process, in particular the S-Boxes. We need to define the behavior of the variables, to continue the tracking of differences through the S-Boxes (to process operations with these new values), that is difficult at byte level (at Step 1, before instantiation). This issue was very time-consuming. Combining existing models ($S1_{XOR}$, $S1_{Basic}$, different ways to define the Key, etc.), some executions have lead to concrete outputs. However, many instances have not been resolved with these models, and some other ones are not optimal (considerations based on other works began on this topic: we initially knew the optimal value of $objS1$).

We can present several new solutions, and some other solutions (including optimal ones) which do not "need" a key difference to be an optimal attack.

Results synthesis These results are presented in table 7.1. For the new solutions, with a key differential characteristic, we have detailed the runtimes in table 7.2.
The solutions represent a value of δX, δK, allowing us to predict δC with a good probability, *i.e.*a probability greater than the uniform probability. That means that for any plaintext X and for any key K, we have a good probability to predict the value of $f_{K+\delta K}(X+\delta X) = C + \delta C$ ($C = f_K(X)$).

N_b	N_k	r	objS1	Result
5	4	3	9	$\delta K = 0$ (non optimal)
5	8	3	1	solution
6	4	3	9	solution
6	6	3	5	solution
7	4	3	9	$\delta K = 0$
8	4	3	9	$\delta K = 0$
8	8	3	5	solution

Table 7.1: Results

The results in the cases mentioned as "solution" are given in Appendix A.

N_b	N_k	r	**Runtime** $S1$	**Runtime** $S2$	**Total runtime**
5	8	3	15	22	37
6	6	3	119	1453	1572
8	8	3	145	1376	1519

Table 7.2: Time spent for solutions (in seconds, on the device presented in 7.3)

Interpretation That proves that introducing a difference not only in the plaintext but also in the key allows us to obtain better results than in the case where the difference was only in the plaintext.
However, we can note that for some cases the optimal difference seems to not exploit this possibility and an[2] optimal solution is found with $\delta K = 0$.

[2]We say "a" solution, because there can be several solutions with the same probability.

Part IV

Conclusion

8 Future Works

After taking a step back from related-key cryptanalysis, in the case of Rijndael and then in the case of other symmetric ciphers, we will conclude about CP, used for this work.

All Rijndael cases

As presented above, this works has consisted in proposing several models to test all the Rijndael cases.

However, some cases (in particular for a significant number of rounds) could not be proceed because of efficiency issues. Other models could be proposed to deal with these problems, being restrictive enough to compute these cases within a reasonable runtime. These first results have shown solutions for simple cases, and highlight that using related-key cryptanalysis is more efficient than "simple" differential cryptanalysis. That's why extending these attacks to general cases is relevant.

Other ciphers

This reasoning could be applied on other symmetric problems.

Indeed, in practical applications, some ciphers have already been broken by related-key attacks, like KASUMI [7] or RC4 [9]. We can see that it can be applied on several kind of symmetric ciphers (stream ciphers, block ciphers, etc.), and that is not an unrealistic model: even if the attacker seems too powerful, it is not unreasonable to suppose that with a good probability and many data exchanged (for instance, packets in internet protocols), because of the birthday paradox, an attack is possible. That was illustrated in these attacks: for instance, in the case of RC4, some safety properties of Wired Equivalent Privacy (WEP), a security algorithm for IEEE 802.11, have been broken, and some attacks were possible to be performed in a few minutes (with a personal computer).

To conclude, these experiences have shown the relevance to study this reasoning on other symmetric ciphers.

About CP

In this case (Rijndael) we have chosen CP for solving related-key cryptanalysis problem. Indeed, as said in section 5, in our case that is the best way *i.e.*faster than the other ones (presented above).

However, it could be interesting either to wonder if that is true for other cryptographic problems, or to prove that is the best way in regards to other tools, for instance Satisfiability Modulo Theory (SMT) or SAT techniques.

9 Conclusion

This internship has been rewarding for several reasons and on several aspects.

Scientific context About fields I have dealt with, even though our formation gives us a slight background in cryptography, many aspects were new; discovering these fields was challenging and I'm happy to have had this opportunity. The reasoning, the way to solve the problems, in particular during the implementation, constraints programming, were really unknown, the field (Rijndael) was new, as well as many concepts.

