Improving scalability and reusability of differential cryptanalysis models using constraint programming

Loïc Rouquette

N° d'ordre NNT: 2022ISAL0094

THESE de DOCTORAT DE L'INSA Lyon
membre de
l'Université de Lyon

Ecole Doctorale N° 512
Informatique et Mathématiques

Spécialité / discipline de doctorat :

Informatique

Soutenue publiquement le 15/11/2022, par :
Loïc Rouquette

Improving scalability and reusability of differential cryptanalysis models using constraint programming.

Devant le jury composé de :

Lecoutre, Christophe	Professeur des Universités Université d'Artois	Président
Dequen, Gilles	Professeur des Universités Université de Picardie Jules Verne	Rapporteur
Naya Plasencia, Maria	Directrice de Recherche INRIA Paris	Rapporteure
Boura, Christina	Maîtresse de Conférences Université de Versailles Saint-Quentin-en- Yvelines	Examinatrice
Derbez, Patrick	Maître de Conférences Université de Rennes	Examinateur
Solnon, Christine	Professeure des Universités INSA Lyon	Directrice de thèse
Minier, Marine	Professeure des Universités Université de Lorraine	Co-directrice de thèse

Département FEDORA – INSA Lyon - Ecoles Doctorales

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	**CHIMIE DE LYON** **https://www.edchimie-lyon.fr** Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage secretariat@edchimie-lyon.fr	**M. Stéphane DANIELE** C2P2-CPE LYON-UMR 5265 Bâtiment F308, BP 2077 43 Boulevard du 11 novembre 1918 69616 Villeurbanne **directeur@edchimie-lyon.fr**
E.E.A.	**ÉLECTRONIQUE, ÉLECTROTECHNIQUE, AUTOMATIQUE** **https://edeea.universite-lyon.fr** Sec. : Stéphanie CAUVIN Bâtiment Direction INSA Lyon Tél : 04.72.43.71.70 secretariat.edeea@insa-lyon.fr	**M. Philippe DELACHARTRE** INSA LYON Laboratoire CREATIS Bâtiment Blaise Pascal, 7 avenue Jean Capelle 69621 Villeurbanne CEDEX Tél : 04.72.43.88.63 **philippe.delachartre@insa-lyon.fr**
E2M2	**ÉVOLUTION, ÉCOSYSTÈME, MICROBIOLOGIE, MODÉLISATION** **http://e2m2.universite-lyon.fr** Sec. : Bénédicte LANZA Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 secretariat.e2m2@univ-lyon1.fr	**Mme Sandrine CHARLES** Université Claude Bernard Lyon 1 UFR Biosciences Bâtiment Mendel 43, boulevard du 11 Novembre 1918 69622 Villeurbanne CEDEX **sandrine.charles@univ-lyon1.fr**
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTÉ** **http://ediss.universite-lyon.fr** Sec. : Bénédicte LANZA Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 secretariat.ediss@univ-lyon1.fr	**Mme Sylvie RICARD-BLUM** Institut de Chimie et Biochimie Moléculaires et Supramoléculaires (ICBMS) - UMR 5246 CNRS - Université Lyon 1 Bâtiment Raulin - 2ème étage Nord 43 Boulevard du 11 novembre 1918 69622 Villeurbanne Cedex Tél : +33(0)4 72 44 82 32 **sylvie.ricard-blum@univ-lyon1.fr**
INFOMATHS	**INFORMATIQUE ET MATHÉMATIQUES** **http://edinfomaths.universite-lyon.fr** Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage Tél : 04.72.43.80.46 infomaths@univ-lyon1.fr	**M. Hamamache KHEDDOUCI** Université Claude Bernard Lyon 1 Bât. Nautibus 43, Boulevard du 11 novembre 1918 69 622 Villeurbanne Cedex France Tél : 04.72.44.83.69 **hamamache.kheddouci@univ-lyon1.fr**
Matériaux	**MATÉRIAUX DE LYON** **http://ed34.universite-lyon.fr** Sec. : Yann DE ORDENANA Tél : 04.72.18.62.44 yann.de-ordenana@ec-lyon.fr	**M. Stéphane BENAYOUN** Ecole Centrale de Lyon Laboratoire LTDS 36 avenue Guy de Collongue 69134 Ecully CEDEX Tél : 04.72.18.64.37 **stephane.benayoun@ec-lyon.fr**
MEGA	**MÉCANIQUE, ÉNERGÉTIQUE, GÉNIE CIVIL, ACOUSTIQUE** **http://edmega.universite-lyon.fr** Sec. : Stéphanie CAUVIN Tél : 04.72.43.71.70 Bâtiment Direction INSA Lyon mega@insa-lyon.fr	**M. Jocelyn BONJOUR** INSA Lyon Laboratoire CETHIL Bâtiment Sadi-Carnot 9, rue de la Physique 69621 Villeurbanne CEDEX **jocelyn.bonjour@insa-lyon.fr**
ScSo	**ScSo*** **https://edsciencessociales.universite-lyon.fr** Sec. : Mélina FAVETON INSA : J.Y. TOUSSAINT Tél : 04.78.69.77.79 melina.faveton@univ-lyon2.fr	**M. Christian MONTES** Université Lumière Lyon 2 86 Rue Pasteur 69365 Lyon CEDEX 07 **christian.montes@univ-lyon2.fr**

*ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

Contents

Introduction & Résumé en français

La cryptographie, des mots grecs *κρυπτός* (kruptós, “caché, secret”) et *γράφειν* (graphein, “écrire”) est la pratique et l’étude des techniques de protection du secret. Avant l’avènement des systèmes d’information, le rôle de la cryptographie était principalement de fournir des systèmes de confidentialité. La cryptographie est utilisée depuis l’antiquité dans les opérations militaires, la diplomatie et l’espionnage. L’un des plus célèbres chiffrement de l’époque est le chiffrement de César (voir Table 1), qui consiste à décaler les lettres de l’alphabet d’un nombre fixe d’éléments, les éléments de ponctuation n’étant pas remplacés. Par exemple, la phrase : `Caesar's cipher` chiffrée avec un décalage de 3 (`A` devient `D`) est `Fhdvdu'v flskhu`.

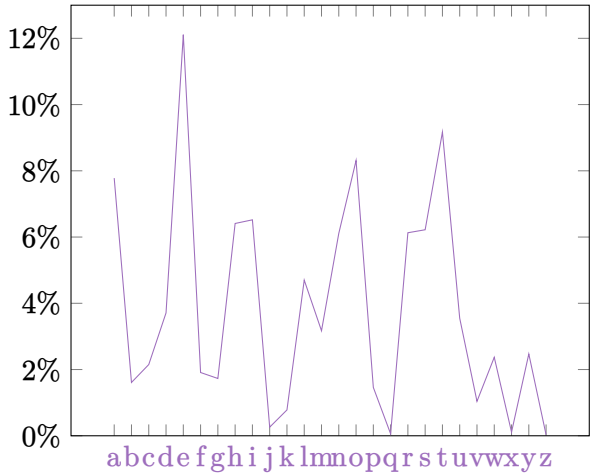
Le chiffrement de César est dit monoalphabétique, chaque caractère est toujours remplacé par le même caractère, *e.g.* le caractère `A` sera toujours remplacé par le caractère `D`. Les chiffrements monoalphabétiques sont très sensibles aux attaques par analyse de fréquence (Figure 1). Comme les lettres de l’alphabet n’apparaissent pas uniformément dans les langues, il est possible d’effectuer une analyse de fréquence sur les symboles du texte chiffré pour établir des correspondances. L’approche du chiffrement par substitution polyalphabétique, comme le chiffrement *autokey* [Bel53] (également connu sous le nom de chiffrement de Vigenère), peut résister à ce type d’analyse.

Les chiffrements par substitution polyalphabétique permettent de coder un même caractère en plusieurs caractères différents. Dans le cas du chiffrement *autokey*, qui a été conçu par Giovan Battista Bellaso en 1553, le caractère chiffré dépend du caractère en clair, de sa position et d’une clé de chiffrement. Même simple à mettre en œuvre, ce chiffrement a résisté avec succès aux attaques jusqu’en 1863, date à laquelle Friedrich Kasiski a publié une méthode [Kas63] pour le déchiffrer.

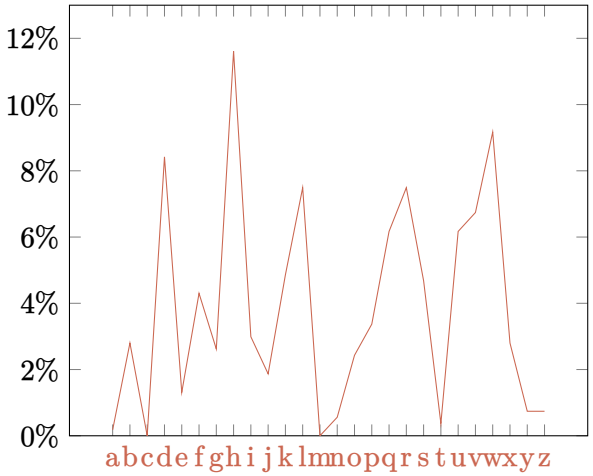
Au cours du 19e siècle, Kerckhoffs a publié deux articles de journaux “La Cryptographie Militaire” [Ker83a; Ker83b] dans lesquels il définit les six principes de conception pour les chiffres militaires. L’un des plus importants est “Il faut qu’il (le système) n’exige pas le secret, et qu’il puisse sans inconvénient tomber entre les mains de l’ennemi”. Ce principe a été repris par Claude Shannon

Texte clair	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Text chiffré	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

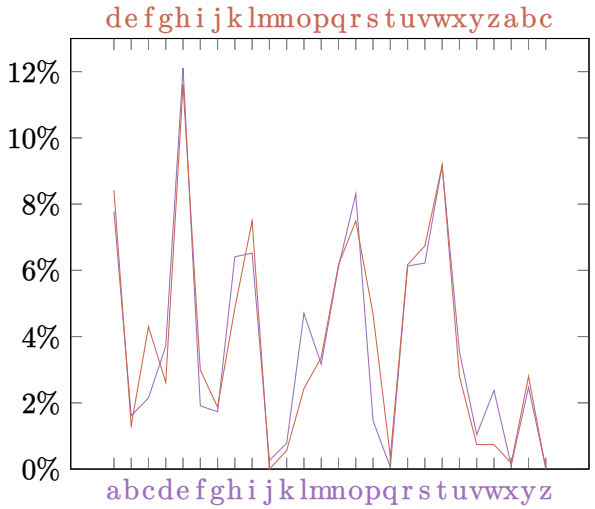
Table 1: *Exemple du chiffrement de César avec un décalage de 3. Chaque lettre de l’alphabet est remplacée par une version décalée de l’alphabet.*



(a) *L'analyse fréquencielle de Roméo et Juliette (version anglaise) (le texte est diponible à* `http://shakespeare.mit.edu/romeo_juliet/full.html` *).*



(b) *L'analyse fréquentielle des quatre premières phrases l'introduction (version anglaise) "Cryptography, from [. . .] not replaced." chiffrées avec le chiffrement de César.*



(c) *La superposition des deux analyses fréquentielles lorsque les symboles du texte chiffré sont décalés de 3 à gauche.*

Figure 1: *Une simple analyse de fréquence d'un texte chiffré avec César. Comme les lettres sont toujours chiffrées avec le même symbole, la fréquence des symboles doit correspondre à la distribution naturelle des lettres d'une langue spécifique, dans notre cas, la langue anglaise. Dans cet exemple, nous analysons les fréquences des lettres de Roméo et Juliette, ce qui donne la Figure 1a. Ensuite, nous analysons les fréquences des symboles d'un texte chiffré par César (Figure 1b). Comme montre la Figure 2c, les deux analyses correspondent assez bien lorsqu'on déplace l'analyse des fréquences du texte chiffré de 3 symboles vers la gauche. Comme nous savons que C n'effectue qu'un décalage, nous pouvons interpoler que la clé du texte chiffré est 3.*

sous la forme “l’ennemi connaît le système” dans [Sha49]. Depuis lors, les chiffrements ont été conçus de manière à ce que la clé soit le seul élément maintenu secret (en théorie) qui soit nécessaire pour déchiffrer les messages, en supposant que l’attaquant ait connaissance du système de chiffrement utilisé.

Le développement des moyens de communication, télégraphe, téléphone et radio, ainsi que les deux guerres mondiales ont porté la cryptographie à un rang supérieur. Elle devient alors un outil stratégique indispensable. En 1917, le ministre allemand des Affaires étrangères Arthur Zimmermann a envoyé un télégramme crypté à l’ambassadeur allemand au Mexique, Heinrich von Eckardt. Ce télégramme demande à l’ambassadeur de contacter les autorités mexicaines pour créer une alliance contre les États-Unis. Ce télégramme est déchiffré par les Britanniques et transmis aux États-Unis, ce qui accélère l’entrée du pays dans le conflit.

Alors que le chiffrement/déchiffrement était souvent effectué à la main pendant la Première Guerre mondiale, ce qui entraînait des erreurs et des retards, la Seconde Guerre mondiale a apporté l’automatisation nécessaire à l’utilisation généralisée de la cryptographie. L’utilisation massive des machines Enigma, Lorenz SZ 40 et SZ 42 par l’armée et les officiers supérieurs allemands a placé la cryptanalyse sur la liste des priorités des Alliés. Le développement de bombes cryptographiques par les Polonais, puis la reprise de leurs travaux par Alan Turing à Bletchley Park (Royaume-Uni), ont conduit à l’essor de la cryptographie et de la cryptanalyse à l’aide de systèmes informatiques. Cela a conduit, par exemple, aux ordinateurs électroniques Colossus Mark I et Colossus Mark II, conçus pour casser le code Lorenz.

De nos jours, la cryptanalyse a un historique sans cesse croissant de failles et d’attaques auxquelles les nouveaux chiffrements doivent faire face. Dans ce contexte, il devient important de trouver de nouvelles méthodes et de nouveaux outils permettant à la fois de trouver les failles lorsqu’elles existent et, si possible, les moyens de les corriger.

D’autre part, le développement de l’informatique et des techniques algorithmiques a permis de résoudre des problèmes de plus en plus complexes.

La programmation par contraintes (**CP**) est l’un des paradigmes de programmation visant à résoudre des problèmes $\mathcal{NP}$-difficiles. Elle a deux facettes : la modélisation, qui consiste à représenter les problèmes au moyen de variables et de contraintes, et la résolution, qui consiste à trouver les solutions. L’une des premières utilisations des contraintes se trouve dans la thèse de doctorat d’Ivan Sutherland, où il décrit le système Sketchpad [Sut63]. Ce système permettait aux utilisateurs de dessiner et de manipuler des objets graphiques contraints sur un écran d’ordinateur. Depuis lors, la programmation par contraintes a été développée et appliquée à une grande variété de domaines, allant de la planification et du routage de véhicules à la bio-informatique en passant par d’autres domaines. Deux autres approches déclaratives bien connues pour résoudre des problèmes $\mathcal{NP}$-difficiles sont la programmation linéaire en nombres entiers (**ILP**) et le problème de satisfaisabilité booléenne (**SAT**).

Dans des recherches récentes, ces approches déclaratives ont été efficacement appliquées aux

problèmes de cryptanalyse [SNC09; Mou+11; GMS16]. Les approches déclaratives permettent de modéliser les problèmes de cryptanalyse dans une formulation proche des outils utilisés par les cryptographes tout en bénéficiant de techniques de résolution éprouvées. Cette approche permet de concilier efficacité de calcul, simplification de la modélisation et fiabilité.

Résumés des Chapitres

Cette thèse a été réalisée dans le cadre du projet ANR Décrypt, dont les enjeux principaux sont :

- d'étudier et de comparer les performances des approches déclaratives existantes (**SAT**, **ILP** et **CP**) ;
- d'améliorer le passage à l'échelle des solveurs **CP** pour résoudre les problèmes de cryptanalyse symétrique ;
- concevoir de nouvelles procédures d'explication
- améliorer les attaques existantes et construire de nouveaux algorithmes qui résistent à ces attaques en utilisant les solveurs **CP**.

Nos principaux objectifs dans cette thèse étaient d'améliorer le passage à l'échelle des solveurs **CP**, ce qui est effectué dans les Chapitres 4 et 5, et d'améliorer les attaques existantes, ce qui est fait dans les Chapitres 4, 6 et 7.

État de l'art

Cette thèse commence par un aperçu scientifique des connaissances nécessaires à la compréhension de ce travail.

Cryptanalyse

Dans le Chapitre 1 nous présentons ce que sont les chiffrements et plus particulièrement les chiffrements symétriques par blocs qui sont les chiffrements sur lesquels nous avons travaillé. Nous introduisons également les notions d'attaques différentielles qui sont des attaques largement répandues de nos jours.

Programmation par contraintes

Le Chapitre 2 introduit les notions essentielles de la programmation par contraintes ainsi que les méthodes de résolution de base.

Modèles de **CP** existants pour résoudre problèmes de cryptanalyse différentielle

Le Chapitre 3 présente deux modèles **CP** existants qui ont été améliorés et adaptés dans le cadre de cette thèse, à savoir le modèle d'attaque différentielle à clés apparentées de Gérault *et al.* pour l'AES, et le modèle d'attaque boomerang de Delaune *et al.* pour Skinny.

Contributions

Cryptanalyse différentielle de Rijndael

Dans le Chapitre 4, nous étendons le modèle de Gerault *et al.* [Gér+20] du cas d'AES au cas Rijndael (AES et Rijndael sont présentés dans la Section 1.2.1), qui inclut davantage de configurations disponibles. Nous améliorons également les performances en entrelaçant mieux les deux étapes du processus de résolution, ce qui nous permet de résoudre toutes les instances de Rijndael sauf une. Nous donnons de nouvelles limites pour les distinguateurs de Rijndael et trouvons deux nouvelles attaques sur ce chiffrement.

Abstract XOR

Dans le Chapitre 5, nous introduisons une nouvelle contrainte globale, nommée *AbstractXOR*, permettant de mieux propager un ensemble d'équations XOR lors de la recherche d'une caractéristique différentielle tronquée. Nous prouvons que la vérification de la contrainte globale *AbstractXOR* est un problème $\mathcal{NP}$-complet et nous fournissons un algorithme en temps polynomial pour résoudre un problème relaxé. Nous appliquons nos nouvelles contraintes à la fois aux chiffrements AES et Midori (qui est présenté dans section 1.2.2). Nous concluons ce chapitre par une comparaison entre cet outil et un processus de résolution en deux étapes utilisant les solveurs **SAT** et **CP**.

Attaques boomerang automatiques sur Rijndael

Dans le Chapitre 6, nous adaptons le modèle d'attaque boomerang de Delaune *et al.* [DDV20] au chiffrement Rijndael en utilisant la modélisation effectuée dans le Chapitre 4. Ce Chapitre est composé de deux parties principales. Premièrement, nous décrivons comment nous l'adaptons à Rijndael. Ensuite, nous donnons de nouvelles limites pour les distinguateurs boomerang de Rijndael et une nouvelle attaque sur 9 tours de Rijndael avec la taille de bloc = 128 bits et la taille de clé = 160 bits.

Attaques rectangles automatiques sur les chiffrements Feistel: Application à WARP

Dans le Chapitre 7, nous étendons le modèle boomerang de Delaune *et al.* aux chiffres de Feistel en utilisant le travail de Boukerrou *et al.* [Bou+20]. Nous appliquons notre modélisation aux chiffrements WARP, TWINE et LBlock–s. Cette modélisation nous permet de trouver de nouveaux distinguateurs pour ces trois chiffrements. Nous intégrons également la phase `KeyRecovery` dans notre modèle pour le chiffrement WARP, ce qui nous permet de trouver une nouvelle attaque

rectangle pour ce chiffrement.

Conclusion

Enfin, nous concluons cette thèse de doctorat par un examen et une analyse de nos contributions et nous indiquons les éventuelles possibilités de recherches futures.

Publications.

- Les contributions décrites dans Differential Cryptanalysis of Rijndael ont été publiées dans [RS22; Rou+22].
- Les contributions décrites dans Abstract XOR ont été publiées dans [RS19; RS20].
- Les contributions décrites dans Automatic Boomerang Attacks on Rijndael sont en cours de soumission.
- Les contributions décrites dans Automatic rectangle attacks on Feistel ciphers: Application to WARP ont été publiées dans [LMR22].

Introduction

Cryptography, from the Greek words *κρυπτός* (kruptós, "hidden, secret") and *γράφειν* (graphein, "to write") is the practice and study of secret communication techniques. Before the advent of information systems, the role of cryptography was mainly to provide systems of confidentiality. Cryptography has been used since antiquity in military operations, diplomacy and espionage. One of the most famous ciphers of the time being Caesar's cipher (see Table 2), which consists in shifting the letters of the alphabet by a fixed number of elements, the punctuation elements are not replaced. For example, the sentence: `Caesar's cipher` encrypted with a shift of 3 (`A` becomes `D`) is `Fhdvdu'v flskhu`.

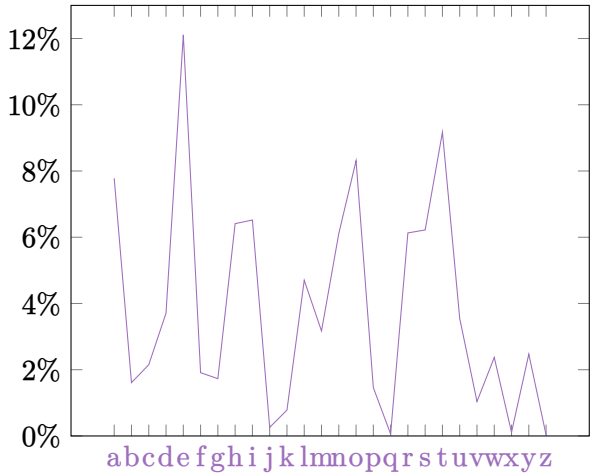
Caesar's cipher is called monoalphabetic, each character is always replaced by the same character, *e.g.* the character `A` will always be replaced by the character `D`. Monoalphabetic ciphers are very susceptible to frequency analysis attacks (Figure 2). As the letters of the alphabet do not appear uniformly in languages, it is possible to perform frequency analysis on the ciphertext symbols to establish matches. The polyalphabetic substitution cipher approach, such as the autokey cipher [Bel53] (also known as Vigenère cipher), can withstand this type of analysis.

Polyalphabetic substitution ciphers allow the same character to be encoded into several different characters. In the case of the autokey cipher, which was designed by Giovan Battista Bellaso in 1553, the encrypted character depends on the plaintext character, its position and an encryption key. Even simple to implement the cipher resisted sucessfully to attacks until 1863 when Friedrich Kasiski published a method [Kas63] for deciphering it.

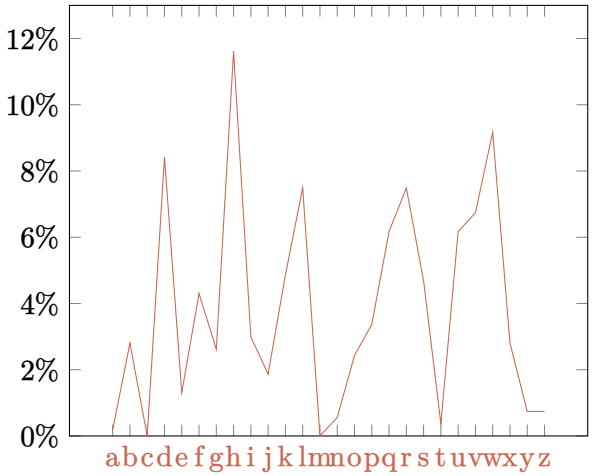
During the 19th century, Kerckhoffs published two journal articles "La Cryptographie Militaire" [Ker83a; Ker83b] in which he defined the six design principles for military ciphers. One of the most important state is "It (the system) must not require secrecy, and must be able to fall into the hands of the enemy without difficulty". This has been taken by Claude Shannon as "the enemy knows the system" in [Sha49]. Since then, ciphers have been designed so that the key is the only element (in theory) that can be used to decipher messages, assuming that the attacker has knowledge of the encryption system used.

Plaintext	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Ciphertext	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

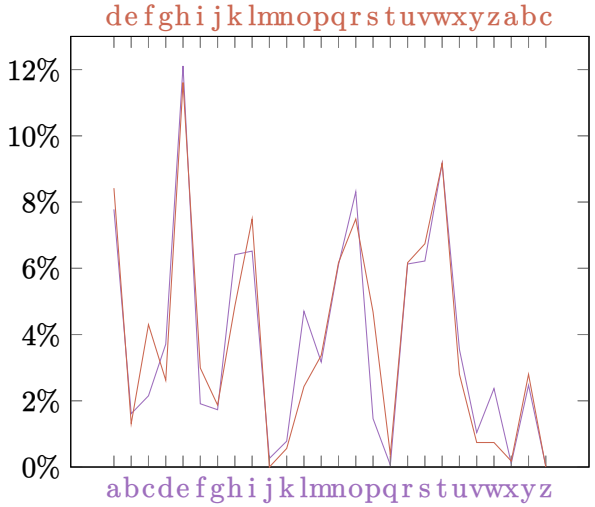
Table 2: *Exemple of caesar's cipher when the shift is 3. Each letter of the alphabet is replaced with a shifted version of the alphabet.*



(a) *The frequency analysis of Romeo and Juliet (the text is available at* `http://shakespeare.mit.edu/romeo_juliet/full.html` *).*



(b) *The frequency analysis of the four first sentences of this thesis "Cryptography, from [. . .] not replaced." ciphered with caesar.*



(c) *Superposition of the two frequency analysis when shifting the ciphertext symbols of 3 to the left.*

Figure 2: *A simple frequency analysis of a ciphertext ciphered with caesar. As letters are always ciphered to the same symbol, the frequency of the symbols should match the natural distribution of the letters of a specific language, in our case the english language. In this example we analyze the letter frequencies of Romeo and Juliet, which gives the Figure 2a. Then we analyze the symbol frequencies of a ciphertext ciphered by caesar (Figure 2b). As shown in Figure 2c, the two analysis match quite well when shifting the frequency analysis of the ciphertext of 3 symbols to the left. As we know that caesar only performs a shift, we can interpolate that the key of the ciphertext is 3.*

The development of the means of communication, telegraph, telephone, and radio, as well as the two world wars brought cryptography to a higher rank. It then became an indispensable strategic tool. In 1917, the German Foreign Minister Arthur Zimmermann sent an encrypted telegram to the German ambassador to Mexico, Heinrich von Eckardt. The telegram asked the ambassador to contact the Mexican authorities to create an alliance against the United States. The telegram was deciphered by the British and forwarded to the United States, which accelerated the country's entry into the conflict.

While encryption/decryption was often done by hand during the First World War, leading to errors and delays, the Second World War brought the automation necessary for the widespread use of cryptography. The massive use of the Enigma, Lorenz SZ 40 and SZ 42 machines by the German army and senior officers put cryptanalysis on the priority list of Allies. The development of cryptographic bombs by the Poles, and the subsequent revival of their work by Alan Turing at Bletchley Park (UK), led to the rise of cryptography and cryptanalysis using computer systems. This led, for example, to the Colossus Mark I and Colossus Mark II electronic computers, designed to break the Lorenz code.

Nowadays, cryptanalysis has an ever-growing history of flaws and attacks that new ciphers have to deal with. In this context, it becomes important to find new methods and tools to both find flaws when they exist and, if possible, ways to fix them.

On the other hand, the development of computer science and algorithmic techniques has made it possible to solve increasingly complex problems.

Constraint programming (**CP**) is one of the programming paradigms aimed at solving $\mathcal{NP}$-Hard problems. It has two faces: modelling, which consists in representing the problems by means of variables and constraints, and solving, which consists in finding the solutions. One of the first uses of constraints was in the PhD thesis of Ivan Sutherland, where he described the Sketchpad system [Sut63]. This system allowed users to draw and manipulate constrained graphical objects on a computer screen. Since then, constraint programming has been developed and applied to a wide variety of fields, ranging from planning and vehicle routing to bioinformatics and other areas. Two other well known declarative approaches for solving $\mathcal{NP}$-Hard problems are Integer Linear Programing (**ILP**) and Satisfiability Boolean formula (**SAT**).

In recent research, these declarative approaches have been effectively applied to cryptanalysis problems [SNC09; Mou+11; GMS16]. Declarative approaches allow cryptanalysis problems to be modelled in a formulation close to the tools used by cryptographers while benefiting from proven solution techniques. This approach allows to reconcile computational efficiency, modelling simplification and reliability.

This thesis was carried out within the framework of the ANR Decrypt project, the main issues of which are:

- to study and compare the performances of existing declarative approaches (**SAT**, **ILP** and **CP**);

- to improve the scaling of **CP** solvers to solve symmetric cryptanalysis problems;
- to design new explanation procedures;
- to improve existing attacks and build new algorithms that resist these attacks using **CP** solvers.

Our main goals in this PhD thesis were to improve the scaling of **CP** solvers, which is done in Chapters 4 and 5, and to improve existing attacks, which is done in Chapters 4, 6 and 7.

This thesis starts with a scientific overview of the knowledge needed to understand this work.

In Chapter 1 we present what ciphers are and more specifically symmetric block ciphers which are the ciphers we have been working on. We also introduce the notions of differential attacks which are widely known attacks nowadays.

Chapter 2 introduces the essential notions of constraint programming as well as the basic solution methods.

Chapter 3 introduces two already existing **CP** models that have been improved and adapted in this PhD thesis, *i.e.*, the related-key differential attack model of Gérault *et al.* for the AES, and the boomerang attack model of Delaune *et al.* for Skinny.

In Chapter 4 we extend the model of Gerault et al. [Gér+20] from the AES case to the Rijndael case (AES and Rijndael are presented in Section 1.2.1), which includes more available configurations. We also improve the performance by better interleaving the two Steps of the solution process which allows us to solve all Rijndael instances but one. We give new bounds for Rijndael distinguishers and find two new attacks on this cipher.

In Chapter 5 we introduce a new global constraint, named *AbstractXOR*, allowing to better propagate a set of xor equations when searching for a truncated differential characteristic. We prove that checking the global constraint *AbstractXOR* is an $\mathcal{NP}$-complete problem and we provide a polynomial time algorithm to solve a relaxed problem. We apply our new constraints on both Rijndael and Midori (which is presented in section 1.2.2) ciphers. We conclude this chapter by a comparison between this tool and a two Step solution process using both **SAT** and **CP** solvers.

In Chapter 6 we adapt the boomerang attack model of Delaune et al. [DDV20] to the Rijndael encryption using the modelling done in Chapter 4. The chapter is composed of two main parts. Firstly we describe how we adapt it to Rijndael. Secondly we give new bounds for boomerang distinguishers of Rijndael and one new attack on 9 rounds of Rijndael with the block size $= 128$ and the key size $= 160$.

In Chapter 7 we extend the boomerang model of Delaune *et al.* to the Feistel ciphers using the work of Boukerrou *et al.* [Bou+20]. We apply our modelling to the WARP, TWINE and LBlock–s ciphers. This modelling allows us to find new distinguishers for the three ciphers. We also integrate the `KeyRecovery` phase in our model for the WARP cipher, which allows us to find a new rectangle attack for this cipher.

Finally, we conclude this PhD thesis with a review and analysis of our contributions and indicate possible further research opportunities.

Publications.

- The contributions described in Differential Cryptanalysis of Rijndael have been published in [RS22; Rou+22].
- The contributions described in Abstract XOR have been published in [RS19; RS20].
- The contributions described in Automatic Boomerang Attacks on Rijndael are currently under submission.
- The contributions described in Automatic rectangle attacks on Feistel ciphers: Application to WARP have been published in [LMR22].

Notations

For iterative block cipher states, we define the following index notations:

- i stands for the round index,
- j stands for the row index,
- k stands for the column index.

Unless stated otherwise, all the data collections are indexed from 0.

Symbol	Description
$\oplus$	is the bitwise xor operation
$x[k]$	is the k^{th} value of the sequence x. $X[i]$ stands for the binary matrix of the state at round i when X denotes the state of an iterative block cipher.
$x[y..z]$	is the subsequence from the y^{th} to the z^{th} (included) element of x
$x[y..z[$	is the subsequence from the y^{th} to the z^{th} (excluded) element of x
$M[j,k]$	is the element at the j^{th} row and the k^{th} column of the matrix M
$S[i,j,k]$	is the element at the i^{th} round, the j^{th} row and the k^{th} column of the state S
$Pr[E]$	is the probability of the event E
$\#S$	is the cardinality of the set S
$[x;\ y]$	is the closed range from x to y
$[x;\ y[$	is the half-open range from x to y (exclusive)
$A\|\|B$	is the binary string concatenation of A and B

Part

I

Context

Chapter

1 Cryptanalysis

Contents

Cryptography is the science of making communications secure against possible threats. It has many goals such as:

- data integrity, *i.e.* ensuring accuracy and consistency of the data,
- data authenticity, *i.e.* ensuring the data sources,
- data confidentiality, *i.e.* ensuring that data is accessible only to authorized entities,
- non-repudiation, *i.e.* ensuring that an author cannot revoke the paternity of his production, and his ownership cannot be disputed.

Cryptanalysis is the mirror twin of cryptography since its aim is to check cryptography rules by trying to break them. In our case, we are mainly focused on the analysis of symmetric ciphers, and our goal is to challenge ciphers about the first rule: data confidentiality.

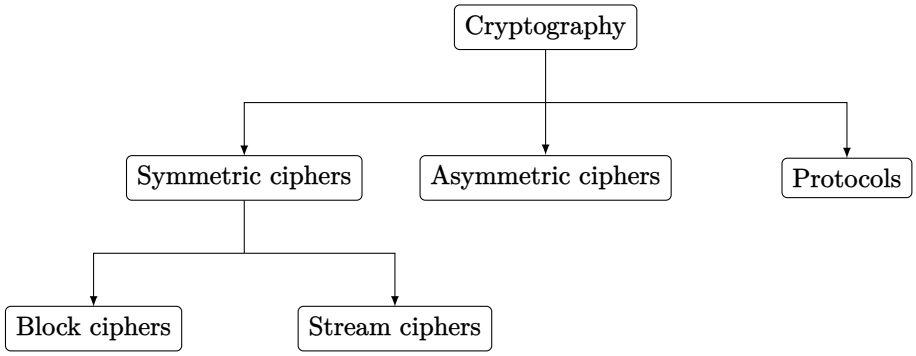


Figure 1.1: *Overview of the field of cryptology [Chr10, p.3]*

Since our work is only focused on symmetric ciphers, we introduce in this chapter the notion of symmetric ciphers. We also present two main classes of symmetric cipher constructions that are **S**ubstitution–**P**ermutation **N**etworks (**SPN**) and **Feistel** constructions and we describe the different ciphers considered in the thesis, *i.e.* Rijndael, Midori, WARP, TWINE and LBlock–s. Finally, we introduce differential cryptanalysis [BS91] which exploits relations between input and output differences to mount attacks.

1.1 Symmetric Ciphers

Symmetric ciphers, or secret key ciphers, are so called in contrast to asymmetric ciphers. While asymmetric ciphers use separate keys for ciphering and deciphering, symmetric ciphers use the same (secret) key to cipher and decipher messages. The key for symmetric ciphers is called the secret key because it can decipher any message ciphered by it (provided the same cipher is used) and thus must be kept secret. Two main kinds of symmetric ciphers are block ciphers and stream ciphers (Figure 1.1). While block ciphers work on fixed length plaintexts, stream ciphers [PP10] work on plaintexts with arbitrary lengths. In this PhD. thesis, we focus on block ciphers. Indeed, block ciphers may be transformed into stream ciphers by using modes of operations.

1.1.1 Modes of operation

Block ciphers can only cipher plaintexts with a fixed size. In practice we want to cipher data for which we do not control the size. Modes of operation are functions that transform block ciphers into stream ciphers by indicating how to apply the block ciphers to a plaintext stream.

The example below, which is not secure, uses the *Electronic Codebook* (**ECB**) encryption mode: each input block is therefore a sub-part of size n of the original message and each ciphered block corresponds to the ciphering of the corresponding input block.

Example 1.1

We try to cipher the message: `"helloworld"`, with a custom - unsafe - block cipher with $n = 32$ bits. If we represent the message with the ASCII encoding, we can encode each symbol with 8 bits and the decimal representation of the message becomes:

$$[104, 101, 108, 108, 111, 119, 111, 114, 108, 100]$$

As our block cipher works on $n = 32$ bits blocks, we will cipher the message by grouping the bits in chunks of 32 bits (note that we must fill the last chunk with defined values to reach the n bits), the message is then sliced in three sub-parts:

$$\texttt{hell} \quad \texttt{owor} \quad \texttt{ldaa}$$

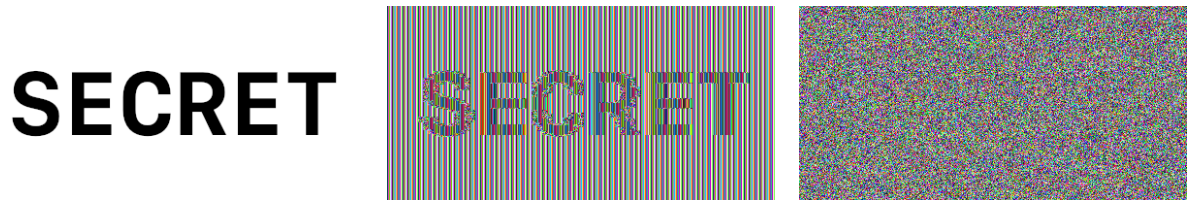
$$[[104, 101, 108, 108], \quad [111, 119, 111, 114], \quad [108, 100, 97, 97]]$$

After applying the cipher on each block we obtain three ciphered blocks of 32 bits :

$$[[117, 115, 97, 98], \quad [98, 107, 100, 104], \quad [121, 114, 112, 113]]$$

Finally we concatenate the three ciphered blocks, convert them with our initial ASCII encoding and get the following ciphered text: `usabbkdhyrpq`.

Modes of operation must be strongly defined as they may introduce weakness in the cipher process. As shown in Figure 1.2, the plaintext message "SECRET" represented in the picture can be seen after being ciphered by AES and the *Electronic CodeBook* (**ECB**) [MVV18] mode of operation, while the same message looks like random noise after being ciphered by AES and the *Cipher Block Chaining Mode* (**CBC**) [Ehr+78]. The main known modes of operation are: the *Electronic CodeBook*, the *Cipher Block Chaining Mode*, the *Output Feedback Mode* (**OFB**) [Mod80], the *Cipher Feedback Mode* (**CFB**) [Mod80] and the *Counter Mode* (**CTR**) [Dwo01].



(a) *The plaintext* bitmap *picture.*

(b) *The ciphertext* bitmap *picture ciphered with* AES *and* ECB. *We observe that the mode of operation is not efficient, since anyone can see the original pattern through the ciphertext directly.*

(c) *The ciphertext* bitmap *picture ciphered with* AES *and* CBC, *here the picture appears to be more like random noise and doesn't give much informations to the attacker.*

Figure 1.2: *The ciphering of a* bitmap *picture using* AES *and two mode of operations:* ECB *and* CBC.

1.1.2 Block ciphers

Block ciphers are so called because they cipher messages in portions of a fixed size. More formally, a block cipher is defined by a ciphering function E that takes two inputs, *i.e.* a key K of size k bits and a block input M, called plaintext, of size n bits. It returns a ciphertext C of size n bits.

Definition 1.1: Block cipher

$$E_K(M) = E(K, M) : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n \text{ and } C = E_K(M)$$

The deciphering function associated to E, denoted D, is the inverse function of E. It takes as input a ciphertext C of size n, the same key (used for ciphering) K of size k and returns the original plaintext M of size n.

Definition 1.2: Associated deciphering function of a block cipher

$$D_K(C) = D(K, C) = E_K^{-1}(C) : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$$
$$\text{with } M = D_K(C) \text{ and } \forall K, D_K(E_K(M)) = M$$

1.1.3 Building a cipher function

In 1949, Shannon identifies two important properties in a cipher method:

the diffusion property: which means how much the plaintext bits are shuffled with the ciphertext bits,

the confusion property: which means how much it is complicated to find back the key bits from the plaintext bits.

To ensure that these two properties are satisfied in a cipher algorithm, a common approach is to use simple cipher function composition to create a more complex cipher function. One of the most common approaches is to create an iterative block cipher. A round function *round* is repeated for a given number of times N_r, hence the cipher function is $E_K = round_{N_r-1} \circ ... \circ round_1 \circ round_0$. To avoid repeating the exact same function N_r times it is possible to add sub-keys generated from the master key K between each *round* function. The cipher becomes:

$$cipher_0 = round_0(M, k_0)$$
$$cipher_i = round_i(cipher_{i-1}, k_i)$$
$$E_K(M) = cipher_{N_r-1}$$

To generate such sub-keys $k_0, k_1, ..., k_{N_r-1}$ cipher algorithms include a `KeySchedule` function (Figure 1.3) that generates sub-keys from the initial key K. In such case, the initial key is called *master key* and the N_r keys are called *sub-keys* or *round-keys*. Moreover it is necessary to add *round constants* to avoid slide attacks [BW99] that could differentiate each round.

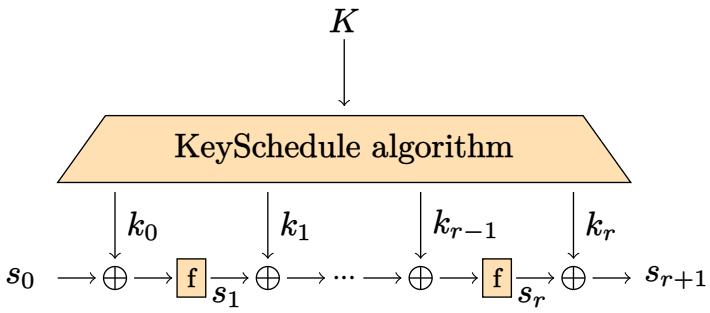


Figure 1.3: *Iterative construction of a block cipher [Jea15].*

1.2 Substitution-Permutation Network

SPN, for **S**ubstitution-**P**ermutation **N**etwork, is a well known family of block ciphers that works with two components which are the substitution layer and the permutation layer. The alternation of the substitution layer and the permutation layer multiple times ensures the diffusion and the confusion properties because of the avalanche effect [Fei73].

The substitution layer is generally composed of parallel S-Boxes, which are bijective. Each S-Box takes as input a fixed sequence of m bits, generally a byte (8 bits) or a nibble (4 bits), of the text and transforms it into another word of m bits. It is possible to encode the S-Boxes with a lookup table of 2^m words of m bits each, in such case the output value $sx = \texttt{SB}[x]$ where `SB` is the lookup table. S-Boxes are an important function in cipher algorithms. They must be non-linear and carefully designed to avoid cryptanalysis attacks.

The permutation layer is typically classified as *compression*, *expansion* and *straight*. It is classified as *compression* when it takes more input bits than it generates output bits, on the opposite it is classified as *expansion* when it takes less input bits than it generates output bits and it is classified as *straight* when it has as many input bits as it generates output bits. In **SPN**, since the round function must be invertible, the only permutation layers that can be used are straight permutation layers. In the case of the Rijndael [DR99], the permutation layer is composed of a shift row function which moves the bytes of the internal state and a mixcolumn operation which multiplies the internal state by a given matrix M.

1.2.1 Rijndael

The Rijndael [DR99] family is a block cipher family denoted $\text{Rijndael}_{C_{len}-K_{len}}$ where C_{len} is the block size and K_{len} is the key size. Both C_{len} and K_{len} must be in $\{128, 160, 192, 224, 256\}$. The standardized version AES [01] is less permissive and only accepts blocks of 128 bits and keys of 128, 192 or 256 bits. Each instance varies according to the block size and to the key size but the ciphering process is the same for all variants, except for the `ShiftRow` operation (given in Table 1.1) and the number of rounds (given in Table 1.2).

For all the versions, the current block at the input of the round i is represented by a $4 \times N_b$ matrix of bytes $X[i]$ where $N_b = (C_{len}/32)$ is the number of columns and where each byte at

Row	0 1 2 3
$C_{len} = 128$	0 1 2 3
$C_{len} = 160$	0 1 2 3
$C_{len} = 192$	0 1 2 3
$C_{len} = 224$	0 1 2 4
$C_{len} = 256$	0 1 3 4

Table 1.1: `ShiftRow` *table* $P_{C_{len}}$. *This table specifies the required number of byte shifts to the left according to the row number and* C_{len}.

C_{len}	128	160	192	224	256
$K_{len} = 128$	10	11	12	13	14
$K_{len} = 160$	11	11	12	13	14
$K_{len} = 192$	12	12	12	13	14
$K_{len} = 224$	13	13	13	13	14
$K_{len} = 256$	14	14	14	14	14

Table 1.2: *The number of rounds* N_r *of* $\text{Rijndael}_{C_{len}-K_{len}}$.

Notation	Description
$X[i]$	the state at the beginning of round i. Note that $X[i]$ is also the state after applying the `AddRoundKey` function on the previous round $X[i-1]$.
$SX[i]$	the state of round i, after applying `SubBytes`.
$Y[i]$	the state of round i, after applying `ShiftRow`.
$Z[i]$	the state of round i, after applying `MixColumns`.
$RK[i]$	the subkey of round i.

Table 1.3: *The summary of the notations of the different internal states of* Rijndael.

row j and column k is denoted by $X[i, j, k]$. The other states use the same notations and are depicted in Table 1.3.

The round function, repeated $N_r - 1$ times, involves four elementary mappings, all linear except the first one. Round i consists of the following transformations:

- `SubBytes`. A bytewise transformation is applied on each byte of the current block using an 8-bit to 8-bit non linear S-box, denoted by `S`:

$$SX[i, j, k] = \mathtt{S}[X[i, j, k]], \forall j \in [0;\ 3], \forall k \in [0;\ N_b[.$$

- `ShiftRows`. A linear mapping rotates to the left all the rows of the current matrix $SX[i]$. The values of the shifts denoted $P_{C_{len}}$ (given in Table 1) depend on C_{len}:

$$Y[i, j, k] = SX[i, j, (P_{C_{len}}[j] + k) \mod N_b], \forall j \in [0;\ 3], \forall k \in [0;\ Nb[.$$

- `MixColumns` is a linear multiplication of each column of the current state by a constant matrix M in the Galois field $GF(2^8)$, that provides the corresponding column of the new state. For a given column $k \in [0;\ N_b[$, if we denote by $\otimes$ the multiplication in $GF(2^8)$, we

have:

$$Z[i,j,k] = \bigoplus_{j=0}^{3} M[l,j] \otimes Y[i,j,k] \text{ with } M = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}, \forall k \in [0;\ N_b[$$

- `AddRoundKey` performs a bitwise xor between the subkey RK_{i+1} and the current state Z_i:

$$X[i+1,j,k] = Z[i,j,k] \oplus RK[i+1,j,k], \forall j \in [0;\ 3], \forall k \in [0;\ N_b[.$$

The subkeys RK_i are generated from the master key K using a `KeySchedule` algorithm composed of byte shifting, S-Box substitutions and xors which is fully described in Algorithm 1.1. We denote by $N_k = K_{len}/32$ the number of columns of the master key K. Note that each subkey RK_i is extracted from a main register WK in the following way:

$$RK[i,j,k] = WK[j, i \times N_b + k], \forall j \in [0;\ 3], \forall k \in [0;\ Nb[.$$

Those $N_r - 1$ rounds are surrounded at the top by an initial key addition with the subkey RK_0 and at the bottom by a final transformation composed by a call to the round function where the `MixColumns` operation is omitted. The global function is represented in Figure 1.4 for the $C_{len} = 128$ and $K_{len} = 128$ variant.

Maximum Distance Separable Property

To ensure good diffusion in the cipher, the matrix associated to the matrix multiplication should have certain mathematical properties. The matrix M of Rijndael has the *Maximum Distance Separable* (**MDS**) property, which ensures that there is:

- 0 modification, when the input column $Y[i,*,k]$ is zero (all bytes are zero),
- at least 5 out of 8 non-zero bytes, when $Y[i,*,k]$ has at least one non-zero byte.

In other words, at each round i and each column $k \in [0;\ N_b[$, we have the following property:

$$\sum_{j=0}^{3} \texttt{is\_non\_zero}(Y[i,j,k]) + \texttt{is\_non\_zero}(Z[i,j,k]) \in \{0,5,6,7,8\}$$

where `is_non_zero(`B`)` is equal to 0 if the byte B is equal to 0, and to 1 otherwise.

input : A key matrix K of $4 \times N_b$ bytes
output : The expanded key WK of $4 \times N_b(N_r + 1)$ bytes

for $k \in [0;\ N_b[$ **and** $j \in [0;\ 3]$ **do**
    WK[j, k] $\leftarrow$ K[j, k]
end
for $k \in [N_b;\ N_b \times (N_r + 1)[$ **do**
    if $k \mod N_k = 0$ **then**
        WK[0, k] = WK[0, k - N_k] $\oplus$ S[WK[1, k - 1]] $\oplus$ RC_i
        for $j \in [1;\ 3]$ **do**
            WK[j, k] = WK[j, k - N_k] $\oplus$ S[WK[$(j + 1) \mod 4$, k - 1]]
        end
    else if $k > 6 \wedge k \mod N_k = 4$ **then**
        for $j \in [0;\ 3]$ **do**
            WK[j, k] = WK[j, k - N_k] $\oplus$ S[WK[j, k - 1]]
        end
    else
        for $j \in [0;\ 3]$ **do**
            WK[j, k] = WK[j, k - N_k] $\oplus$ WK[$(j + 1) \mod 4$, k - 1]
        end
    end
end
return WK

Algorithm 1.1: *Rijndael* `KeySchedule` *function. The round constants* RC_i *is given in appendix (Figure A.1). In differential cryptanalysis these constants are ignored since they are cancelled when applying the xor operator.*

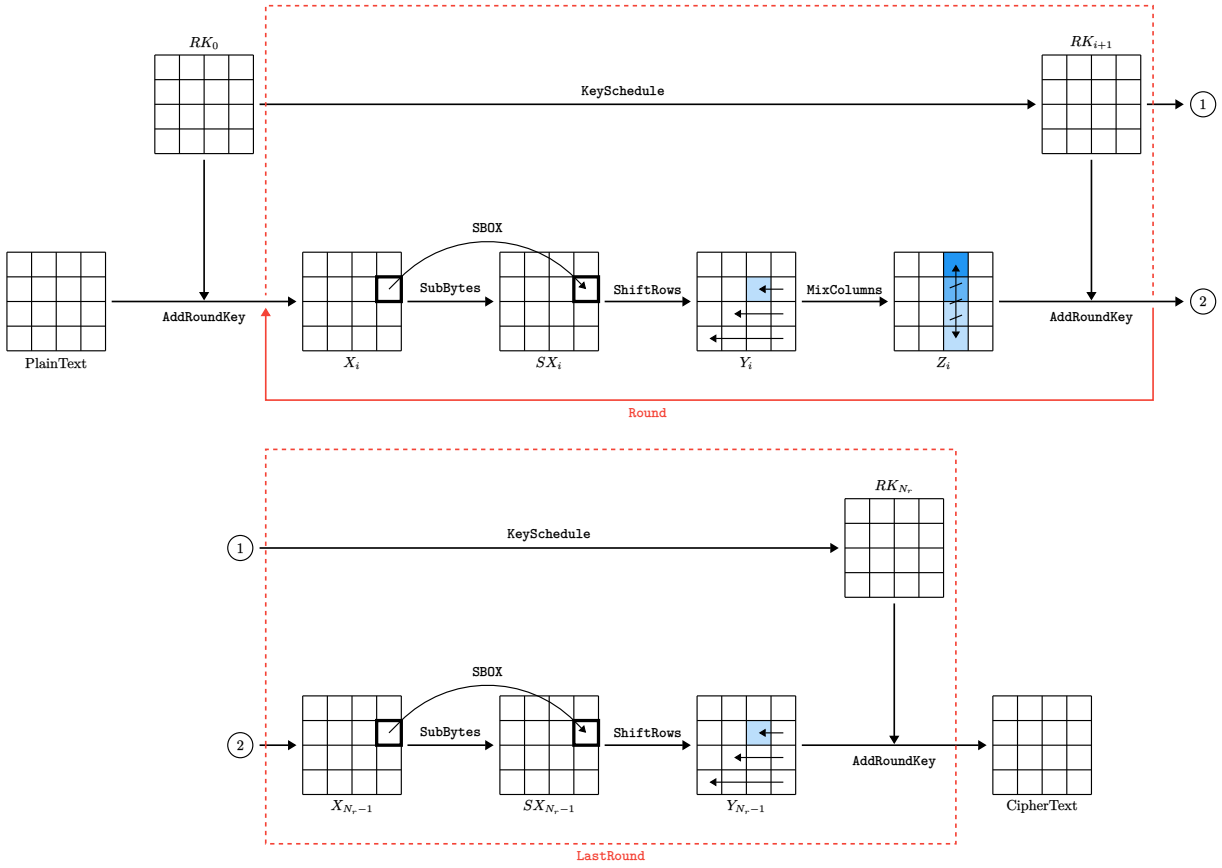


Figure 1.4: *Schema of* Rijndael$_{128-128}$.

	block size (C_{len})	key size	cell size	number of rounds (N_r)
Midori$_{64}$	64	128	4 (nibble)	16
Midori$_{128}$	128	128	8 (byte)	18

Table 1.4: *The possible configurations of* Midori.

Notation	Description
$X[i]$	the state at the beginning of round i. Note that $X[i]$ is also the state after applying the `KeyAdd` function on the previous round $X[i-1]$.
$SX[i]$	the state of round i, after applying `SubCell`.
$Y[i]$	the state of round i, after applying `ShuffleCell`.
$Z[i]$	the state of round i, after applying `MixColumns`.
$RK[i]$	the subkey of round i.

Table 1.5: *The summary of the notations of the different internal states of* Midori.

1.2.2 Midori

Midori [Ban+15] is a block cipher family denoted Midori$_{C_{len}}$ where C_{len} is the block size. C_{len} must be either 64 or 128. The possible configurations for Midori are shown in Table 1.4.

The number of rounds N_r depends on the block size C_{len} and is either 16 or 18. For all the versions, the current block at the input of the round i is represented by a 4×4 matrix of nibbles (resp. bytes) for Midori$_{64}$ (resp. Midori$_{128}$) and is denoted $X[i]$ (the other states are depicted in Table 1.5). Each cell (either nibble or byte depending on the block size) at row j and column k is denoted by $X[i,j,k]$. The round function, repeated $N_r - 1$ times, involves four elementary mappings, all linear except the first one. Round i consists of the following transformations:

- `SubCell`. A bytewise transformation is applied on each cell of the current block using a 4-bit to 4-bit non linear S-box $\texttt{Sb}_0$ (given in Table 1.6) for Midori$_{64}$. Midori$_{128}$ uses four 8-bit to 8-bit non linear S-Boxes $\texttt{SSb}_{[x]}$ (available in [Ban+15]):

$$SX[i,j,k] = \texttt{Sb}_0[X[i,j,k]], \forall j \in [0;\ 3], \forall k \in [0;\ 3] \text{ for Midori}_{64},$$
$$SX[i,j,k] = \texttt{SSb}_{[(j\times 4+k) \mod 4]}[X[i,j,k]], \forall j \in [0;\ 3], \forall k \in [0;\ 3] \text{ for Midori}_{128}.$$

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$\texttt{Sb}_0[x]$	C	A	D	3	E	B	F	7	8	9	1	5	0	2	4	6

Table 1.6: *4-bit bijective S-Box* $\texttt{Sb}_0$ *in hexadecimal form.*

- `ShuffleCell`. Each cell of the state is permuted as follows:

$$\text{if } Y[i] = \begin{bmatrix} s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ s_8 & s_9 & s_{10} & s_{11} \\ s_{12} & s_{13} & s_{14} & s_{15} \end{bmatrix} \text{ then } SX[i] = \begin{bmatrix} s_0 & s_{10} & s_5 & s_{15} \\ s_{14} & s_4 & s_{11} & s_1 \\ s_9 & s_3 & s_{12} & s_6 \\ s_7 & s_{13} & s_2 & s_8 \end{bmatrix}$$

- `MixColumns` is a linear multiplication of each column of the current state by a constant matrix M in the Galois field $GF(2^4)$ (resp. $GF(2^8)$) for Midori_{64} (resp. Midori_{128}), that provides the corresponding column of the new state. For a given column $k \in [0;3]$, we have:

$$Z[i,j,k] = \bigoplus_{j=0}^{3} M[l,j] \otimes Y[i,j,k] \text{ with } M = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

- `KeyAdd` performs a bitwise xor between the subkey RK_i of round i and the current state Z_i:

$$X[i+1,j,k] = Z[i,j,k] \oplus RK[i,j,k], \forall j \in [0;3], \forall k \in [0;3].$$

The subkeys RK_i are generated from the master key K using a round key generation algorithm. For Midori_{64}, the master key K is decomposed into two subkeys $K[0]$ and $K[1]$, where $K[0]$ is composed of the bits of first half of K and $K[1]$ of the second half, then:

$$WK = K[0] \oplus K[1]$$
$$RK[i] = K[(i \mod 2)] \oplus \alpha[i]$$

where $\alpha[i]$ is a 4×4 constant binary matrix (available in [Ban+15]).

For Midori_{128} we have:

$$WK = K$$
$$RK[i] = K \oplus \alpha[i]$$

Those $Nr - 1$ rounds are surrounded at the top by an initial key addition with WK and at the bottom by a final transformation composed by a call to the round function where the `ShuffleCell` and `MixColumns` operations are omitted as depicted in Figure 5.4.

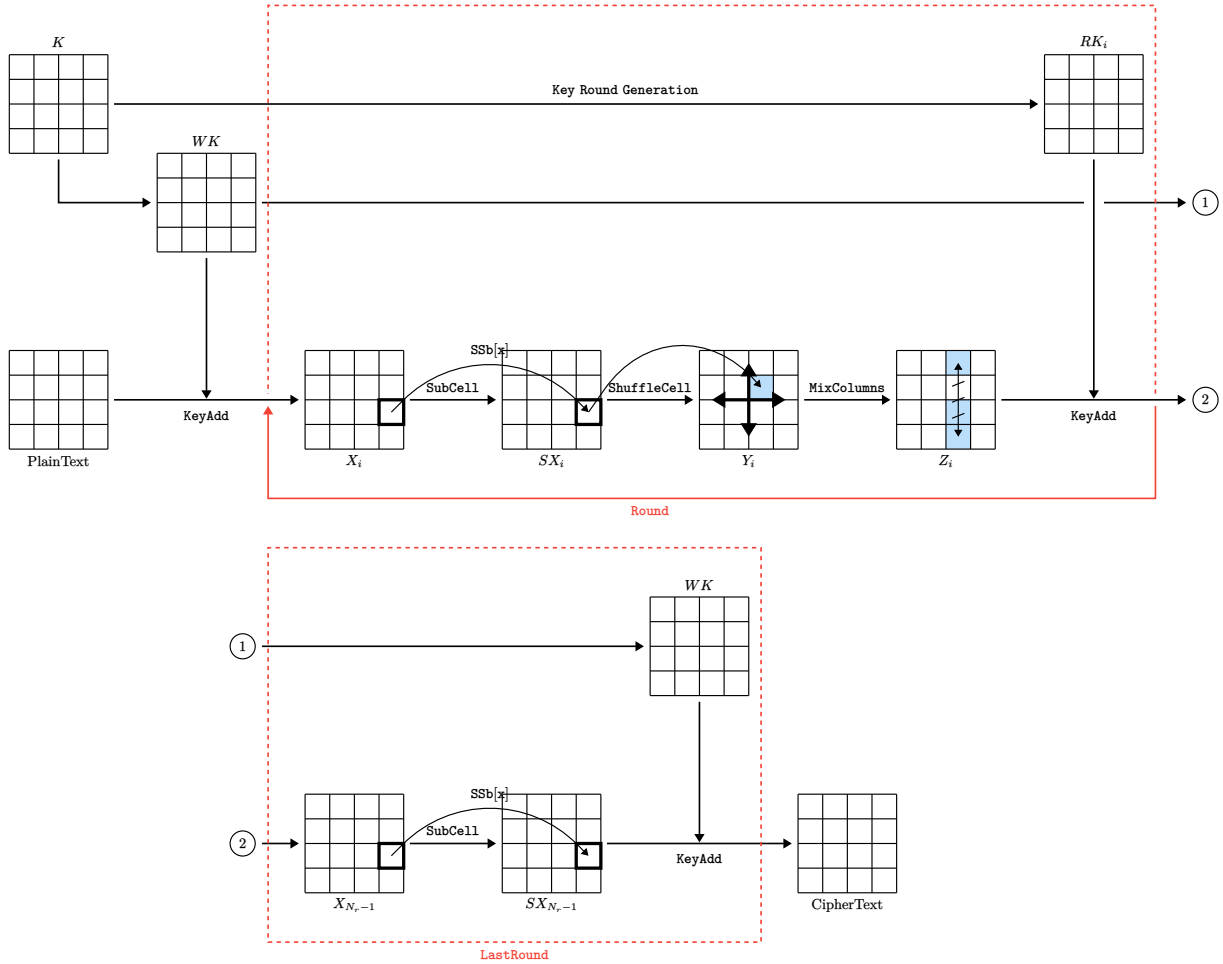


Figure 1.5: *Schema of* Midori_{128}.

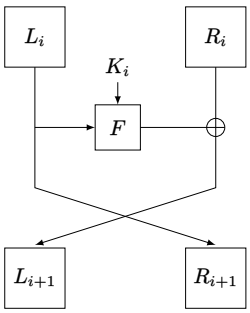


Figure 1.6: *Feistel diagram*

Quasi-Maximum Distance Separable Property

As with Rijndael, Midori uses a multiplication matrix M for the `MixColumns` operation. In the case of Midori, the multiplication matrix contains only 0's and 1's which does not allow the **MDS** property. However, the Midori matrix M has the *Quasi*-**MDS** property, which ensures that there is:

- 0 modification, when the input column $Y[i, *, k]$ is zero (all bytes are zero),
- at least **4**[1] out of 8 non-zero bytes, when $Y[i, *, k]$ has at least one non-zero byte.

In other words, at each round i and each column $k \in [0;\ N_b[$, we have the following property:

$$\sum_{j=0}^{3} \texttt{is\_non\_zero}(Y[i,j,k]) + \texttt{is\_non\_zero}(Z[i,j,k]) \in \{0,4,5,6,7,8\}$$

1.3 Feistel networks

Feistel networks [Fei74] are a second family of block ciphers. In the same way as **SPN**, Feistel networks use the notion of round functions, but the overall construction is done differently: the plaintext is sliced into two parts called L and R (see Figure 1.6). At each iteration i, the round function F is applied to a part of the internal state and its output is XORed with the second part of the internal state, then the two parts are inverted:

$$L_{i+1} = F(K_i, L_i) \oplus R_i$$
$$R_{i+1} = L_i$$

Whereas **SPN** round functions must be invertible, Feistel network allows to use non-invertible round functions as the decryption function can be computed with:

$$L_i = R_{i+1}$$
$$R_i = F(K_i, R_{i+1}) \oplus L_{i+1}$$

The Feistel network is able to transform functions into permutations.

[1] Versus **5** for the **MDS** property.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[x]	C	A	D	3	E	B	F	7	8	9	1	5	0	2	4	6

Table 1.7: *4-bit S-box of* WARP.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(x)$	31	6	29	14	1	12	21	8	27	2	3	0	25	4	23	10

x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\pi(x)$	15	22	13	30	17	28	5	24	11	18	19	16	9	20	7	26

Table 1.8: *Shuffle π mixing the 32 branches of* WARP.

1.3.1 WARP

WARP[Ban+20] is a lightweight block cipher that has been recently presented at SAC 2020 by Banik *et al.* The main objective of the designers was to propose a cipher that could be used as a direct replacement of AES_{128} (thus with a 128-bit block and key) but that would be lighter in terms of hardware footprint. This challenge was met with flying colours as evidenced by the impressive reported number of around 800 Gate Equivalents (GEs) for a serialized circuit of WARP.

Description. The cipher follows a variant of a Type-2 *Generalized Feistel Network* (**GFN**) [Shi+07; ZMI90] using 32 branches of 4 bits each. Special care was taken to the selection of the 32-branch permutation π in order to optimize both the diffusion and the number of active S-Boxes in a differential or linear trail. The cipher iterates 41 rounds, where the final round misses π.

In detail, the 128-bit internal state is split over 32 branches of 4 bits. At the input of round r, the values of the 32 nibbles are denoted $X[i,0]$ to $X[i,31]$. They go through five elementary mappings in each (full) round, as depicted in Figure 1.7. Each nibble with an even index $X[i,2k]$ is modified by the F function, which consists of the application of a 4-bit S-box (denoted S in the following, and given in Table 1.7) followed by a round key addition. The result is then XORed with $X[i,2k+1]$, a constant is added to $X[i,1]$ and $X[i,3]$ and finally the 32 branches are shuffled by the π permutation given in Table 1.8. Since the values of the round constants have no impact on our analysis we do not include them. To have the full description of the cipher, the reader may refer to the specification [Ban+20] of WARP.

The key schedule is linear and relies on a 128-bit master key seen as the concatenation of two 64-bit keys: $K = K0||K1$. Each half is used alternatively as the round key, starting with the 16 nibbles of $K0$ that are used in the first round.

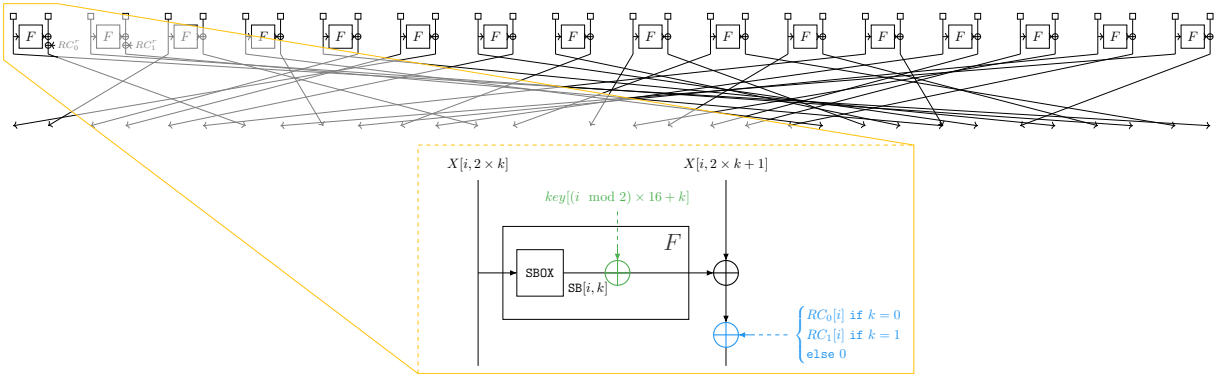


Figure 1.7: *One Round of* WARP. *The constant addition in blue (—) plays no role when searching for differential properties, and the round key addition in green (—) can be ignored when considering the single key scenario.*

Notation	Description
$X[i]$	the state at the beginning of round i. Note that $X[i]$ is also the state after applying the `ShuffleCell` function on the previous round $X[i-1]$.
$ARK[i]$	the state of round i, after applying `AddRoundKey`.
$SX[i]$	the state of round i, after applying `SubCell`.
$XSX[i]$	the state of round i, after applying `XORState`.
$RK[i]$	the subkey of round i.

Table 1.9: *The summary of the notations of the different internal states of* TWINE.

1.3.2 Twine

TWINE [Suz+13] is a 64-bit block cipher with 80 or 128-bit key. We write TWINE_{80} or TWINE_{128} to denote the key length. The global structure of TWINE is a variant of Type-2 **GFN** [Shi+07; ZMI90] with 16 4-bit sub-blocks $X[i]$, the other states are depicted in Table 1.9.

A round function of TWINE consists of a nonlinear layer F using a 4-bit S-box and a diffusion layer, which is a permutation on 16 blocks.

- `AddRoundKey`. The `AddRoundKey` is a key addition ($\oplus$) between the internal state $X[i]$ and the round key $RK[i]$.

$$ARK[i,k] = X[i,(2 \times k)] \oplus RK[i,k], \forall k \in [0;\ 7]$$

- `SubCell`. A transformation is applied on each cell of the current block using a 4-bit to 4-bit non linear S-Box (given in Table 1.10).

$$SX[i,k] = S(ARK[i,k]), \forall k \in [0;\ 7]$$

- `XORState` performs a xor between the half state which goes through the F function and

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[x]	C	0	F	A	2	B	9	5	8	3	D	7	1	E	6	4

Table 1.10: *4-bit S-Box of* TWINE

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(x)$	5	0	11	6	3	12	3	8	13	6	9	2	15	10	11	14

Table 1.11: *Shuffle π mixing the 16 branches of* TWINE.

the other half.

$$\begin{matrix} XSX[i, 2 \times k] = X[i, 2 \times k] \\ XSX[i, 2 \times k + 1] = SX[i, k] \oplus X[i, 2 \times k + 1] \end{matrix}, \forall k[0;\ 7]$$

- `ShuffleCell` is a cell permutation using the permutation table π given in Table 1.11.

$$X[i + 1, \pi[k]] = XSX[i, k], \forall k \in [0;\ 15]$$

The round function is applied 36 times for both versions, in the last round the `ShuffleCell` operation is omitted.

Since we only implement single key distinguisher search on TWINE we do not describe the `KeySchedule` algorithm of TWINE which would add unnecessary complexity. Readers that want more details about it can find it in [Suz+13].

1.3.3 LBlock-s

LBlock–s is a simplified version of LBlock [WZ11] used as sub-cipher in LAC [Zha+14]. It can be represented as a Type-2 GFS [Shi+07; ZMI90] as shown in [SN14], this representation allows to represent LBlock–s in the same structure as WARP and TWINE (see Figure 1.8).

LBlock–s is a 64-bit block cipher which accepts an 80-bit secret key. The number of iterations is either 16 or 32 depending on the position of the cipher in LAC. Instead of LBlock, which uses 10 different S-Boxes, LBlock–s only use one S-Box depicted in Table 1.12.

The round function is composed of `AddRoundKey`, an S-Box layer (using the S-Box in Table 1.12) and a permutation layer using the permutation given in Table 1.13). We do not describe the `KeySchedule` here as it is not used in this thesis. Readers who want to go deeper into the subject can refer to [Zha+14; WZ11].

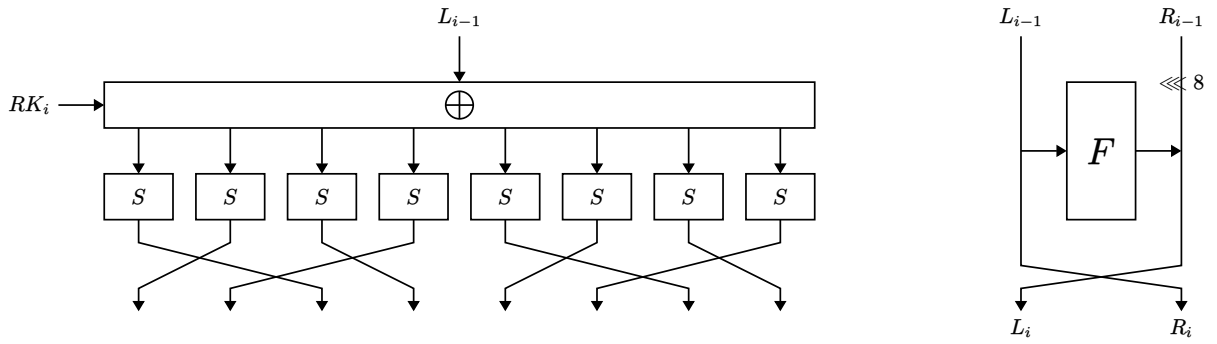


Figure 1.8: *One round of* LBlock–s. *On the left, the round function of* LBlock–s *is represented in the same structure as the one given in the LBlock specifications [WZ11; Zha+14]. On the right, the round function of* LBlock–s *is represented as a Type-2 GFS [SN14].*

1.4 Analysis and Security of symmetric ciphers

Cryptanalysis refers to the process of understanding information systems in order to expose its hidden aspects. It's used to breach cryptographic security systems and gain access to the contents of encrypted messages, even if the encryption key remains is unknown. A cryptanalysis attack can be classified according to the data it requires which will be described now.

1.4.1 Amount of information available to the attacker

An encryption scheme $C = E(K, M)$ is secure if for any ciphertext, the probability of identifying M is *negligible*. In practice, the attacker can access different levels of information, *e.g.* it may be possible to collect ciphertexts by sniffing network packages or cipher chosen plaintexts through a public API. The security of the encryption scheme can be represented in terms of games where the attacker has only some capabilities and his target is to break the encryption system:

Ciphertext-only attacks: the attacker has access only to a collection of ciphertexts, it is one of the most difficult cryptanalysis;

Known-plaintext attacks: the attacker has a set of ciphertexts to which he knows the corresponding plaintexts;

Chosen-plaintext attacks: the attacker can obtain the ciphertexts corresponding to a set of

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[x]	E	9	F	0	D	4	A	B	1	2	8	3	7	6	C	5

Table 1.12: *4-bit S-Box of* LBlock–s.

x	0	1	2	3	4	5	6	7
$\pi(x)$	2	0	3	1	6	4	7	5

Table 1.13: *Shuffle π mixing the 8 branches of* LBlock–s.

plaintexts of his own choosing;

Chosen-ciphertext attacks: the attacker can obtain the plaintexts corresponding to a set of ciphertexts of his own choosing.

The common representation of the game is to have an oracle containing either a random permutation or the encryption scheme E with a probability of 0.5 each. The ability of the oracle is fixed by the attack type. The attacker can send various queries to the oracle. After all the requests sent by the attacker, if he cannot tell, with a high probability, whether the oracle is using a random permutation or the encryption scheme E we say that the encryption scheme is *indistinguishable* under the attack conditions.

1.4.2 Distinguishers

For a given attack model, which is defined by the kind of requests that the attacker may submit to the oracle, a distinguisher is an algorithm that breaks, with a high probability, the *indistinguishability* property of ciphers. Distinguishers are usually made by looking for relations between plaintexts and ciphertexts.

1.4.3 Differential cryptanalysis

In 1990, Biham and Shamir introduced the notion of differential cryptanalysis [BS91] on the **D**ata **E**ncryption **S**tandard (**DES**) [77]. The main idea of differential cryptanalysis is to analyse the relation between input differences and output differences. For a pair of messages m_0, m_1 where m_1 is constructed by injecting a difference δ_{in} into m_0, *i.e.* $m_1 = m_0 \oplus \delta_{in}$, we compute the probability $Pr[E_K(m_0) \oplus E_K(m_1) = \delta_{out}]$ to observe a given output difference δ_{out}. For every linear operation l, the differential output can be computed in a deterministic way with $l(m_0) \oplus l(m_1) = l(m_0 \oplus m_0 \oplus \delta_{in}) = l(\delta_{in})$, but for non linear operations, such as the S-Boxes, it is not possible - in the general case - to compute deterministically the output value. In the case of iterative ciphers, a stronger attacker can also inject differences in the key, this kind of attack is called related-key differential attacks [Bih94]. In a perfect cipher, the probability to observe a given output difference should be near to 2^{-n} where n is the size of the plaintext.

Computing the exact probability to observe a given output difference for a given input difference is not possible in practice since we should enumerate 2^{2n} values. To approximate the computation of the distinguisher probability, the notion of differential characteristic (Figure 1.9) has been introduced [BS91]. The idea is to fix, not only the input differences and the output differences, but also all the differences that are propagated in the internal states of the cipher. The probability of the differential can then be computed as the sum of the different differential characteristics that have the same inputs and outputs. A common approach is to consider the probability of the best differential characteristic to be a close enough approximation of the differential probability. Usually to compute the probability of a differential characteristic, we consider that the different non-linear operators are independent, so the total probability can be computed as a product of the different non-linear transition probabilities. In Figure 1.9, the probability of the first differential

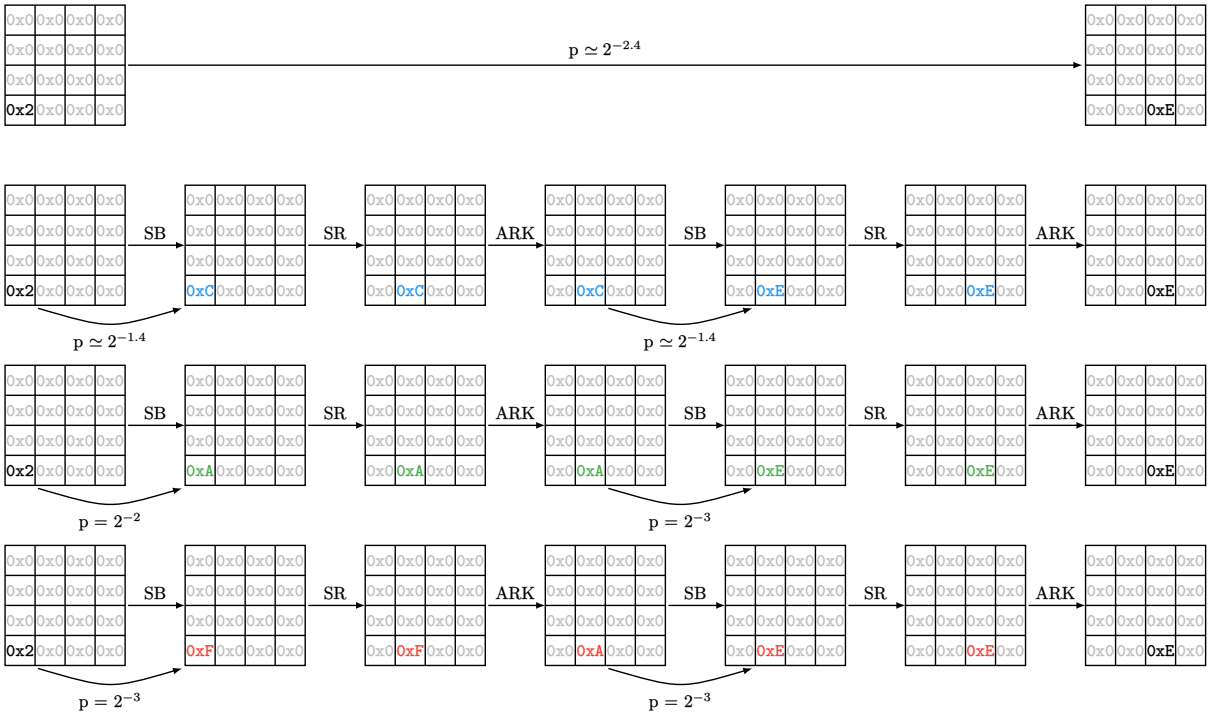


Figure 1.9: *Differential and differential characteristics on a simple toy - unsafe - cipher. The* SB *operator is a non-linear mapping working on a 4-bit sequence. Its differential transitions are available in Figure 1.10. The* SR *operator performs a shift on the rows and* ARK *performs a* XOR *between the text and the key. On the top the differential represents the* $(\delta_{in}, \delta_{out})$ *pair of differences with a probability close to* $2^{-2.4}$*. On the bottom three differential characteristics have been computed; the first one uses the transitions* $2 \rightsquigarrow C \rightsquigarrow E$ *with a probability close to* $2^{-2.8}$*, the second one uses the transitions* $2 \rightsquigarrow A \rightsquigarrow E$ *with a probability* 2^{-5} *and the last one uses the transitions* $2 \rightsquigarrow F \rightsquigarrow E$ *with a probability* 2^{-6}*.*

characteristic is equal to $2^{-2.8} = 2^{-1.4} \times 2^{-1.4}$ which are the probabilities to have the transition `0x2` $\rightsquigarrow$ `0xC` and the probability to have the transition `0xC` $\rightsquigarrow$ `0xE`, assuming that the S-Boxes are independent. Since the SR operation is linear, its transition probability is either equal to 1, when the output difference is valid or 0 when the output difference is invalid. Moreover if we compute a single key distinguisher the probability of the ARK operator is also equal to 1 since all the key differential bits are equal to 0 (differences are not allowed in the key).

To compute the substitution layer probability, we generate for each non-linear operator S a **D**ifference **D**istribution **T**able (**DDT**$_S$ [BS91]) which represents the probability to observe a pair $(\delta_{in}, \delta_{out})$ of differential variables where δ_{in} is the input of the non-linear operator and δ_{out} is its output. Figure 1.10 represents the number of transitions of a chosen `SubBytes`. For each δ_{in} it is possible to compute the probability to observe δ_{out} by counting the number of good transitions divided by the total number of possible transitions, for an n-bit to n-bit function:

$$DDT_S(\delta_{in}, \delta_{out}) = \frac{\#\{x \in \{0,1\}^n \mid S(x) \oplus S(x \oplus \delta_{in}) = \delta_{out}\}}{2^n} \tag{1.1}$$

We can observe in Figure 1.10 that we have a special case of 16 out of 16 valid transitions for the transition $0 \rightsquigarrow 0$ which comes from $\forall x, S(x) \oplus S(x \oplus 0) = 0$.

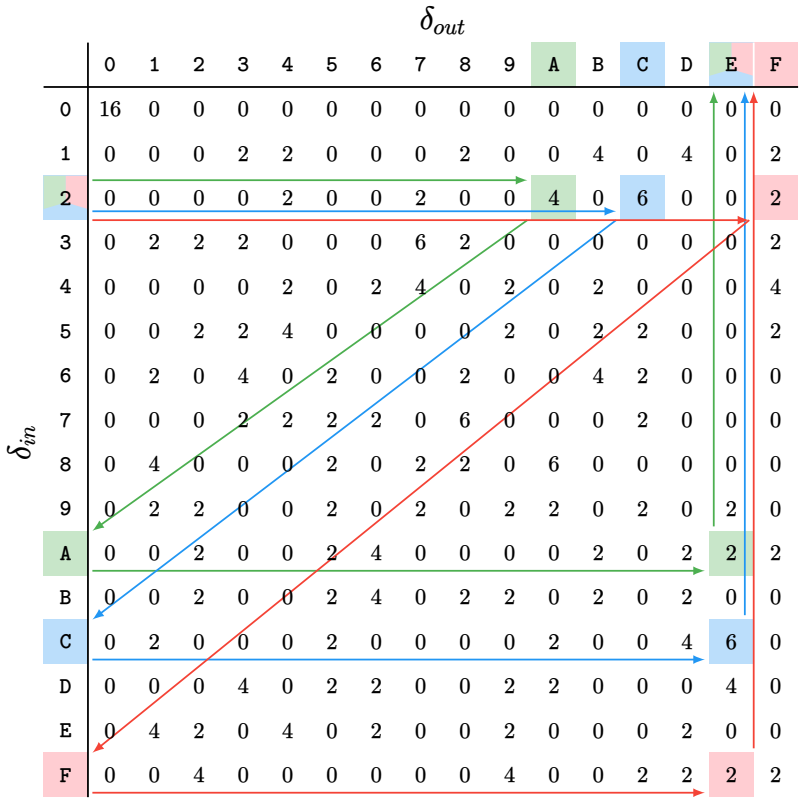


Figure 1.10: *The number of transitions from δ_{in} to δ_{out} for an unsecure random substitution used in Figure 1.9.*

Truncated Differential. To improve the scalability of the differential computation, in 1995 Knudsen introduced the notion of *truncated differential* [Knu95]. The core idea is to solve the problem in two steps. In the first step we abstract the internal state differences to represent if a word contains a difference or not, *i.e.* if the difference δ_x is equal to 0 or not. To do so, for each word variable δ_x we use an abstraction variable Δ_x with $\Delta_x = 0 \iff \delta_x = 0$ and $\Delta_x = 1 \iff \delta_x \in [1;\ 2^n - 1]$, where n is the word size. We say that transition is active when it has an input difference different from 0. We can reach the probability 1 in two cases, the first is the use of a valid linear transition, the second is the use of a non-linear transition with no input difference. Therefore, we can compute an upper bound approximation of the probability by counting the number of non-linear active transitions while ensuring that only valid transitions are used. For each non-linear function S of the cipher, we can compute the maximum transition probability for non null values with the formula:

$$\mathrm{P}_S^{\mathrm{DDT}} = \max_{(\delta_{in},\delta_{out})\in[1;\ 2^n-1]^2} DDT_S(\delta_{in}, \delta_{out})$$

Let be a cipher function E_K that uses only one kind of non-linear function NL. We denote $\#NL$ the number of non-null sequences that pass through NL. The upper bound probability (p^{UB}) of the truncated characteristic (T_{E_K}) can then be computed with:

$$p^{UB}(T_{E_K}) = (\mathrm{P}_S^{\mathrm{DDT}})^{\#NL}$$

When using different S-Boxes in the same cipher, it is possible to use the same technique with the **DDT** of each of the S-Boxes. Once the first step is computed, we obtain a truncated differential characteristic which tells us for each word that goes through a non linear operator whether it contains a difference or not. The aim of the second step is to compute, for each truncated solution, the maximum probability by fixing concrete values in the active sequences. It may be possible that truncated solutions do not lead to a valid Step-2 solution since abstractions are done during the first step. Indeed, it is possible that the truncated differential forces equalities, or inequalities, that are impossible to hold in Step-2 because the values of some sequences are incompatible in the concrete domain.

1.4.4 Boomerang attacks [Wag99]

In differential cryptanalysis the highest probability of the distinguishers considerably decreases as the number of rounds increases. It is due to the fact that each round will add a (non strict) positive number of active non linear operations. To try to mitigate this drawback Wagner introduces in 1999 the notion of boomerang attacks [Wag99]. The idea is to see the cipher E_K as a composition of two functions: $E_K(M) = (E_{1K} \circ E_{0K})(M)$, hence the number of rounds of each differential remains lower. To construct the boomerang, we choose a pair of plaintexts (P, P') that is ciphered into (C, C'). We XOR a difference γ on (C, C') to obtain a pair (D, D') and we decipher them in order to obtain (Q, Q') as depicted in Figure 1.11. Given that, if we use a differential characteristic $\alpha \rightsquigarrow \beta$ for E_{0K} and a differential characteristic $\delta \rightsquigarrow \gamma$ for E_{1K}^{-1} we have the following equations:

$$\begin{aligned} E_{0K}(Q) \oplus E_{0K}(Q') &= E_{0K}(P) \oplus E_{0K}(P) \oplus E_{0K}(P') \oplus E_{0K}(P') \oplus E_{0K}(Q) \oplus E_{0K}(Q') \\ &= E_{0K}(P) \oplus E_{1K}^{-1}(C) \oplus E_{0K}(P') \oplus E_{1K}^{-1}(C') \oplus E_{1K}^{-1}(D) \oplus E_{1K}^{-1}(D') \\ &= (E_{0K}(P) \oplus E_{0K}(P')) \oplus (E_{1K}^{-1}(C) \oplus E_{1K}^{-1}(D)) \oplus (E_{1K}^{-1}(C') \oplus E_{1K}^{-1}(D')) \\ &= \beta \oplus \gamma \oplus \gamma \\ &= \beta \end{aligned}$$

To construct a boomerang we choose the plaintext P and we generate the second plaintext P' with $P' = P \oplus \alpha$ then we cipher them to obtain C and C'. D (resp. D') can be computed from C (resp. C') with $D = C \oplus \delta$ (resp. $D' = C' \oplus \delta$). Then we decipher them in order to obtain Q and Q'. At the end we can compute the boomerang distinguisher probability with:

$$Pr[E_K^{-1}(E_K(P) \oplus \delta) \oplus E_K^{-1}(E_K(P \oplus \alpha) \oplus \delta) = \alpha] = p^2 q^2$$

Where p is the differential characteristic probability of E_{0K} and q is the differential characteristic probability of E_{1K}^{-1}.

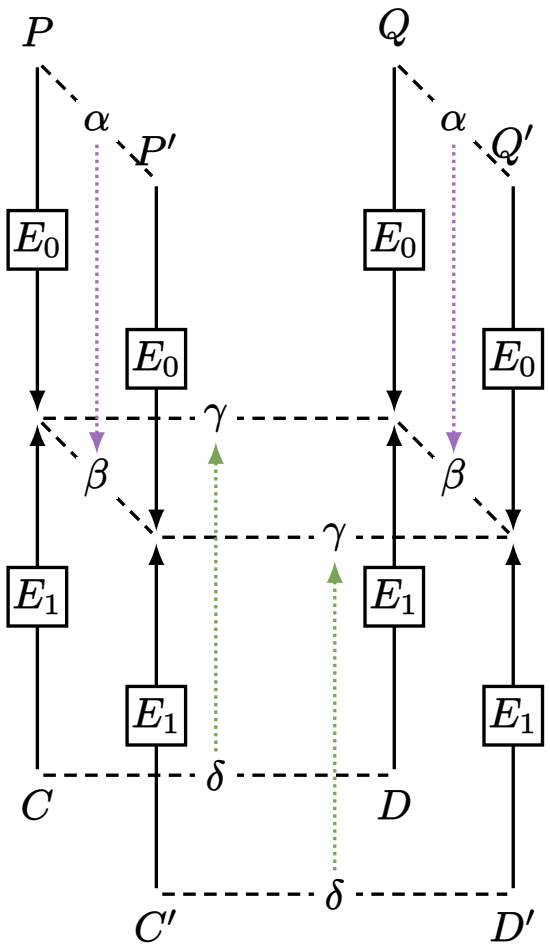


Figure 1.11: *A schematic of the basic boomerang attack [Wag99].*

1.4.5 Attacks implementation

Finding good distinguishers may be not enough to mount efficient cryptanalysis attacks. Some time non-optimal distinguishers may be used to mount more efficient attacks and the attacker has to implement smart approaches to hope to have a lower complexity than brute force attacks. It is sometimes possible to use specific data structures or redundant information to speed up the computation process or the memory usage. We will see here how to compute time complexity based on a differential distinguisher, thus deciding if they are feasible in real world situations.

Differential Attacks [BS91]

We consider a differential attack example from [Hey02], for which the attacked cipher is presented in Figure 1.12. The cipher is composed of three operations. The notations for the cipher are the following: P stands for the plaintext, C for the ciphertext. Variables X, SX and R represent the internal state of the cipher. X is the input state of the round function (or the state after the `key_mixing` function). SX is the state after the `substitution` function and R the state after the `permutation` function. The SK variables stand for round sub-keys. We consider only a single key attack, thus we do not present the `KeySchedule` to avoid introducing additional concepts. The overall algorithm of the cipher is given in Algorithm 1.2.

We note $X[i,k]$ the k^{th} bit of the sequence of the i^{th} round of X and $X[i,a..b]$ the subsequence from the a^{th} bit to the b^{th} bit of the i^{th} round of X, with $a \leq b$.

`KeyMixing`: The `KeyMixing` operation performs a XOR between the bits of the text and the SubKey of the round, this is done in lines 2, 12 and 19 of Algorithm 1.2.

`Substitution`: The `Substitution` operation is composed of 4 identical $4 \rightarrow 4$ S-Boxes (4 input bits to 4 output bits) performed in parallel (Table 1.14). This is done in lines 6 and 16 of Algorithm 1.2.

`Permutation`: The `Permutation` operation is composed of a permutation at the bit level using the permutation table π (Table 1.15). This is done in line 9 of Algorithm 1.2.