That has required an adaptation time, began with reading of papers, in particular, the one which syntheses the work done on the AES. Maybe the fact that not only fields but also the approach were completely unknown at the beginning of the internship have made the beginning a little slow, to adapt to this new environment.

Internship Context Due to the exceptional situation (COVID-19 crisis, teleworking 2 weeks after the beginning of the internship until the end, etc.), the work has been slowed done and communication less efficient. That is why, in regards to the objectives defined before the internship, the work focused on Rijndael, but some steps were more time consuming as expected.

Completing an internship in a Loria team, in public research, was a first and a rewarding experience, that makes sens in regard to my career plan once my stage completed (and so formation ended). That's why, in addition to the formal and compulsory acknowledgments at the beginning of this report, I would like to personally thank my internship supervisor, Ms Marine Minier, to have give me the opportunity to complete this internship within this team. This experience of an internship in this academic context and in this field was unique, and determining to decide knowingly which way to chose for career plans.

Bibliography

[1] Armin Biere. Yet another local search solver and lingeling and friends entering the sat competition 2014. pages 39–40, 01 2014. 37

[2] 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. 27

[3] 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. 20, 27, 28

[4] 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 *LNCS*, pages 322–344. Springer, 2010. 24

[5] Geoffrey Chu and Peter J. Stuckey. Chuffed solver description, 2014. Available at `http://www.minizinc.org/challenge2014/description_chuffed.txt`. 37

[6] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013. 8, 14, 16, 20, 51

[7] Orr Dunkelman, Nathan Keller, and Adi Shamir. A practical-time attack on the a5/3 cryptosystem used in third generation gsm telephony. Cryptology ePrint Archive, Report 2010/013, 2010. `https://eprint.iacr.org/2010/013`. 46

[8] FIPS 197. Advanced Encryption Standard. Federal Information Processing Standards Publication 197, 2001. U.S. Department of Commerce/N.I.S.T. 15

[9] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of rc4. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography*, pages 1–24, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. 46

[10] 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 - Part I*, volume 8042 of *LNCS*, pages 183–203. Springer, 2013. 24

[11] Gecode Team. Gecode: Generic constraint development environment, 2006. Available from `http://www.gecode.org`. 21, 37

[12] David Gerault, Pascal Lafourcade, Marine Minier, and Christine Solnon. Computing aes related-key differential characteristics with constraint programming. *Artificial Intelligence*, 278:103183, 2020. 5, 19, 21, 24

[13] Henri Gilbert and Marine Minier. A collision attack on 7 rounds of rijndael. In *The Third Advanced Encryption Standard Candidate Conference, April 13-14, 2000, New York, New York, USA*, pages 230–241. National Institute of Standards and Technology„ 2000. 11

[14] David Gérault, 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 *LNCS*, pages 584–601. Springer, 2016. 24, 31

[15] Stefan Lucks. Attacking seven rounds of rijndael under 192-bit and 256-bit keys. In *The Third Advanced Encryption Standard Candidate Conference, April 13-14, 2000, New York, New York, USA*, pages 215–229. National Institute of Standards and Technology„ 2000. 11

[16] Marine Minier, Christine Solnon, and Julia Reboul. Solving a Symmetric Key Cryptographic Problem with Constraint Programming. In *13th International Workshop on Constraint Modelling and Reformulation (ModRef), in conjunction with CP'14*, pages 1–13, 2014. 29

[17] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In Chuan-Kun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology*, pages 57–76. Springer, 2012. 37

[18] 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 *LNCS*, pages 529–543. Springer, 2007. 22, 37

[19] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 55–64. Springer, 1993. 11

[20] Gurobi Optimization. Gurobi optimizer reference manual, 2018. 38

[21] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation.* TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016. 21, 37

[22] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence).* Elsevier Science Inc., New York, NY, USA, 2006. 21

[23] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009. 37

[24] Neng-Fa Zhou and Håkan Kjellerstrand. Optimizing SAT encodings for arithmetic constraints. In *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017*, volume 10416 of *Lecture Notes in Computer Science*, pages 671–686. Springer, 2017. 37

[25] Neng-Fa Zhou, Hakan Kjellerstrand, and Jonathan Fruhman. *Constraint Solving and Planning with Picat.* Springer, 2015. 33, 37

List of Figures

List of Tables

Glossary

AES Advanced Encryption Standard. 4

ARK AddRoundKey. 13

CP Constraints Programming. 21, 46

CSPs Constraint Satisfaction Problems. 21

GAC Generalized Arc Consistency. 22

ILP Integer Linear Programming. 21

KS KeySchedule. 13

MC MixColumns. 12

NIST National Institute of Standards and Technology. 8

SAT Boolean SATisfiability. 21, 47

SB SubBytes. 11

SMT Satisfiability Modulo Theory. 47

SR ShiftRows. 12

WEP Wired Equivalent Privacy. 46

Appendices

A Solutions

Files are too large to be displayed here. They are available in a GitHub repository at the following address: https://github.com/Amb02/MasterThesis_solutions.

Format The file names are written as follows: $[N_b]$-$[N_k]$-$[r]$_S2. The value of $objS1$ is written in Table 7.1. For each result in all files, the probability (optimal) preced the result values.

Résumé

La cryptographie et notamment la cryptographie symétrique est une pierre angulaire des communications. Elle permet de garantir des propriétés comme la confidentialité, l'intégrité ou la signature. Alors que la cryptographie à clé publique repose sur des problèmes bien connus et difficiles à résoudre, la cryptographie symétrique utilise des opérations élémentaires itérées un grand nombre de fois. Les primitives les plus importantes en cryptographie symétriques sont les fonctions de hachage permettant de garantir l'intégrité, les chiffrements à flot et les chiffrements par blocs qui eux, garantissent la confidentialité des échanges.

Au cours de la dernière décennie, beaucoup de résultats de cryptanalyse se sont intéressés à la recherche de chemins différentiels dans les chiffrements par blocs où un attaquant introduit une différence en entrée du chiffrement et prédit la différence en sortie sur les chiffrés correspondants avec une certaine probabilité que l'on veut la plus éloignée possible de la probabilité uniforme.
La cryptanalyse différentielle a également étendu le modèle d'attaque au cas dit à clés liées où on s'autorise à introduire des différences non seulement dans les clairs mais également dans les clés. Ces travaux impliquent généralement la résolution de problèmes fortement combinatoires (NP-difficiles) qui peuvent être résolus par des bibliothèques de programmation par contraintes. Le problème est décrit de façon déclarative, puis des algorithmes génériques explorent l'espace de recherche de façon systématique.La programmation par contraintes a été utilisée pour résoudre des problèmes de cryptanalyse différentielle de chiffrements par blocs notamment dans le modèle dit de différentielles à clés liées contre l'AES.

Les travaux menés ont consisté à poursuivre cette démarche en étudiant le comportement de Rijndael face à ce type d'attaques.

286 mots

Mots-clefs :

- cryptographie symétrique
- Rijndael
- cryptanalyse différentielle
- programmation par contraintes

Abstract

Cryptography, in particular symmetric cryptography, is a cornerstone of communications. It ensures safety properties like confidentiality, integrity, or authentication. Asymmetric cryptography is based on well-known problems, hard-solvable. *A contrario*, symmetric cryptography uses elementary operations iterated many times. The most important primitives in symmetric cryptography are hash functions (to ensure integrity), stream ciphering and block ciphering (to ensure confidentiality of communications).

For the last decade, many cryptanalysis results have focused on the search for differential paths in block ciphers. In these attacks, an attacker introduces an input difference in the ciphering process to predict the output difference (difference on the ciphered), with a probability, as far as possible from the uniform probability.
Differential cryptanalysis has been extended to related-key attacks, where the input concerns not only a difference in the input plaintexts but also in the keys. That implies solving of NP-difficult problems. A solution to solve theses instances is to use constraint programming. The problem is described in a declarative way, defining the set of possibilities where variables are constrained, and generic algorithms deduce from these constraints the solutions satisfying these constraints. A search tree is built and constraints are propagated at each node to reduce the search space. Constraints programming has been used to solve problems of differently cryptanalysis in block ciphering, in particular related-key cryptanalysis in the case of the AES.

The works lead during this internship was to study the behavior of Rijndael in the face of this kind of attack.

244 words

Keywords :

- symmetric cryptography
- Rijndael
- differential cryptanalysis
- constraint programming