Knowing the best differential $(\delta_{in} \rightsquigarrow \delta_{out})$ over $r-1$ rounds of encryption of a cipher, if the attacker could request ciphers on plaintexts of his choice, it is possible - depending on the probability of the distinguisher - to find the key faster than exhaustive search. In our example (Figure 1.13), we consider the distinguisher to be:

$$[0,0,0,0,\ 1,0,1,1,\ 0,0,0,0,\ 0,0,0,0] \rightsquigarrow [0,0,0,0,\ 0,1,1,0,\ 0,0,0,0,\ 0,1,1,0]$$

Data: A plaintext P of 16 bits and a subkeys matrix SK of 4×16 bits.
Result: A ciphertext C of 16 bits

for $k \leftarrow 0$ **to** 15 **do**
  $X[0,k] \leftarrow P[k] \oplus SK[0,k]$
end
for $i \leftarrow 0$ **to** 2 **do**
  for $k \leftarrow 0$ **to** 3 **do**
    $SX[i, k \times 4..(k+1) \times 4 - 1] \leftarrow \texttt{S}(X[i, k \times 4..(k+1) \times 4 - 1])$
  end
  for $k \leftarrow 0$ **to** 15 **do**
    $R[i, \pi[k]] \leftarrow SX[i,k]$
  end
  for $k \leftarrow 0$ **to** 15 **do**
    $X[i+1,k] \leftarrow R[i,k] \oplus SK[i+1,k]$
  end
end
for $k \leftarrow 0$ **to** 3 **do**
  $SX[3, k \times 4..(k+1) \times 4] \leftarrow \texttt{S}(X[3, k \times 4..(k+1) \times 4 - 1])$
end
for $k \leftarrow 0$ **to** 15 **do**
  $C[k] \leftarrow SX[3,i] \oplus SK[4,i]$
end
return C

Algorithm 1.2: *The overall description of the toy example cipher of [Hey02].*

input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
output	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

Table 1.14: *The S-Box representation in hexadecimal*

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(x)$	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15

Table 1.15: *The permutation table π*

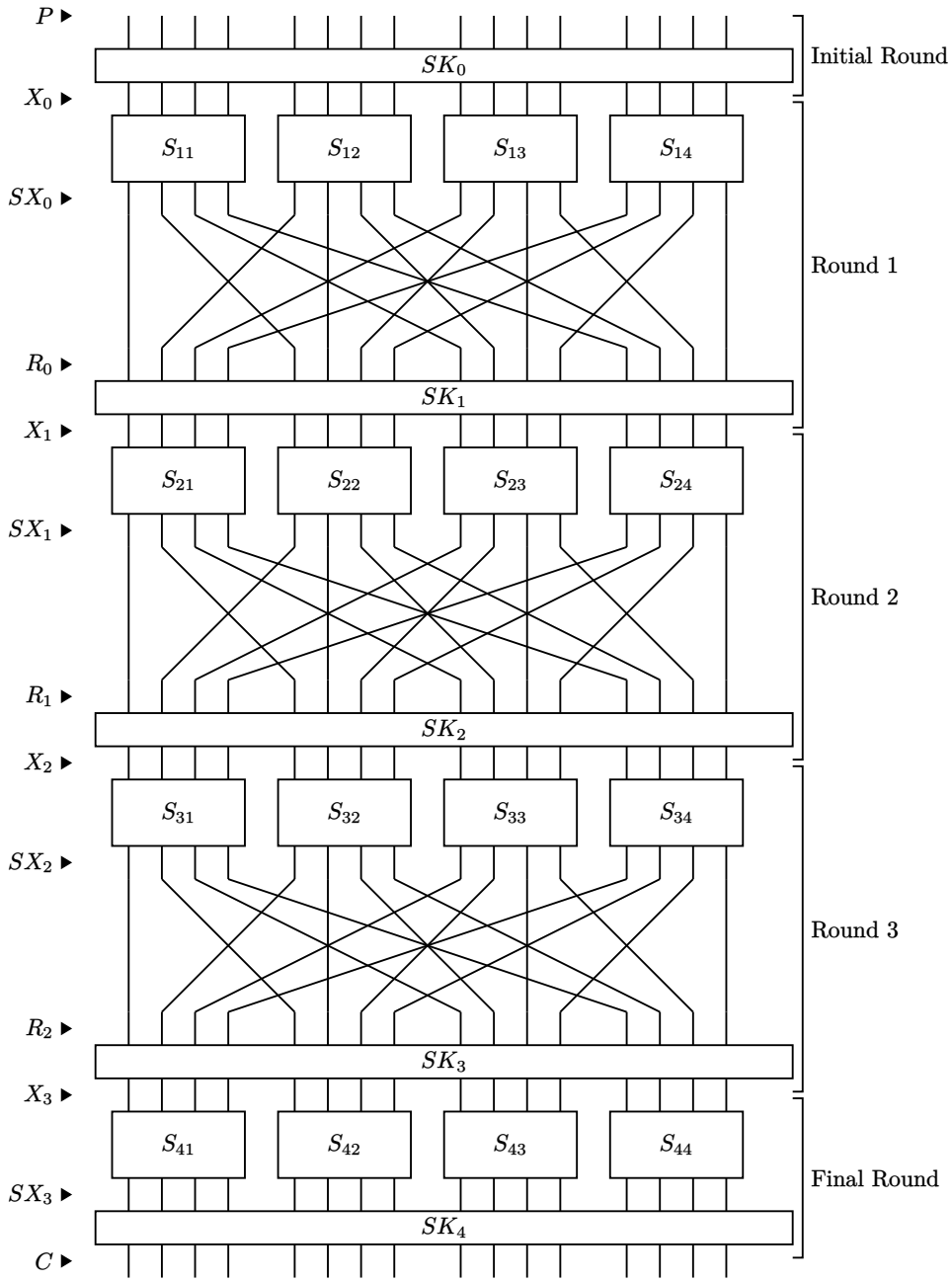


Figure 1.12: *Toy cipher example [Hey02]. The cipher is composed of three functions,* ***key_mixing****,* ***substitution*** *and* ***permutation*** *which are done in 5 rounds. The final round has less operations than the other rounds, as a final round would add no strength.*

Attack steps.

1. The attacker asks the oracle the encryption of two plaintexts m and m' for which the difference is δ_{in}. The difference of $f_k(m)$ and $f_k(m')$ is noted δ_C, where $f_k(m)$ is the encryption of a plaintext m using a cipher function f with a secret key k.

As δ_{out} has a higher probability to appear than other output differences, the attacker should find δ_{out} with a high probability if he computes the differences before the last round encryption. We call the partial key, the bits of the last subkey which are XORed with an active S-Box during the last round. In our example, the partial key is composed of $SK_5[4..7]$ and $SK_5[12..15]$.

2. The attacker enumerates all the possible values for the partial key. Each time the decryption of the last round of m and m' leads to a difference of δ_{out} the attacker increments the score of the partial key.
3. The attacker restarts phases 1 and 2 until he has enough information, the number of pairs of required plaintexts depends on the distinguisher probability and the construction of the S-Boxes used in the cipher.
4. For each partial key, sorted in descending order, the attacker enumerates all the possible values for the other bits - which are not in the partial key - and tries to cipher a fixed message for which he knows the ciphertext. If the ciphertext used with the generated key matches the ciphertext given by the oracle, then the attacker has found the key. If the attacker did not find the valid ciphertext after enumerating all the possible values, he tries the next partial key.

1.5 Discussion

In this chapter, we have seen the basics of differential cryptanalysis. As we have seen, computing differential characteristics in a naïve implementation is not possible in practice since the exhaustive search will have a complexity of 2^{2n}. To counter this drawback dedicated approaches [FJP13; BN10] have been studied to speed up the solving time. This type of approaches is generally specific for a given attack applied to a given cipher. In 2011, Mouha *et al.* use the Integer Linear Programming (**ILP**) approach to compute maximal linear and differential characteristics [Mou+11]. The novel idea is to use generic solvers, such as **ILP**, Boolean satisfiabiliy (**SAT**) or Constraint Programming (**CP**), to solve cryptanalysis problems. Since then, several publications have been made using **SAT** [MP13; KLT15; SWW18], **ILP** [Sun+14; Abd+17] and **CP** [GMS16; Sun+17; Gér+18] solvers to solve linear and differential cryptanalysis problems.

In the next chapter, we will introduce Constraint Programming, Boolean satisfiability and Integer Linear Programming.

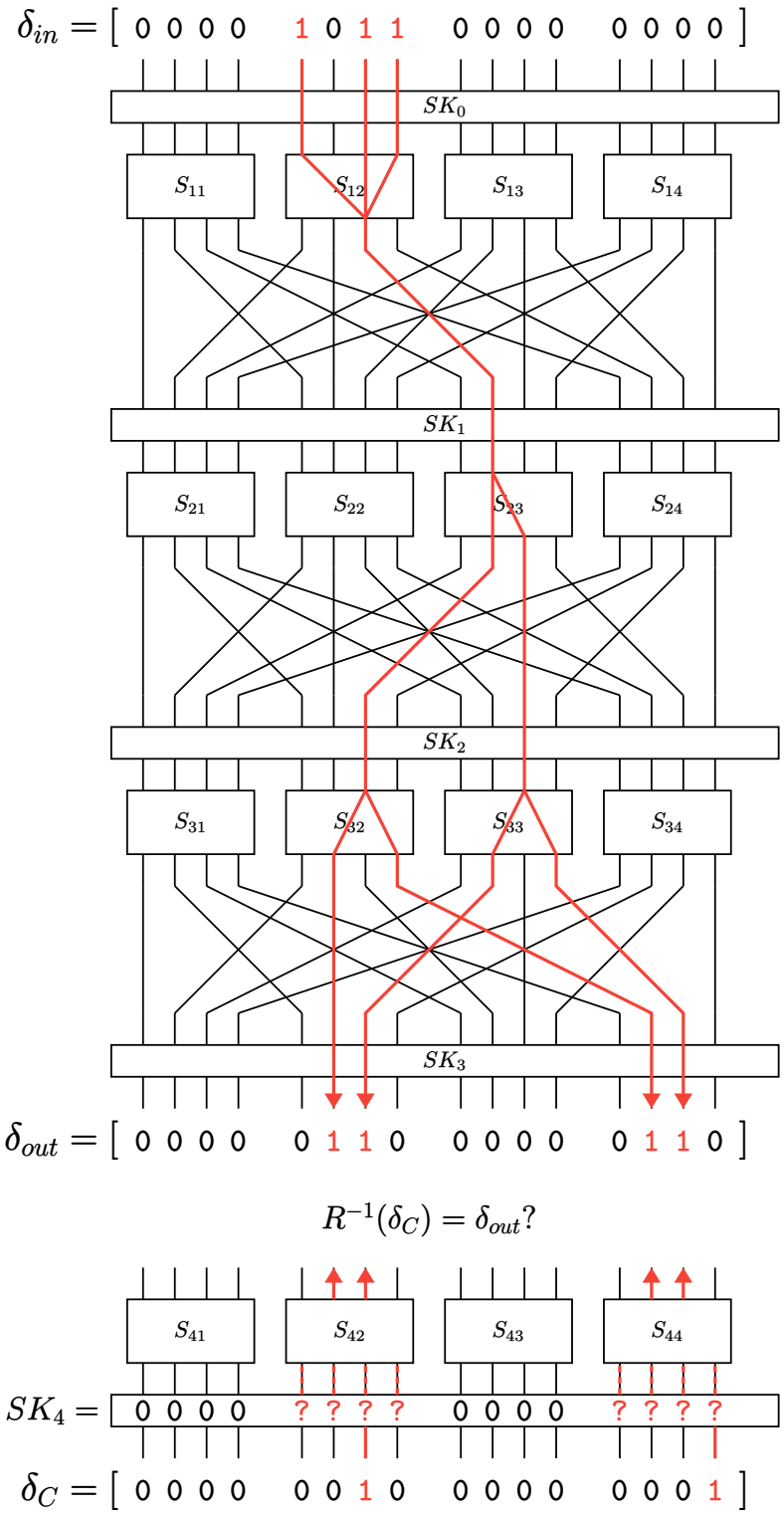


Figure 1.13: *Making a differential attack. Knowing a differential characteristic* $(\delta_{in} \rightsquigarrow \delta_{out})$*, the attacker chooses two messages* (m, m') *such that* $m \oplus m' = \delta_{in}$*. The output difference is* δ_C*. The attacker can then go back to the last round to try to evaluate the value of the last subkey used by checking if it is possible to meet the* δ_{out} *difference at the round* $r - 1$*.*

Chapter

2 Constraint Programming

> *Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.*
>
> — Eugene C. Freuder

Contents

Combinatorial problems may be solved by using declarative approaches: the problem to be solved is modelled by means of variables and constraints, and the model is solved by a generic solver. There are three main kinds of declarative approaches:

- Boolean satisfiability (**SAT**), where models are logical formulae defined over Boolean variables;
- Integer Linear Programming (**ILP**), where models are linear inequalities over Integer variables;
- Constraint Programming (**CP**), where a wide range of variables and constraints are available, the only limitation being their implementation in the considered **CP** solver.

In this chapter we will see what is a Constraint Satisfaction Problem (**CSP**) and what is a Constrained Optimization Problem (**COP**). We will then introduce the notion of computation complexity and what are the mechanisms used to solve **CP** problems.

2.1 Modelling by means of variables and constraints

In this section we see what are Constraint Satisfaction and Constrained Optimization problems. We also introduce the notion of solution for this kind of problems.

2.1.1 Constraint Satisfaction Problems

A **CSP** is a mathematical problem defined as below:

Definition 2.1: Constraint Satisfaction Problem (CSP) [RVW06]

A constraint satisfaction problem (**CSP**) is defined by a triplet (X, D, C) where :

- $X = \{x_1, ..., x_n\}$ is the set of variables of the problem;
- $D = \{D_1, ..., D_n\}$ is the set of domains of the variables, *i.e.* D_k is the set of values that may be assigned to x_k;
- $C = \{C_1, ..., C_m\}$ is a set of constraints. A constraint $C_i = (X_i, R_i)$ is defined by a tuple $X_i = (x_{i_1}, ..., x_{i_k})$ of variables (called the scope) and a relation $R_i \subseteq D_{i_1} \times ... \times D_{i_k}$ which defines the set of values allowed simultaneously for the variables of X_i. This relation may be defined either in extension, by listing all the allowed tuples (or, conversely, all the forbidden tuples), or in intention by using mathematical operators.

In this chapter, we only consider finite domain **CSP**s, such that variable domains only contain a finite set of values.

Example 2.1

Let us take a toy example to represent how constraint programming works. We want to solve a 3×3 simple sudoku. The rules are easy, the board game is defined by a 3×3 matrix of cells, each cell must take a value in $[1;\ 3]$. Moreover for all cells in the same row and for all cells in the same columns, the cells must have a different values. We can represent the game by a matrix of 9 variables, each variable representing a cell:

$$\begin{matrix} x_0 & x_1 & x_2 \\ x_3 & x_4 & x_5 \\ x_6 & x_7 & x_8 \end{matrix}$$

In our case, the CSP will be defined as (X, D, C), with:

$$X = \{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\},$$
$$D = \{[1;\ 3], [1;\ 3], [1;\ 3], [1;\ 3], [1;\ 3], [1;\ 3], [1;\ 3], [1;\ 3], [1;\ 3]\},$$
$$C = \{(X_0, R_0), (X_1, R_1), (X_2, R_2), (X_3, R_3), (X_4, R_4), (X_5, R_5)\}$$
where $X_0 = \{x_0, x_1, x_2\}, X_1 = \{x_3, x_4, x_5\}, X_2 = \{x_6, x_7, x_8\},$
where $X_3 = \{x_0, x_3, x_6\}, X_4 = \{x_1, x_4, x_7\}, X_5 = \{x_2, x_5, x_8\}$
and $R_0 = R_1 = ... = R_5 = \{(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)\}$

The pair (X_0, R_0) (resp. (X_1, R_1) and (X_2, R_2)) represents the all different constraint on the first (resp. second and third) row. The pair $(X3, R_3)$ (resp. (X_4, R_4) and (X_5, R_5)) represents the constraint all different on the first (resp. second and third) column.
The constraint (X_0, R_0) can also be represented in intention by using the $\neq$ operator:

$$X_0 = (x_0, x_1, x_2), R_0 = (x_0 \neq x_1) \wedge (x_0 \neq x_2) \wedge (x_1 \neq x_2)$$

To indicate what is a valid solution, we need to introduce first the notion of instantiation.

Definition 2.2: Instantiation [RVW06]

Given a CSP $= (X, D, C)$, and a tuple of variables $Y = (x_1, ..., x_k) \subseteq X$

- An instantiation I on Y is an assignment of values $(v_1, ..., v_k)$ to the variables $x_1, ..., x_k$. I can be denoted by $((x_i, v_i), ..., (x_k, v_k))$ where (x_i, v_i) denotes the value v_i for x_i. A projection of I on the tuple of variable Y, denoted $proj_Y(I)$ is the tuple of values defined by: $proj_Y(I) = (v_1, ..., v_k)$.
- An instantiation I on Y is valid if for all $(x_i, v_i) \in I, v_i \in D_i$.
- When $Y = X$ we say the instantiation is total, otherwise, the instantiation is said to be partial.
- A total instantiation I is consistent if $\forall c_i = (X_i, R_i) \in C$, the projection of I on X_i belongs to R_i.

A **solution** is a total and consistent instantiation. The set of solutions of the CSP is denoted by $sol(X, D, C)$.

Once the **CSP** has been formally defined, solvers may have three aims:

1. to answer if the **CSP** has a solution or not, which is called a decision problem [Pap94];
2. to return a solution for the given **CSP**, if it has a solution;
3. to enumerate all the solutions for the given **CSP**.

Below, we show how to check whether an instantiation is a solution or not.

Example 2.2

We want to check if the following instantiation is a solution:

$$\begin{pmatrix}(x_0,1),(x_1,2),(x_2,3),\\(x_3,2),(x_4,3),(x_5,1),\\(x_6,3),(x_7,1),(x_8,2)\end{pmatrix}$$

The instantiation can be represented by the n-tuple $T = (1,2,3,2,3,1,3,1,2)$. To check if the n-tuple is valid, for all constraints C_j in the CSP, we need to check if the projection of the n-tuple onto X_j belongs to the relation R_j.

$$\begin{aligned}
proj_{X_0}(T) &= (1,2,3) \text{ and } (1,2,3) \in R_0,\\
proj_{X_1}(T) &= (2,3,1) \text{ and } (2,3,1) \in R_1,\\
proj_{X_2}(T) &= (3,1,2) \text{ and } (3,1,2) \in R_2,\\
proj_{X_3}(T) &= (1,2,3) \text{ and } (1,2,3) \in R_3,\\
proj_{X_4}(T) &= (2,3,1) \text{ and } (2,3,1) \in R_4,\\
proj_{X_5}(T) &= (3,1,2) \text{ and } (3,1,2) \in R_5
\end{aligned}$$

In our case, all the projections $proj_{X_j}$ belong to their respective relation R_j, thus the instantiation is a solution of the problem.

Special case of SAT. **SAT** is a special case of **CSP** that only accepts Boolean variables and Boolean formulae. Those restrictions allow to use dedicated algorithms such as the DPLL [DP60; DLL62] and the CDCL [SS96; SS99] algorithms which provide very good performances. In differential cryptanalysis, **SAT** is efficient to compute truncated differential characteristics since they contain a lot of Boolean variables and formulae. The non-Boolean part of the model can be translated into a Boolean one by introducing temporary variables and transforming constraints into Boolean formulae. Even if this can generate an exponential number of variables, it scales pretty well in practice.

2.1.2 Constrained Optimization Problems

A variant of **CSP** is Constrained Optimization Problem. Now the goal is not to say whether a solution exists or not, but to find the best solution with respect to a given objective function.

Definition 2.3: Constrained Optimization Problem (COP) [RVW06]

A Constrained Optimization Problem (**COP**) is defined by a quadruplet (X, D, C, f) where (X, D, C) is a CSP and $f : X \to \mathbb{R}$ is a function which is to be maximized or minimized.

A **solution** of a COP $= (X, D, C, f)$ is a solution of the CSP $= (X, D, C)$, and a solution is optimal if it minimizes or maximizes (in regards to the objective) $f(X)$.

Example 2.3

We take the same example as in Example 2.1 and we add the objective function that consists in minimizing the values of the top-left bottom-right diagonal variables, such that $COP = (X, D, C, f)$ where (X, D, C) is the same tuple as in Example 2.1 and

$$f(X) = \sum_{i=0}^{2} x_{4i}$$

In such case, the instantiation given in Example 2.2 is still a solution but is not an optimal solution of the COP since $1 + 3 + 2$ is not the optimal value.
An optimal solution of the COP is:

$$\begin{pmatrix} (x_0, 1), (x_1, 2), (x_2, 3), \\ (x_3, 3), (x_4, 1), (x_5, 2), \\ (x_6, 2), (x_7, 3), (x_8, 1) \end{pmatrix}$$

Special case of ILP. **ILP** is a special case of **COP** where the variables are Integer variables, constraints are restricted to linear inequations and the objective function f is linear. As with **SAT**, the type of variables and the restriction of constraints allow the use of dedicated algorithms such as Cutting-plane [Gom58], the Branch and bound [LD10; LW66] or the Branch and cut [PR91] methods.

2.1.3 Computational complexity

While the modelling part is interested in how to represent problems the solving part focuses on the solving techniques for solving these problems. In computer science it can be interesting to study the computational complexity to know how difficult a problem is to solve a-priori.

Some particular kinds of **CSPs**, such as the linear assignment problem [FT87], can be solved in polynomial time, *i.e.* there exists an algorithm whose computation time is polynomial with respect to the instance size, the size being the number of bits necessary to represent the input data. **CSPs** with finite domains are $\mathcal{NP}$-complete in the general case, *i.e*, they cannot be solved in polynomial time unless $\mathcal{P} = \mathcal{NP}$, but it is possible the verify that a solution is valid in polynomial time.

In the modeling phase, the only limit to the expressiveness of the model is the list of constraints that are implemented in the solvers. In some cases, it may be useful to create new constraints that will be able to solve or to model a particular problem more efficiently, *e.g.* the `alldifferent` constraint [Rég94]. When creating a new constraint, it is interesting to analyze the complexity of checking the satisfiability of this constraint. For example, we can decide in polynomial time if there exists a valid instantiation for an `alldifferent` constraint [Rég94], whereas this problem is $\mathcal{NP}$-complete for the `setSum` global constraint, which ensures that the sum of the values assigned to a set of variable equals a given sum [Bes+04].

Data: A CSP (X, D, C) and a partial and consistent instantiation I
Result: A solution (if I may be extended to a solution of (X, D, C)), or $\emptyset$ (overwise)

```
1  fn branch-and-propagate(X, D, C, I) {
2      if len(I) = len(X) then // I is complete
3          return I
4      else
5          select an unassigned variable x_i ∈ X
6          for v_i ∈ D(x_i) do
7              (X, D', C', I') ← propagate(X, D, C, I ∪ (x_i, v_i))
8              sol ← branch-and-propagate(X, D', C', I')
9              if sol ≠ ∅ then
10                 return sol
11             end
12         end
13         return ∅
14     end
15 }
```

Algorithm 2.1: *Branch and propagate algorithm. The ordering heuristics are used at lines* 5 *and* 6 *when the algorithm needs to select the next variable and the next value to test. The constraints propagation phase is at line* 7*.*

To prove that a new problem A is $\mathcal{NP}$-complete, we must first prove that it belongs to the $\mathcal{NP}$ class, which contains all problems that may be solved in polynomial time by a non deterministic Turing machine [Pap94]. This basically involves showing that there exists a polynomial time algorithm to check whether a given certificate is a solution or not. Then, we must prove that A is at least as hard as any other $\mathcal{NP}$-complete problem, and this is done by defining a polynomial algorithm that transforms any instance I of a know $\mathcal{NP}$-complete problem to an instance I' of A in such way that a solution of I' may be used to build the solution of I in polynomial time.

2.2 Solving CSPs and COPs

As mentioned earlier, **SAT** and **ILP** solvers use dedicated algorithms, which will not be presented here. In this section, we present algorithms for solving **CSP**s and **COP**s in Constraint Programming.

2.2.1 Branch & Propagate

CP solvers work with algorithms that contain two phases, as described in Algorithm 2.1: branching and propagation. The first phase contains heuristic algorithms to select the next variables and values to explore (*l. 5* and *6* of Algorithm 2.1). The propagation phase (*l. 7* of Algorithm 2.1) includes algorithms to reduce the domains of the variables in regards of the constraints they have and the assignments that have already been made. Such algorithms are called propagators and perform a tightening of the **CSP**.

2.2.2 Constraint propagation

Constraint propagators are algorithms that are designed to filter variable domains in order to ensure some local consistency (or detect inconsistency), they are the software component that links the model expressiveness and solution computation. The most known local consistency is Generalized Arc Consistency (**GAC**) [Rég94].

Given a constraint $c_i = (X_i, R_i)$ such that $X_i = (x_{i1}, ..., x_{ir})$, and a tuple $t \in R_i$ such that $t = (v_{i1}, ...v_{ir})$, let $t_{\downarrow x_{ij}}$ denote the value associated with x_{ij} in t, *i.e*, $t_{\downarrow x_{ij}} = v_{ij}$.

A constraint $c_i = (X_i, R_i)$ is **GAC** if and only if for every variable $x_j \in X_i$ and every value $v_j \in D_j$, the couple (x_j, v_j) has a support in R_i, *i.e.*, there exists a tuple $t \in R_i$ such that $t_{\downarrow x_j} = v_j$ and for every other variable $x_k \in X_i \setminus \{x_j\}$, $t_{\downarrow x_k} \in D_k$.

A better filtering allows to filter variable domains earlier in the search, therefore pruning the search tree more efficiently. Unfortunately, in the general case, better filtering implies a higher computation time. Therefore, it is necessary to find a compromise between filtering quality and solving time in order to improve the global solving time.

A common approach to improve the filtering process is to create constraints that have an higher level of knowledge. This can be done by using global constraints. For example, in Example 2.1 we modelled the first row by:

$$X_0 = (x_0, x_1, x_2)$$
$$R_0 = (x_0 \neq x_1) \wedge (x_1 \neq x_2) \wedge (x_0 \neq x_1)$$

but the model can also be represented by the global constraint `alldifferent`.

$$X_0 = (x_0, x_1, x_2)$$
$$R_0 = \texttt{alldifferent}(x_0, x_1, x_2)$$

Ensuring the **GAC** of this global constraint is stronger than ensuring the **GAC** of each binary different constraint separately, and this may be done in polynomial time by using dedicated propagator [Rég94]. In such case, while the computational complexity remains low, the pruning quality is increased, which leads to a considerable saving in computing time.

2.2.3 Branch & Bound

Constrained Optimization Problems do not only require finding a solution, but require finding an optimal solution with respect to a given objective function. A common technique used is the branch and bound technique. The problem solving is done in the same way as for the branch and propagate algorithm but adds a cutting technique allowing to prune the search tree earlier and thus limit the number of states to visit.

To explain the flow of the algorithm, we consider that the objective function f is to minimize. The first step of the search consists in finding a solution s_0. If there is no solution, the search stops. The first solution will be an upper bound for the optimal solution s^*, *i.e.* $f(s^*) \leq f(s_0)$. After finding this solution, the algorithm continues the search. At each decision node, thanks to a bound function, the algorithm evaluates if the current partial solution still allows to find a better solution than the best solution found until now. Once a new solution is found, the upper bound is updated and the search continues. This is done until it is no longer possible to find a new solution. At the end of the search, the last solution is then the best possible solution.

2.3 Discussion

In this chapter we have introduced the Constraint Satisfaction and Constrained Optimization Problems. We have also detailed why they are generally hard to solve and spoke about solving techniques on generic problems. Two special cases of **CSPs** and **COPs** can also be defined, *i.e.* **SAT** and **ILP**. In **SAT** problems, only Boolean variables and Boolean formulae are allowed, while in **ILP** only Integer variables with inequalities are allowed. Although the subsets are less expressive than Constraint Programming, it is possible to move from one paradigm to another by changing the model of the problem. Some tools are available to convert models into different representations, *e.g.* `picat` [ZK16] is able to transform a flatzinc **CP** model into **SAT** or **ILP** problems and then solve them by using different backend solvers, lingeling [Bie] for **SAT** and Gurobi [Gur22] for **ILP**. Since **SAT** and **ILP** have specific type of variables and constraints, the solvers include dedicated solving approaches.

In this thesis, the main contribution to constraint programming was the creation of a new global constraint called *AbstractXOR*. This work is described in Chapter 5. In other chapters, **SAT** has been used to solve some models particularly adapted to the Boolean formulation while **CP** has been used to model differential characteristics which require to model Difference Distribution Tables (**DDT**s) efficiently.

Chapter

3 Existing declarative models for solving differential cryptanalysis problems

Contents

In this chapter we present the two models from which we started our work. The first model, by Gérault *et al.*, is a model to compute related-key differential attacks on AES. It uses a two-stage solving with a Step-1 model for truncated differential characteristics and a Step-2 model for differential characteristics. The second model, by Delaune *et al.*, is used to compute boomerang attacks on Skinny. It also works with the two-step solving process.

3.1 Gérault *et al.*'s model [Gér+20]

The model of Gérault *et al.* [Gér+20] aims at computing the optimal differential characteristics for AES. The solution of the problem follows the two steps of [Knu95], namely:

- the computation of truncated differential characteristics (Step-1),
- the computation of the optimal differential characteristics (Step-2) from the truncated ones computed in Step-1.

To understand how the Step-1 model is created, it is first important to understand how to transform the specification of a cipher into a model for computing differential characteristics.

3.1.1 Build a **CP** model from a cipher specifications

In this section, we will see how to transform the specifications of a cipher, AES_{128} in our case, into a **CP** model allowing to compute differential characteristics.

As a reminder, constraint programming makes it possible to define a set of variables and the relationships that rule them. These relationships can be represented implicitly or explicitly. In the case of the operator XOR we can represent the equation:

$$a \oplus b \oplus c = 0$$

- implicitly with: $a \oplus b \oplus c = 0$, if the solver has the constraint modelling the XOR,
- explicitly with: $(a, b, c) \in T_{\texttt{XOR}}$, where $T_{\texttt{XOR}}$ is the set of tuples of values that can be taken by a, b and c simultaneously.

Even if Step-1 is computed before Step-2, it is simpler to start with the modelling of Step-2 before presenting the modelling of Step-1.

Variables introduction

AES is an iterative symmetrical block cipher working on matrices of 4×4 bytes on N_r rounds. In the case of a differential attack, we want to know how an input difference δ_{in} is transformed into an output difference δ_{out}. To know this propagation, it is necessary to track the successive values of the differences throughout the encryption process, *i.e.* to know the differences between each byte of the internal state. These differential bytes δ can be represented by an integer in $[0;\ 255]$. Knowing that AES (described in Section 1.2.1) contains the states X, SX, Y, Z, WK and SWK (see Table 1.3 and Algorithm 1.1), we introduce the variables $\delta_X, \delta_{SX}, \delta_Y, \delta_Z, \delta_{WK}$ and δ_{SWK} to represent those differences, *e.g.*:

$$\delta_X[i, j, k] = X_0[i, j, k] \oplus X_1[i, j, k]$$
(considering the computation of a differential between X_0 and X_1).

We also introduce the variables δ_{RK} and δ_{SRK} which correspond to the bytes of the various round-keys of the `KeySchedule`. The variables δ_{RK} and δ_{SRK} are strictly equivalent to the δ_{WK} and δ_{SWK} variables and can be linked to them with the formulae:

$$\forall i \in [0;\ N_r], \forall j \in [0;\ 3], \forall k \in [0;\ 3],$$
$$\delta_{RK}[i, j, k] = \delta_{WK}[j, i \times 4 + k]$$

$$\forall i \in [0;\ N_r], \forall j \in [0;\ 3], \forall k \in [0;\ 3],$$
$$\delta_{SRK}[i, j, k] = \delta_{SWK}[j, i \times 4 + k]$$

Plaintext and ciphertext differences are not modelled as they are condition free and can be recovered from the other variables.

In summary, we have:

- δ_X, a 3-dimensional matrix of $N_r \times 4 \times 4$ integer variables,
- δ_{SX}, a 3-dimensional matrix of $N_r \times 4 \times 4$ integer variables,
- δ_Y, a 3-dimensional matrix of $N_r \times 4 \times 4$ integer variables,
- δ_Z, a 3-dimensional matrix of $N_r - 1 \times 4 \times 4$ integer variables, knowing that `MixColumns` is not omitted in the last round,
- δ_{WK}, a 2-dimensional matrix of $4 \times 4(N_r + 1)$ integer variables,
- δ_{SWK}, a 2-dimensional matrix of $4 \times 4(N_r + 1)$ integer variables,
- δ_{RK}, a 3-dimensional matrix of $N_r + 1 \times 4 \times 4$ integer variables,
- δ_{SRK}, a 3-dimensional matrix of $N_r + 1 \times 4 \times 4$ integer variables.

Note: knowing that `KeySchedule` passes only the last columns of the round keys in S-Boxes, it is possible to model δ_{SRK} only for the third column.

Round constants. In the case of differential attacks, constants that are added (using the XOR operator) to the state or to the key can be ignored, since $C \oplus C = 0$. In the case of AES the round constants used in the `KeySchedule` are ignored.

Once the variables have been set we also need to model the cipher operations.

Operations modelling

`SubBytes`.

The `SubBytes` operation passes all the bytes of X through the Rijndael S-Box. Since this operation is non-linear, it is necessary to model the transition probability between the δ_X state and the δ_{SX} state. To do this, we introduce additional variables $p[i,j,k]$ where $p[i,j,k] = -log_2(Pr[\delta_X[i,j,k] \rightsquigarrow \delta_{SX}[i,j,k]])$ (see Equation 1.1) for each pair of $(\delta_X[i,j,k], \delta_{SX}[i,j,k])$. This relationship is represented using a table constraint:

$$\forall i \in [0;\ N_r[, \forall j \in [0;\ 3], \forall k \in [0;\ 3],$$
$$(\delta_X[i,j,k], \delta_{SX}[i,j,k], p[i,j,k]) \in T_{DDT}$$

where T_{DDT} is the set of triples $(\delta_{in}, \delta_{out}, -log_2(DDT(\delta_{in}, \delta_{out})))$ such that $\delta_{in} \in [0;\ 255]$, $\delta_{out} \in [0;\ 255]$ and $DDT(\delta_{in}, \delta_{out}) > 0$. Since we only want to have possible transitions, T_{DDT} does not include transitions of probability 0. In the case of AES, the possible values for $p[i,j,k]$ are $\{0, 6, 7\}$ (which represent the probabilities: $2^0, 2^{-6}$ and 2^{-7}).

The same reasoning can be carried out for the variables δ_{WK} and δ_{SWK}, with the difference that only some columns of the `KeySchedule` pass through S-Boxes. The values associated with δ_{WK} and δ_{SWK} are named p_{WK}. p_{WK} can be translated to p_{RK} by using the same formula as for δ_{WK} and δ_{RK}.

`ShiftRow`.

`ShiftRow` is a leftward byte shift. It can be represented using the equality constraint:

$$\begin{aligned}&\forall i \in [0;\ N_r[, \forall j \in [0;\ 3], \forall k \in [0;\ 3],\\ &\quad \delta_Y[i,j,k] = \delta_{SX}[i,j,(j+k) \mod 4]\end{aligned}$$

`MixColumns`.

`MixColumns` is the multiplication of Y by the matrix M. This multiplication requires to model the XOR operator as well as the Galois field $GF(2^8)$ multiplication, noted $\otimes$, and can be represented in several ways.

One of the simplest methods is to calculate the values associated with δ_Y for the different values of M, *i.e.* $2\delta_Y$ (where $2\delta_Y = 2 \otimes \delta_Y$) and $3\delta_Y$ (where $3\delta_Y = 3 \otimes \delta_Y$) using table constraints, then combine them to calculate the values of δ_Z:

$$\begin{aligned}&\forall i \in [0;\ N_r - 2], \forall j \in [0;\ 3], \forall k \in [0;\ 3],\\ &\quad (\delta_Y, 2\delta_Y) \in T_{gmul_2},\\ &\qquad \text{where } T_{gmul_2} = \{(\delta_Y, 2\delta_Y) \in [0;\ 255]^2 \mid 2\delta_Y = 2 \otimes \delta_Y\}\\ &\quad (\delta_Y, 3\delta_Y) \in T_{gmul_3},\\ &\qquad \text{where } T_{gmul_3} = \{(\delta_Y, 3\delta_Y) \in [0;\ 255]^2 \mid 3\delta_Y = 3 \otimes \delta_Y\}\end{aligned}$$

$$\begin{aligned}&\forall i \in [0;\ N_r - 2], \forall k \in [0;\ 3],\\ &\quad (\delta_Z[i,0,k], 2\delta_Y[i,0,k], 3\delta_Y[i,1,k], \delta_Y[i,2,k], \delta_Y[i,3,k]) \in T_{XOR5}\\ &\quad (\delta_Z[i,1,k], \delta_Y[i,0,k], 2\delta_Y[i,1,k], 3\delta_Y[i,2,k], \delta_Y[i,3,k]) \in T_{XOR5}\\ &\quad (\delta_Z[i,2,k], \delta_Y[i,0,k], \delta_Y[i,1,k], 2\delta_Y[i,2,k], 3\delta_Y[i,3,k]) \in T_{XOR5}\\ &\quad (\delta_Z[i,3,k], 3\delta_Y[i,0,k], \delta_Y[i,1,k], \delta_Y[i,2,k], 2\delta_Y[i,3,k]) \in T_{XOR5}\end{aligned}$$

where T_{XOR5} is the set of 5-ary tuples (a,b,c,d,e) where $a \oplus b \oplus c \oplus d \oplus e = 0$.

In practice, T_{XOR5} is too large to be used as is. Therefore, `MixColumns` is not implemented in this way but the underlying idea remains the same. For example, it is possible to introduce temporary variables to decompose XOR equations into smallest ones.

`AddRoundKey`.

`AddRoundKey` performs a XOR between the state and the round-key. It can also be represented by a constraint table.

$$\begin{aligned}&\forall i \in [0;\ N_r[, \forall j \in [0;\ 3], \forall k \in [0;\ 3],\\ &\quad (\delta_X[i+1,j,k], \delta_{RK}[i+1,j,k], \delta_Z[i,j,k]) \in T_{XOR3}\end{aligned}$$

T_{XOR3} has the same meaning as T_{XOR5} for 3-ary equations.

`KeySchedule`.

The `KeySchedule` contains both XOR and S-Boxes, it can be modelled with constraint tables for the XOR and the technique presented in fro `SubBytes`.

Objective function

Under the assumption that the S-Boxes are independent, we can compute the differential characteristic probability by summing the p and p_{RK} values (as they are in $\log_2$). The objective function is so to minimize obj (which maximizes the real probability) with:

$$obj = \sum_{i=0}^{N_r-1}\sum_{j=0}^{3}\sum_{k=0}^{3} p[i,j,k] + \sum_{i=0}^{N_r-1}\sum_{j=0}^{3} p_{RK}[i,j,3]$$

Note we only sum on $p_{RK}[i,j,3]$ since the last column of $RK[i]$ is the only column in the `KeySchedule` that passes through an S-Boxes for AES_{128}.

3.1.2 Computation of truncated characteristics (Step-1)

The previous Step-2 model allows to compute optimal differential characteristics for AES_{128}. However, it is not sufficient to carry out the search. Indeed, the search is too complex to be solved in a reasonable amount of time by using only this model. A common technique is to compute previously truncated differential characteristics and use them to fix differential byte values. In truncated differential characteristics each differential byte δ_A is abstracted by a differential Boolean Δ_A with $\Delta_A = 0 \iff \delta_A = 0$ and $\Delta_A = 1 \iff \delta_A \in [1;\ 255[$. In this section, we introduce the model of Minier *et al.* [MSR14] which is a first abstraction of the previous modelling, then we continue with the model of Gérault *et al.* [Gér+20] which uses advanced abstraction techniques that improve tightness of the model.

Variables abstractions

During Step-1, we are only aware of the Δ_A variables and we want to minimize the hamming weight of variables passing through S-Boxes (*i.e.* Δ_{SX} and Δ_{SRK}). In the case of AES_{128}, we have the $\Delta_X, \Delta_{SX}, \Delta_Y, \Delta_K, \Delta_{WK}, \Delta_{SWK}, \Delta_{RK}$ and Δ_{SRK} variables which correspond to their respective differential bytes, *e.g.* $\Delta_X[0,0,0]$ corresponds to $\delta_X[0,0,0]$.

Operation abstractions

`SubBytes`. In many ciphers, as in AES, the S-Boxes are bijective, so we have $S(x) \oplus S(x \oplus 0) = 0$ for any x. This means that if there is a difference at the input of the S-Box, then a difference necessarily remains at the output of the S-Box. When there is no difference at the input of the S-Box, then there can be no difference at the output of the S-Box. This can be translated by:

$$\begin{aligned} &\forall i \in [0;\ N_r[, \forall j \in [0;\ 3], \forall k \in [0;\ 3], \\ &\quad \Delta_X[i,j,k] = \Delta_{SX}[i,j,k] \end{aligned}$$

The same constraint can be used for the Δ_{WK} and the Δ_{SWK} variables.

`ShiftRow`.

`ShiftRow` is a leftward byte shift. It can be represented directly using the equality constraint:

$$\begin{aligned} &\forall i \in [0;\ N_r[, \forall j \in [0;\ 3], \forall k \in [0;\ 3], \\ &\quad \Delta_Y[i,j,k] = \Delta_{SX}[i,j,(j+k) \mod 4] \end{aligned}$$

`MixColumns`.

The abstraction of `MixColumns` is a quite more complex due to the mix of XORs and Galois Field multiplications. Usually the **MDS** property (see Section 20) is used to model this operation. We remind that the **MDS** property ensures that the sum of non-null input and output differences is either 0 or greater than 4:

$$\begin{aligned} &\forall i \in [0;\ N_r - 2], \forall k \in [0;\ 3], \\ &\quad \sum_{j=0}^{3} \Delta_Y[i,j,k] + \sum_{j=0}^{3} \Delta_Z[i,j,k] \in \{0,5,6,7,8\} \end{aligned}$$

`AddRoundKey`.

The add round key performs a XOR between the state and the key. This requires to focus how to abstract the XOR operator properly.

For three differential Boolean Δ_A, Δ_B and Δ_C with $\delta_a \oplus \delta_b \oplus \delta_c = 0$, the set of possible solutions for Δ_A, Δ_B and Δ_C is depicted in Table 3.1.

To match this solutions, the XOR abstraction can be modelled by the following constraint:

$$\texttt{XOR}(\Delta_A, \Delta_B, \Delta_C) = \Delta_A + \Delta_B + \Delta_C \neq 1.$$

The `AddRoundKey` is then:

$$\begin{aligned} &\forall i \in [0;\ N_r[, \forall j \in [0;\ 3], \forall k \in [0;\ 3], \\ &\quad \texttt{XOR}(\Delta_X[i+1,j,k], \Delta_{RK}[i+1,j,k], \Delta_Z[i,j,k]) \end{aligned}$$

$dom(\delta_A)$	$dom(\delta_B)$	$dom(\delta_A \oplus \delta_B)$	Δ_A	Δ_B	$\Delta_A \oplus_{abs} \Delta_B$
$\{0\}$	$\{0\}$	$\{0\}$	0	0	0
$\{0\}$	$[1;\ 255]$	$[1;\ 255]$	0	1	1
$[1;\ 255]$	$\{0\}$	$[1;\ 255]$	1	0	1
$[1;\ 255]$	$[1;\ 255]$	$[0;\ 255]$	1	1	$\begin{cases} 1 \text{ when } A \neq B \\ 0 \text{ when } A = B \end{cases}$

Table 3.1: *The abstraction of the* XOR *operator. The possible values of a difference δ_X is denoted by $dom(\delta_X)$. When both bytes are equals to 0, then the output is also equals to 0. When only one of the two bytes is equal to 0, then we know that the output is strictly positive. When both bytes are positive, we cannot know if the output should be in $\{0\}$ or in $[1;\ 255]$.*

`KeySchedule`.

Like for Step-2, the abstraction of the `KeySchedule` can be modelled using the techniques used for `SubBytes` and `AddRoundKey`.

3.1.3 Summary

The AES model that uses the techniques above has been presented by Minier *et al.* in [MSR14] and then taken over in [Gér+20] under the name of CP_{Basic} model. It is fully detailed in Model 3.1 and its constraints are explained below:

- C_1 models the function to minimize, here we want to activate the less S-Boxes as possible,
- C_2 models the `SubBytes` operation and the S-Box translations in the `KeySchedule`,
- C_3 models the `AddRoundKey` operation,
- C_4 models the `ShiftRow` operation,
- C_5 and C_6 model the `MixColumns` operation. The first constraint uses the **MDS** property of M, the second is used to model the special case for the final round of AES, for which `MixColumns` is omitted,
- C_7 and C_8 model the `KeySchedule`.

This model is not tight enough to compute truncated solutions as it generates a lot of infeasible truncated differential characteristics, leading in a waste of computation time during Step-2 when the process will try to instantiate these invalid solutions. This lack is mainly due to the bad quality of the XOR abstraction as depicted in the next example.

Example 3.1

We abstract the two following equations:

$$\delta_A \oplus \delta_B \oplus \delta_C \oplus \delta_D = 0$$
$$\delta_C \oplus \delta_D \oplus \delta_E = 0$$

and obtain the two constraints:

$$\begin{array}{ccccccccccc} \Delta_A & + & \Delta_B & + & \Delta_C & + & \Delta_D & & & \neq 1 \\ & & & & \Delta_C & + & \Delta_D & + & \Delta_E & \neq 1 \end{array}$$

If we try the instantiation $I = ((\Delta_A, 1), (\Delta_B, 0), (\Delta_C, 1), (\Delta_D, 1), (\Delta_E, 0))$, we obtain the system:

$$\begin{array}{ccccccccccc} 1 & + & 0 & + & 1 & + & 1 & & & \neq 1 \\ & & & & 1 & + & 1 & + & 0 & \neq 1 \end{array}$$

In the abstract representation, the system is valid since both sums are different of one, but the concrete system is invalid since if we XOR the two equations we obtain: $\delta_A \oplus \delta_B \oplus \delta_E = 0$ which should be abstracted by $\Delta_A + \Delta_B + \Delta_E = 1 + 0 + 0 \neq 1$ which is invalid.

Improving the model tightness

To counter this drawback Gérault *et al.* use advanced constraints to tighten the Step-1 model.

Firstly they introduce new XOR equations generated from the `KeySchedule` called $xorEq_l$. Hence the new equations are redundant in the concrete representation, they are not in the abstract one and can filter invalid cases such as the one shown just before.

In addition to that, they introduce *diff* variables which indicate whether two differential bytes are different or not. This work leads to the Model 3.2, which will be explained here.

The new introduced constraints are:

- C'_9 introduces the notion of *diff* variables. The *diff* variables are new variables that represent whether two bytes are different or not, *i.e.* $diff_{\delta_A,\delta_B} = 1 \iff \delta_A \neq \delta_B$. The constraint C'_7 models the symmetry of the difference relation, *i.e.*, $A \neq B \iff B \neq A$,
- C'_{10} ensures the transitivity of the equality property for the *diff* variables, *i.e.* $((\delta_A = \delta_B) \wedge (\delta_B = \delta_C)) \implies (\delta_A = \delta_C)$. This can be modelled with $diff_{\delta_A,\delta_B} + diff_{\delta_B,\delta_C} + diff_{\delta_A,\delta_C} \neq 1$ since it avoids to have only one active difference: either there is no difference and $\delta_A = \delta_B = \delta_C$ or there is at least two differences because we cannot have something like $(\delta_A = \delta_B) \wedge (\delta_B = \delta_C) \wedge (\delta_A \neq \delta_C)$,
- C'_{11} links the *diff* variables to their corresponding Δ variables,
- C'_{12} asserts that the XOR result of a variable $\delta_A = \delta_B \oplus \delta_C$ (resp. δ_B and δ_C) is only strictly positive if there is a difference between δ_B and δ_C (resp. (δ_A and δ_C) and (δ_A and δ_B)),
- C'_{13} performs the same reasoning on equations with four variables,
- C'_{14} encodes the **MDS** property over the differences in the Y and the Z states. Since `MixColumns` is linear we have $\delta_Z \oplus \delta_{Z'} = \texttt{MixColumns}(\delta_Y) \oplus \texttt{MixColumns}(\delta'_Y) = \texttt{MixColumns}(\delta_Y \oplus \delta'_Y)$, thus the **MDS** property also holds on the *diff* variables for Y and Z. This constraint is the one that eliminates the most inconsistent solutions during Step-1.
- C'_{15} links the *diff* variables for the X, Z and K states which are linked by the `AddRoundKey` operation.

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

$$(C_2) \quad \forall \delta B \in Sboxes_l, \Delta_{SB} = \Delta_B$$

$$(C_3) \quad \forall i \in [0;\ N_r - 2], \forall j,k \in [0;\ 3], \texttt{XOR}(\Delta_Z[i,j,k], \Delta_K[i+1,j,k], \Delta_X[i+1,j,k])$$

$$(C_4) \quad \forall i \in [0;\ N_r[, \forall j,k \in [0;\ 3], \Delta Y_i[j][k] = \Delta_{SX}[i,j,(j+k) \mod 4]$$

$$(C_5) \quad \forall i \in [0;\ N_r - 2], \forall k \in [0;\ 3], \left(\sum_{j=0}^{3} \Delta_Y[i,j,k] + \Delta_Z[i,j,k] \right) \in \{0,5,6,7,8\}$$

$$(C_6) \quad \forall j,k \in [0;\ 3], \Delta_Z[r-1,j][k] = \Delta_Y[r-1,j,k]$$

$$(C_7) \quad \forall i \in [0;\ N_r[, \forall j \in [0;\ 3], \texttt{XOR}(\Delta_K[i+1,j,0], \Delta[i,j,0], \Delta_{SRK}[i,(j+1) \mod 4][3])$$

$$(C_8) \quad \forall i \in [0;\ N_r[, \forall j \in [0;\ 3], \forall k \in [1,3], \texttt{XOR}(\Delta_K[i+1,j,k], \Delta_K[i+1,j,k-1], \Delta_K[i,j,k])$$

Model 3.1: *Constraints of the CP_{Basic} model for Step-1 of AES_{128}. [Gér+20]*

$$(C'_9) \quad \forall D \in \{DK_j, DY_j, DZ_j : j \in [0;\ 3]\}, \forall \{\delta B_1, \delta B_2\} \subseteq D, \mathit{diff}_{\delta B_1, \delta B_2} = \mathit{diff}_{\delta B_2, \delta B_1}$$

$$(C'_{10}) \quad \forall D \in \{DK_j, DY_j, DZ_j : j \in [0;\ 3]\}, \forall \{\delta B_1, \delta B_2, \delta B_3\} \subseteq D, \mathit{diff}_{\delta B_1, \delta B_2} + \mathit{diff}_{\delta B_2, \delta B_3} + \mathit{diff}_{\delta B_1, \delta B_3} \neq 1$$

$$(C'_{11}) \quad \forall D \in \{DK_j, DY_j, DZ_j : j \in [0;\ 3]\}, \forall \{\delta B_1, \delta B_2\} \subseteq D, \mathit{diff}_{\delta B_1, \delta B_2} + \Delta B_1 + \Delta B_2 \neq 1$$

$$(C'_{12}) \quad \forall (\delta B_1 \oplus \delta B_2 \oplus \delta B_3 = 0) \in xorEq_l, (\mathit{diff}_{\delta B_1, \delta B_2} = \Delta B_3) \wedge (\mathit{diff}_{\delta B_1, \delta B_3} = \Delta B_2) \wedge (\mathit{diff}_{\delta B_2, \delta B_3} = \Delta B_1)$$

$$(C'_{13}) \quad \forall (\delta B_1 \oplus \delta B_2 \oplus \delta B_3 \oplus \delta B_4 = 0) \in xorEq_l, (\mathit{diff}_{\delta B_1, \delta B_2} = \mathit{diff}_{\delta B_3, \delta B_4}) \wedge (\mathit{diff}_{\delta B_1, \delta B_3} = \mathit{diff}_{\delta B_2, \delta B_4}) \wedge (\mathit{diff}_{\delta B_1, \delta B_4} = \mathit{diff}_{\delta B_2, \delta B_3})$$

$$(C'_{14}) \quad \forall i_1, i_2 \in [0;\ N_r - 2], \forall k_1, k_2 \in [0;\ 3], \sum_{j=0}^{3} \mathit{diff}_{\delta_Y[i_1,j,k_1], \delta_Y[i_2,j,k_2]} + \mathit{diff}_{\delta_Z[i_1,j,k_1], \delta_Z[i_2,j,k_2]} \in \{0,5,6,7,8\}$$

$$(C'_{15}) \quad \forall i_1, i_2 \in [0;\ N_r - 2], \forall j, k_1, k_2 \in [0;\ 3], \mathit{diff}_{\delta_K[i_1+1,j,k_1], \delta_K[i_2+1,j,k_2]} + \mathit{diff}_{\delta_Z[i_1,j,k_1], \delta_Z[i_2,j,k_2]} + \Delta_X[i_1+1,j,k_1] + \Delta_X[i_2+1,j,k_2] \neq 1$$

Model 3.2: *Constraints of the CP_{XOR} model for Step-1 [Gér+20]. The model reuses the constraints (C_1) to (C_8) of the CP_{Basic} model*

3.1.4 Incomplete Step-1 Solutions

As pointed out in [Gér+20], some AES instances have a huge number of Step-1 solutions. Many of these solutions have exactly the same values for the Boolean variables in Δ_B (corresponding to S-Boxes), and they only differ on the values of other Boolean variables (that do not correspond to S-Boxes). For example, when the key has 192 bits and the number of rounds is equal to 10, there are 27,548 different Step-1 solutions. However, there are only 7 different assignments of values to the variables in Δ_B. In order to list only solutions with different S-Boxes, we add, at the beginning of each search, a constraint that prohibits having the same set of active S-Boxes as the previous solutions. Since the other variables may have different values between two Step-1 solutions, only the S-Box variables are taken into account when initialling the Step-2 domains.

3.1.5 Computation of optimal characteristics for a given truncated characteristics

When we have the Step-1 model and the Step-2 model, we need an interface to switch from the first one to the second one. This is done by reducing the differential byte domains knowing their abstract counterparts. During the Step-1 process, only S-Box variables are instantiated, thus the initialization of the Step-2 variable domains is done in the following way:

```
1  if δ_B ∈ Sboxes_l then
2  |  given δ_SB = S(δ_X) and p = −log_2(Pr[δ_B ⇝ δ_SB])
3  |  if Δ_B = 0 then
4  |  |  dom(δ_B) ← {0}
5  |  |  dom(δ_SB) ← {0}
6  |  |  dom(p) ← {0}
7  |  else
8  |  |  dom(δ_B) ← [1; 255]
9  |  |  dom(δ_SB) ← [1; 255]
10 |  |  dom(p) ← {6, 7}
11 |  end
12 else
13 |  dom(δ_B) ← [0; 255]
14 end
```

Since probability variables with null values do not enter into account when summing the $-\log_2$ probabilities, they can be discarded from the Step-2 model.

The complete procedure then consists of (i) finding the truncated differential characteristic with the minimum number of S-Boxes, (ii) enumerating the set of different truncated differential characteristics with the minimum number of S-Boxes found in (i) and (iii) calculating for each truncated differential characteristic the optimal Step-2 solution. The best solution is then the best solution among the optimal solutions found in Step-2.

In the next section, we see the model proposed by Delaune *et al.* to compute boomerang

distinguishers.

3.2 Delaune *et al.*'s model [DDV20]

In [DDV20], Delaune *et al.* propose a model divided into two steps to search for optimal boomerang distinguishers (described in Section 1.4.4) on **SPN** ciphers. In Step-1, a **MILP** model searches for truncated boomerangs where each S-Box is represented by 6 Boolean variables (described below). The solutions of Step-1 are the input of a Step-2 search that tries to instantiate those truncated boomerangs with concrete nibble (4-bit sequence) differences so that the distinguisher has the highest possible probability.

A boomerang distinguisher uses two differential trails, one is called the upper trail and determines α, the input difference of the distinguisher. The other one is called the lower trail and determines δ, the output of the distinguisher (see Figure 1.11). In the model proposed in [DDV20] the division as a sandwich [DKS10; DKS14] is not made but the upper and lower trails are searched on all the rounds. In what follows we denote by $\delta_X[i,k]$ the nibble difference at the input of an S-Box and by $\delta_{SX}[i,k]$ the corresponding output difference of the S-Box.

3.2.1 The S-Box abstraction in truncated boomerangs

The boomerang model of [DDV20] on Skinny uses six variables for each S-Box in its Step-1: 3 variables relate to the upper trail (in the encryption direction) whereas the 3 others relate to the lower trail (in the decryption direction). These variables are used to select the proper boomerang transition tables (described right after) and are defined as:

- $\Delta_{X\mathsf{up}}[i,k]$[1] (respectively $\Delta_{X\mathsf{lo}}[i,k]$) is a Boolean variable that indicates if the nibble difference $\delta_X[i,k]$ is active in the upper (resp. lower) trail, considering that it represents the S-Box input,
- $\mathsf{free}_{X\mathsf{up}}[i,k]$ (respectively $\mathsf{free}_{X\mathsf{lo}}[i,k]$) is a Boolean variable that indicates if the nibble difference $\delta_X[i,k]$ is free of conditions, *i.e.* it can take any value with a uniform probability in the upper (resp. lower) trail,
- $\mathsf{free}_{SX\mathsf{up}}[i,k]$ is a Boolean variable that indicates if the nibble difference $\delta_{SX}[i,k]$ can take any value with a uniform probability in the upper trail as an output of the S-Box. $\mathsf{free}_{SX\mathsf{lo}}[i,k]$ is a Boolean variable that indicates if the nibble difference $\delta_{SX}[i,k]$ can take any value with a uniform probability in the lower trail.

Note that the $\mathsf{free}_{SX\mathsf{up}}[i,k]$ variable represents the state of the variable *after* the S-Box in the *encryption* direction, so $\mathsf{free}_{SX\mathsf{lo}}[i,k]$ can be seen as the *input* state of the S-Box in the *decryption* direction.

Several constraints describe the relations between these variables, starting with the one modelling the propagation of the free states through the S-Boxes: if a variable is free before an S-Box, it is

[1] These variables are called $\texttt{isActive}_\texttt{X}$ in [DDV20], but we call them Δ_X to be consistent with the notations introduced in [Gér+20] and this PhD thesis.

also free after the S-Box. Since the propagation is done in the opposite direction for the lower trail, the implication is in the other direction for the lo variables.

$$\mathsf{free}_{X\mathsf{up}} \implies \mathsf{free}_{SX\mathsf{up}}$$
$$\mathsf{free}_{SX\mathsf{lo}} \implies \mathsf{free}_{X\mathsf{lo}}$$

The second rule ensures that if an S-Box output is free then the S-Box input must be non-zero. Again the lower trail is reversed since it represents the decryption direction.

$$\mathsf{free}_{SX\mathsf{up}} \implies \Delta_{X\mathsf{up}}$$
$$\mathsf{free}_{X\mathsf{lo}} \implies \Delta_{X\mathsf{lo}}$$

The third rule ensures that we can compute the probability of the S-Box by setting a minimum number of parameters.

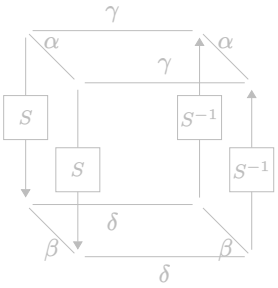
$$\neg(\mathsf{free}_{X\mathsf{up}} \wedge \mathsf{free}_{X\mathsf{lo}})$$
$$\neg(\mathsf{free}_{SX\mathsf{up}} \wedge \mathsf{free}_{SX\mathsf{lo}})$$

Finally, for any linear operation there is a constraint stating that if any input variable is free then all the output variables on which it depends are also free.

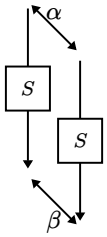
Boomerang transition tables

As for the differential attacks, boomerang attacks use transition tables to compute S-Box transition probabilities. In the case of boomerang attacks, we have several tables which depend on the S-Box states. Variables and rules seen just before ensure that each input and output of an S-Box of a trail may have 3 different states. These three states are: fixed to 0, fixed to a positive value or free. This states allow to use one of the transitions depicted in Figure 3.1. In the model, a transition table is chosen if its corresponding predicate is true (see Model 3.3).

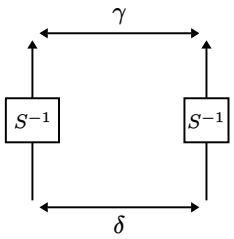
Given this set of constraints, the solver is going to choose the best truncated trail among the ones with a valid propagation of differences, where the quality of a trail is measured by the best probability it might reach. Based on this, the probability of the Step-1 solution is calculated by assuming that the best transition in each table is met. Once the best solution for Step-1 is found it is given as input to the Step-2 model, which looks for a concrete instance of the upper and lower trails, again with the objective of reaching the best possible probability.



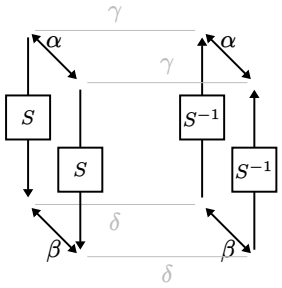
(a) *A pair of S-Boxes representation in Delaune* et al. *model with α, β, γ and δ parameters. α is the upper trail S-Box input difference, β the upper trail S-Box output difference, γ is the lower trail S-Box output difference (in S^{-1} direction) and δ the lower trail input S-Box difference (in S^{-1} direction). In the following figures, hidden variables are variables fixed to 0, black variables are positive fixed variables and gray variables are free.*



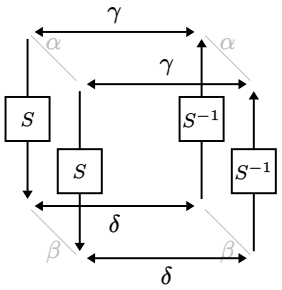
(b) *Transition representing a* DDT *for the upper trail. This transition requires to known α and β parameters. The γ and δ parameters are also known but fixed to* 0.



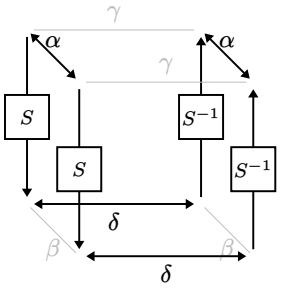
(c) *Transition representing a* DDT *for the lower trail. This transition requires to known γ and δ parameters. The α and β parameters are also known but fixed to* 0.



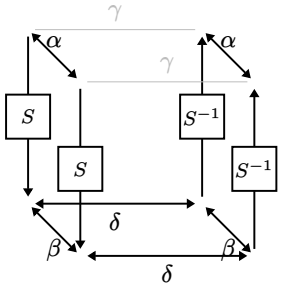
(d) *Transition representing a* $\texttt{DDT}^2$ *for the upper trail. This transition requires to known α and β parameters.*



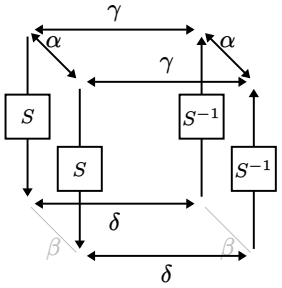
(e) *Transition representing a* $\texttt{DDT}^2$ *for the lower trail. This transition requires to known γ and δ parameters.*



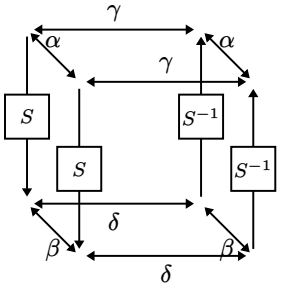
(f) *Transition representing a* BCT. *This transition requires to known α and δ parameters.*



(g) *Transition representing an* UBCT. *This transition requires to known α, β and δ parameters.*



(h) *Transition representing a* LBCT. *This transition requires to known α, γ and δ parameters.*



(i) *Transition representing an* EBCT. *This transition requires to known α, β, γ and δ parameters.*

Figure 3.1: *The different transitions used in the model of Delaune* et al.*. Each transition requires a set of variables to be calculated.*

$$\texttt{predicate}\ \ \texttt{isBCT}_{\texttt{X}}(i,j,k) = \left(\begin{array}{c}\Delta_{X_{\text{up}}}[i,j,k] \wedge \neg\text{free}_{X_{\text{up}}}[i,j,k] \wedge \text{free}_{SX_{\text{up}}}[i,j,k] \\ \wedge\ \Delta_{X_{\text{lo}}}[i,j,k] \wedge \text{free}_{X_{\text{lo}}}[i,j,k] \wedge \neg\text{free}_{SX_{\text{lo}}}[i,j,k]\end{array}\right)$$

$$\texttt{predicate}\ \ \texttt{isDDT}_{\texttt{X}}(i,j,k) = \texttt{isDDT}_{\texttt{X}}^{\text{up}}(i,j,k) \vee \texttt{isDDT}_{\texttt{X}}^{\text{lo}}(i,j,k)$$

$$\texttt{predicate}\ \ \texttt{isDDT}_{\texttt{X}}^{\text{up}}(i,j,k) = \left(\Delta_{X_{\text{up}}}[i,j,k] \wedge \neg\Delta_{X_{\text{lo}}}[i,j,k] \wedge \neg\text{free}_{X_{\text{up}}}[i,j,k] \wedge \neg\text{free}_{SX_{\text{up}}}[i,j,k]\right)$$

$$\texttt{predicate}\ \ \texttt{isDDT}_{\texttt{X}}^{\text{lo}}(i,j,k) = \left(\neg\Delta_{X_{\text{up}}}[i,j,k] \wedge \neg\text{free}_{X_{\text{lo}}}[i,j,k] \wedge \neg\text{free}_{SX_{\text{lo}}}[i,j,k] \wedge \Delta_{X_{\text{lo}}}[i,j,k]\right)$$

$$\texttt{predicate}\ \ \texttt{isDDT}_{\texttt{X}}^{2}(i,j,k) = \texttt{isDDT}_{\texttt{X}}^{2\text{up}}(i,j,k) \vee \texttt{isDDT}_{\texttt{X}}^{2\text{lo}}(i,j,k)$$

$$\texttt{predicate}\ \ \texttt{isDDT}_{\texttt{X}}^{2\text{up}}(i,j,k) = \left(\begin{array}{c}\Delta_{X_{\text{up}}}[i,j,k] \wedge \neg\text{free}_{X_{\text{up}}}[i,j,k] \wedge \neg\text{free}_{SX_{\text{up}}}[i,j,k] \\ \wedge\ \ \Delta_{X_{\text{lo}}}[i,j,k] \wedge \text{free}_{X_{\text{lo}}}[i,j,k] \wedge \text{free}_{SX_{\text{lo}}}[i,j,k]\end{array}\right)$$

$$\texttt{predicate}\ \ \texttt{isDDT}_{\texttt{X}}^{2\text{lo}}(i,j,k) = \left(\begin{array}{c}\Delta_{X_{\text{up}}}[i,j,k] \wedge \text{free}_{X_{\text{up}}}[i,j,k] \wedge \text{free}_{SX_{\text{up}}}[i,j,k] \\ \wedge\ \ \Delta_{X_{\text{lo}}}[i,j,k] \wedge \neg\text{free}_{X_{\text{lo}}}[i,j,k] \wedge \neg\text{free}_{SX_{\text{lo}}}[i,j,k]\end{array}\right)$$

$$\texttt{predicate}\ \ \texttt{isLBCT}_{\texttt{X}}(i,j,k) = \left(\begin{array}{c}\Delta_{X_{\text{up}}}[i,j,k] \wedge \neg\text{free}_{X_{\text{up}}}[i,j,k] \wedge \text{free}_{SX_{\text{up}}}[i,j,k] \\ \wedge\ \Delta_{X_{\text{lo}}}[i,j,k] \wedge \neg\text{free}_{X_{\text{lo}}}[i,j,k] \wedge \neg\text{free}_{SX_{\text{lo}}}[i,j,k]\end{array}\right)$$

$$\texttt{predicate}\ \ \texttt{isUBCT}_{\texttt{X}}(i,j,k) = \left(\begin{array}{c}\Delta_{X_{\text{up}}}[i,j,k] \wedge \neg\text{free}_{X_{\text{up}}}[i,j,k] \wedge \neg\text{free}_{SX_{\text{up}}}[i,j,k] \\ \wedge\ \Delta_{X_{\text{lo}}}[i,j,k] \wedge \text{free}_{X_{\text{lo}}}[i,j,k] \wedge \neg\text{free}_{SX_{\text{lo}}}[i,j,k]\end{array}\right)$$

$$\texttt{predicate}\ \ \texttt{isEBCT}_{\texttt{X}}(i,j,k) = \left(\begin{array}{c}\Delta_{X_{\text{up}}}[i,j,k] \wedge \neg\text{free}_{X_{\text{up}}}[i,j,k] \wedge \neg\text{free}_{SX_{\text{up}}}[i,j,k] \\ \wedge\ \Delta_{X_{\text{lo}}}[i,j,k] \wedge \neg\text{free}_{X_{\text{lo}}}[i,j,k] \wedge \neg\text{free}_{SX_{\text{lo}}}[i,j,k]\end{array}\right)$$

Model 3.3: *Link between binary variables and tables: for each table* $\texttt{T} \in \{\texttt{BCT}, \texttt{DDT}, \texttt{DDT}^2, \texttt{LBCT}, \texttt{UBCT}, \texttt{EBCT}\}$*, the predicate* $\texttt{isT}_{\texttt{X}}(i,j,k)$ *is true iff table* $\texttt{T}$ *must be used to link* $\delta_{X\text{up}}[i,k], \delta_{SX\text{up}}[i,j,k], \delta_{X\text{lo}}[i,k]$ *and* $\delta_{SX\text{lo}}[i,k]$*.*

3.2.2 Step-2 boomerang

The Step-2 modelling of boomerang attacks reuses the modelling techniques seen for simple differential attacks but has some special features.

The first difference is the doubling of variables and constraints. Indeed, one differential must be modelled for the upper trail and another one for the lower trail. We will see later how these two trails are linked using the transitions between the S-Boxes.

The second difference is the appearance of free variables. Since these variables can take any value uniformly without impacting the results they can be removed from the model. In addition, the set of equations containing a free variable can also be removed from the model.

The last feature is the modelling of S-Boxes, which leads us to see how to link Step-1 and Step-2.

3.2.3 Linking Step-1 and Step-2 boomerang

The link between Step-1 and Step-2 is made in the similar way as presented for differentials. The main difference is that Step-1 defines which transitions will be used in Step-2. Let be δ_B a variable of the model, we have:

```
1 if ΔB = 0 then
2 |  dom(δB) ← {0}
3 else
4 |  if freeB then
5 |  |  dom(δB) ← [0; 16[ // The variable may be removed.
6 |  else
7 |  |  dom(δB) ← [1; 16[
8 |  end
9 end
```

As in the differential models, only S-Boxes are enumerated during Step-1. In the case of non S-Box variables, it is possible to compute their Δ and free values by propagating the Δ and free states of the S-Boxes through the linear part of the cipher. Once we have recovered these states, we can use the previous technique to obtain their concrete domain.

Linking the two trails. Like in differential model, we use table constraints to constraint the S-Boxes. These table constraints are also the constraints responsible for the consistency between the two trails.

Let be the following example:

Example 3.2

Let be a pair of S-Boxes $(\delta_{X\text{up}}, \delta_{X\text{lo}})$ where $\delta_{X\text{lo}}$ is the lower trail counterpart of $\delta_{X\text{up}}$. The Step-1 model returns these values:

$$\Delta_{X\text{up}} = 1 \quad \text{free}_{X\text{up}} = 0 \quad \text{free}_{SX\text{up}} = 1$$
$$\Delta_{X\text{lo}} = 1 \quad \text{free}_{X\text{lo}} = 0 \quad \text{free}_{SX\text{lo}} = 0$$

The variable domains are:

$$\texttt{dom}(\delta_{X\text{up}}) = [1;\ 16[\quad \texttt{dom}(\delta_{SX\text{up}}) = [0;\ 16]$$
$$\texttt{dom}(\delta_{X\text{lo}}) = [1;\ 16[\quad \texttt{dom}(\delta_{SX\text{lo}}) = [1;\ 16[$$

If we refer to Model 3.3, we have to use the LBCT table which relates to the $\delta_{X\text{up}}, \delta_{X\text{lo}}$ and $\delta_{SX\text{lo}}$ variables. So we add the following constraint to the model:

$$(\delta_{X\text{up}}, \delta_{X\text{lo}}, \delta_{SX\text{lo}}) \in \mathrm{T}_{\texttt{LBCT}}$$

As can be seen, the variable $\delta_{SX\text{up}}$ is not constrained by the LBCT table. Moreover, since its free state is propagated by the cipher, it cannot be constrained anymore and may be removed from the model.

Additional information on the propagation of free variables. In the linear part of encryption, as with S-boxes, the appearance of a free input parameter in a function automatically causes all its output variables to become free. This state is then propagated for all successive states. Therefore, when a state goes to the free state, it becomes unconstrained and can be removed from the model.

3.3 Conclusion

In this chapter we have seen two advanced models for calculating differential characteristics. The first, by Gérault *et al.*, is a model for calculating differential characteristics in related-key mode. It uses advanced constraints in Step-1 in order to obtain good performances that allow reaching the last rounds of AES. The second model, by Delaune *et al.* is used to automate the search for boomerang distinguishers on Skinny. Here again, the model uses advanced techniques to calculate the probability of a boomerang return with great accuracy. Although both models are powerful, they can be improved or extended. During this thesis, we extended the work of Gérault *et al.* to Rijndael while improving the performance on AES in Chapter 4. We also took an orthogonal approach by trying to develop new solving techniques during Step-1 in Chapter 5. We took up the work of Delaune *et al.* in order to adapt it to Rijndael in a first step (Chapter 6), then to extend it to the Feistel family in a second step (chapter 7). In order to prove the viability of our

model on Feistel ciphers we applied it to WARP, TWINE and LBlock–s.

Part

II

Contributions

Chapter

4 Differential Cryptanalysis of Rijndael

Contents

Scaling from the AES to Rijndael can only be made with a tight model for Step-1. Models described in [BN10; FJP13] do not scale when increasing the block size and the key size. Only the model [Gér+20] which is described in section 3.1 has a sufficiently small number of solutions found at Step-1 to hope that the computational time will be reasonable.

In this chapter, we show how to adapt the two-step solving process of [Gér+20] dedicated to the AES to compute optimal related-key differential characteristics for Rijndael [01]. Both steps are solved with Constraint Programming (**CP**) solvers: Picat-SAT for Step-1 and Choco for Step-2. We improve the approach of [Gér+20] by better interleaving Steps 1 and 2 and exploiting bounds to stop the search sooner. We also improve the Step-2 process of [Gér+20] by decomposing the constraints associated with `MixColumns`. These improvements allow us to compute the optimal differential characteristics for all Rijndael instances but one within a reasonable amount of time.

Rijndael, which is fully described in Section 1.2.1, is a family of block ciphers (more precisely it is composed of 25 instances of the same cipher where the block size and the key size vary) originally proposed at the AES competition. But the NIST only retained as a standard its 128-bit-block version under the key sizes 128, 192 and 256 bits and studying the security of Rijndael may be interesting to enlighten the AES standardization process. Among the most interesting results, we obtain a 12-round (over 13 rounds) related-key differential distinguisher and attack for Rijndael with a block size equal to 128 bits and a key size equal to 224 bits. We also obtain an 11-round

related-key differential distinguisher for Rijndael with a block size equal to 160 bits and a key size equal to 256 bits leading to an attack on 12 rounds out of 14.

When looking at the state of the art concerning the cryptanalysis of Rijndael, some of the results are in the single key scenario [NP07; Zha+08; GM08], [Wan+13; Min17; Liu+19] or in the related-key scenario [Wan+15] and none of those attacks exceeds 10 rounds.

The rest of this chapter is organized as follows: in Section 4.1, we detail the methods and our **CP** models; in Section 4.2, we sum up all the related-key differential characteristics distinguishers we obtained, give all solving times and compare them with those of [Gér+20]; in Section 4.3, we present two attacks based on the most efficient distinguishers and finally, in Section 4.4, we conclude this chapter.

4.1 The solving process

In this section, we describe how to compute the optimal related-key differential characteristics for Rijndael by adapting and improving the approach introduced in [Gér+20] for the AES. We first describe the two-step solving process; then, we describe the **CP** models associated with each of these two steps.

4.1.1 Two-step solving process

Finding the best related-key differential characteristic is a highly combinatorial problem. In 1995, Knudsen has introduced truncated differentials [Knu95] that can be used to improve the scalability of the attacks. The core idea is to solve the problem in two steps: In Step-1, we compute a truncated differential characteristic S_1 where each differential byte δA of the ciphering process is replaced with a boolean variable ΔA that indicates whether δA contains a difference or not (i.e., $\Delta A = 0 \iff \delta A = 0$ and $\Delta A = 1 \iff \delta A \in [1;\ 2^8 - 1]$); In Step-2, we instantiate S_1 into a differential characteristic S_2: for each boolean variable ΔA, if ΔA is equal to 0 in S_1, then δA is equal to 0 in S_2; otherwise δA must belong to $[1;\ 2^8 - 1]$. Note that some truncated characteristics cannot be instantiated to a characteristic because some abstractions are done at Step-1.

As `SubBytes` is the only non-linear operation, the probability of a differential characteristic only depends on the values of the differential bytes that pass through S-Boxes, under the Markov assumption that rounds are independent. We denote $\delta_{\texttt{SB}}$ this set of bytes (including those in the `KeySchedule`), and $\Delta_{\texttt{SB}}$ the corresponding set of Boolean variables.

A theoretical upper bound on the probability of the best differential characteristic may be computed by searching for the truncated differential characteristic which minimizes the number of active S-Boxes $NB_{\texttt{SB}} = \#\{\Delta A \mid \Delta A \in \Delta_{\texttt{SB}} \wedge \Delta A = 1\}$. As 2^{-6} is the maximal differential probability of the Rijndael S-Box, the best probability is upper bounded by $UB = 2^{-6 \cdot NB_{\texttt{SB}}}$.

UB may be larger than the actual best probability because it may be possible that the best

Data: The size K_{len} of the key, the size C_{len} of the block and the number r of rounds
Result: An optimal related-key differential characteristic S^*

$NB_{\mathtt{SB}} \leftarrow$ `step1-opt`(N_k, N_b, r)
$UB \leftarrow 2^{-6 \cdot NB_{\mathtt{SB}}}$
$LB \leftarrow 0$
while $LB < UB$ **do**
  $S_1 \leftarrow$ `step1-next`$(N_k, N_b, r, NB_{\mathtt{SB}})$
  if $S_1 \neq$ null **then**
    $S_2 \leftarrow$ `step2-opt`(N_k, N_b, r, LB, S_1)
    if $S_2 \neq$ null **then**
      $S^* \leftarrow S_2$
      $LB \leftarrow$ probability of S_2
    end
  else
    $NB_{\mathtt{SB}} \leftarrow NB_{\mathtt{SB}} + 1$
    $UB \leftarrow 2^{-6 \cdot NB_{\mathtt{SB}}}$
  end
end
return S^*

Algorithm 4.1: *Computation of optimal related-key differential characteristics for* Rijndael.

truncated differential characteristic cannot be instantiated into a differential characteristic, or because some non null differential bytes that go through S-Boxes have a probability equal to 2^{-7} instead of 2^{-6}. Hence, the best differential characteristic is searched by alternating Step-1 and Step-2 in an iterative process which is described in Algorithm 4.1. First, we call *Step1-opt* to compute NB_{SB}, a lower bound of the number of active S-Boxes in a truncated differential, and this number is used to compute a first upper bound UB on the probability. The lower bound LB on the probability is initialised to 0. Then, at each iteration of the while loop, we call *Step1-next* to compute the next truncated differential characteristic with $NB_{\mathtt{SB}}$ active S-Boxes: each time this function is called, it returns a new Step-1 solution with $NB_{\mathtt{SB}}$ active S-Boxes until they all have been computed (in this latter case, *Step1-next* returns *null*). If a new Step-1 solution S_1 has been computed, then *Step2* is called to search for the differential characteristic S_2 corresponding to S_1 whose probability is larger than LB and maximal: if such a characteristic exists, then the best characteristic S^* is updated to S_2 and LB is updated to the probability of S_2. When *Step1-next* returns *null*, all truncated characteristics with $NB_{\mathtt{SB}}$ active S-Boxes have been enumerated. In this case, we increment $NB_{\mathtt{SB}}$ and update consequently the upper bound UB. We stop iterating when UB becomes smaller than or equal to LB: in this case S^* is equal to the optimal differential characteristic.

Algorithm 4.1 is different from the one used in [Gér+20]: it avoids computing useless Step-1 solutions by updating LB and UB and stopping the process when $LB \geq UB$. *Step1-opt*, *Step1-next* and *Step2* are implemented with **CP** solvers and the corresponding **CP** models are described in the next two sections.

4.1.2 Step-1

Both *Step1-opt* and *Step1-next* compute truncated differential characteristics: *Step1-opt* searches for the truncated characteristic that minimises $NB_{\texttt{SB}}$, whereas *Step1-next* searches for the next truncated characteristic given $NB_{\texttt{SB}}$. Both problems share the same constraints which are described in this section. *Step1-opt* is a COP which is obtained by adding the objective function: *minimise* $NB_{\texttt{SB}}$. *Step1-next* is a CSP which is obtained by assigning the variable $NB_{\texttt{SB}}$ to the optimal solution of *Step1-opt*.

A key point for Algorithm 4.1 to be efficient is to avoid as much as possible computing truncated characteristics which cannot be instantiated at Step-2. To this aim, we consider the model introduced in [Gér+20] which is tighter than the model of [GMS16], *i.e.*, it computes fewer truncated characteristics that cannot be instantiated at Step-2. This model has been defined for the AES, and we show in this section how to extend it to Rijndael.

4.1.3 Constraints associated with Rijndael transformations

A basic Step-1 model for Rijndael is displayed in Model 4.1. It is a straightforward adaptation of the model described in Section 3.1.2: the only differences are the constraints (A3), (A5), (A6) and (A7), which model `ShiftRow` and the `KeySchedule`.

Constraint (A1) relates $NB_{\texttt{SB}}$ with the number of active S-Boxes. The other constraints are derived from Rijndael round function transformations.

`SubBytes`: As `SubBytes` is bijective, there is an output difference if and only if there is an input difference. The `SubBytes` transformation at Boolean level is thus abstracted by an identity mapping ΔX_i and ΔSX_i (Constraint (A2)).

`ShiftRow`: As `ShiftRow` is just a shift at byte level, its abstraction in Step-1 is directly expressed as the equivalent shift as defined in Constraint (A3).

`MixColumns`: Multiplications of `MixColumns` cannot be mapped into the Boolean domain as the coefficients of M belong to GF(2^8). Thus, instead of encoding multiplications, we exploit the **MDS** (Maximum Distance Separable explained in Paragraph Maximum Distance Separable Property of Section 1.2.1) property of the `MixColumns` transformation as defined in Constraint (A4).

`KeySchedule`: the whole `KeySchedule` process of Rijndael is described in Algorithm 1.1. The variables that pass through S-Boxes are unchanged, as stated in Constraint (A6). XORs are modelled by Constraint (A7) which prevents every triple of boolean variables involved in a same XOR from having exactly one difference.

However, this simple model generates many truncated characteristics which cannot be instantiated at Step-2. This mainly comes from the fact that XORs performed by `AddRoundKey` and `KeySchedule` modelled by constraints which simply prevent the sum of differences to be equal to 1.

$$NB_{\texttt{SBox}} = \sum_{\Delta_A \in \Delta_{\texttt{SB}}} \Delta_A \tag{A1}$$

$$\forall i \in [0; N_r[, \forall j \in [0;3], \forall k \in [0; N_b[, \tag{A2}$$

$$\Delta_{SX}[i,j,k] = \Delta_X[i,j,k]$$

$$\forall i \in [0; N_r[, \forall j \in [0;3], \forall k \in [0; N_b[, \tag{A3}$$

$$\Delta_Y[i,j,k] = \Delta_{SX}[i,j,P_{N_b}[j]+k \mod N_b]$$

$$\forall i \in [0; N_r[, \forall k \in [0; N_b[, \tag{A4}$$

$$\sum_{j\in[0;3]} \Delta_Z[i,j,k] + \sum_{j\in[0;3]} \Delta_Y[i,j,k] \in \{0,5,6,7,8\}$$

$$\forall i \in [0; N_r[, \forall j \in [0;3], \forall k \in [0; N_b[, \tag{A5}$$

$$\Delta_{RK}[i,j,k] = \Delta_{WK}[j,(i+1)\times N_b + k]$$
$$\Delta_X[i+1,j,k] + \Delta_Z[i,j,k] + \Delta_{RK}[i,j,k] \neq 1$$

$$\forall \omega \in [N_b; N_b \times (N_r+1)[\text{ such that } \texttt{isSbCol}(\omega), \forall j \in [0;3], \tag{A6}$$

$$\Delta_{SWK}[j,\omega] = \Delta_{WK}[j,\omega]$$

where $\texttt{predicate isSbCol}(\omega) = \omega > N_k - 1 \wedge \omega < N_b \times (N_r+1) - 1 \wedge$
$(\omega \mod N_k = N_k - 1 \vee (N_k > 6 \wedge \omega \mod N_k = 3))$

$$\forall \omega \in [N_b; N_b \times (N_r+1)[, \forall j \in [0;3], \tag{A7}$$

$$\begin{aligned}
\textbf{if } \omega \mod N_k = 0 &: \Delta_{WK}[j,\omega] + \Delta_{WK}[j,\omega - N_k] + \Delta_{SWK}[(j+1) \mod 4, \omega - 1] \neq 1 \\
\textbf{else if } N_k > 6 \wedge \omega \mod N_k = 4 &: \Delta_{WK}[j,\omega] + \Delta_{WK}[j,\omega - N_k] + \Delta_{SWK}[j,\omega - 1] \neq 1 \\
\textbf{else} &: \Delta_{WK}[j,\omega] + \Delta_{WK}[j,\omega - N_k] + \Delta_{WK}[j,\omega - 1] \neq 1
\end{aligned}$$

Model 4.1: *Basic Step-1 model for* Rijndael

Thus, we show how to refine this in the next two paragraphs.

Inference of new XOR equations from the KeySchedule

In Model 4.1, every XOR equation $\delta A \oplus \delta B \oplus \delta C = 0$ is represented by a sum constraint $\Delta A + \Delta B + \Delta C \neq 1$. This simple model is not sharp enough and generates a lot of truncated solutions that cannot be instantiated at Step-2. For example, the two XOR equations

$$\delta A \oplus \delta B \oplus \delta C = 0 \text{ and } \delta B \oplus \delta C \oplus \delta D = 0$$

are represented by the two sum constraints

$$\Delta A + \Delta B + \Delta C \neq 1 \text{ and } \Delta B + \Delta C + \Delta D \neq 1 \ .$$

When reasoning at the byte level, we easily infer that we cannot have $\delta A = 0$ and $\delta D \neq 0$, whatever the values of δB and δC are. However, when reasoning at the boolean level, the two sum constraints may be satisfied when $\Delta A = 0$ and $\Delta D = 1$ (*e.g.*, when $\Delta B = \Delta C = 1$).

To sharpen the Step-1 model and reduce the number of Step-1 solutions that cannot be instantiated at Step-2, we generate new XOR equations from the initial set of equations, by XORing them. These new equations do not change the set of solutions at the byte level. However, at the boolean level, they remove some of the truncated solutions that cannot be instantiated at Step-2. For example, when XORing the two XOR equations of our previous example, we obtain the equation $\delta A \oplus \delta D = 0$. When adding the constraint $\Delta A + \Delta D \neq 1$ to the two sum constraints, we prevent the search from generating solutions with $\Delta A = 0$ and $\Delta D = 1$.

This trick has been introduced in [Gér+20] for the AES and detailled in Section 3.1.3 of Chapter 3, and we extend it to Rijndael in a straightforward way. More precisely, we consider the set of all XOR equations coming from the `KeySchedule` (this set corresponds to Constraint (A7) of Model 4.1). From this set, we generate all possible equations that involve no more than 4 variables by recursively XORing these equations[1]. This set of new equations is denoted ExtXOR.

Introduction to *diff* variables

As done in [Gér+20], we also introduce *diff* variables to reason on differences at the byte level, this new Model is presented in Model 4.2.

Constraints (E1) (for `KeySchedule`) and (E2) (for the `MixColumns`) relate *diff* variables to their respective Δ. These constraints also ensure symmetry, *i.e.*, $diff_{A,B} = diff_{B,A}$. Constraints (E3) and (E4) ensure transitivity (*i.e.*, if $\delta A = \delta B$ and $\delta B = \delta C$, then $\delta A = \delta C$) by constraining the sum of the corresponding *diff* variables to be different from 1.

Constraint (E5) relates *diff* variables associated with the subkey RK_i with *diff* variables associated

[1] We do not generate equations with more than 4 variables as the number of new equations grows exponentially with respect to their size.

with the expanded key WK.

Constraints (E6) and (E7) are associated with the new XOR equations in ExtXOR. Two cases are considered: equations with three variables in Constraint (E6), and equations with four variables in Constraint (E7). In both cases, if at least one variable involved in the equation belongs to Δ_{SB}, then the constraint simply prevents the sum of the variables to be equal to 1. Otherwise, we exploit *{diff}* variables to tighten the constraint.

Finally, Constraints (E8) and (E9) ensure the **MDS** property of `MixColumns` on differences between pairs of columns (this constraint is partly equivalent with the linear incompatibility of [FJP13]).

Incomplete Step-1 Solutions

As Rijndael is a generalization of the AES, this is also true for Rijndael. Hence, as proposed in [Gér+20] and explained in Section 3.1.4, we enumerate incomplete solutions such that only the variables in Δ_{SB} are assigned.

Step-2

Given a Step-1 solution (corresponding to a truncated characteristic), Step-2 aims at searching for the corresponding characteristic which has the largest probability, and such that this largest probability is larger than the best probability found so far (LB).

The **CP** model used to solve Step-2 is described in Model 4.3. The main difference with the model of Gérault *et al.* is the modelling of `ShiftRow`, `KeySchedule` and `MixColumns`. As Rijndael allows some parameters that are not allowed in AES, `ShiftRow` and `KeySchedule` for Rijndael may be sightly different from ones of AES. The modelling change of `MixColumns` is a design choice to try to speed up the solving process.

`KeySchedule`. The main difference between the `KeySchedule` of Rijndael and the `KeySchedule` of AES if the addition of a S-Box transition when $K_{len} > 192$. Constraint (C9) models the S-Boxes using constraint tables. Constraint (C10) models the XORs of the `KeySchedule`, using table $\mathtt{T}_{\oplus}$. Note that we do not represent XORs with constants as they are cancelled by differential cryptanalysis.

`ShiftRow`. Constraint (C3) models the `ShiftRow` operation for Rijndael. Since the performed shift depends of the block length parameter, the operation may differ from the one of AES. We adapt the constraint of [Gér+20] to match this modification.

`MixColumns`. Constraints (C4) to (C7) represent the `MixColumns` operation. We introduce new integer variables to represent the result of applying the Galois Field multiplication to a byte: for each value $v \in \{2, 3\}$, and each byte $\delta Y_i[j,k]$, the variable $v\delta Y_i[j,k]$ is constrained to be equal to $v \otimes \delta Y_i[j,k]$ by the table constraint (C4), where $\mathtt{T}_{\mathtt{MULv}}$ contains every couple $(\delta A, \delta B) \in [0;\ 255]^2$ such that $\delta B = v \otimes \delta A$. Then, Constraint (C5) ensures that $\delta Z_i[j,k]$ is equal to the result of XORing four bytes (corresponding to the bytes at column k of δY_i multiplied by the coefficients

$$\forall \omega_1, \omega_2 \in [0; N_b \times (N_r + 1)[, \forall j \in [0;3] \text{ where } \omega_2 > \omega_1 \tag{E1}$$

$$\begin{aligned} \mathit{diff}_{WK[j,\omega_1],WK[j,\omega_2]} + \Delta WK[j,\omega_1] + \Delta WK[j,\omega_2] &\neq 1 \\ \mathit{diff}_{WK[j,\omega_1],WK[j,\omega_2]} &= \mathit{diff}_{WK[j,\omega_2],WK[j,\omega_1]} \end{aligned}$$

$$\forall i_1, i_2 \in [0; N_r - 2], \forall j \in [0;3], \forall k_1, k_2 \in [0; N_b[\text{ where } (i_2, k_2) > (i_1, k_1) \tag{E2}$$

$$\begin{aligned} \mathit{diff}_{Y[i_1,j,k_1],Y[i_2,j,k_2]} + \Delta Y[i_1,j,k_1] + \Delta Y[i_2,j,k_2] &\neq 1 \\ \mathit{diff}_{Y[i_1,j,k_1],Y[i_2,j,k_2]} &= \mathit{diff}_{Y[i_2,j,k_2],Y[i_1,j,k_1]} \\ \mathit{diff}_{Z[i_1,j,k_1],Z[i_2,j,k_2]} + \Delta Z[i_1,j,k_1] + \Delta Z[i_2,j,k_2] &\neq 1 \\ \mathit{diff}_{Z[i_1,j,k_1],Z[i_2,j,k_2]} &= \mathit{diff}_{Z[i_2,j,k_2],Z[i_1,j,k_1]} \end{aligned}$$

$$\forall \omega_1, \omega_2, \omega_3 \in [0; N_b \times (N_r + 1)[, \forall j \in [0;3] \text{ where } \omega_3 > \omega_2 > \omega_1 \tag{E3}$$

$$\mathit{diff}_{WK[j,\omega_1],WK[j,\omega_2]} + \mathit{diff}_{WK[j,\omega_1],WK[j,\omega_3]} + \mathit{diff}_{WK[j,\omega_2],WK[j,\omega_3]} \neq 1$$

$$\forall i_1, i_2, i_3 \in [0; N_r - 2], \forall j \in [0;3], \forall k_1, k_2, k_3 \in [0; N_b[\text{ where } (i_3, k_3) > (i_2, k_2) > (i_1, k_1) \tag{E4}$$

$$\begin{aligned} \mathit{diff}_{Y[i_1,j,k_1],Y[i_2,j,k_2]} + \mathit{diff}_{Y[i_2,j,k_2],Y_{i_3}[j,k_3]} + \mathit{diff}_{Y_{i_3}[j,k_3],Y[i_1,j,k_1]} &\neq 1 \\ \mathit{diff}_{Z[i_1,j,k_1],Z[i_2,j,k_2]} + \mathit{diff}_{Z[i_2,j,k_2],Z_{i_3}[j,k_3]} + \mathit{diff}_{Z_{i_3}[j,k_3],Z[i_1,j,k_1]} &\neq 1 \end{aligned}$$

$$\forall i_1, i_2 \in [0; N_r[, \forall j \in [0;3], \forall k_1, k_2 \in [0; N_b[\tag{E5}$$

$$\mathit{diff}_{RK[i_1,j,k_1],RK[i_2,j,k_2]} = \mathit{diff}_{WK[j,(i_1+1)\times N_b+k],WK[j,(i_2+1)\times N_b+k]}$$

For each equation $\delta_{B1} \oplus \delta_{B2} \oplus \delta_{B3} = 0$ in ExtXOR, (E6)

$$\begin{aligned} \text{if } \{\Delta_{B1}, \Delta_{B2}, \Delta_{B3}\} \cap \Delta_{SB} \neq \emptyset \text{ then} \quad & \Delta_{B1} + \Delta_{B2} + \Delta_{B3} \neq 1 \\ \text{if } \{\Delta_{B1}, \Delta_{B2}\} \cap \Delta_{SB} = \emptyset \text{ then} \quad & \mathit{diff}_{B1,B2} = \Delta_{B3} \\ \text{if } \{\Delta_{B2}, \Delta_{B3}\} \cap \Delta_{SB} = \emptyset \text{ then} \quad & \mathit{diff}_{B2,B3} = \Delta_{B1} \\ \text{if } \{\Delta_{B1}, \Delta_{B3}\} \cap \Delta_{SB} = \emptyset \text{ then} \quad & \mathit{diff}_{B1,B3} = \Delta_{B2} \end{aligned}$$

For each equation $\delta_{B1} \oplus \delta_{B2} \oplus \delta_{B3} \oplus \delta_{B4} = 0$ in ExtXOR, (E7)

$$\begin{aligned} \text{if } \{\Delta_{B1}, \Delta_{B2}, \Delta_{B3}, \Delta_{B4}\} \cap \Delta_{SB} \neq \emptyset \text{ then} \quad & \Delta_{B1} + \Delta_{B2} + \Delta_{B3} + \Delta_{B4} \neq 1 \\ \text{else} \quad & \mathit{diff}_{B1,B2} = \mathit{diff}_{B3,B4} \\ & \mathit{diff}_{B1,B3} = \mathit{diff}_{B2,B4} \\ & \mathit{diff}_{B1,B4} = \mathit{diff}_{B2,B3} \end{aligned}$$

$$\forall i_1, i_2 \in [0; N_r - 2], \forall k_1, k_2 \in [0; N_b[\text{ where } (i_2, k_2) > (i_1, k_1) \tag{E8}$$

$$\sum_{j \in [0;3]} \mathit{diff}_{Y[i_1,j,k_1],Y[i_2,j,k_2]} + \sum_{j \in [0;3]} \mathit{diff}_{Z[i_1,j,k_1],Z[i_2,j,k_2]} \in \{0,5,6,7,8\}$$

$$\forall i_1, i_2 \in [0; N_r - 2], \forall j \in [0;3], \forall k_1, k_2 \in [0; N_b[\text{ where } (i_2, k_2) > (i_1, k_1) \tag{E9}$$

$$\mathit{diff}_{RK[i_1,j,k_1],RK[i_2,j,k_2]} + \mathit{diff}_{Z[i_1,j,k_1],Z[i_2,j,k_2]} + \Delta X[i_1+1,j,k_1] + \Delta X[i_2+1,j,k_2] \neq 1$$

Model 4.2: *Additional constraints for the refined Step-1 model for* Rijndael

$$obj = \sum_{\delta A \in \delta_{SB}} p_A \tag{C1}$$

$$\forall i \in [0; N_r[, \forall j \in [0;3], \forall k \in [0; N_b[, \tag{C2}$$

$$\langle \delta X[i,j,k], \delta SX[i,j,k], p_{X_i}[j,k] \rangle \in \mathtt{T_{SBOX}}$$

$$\forall i \in [0; N_r[, \forall j \in [0;3], \forall k \in [0; N_b[, \tag{C3}$$

$$\delta Y[i,j,k] = \delta SX[i,j,P_{N_b}[j] + k \mod N_b]$$

$$\forall i \in [0; N_r - 2], \forall k \in [0; N_b[, \forall j \in [0;3], \forall v \in \{2,3\} \tag{C4}$$

$$\langle \delta Y[i,j,k], v\delta Y[i,j,k] \rangle \in \mathtt{T_{MULv}}$$

$$\forall i \in [0; N_r - 2], \forall k \in [0; N_b[, \tag{C5}$$

$$\begin{array}{lll}
\langle 2\delta Y[i,0,k], 3\delta Y[i,1,k], a[i,k] \rangle \in \mathtt{T}_\oplus & \langle \delta Y[i,2,k], \delta Y[i,3,k], b[i,k] \rangle \in \mathtt{T}_\oplus & \langle a[i,k], b[i,k], \delta Z[i,0,k] \rangle \in \\
\langle \delta Y[i,0,k], 2\delta Y[i,1,k], c[i,k] \rangle \in \mathtt{T}_\oplus & \langle 3\delta Y[i,2,k], \delta Y[i,3,k], d[i,k] \rangle \in \mathtt{T}_\oplus & \langle c[i,k], d[i,k], \delta Z[i,1,k] \rangle \in \\
\langle \delta Y[i,0,k], \delta Y[i,1,k], e[i,k] \rangle \in \mathtt{T}_\oplus & \langle 2\delta Y[i,2,k], 3\delta Y[i,3,k], f[i,k] \rangle \in \mathtt{T}_\oplus & \langle e[i,k], f[i,k], \delta Z[i,2,k] \rangle \in \\
\langle 3\delta Y[i,0,k], \delta Y[i,1,k], g[i,k] \rangle \in \mathtt{T}_\oplus & \langle \delta Y[i,2,k], 2\delta Y[i,3,k], h[i,k] \rangle \in \mathtt{T}_\oplus & \langle g[i,k], h[i,k], \delta Z[i,3,k] \rangle \in
\end{array}$$

$$\forall i \in [0; N_r - 2], \forall k \in [0; N_b[, \forall j \in [0;3], \forall v \in \{9,11,13,14\} \tag{C6}$$

$$\langle \delta Z[i,j,k], v\delta Z[i,j,k] \rangle \in \mathtt{T_{MULv}}$$

$$\forall i \in [0; N_r - 2], \forall k \in [0; N_b[, \tag{C7}$$

$$\begin{array}{lll}
\langle 14\delta Z[i,0,k], 11\delta Z[i,1,k], m[i,k] \rangle \in \mathtt{T}_\oplus & \langle 13\delta Z[i,2,k], 9\delta Z[i,3,k], n[i,k] \rangle \in \mathtt{T}_\oplus & \langle m[i,k], n[i,k], \delta Y[i,0, \\
\langle 9\delta Z[i,0,k], 14\delta Z[i,1,k], o[i,k] \rangle \in \mathtt{T}_\oplus & \langle 11\delta Z[i,2,k], 13\delta Z[i,3,k], p[i,k] \rangle \in \mathtt{T}_\oplus & \langle o[i,k], p[i,k], \delta Y[i,1,k \\
\langle 13\delta Z[i,0,k], 9\delta Z[i,1,k], q[i,k] \rangle \in \mathtt{T}_\oplus & \langle 14\delta Z[i,2,k], 11\delta Z[i,3,k], r[i,k] \rangle \in \mathtt{T}_\oplus & \langle q[i,k], r[i,k], \delta Y[i,2,k \\
\langle 11\delta Z[i,0,k], 13\delta Z[i,1,k], s[i,k] \rangle \in \mathtt{T}_\oplus & \langle 9\delta Z[i,2,k], 14\delta Z[i,3,k], t[i,k] \rangle \in \mathtt{T}_\oplus & \langle s[i,k], t[i,k], \delta Y[i,3,k
\end{array}$$

$$\forall i \in [0; N_r - 2], \forall j \in [0;3], \forall k \in [0; N_b[, \tag{C8}$$

$$\langle \delta X_{i+1}[j,k], \delta Z[i,j,k], \delta WK[j, (i+1) \times N_b + k] \rangle \in \mathtt{T}_\oplus$$

$$\forall \omega \in [N_b; N_b \times (N_r + 1)[\text{ such that } \mathtt{isSbCol}(\omega), \forall j \in [0;3], \tag{C9}$$

$$\langle \delta WK[j,\omega], \delta SWK[j,\omega], p_{WK}[j,\omega] \rangle \in \mathtt{T_{SBOX}}$$

$$\forall j \in [0;3], \forall \omega \in [N_b; N_b \times (N_r + 1)[, \tag{C10}$$

$$\begin{array}{rl}
\text{if } \omega \mod N_k = 0 \text{ then} & \langle \delta WK[j,\omega], \delta WK[j,\omega - N_k], \delta SWK[(j+1) \mod 4, \omega - 1] \rangle \in \mathtt{T}_\oplus \\
\text{elsif } N_k > 6 \wedge k \mod N_k = 4 \text{ then} & \langle \delta WK[j,\omega], \delta WK[j,\omega - N_k], \delta SWK[j,\omega - 1] \rangle \in \mathtt{T}_\oplus \\
\text{else} & \langle \delta WK[j,\omega], \delta WK[j,\omega - N_k], \delta WK[j,\omega - 1] \rangle \in \mathtt{T}_\oplus
\end{array}$$

Model 4.3: *Step-2 model for* Rijndael.

at row j of M). Again, this is done by using table constraints. The main novelty with respect to the model introduced in [Gér+20] for the AES is that we introduce new variables (denoted $a_i[k]$, $b_i[k]$, etc), and we decompose the relation into three constraints such that each constraint ensures that the XOR of three variables is equal to zero. For example, the relation

$$\delta Z_i[0,k] \oplus 2\delta Y_i[0,k] \oplus 3\delta Y_i[1,k] \oplus \delta Y_i[2,k] \oplus \delta Y_i[3,k] = 0$$

is decomposed into the three following constraints:

$$\begin{aligned} \langle 2\delta Y_i[0,k], 3\delta Y_i[1,k], a_i[k] \rangle &\in \mathtt{T}_{\oplus} \\ \langle \delta Y_i[2,k], \delta Y_i[3,k], b_i[k] \rangle &\in \mathtt{T}_{\oplus} \\ \langle a_i[k], b_i[k], \delta Z_i[0,k] \rangle &\in \mathtt{T}_{\oplus} \end{aligned}$$

where $\mathtt{T}_{\oplus}$ is the table which contains every triple $(\delta A, \delta B, \delta C) \in [0;\ 255]^3$ such that $\delta A \oplus \delta B \oplus \delta C = 0$. This decomposition allows us to remove some variables and simplify constraints when we know that some variables are equal to 0. For example, if $\Delta Y_i[0,k] = 0$ in the truncated characteristic, then we infer that $2\delta Y_i[0,k] = 0$ (because $2 \otimes 0 = 0$) and $a_i[k] = 3\delta Y_i[1,k]$ (because $0 \oplus \delta Y_i[1,k] \oplus a_i[k] = 0$). Hence, in this case the three previous constraints are replaced with:

$$\langle \delta Y_i[2,k], \delta Y_i[3,k], b_i[k] \rangle \in \mathtt{T}_{\oplus} \text{ and } \langle 3\delta Y_i[1,k], b_i[k], \delta Z_i[0,k] \rangle \in \mathtt{T}_{\oplus}.$$

Constraints (C6) and (C7) are redundant constraints that model the $\texttt{MixColumns}^{-1}$ operation: They do not change the solutions, but they speed up the solution process by allowing the solver to propagate in both forward (from the plaintext to the ciphertext) and backward (from the ciphertext to the plaintext) directions.

4.2 Results

The Step-1 model is implemented with the MiniZinc 2.4.3 modelling language[2]. This language is accepted by many **CP** solvers and preliminary experiments have shown us that the best performing solver is Picat-SAT[3] 2.8.6: This solver first translates the MiniZinc model into a **SAT** instance and then uses the Lingeling **SAT** solver to solve the **SAT** instance. The Step-2 model is implemented and solved with the **CP** library Choco[4] v4.10.2.

In Figure 4.1, we compare solving times of the approach of [Gér+20] with those of our new approach on the AES instances in order to evaluate the interest of our two modifications, *i.e.*, (i) the interleaving of Steps 1 and 2 and the active use of LB and UB to stop the search whenever LB≥UB (see Section Two-step solving process), and (ii) the decomposition of the `MixColumns` constraint into 3 smaller table constraints (see Section `MixColumns`). For this experiment, all runs have been performed on a single core of an Intel Xeon CPU E3 at 3.50 GHz with 4 cores under

[2] https://github.com/MiniZinc/MiniZincIDE/releases/tag/2.4.3
[3] http://picat-lang.org/download/picat28_6_linux64.tar.gz
[4] https://github.com/chocoteam/choco-solver/releases/tag/4.10.2

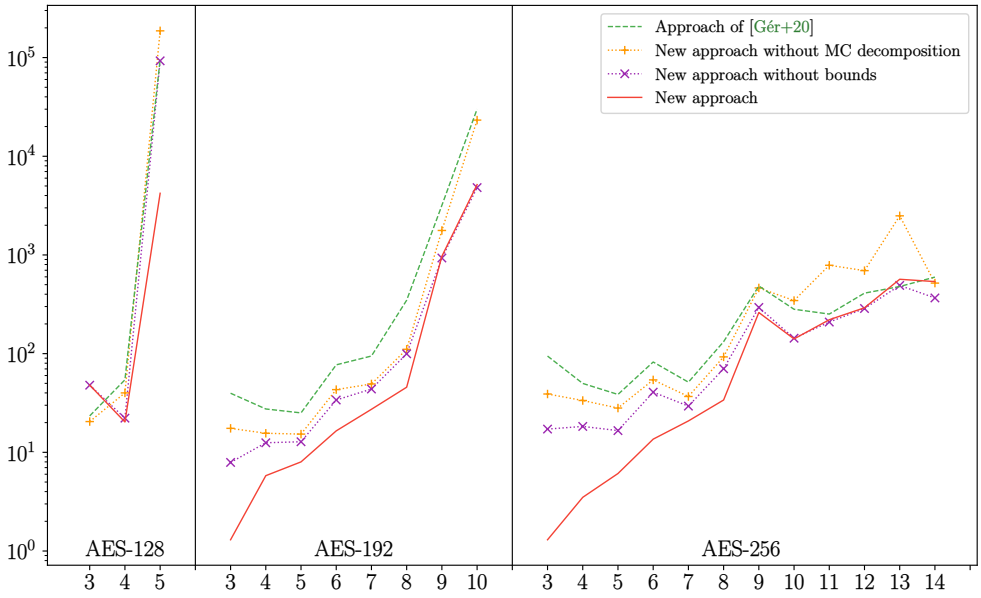


Figure 4.1: *Comparison of the approach of [Gér+20] (in green) with our new approach (in red), our new approach without* `MixColumns` *decomposition (in orange), and our new approach without exploiting bounds (in purple). Each point* (x, y) *corresponds to an* AES *instance (with* $K_{len} = 128$ *on the left,* $K_{len} = 192$ *on the middle, and* $K_{len} = 256$ *on the right):* y *gives the CPU time in seconds needed to solve it (logscale) when there are* $N_r = x$ *rounds.*

a Linux Ubuntu 20.04.1 (Focal Fossa) using at most 16 GB of RAM. There are two instances for which our new approach needs slightly more time than the approach of [Gér+20]: AES_{128} when $N_r = 3$ (48s instead of 23s) and AES_{256} when $N_r = 13$ (567s instead of 479s). For the 21 remaining instances our new approach is faster and, in some cases the difference is very large, *e.g.*, $4,217$s instead of $95,389$s for AES_{128} when $N_r = 5$, or $5,163$s instead of $30,059$s for AES_{192} when $N_r = 10$. To evaluate the interest of each of our two modifications separately, we also display our new approach without (ii), and our new approach without (i). In many cases, each modification improves the solution process, and the combination of these two modifications is even better. However, modification (i) deteriorates the solution process when $N_r \geq 10$ for AES_{256}. This comes from the fact that, for these instances, the optimal solution is strictly smaller than $2^{-6 \cdot NB_{\mathrm{SBOX}}}$ so that the lower bound LB cannot be used to stop the search.

We give in Tables 4.1 and 4.2 the results of Algorithm 4.1 for every key length $K_{len} \in \{128, 160, 192, 224, 256\}$, every block size $C_{len} \in \{128, 160, 192, 224, 256\}$, and every number of rounds $N_r \in [3; x]$ where x is the maximum number of rounds authorized (*i.e.*, the maximal number of rounds for which $NB_{\mathtt{SBOX}}$ is smaller than the key length divided by 6 and the number of active S-Boxes in the plaintext part is smaller than the block length divided by 6). For this experiment, all runs have been performed on a single core of an Intel Xeon E5-2630 v4 at 3.10 Ghz with 10 cores under a Linux Debian 10 (Buster) using at most 16 GB of RAM (default JVM configuration). This architecture was provided by the Grid5000 cluster [Bal+13].

Please note that there is a slight difference between the model used for Figure 4.1 and the model used for Tables 4.1 and 4.2. Indeed, the model in [Gér+20] ignores the S-Boxes of the last round subkey. When the key has 128 or 256 bits, this does not change anything. However, when the key has 192 bits this may change results. To allow a fair comparison with [Gér+20], we also ignore the S-Boxes of the last round key in all models compared in Figure 4.1. However, in Tables 4.1 and 4.2, we do consider the S-Boxes of the last round subkey. Therefore, when the key has 192 bits and the text 128 bits, some probabilities may be greater than those reported in [Gér+20], *e.g.*, for the instance $\mathsf{Rijndael}_{128-192}$ with 7 rounds the maximal probability is 2^{-84} instead of 2^{-78}.

One instance (when $C_{len} = 128$, $K_{len} = 160$, and $N_r = 8$) was stopped after 38 days of computation. For this instance, the output value of *Step1-opt* is 23. *Step1-enum* has enumerated 7 truncated characteristics with 23 active S-Boxes and none of them can be instantiated into a Step-2 characteristic. So far, we have enumerated 213 truncated characteristic with 24 active S-Boxes and none of them can be instantiated into a Step-2 characteristic. Hence, for this instance the current upper bound is $UB = 2^{-150}$. We have computed 189 instances with 25 active S-Boxes and 1048 instances with 26 active S-Boxes and the smaller probability is $LB = 2^{-160}$.

All other instances have been solved within a reasonable amount of time: 82 are solved within 1,000s; 24 need more than 1,000s and less than 10,000s (*i.e.*, less than three hours); 10 need more than 10,000s and less than 100,000s (*i.e.*, less than 28 hours); and finally 2 need more than 28 hours and less than 3 days.

In Tables 4.1 and 4.2, o_1 is the output value of *Step1-opt* (called at line 1 of Algorithm 4.1), i.e., the initial value of $NB_{\mathtt{SB}}$; p is the output value of Algorithm 4.1, i.e., the probability of the optimal related-key differential characteristic; and time is the total CPU time spent by Algorithm 4.1 in seconds (this time both includes the running times of Picat-SAT and of Choco). We also report the number o_2 of active S-Boxes in the optimal differential characteristic. In most cases (91 out of 122 cases), $o_1 = o_2$ and $p > 2^{-6\cdot(o_1+1)}$. In these cases, Algorithm 1 has enumerated Step-1 solutions for only one value of $NB_{\mathtt{SB}}$, and LB became larger than or equal to UB the first time $NB_{\mathtt{SB}}$ has been incremented.

In 17 cases (marked with ^{c} just after o_2), $o_1 = o_2$ but it has been necessary to increment $NB_{\mathtt{SB}}$ at least once in order to check that no better characteristic can be found with more active S-Boxes. For example, when $C_{len} = 128$, $K_{len} = 224$, and $r = 9$, the best differential characteristic has 22 active S-Boxes and its probability is 2^{-139}. As $2^{-139} < 2^{-6\cdot 23}$, Algorithm 1 has incremented $NB_{\mathtt{SB}}$ in order to check that it is not possible to have a larger probability (equal to 2^{-139}) with 23 active S-Boxes.

In 2 cases (marked with ! just after o_2), $o_2 \geq o_1 + 1$ because none of the Step-1 truncated characteristic with o_1 active S-Boxes can be instantiated into a Step-2 characteristic. In these two cases, Algorithm 1 has incremented $NB_{\mathtt{SB}}$ in order to enumerate Step-1 solutions with $o_1 + 1$ active S-Boxes and find the best differential characteristic.

Finally, in 13 cases (marked with ↑ just after o_2), $o_2 \geq o_1 + 1$ because a better characteristic has been found with $o_1 + n$ active S-Boxes (though at least one Step-1 solution can be instantiated into a Step-2 solution).

4.3 Attacks

We describe in this section the best attacks we could mount based on the distinguishers found in the previous section. More precisely, two particular distinguishers have a real interest in terms of attacks. The first one is an 11-round related-key differential characteristic distinguisher on $\mathsf{Rijndael}_{128-224}$ (presented in Table 4.3 that allows us to mount an attack on 12 rounds (out of 13) of this cipher. There also exists a 12-round distinguisher for Rijndael-128-224 but due to its very low probability (equal to 2^{-127}) for the data path, we do not manage to transform this distinguisher into an attack. And second, we also mount an attack on 12 rounds of $\mathsf{Rijndael}_{160-256}$ (it has 14 rounds) based on the 11-round related-key differential characteristic distinguisher (presented in Table 4.4).

4.3.1 Weak key attack on 12 rounds of Rijndael-128-224

First, remember that the 12th round of Rijndael-128-224 is the last round for our attack so it does not contain a `MixColumns` operation. We base our attack on the distinguisher presented in Table 4.3. This distinguisher has a probability equal to 2^{-169}: 2^{-103} coming from the state and 2^{-66} coming from the key.

Thus, the attack process is the following one. We submit $M = 2^{103+\epsilon}$ pairs of plaintexts X and X' with the difference specified in the first line of Table 4.3 under the keys K and $K' = K \oplus \delta K$ with the difference specified in the first line (second column) of Table 4.3. Then a possible propagation of the difference is the one shown in Table 4.3, and we obtain the corresponding ciphertexts C and C'.

We know from Table 4.3 that the output of the 11th round (and the beginning of the 12th round) is of the form $\delta X_{12} = \begin{pmatrix} 01 & 01 & 1F & 1F \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$. After passing through `SubBytes` and `ShiftRows`, it becomes: $\delta SX_{12} = \begin{pmatrix} ? & ? & ? & ? \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$. From the keyschedule, the subkey difference δK_{12} will be of the form $\begin{pmatrix} 21 & A \oplus 01 & A & A \oplus 01 \\ 1F & B & B & B \\ 1F & C & C & C \\ 21 & D & D & D \end{pmatrix}$ where A, B, C and D are unknown difference. Thus the

Results when $C_{len} = 128$ and $K_{len} \in \{128, 160, 192\}$

	$K_{len} = 128$				$K_{len} = 160$				$K_{len} = 192$			
	o_1	o_2	p	time	o_1	o_2	p	time	o_1	o_2	p	time
$Nr = 3$	5	5	2^{-31}	13	4	4	2^{-24}	5	1	1	2^{-6}	1
$Nr = 4$	12	12	2^{-75}	31	5	5	2^{-30}	21	4	4	2^{-24}	6
$Nr = 5$	17	17	2^{-105}	8,304	10	10	2^{-60}	12	5	5	2^{-30}	8
$Nr = 6$					17	17	2^{-108}	641	10	10	2^{-60}	17
$Nr = 7$					19	19	2^{-120}	1,089	14	14	2^{-84}	46
$Nr = 8$					23	≥ 24!	$2^{-160} \leq p \leq 2^{-150}$	$> 10^6$	18	18	2^{-108}	83
$Nr = 9$									24	24	2^{-146}	1,800

Results when $C_{len} = 128$ and $K_{len} \in \{224, 256\}$

	$K_{len} = 224$				$K_{len} = 256$			
	o_1	o_2	p	time	o_1	o_2	p	time
$Nr = 3$	1	1	2^{-6}	1	1	1	2^{-6}	1
$Nr = 4$	3	3	2^{-18}	3	3	3	2^{-18}	3
$Nr = 5$	6	6	2^{-36}	8	3	3	2^{-18}	5
$Nr = 6$	8	8	2^{-48}	14	5	5	2^{-30}	13
$Nr = 7$	13	13	2^{-78}	35	5	5	2^{-30}	18
$Nr = 8$	18	18	2^{-112}	1,593	10	10	2^{-60}	32
$Nr = 9$	22	22^c	2^{-139}	2,425	15	15	2^{-92}	346
$Nr = 10$	24	24^c	2^{-151}	1,834	16	16	2^{-98}	159
$Nr = 11$	27	27^c	2^{-169}	1,823	20	20	2^{-122}	330
$Nr = 12$	34	34^c	2^{-212}	9,561	20	20	2^{-122}	277
$Nr = 13$					24	24	2^{-146}	420
$Nr = 14$					24	24	2^{-146}	557

Results when $C_{len} = 160$ and $K_{len} \in \{128, 160, 192\}$

	$K_{len} = 128$				$K_{len} = 160$				$K_{len} = 192$			
	o_1	o_2	p	time	o_1	o_2	p	time	o_1	o_2	p	time
$Nr = 3$	9	9	2^{-54}	6	5	5	2^{-30}	880	4	4	2^{-24}	4
$Nr = 4$	18	18	2^{-112}	49,501	10	10	2^{-60}	11	6	6	2^{-36}	7
$Nr = 5$					17	17	2^{-107}	621	9	9	2^{-54}	15
$Nr = 6$					21	22!	2^{-138}	36,788	15	15	2^{-90}	62
$Nr = 7$									19	19	2^{-117}	600
$Nr = 8$									23	23	2^{-141}	2,059

Results when $C_{len} = 160$ and $K_{len} \in \{224, 256\}$

	$K_{len} = 224$				$K_{len} = 256$			
	o_1	o_2	p	time	o_1	o_2	p	time
$Nr = 3$	2	2	2^{-12}	2	1	1	2^{-6}	2
$Nr = 4$	5	5	2^{-31}	16	4	4	2^{-24}	4
$Nr = 5$	10	10	2^{-60}	18	6	6	2^{-36}	14
$Nr = 6$	15	15	2^{-90}	40	12	12	2^{-72}	42
$Nr = 7$	20	20	2^{-124}	402	15	15	2^{-93}	226
$Nr = 8$	24	24	2^{-148}	783	20	20	2^{-124}	755
$Nr = 9$	30	30^c	2^{-190}	13,081	23	23^c	2^{-146}	2,284
$Nr = 10$					27	27^c	2^{-169}	4,927
$Nr = 11$					32	32^c	2^{-204}	15,497

Table 4.1: *Summary of the best related-key differential characteristics for* Rijndael *when* $C_{len} \in \{128, 160\}$. *The time is given in seconds.*

Results when $C_{len} = 192$ and $K_{len} \in \{128, 160, 192\}$

	$K_{len} = 128$				$K_{len} = 160$				$K_{len} = 192$			
	o_1	o_2	p	time	o_1	o_2	p	time	o_1	o_2	p	time
$Nr = 3$	9	9	2^{-54}	7	6	6	2^{-37}	20	5	5	2^{-30}	199
$Nr = 4$					15	15	2^{-94}	92	9	9	2^{-54}	15
$Nr = 5$					19	19	2^{-118}	183	14	$15\uparrow$	2^{-90}	146
$Nr = 6$									19	19	2^{-117}	864
$Nr = 7$									25	25	2^{-153}	2,101

Results when $C_{len} = 192$ and $K_{len} \in \{224, 256\}$

	$K_{len} = 224$				$K_{len} = 256$			
	o_1	o_2	p	time	o_1	o_2	p	time
$Nr = 3$	4	4	2^{-24}	7	1	1	2^{-6}	2
$Nr = 4$	8	8	2^{-48}	13	5	5	2^{-30}	10
$Nr = 5$	15	15	2^{-95}	387	12	12	2^{-72}	84
$Nr = 6$	16	$17\uparrow$	2^{-103}	1,349	17	17	2^{-106}	452
$Nr = 7$	24	24^c	2^{-157}	11,908	18	18	2^{-112}	551
$Nr = 8$	32	$33\uparrow^c$	2^{-205}	91,983	24	24	2^{-149}	951
$Nr = 9$					29	29	2^{-179}	3,397
$Nr = 10$					38	38^c	2^{-236}	88,076

Results when $C_{len} = 224$ and $K_{len} \in \{128, 160, 192\}$

	$K_{len} = 128$				$K_{len} = 160$				$K_{len} = 192$			
	o_1	o_2	p	time	o_1	o_2	p	time	o_1	o_2	p	time
$Nr = 3$	9	9	2^{-54}	13	9	9	2^{-54}	9	6	6	2^{-37}	39
$Nr = 4$					19	19^c	2^{-122}	2,742	13	13	2^{-78}	35
$Nr = 5$									20	20	2^{-124}	1,040
$Nr = 6$									28	$29\uparrow$	2^{-179}	18,632

Results when $C_{len} = 224$ and $K_{len} \in \{224, 256\}$

	$K_{len} = 224$				$K_{len} = 256$			
	o_1	o_2	p	time	o_1	o_2	p	time
$Nr = 3$	6	6	2^{-36}	8	4	4	2^{-24}	10
$Nr = 4$	13	13	2^{-79}	121	8	8	2^{-48}	22
$Nr = 5$	16	$17\uparrow$	2^{-103}	1,562	15	$16\uparrow$	2^{-97}	3,267
$Nr = 6$	23	23^c	2^{-150}	1,511	18	$19\uparrow$	2^{-115}	5,049
$Nr = 7$	31	31^c	2^{-196}	49,429	20	$21\uparrow$	2^{-128}	1,378
$Nr = 8$					28	$30\uparrow$	2^{-182}	18,377
$Nr = 9$					37	37^c	2^{-241}	210,290

Results when $C_{len} = 256$ and $K_{len} \in \{128, 160, 192\}$

	$K_{len} = 128$				$K_{len} = 160$				$K_{len} = 192$			
	o_1	o_2	p	time	o_1	o_2	p	time	o_1	o_2	p	time
$Nr = 3$	9	9	2^{-54}	15	9	9	2^{-54}	13	9	9	2^{-54}	12
$Nr = 4$					20	$21\uparrow$	2^{-130}	4,157	18	18	2^{-110}	824
$Nr = 5$									24	24	2^{-148}	4,624

Results when $C_{len} = 256$ and $K_{len} \in \{224, 256\}$

	$K_{len} = 224$				$K_{len} = 256$			
	o_1	o_2	p	time	o_1	o_2	p	time
$Nr = 3$	6	6	2^{-37}	33	5	5	2^{-30}	34
$Nr = 4$	18	18^c	2^{-115}	65,672	13	13	2^{-79}	276
$Nr = 5$	28	29!	2^{-179}	455,210	16	$17\uparrow$	2^{-103}	3,084
$Nr = 6$					20	$21\uparrow$	2^{-128}	2,170
$Nr = 7$					27	$29\uparrow$	2^{-176}	9,237
$Nr = 8$					37	37^c	2^{-240}	191,581

Table 4.2: *Summary of the best related-key differential characteristics for* Rijndael *when* $C_{len} \in \{192, 224, 256\}$. *The time is given in seconds.*

Round	$\delta X_i = X_i \oplus X'_i$ (before SBOX) δSBX_i (after SBOX)	δRK_i	Pr(States)	Pr(Key)
$i = 0$	005D005D 00A300A3 00A300A3 00FE00FE	015C005D 00A300A3 00A300A3 00FE00FE	–	–
$i = 1$	01010000 00000000 00000000 00000000 1F1F0000 00000000 00000000 00000000	21210001 1F1F0000 1F1F0000 21210000	$2^{-2\times 6}$	–
2	1F1F0001 00000000 00000000 00000000 A3A3001F 00000000 00000000 00000000	5D5D0021 A3A3001F A3A3001F FEFE0021	$2^{-3\times 6}$	–
3	0000001F 00000000 00000000 00000000 000000A3 00000000 00000000 00000000	0000015C 000000A3 000000A3 000000FE	2^{-6}	–
4	00001F1F 00000000 00000000 00000000 00001F1F 00000000 00000000 00000000	01013E3E 00001F1F 00001F1F 00002121	$2^{2\times(-6)}$	2^{-6}
5	01010000 00000000 00000000 00000000 1FA30000 00000000 00000000 00000000	3E5C0001 1FA30000 1FA30000 21FE0000	2^{-6-7}	$2^{-6-3\times 7}$
6	00010001 00000000 00000000 00000000 001F001F 00000000 00000000 00000000	003E003E 001F001F 001F001F 00210021	$2^{2\times(-6)}$	$2^{-6-3\times 7}$
7	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	01010000 00000000 00000000 00000000	–	–
8	01010000 00000000 00000000 00000000 1F1F0000 00000000 00000000 00000000	3E3E0001 1F1F0000 1F1F0000 21210000	$2^{2\times(-6)}$	–
9	00000001 00000000 00000000 00000000 0000001F 00000000 00000000 00000000	0000003E 0000001F 0000001F 00000021	2^{-6}	–
10	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	00000101 00000000 00000000 00000000	–	–
11	00000101 00000000 00000000 00000000 00001F1F 00000000 00000000 00000000	01012121 00001F1F 00001F1F 00002121	$2^{2\times(-6)}$	2^{-6}
output	01013E3E 00001F1F 00001F1F 00002121			

Table 4.3: *The Best related key differential characteristic we found on 11 rounds of* $\mathsf{Rijndael}_{128-224}$ *with probability equal to* 2^{-169}*. The four words represent the four rows of the state and are given in hexadecimal notation. Note that the last round does not contain the* `MixColumns` *operation.*

difference between C and C' will be of the form $\delta C = \begin{pmatrix} ? & ? & ? & ? \\ 1F & B & B & B \\ 1F & C & C & C \\ 21 & D & D & D \end{pmatrix}$.

So the attack works as follows:

- We filter on the values $1F$, $1F$, and 21 at positions $(1,0)$, $(2,0)$, and $(3,0)$ in δC before the last `ShiftRows`. It remains $2^{103+\epsilon-20} = 2^{83+\epsilon}$ pairs of plaintexts/ciphertexts. Moreover, we know that the three bytes at positions $(1,1), (1,2)$ and $(1,3)$ must be equal (this remark also holds for the second and the third rows). This leads to another filter of 48 bits.
- We guess the byte value of K_{12} at position $(0,0)$ with a cost of 2^8. Then, we decipher this byte from C and C' to check if it is equal to 01 at the input of the 12th round. Then, it filters 2^{-8} values. Moreover, the known byte at position $(0,0)$ in K_{12} gives us the difference D (due to the keyschedule construction).
- We can guess the byte at position $(1,0)$ in K_{12} and check the value at the input of the 12th round at position $(1,0)$, by deciphering from C and C'. Then, it filters 2^{-8} values. Moreover, we can compute the difference A.
- We can guess the three bytes at positions $(0,1)$, $(0,2)$, and $(0,3)$ in K_{12} and check the value at the input of the 12th round at the same position knowing the difference A, by deciphering from C and C'. Then, it filters 2^{-24} values.

- We can guess the byte at position $(3,0)$ in K_{12} and check the value at the input of the 12th round at position $(3,0)$, by deciphering from C and C'. Then, it filters 2^{-8} values.
- We can guess the byte at position $(2,0)$ in K_{12} and check the value at the input of the 12th round at position $(2,0)$, by deciphering from C and C'. Then, it filters 2^{-8} values.

Then, we have guessed 7 bytes of the subkey K_{12}, 56 bits of key, and we have filtered an equivalent of 68 bits, leading to keep $2^{103+\epsilon-68} = 2^{35+\epsilon}$ pairs of plaintexts/ciphertexts. After guessing the 7-byte difference in the subkey K_{12}, $\delta_{K_{12}}$ is fully determined. Thus, guessing the bytes of one key state could determine the bytes of the related-key state.

The related-key differential characteristic given in Table 4.3 has a probability to happen for the state part equals to 2^{-103}. Thus, if we use 2^{104} plaintexts/ciphertexts in the related-key differential attack on 12 rounds of Rijndael$_{128-224}$, then the right difference of the 56 bits of the last subkey will be counted at least twice on average whereas the probability that a bad key appears twice is really low (around $2^{32-68} = 2^{-34}$). The success probability computed using the results of [Sel08] is around 97%. The time complexity of the attack is about 2^{105} encryptions and the attack succeeds if the key follows the characteristic described in Table 4.3. In other words, we have a set of weak keys of size $2^{224-66} = 2^{158}$.

The 168 remaining bits of the master key can be guessed through guessing more bytes in K_{11} and in K_{12} and filtering according to the remaining values in δX_{11} and the S-Boxes of the `KeySchedule`.

4.3.2 Weak key attack on 12 rounds of Rijndael-160-256

Round	$\delta X_i = X_i \oplus X'_i$ (before `SBOX`) δSBX_i (after `SBOX`)	δRK_i	Pr(States)	Pr(Key)
$i=0$	E094E0E082 7000700041 1400700041 701F9000C3	E000E0E000 7000700041 7000700041 70909000C3	–	–
$i=1$	0094000082 0000000000 6400000000 008F000000 0041000070 0000000000 2000000000 0010000000	008282E0E0 0041007070 0041007070 00C3000090	$2^{4\times(-6)}$	2^{-6}
2	000082D000 0000000000 0000000000 0000000000 0000414100 0000000000 0000000000 0000000000	00E0828200 0000414100 0000414100 0000C3C300	$2^{2\times(-7)}$	2^{-6}
3	00E0000000 0000000000 0000000000 0000000000 0070000000 0000000000 0000000000 0000000000	82E00000E0 0070000000 0070000000 0090000000	2^{-7}	2^{-6-7}
4	82000000E0 0000000000 0000000000 0000000000 4100000070 0000000000 0000000000 0000000000	00828200E0 4100000070 4100000070 C300000090	$2^{2\times(-7)}$	–
5	8282820000 0000000000 0000000000 0000000000 7070700000 0000000000 0000000000 0000000000	E0E0000082 7070700000 7070700000 9090900000	$2^{3\times(-6)}$	$2^{3\times(-7)}$
6	0000E00082 0000000000 0000000000 0000000000 0000700070 0000000000 0000000000 0000000000	0000E000E0 0000700070 0000700070 0000900090	2^{-6-7}	–
7	0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000	E082000000 0000000000 0000000000 0000000000	–	2^{-6}
8	E082000000 0000000000 0000000000 0000000000 7070000000 0000000000 0000000000 0000000000	E0E000E000 7070000000 7070000000 9090000000	2^{-6-7}	2^{-6}
9	000000E000 0000000000 0000000000 0000000000 0000007000 0000000000 0000000000 0000000000	000000E000 0000007000 0000007000 0000009000	2^{-7}	–
10	0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000	00E0828282 0000000000 0000000000 0000000000	–	2^{-6}
11	00E0828282 0000000000 0000000000 0000000000 0082707070 0000000000 0000000000 0000000000	82E0E0E000 0070707070 0070707070 00E0E0E0E0	$2^{4\times(-6)}$	2^{-6}
Output	?????????? 00F20000?? 00F20000?? 000D0000??			

Table 4.4: *The Best related key differential characteristic we found on 11 rounds of* Rijndael$_{160-256}$ *with probability equal to* 2^{-204}*. The four words represent the four rows of the state and are given in hexadecimal notation. Note that the last round does not contain the* `MixColumns` *operation.*

In the same way, from the related-key differential characteristic distinguisher on 11-round of $\mathsf{Rijndael}_{160-256}$ presented in Table 4.4, we can easily mount a 12-round attack against $\mathsf{Rijndael}_{160-256}$ that has 14 rounds. Note that the 12th round does not contain the `MixColumns` operation as it is the last round of our attack. The 11-round related-key differential characteristic distinguisher presented in Table 4.4 has a probability equal to 2^{-204}: 2^{-128} coming from the difference in the state and 2^{-76} coming from the key.

Thus, the attack process is the following one. We submit $M = 2^{128+\epsilon}$ pairs of plaintexts X and X' with the difference specified in the first line of Table 4.4 under the keys K and $K' = K \oplus \delta K$ with the difference specified in the first line (second column) of Table 4.4. Then a possible propagation of the difference is the one shown in Table 4.4. Then, we obtain the corresponding ciphertexts C and C'.

Then, we know from Table 4.4 that the output of the 11th round (and the beginning of the 12th round) is of the form $\delta X_{12} = \begin{pmatrix} 82 & FF & 0 & 0 & E0 \\ 0 & F2 & 0 & 0 & 0 \\ 0 & F2 & 0 & 0 & 0 \\ 0 & ED & 70 & 70 & 70 \end{pmatrix}$. After passing through `SubBytes` and `ShiftRows`, it becomes: $\delta SX_{12} = \begin{pmatrix} ? & ? & 0 & 0 & ? \\ ? & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ? \\ ? & ? & 0 & ? & ? \end{pmatrix}$. From the keyschedule, the subkey differences δK_{12} will be of the form $\begin{pmatrix} 82 & 0 & 82 & 0 & F \\ A & A & A & A & D \\ B & B & B & B & E \\ C & C & C & C & E0 \end{pmatrix}$ where A, B, C, D, E and F are unknown difference. Thus the difference between C and C' will be of the form $\delta C = \begin{pmatrix} ? & ? & 82 & 0 & ? \\ ? & A & A & A & D \\ B & B & B & B & ? \\ ? & ? & C & ? & ? \end{pmatrix}$.

So the attack works as follows:

- For all the $2^{128+\epsilon}$ encrypted pairs of plaintexts/ciphertexts, we filter on the values 82 and 0 at positions $(0,2)$ and $(0,3)$ in δC. This filters 2^{-16} values. Then, it remains $2^{112+\epsilon}$ encrypted pairs of plaintexts/ciphertexts.
- We guess the three bytes at positions $(1,4)$, $(2,4)$, and $(3,4)$ in K_{11} for a cost of 2^{24}. This gives us the values of differences A, B and C. With those known values, we filter on δSX_{12} on the 8 positions that are equal to 0 after removing A, B, and C (positions $(1,1), (1,2), (1,3), (2,0), (2,1), (2,2), (2,3)$ and $(3,2)$). This filters 2^{-64} values.
- We guess 6 bytes of K_{12} (those at positions $(0,0), (0,1), (1,0), (3,0), (3,1)$ and $(3,3)$). We filter the corresponding 2^{-48} values on δX_{12} (before the S-Boxes) at the same positions.
- We guess the byte at position $(2,3)$ in K_{12} to get one new byte in δK_{12} at position $(1,4)$

equal to D and check if the difference is equal to 0 at position $(1,4)$ in δX_{12}. It filters 2^{-8} values.

- We guess the byte at position $(3,4)$ in K_{12} to filter one byte of value in δX_{12} at position $(3,4)$. We guess another byte at position $(2,4)$ in K_{12} to filter the byte value at position $(2,4)$ in δX_{12}. And finally, we guess the two bytes at positions $(1,3)$ and $(0,4)$ in K_{12} to filter the byte value at position $(0,4)$ in δX_{12}.

We have guessed a total of 112 key bits and we have filtered, in the initial step 16 bits and the equivalent of 32 bits in the second step and the last step leading to stay with $2^{80+\epsilon}$ pairs of plaintexts/ciphertexts.

The related-key differential characteristic given in Table 4.4 has a probability to happen for the state part equals to 2^{-128}. Thus, if we use 2^{129} plaintexts/ciphertexts in the related-key differential attack on 12 rounds of $\mathsf{Rijndael}_{160-256}$, then the right difference of the 112 bits of the last subkey will be counted at least twice on average whereas the probability that a bad key appears twice is really low (around $2^{81-112} = 2^{-31}$). The success probability computed using the results of [Sel08] is around 97% also. The time complexity of the attack is about 2^{130} encryptions and the attack succeeds if the key follows the characteristic described in Table 4.4. In other words, we have a set of weak keys of size $2^{256-76} = 2^{180}$.

The 144 remaining bits of the master key can be guessed through guessing more bytes in K_{11} and in K_{12} and filtering according the remaining values in δX_{11} and the S-Boxes of the key schedule.

4.4 Conclusion

In this chapter, we have extended and improved the models initially proposed in [Gér+20] for the AES to the 25 instances of Rijndael. This allowed us to compute optimal related-key differential characteristics for all Rijndael instances but one (and provide upper and lower bounds for the remaining one). We sum up in Table 4.5 the best attacks described in this chapter.

Instance	Nb rounds	N_r	Time	Number of keys
$\mathsf{Rijndael}_{128-224}$	12	13	2^{105}	2^{158}
$\mathsf{Rijndael}_{160-256}$	12	14	2^{130}	2^{180}

Table 4.5: *Summary of the best related-key differential attacks we found on different* Rijndael *instances. The last column displays the number of keys for which the attack works.*

Those results are obtained using a two-step strategy that is feasible in terms of memory usage and time consumption. This strategy is modelled in MiniZinc, and it is solved by combining two solvers: Picat-SAT for Step-1 and Choco for Step-2.

As the reader can see, abstracting the ciphers to model Step-1 can be complicated. Moreover, using multiple solvers can be a source of errors, *e.g.* when transforming a solution from one solver to the model of another solver. Although generic solvers are like black boxes, it is necessary to know how they work in order to fully exploit their power. Indeed, a model could very well be

efficient on a **SAT** solver whereas it would not be efficient on **ILP** or **CP** solvers.

In the next chapter, we propose a new constraint named *abstractXOR* dedicated to the abstraction of the XOR operator in Step-1 in order to simplify the work of cryptographers; this allows them (i) to use only one **CP** solver, (ii) to transcribe more directly the cipher in the model without going through intermediate variables or new equations.

Chapter

5 Abstract XOR

Contents

5.1 Motivations

To motivate the introduction of a global constraint for propagating abstract XORs during Step-1, let us consider the **CP** model for computing a maximum differential characteristic (**MDC**) on Midori, displayed in Model 5.1. This model is derived in a straightforward way from the definition of the cipcher algorithm, which is detailed in 1.2.2. The goal is to maximise the sum of all $\log_2$ probabilities $P[i,k]$. Constraints (C1) to (C4) correspond to the 4 operations applied at each round:

- (C1) is the table constraint corresponding to `SubBytes`;
- (C2) corresponds to `ShiftRow`, which moves bytes from position b in $\delta SX[i]$ to position $f(b)$ in $\delta \mathrm{Y}[i]$;
- (C3) and (C4) correspond to `MixColumns` and `AddRoundKey`, respectively, and only involve XOR operations.

$$Maximise \sum_{i=0}^{r-1} \sum_{k=0}^{15} P[i,k] \tag{C0}$$

$\forall i \in [0;\ r[, \forall k \in [0;\ 15]$:

$$(\delta_X[i,k], \delta_{SX}[i,k], P[i,k]) \in subBytesTable_k \tag{C1}$$

$$\delta_Y[i, f(k)] = \delta_{SX}[i,k] \tag{C2}$$

$$\delta_Z[i,k] \oplus \delta_Y[i,(k+4)\%16] \oplus \delta_Y[i,(k+8)\%16] \oplus \delta_Y[i,(k+12)\%16] = 0 \tag{C3}$$

$$\delta_Z[i,k] \oplus \delta_K[k] \oplus \delta_X[i+1,k] = 0 \tag{C4}$$

Model 5.1: *MDC problem for* Midori_{128}.

When considering a two-step solving process, a basic Step-1 model is displayed in Model 5.2. It is very similar to the problem of Model 5.1. The main difference is that $\log_2$ probability variables ($P[i,k]$) are removed, and the objective function and constraint C1 are replaced with constraint (A0) which ensures that the number of $\Delta[X_i][b]$ variables assigned to 1 is equal to a given value n. Constraint A1 comes from the fact that $\delta_X[i,k] = 0$ iff $\delta_{SX}[i,k] = 0$. Finally, XOR constraints C3 and C4 are replaced with abstract XOR constraints A3 and A4: an abstract XOR constraint $\Delta_1 \circ \ldots \circ \Delta_l = 0$ is satisfied iff, for each Δ_j assigned to 1 there exists an integer value in $[1;\ 255]$ such that the XOR of all these values is equal to 0.

When encoding each abstract XOR constraint $\Delta_1 \circ \Delta_l = 0$ by $sum_{i=1}^{l} \Delta_i \neq 1$, the resulting model has a lot of solutions ant most of these solutions cannot be instantiated into Step-2 solutions. In his PhD thesis [Gér18], Gerault introduced a new model that has far fewer solutions. This model reuses the same techniques as those presented in Chapter 3.1 and is therefore more complicated than the basic one.

Our goal is to ease the design of **CP** models for computing **MDC**s, while ensuring an efficient solving process. To this aim, we introduce a global constraint which propagates XORs in a global way in order to reduce the number of Step2-inconsistent solutions.

In Section 5.2, we introduce notations and preliminary definitions.

In Section 5.3, we introduce the *abstractXOR* constraint which ensures that a set of abstract XOR constraints is Step2-consistent. We show that deciding of *abstractXOR* feasibility is $\mathcal{NP}$-complete when differential variables are constrained to belong to $[0;\ 255]$ whereas it is polynomial when the domain of differential variables is not upper bounded. Hence, we relax *abstractXOR* by removing this upper bound. In Section 5.4, we introduce two propagators for *abstractXOR*: the first one simply ensures feasibility, and the second one ensures Generalised Arc Consistency (**GAC**).

$$n = \sum_{i=0}^{r-1} \sum_{k=0}^{15} \Delta_X[i,k] \tag{A0}$$

$\forall i \in [0;\ r[, \forall k \in [0;\ 15]$:

$$\Delta_X[i,k] = \Delta_{SX}[i,k] \tag{A1}$$

$$\Delta_Y[i,k] = \Delta_{SX}[i, f(k)] \tag{A2}$$

$$\Delta_Z[i,k] \circ \Delta_Y[i,(k+4)\%16] \circ \Delta_Y[i,(k+8)\%16] \circ \Delta_Y[i,(k+12)\%16] = 0 \tag{A3}$$

$$\Delta_\mathsf{Z}[i,k] \circ \Delta_K[k] \circ \Delta_X[i+1,k] = 0 \tag{A4}$$

Model 5.2: *Step-1 problem for* Midori_{128}.

In Section 5.6, we experimentally evaluate them on two **MDC** problems (related-key and single-key) for Midori and AES. We also provide their the evaluation of *abstractXOR* on Rijndael against the model shown in the previous Chapter.

5.2 Notations and definitions

Given two integer values a and b, $[a;\ b]$ denotes the set of all integer values ranging from a to b. $[a;\ b[$ denotes the set of all integer values ranging from a to $b-1$. $\mathbb{N}^+$ denotes the set of all natural numbers (excluding 0).

A denotes a set of variables such that the domain of each variable $\Delta_k \in \mathsf{A}$ is $D(\Delta_k) \subseteq \{0,1\}$. Δ_k is assigned iff $\#D(\Delta_k) = 1$, and an assignment is complete if all variables of A are assigned. A^0 denotes the set of variables assigned to 0 and $\mathsf{A} \setminus \mathsf{A}^0$ denotes the set of variables that are either assigned to 1 or not yet assigned.

C denotes a set of abstract XOR constraints defined on A, where each abstract XOR constraint is of the form $\Delta_1 \circ \ldots \circ \Delta_l = 0$ and corresponds to a XOR equation $\delta_1 \oplus \ldots \oplus \delta_l = 0$.

$C_{\downarrow \mathsf{A}^0}$ denotes the set of XOR constraints obtained from C by (i) replacing each $\Delta_k \in \mathsf{A}^0$ with 0, (ii) replacing each $\Delta_k \in \mathsf{A} \setminus \mathsf{A}^0$ with an integer variable δ_k whose domain is $D(\delta_k) = \mathbb{N}^+$, and (iii) replacing each abstract XOR $\circ$ with the bitwise XOR $\oplus$. Examples are displayed in Figure 5.1 (equations of $C_{\downarrow \mathsf{A}^0}$ are simplified by replacing $\delta_k \oplus 0$ with δ_k).

$C_{\downarrow \mathsf{A}^0}$ is represented by a matrix M which contains one row for each equation and one column for each variable in $\mathsf{A} \setminus \mathsf{A}^0$: $M[j,k] = 1$ if δ_k occurs in the equation j of $C_{\downarrow \mathsf{A}^0}$; otherwise, $M[j,k] = 0$. We denote n and m the numbers of rows and columns of M.

For each row $j \in [0; n[$, we define $nonZero_j = \{k \in [0; m[\ \mid M[j,k] = 1\}, pivot_j = \min nonZero_j$,

Equations:

$$\mathsf{A} = \{\Delta_0, \Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5, \Delta_6\}$$

$$C = \left\{ \begin{array}{c} \Delta_0 \circ \Delta_3 \circ \Delta_5 \circ \Delta 6 = 0, \\ \Delta_1 \circ \Delta_3 \circ \Delta_4 \circ \Delta_6 = 0, \\ \Delta_2 \circ \Delta_4 \circ \Delta_5 = 0 \end{array} \right\}$$

Example 5.1

$C_{\downarrow \mathsf{A}^0}$:
$\delta_0 \oplus \delta_3 \oplus \delta_5 = 0$
$\delta_1 \oplus \delta_3 \oplus \delta_4 = 0$
$\delta_2 \oplus \delta_4 \oplus \delta_5 = 0$

M:

δ_0	δ_1	δ_2	δ_3	δ_4	δ_5
1	0	0	1	0	1
0	1	0	1	1	0
0	0	1	0	1	1

Example 5.2

$C_{\downarrow \mathsf{A}^0}$:
$\delta_0 \oplus \delta_3 \oplus \delta_5 = 0$
$\delta_3 \oplus \delta_4 = 0$
$\delta_4 \oplus \delta_5 = 0$

M:

δ_0	δ_3	δ_4	δ_5
1	1	0	1
0	1	1	0
0	0	1	1

Figure 5.1: *Top: A set* A *of variables and a set* C *of abstract* XOR *constraints. Bottom:* $C_{\downarrow \mathsf{A}^0}$ *and* M *when* Δ_6 *is assigned to 0 (Example 5.1, on the left) and when* Δ_1, Δ_2, *and* Δ_6 *are assigned to 0 (Example 5.2, on the right). In Example 5.1,* M *is in RRE form and* $nonZero_0 = \{0, 3, 5\}$, $pivot_0 = 0$, *and* $nonPivot_0 = \{3, 5\}$. *In Example 5.2,* M *is not in RRE form because the pivot columns of rows 1 and 2 have two non-zero cells.*

and $nonPivot_j = nonZero_j \setminus \{pivot_j\}$.

M is in *reduced row-echelon (RRE)* form iff, for every row $j \in [0; n[$, there is exactly one non-zero cell in column $pivot_j$, *i.e.*, $\sum_{j'=0}^{n-1} M[j', pivot_j] = 1$ (see examples in Figure 5.1).

5.3 Definition and complexity of Abstract XOR

When computing **MDC**s in a two-step process, we aim at minimizing as much as possible the number of Step-1 solutions which are Step2-inconsistent. As many Step2-inconsistencies come from the fact that xor constraints are poorly abstracted at Step-1, we introduce a global constraint to obtain a tighter Step-1 model.

Definition 5.1

Given an integer value $u > 0$, the constraint $abstractXOR_{u,C}(\Delta)$ is satisfied by a complete assignment iff $C_{\downarrow \Delta^0} \cup \{\Delta_k \leq u : \Delta_k \in \Delta \setminus \Delta^0\}$ is consistent.

Let us consider Example 5.1 of Figure 5.1. If $u = 3$, then $abstractXOR_{u,C}(\Delta)$ is satisfied because there exists a solution of $C_{\downarrow \Delta^0}$ such that every δ_k belongs to $[1;\ 3]$ (*e.g.*, $\delta_0 = \delta_4 = 1$, $\delta_1 = \delta_5 = 2$, and $\delta_2 = \delta_3 = 3$).

However, if $u = 2$, then $abstractXOR_{u,C}(\Delta)$ is not satisfied because $C_{\downarrow \Delta^0}$ has no solution when every δ_k must belong to $[1;\ 2]$.

In Example 5.2, $abstractXOR_{u,C}(\Delta)$ is not satisfied because $(\delta_3 \oplus \delta_4 = 0 \wedge \delta_4 \oplus \delta_5 = 0) \Rightarrow (\delta_3 = \delta_4 = \delta_5) \Rightarrow (\delta_3 \oplus \delta_5 = 0)$. Therefore, δ_1 must be equal to 0, which is impossible as δ_1 must belong to $[1, u]$.

abstractXOR allows us to easily model Step-1 problems. For example, for Midori_{128}, we replace constraints (A3) and (A4) with $abstractXOR_{255,C}(\Delta)$ where $C = \{(A3), (A4)\}$, and Δ contains

all variables involved in (A3) or (A4). The resulting model has less Step2-inconsistent solutions than the basic model obtained by replacing (A3) and (A4) with constraints that ensure that the sum is different from 1: *abstractXOR* ensures the consistency of (C3) and (C4) at Step-2, whereas the basic model only ensures the feasibility of each XOR separately.

However, checking the feasibility of *abstractXOR* is intractable.

Theorem 5.1

Deciding if a complete assignment satisfies $abstractXOR_{u,C}(\Delta)$ is an $\mathcal{NP}$-complete problem.

Proof 5.1

To decide whether $abstractXOR_{u,C}(\Delta)$ is satisfied by a complete assignment, we must decide whether $C_{\downarrow\Delta^0}$ is consistent when all δ_k variables occurring in $C_{\downarrow\Delta^0}$ are constrained to belong to $[1, u]$. This problem trivially belongs to $\mathcal{NP}$ as we can decide in polynomial time whether a given assignment of all δ_k variables satisfies $C_{\downarrow\Delta^0}$.
To show that it is $\mathcal{NP}$-complete, we give a reduction from the graph colouring problem, which aims at deciding if we can assign a colour $c_j \in [1;\ u]$ to each vertex j of a graph so that $c_j \neq c_k$ for each edge (j, k). Given a graph G, we associate a variable δ_j (resp. δ_{jk}) with every vertex j (resp. edge (j, k)) of G, and we define the XOR constraints:
$C = \{\delta_j \oplus \delta_k \oplus \delta_{jk} = 0 : (j, k) \text{ is an edge of } G\}$.
If each variable must belong to $[1;\ u]$, then each XOR constraint associated with an edge (j, k) ensures that $\delta_j \neq \delta_k$ (because $\delta_j = \delta_k \Leftrightarrow \delta_{jk} = 0$). Hence, we can show that every solution of C corresponds to a valid colouring of G, and vice-versa.

Now, let us show that we can decide if *abstractXOR* is satisfied in polynomial time when δ_k variables are not upper bounded. In this case, we have to decide if $C_{\downarrow\Delta^0}$ is consistent. We first show how to put the matrix M associated with $C_{\downarrow\Delta^0}$ in RRE form. This is done by Algorithm 5.1, which uses a principle similar to Gaussian elimination of linear equations.

Algorithm 5.1 does not change the set of solutions because it only removes empty rows (line 4), or replaces a row j' with the result of XORing it with another row j (line 7).

To show that Algorithm 5.1 puts M in RRE form, we show that the comment after line 2 is an invariant property of the loop lines 2-7. This property is trivially satisfied at the first iteration when $j = 0$ and, if it is satisfied at some iteration, then it is satisfied at the next iteration: if row j is empty (line 3) then it is removed and j is not incremented so that the property is still satisfied; otherwise (lines 4-7), every row $j' \neq j$ which contains a non-zero cell on column $pivot_j$ is XORed with row j so that column $pivot_j$ only contains one non-zero cell on row j just after lines 5-6.

The complexity of this algorithm is $\mathcal{O}(mn^2)$.

We use Property *atLeast2* (defined below) to decide if $C_{\downarrow\Delta^0}$ is consistent.

```
1  j ← 0
2  while j < n do
3  |  /* every row j' ∈ [0; j − 1[ is in RRE form, i.e., Σ_{j''=0}^{n−1} M[j'', pivot'_j] = 1   */
4  |  if nonZero_i = ∅ then remove row j and decrement n
5  |  else
6  |  |  for each row j' ∈ [0; n[ such that j' ≠ j and M[j', pivot_j] = 1 do
7  |  |  |  for each column k' ∈ [0; m[ do  M[j', k'] ← M[j', k'] ⊕ M[j, k']
8  |  |  end
9  |  |  j ← j + 1
10 |  end
11 end
```

Algorithm 5.1: *RRE form of an $n \times m$ matrix M*

Definition 5.2: Property *atLeast2*

A matrix M in RRE form satisfies Property *atLeast2* if each row has at least two non-zero cells, *i.e.*, $\forall j \in [0;\ n[, \#nonZero_j \geq 2$.

Theorem 5.2

$C_{\downarrow\Delta^0}$ is consistent iff its associated matrix M in RRE form satisfies Property *atLeast2*.

Proof 5.2

If M does not satisfy Property *atLeast2*, then it contains a row with exactly one non-zero cell, *i.e.*, there exists an equation of the form $\delta_k = 0$. In this case $C_{\downarrow\Delta^0}$ is inconsistent as $D(\delta_k) = \mathbb{N}^+$.

If M satisfies Property *atLeast2*, then we can always build a solution for $C_{\downarrow\Delta^0}$. The idea is to first assign values to variables associated with non-pivot columns, and then compute values of variables associated with pivot columns by XORing the corresponding non-pivot variables. To ensure that values computed for pivot variables are always different from 0, we have to choose carefully the values of non-pivot variables. More precisely, non-pivot variables are assigned one after the other. When choosing a value for a non-pivot variable δ_k, for each row j such that $k \in nonPivot_j$, if all variables of $nonPivot_j$ but δ_k are already assigned, then we must choose a value different from the result of the XOR of these assigned variables. As the domains of δ_k variables are not upper bounded, we can always build a solution.

A consequence of Theorem 5.2 is that we can decide in polynomial time if a complete assignment satisfies $abstractXOR_{\infty,C}(\Delta)$. Indeed, this amounts to deciding whether $C_{\downarrow\Delta^0}$ is consistent. This can be done by using Algorithm 5.1 to put the matrix M associated with $C_{\downarrow\Delta^0}$ in RRE form, and then checking that Property *atLeast2* is satisfied.

5.4 Propagation of Abstract XOR

As deciding of the satisfaction of $abstractXOR_{u,C}(\Delta)$ is polynomial when $u = \infty$, we consider that $u = \infty$ from now on.

In this section, we introduce an algorithm that checks feasibility (Section 5.4.1), an algorithm that ensures Generalised Arc Consistency (Section 5.4.2), and we discuss implementation and complexity issues (Section 5.4.3).

5.4.1 Checking feasibility of Abstract XOR

```
1 for each row j ∈ [0; n[ such that nonPivotj = ∅ do
2 |   if D(Δpivot_i) = {1} then return failure
3 |   else
4 |   |   remove 1 from D(Δpivot_i)
5 |   |   remove row j and decrement n
6 |   |   remove column pivot_j and decrement m
7 |   end
8 end
9 return success
```

Algorithm 5.2: *Check Property atLeast2 of an $n \times m$ matrix M in RRE form*

Before starting the search, we build the matrix M associated with $C_{\downarrow\Delta^0}$ and use Algorithm 5.1 to put it in RRE form. During the search, we maintain M in RRE form: each time a variable $\Delta_k \in \Delta$ is assigned to 0, we remove column k from M and, if k is the pivot column of a row j, we execute lines 3-6 of Algorithm 5.1.

Once M is in RRE form, we check feasibility by exploiting Theorem 5.2, as shown in Algorithm 5.2: for each row j with only one non-zero cell, if Δ_{pivot_j} is assigned to 1 we trigger failure, otherwise we assign 0 to Δ_{pivot_j} and remove row j and column $pivot_j$ from M.

5.4.2 Ensuring the Generalized Arc Consistency of Abstract XOR

To ensure GAC, we must ensure that for each variable $\Delta_k \in \Delta$ and each value $v \in D(\Delta_k)$, the couple (Δ_k, v) has a support, *i.e.*, there exists a consistent assignment which assigns v to Δ_k and a value $v' \in D(\Delta_{k'})$ to every other variable $\Delta_{k'} \in \Delta \setminus \{\Delta_k\}$. By maintaining M in RRE form and ensuring Prop. *atLeast2*, we ensure that $(\Delta_k, 1)$ has a support for each variable $\Delta_k \in \Delta$ such that $1 \in D(\Delta_k)$. Also $(\Delta_k, 0)$ has a support for every variable $\Delta_k \in \Delta$ assigned to 0. However, when Δ_k is not assigned, $(\Delta_k, 0)$ may not have a support. This occurs when there exist $\Delta_k, \Delta_{k'} \in \Delta \setminus \Delta^0$ such that $D(\Delta_k) = \{0,1\} \wedge D(\Delta_{k'}) = \{1\} \wedge C_{\downarrow\Delta^0} \Rightarrow (\delta_k = \delta_{k'})$. In this case, the couple $(\Delta_k, 0)$ has no support because $C_{\downarrow\Delta^0} \wedge (\delta_k = 0) \wedge (\delta_{k'} = 1)$ is inconsistent. Hence, to ensure GAC we need to identify cases where the equality of two variables is a logical consequence of $C_{\downarrow\Delta^0}$. This is done by the following theorem.

Theorem 5.3

For each pair of variables $\{\Delta_k, \Delta_{k'}\} \subseteq \Delta \setminus \Delta^0$, $C_{\downarrow\Delta^0} \Rightarrow (\delta_k = \delta_{k'})$ iff one of the following cases holds in the matrix M in RRE form associated with $C_{\downarrow\Delta^0}$:

Case 1: $\exists j \in [0;\ n[, nonZero_j = \{k, k'\}$

Case 2: $\exists j, j' \in [0;\ n[, (pivot_j = k) \wedge (pivot_{j'} = k') \wedge (nonPivot_j = nonPivot'_j)$

Proof 5.3

Case 1 occurs when M contains a row j with exactly two non-zero cells, and this row corresponds to the equation $\delta_k = \delta_{k'}$.

Case 2 occurs when M contains 2 rows j and j' such that $nonPivot_j = nonPivot_{j'}$. These rows imply that $\delta_{pivot_j} = \delta_{pivot_{j'}}$ because both δ_{pivot_j} and $\delta_{pivot_{j'}}$ are equal to the result of XORing a same set of variables.

There is no other case where $C_{\downarrow\Delta^0} \Rightarrow (\delta_k = \delta_{k'})$ because, when M is in RRE form, every row j has a different pivot column $pivot_j$. Therefore, every equation in $C_{\downarrow\Delta^0}$ contains a different pivot variable δ_{pivot_j}. Hence, δ_k and $\delta_{k'}$ are constrained to be equal either because they occur in a same equation without any other variable, or because they are the pivot variables of two different equations which share the same non-pivot variables.

Let us illustrate these two cases on Example 5.1:

- If $\Delta^0 = \{\Delta_4, \Delta_6\}$ then $C_{\downarrow\Delta^0}$ contains the equation $\delta_1 \oplus \delta_3 = 0$, and if $D(\Delta_3) = \{1\}$ and $D(\Delta_1) = \{0, 1\}$, then $(\Delta_1, 0)$ has no support.
- If $\Delta^0 = \{\Delta_2, \Delta_4, \Delta_5\}$, then

$$C_{\downarrow\Delta^0} \text{ is equal to: } \begin{array}{l} \delta_0 \oplus \delta_3 \oplus \delta_6 = 0 \\ \delta_1 \oplus \delta_3 \oplus \delta_6 = 0 \end{array}$$

$$M \text{ is equal to: } \begin{array}{c} \begin{array}{cccc} \delta_0 & \delta_1 & \delta_3 & \delta_6 \end{array} \\ \left[\begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{array}\right] \end{array}$$

This implies that the pivot variables δ_0 and δ_1 are both equal to $\delta_3 \oplus \delta_6$, and if $D(\Delta_0) = \{1\}$ and $D(\Delta_1) = \{0, 1\}$, then $(\Delta_1, 0)$ has no support.

To maintain GAC during the search, we call Algorithm 5.3 each time a variable must be assigned to 1. This algorithm uses a queue Q of variables that must be assigned to 1. At each iteration of the loop lines 2-11, a variable Δ_k is dequeued from Q, and it is assigned to 1. This assignment is propagated on every variable $\Delta k'$ such that $C_{\downarrow\Delta^0} \Rightarrow (\delta_k = \delta_{k'})$. We exploit Theorem 5.3 to identify these variables:

- Case 1 has two sub-cases: if k is the pivot column of a row j, we simply check if $nonPivot_j$ is reduced to a singleton (line 5); otherwise, we have to search for every row j' such that $nonPivot_{j'}$ only contains k (lines 10-11).

```
1  let Q be an empty queue; enqueue Δ_k in Q
2  while Q is not empty do
3  |  dequeue a variable Δ_k from Q and remove 0 from D(Δ_k)
4  |  if k is the pivot column of a row j then
5  |  |  if nonPivot_j = {k'} and D(Δ_k') = {0,1} then enqueue Δ_k' in Q
6  |  |  else
7  |  |  |  for each j' ∈ [0; n[ such that nonPivot_j = nonPivot_j' do
8  |  |  |  |  if D(Δ_pivot_j') = {0,1} then enqueue Δ_pivot_j' in Q
9  |  |  |  end
10 |  |  end
11 |  else
12 |  |  for each j ∈ [0; n[ such that nonPivot_j = {k} do
13 |  |  |  if D(Δ_pivot_j') = {0,1} then enqueue Δ_pivot_j' in Q
14 |  |  end
15 |  end
16 end
```

Algorithm 5.3: *Propagation of the assignment of 1 to a variable* Δ_k

- Case 2 only holds when k is the pivot column of a row j, and we have to search for every row j' such that $nonPivot_j = nonPivot_{j'}$ (lines 7-8).

Also, each time a variable is assigned to 0, we proceed as explained in Section 4.1 to check feasibility. Then, for each line which has been modified when executing lines 3-6 of Algorithm 5.1, we check if cases 1 or 2 of Theorem 5.3 hold and imply that $\delta_k = \delta_{k'}$ with $D(\Delta_k) = \{0,1\}$ and $D(\Delta_{k'}) = \{1\}$: in this case, we call Algorithm 5.3 to propagate the assignment of 1 to Δ_k.

5.4.3 Implementation and Complexities

Sparse Sets. Our propagators mainly involve traversing non-zero cells of rows and columns of M. As M is very sparse, we represent its rows and columns with sparse sets [Le +13]: each sparse set contains the non-zero cells of a row or a column. This allows us to visit every non-zero cell of a column (resp. row) in linear time with respect to the number of non-zero cells instead of $\mathcal{O}(m)$ (resp. $\mathcal{O}(n)$), and to decide in constant time if an element belongs to a set. Sparse sets also allow to restore sets in constant time when backtracking, provided that we only remove elements at each choice point. Unfortunately, this is not the case here as new non-zero cells may appear when XORing lines. Hence, when backtracking, we undo all operations done before the recursive call.

Time complexity of the propagators. Let n_1 (resp. m_1) be the maximum number of non-zero cells in a row (resp. a column) of M. When using sparse sets, the complexity of putting M in RRE form, as described by Algorithm 5.1, becomes $\mathcal{O}(nn_1m_1)$.

The complexity of the propagation of the assignment of a variable to 0 is $\mathcal{O}(n_1m_1)$. Indeed, when a variable Δ_k is assigned to 0, we have to (i) remove column k, (ii) execute lines 3-7 of Algorithm 5.1 if k is a pivot column, and (iii) run Algorithm 5.2. The complexity of this depends on whether k is a pivot column or not:

- if k is a pivot column, then (i) is achieved in $\mathcal{O}(1)$ as column k only contains one non-zero cell; (ii) is achieved in $\mathcal{O}(n_1 m_1)$; and (iii) is achieved in $\mathcal{O}(n_1)$ provided that we keep track of the rows that have been modified at step (ii);
- if k is not a pivot column, then (i) is achieved in $\mathcal{O}(n_1)$ and (iii) is achieved in $\mathcal{O}(n_1)$ provided that we keep track of the rows that have been modified at step (i).

The complexity of the propagation of the assignment of a variable to 1 by Algorithm 5.3 is $\mathcal{O}(mn_1m_1)$.

Indeed, in the worst case, this implies to assign 1 to every other variable. Hence, the loop lines 2-11 is performed $\mathcal{O}(m)$ times. The loop lines 7-8 is iterated $\mathcal{O}(n_1)$ times (we traverse non-zero cells of the column of a variable in $nonPivot_j$ to identify the rows j' for which we have to check if $nonPivot_j = nonPivot_{j'}$), and we decide if $nonPivot_j = nonPivot_{j'}$ in $\mathcal{O}(m_1)$. The loop lines 10-11 is iterated $\mathcal{O}(n_1)$ times as we only have to consider the non-zero cells of column k.

Implementation. Our global constraint has been implemented in Java and integrated in Choco 4 [PFL16]. As its propagators are rather expensive, we give a low priority to *abstractXOR* so that, at each node of the search tree, Choco propagates all other constraints before propagating *abstractXOR*.

The source code is available at `https://gitlab.inria.fr/lrouquet/abstract-xor-library`.

5.5 Advanced Techniques for Rijndael

In most cases, **CP** solvers use backtrack methods as seen in the Chapter 2. These algorithms require two heuristics to work: the variable ordering heuristic and the value ordering heuristic. The variable ordering heuristic aims at defining what should be the next variable to be instantiated when the solver has to extend the current partial solution. The value heuristic has the same objective, but for values.

The `setSum` constraint [Bes+04], which is used in the objective function, is hard to propagate efficiently. To counter this problem, we add higher level information as described in Section 5.5.1 and we show how to adapt DomOverWDeg [Bou+04] - which is a good performing and commonly accepted variable ordering heuristic - to our problem in Section 5.5.2.

5.5.1 Higher level constraints

AES performs the XORs (`AddRoundKey` and `KeySchedule`) column by column. In this case, it is interesting to see if it is possible to gain information about the abstraction of the XOR at the column level (Figure 5.2).

As we saw earlier, the abstraction of the XOR operator is complex. The first abstraction of the operator XOR for three variables δ_A, δ_B and δ_C such that: $\delta_A \oplus \delta_B = \delta_C$ (which is equivalent to $\delta_A \oplus \delta_B \oplus \delta_C = 0$) is $\Delta_A + \Delta_B + \Delta_C \neq 1$. This abstraction can also be represented using these

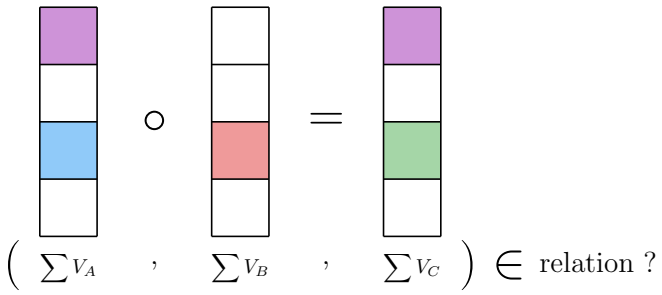


Figure 5.2: XOR *on columns. The objective is to determine the values of* $\sum V_A, \sum_{V_B}$ *and* $\sum_{V_C}$.

two constraints:

$$\Delta_A + \Delta_B \geq \Delta_C$$
$$\Delta_C \implies (\Delta_A = \Delta_B)$$

The first constraint indicates that there can only be a difference in the output of the XOR if there is at least one difference in the input. The second constraint indicates that if Δ_C is equal to zero then necessarily $\Delta_A = \Delta_B$ (since we have: $\delta_C = 0 \implies \delta_A = \delta_B$).

The interest of this decomposition is that it can be extended to vectors, *i.e.* three vectors V_A, V_B and V_C of size n, such that: $\forall j \in [0;\ n[, V_A[j] \oplus V_B[j] \oplus V_C[j] = 0$, we have:

$$\forall (X, Y, Z) \in \{(A, B, C), (C, A, B), (B, C, A)\},$$
$$\sum_{j=0}^{n-1} (V_X[j] + V_Y[j]) \geq \sum_{j=0}^{n-1} V_Z[j]$$
$$\sum_{j=0}^{n-1} V_Z[j] = 0 \implies \left(\sum_{j=0}^{n-1} V_X[j] = \sum_{j=0}^{n-1} V_Y[j] \right)$$

This information can be used to filter out inconsistent states more quickly and to better propagate the number of active differential bytes through the vectors. It can also be used to help the DomOverWDeg heuristic [Bou+04] as we will see below.

5.5.2 Custom heuristic

Variable ordering heuristic

As shown in Figure 5.3, the value of the Δ variables is summed up by column and by round. This representation has two advantages:

1. it decomposes the sum of active S-boxes to more effectively maintain the `setSum` constraint which is related to the objective function,
2. it makes it possible to group the variables by sets. As a result, each Δ variable now belongs to three levels of constraint, constraints on the Δ variables, constraints on the sums of columns and constraints on the sums of rounds, which makes it possible to disuniformize the score of the Δ variables between the columns and between the turns, which was not

possible using *abstractXOR* alone.

The main modification we have made to DomOverWDeg is the aggregation of the costs of the variables. Indeed, the weight of a variable is no longer calculated only from its own information, but from its decomposition. In the case of the variable $\sum_{j=0}^{3}\sum_{k=0}^{3}\Delta_X[j,k]$, the score of the variable is calculated from the set of variables $\{\sum_{j=0}^{3}\Delta_X[y,0], \sum_{j=0}^{3}\Delta_X[y,1], \sum_{j=0}^{3}\Delta_X[y,2], \sum_{j=0}^{3}\Delta_X[y,3]\}$ whose own score is calculated from the variables $\Delta[j,k]$ as shown in Figure 5.3.

$$\begin{aligned}
&\texttt{weight}(\sum_{j=0}^{3}\sum_{k=0}^{3}\Delta_X[j,k])\\
&= \underset{j=0}{\overset{3}{\text{Agg}}}\,\texttt{weight}(\sum_{k=0}^{3}X[j,k])\\
&= \underset{j=0}{\overset{3}{\text{Agg}}}\,\underset{k=0}{\overset{3}{\text{Agg}}}\,\texttt{weight}(\Delta_X[j,k])
\end{aligned}$$

where the aggregation function Agg is an aggregation of the DomOverWDeg score of the variable and its decomposition variables.

After some empirical tries, the last formula we used is:

$$\texttt{weight}(v) = (1.0 + DomOverWDeg(v)) \times \prod_{d \in d_{Vars} \text{ of } v} \texttt{weight}(v)$$

where d_{Vars} of v means the decomposition variables of v.

Value ordering heuristic

We use the knowledge of the problem to create a value ordering heuristic. Since our goal is to maximize the overall probability, we can select S-box values that maximize this probability. When a value associated with a variable belonging to an S-box, δ_{in}, δ_{out} or p is to be chosen we select the best possible value for that variable. The `DDT` of Rijndael have only three non-null values: $1, 2^{-6}$ and 2^{-7}. If we denote v_{in} and v_{out} the values of δ_{in} and δ_{out}, for each v_{in} (excepted 0), there is only one transition $v_{in} \rightsquigarrow v_{out}$ where $Pr[v_{in} \rightsquigarrow v_{out}] = 2^{-6}$ and 126 transitions where $Pr[v_{in} \rightsquigarrow v_{out}] = 2^{-7}$. The value ordering heuristic works like this:

- in the case of a p variable, we select the highest remaining probability first,
- in the case of a δ_{in} variable:
  1. if possible we select the value 0 first in order to reach the probability 1,
  2. else if its relative δ_{out} is fixed, we try to the select the value of the only transition which have a 2^{-6} probability,
  3. else we use the default solver heuristic, as we can only reach the 2^{-7} probability,
- in the case of a δ_{out} variable we use the same techniques as for δ_{in} variables.

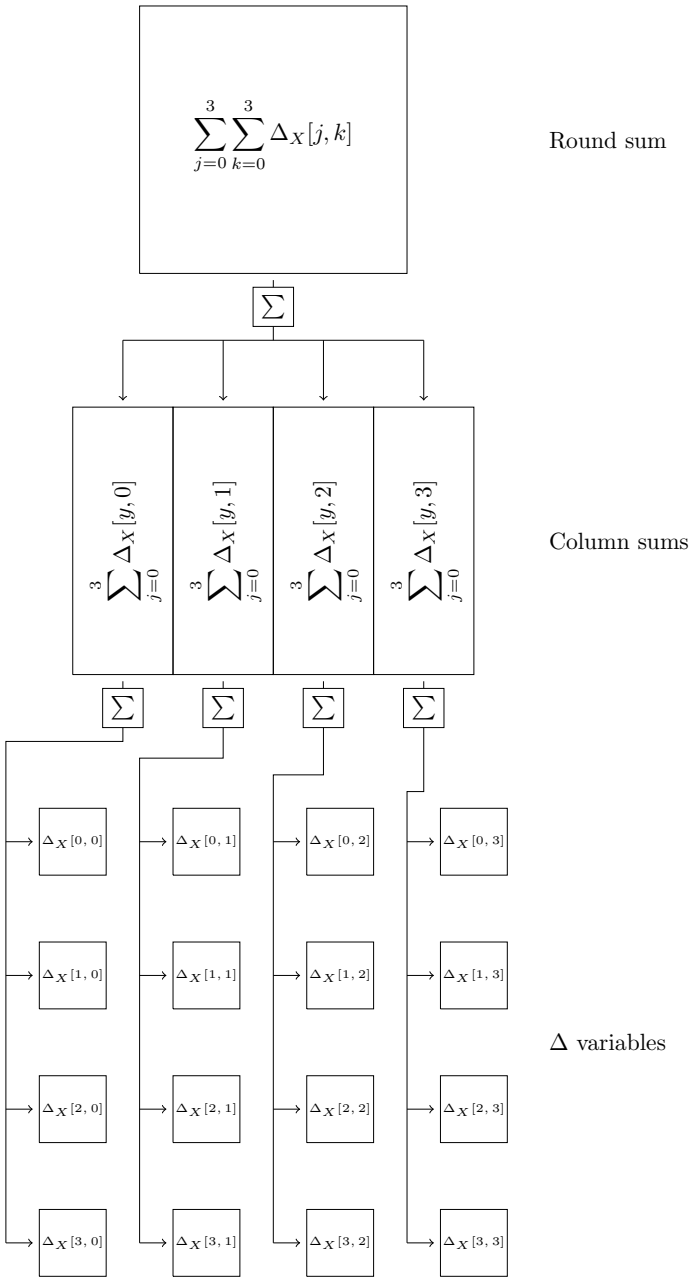


Figure 5.3: *Round sum decomposition.*

5.6 Experimental evaluation

In this section, we experimentally evaluate the interest of *abstractXOR*. We first consider the related-key **MDC** problem, where differences can be injected both in the key and the input text, and we report results for Midori in Section 5.6.1 and for AES in Section 5.6.2. In Section 5.6.3, we consider the single-key **MDC** problem, where differences are injected only in the input text. All experiments have been done on a single core of an Intel Xeon E3-1270v3 (3.50 GHz) with 32 GB RAM.

5.6.1 Related-Key MDC for Midori

Description of the problem. The related-key **MDC** for Midori is described in Section 5.1 for the case where the input text X_0 is a sequence of 128 bits (denoted Midori_{128}). Midori is also defined for 64 bit texts (denoted Midori_{64}). In this case, `SubBytes` and *subBytesTable* are defined for 4 bit sequences instead of 8 bit sequences as depicted in Section 1.2.2. Also, a `KeySchedule` is used to compute a new subkey at each round (see [Ban+15] for details).

We consider different values for r, ranging from 3 to the number of rounds defined in [Ban+15], *i.e.*, 16 (resp. 20) for Midori_{64} (resp. Midori_{128}). For each value of r, the constant n used in constraint (A0) of Model 5.2 is set to the smallest value for which there exists a solution, as this is the most difficult instance: instances with smaller values of n are often trivially inconsistent, whereas instances with larger values are useless.

We report results on two problems: $Enum_1$ aims at enumerating all solutions of the Step-1 problem described in Model 5.2 for Midori_{128}; Opt_{1+2} aims at finding the **MDC** whose probability is maximal as described in Model 5.1 for Midori_{128}.

Models for Enum_1. We consider two models. The first one, denoted $Enum_1$ *Global*, is derived from Model 5.2 in a straightforward way, by replacing (A3) and (A4) with $abstractXOR_{\infty,C}(\Delta)$ where $C = \{(A3), (A4)\}$ and Δ contains all variables occurring in (A3) or (A4). It is implemented in Java with Choco 4 [PFL16], and we consider two propagators: $Global_{Feas}$ only checks feasibility, as described in Section 5.4.1, and $Global_{GAC}$ ensures **GAC**, as described in Section 5.4.2. In both cases, as the model of Midori is enough faster with the DomOverWDeg variable ordering heuristic [Bou+04], we do not use the advanced heuristics presented in Section 5.5.

The second model, denoted $Enum_1$ *Advanced*, is introduced in [Gér18] and uses the same advanced techniques used for AES presented in Section 3.1. This model is much more difficult to design than *Global*. For this model, we report results obtained by Picat-SAT [ZK16], which encodes the problem into a **SAT** formula and then uses the **SAT** solver Lingeling [Bie]. We made experiments with other **CP** solvers (such as Choco, Gecode or Chuffed, for example), and we only report results obtained with Picat-SAT because it scales much better.

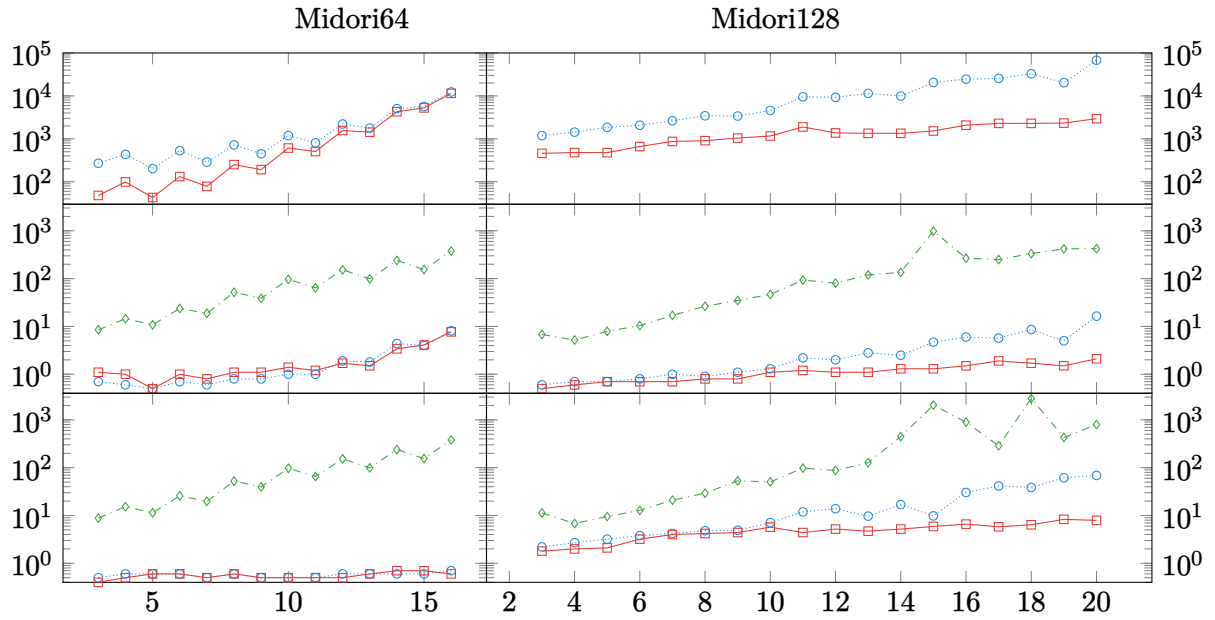


Figure 5.4: *Comparison of* Global_{Feas} (⋯○⋯), Global_{GAC} (—□—), *and Advanced* (-·◇·-) *for* Midori. *The x-axis gives the number of rounds* r*, and the y-axis the number of choice points for* Enum_1 *(up), and the run time for* Enum_1 *(Middle) and for* Opt_{1+2} *(bottom). Times are in seconds.*

Models for Opt_{1+2}. The problem described in Model 5.1 cannot be solved within a reasonable amount of time (even for the smallest value of r) without decomposing it into two steps, as described in Section 5.1. We consider two models for this two step process. Opt_{1+2} *Global* simply merges $Enum_1$ *Global* with the model of Model 5.1, and adds a constraint which relates δ_k and Δ_k variables, *i.e.*, $\delta_k = 0 \Leftrightarrow \Delta_k = 0$. Also, we add a variable ordering heuristic to assign Δ_k variables before δ_k variables. This model is implemented in Choco 4.

Opt_{1+2} *Advanced* uses $Enum_1$ *Advanced* to search for Step-1 solutions. However, we do not merge this model with the Step-2 model of Model 5.1 and use a single solver to solve the two steps because **CP** solvers like Choco cannot efficiently solve $Enum_1$ *Advanced* whereas **SAT** solvers like Lingeling cannot efficiently solve Step-2 [Gér+20]. Hence, Opt_{1+2} *Advanced* uses Picat-SAT to solve $Enum_1$ *Advanced*, and each time a Step-1 solution s is found, it uses Choco with the model of Model 5.1 to search for the best Step-2 solution associated with s. This process is stopped either when there is no more Step-1 solution, or when an optimal Step-2 solution is found (*i.e.*, a solution such that all $P[i,k]$ variables are assigned to -2 as this is the largest possible value).

Results. On the top row of Figure 5.4, we display the number of choice points needed to enumerate all Step-1 solutions. $Global_{GAC}$ explores less choice points than $Global_{Feas}$, though the difference is very small for Midori_{64} when $r \geq 12$.

In the middle row of Figure 5.4, we display the CPU time spent to enumerate all Step-1 solutions. For Midori_{64}, the two *Global* variants have very similar times whereas, for Midori_{128}, $Global_{GAC}$ is faster than $Global_{Feas}$. *Advanced* is much slower than *Global*.

In the bottom row of Figure 5.4, we display the CPU time needed to solve the full **MDC** problem. For Midori_{64}, $Global_{Feas}$ and $Global_{GAC}$ have very similar results, and are much faster

than *Advanced*. For Midori$_{128}$, *Global*$_{GAC}$ is faster than *Global*$_{Feas}$, which is faster than *Advanced*, especially when r increases.

5.6.2 Related-Key MDC for AES

Like in Section 5.6.1, we consider two problems: *Enum*$_1$ aims at enumerating all Step-1 solutions, and *Opt*$_{1+2}$ aims at finding the optimal **MDC**.

Models for *Enum*$_1$. We consider two **CP** models. *Global* is derived in a straightforward way from the definition of AES and the **MDS** property by replacing all XOR equations with an *abstractXOR* global constraint. It is implemented with Choco 4, and we consider two propagators (ensuring feasibility and GAC, respectively). Both models use the heuristic techniques described in Section 5.5.

Advanced is the model introduced in [Gér+20] and recalled in Section 3.1. It uses a preprocessing step to infer new XOR equations from the `KeySchedule`, and it adds new variables and constraints to remove *Step2-inconsistent* solutions by reasoning on equality relations between Δ_k variables. This model is much more difficult to design than *Global*. It is implemented with Picat-SAT.

Models for *Opt*$_{1+2}$. Like in Section 5.6.1, *Global* solves the two steps with a single model implemented with Choco 4 whereas *Advanced* enumerates Step-1 solutions with Picat-SAT and searches for optimal **MDC**s with Choco 4.

Extending the AES model to Rijndael

In order to further analyse *abstractXOR*, we compared the *Global*$_{GAC}$ version on Rijndael against our **SAT** / **CP** model described in the previous chapter. The time limit is set to 2 time the solving time of the **SAT** / **CP** model with a minimum of 10 seconds to compensate for the start-up of the Java virtual machine. The results tables are available in Appendix A.2. On the majority of small instances, *Global*$_{GAC}$ performs better than the **SAT** / **CP** version, but *Global*$_{GAC}$ performs less well on larger instances where the hybrid model is mostly faster.

5.6.3 Experimental results for the single-key problem

In the single-key differential attack, differences are introduced only in the initial text X_0, and no difference is introduced in the key, *i.e.*, $\delta K = 0$. Like for related-key, we consider two problems: *Enum*$_1$ (to enumerate all Step-1 solutions), and *Opt*$_{1+2}$ (to find the optimal **MDC**). We also consider two block ciphers, *i.e.*, Midori and AES. In all cases, we consider *Global* and *Advanced* models, and these models are obtained from related-key models by assigning 0 to all variables associated with the key.

CPU times are reported in Table 5.1. For AES, the problem is the same whatever the length of the key (128, 192, or 256), as there is no difference in the key. For Midori, *Enum*$_1$ is the same whatever the length of the initial text (64 or 128) as bit sequences are abstracted by Boolean values. However, *Opt*$_{1+2}$ is different for Midori$_{64}$ and Midori$_{128}$.

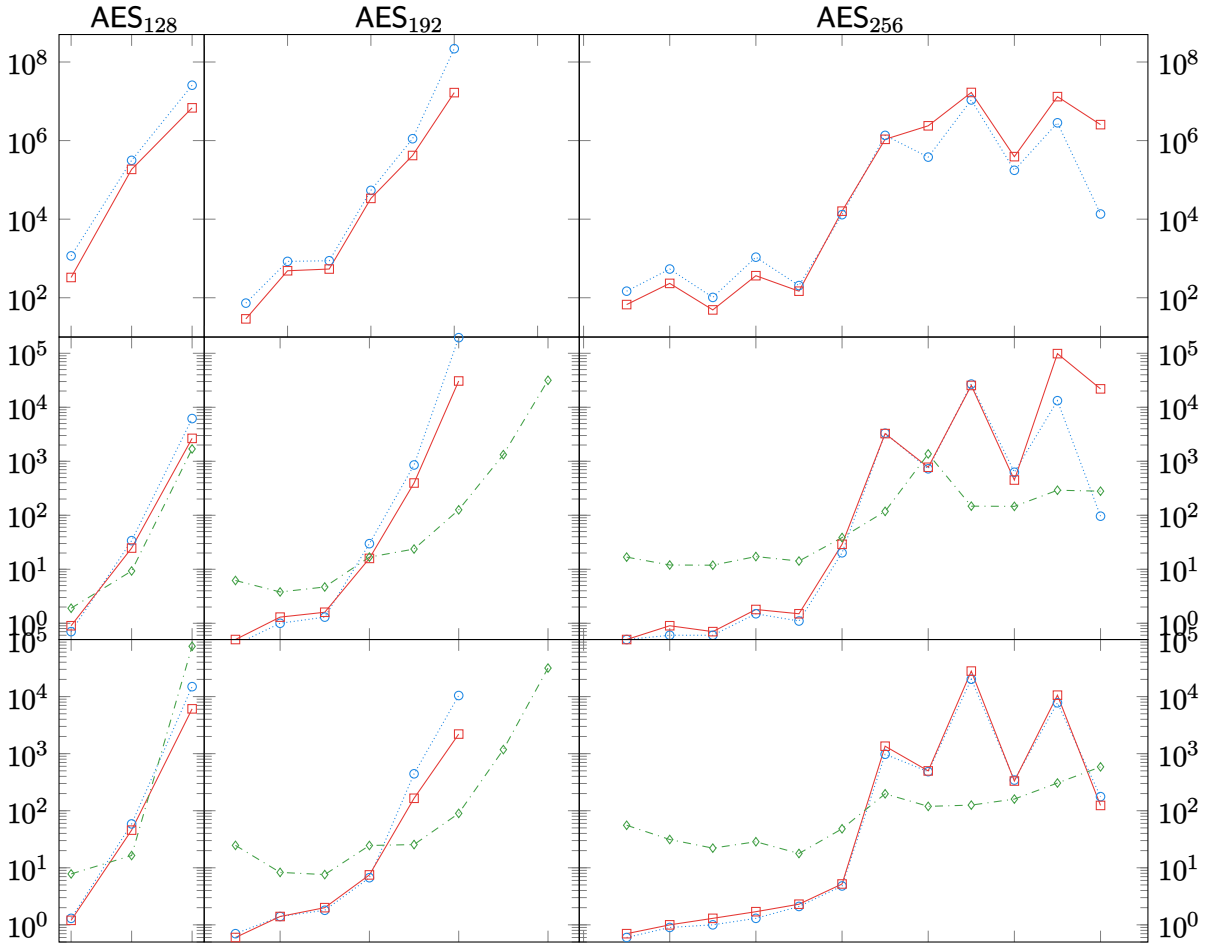


Figure 5.5: *Comparison of* Global$_{Feas}$ *(⋯○⋯),* Global$_{GAC}$ *(—□—), and* Advanced *(-·◇·-) for* AES*. The x-axis gives the number of rounds r, and the y-axis the number of choice points for* Enum$_1$ *(up), and the run time for* Enum$_1$ *(Middle) and* Opt$_{1+2}$ *(bottom). Times are in seconds.*

Surprisingly, single-key problems are much harder to solve than related-key ones, though the size of the search space is smaller (as all variables associated with the key are assigned to 0). This comes from the fact that the number of differences (defined by the constant n in Model 5.2) is strongly increased: n is increased from 3 (resp. 4 and 5) to 7 (resp. 16 and 23) when $r = 3$ (resp. 4 and 5) for Midori, and from 5 (resp. 12) to 9 (resp. 25) when $r = 3$ (resp. 4) for AES.

Results for Midori. *Advanced* finds much more Step-1 solutions than *Global*: it finds 64 (resp. 4,908) solutions when $r = 3$ (resp. 4), whereas *Global* finds 16 (resp. 68) solutions. Every solution found by *Advanced* and not by *Global* is *Step2-inconsistent* and *Advanced* spends a lot of time to enumerate these useless solutions. Hence, *Advanced* is not able to solve Midori within one hour when $r > 3$. When $r = 4$, *Advanced* is able to solve $Enum_1$ in 59,036s, but it is not able to solve Opt_{1+2} within a reasonable amount of time because most Step-1 solutions are *Step2-inconsistent*.

Global is able to solve up to $r = 5$ (resp. $r = 4$) for Midori$_{64}$ (resp. Midori$_{128}$). Step-2 is much harder for Midori$_{128}$ than for Midori$_{64}$ because differential variables associated with the text take their values in $[0;\ 255]$ for Midori$_{128}$ and in $[0;\ 16]$ for Midori$_{64}$.

	Enum_1			Opt_{1+2}					
	Midori 64 and 128			Midori_{64}			Midori_{128}		
r	G_{Feas}	G_{GAC}	Adv	G_{Feas}	G_{GAC}	Adv	G_{Feas}	G_{GAC}	Adv
3	0.6	0.6	8.7	0.7	0.6	11.1	1.3	8.1	12.5
4	22.4	8.0	-	17.6	11.1	-	434.9	290.5	-
5	2897.8	686.8	-	2608.1	689.7	-	-	-	-

	AES 128, 192, and 256					
	Enum_1			Opt_{1+2}		
	G_{Feas}	G_{GAC}	Adv	G_{Feas}	G_{GAC}	Adv
3	1.3	0.7	7.5	1.1	1.0	55.0
4	-	-	-	1.2	1.4	-

Table 5.1: *Single-Key results: Time (in seconds) needed by* Global_{Feas} *(*G_{Feas}*),* Global_{GAC} *(*G_{GAC}*), and* Advanced *(*Adv*) for* Midori *(left) and* AES *(right). We report '-' when time exceeds 3600s.*

Results for AES. When $r = 3$, both $Enum_1$ and Opt_{1+2} are quickly solved, and *Global* is an order faster than *Advanced*. When $r = 4$, there is a huge number of Step-1 solutions (we have enumerated 1,715,652 solutions within a 24 hour time limit with $Global_{GAC}$, and all these solutions are *Step2-consistent*). Hence, *Global* fails at enumerating all Step-1 solutions within a reasonable amount of time. However, when merging Step-1 and Step-2 models to solve Opt_{1+2}, we find an optimal solution in less than 2s (the optimality proof is trivial because all $P[i, k]$ variables are assigned to the largest possible value).

When $r = 4$, the probability of the optimal **MDC** is equal to 2^{-150}, which is smaller than 2^{-128}. Hence, this **MDC** is useless to mount attacks. However, the fact that *Global* is able to enumerate a huge number of Step-1 solutions in a reasonable amount of time opens new perspectives: we can search for a set of **MDC**s that share the same values in the initial text δX_0 and in the cipher text δX_r, and combine these **MDC**s to find better differentials.

5.7 Conclusion

We have introduced a new global constraint which eases the design of models for computing **MDC**s: these models are straightforwardly derived from problem definitions. This global constraint allows us to compute **MDC**s much faster than advanced models (which are much more difficult to design and which combine **SAT** and **CP** solvers) for single-key and related-key Midori, and for single-key AES. However, for related-key AES, it fails at solving the two largest instances of AES_{192} within a reasonable amount of time, and **SAT** has better scale-up properties for enumerating Step-1 solutions. As pointed out in [Gér+20], clause learning is a key ingredient for solving this problem, and further work will aim at improving scale-up properties of Choco on this problem by adding clause learning to Choco. Moreover improvements of the model **SAT** / **CP** presented in the previous Chapter widens the gap between the use of the **SAT** / **CP** hybrid model against the $Global_{GAC}$ model. However, even if the overall performance is not competitive with hybrid models, the *abstractXOR* constraint significantly improves Choco's performance on this type of modeling.

We believe our new global constraint opens promising perspectives for cryptographs, and we aim

at using it to solve new differential cryptanalysis problems such as those studied in [Cid+18] or [Tod+17], and new symmetric block ciphers such as Skinny [Bei+16].

As the models in the following chapters become even more complex, we favour the hybrid **SAT** / **CP** approach of the previous chapter. We start with the modelling of the boomerang attack on Rijndael in the next Chapter and continue with the generalization of the Delaune *et al.* model to Feistel ciphers in Chapter 7.

Chapter

6 Automatic Boomerang Attacks on Rijndael

Contents

6.1 Introduction

The boomerang attack [Wag99] was introduced at FSE'99 as a variant of differential attacks [BS91]. A cipher E is seen as the decomposition of two subciphers: $E = E_1 \circ E_0$ where the differential analysis takes place in each subcipher. Boomerang attacks are efficient when the cipher E has short differentials with high probabilities. They have been generalized to the related-key case in [BDK05]. Recently, new insights on what exactly happens in the middle (at the junction of E_1 and E_0) have been investigated. First, in [Cid+18], a special table named *Boomerang Connectivity Table* (BCT) has been introduced for Substitution-Permutation-Networks (**SPN**) to compute the probability of the middle round. Second, a careful analysis of the Skinny cipher has been provided in [DDV20] to automatically take into account more possible dependencies that could happen in the middle part of the cipher considering or not related-key. As in the previous chapters, the proposed search is divided in two steps: in a first step, the possible differences are modeled by Boolean variables and this step aims at minimizing an upper bound of the probability of the truncated boomerang distinguisher; in a second step that takes as input the trails found at

Step-1, the model aims at maximizing the overall probability considering that the active S-boxes depend on the output of the Step-1.

Thus, in Step-1, we compute a truncated related-key boomerang S_1 where each differential byte δ_A of the ciphering process is replaced by a Boolean variable Δ_A that indicates whether δ_A contains a difference or not. In Step-2, we instantiate S_1 into a related-key boomerang distinguisher. Note that some truncated boomerangs cannot be instantiated to a boomerang because some abstractions are done at Step-1.

In this chapter, we implement and adapt for the Rijndael case [DR99] the two-step solving process of [DDV20] originally proposed for Skinny for computing related-key boomerang differential characteristics. Those problems are solved with Constraint Programming (**CP**): for the first step, we use Picat-SAT [ZK16], and for the second step, Choco [PFL16].

When looking at the state of the art concerning the cryptanalysis of Rijndael, some of the results are in the single key scenario [NP07; Zha+08; GM08], [Wan+13; Min17; Liu+19] or in the related-key scenario [Wan+15] and none of those attacks does better than the results presented here.

The rest of this Chapter is organized as follows: in Section 6.2, we detail the methods and our **CP** models; in Section Section 6.3, we sum up all the related-key boomerang distinguishers we obtained and we present two attacks based on the most efficient distinguishers; and finally, in Section 6.4, we conclude this chapter.

6.2 Automatic Search of Related-Key Boomerangs Distinguishers on Rijndael

In this section, we detail the way we implemented the model presented in Section 3.2 to fit the case of Rijndael. In the same way, we divided our search into two steps: in Step-1, we search for truncated boomerang distinguishers with minimal hamming weight whereas in Step-2, given the output Boolean differences of Step-1, we search for the instantiated boomerang distinguisher with the best probability. We describe in this Section each of these two steps.

6.2.1 Step1: Automatic Search of Related-Key Truncated Boomerang Distinguishers

The first step of a related-key boomerang attack may be divided into two parallel searches of related-key truncated differential characteristics (one for the upper trail and one for the lower trail) and some glue needs to be added for the middle part using the Boolean free variables propagation. For each differential byte δ_A (where $A \in \{X[i,j,k], SX[i,j,k], Y[i,j,k], Z[i,j,k], RK[i,j,k] : i \in [0;\ N_r[, j \in [0;\ 3], k \in [0;\ N_b[\})$, we define 4 Boolean variables, *i.e.*, $\Delta_{A_{\text{up}}}$, $\text{free}_{A_{\text{up}}}$, $\Delta_{A_{\text{lo}}}$, and $\text{free}_{A_{\text{lo}}}$. The meaning of these variables is the same as in [DDV20] and detailed in Section 3.2.

As Rijndael also has S-Boxes in the `KeySchedule`, we also introduce 4 Boolean variables for each `RoundKey` differential byte $\delta_{RK}[i,j,k]$ that passes through an S-Boxes (as defined in algorithm

1.1), denoted $\Delta_{SRK_{\mathsf{up}}}[i,j,k]$, $\mathsf{free}_{SRK_{\mathsf{up}}}[i,j,k]$, $\Delta_{SRK_{\mathsf{lo}}}[i,j,k]$, and $\mathsf{free}_{SRK_{\mathsf{lo}}}[i,j,k]$: these variables model the fact that there is an output difference and that this output difference is free of condition for the upper and lower trails, respectively.

Since Rijndael's `KeySchedule` is represented by a 2-dimensional matrix, we introduce the same Δ and free variables for WK_{up}, WK_{lo}, SWK_{up} and SWK_{lo} which correspond to the variables RK_{up}, RK_{lo}, SRK_{up} and SRK_{lo}. The RK and WK variables are linked by the equations:

$$\begin{aligned}
\Delta_{RK_{trail}}[i,j,k] &= \Delta_{WK_{trail}}[j,(i+1)\times Nb+k] \\
\mathsf{free}_{RK_{trail}}[i,j,k] &= \mathsf{free}_{WK_{trail}}[j,(i+1)\times Nb+k] \\
\Delta_{SRK_{trail}}[i,j,k] &= \Delta_{SWK_{trail}}[j,(i+1)\times Nb+k] \\
\mathsf{free}_{SRK_{trail}}[i,j,k] &= \mathsf{free}_{SWK_{trail}}[j,(i+1)\times Nb+k]
\end{aligned}$$

where *trail* can be either up or lo.

Constraints are added between Δ variables to model Rijndael operators, and we use the same constraints as those introduced in Model 4.1 and Model 4.2, except that these constraints are duplicated for the upper and the lower trail, respectively. For example, the constraint associated with `AddRoundKey` in Model 4.1 is:

$$\Delta_X[i+1,j,k] + \Delta_Z[i,j,k] + \Delta_{RK}[i,j,k] \neq 1$$

In our model, this constraint becomes:

$$\begin{aligned}
\Delta_{X_{\mathsf{up}}}[i+1,j,k] + \Delta_{Z_{\mathsf{up}}}[i,j,k] + \Delta_{RK_{\mathsf{up}}}[i,j,k] &\neq 1 \\
\Delta_{X_{\mathsf{lo}}}[i+1,j,k] + \Delta_{Z_{\mathsf{lo}}}[i,j,k] + \Delta_{RK_{\mathsf{lo}}}[i,j,k] &\neq 1
\end{aligned}$$

We do not detail these constraints here and refer the reader to Model 4.1 and Model 4.2.

Besides these constraints, we add the new constraints defined in Model 6.1:

- Constraints (B1) to (B5) relate free variables together: (B1) corresponds to `AddRoundKey`, (B2) to `ShiftRow`, (B3) to `MixColumns`, (B4) and (B5) to the `KeySchedule`. Note that for each round operation (`AddRoundKey`, `ShiftRow`, and `MixColumns`), we have one constraint in the encryption direction for the upper trail, and one constraint in the decryption direction for the lower trail. For the `KeySchedule`, there are also two constraints but they are both in the encryption direction because subkeys are all computed from the master key, in both trails.
- Constraints (B6) and (B7) define the S-Box rules that glue the two trails as done in [DDV20].
- Constraint (B8) defines the objective function *obj* that must be minimized (as we consider $-\log_2$ values). There are 6 tables implied in the computation of the objective function: `DDT`, $\texttt{DDT}^2$, `BCT`, `EBCT`, `LBCT`, and `UBCT`. The predicates used to choose the correct tables are given in Model 3.3. These predicates are extended to RK variables in

a straightforward way, by replacing X with RK. Each predicate $\texttt{isT}_{\texttt{X}}$ and $\texttt{isT}_{\texttt{RK}}$ (with $\texttt{T} \in \{\texttt{DDT}, \texttt{DDT}^2, \texttt{BCT}, \texttt{EBCT}, \texttt{LBCT}, \texttt{UBCT}\}$) is multiplied by the $-\log_2$ of the maximum probability of table T, denoted $\texttt{P}_{\texttt{T}}$.

Implementation Our Step-1 model has been implemented in MiniZinc [Net+07], which is a high-level and solver-independent language for modeling constraint satisfaction and optimization problems. MiniZinc models are then compiled into FlatZinc, a solver input language that is understood by a wide range of solvers (such as Choco [PFL16], Chuffed [CS14], or Picat-SAT [ZK16]). In our experiments, we have used Picat-SAT as it is the most efficient for our Step 1 problem.

6.2.2 Step 2: Instantiating the Related-Key Truncated Boomerang Distinguishers

In this section, we describe how to solve Step-2, which aims at computing the maximal probability of a related-key boomerang distinguisher corresponding to a given truncated distinguisher computed in Step 1 (as explained in the previous section). We first describe the mathematical model and then show how it may be easily implemented using a constraint programming language.

For each round $i \in [1;\ N_r[$[1], each row $j \in [0;\ 3]$ and each column $k \in [0;\ N_b[$, if $\mathsf{free}_{A_{\mathsf{up}}}[i,j,k]$ (resp. $\mathsf{free}_{A_{\mathsf{lo}}}[i,j,k]$) is true in the Step-1 solution then the corresponding differential byte $\delta_{A_{\mathsf{up}}}[i,j,k]$ (resp. $\delta_{A_{\mathsf{lo}}}[i,j,k]$) at Step-2 may take any value with uniform probability and is free of constraints. Hence we do not introduce differential byte δ_A for these truncated variables. Otherwise, when $\mathsf{free}_{A_{\mathsf{up}}}[i,j,k]$ (resp. $\mathsf{free}_{A_{\mathsf{lo}}}[i,j,k]$) is false, we introduce an integer variable $\delta_{A_{\mathsf{up}}}[i,j,k]$ (resp. $\delta_{A_{\mathsf{lo}}}[i,j,k]$) in the Step-2 model.

For S-Box variables, the possible values of this integer variable depends on the value of its associated Δ Boolean variable (assigned at Step 1): if $\Delta_{X_{\mathsf{up}}}[i,j,k]$ (resp. $\Delta_{X_{\mathsf{lo}}}[i,j,k], \Delta_{SX_{\mathsf{up}}}[i,j,k]$ and $\Delta_{SX_{\mathsf{lo}}}[i,j,k]$) is false, then $\delta_{X_{\mathsf{up}}}[i,j,k]$ (resp. $\delta_{X_{\mathsf{lo}}}[i,j,k], \delta_{SX_{\mathsf{up}}}[i,j,k]$ and $\delta_{SX_{\mathsf{lo}}}[i,j,k]$) is assigned to 0; otherwise, its set of possible values is $[1;\ 255]$. The same operation is done for the $\delta_{RK_{\mathsf{up}}}, \delta_{RK_{\mathsf{lo}}}, \delta_{SRK_{\mathsf{up}}}$ and $\delta_{SRK_{\mathsf{lo}}}$ variables.

Considering that `ShiftRow`, `MixColumns` and `AddRoundKey` are linear functions, it is possible to infer the free state of the $\delta_{Y_{\mathsf{up}}}, \delta_{Y_{\mathsf{lo}}}, \delta_{Z_{\mathsf{up}}}$ and $\delta_{Z_{\mathsf{lo}}}$ variables from the free state of $\delta_{X_{\mathsf{up}}}$, $\delta_{X_{\mathsf{lo}}}$, $\delta_{SX_{\mathsf{up}}}$, $\delta_{SX_{\mathsf{lo}}}$, $\delta_{RK_{\mathsf{up}}}$, $\delta_{RK_{\mathsf{lo}}}$, $\delta_{SRK_{\mathsf{up}}}$ and $\delta_{SRK_{\mathsf{lo}}}$ variables. Hence, the $\delta_{Y_{\mathsf{up}}}, \delta_{Y_{\mathsf{lo}}}, \delta_{Z_{\mathsf{up}}}$ and $\delta_{Z_{\mathsf{lo}}}$ differential variables are only introduced in Step-2 when they are not free.

Finally, we introduce integer variables that represent $-\log_2$ probabilities associated with S-boxes. For each round $i \in [1;\ N_r[$, and each row $j \in [0;\ 3]$ and column $k \in [0;\ N_b[$, we define an integer variable $p[i,j,k]$ which corresponds to the $-\log_2$ probability of crossing both the upper trail S-box (from $\delta_{\mathsf{Xup}}[i,j,k]$ to $\delta_{\mathsf{SXup}}[i,j,k]$ and the lower trail S-box (from $\delta_{\mathsf{SXlo}}[i,j,k]$ to $\delta_{\mathsf{Xlo}}[i,j,k]$). The values of Δ and free Boolean variables computed at Step-1 are used to determine which constraint must be used to link $p[i,j,k]$ variables with their corresponding differential variables as defined

[1] The distinguisher is computer from round 1 to round $N_r - 1$.

ROUNDS:

$\forall i \in [0; N_r - 2], \forall j \in [0; 3], \forall k \in [0; N_b[,$ (B1)

$$\mathsf{free}_{X_{\text{up}}}[i+1, j, k] = \mathsf{free}_{Z_{\text{up}}}[i, j, k] \vee \mathsf{free}_{RK_{\text{up}}}[i, j, k]$$
$$\mathsf{free}_{Z_{\text{lo}}}[i, j, k] = \mathsf{free}_{X_{\text{lo}}}[i+1, j, k] \vee \mathsf{free}_{RK_{\text{lo}}}[i, j, k]$$

$\forall i \in [0; N_r[, \forall j \in [0; 3], \forall k \in [0; N_b[,$ (B2)

$$\mathsf{free}_{Y_{\text{up}}}[i, j, k] = \mathsf{free}_{SX_{\text{up}}}[[i, j, P_{N_b}[j] + k \mod N_b]]$$
$$\mathsf{free}_{Y_{\text{lo}}}[i, j, k] = \mathsf{free}_{SX_{\text{lo}}}[[i, j, P_{N_b}[j] + k \mod N_b]]$$

$\forall i \in [0; N_r[, \forall j \in [0; 3], \forall k \in [0; N_b[,$ (B3)

$$\mathsf{free}_{Z_{\text{up}}}[i, j, k] = \mathsf{free}_{Y_{\text{up}}}[i, 0, k] \vee \mathsf{free}_{Y_{\text{up}}}[i, 1, k] \vee \mathsf{free}_{Y_{\text{up}}}[i, 2, k] \vee \mathsf{free}_{Y_{\text{up}}}[i, 3, k]$$
$$\mathsf{free}_{Y_{\text{lo}}}[i, j, k] = \mathsf{free}_{Z_{\text{lo}}}[i, 0, k] \vee \mathsf{free}_{Z_{\text{lo}}}[i, 1, k] \vee \mathsf{free}_{Z_{\text{lo}}}[i, 2, k] \vee \mathsf{free}_{Z_{\text{lo}}}[i, 3, k]$$

KEY:

$\forall i \in [0; N_r[, \forall j \in [0; 3], \forall k \in [0; N_b[,$ (B4)

$$\mathsf{free}_{RK_{\text{up}}}[i, j, k] = \mathsf{free}_{WK_{\text{up}}}[j, (i+1) \times N_b + k]$$
$$\mathsf{free}_{RK_{\text{lo}}}[i, j, k] = \mathsf{free}_{WK_{\text{lo}}}[j, (i+1) \times N_b + k]$$

$\forall j \in [0; 3], \forall \omega \in [0; N_b \times (N_r + 1)[,$ (B5)

$$\textbf{if } \omega \mod N_k = 0 : \mathsf{free}_{WK_{\text{up}}}[j, \omega] = \mathsf{free}_{WK_{\text{up}}}[j, \omega - N_k] \vee \mathsf{free}_{SWK_{\text{up}}}[(j+1) \mod 4, \omega - 1]$$
$$\textbf{else if } N_k > 6 \wedge k \mod N_k = 4 : \mathsf{free}_{WK_{\text{up}}}[j, \omega] = (\mathsf{free}_{WK_{\text{up}}}[j, \omega - Nk] \vee \mathsf{free}_{SWK_{\text{up}}}[j, \omega - 1])$$
$$\textbf{else} : \mathsf{free}_{WK_{\text{up}}}r[j, \omega] = (\mathsf{free}_{WK_{\text{up}}}[j, \omega - Nk] \vee \mathsf{free}_{WK_{\text{up}}}[j, \omega - 1])$$

$$\textbf{if } \omega \mod N_k = 0 : \mathsf{free}_{WK_{\text{lo}}}[j, \omega] = \mathsf{free}_{WK_{\text{lo}}}[j, \omega - N_k] \vee \mathsf{free}_{SWK_{\text{lo}}}[(j+1) \mod 4, \omega - 1]$$
$$\textbf{else if } N_k > 6 \wedge k \mod N_k = 4 : \mathsf{free}_{WK_{\text{lo}}}[j, \omega] = (\mathsf{free}_{WK_{\text{lo}}}[j, \omega - Nk] \vee \mathsf{free}_{SWK_{\text{lo}}}[j, \omega - 1])$$
$$\textbf{else} : \mathsf{free}_{WK_{\text{lo}}}r[j, \omega] = (\mathsf{free}_{WK_{\text{lo}}}[j, \omega - Nk] \vee \mathsf{free}_{WK_{\text{lo}}}[j, \omega - 1])$$

OVERALL CONSTRAINTS IN BOTH DIRECTIONS:

$\forall i \in [0; N_r[, \forall j \in [0; 3], \forall k \in [0; N_b[,$ (B6)

$$\mathsf{free}_{SX_{\text{up}}}[i, j, k] \implies \Delta_{X_{\text{up}}}[i, j, k], \quad \mathsf{free}_{X_{\text{lo}}}[i, j, k] \implies \Delta X_{\text{lo}}[i, j, k]$$
$$\mathsf{free}_{X_{\text{up}}}[i, j, k] \implies \mathsf{free}_{SX_{\text{up}}}[i, j, k], \quad \mathsf{free}_{SX_{\text{lo}}}[i, j, k] \implies \mathsf{free}_{X_{\text{lo}}}[i, j, k]$$
$$\mathsf{free}_{X_{\text{up}}}[i, j, k] + \mathsf{free}_{X_{\text{lo}}}[i, j, k] + \mathsf{free}_{SX_{\text{up}}}[i, j, k] + \mathsf{free}_{SX_{\text{lo}}}[i, j, k] \leq 2$$

$\forall j \in [0; 3], \forall \omega \in [0; N_b \times (N_r + 1)[$ such that $\texttt{isSbCol}(\omega)$ (B7)

$$\mathsf{free}_{SWK_{\text{up}}}[i, \omega] \implies \Delta_{WK_{\text{up}}}[i, \omega], \quad \mathsf{free}_{WK_{\text{lo}}}[i, \omega] \implies \Delta_{WK_{\text{lo}}}[i, \omega]$$
$$\mathsf{free}_{WK_{\text{up}}}[i, \omega] \implies \mathsf{free}_{SWK_{\text{up}}}[i, \omega], \quad \mathsf{free}_{SWK_{\text{lo}}}[i, \omega] \implies \mathsf{free}_{WK_{\text{lo}}}[i, \omega]$$
$$\mathsf{free}_{WK_{\text{up}}}[i, \omega] + \mathsf{free}_{SWK_{\text{lo}}}[i, \omega] + \mathsf{free}_{WK_{\text{up}}}[i, \omega] + \mathsf{free}_{SWK_{\text{lo}}}[i, \omega] \leq 2$$

where $\texttt{predicate isSBCol}(\omega) = (\omega \mod N_k = 0) \vee (N_k > 6 \wedge \omega \mod N_k = 4)$.

OBJECTIVE FUNCTION: (B8)

$$obj = \sum_{i=1}^{N_r-2} \sum_{j=0}^{3} \sum_{k=0}^{N_b-1} \left(\begin{array}{ccccc} \mathtt{P}_{DDT} \times \texttt{isDDT}_{\texttt{X}}[i,j,k] & + & \mathtt{P}_{DDT^2} \times \texttt{isDDT}_{\texttt{X}}{}^2[i,j,k] & + \\ \mathtt{P}_{BCT} \times \texttt{isBCT}_{\texttt{X}}[i,j,k] & + & \mathtt{P}_{UBCT} \times \texttt{isUBCT}_{\texttt{X}}[i,j,k] & + \\ \mathtt{P}_{EBCT} \times \texttt{isEBCT}_{\texttt{X}}[i,j,k] & & & \end{array} \right)$$
$$+ \sum_{i=1}^{N_r-1^{\star}} \sum_{j=0}^{3} \sum_{k=0}^{N_b-1^{\star}} \left(\begin{array}{ccccc} \mathtt{P}_{DDT} \times \texttt{isDDT}_{\texttt{RK}}[i,j,k] & + & \mathtt{P}_{DDT^2} \times \texttt{isDDT}_{\texttt{RK}}{}^2[i,j,k] & + \\ \mathtt{P}_{BCT} \times \texttt{isBCT}_{\texttt{RK}}[i,j,k] & + & \mathtt{P}_{UBCT} \times \texttt{isUBCT}_{\texttt{RK}}[i,j,k] & + \\ \mathtt{P}_{EBCT} \times \texttt{isEBCT}_{\texttt{RK}}[i,j,k] & & & \end{array} \right)$$

* where $i \times N_b + k$ is an S-Box column in the `KeySchedule`

Model 6.1: *Model linking together the* `free` *variables for a related-key boomerang computation.*

$$\texttt{isDDT}_\texttt{X}\texttt{up}(i,j,k) \implies (\delta_{X_{\text{up}}}[i,j,k], \delta_{SX_{\text{up}}}[i,j,k], p[i,j,k]) \in T_{DDT}$$
$$\texttt{isDDT}_\texttt{X}\texttt{lo}(i,j,k) \implies (\delta_{X_{\text{lo}}}[i,j,k], \delta_{SX_{\text{lo}}}[i,j,k], p[i,j,k]) \in T_{DDT}$$
$$\texttt{isDDT}^2_\texttt{X}\texttt{lo}(i,j,k) \implies (\delta_{X_{\text{up}}}[i,j,k], \delta_{SX_{\text{up}}}[i,j,k], p[i,j,k]) \in T_{DDT^2}$$
$$\texttt{isDDT}^2_\texttt{X}\texttt{up}(i,j,k) \implies (\delta_{X_{\text{lo}}}[i,j,k], \delta_{SX_{\text{lo}}}[i,j,k], p[i,j,k]) \in T_{DDT^2}$$
$$\texttt{isBCT}_\texttt{X}(i,j,k) \implies (\delta_{X_{\text{up}}}[i,j,k], \delta_{SX_{\text{lo}}}[i,j,k], p[i,j,k]) \in T_{BCT}$$
$$\texttt{isLBCT}_\texttt{X}(i,j,k) \implies (\delta_{X_{\text{up}}}[i,j,k], \delta_{X_{\text{lo}}}[i,j,k], \delta_{SX_{\text{lo}}}[i,j,k], p[i,j,k]) \in T_{LBCT}$$
$$\texttt{isUBCT}_\texttt{X}(i,j,k) \implies (\delta_{X_{\text{up}}}[i,j,k], \delta_{SX_{\text{up}}}[i,j,k], \delta_{SX_{\text{lo}}}[i,j,k], p[i,j,k]) \in T_{UBCT}$$
$$\texttt{isEBCT}_\texttt{X}(i,j,k) \implies$$
$$(\delta_{X_{\text{up}}}[i,j,k], \delta_{SX_{\text{up}}}[i,j,k], \delta_{X_{\text{lo}}}[i,j,k], \delta_{SX_{\text{lo}}}[i,j,k], p[i,j,k]) \in T_{EBCT}$$

Model 6.2: *Constraints that relate $p[i,j,k]$ integer variables with differential variables (using predicates defined in Model 3.3). For each table $t \in \{DDT^2, DDT, BCT, UBCT, EBCT, LBCT\}$, T_t denotes the set of all tuples with a non null probability. For example, T_{DDT} contains all triples $(\delta_{in}, \delta_{out}, p)$ such that $-\log_2(DDT(\delta_{in}, \delta_{out})) = p$ and $p \neq 0$.*

in Model 6.2. The same operation can be applied to the S-Box columns of the `KeySchedule` by introducing $p_{RK}[i,j,k]$ variables. $p_{RK}[i,j,k]$ is the probability for the `RoundKey` byte at round i, row j and column k to pass its S-Box. Knowing that not all the `RoundKey` columns go through an S-Box, we only introduce a $p_{RK}[i,j,k]$ variable when necessary.

δ_X, δ_{SX}, δ_Y, δ_Z, δ_K, and δ_{RK} variables are constrained with respect to `SubBytes`, `ShiftRow`, `MixColumns`, `AddRoundKey`, and `KeySchedule` as described in Model 4.3, using table constraints. However, this model is duplicated for the upper and lower trails, respectively.

The objective function is then the sum of all $p[i,j,k]$ and $p_{RK}[i,j,k]$ integer variables and the goal is to minimize this sum.

Implementation Our Step-2 model widely uses table constraints, which are constraints of the form $(x_1, \ldots, x_n) \in T$ where $x_1, \ldots, x_n$ are integer variables and T is a set of allowed tuples (of arity n). Table constraints are one of the main advantage of using **CP** because those tables can be defined on large alphabet and are directly handled by **CP** solvers.

This model has been implemented with the Choco CP library version 4.10.6 [PFL16].

6.2.3 Combining the two Steps

Step-1 is in fact composed of two different sub-problems: the first one, *Step1-Opt*, searches for the best possible objective value of *obj* (denoted *obj*$*$), and the second one, *Step1-Next*, searches for Step-1 solutions with the *obj* value fixed to *obj*$*$. As there are usually many Step-1 solutions, we do not compute all of them at once, but we enumerate them one at a time and, for each enumerated Step-1 solution, Step-2 is performed to search for the best related-key boomerang differential characteristic corresponding to it, as done in the previous Chapter. This search that interleaves *Step1-next* and Step 2 is iterated until finding the optimal related-key boomerang differential characteristic, or detecting that the optimal probability exceeds the block or key

exhaustive search according to the Rijndael instance. Note that it may be possible that the optimal related-key boomerang differential characteristic has more than $obj*$ active S-boxes (either because there is no Step-2 solution with $obj*$ active S-boxes, or because it is possible to have a larger probability with more active S-boxes). Hence, the interleaved process increases $obj*$ until proving optimality of the best found differential characteristic.

Note also that in the original paper [DDV20], the authors also propose a way to compute the clusters induced. One of the main differences between Rijndael and Skinny relies on the fact that the linear part of Skinny is composed of XOR whereas the one of Rijndael includes multiplication in a finite field. As stated in [CR15; DR06], it is out of computational reach to compute such clusters for the AES and thus Rijndael. So, due to the very high computational cost of our method without the cluster computation, we do not include any cluster in our approach.

6.3 Attacks

6.3.1 From the Distinguisher to the Attack

Once an efficient related-key boomerang distinguisher between rounds 1 and $N_r - 2$ is found, there exist several techniques to extend it to a key recovery attack (see for example [Don+21] for a complete survey). We focus here on the method proposed in [Zha+20] even if it concerns algorithms with linear key schedule but it could be adapted for algorithms with non linear key schedule. Thus, we will apply their techniques to recover master key bits in round 0 and round $Nr - 1$ that does not contain `MixColumns` operation. Due to the very high diffusion of Rijndael, we do not investigate to add more rounds at the beginning or at the end of the cipher. So, let us first introduce the technique described in [Zha+20].

The parameters on which the complexities of an attack are computed are the following ones:

- The distinguisher E_d with N_d rounds is placed in the middle of the attack. The input difference of the distinguisher is α whereas the output difference is δ.
- We add N_a rounds E_a at the beginning and N_f rounds E_f at the end.
- At the beginning, in the deciphering direction, for N_a rounds, the difference α is extended backward and with probability 1 to a truncated difference α' with r_a possibly active bits and $n - r_a$ inactive bits.
- At the end, in the ciphering direction, for N_f rounds, the difference δ is extended with probability one to a truncated difference δ', with r_f possibly active bits and $n - r_f$ inactive bits.

Then, the attack could work on $N_a + N_d + N_f$ rounds by guessing some key materials appearing before and after the distinguisher and by counting how many times the distinguishing property happens. The correct key bits have the higher counters. In the following, the guessed key bits at the beginning are denoted by m_a and the guessed key bits at the end are denoted by m_f.

The attack of [Zha+20] works as follows where s is the expected number of right pairs:

1. Build $y = \sqrt{s} \cdot 2^{n/2-r_a}/\sqrt{p^2q^2r}$ structures of 2^{r_a} plaintexts each, and store them with their associated plaintexts.
2. For each possible value of the m_a key bits:
   - Initialize 2^{m_f} key counters.
   - Partially encrypt each plaintext M_1 of each structure using the guessed m_a key bits up to the beginning of E_d. Add α to the computed value and decrypt it up to the plaintext, to obtain M_2. Construct the set S (of size $y \cdot 2^{r_a}$) given by: $S = \{(M_1, C_1, M_2, C_2)$ such as $E_a(M_1, K_1) \oplus E_a(M_2, K_2) = \alpha\}$.
   - Insert S into a hash table H indexed by the $n - r_f$ bits that are inactive in δ'. Each collision defines a quartet (C_1, C_2, C_3, C_4).
   - Use these quartets to determine the correct m_f key bits. The time complexity of this stage is denoted ϵ.

The time complexity of the attack is dominated by stage 2.(b) or stage 2.(d). The complexity, given in number of encryptions, for stage 2.(b) is

$$2^{m_a+r_a} \cdot y \cdot \mu = 2^{m_a+n/2} \cdot \sqrt{s} \cdot \frac{1}{\sqrt{p^2q^2r}} \cdot \mu$$

whereas the complexity of stage 2.(d) is

$$s \cdot 2^{m_a-n+2r_f}/(p^2q^2r) \cdot \epsilon$$

Then, the success probability of the related-key boomerang attack could be approximated by the success probability of a differential attack given in [Sel08]:

$$P_s = \Phi(\frac{\sqrt{sS_N} - \Phi^{-1}(1-2^{-h})}{\sqrt{S_N+1}}),$$

where S_N is the signal-to-noise ratio, so is equal to $p^2q^2r/2^{-n}$ and h is the advantage.

To adapt this attack to the case of a non-linear key-schedule, one has just to compute the correct quartet of keys before the attack. Then, the probability to have a correct quartet only depends on the probability of the S-boxes induced by the `KeySchedule` and could be directly computed from the distinguisher. And the probability of the distinguisher is thus computed without the probability appearing in the `KeySchedule`. Then, it could be maybe more considered as a weak key attack because all the keys could not provide a right quartet. If we denote by $P_{\texttt{KeySchedule}}$ the probability of the `KeySchedule` and by P_{E_d} the probability of the encryption path, then, the cost of this stage is about $4/P_{\texttt{KeySchedule}}$.

6.3.2 Results

All experiments are run on a virtual machine `Ubuntu 18.04.5 LTS x86_64` with an `Intel Xeon Gold 5118` processor and `32 Gio` of RAM. The requirements are : `Java 10.0.12 OpenJDK`, `Gradle 6.8`, `MiniZinc 2.5.5`, `Picat 3.1.2` and `Choco 4.10.6`. Each instance is run on a single thread.

We put a time out of 6 months. After, those 6 months, the results we obtained are summed up in Table A.1 in Appendix A for both Step-1 and Step-2 computations for the related-key boomerang distinguisher on Rijndael.

Even if our models could not reach the largest instances of Rijndael, we obtain the following results:

- $\text{Rijndael}_{128-160}$ with 7 rounds, best probability: 2^{-95}. The version with 8 rounds is not reachable.
- $\text{Rijndael}_{160-128}$ with 4 rounds, best probability: 2^{-18}. Rijndeal-160-128 with 5 rounds is not reachable.
- $\text{Rijndael}_{192-160}$ with 5 rounds, best probability: 2^{-73}. The version with 6 rounds is not reachable.

Moreover, we have those partial results (only Step 1 have finished to run) for:

- $\text{Rijndael}_{160-160}$ for 6 rounds with an upper bound equal to 2^{-118};
- $\text{Rijndael}_{160-192}$ for 7 rounds with an upper bound equal to 2^{-102};
- $\text{Rijndael}_{160-224}$ for 8 rounds with an upper bound equal to 2^{-114};
- $\text{Rijndael}_{160-256}$ for 9 rounds with an upper bound equal to 2^{-120};
- $\text{Rijndael}_{192-128}$ for 4 rounds with an upper bound equal to 2^{-36};
- $\text{Rijndael}_{192-192}$ for 6 rounds with an upper bound equal to 2^{-84}.

The Step 2 for each instance has taken between 8 and 15 days. For example, for $\text{Rijndael}_{192-160}$ for 6 rounds the computation for Step 2 has taken 8 days.

Figure 6.1 displays some statistics about computation times. The upper part of the Figure represents which Step over the three is the most time consuming while the lower part of the Figure represents the computation time. We can see that the Step-1 (Step1-Opt + Step1-Enum) step is the most time consuming in general, expected for 6 (over 37) instances. Moreover, we see that the instances where Step-2 is the most time consuming are not among the most difficult instances. Hence improvements should target Step-1 modelling and resolution to improve the overall performances.

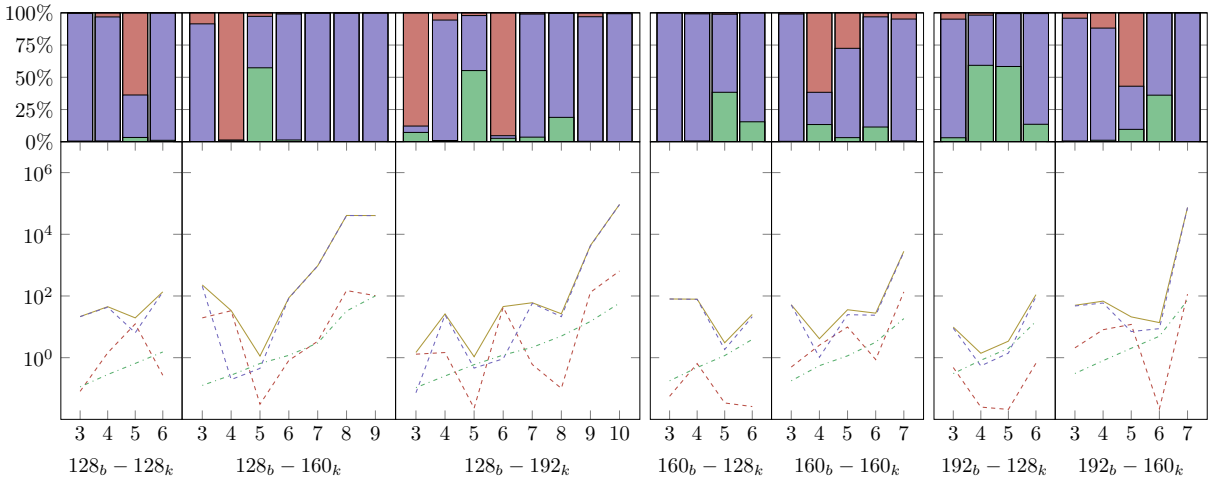


Figure 6.1: *Computation times for instances $X_b - Y_k$ where X is the block length and Y is the key length. The upper part of the chart represents the computation time proportion (in percentage of the total computation time) between Step-1-Opt (■), Step-1-Enum (■) and Step2-Opt (■). In the lower part, the chart represents the computation time (in seconds) for each step: Step1-Opt (-·-), Step1-Enum (- -) and Step2-Opt (- -) and the cumulative total time (—).*

6.3.3 Attack on 9 rounds of Rijndael$_{128-160}$

The 9 rounds attack of Rijndael$_{128-160}$ is presented in Figure A.7 in Appendix A. The distinguisher works for rounds 1 to 8 and has a probability of 2^{-56} for the encryption part and of 2^{-39} for the `KeySchedule`.

Thus, the attack implies the following parameter: $N_a = 1$, $r_a = 32$, $n - r_a = 96$, $N_f = 1$, $r_f = 32$, $n - r_f = 96$, $m_a = 32$, $m_f = 32$, $P_{E_d} = 2^{-56}$ and $P_{\texttt{KeySchedule}} = 2^{-39}$. Thus, applying the previous attack, with $s = 4$, we need to cipher 2^{61} structures of 2^{32} plaintexts. The complexity of the attack is dominated by stage 2.(b) and is equal to 2^{122} encryptions. The success probability of the attack is equal to 97,67%. The probability that a right quartet of keys is found is equal to 2^{39} and the complexity to find such a quartet is equal to 2^{41} encryptions. An other way to say that is that the number of keys that work for our attack is equal to $2^{160-39} = 2^{121}$.

6.4 Conclusion

In this chapter, we have presented the related-key boomerang distinguishers and the related-key boomerang attacks we obtained for some of the 25 instances of the block cipher Rijndael. Among our most significant results, we obtained a 9-round attack on Rijndael$_{128-160}$ which has 11 rounds.

However, the computational costs of our models are prohibitive for the largest Rijndael instances. So, we plan to try to improve those models and notably the way the Step-1 is computed to try to reach the missing instances.

In the next Chapter, we see how to adapt the same techniques to Feistel ciphers.

Chapter

7 Automatic rectangle attacks on Feistel ciphers: Application to WARP

Contents

As said before, Boomerang distinguishers [Wag99] were introduced at FSE'99 as a variant of differential distinguishers [BS91] taking advantage of the existence of short differentials of high probability. In its simplest version, the attacker sees the cipher E as the composition of two subciphers ($E = E_1 \circ E_0$) and makes use of a differential for each part.

If at first it was thought that these two differentials can be selected freely, following advances like [Mur11] showed that the interdependence should be carefully treated, as incompatibilities or better-than-expected probabilities might occur.

As a result, searching for the best boomerang distinguisher does not simply reduce to finding two differentials of high probability, and thus emerged a need for automated tools that would take into account the possible events in the middle, later formalized by the `BCT` [Cid+18] (for **SPN**s) and `FBCT` [Bou+20] (for Feistels) theories. Two techniques have been proposed recently to address this issue. In[HBS21], the authors proposed to give as input to a **MILP** model the size of the middle part (where dependencies happen) and to take into account one type of dependency (the so-called ladder switch [BK09]). A more precise approximation of the probability of the middle rounds was then obtained with the `BCT` framework or experimentally. A second technique has been proposed in [DDV20], that directly takes into account all the possible middle dependencies and does not require that the attacker specifies the size of the middle part.

If many papers start by looking for the best distinguisher to next turn it into an attack, better results might be obtained by taking into account the incidence of the distinguisher on the key recovery phase. An example of this was given in [ZDJ19] when building a rectangle attack on Deoxys, and further discussions were provided in [Qin+21] with results on Skinny. The latter presents an automated tool that takes the model provided in [DDV20] and adds relations to include the dominating factors of the key-recovery phase so that the resulting model directly looks for an optimization of the attack as a whole.

7.1 Introduction

In this chapter, we propose to study the security of the recently published block cipher WARP [Ban+20] against boomerang techniques. To do so, we start by showing how to adapt the model developed in [DDV20] to the case of Feistel ciphers, since the original tool was developed for **SPN** ciphers in general and for Skinny in particular.

Since the execution time of the simple model is exponential in the number of rounds, covering more than 20 rounds of WARP is out of reach. We thus propose several techniques to speed up the model and to guide it to the solutions. By counting different solutions with the same input and output differences, we were able to find a 23-round distinguisher that covers 2 more rounds than the best result to date.

Second, we show how to extend this model to search for rectangle attacks, following the method developed in [Qin+21]. This extra step ensures that both the key recovery part and the distinguisher are optimized together to reach a (close to) optimal attack as a whole. Finally, we describe a 26-round attack on WARP, again reaching the best result to date.

Our analysis shows that the designers' choice of positioning the key addition after the S-box in the Feistel round function (which is justified by the need to avoid complementation properties) turns in favour of the attacker.

Our results on WARP with a comparison with previous works are summarized in Table 7.1. The code of our tool is available at: `https://gitlab.inria.fr/lrouquet/boomerang-warp-fse-23`.

Outline. Section 7.2 is dedicated to the description of our model searching for boomerang distinguishers and to the discussion of our result on 23 rounds, that we can easily extend by 2 rounds. Our techniques to improve the execution time of the model are presented.

Section 7.3 shows how to turn the previous model into one searching for rectangle attacks and in particular how the position of the key addition turns favorable to the attacker, leading us to a 26-round attack.

Technique	Rounds	Probability	Time	Data	Mem.	Ref.
DC distinguisher	18	2^{-122}	-	-	-	[KY21]
DC distinguisher	20	$2^{-122.71}$	-	-	-	[TB21a]
ID distinguisher	21	1	-	-	-	[Ban+20]
Boomerang distinguisher	21	$2^{-121.11}$	-	-	-	[TB21a]
Boomerang distinguisher	23	2^{-124}	-	-	-	Subsection 7.2.2
Boomerang distinguisher	23	$2^{-115.59}$	-	-	-	[HNE22]
Differential attack	21	-	2^{113}	2^{113}	2^{72}	[KY21]
Differential attack	23	-	$2^{106.68}$	$2^{106.62}$	$2^{106.62}$	[TB21a]
Rectangle attack	24	-	$2^{125.18}$	$2^{126.06}$	$2^{127.06}$	[TB21a]
Rectangle attack	26	-	$2^{115.9}$	$2^{120.6}$	$2^{120.6}$	Section 7.3

Table 7.1: *Complexities of the existing results on* WARP. *Note that for all the distinguishers presented here we can add 2 rounds for free, see Subsection 7.2.2. ID = Impossible Differential, DC = Differential Characteristic.*

Previous results. The designers of WARP claimed single-key security and did not claim any security in related-key and known/chosen-key settings. They provided a rather comprehensive security analysis of their design, with a discussion of differential, linear, impossible differential, integral, meet-in-the middle and invariant subspace attacks. The longer distinguisher they mentioned is a 21-round impossible differential distinguisher.

To the best of our knowledge, two external cryptanalyses have been reported to date, both studying differential-based attacks. In the article [KY21] by Kumar and Yadav, a 21-round differential attack is presented (based on a 16-round differential characteristic), with a time and data complexity of 2^{113}. Concurrently with our work, a 23-round differential attack and a 24-round rectangle attack were reported by Teh and Biryukov in [TB21a]. The work done by Hadipour *et al.* [HNE22] was published after this chapter was written.

7.1.1 Boomerang attacks on Feistel ciphers

As shown for instance in the analysis of Sean Murphy [Mur11], the naive approximation of the probability of a boomerang distinguisher might turn wrong due to an incompatibility between the upper and the lower differentials. To solve this problem, Dunkelman *et al.* introduced the sandwich attack [DKS10] which adds a middle part E_m in the rewriting of E (namely $E = E_1 \circ E_m \circ E_0$) to isolate and study separately the rounds where the two differentials are interdependent. This middle part is called boomerang switch [BK09]; if we denote by r the probability that E_m connects the upper and the lower trails, the final probability of the distinguisher becomes p^2q^2r.

Computing the middle part probability. A recent line of works showed how to approximate the value of r with the use of various tables. The first to be introduced is the `BCT` [Cid+18], developed by Cid *et al.* to deal with 1-round boomerang switches in the case of SPN ciphers. In this Chapter we focus on the Feistel case and thus recall the `FBCT`, the `FBDT` and the `FBET`, introduced in [Bou+20]. The notation is recalled in Definition 7.1.

Definition 7.1: FBCT, FBDT and FBET [Bou+20]

Let S be a function from $\mathbb{F}_2^n$ to itself, and $(\Delta_i, \delta, \nabla_o, \alpha)$ be elements of $(\mathbb{F}_2^n)^4$. The Feistel Boomerang Connectivity Table (`FBCT`), Feistel Boomerang Difference Table (`FBDT`) and Feistel Boomerang Extended Table (`FBET`) of S are given by:

$$FBCT(\Delta_i, \nabla_o) = \#\left\{x \in \mathbb{F}_2^n \,|S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = 0\right\}.$$

$$FBDT(\Delta_i, \delta, \nabla_o) = \#\left\{x \in \mathbb{F}_2^n \,\middle|\, \begin{array}{l} S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = 0, \\ S(x) \oplus S(x \oplus \Delta_i) = \delta. \end{array}\right\}.$$

$$FBET(\Delta_i, \delta, \nabla_o, \alpha) = \#\left\{x \in \mathbb{F}_2^n \,\middle|\, \begin{array}{l} S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = 0, \\ S(x) \oplus S(x \oplus \Delta_i) = \delta, \\ S(x \oplus \Delta_i) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = \alpha. \end{array}\right\}.$$

The table used to compute the 1-round probability depends on which input and output differences of the S-box are fixed: for instance, the `FBCT` is chosen when only the inputs Δ_i and ∇_o are fixed (see Figure 7.1).

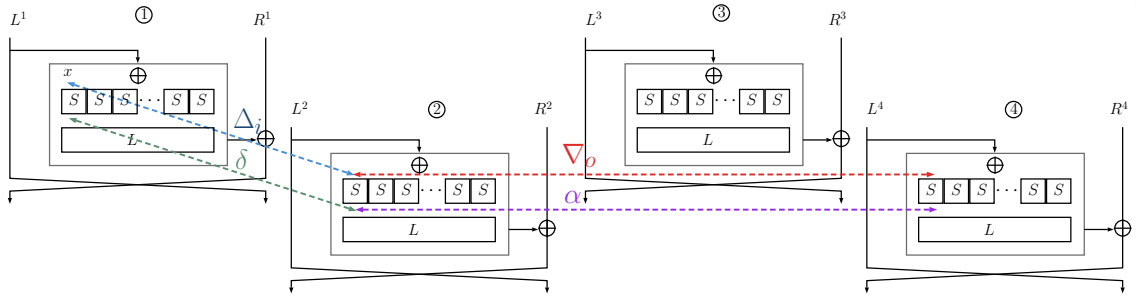


Figure 7.1: *View of the parameters of the tables: Δ_i is the input difference and δ is the output difference of S when looking at the difference between state ① and ②. ∇_o is the input difference of the same S-box S when looking at the difference between state ① and ③ and α is its output difference. We focus on the case where the differences are the same on parallel sides.*

From the distinguisher to the Attack Once an efficient boomerang distinguisher is found, there exist several techniques to extend it to a key recovery attack, as summarized for instance in [Don+21]. In this Chapter we focus on the technique devised by Zhao *et al.* in [Zha+20].

The parameters on which the complexities of an attack depend are shown on the right in Figure 7.2. The key recovery works by adding few rounds before and after the N_d distinguisher rounds E_d. We denote E_b the N_b rounds prepended and by E_f the N_f appended rounds. The attacker extends backward with probability one the input difference α, obtaining a truncated difference

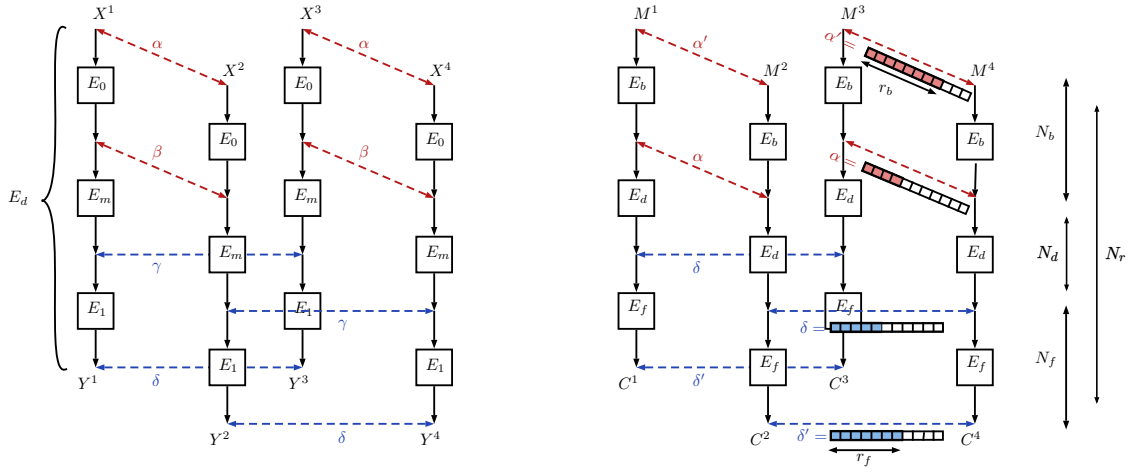


Figure 7.2: *Sandwich distinguisher (left) and setting for an attack, including the key recovery (right).*

α' with r_b possibly active bits and $n - r_b$ inactive bits. In the same way, δ is extended forward with probability one over the N_f rounds, giving a truncated difference equal to δ', with $n - r_f$ inactive bits.

Depending on the parameters, two factors might be dominating the time complexity; either the cost of stage 2.(b) or the cost of the last stage. Their time complexity is respectively $2^{m_b+r_b} \cdot y \cdot \mu = 2^{m_b+n/2} \cdot \sqrt{s} \cdot \frac{1}{\sqrt{p^2q^2r}} \cdot \mu$ and $s \cdot 2^{m_b-n+2r_f}/(p^2q^2r) \cdot \epsilon$ encryptions. Since stage 2.(b) does partial encryptions over E_b, μ can be approximated by $\frac{N_b}{N_b+N_d+N_f}$ while ϵ corresponds to the cost of gradually decrypting rounds to check the validity of a key guess, so we decide to approximate it by $\frac{1}{s}$.

Sucess probability. We use the formula devised in [Sel08] for differential cryptanalysis (and later used in the context of rectangle attacks) to evaluate the probability of finding the correct key:

$$P_s = \Phi\left(\frac{\sqrt{sS_N} - \Phi^{-1}(1 - 2^{-h})}{\sqrt{S_N + 1}}\right),$$

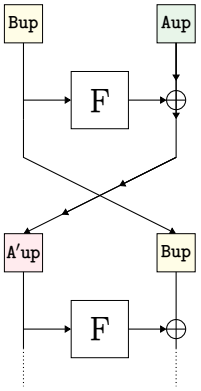
where S_N is the signal-to-noise ratio, so is equal to $p^2q^2r/2^{-n}$ and h is the advantage.

7.2 Automatic Search of Boomerang Distinguishers

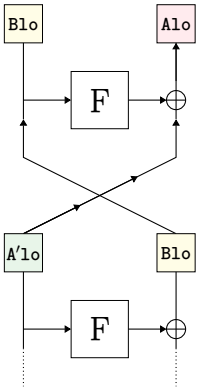
7.2.1 Automatic Search of Truncated Boomerang Distinguishers for Feistel Ciphers

This section describes how to build an automated tool that searches for truncated boomerang distinguishers for Feistel ciphers. Our method follows the idea developed by Delaune *et al.* for Skinny in [DDV20] but makes the required adjustments to fit the Feistel structure.

The model presented in [DDV20] for **SPN** ciphers treats differently the S-boxes of the lower and upper trail to take into account the direction in which they are computed. Given the specific



(a) *Feistel forward round.*



(b) *Feistel backward round.*

Figure 7.3: *Encryption (7.3a) and decryption (7.3b) procedure of a classical Feistel cipher. Note that the F function is never inverted and that the only difference comes from the direction in which the* XOR *is computed.*

property of Feistel ciphers (that are their own inverse) and as illustrated in figure 7.3, our model does not have to make this distinction, so we end up with the same constraints for the S-boxes in the upper and in the lower trail:

$$\begin{aligned} \mathsf{free}_{X_{\mathsf{up}}} &\implies \mathsf{free}_{SX_{\mathsf{up}}} \\ \mathsf{free}_{X_{\mathsf{lo}}} &\implies \mathsf{free}_{SX_{\mathsf{lo}}} \end{aligned} \tag{R1}$$

For the same reason we change the second constraint as follows:

$$\begin{aligned} \mathsf{free}_{SX_{\mathsf{up}}} &\implies \Delta_{X_{\mathsf{up}}} \\ \mathsf{free}_{SX_{\mathsf{lo}}} &\implies \Delta_{X_{\mathsf{lo}}} \end{aligned} \tag{R2}$$

Knowing which input and output differences are fixed for every S-box allows to select the correct table from Definition 7.1 to compute the associated boomerang probability. For instance, if the two input differences Δ_i and ∇_o are fixed while α and δ are free parameters, the required table is $FBCT(\Delta_i, \nabla_o)$.

For the model, it corresponds to the case where the input value of X_{up} and the input value of X_{lo} are fixed, so where $\mathsf{free}_{X_{\mathsf{up}}}$ and $\mathsf{free}_{X_{\mathsf{lo}}}$ are assigned to `false`. α and δ being free means that $\mathsf{free}_{SX_{\mathsf{up}}}$ and $\mathsf{free}_{SX_{\mathsf{lo}}}$ are equal to `true`, so we end up with constraint 7.1 that indicates when the $FBCT$ table is required to compute a 1-round probability.

$$\texttt{predicate isFBCT}(i,k) = \begin{pmatrix} \Delta_{X_{\mathsf{up}}}[i, 2\times k] \wedge \neg\mathsf{free}_{X_{\mathsf{up}}}[i, 2\times k] \wedge \mathsf{free}_{SX_{\mathsf{up}}}[i,k] \\ \wedge \\ \Delta_{X_{\mathsf{lo}}}[i, 2\times k] \wedge \neg\mathsf{free}_{X_{\mathsf{lo}}}[i, 2\times k] \wedge \mathsf{free}_{SX_{\mathsf{lo}}}[i,k] \end{pmatrix}$$

Model 7.1: *isFBCT predicate.*

Other tables are built in the same way and we obtain constraints F1 to F5 of Model 7.2 that select the correct table according to which variables are fixed or not.

To ensure that the probability of each S-box can be computed using one of the Feistel boomerang transitions, we add the following constraint:

$$\begin{aligned} \mathsf{free}_{X_{\mathsf{up}}} + \mathsf{free}_{SX_{\mathsf{lo}}} &\leq 1 \\ \mathsf{free}_{X_{\mathsf{lo}}} + \mathsf{free}_{SX_{\mathsf{up}}} &\leq 1 \end{aligned} \tag{R3}$$

While S-boxes are treated in the same way in the upper and lower trail, special care has to be taken to correctly propagate knowledge through the XOR operations. In the upper trail (Figure 7.3a) we have the following equation: $A' = F(B) \oplus A$ while for the lower trail (Figure 7.3b) we have: $A = F(B) \oplus A'$. This leads to the following distinction in the constraints:

$$\begin{aligned} \mathsf{free}_{\mathtt{A'}_{\mathsf{up}}} &= (\mathsf{free}_{F(\mathtt{B}_{\mathsf{up}})} \vee \mathsf{free}_{\mathtt{A}_{\mathsf{up}}}) \\ \mathsf{free}_{\mathtt{A}_{\mathsf{lo}}} &= (\mathsf{free}_{F(\mathtt{B}_{\mathsf{lo}})} \vee \mathsf{free}_{\mathtt{A'}_{\mathsf{lo}}}) \end{aligned} \tag{R4}$$

Constraints R1 to R3 are the core mechanisms of the boomerang model for Feistel ciphers. They must be applied on every S-box transition. Constraint R4 must be used on the parts of the state that are XORed together.

Resulting model. The complete model is provided in Model 7.2. Its first half is dedicated to the selection of the correct boomerang table. The second part starts with constraint F6 which ensures that the trails are active (*i.e.* that there is at least one difference in α and δ). Constraints F7 to F10 define the propagation from one round to the other, while the block of constraints F11 corresponds to the constraints R1, R2 and R3 explained at the beginning of this section and model the S-box transition. The model ends with the objective F12 given here in its naive form and that can be simplified as we now discuss.

Improvements

Weighted sum simplification. Given a model looking for differential characteristics, an upper bound of the probability is obtained by multiplying the number of active S-boxes found during Step-1 by the log_2 of the maximum probability of the transition of an active S-box, that is $UB = 2^{-\mathtt{P}_{DDT} \times \#SB}$.

Similarly, for a model looking for boomerang distinguishers, the upper bound has to take into account the various tables that are possible (the ones that can be selected in the first half of Model 7.2) and for each of them their maximum probability, denoted $\mathtt{P}_{DDT}, \mathtt{P}_{DDT^2}, ..., \mathtt{P}_{FBDT}$. The objective (and consequently the bound) thus corresponds to a weighted sum, as shown in F12, Model 7.2.

$$\texttt{predicate isDDT}(i,k) = \left(\begin{array}{c} \neg\Delta_{X_{\mathsf{up}}}[i,2\times k] \\ \wedge \\ \Delta_{X_{\mathsf{lo}}}[i,2\times k] \wedge \neg\mathsf{free}_{X_{\mathsf{lo}}}[i,2\times k] \wedge \neg\mathsf{free}_{SX_{\mathsf{lo}}}[i,k] \end{array}\right) \bigvee \left(\begin{array}{c} \Delta_{X_{\mathsf{up}}}[i,2\times k] \wedge \neg\mathsf{free}_{X_{\mathsf{up}}}[i,2\times k] \wedge \neg\mathsf{free}_{SX_{\mathsf{up}}}[i,k] \\ \wedge \\ \neg\Delta_{X_{\mathsf{lo}}}[i,2\times k] \end{array}\right) \tag{F1}$$

$$\texttt{predicate isDDT}^2(i,k) = \left(\begin{array}{c} \Delta_{X_{\mathsf{up}}}[i,2\times k] \wedge \mathsf{free}_{X_{\mathsf{up}}}[i,2\times k] \wedge \mathsf{free}_{SX_{\mathsf{up}}}[i,k] \\ \wedge \\ \Delta_{X_{\mathsf{lo}}}[i,2\times k] \wedge \neg\mathsf{free}_{X_{\mathsf{lo}}}[i,2\times k] \wedge \neg\mathsf{free}_{SX_{\mathsf{lo}}}[i,k] \end{array}\right) \bigvee \left(\begin{array}{c} \Delta_{X_{\mathsf{up}}}[i,2\times k] \wedge \neg\mathsf{free}_{X_{\mathsf{up}}}[i,2\times k] \wedge \neg\mathsf{free}_{SX_{\mathsf{up}}}[i,k] \\ \wedge \\ \Delta_{X_{\mathsf{lo}}}[i,2\times k] \wedge \mathsf{free}_{X_{\mathsf{lo}}}[i,2\times k] \wedge \mathsf{free}_{SX_{\mathsf{lo}}}[i,k] \end{array}\right) \tag{F2}$$

$$\texttt{predicate isFBCT}(i,k) = \left(\begin{array}{c} \Delta_{X_{\mathsf{up}}}[i,2\times k] \wedge \neg\mathsf{free}_{X_{\mathsf{up}}}[i,2\times k] \wedge \mathsf{free}_{SX_{\mathsf{up}}}[i,k] \\ \wedge \\ \Delta_{X_{\mathsf{lo}}}[i,2\times k] \wedge \neg\mathsf{free}_{X_{\mathsf{lo}}}[i,2\times k] \wedge \mathsf{free}_{SX_{\mathsf{lo}}}[i,k] \end{array}\right) \tag{F3}$$

$$\texttt{predicate isFBDT}(i,k) = \left(\begin{array}{c} \Delta_{X_{\mathsf{up}}}[i,2\times k] \wedge \neg\mathsf{free}_{X_{\mathsf{up}}}[i,2\times k] \wedge \neg\mathsf{free}_{SX_{\mathsf{up}}}[i,k] \\ \wedge \\ \Delta_{X_{\mathsf{lo}}}[i,2\times k] \wedge \neg\mathsf{free}_{X_{\mathsf{lo}}}[i,2\times k] \wedge \mathsf{free}_{SX_{\mathsf{lo}}}[i,k] \end{array}\right) \bigvee \left(\begin{array}{c} \Delta_{X_{\mathsf{up}}}[i,2\times k] \wedge \neg\mathsf{free}_{X_{\mathsf{up}}}[i,2\times k] \wedge \mathsf{free}_{SX_{\mathsf{up}}}[i,k] \\ \wedge \\ \Delta_{X_{\mathsf{lo}}}[i,2\times k] \wedge \neg\mathsf{free}_{X_{\mathsf{lo}}}[i,2\times k] \wedge \neg\mathsf{free}_{SX_{\mathsf{lo}}}[i,k] \end{array}\right) \tag{F4}$$

$$\texttt{predicate isFBET}(i,k) = \left(\begin{array}{c} \Delta_{X_{\mathsf{up}}}[i,2\times k] \wedge \neg\mathsf{free}_{X_{\mathsf{up}}}[i,2\times k] \wedge \neg\mathsf{free}_{SX_{\mathsf{up}}}[i,k] \\ \wedge \\ \Delta_{X_{\mathsf{lo}}}[i,2\times k] \wedge \neg\mathsf{free}_{X_{\mathsf{lo}}}[i,2\times k] \wedge \neg\mathsf{free}_{SX_{\mathsf{lo}}}[i,k] \end{array}\right) \tag{F5}$$

Model 7.2: *Model searching for truncated boomerangs on* WARP*, part 1/2: table selection.*

$$\begin{aligned}&\sum_{k=0}^{31}(\Delta_{X_{\text{up}}}[\text{first distinguisher round},k])\neq 0\\&\sum_{k=0}^{31}(\Delta_{X_{\text{lo}}}[\text{last distinguisher round},k])\neq 0\end{aligned}\tag{F6}$$

$\forall i \in$ all rounds*, $\forall k \in [0;\ \texttt{BR}/2[$

$$\begin{aligned}&\text{free}_{X_{\text{up}}}[i+1,\pi_{even}[k]]=\text{free}_{X_{\text{up}}}[i,2\times k]\\&\text{free}_{X_{\text{up}}}[i+1,\pi_{odd}[k]]=(\text{free}_{X_{\text{up}}}[i,2\times k+1]\vee\text{free}_{SX_{\text{up}}}[i,k])\end{aligned}\tag{F7}$$

$\forall i \in$ all rounds, $\forall k \in [0;\ \texttt{BR}/2[$

$$\begin{aligned}&\text{free}_{X_{\text{lo}}}[i,2\times k]=\text{free}_{X_{\text{lo}}}[i+1,\pi_{even}[k]]\\&\text{free}_{X_{\text{lo}}}[i,2\times k+1]=(\text{free}_{X_{\text{lo}}}[i+1,\pi_{odd}[k]]\vee\text{free}_{SX_{\text{lo}}}[i,k])\end{aligned}\tag{F8}$$

$\forall i \in$ all rounds, $\forall k \in [0;\ \texttt{BR}/2[$

$$\begin{aligned}&\Delta_{X_{\text{up}}}[i+1,\pi_{even}[k]]=\Delta_{X_{\text{up}}}[i,2\times k]\\&\Delta_{X_{\text{up}}}[i+1,\pi_{odd}[k]]+\Delta_{X_{\text{up}}}[i,2\times k+1]+\Delta_{X_{\text{up}}}[i,2\times k]\neq 1\end{aligned}\tag{F9}$$

$\forall i \in$ all rounds, $\forall k \in [0;\ \texttt{BR}/2[$

$$\begin{aligned}&\Delta_{X_{\text{lo}}}[i+1,\pi_{even}[k]]=\Delta_{X_{\text{lo}}}[i,2\times k]\\&\Delta_{X_{\text{lo}}}[i+1,\pi_{odd}[k]]+\Delta_{X_{\text{lo}}}[i,2\times k+1]+\Delta_{X_{\text{lo}}}[i,2\times k]\neq 1\end{aligned}\tag{F10}$$

$\forall i \in$ all rounds, $\forall k \in [0;\ \texttt{BR}/2[$

$$\begin{aligned}&\text{free}_{SX_{\text{up}}}[i,k]\implies\Delta_{X_{\text{up}}}[i,2\times k]\\&\text{free}_{SX_{\text{lo}}}[i,k]\implies\Delta_{X_{\text{lo}}}[i,2\times k]\\&\text{free}_{X_{\text{up}}}[i,2\times k]\implies\text{free}_{SX_{\text{up}}}[i,k]\\&\text{free}_{X_{\text{lo}}}[i,2\times k]\implies\text{free}_{SX_{\text{lo}}}[i,k]\\&\text{free}_{X_{\text{up}}}[i,2\times k]+\text{free}_{SX_{\text{lo}}}[i,k]\leq 1\\&\text{free}_{X_{\text{lo}}}[i,2\times k]+\text{free}_{SX_{\text{up}}}[i,k]\leq 1\end{aligned}\tag{F11}$$

$$obj=\sum_{i\in^{\text{distinguisher}}_{\text{rounds}}}\sum_{k=0}^{15}\left(\begin{array}{lll}\texttt{P}_{DDT}\times\texttt{isDDT}[i,k] & + & \texttt{P}_{DDT^2}\times\texttt{isDDT}^2[i,k] & +\\ \texttt{P}_{FBCT}\times\texttt{isFBCT}[i,k] & + & \texttt{P}_{FBDT}\times\texttt{isFBDT}[i,k] & +\\ \texttt{P}_{FBET}\times\texttt{isFBET}[i,k] & & & \end{array}\right)\tag{F12}$$

$$\texttt{minimize}\ obj$$

*: all rounds include E_b, E_d and E_f rounds of Figure 7.2 while distinguisher rounds only include E_d rounds.

Model 7.2: *Model searching for truncated boomerangs on* WARP*, part 2/2.* π_{even} *and* π_{odd} *correspond to the subparts of the* π *permutation for even and odd inputs only.* BR *is the number of branches of the cipher, so is equal to 32 in the case of* WARP*.*

Even if the semantic remains the same, reordering this weighted sum may have a huge impact on the solving time.
The first simplification that can be done corresponds to cases where a table has a maximum probability of 1. In such a setting, the table can simply be ignored during Step-1. This happens for the FBCT of WARP. The second simplification occurs when different tables have the same maximum probability, in which case they can be grouped by their respective maximum probabilities. For WARP, such an equality happens for the DDT, the FBDT and the FBET which have the same maximum denoted $\mathtt{P}_{isTable}$. Also, the $\mathtt{DDT}^2$ can be handled by counting them twice more than the DDT in the sum. Thus, the *obj* function can be simplified as follows:

$$obj = \sum_{i \in \substack{\text{distinguisher} \\ \text{rounds}}} \sum_{k=0}^{15} \left(\begin{array}{ll} \mathtt{P}_{isTable} \times (\mathtt{isDDT}[i,k] \vee \mathtt{isFBDT}[i,k] \vee \mathtt{isFBET}[i,k] \vee \mathtt{isDDT}^2[i,k]) & + \\ \mathtt{P}_{isTable} \times \mathtt{isDDT}^2[i,k] & \end{array} \right)$$

In addition to this, since there is a single maximum probability for all the tables (except for the FBCT removed previously), we can rewrite the weighted sum as:

$$obj = \mathtt{P}_{isTable} \times \sum_{i \in \substack{\text{distinguisher} \\ \text{rounds}}} \sum_{k=0}^{15} \Big(\quad (\mathtt{isDDT}[i,k] \vee \mathtt{isFBDT}[i,k] \vee \mathtt{isFBET}[i,k] \vee \mathtt{isDDT}^2[i,k]) + \mathtt{isDDT}^2[i,k] \quad \Big)$$

Finally, once the objective function is simplified we can use the Quine-McCluskey algorithm to create a minimized Boolean predicate $\mathtt{isTable}[i,k] = (\mathtt{isDDT}[i,k] \vee \mathtt{isFBDT}[i,k] \vee \mathtt{isFBET}[i,k] \vee \mathtt{isDDT}^2[i,k])$ and use it in the weighted sum:

$$obj = \mathtt{P}_{isTable} \times \sum_{i \in \substack{\text{distinguisher} \\ \text{rounds}}} \sum_{k=0}^{15} ((\mathtt{isTable}[i,k]) + \mathtt{isDDT}^2[i,k])$$

Incremental search. In our model, the objective function non-strictly decreases as the number of rounds r increases, since if we do not add an active S-box we will have the same optimal probability while if we do add one S-box the probability will always be equal (if the maximal possible probability of the table is 1) or lower (if the maximum possible probability is strictly less than 1). We can use this information to lower bound the objective function for $r+1$ rounds when we know the optimal probability for r rounds:

$$\mathtt{P}_{isTable} \times \sum_{i'=N_b}^{N_b+r} \sum_{k=0}^{15} ((\mathtt{isTable}[i',k]) + \mathtt{isDDT}^2[i',k]) \geq obj_r$$

This observation allows to give the model additional information about the minimum bound, which makes it stop the search earlier and therefore save execution time.

Forcing the pattern of the solution. The previous model allows to find distinguishers for up to 20 rounds of WARP (in both Step-1 and Step-2), but more rounds are out of reach as the computation time grows exponentially in the number of rounds.

However, we found out that all the optimal solutions returned for $N_d = 15$ to $N_d = 20$ have a specific pattern of the form `1-1-0-1-1`, that is contain a sequence of 5 rounds with 1 active S-box in the first round, 1 active S-box in the second round, 0 in the third one, and so on.

While we cannot formally prove that this pattern is going to appear in the optimal solutions for 21 rounds and more, we believe that there are high chances that it does, so we decided to add a Step-1 constraint that forces the solutions to follow this specific pattern. This assumption seems reasonable as we did not observe a break in the probability chart.

Formally, it gives (note that we do not fix the position where the pattern appears):

$$\bigvee_{i=N_b}^{N_b+N_d-5} \left(\bigwedge_{i'=i}^{i+4} \left(\sum_{k=0}^{15} \Delta_{X_{\mathsf{up}}}[i', 2 \times k] = \texttt{pattern}[i'] \right) \right) \text{ with } \texttt{pattern} = [1,1,0,1,1]$$

Instantiating the Truncated Boomerang Distinguishers

As mentioned before, we decompose our analysis into two steps. The first one (described above) implements the search of truncated boomerang distinguishers and is written in Picat SAT [ZK16], the **SAT** compiler in the Picat system. Each S-box of each round is associated to 6 bits: 3 for the upper trail and 3 for the lower trail. They indicate if an S-box is active or not, if the S-box input is free or not and if the S-box output is free or not.

The second step looks for concrete instantiations of the previous truncated solutions. It is written in the open source Java constraint programming library Choco [PFL16]. This step is also inspired from the one of [DDV20].

CP Model

The Constraint Programming model of Step-2 takes as input the results of Step-1 to know the general shape of the distinguisher, in particular which nibbles are inactive. To transform a truncated solution into a concrete one we need to assign values to the nibbles. For each pair of nibble abstraction $(\Delta_X, \mathsf{free}_X)$ we create a variable δ_X whose domain depends on the Step-1 solution:

$$\delta_X \in \begin{cases} [0,16[& \texttt{if } \mathsf{free}_X = \texttt{true} \\ \{0\} & \texttt{else if } \Delta_X = \texttt{false} \\ [1;16[& \texttt{else} \end{cases}$$

In the same way, δ_{SX} variables are created depending on the value of the pair $(\Delta_X, \mathsf{free}_{SX})$. As the free variables can take any value from the nibble domain and are not constrained by the

model we ignore them in Step-2.

The exact probability of each round is computed by using table constraints, which are tables containing all the possible (or impossible) transitions. For instance, describing that $x \oplus y = z$ for binary variables could be done with the table constraint:

$$(x, y, z) \in Tab_{\oplus} \text{ where } Tab_{\oplus} = \{(0,0,0), (0,1,1), (1,0,1), (1,1,0)\}.$$

One table constraint is created for each of the tables appearing in the probability computation (DDT, DDT^2, FBCT, FBDT and FBET), and we also make one to handle the XOR over nibbles. In addition to indicating the valid transitions, it also contains a third variable corresponding to the absolute value of the base 2 logarithm of its probability. The truncated solutions outputted by Step-1 completely define which table is used.
For example if we have: $(\Delta_{X_{\text{up}}}[i, 2 \times k] \wedge \neg \text{free}_{X_{\text{up}}}[i, 2 \times k] \wedge \neg \text{free}_{SX_{\text{up}}}[i, k] \wedge \neg \Delta_{X_{\text{lo}}}[i, 2 \times k])$, which corresponds to a DDT transition in the upper trail, we add the constraint: $(\delta_{X_{\text{up}}}[i, 2 \times k], \delta_{SX_{\text{up}}}[i, k], p[i, k]) \in Tab_{\text{DDT}}$. The objective function is then the following sum:

$$obj_2 = \sum_{i \in \substack{\text{distinguisher} \\ \text{rounds}}} \sum_{k=0}^{15} \Big(p[i, k]\Big)$$

Combining the two Steps. Step-1 is composed of two different strategies: $Step1 - Opt$ that searches for the truncated boomerang with the best objective obj and $Step1 - Next$ that enumerates one by one the solutions that reach this minimum obj. The best obj value is an upper bound (UB) that can not always be reached as Step-1 is an abstraction (some truncated solutions may not have concrete instances). The lower bound (LB) is fixed to 0 since it is the lowest possible value. For a given number of rounds, we first run $Step1 - Opt$ to find UB, and we next interleave $Step1 - Next$ with Step-2 to obtain a concrete boomerang with the best probability. Once done, we update LB with this new value and repeat the process for all the Step-1 optimal solutions. If a Step-2 is returned it means that the solution has a better probability than the given LB, so we update LB and we continue the search. If no Step-1 is found it means that we have already seen all the Step-1 solutions that can match UB, so we degrade UB and we continue the search with $Step1 - Opt$. If $LB = UB$ we have found the best solution available. Note that the model generates many solutions in Step-1 and most of the time we stop the search when $LB = UB$ instead of enumerating all possible Step-1 solutions.

Clusters.

Once the optimal solution has been found for Step-1 and Step-2 (this solution is hereafter denoted $< S_1opt, S_2opt >$), the goal is to obtain a better approximation of the actual probability of the boomerang distinguisher by considering clusters. Indeed, the solution returned by Step-2 has most of its S-box transitions fixed, while the only differences that matter when considering a boomerang distinguisher are the input and output differences (α and δ in Figure 7.2).

To get closer to this actual probability, we start by generating multiple Step-1 solutions that have their truncated differences in the first round and in the last round equal to the ones of S_1opt. The objective is to take into account many solutions that are all different one from the others. We need to be carefull about what being different means in our context as the situation is a bit more subtil than for a differential attack (for which the only two possible S-box status are "active" and "inactive").

In our model, we have the special case of the `free` S-box inputs that can take any value uniformly. If we focus on one particular S-box, the case of a fixed active input difference can be seen as contained in this one, so we must not count these cases as two independent ones. To be on the safe side, we choose to consider that two Step-1 solutions are different if at least one of their S-boxes is not free and inactive in one while it is not free and active in the other.

We thus search for the Step-2 solutions corresponding to these, with the additional condition that the nibble differences in the first and last rounds are the one of S_2opt. We sum over the different values of obj_2 in a variable called OBJ. To avoid counting solutions with too low probabilities we leave out the solutions of probability lower than $2^{-10} \times p(S_1opt)$ for both Step-1 and Step-2. Still, the large number of solutions forces us to set a limit on the number of solutions enumerated in Step-2 for a given Step-1 solution: we set this limit to 2^{20}.

To check the validity of our approach, we wanted to compare the result of the simple model and of the model with the clusters with what can be experimentally observed. To do so, we decided to pick a Feistel cipher with a smaller block size than WARP with the hope that the clustering effect would be easier and faster to observe. We selected the 64-bit block cipher TWINE [Suz+13] reduced to 12 rounds. We first computed its experimental probability by fixing the differences α and δ and we counted how many times the boomerang comes back. For the considered example, we obtained a probability close to $2^{-25.68}$ when testing 2^{30} plaintexts with 2^4 keys with a Rust experiment (Figure 7.4). The corresponding optimal Step-1 solution has a probability of 2^{-26} and is instantiated in Step-2 with a trail of probability 2^{-26}. When aggregating solutions with the same input and output differences we obtain an approximation of the distinguisher probability of $2^{-25.15}$ which is close to the experimental result, albeit slightly exceeding it. We also test our model for WARP (Figure 7.3). These tests were realized without taking the cluster effect into account since the gain is starting to be observable for too small probabilities.

Rounds	Cluster	obj_{step_2}	gain
18	2^{-54}	2^{-58}	2^{4}
19	2^{-66}	2^{-70}	2^{4}
20	2^{-76}	2^{-84}	2^{8}
21	2^{-96}	2^{-104}	2^{8}
22	2^{-108}	2^{-120}	2^{12}
22	2^{-124}	2^{-140}	2^{16}

Table 7.2: *Best distinguishers found after 2 days when summing up boomerang characteristics in the same cluster for the best Step-1 solutions on* WARP.

7.2.2 23-round Distinguisher on WARP

Implementation details. The Step-1 is written in MiniZinc [Net+07] and runs on Picat [ZK16] which uses the Lingeling solver [Bie] under the hood, the Step-2 is written in Choco [PFL16] version 4.10.6 which is a dedicated framework for Constraint Programming running on the Java Virtual Machine. We choose the Picat solver to solve the Step-1 as it is a **SAT** solver, so is especially suited to problems on Boolean formulae. Previous works like [Lib+21] have shown that Picat has good performances on multiple Step-1 models. Since the Step-2 contains a lot of table constraints, it appears that **CP** solvers are more adapted. The experiments are run on a virtual machine `Ubuntu 18.04.5 LTS x86_64` with an `Intel Xeon Gold 5118` processor and `32 Gio` of RAM. The requirements are : `Java 10.0.12 OpenJDK`, `Gradle 6.8`, `MiniZinc 2.5.5`, `Picat 3.1.2` and `Choco 4.10.6`. Each instance is run on a single thread.

We found 20 instances (from 3 rounds up to 22 rounds) with a probability better than 2^{-128}. They all took less than 48 hours to solve.

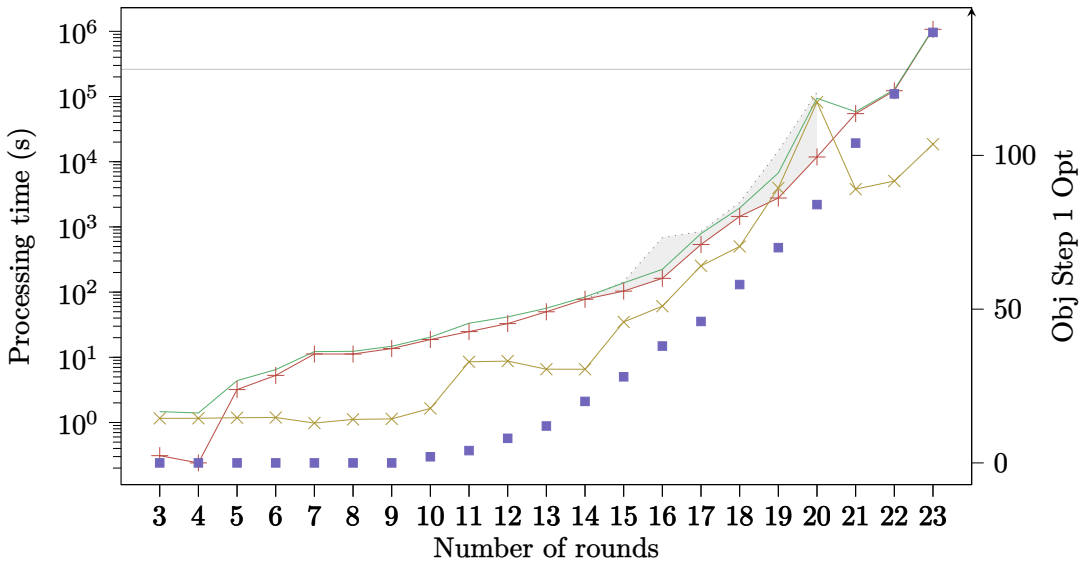


Figure 7.4: *Execution time of Step-1 Opt enforcing the `1-1-0-1-1` pattern (in seconds) (+), execution time of Step-1 Enum + Step-2 Opt (×), Total time (—). Best probability found with Step-1 Opt (■). Time of Step-1 Opt without the `1-1-0-1-1` pattern (···). The black line corresponds to the probability* 2^{-128}.

Without taking into account the clusters, the longest distinguisher that can be obtained is a

22-round boomerang of probability 2^{-120}. By summing up several boomerang trails inside one 23-round solution we are able to build a distinguisher of 23 rounds with probability 2^{-124}. By exploiting the position of the key addition, it can easily be extended to a 25-round distinguisher, thanks to the easy trick that we now present[1].

The distinguisher is depicted in Appendices B.3.1 and B.3.2. Its 32 nibbles of input and output differences are given by:

$$\begin{aligned}\alpha &= \texttt{57 00 00 07} \quad \texttt{00 00 57 57} \quad \texttt{07 57 00 07} \quad \texttt{00 00 57 00}\\ \delta &= \texttt{70 05 00 70} \quad \texttt{05 00 70 70} \quad \texttt{00 00 70 70} \quad \texttt{00 00 00 05}\end{aligned}$$

(note that for simplicity we kept the last round permutation in the figures and here). To exploit this distinguisher, an attacker would ask for the encryption of a large number of pairs M^1, M^2 verifying $M^1 \oplus M^2 = \alpha$, and build two new ciphertexts by computing $C^3 = E(M^1) \oplus \delta$ and $C^4 = E(M^2) \oplus \delta$. She would then ask for the corresponding plaintexts and check if $E^{-1}(C^3) \oplus E^{-1}(C^4) = \alpha$.

Since the round keys are added after the application of the S-boxes, an attacker can compute the difference entering the second round of the upper trail, and similarly the difference at the input of the S-boxes of the penultimate round of the lower trail. This easily leads to an extension of two rounds of any boomerang distinguisher. The attacker starts by picking a random message $M^1 = M^1[0], M^1[1], \cdots, M^1[31]$, and computes M^2 according to the difference she wants to observe one round later. For instance, it would give the begining of M^2 to be

$$M^2 = M^1[0], M^1[1], M^1[2] \oplus \texttt{0x7}, M^1[3] \oplus \texttt{0x5} \oplus S(M^1[2]) \oplus S(M^1[2] \oplus \texttt{0x7}), \cdots$$

A similar idea gives C^3 in function of C^1 and C^4 in function of C^2, and the boomerang returns if M^3 and M^4 verify

$$M^4 = M^3[0], M^3[1], M^3[2] \oplus \texttt{0x7}, M^3[3] \oplus \texttt{0x5} \oplus S(M^3[2]) \oplus S(M^3[2] \oplus \texttt{0x7}), \cdots .$$

7.3 Automatic Search of Rectangle Attacks

As already discussed in [ZDJ19] in the case of Deoxys–BC, the best rectangle distinguishers do not always lead to the best attacks, and choosing a sub-optimal (in terms of probability) distinguisher might allow to cover more rounds in the key recovery phase, and then to attack a bigger version of the cipher.

Following this idea, Lingyue Qin *et al.* proposed an automatic model that directly searches for an attack [Qin+21] by taking into account the dominating factors of the key-recovery step. The model simply minimizes the time complexity of the attack instead of maximizing the probability of the distinguisher, while making sure that the data complexity does not exceed the full codebook.

[1] Note that a similar trick can be used to extend the 21-round impossible differential distinguisher proposed by the designers of WARP to a 23-round distinguisher.

Rounds	Model	Experiment	Number of tries
3	2^{0}	2^{0}	$2^{4} \times 16$
4	2^{0}	2^{0}	$2^{4} \times 16$
5	2^{0}	2^{0}	$2^{4} \times 16$
6	2^{0}	2^{0}	$2^{4} \times 16$
7	2^{0}	2^{0}	$2^{4} \times 16$
8	2^{0}	2^{0}	$2^{4} \times 16$
9	2^{0}	2^{0}	$2^{4} \times 16$
10	2^{-2}	$2^{-1.93}$	$2^{6} \times 16$
11	2^{-4}	$2^{-3.94}$	$2^{8} \times 16$
12	2^{-8}	$2^{-6.39}$	$2^{12} \times 16$
13	2^{-12}	$2^{-8.80}$	$2^{16} \times 16$
14	2^{-20}	$2^{-18.02}$	$2^{24} \times 16$
15	2^{-28}	$2^{-25.65}$	$2^{28} \times 16$
16	2^{-38}	$2^{-35.65}$	$2^{36} \times 11$[1]

Table 7.3: *The experimental evaluation of our Model on* WARP*. The table gives the experimental probability of the distinguishers computed by our model compared to their expected value. The number of tries is the number of pairs of plaintext × the number of key tried. As the reader can see, the model is close to the experimental evaluation. The code used for the experimental evaluation passes the test vectors given by the* WARP *authors. The source code is available at* https://gitlab.inria.fr/lrouquet/boomerang-distinguisher-experimental-evaluation-on-WARP*.* [1] *The computation time was too long and stopped after 11/16 keys.*

Rounds	Model	Experiment	Number of tries
3	2^{-0}	2^{-0}	$2^{4} \times 16$
4	2^{-0}	2^{-0}	$2^{4} \times 16$
5	2^{-0}	2^{-0}	$2^{4} \times 16$
6	2^{-2}	$2^{-1.84}$	$2^{6} \times 16$
7	2^{-4}	$2^{-3.56}$	$2^{8} \times 16$
8	2^{-6}	$2^{-5.90}$	$2^{10} \times 16$
9	2^{-8}	$2^{-7.81}$	$2^{12} \times 16$
10	2^{-14}	$2^{-13.66}$	$2^{18} \times 16$
11	2^{-20}	$2^{-17.54}$	$2^{24} \times 16$
12	2^{-26}	$2^{-25.68}$	$2^{30} \times 16$
13	2^{-34}	$2^{-32.93}$	$2^{38} \times 16$

Table 7.4: *The experimental evaluation of our Model on* TWINE_{80}*. The table gives the experimental probability of the distinguishers computed by our model compared to their expected value. The number of tries is the number of pairs of plaintext × the number of key tried. As the reader can see, the model is really close to the experimental evaluation. The code used for the experimental evaluation passes the test vectors given by the* TWINE *authors. The source code is available at* https://gitlab.inria.fr/lrouquet/boomerang-distinguisher-experimental-evaluation-on-TWINE*.*

The number of rounds on which is run the model is gradually increased until the returned time complexity exceeds the cost of an exhaustive search of the secret key. This technique turned effective as it leads to improved attacks on the **SPN** ciphers Skinny and ForkSkinny [Qin+21].

In this section we study how to apply a similar idea to find good attack parameters for WARP and show that when considering the attack technique introduced by Zhao *et al.* in [Zha+20] there are at least two possible improvements in comparison to a variant of WARP with the key addition positioned before the S-box. The first one is the reduction of the value of m_b (the number of key bits that have to be guessed in the upper rounds), and the second one is the potential growth of the number of filtering bits, that is of $n - r_f$.

These two improvements are crucial since the two predominating factors of the time complexity of the attack of [Zha+20] are $2^{m_b+n/2} \cdot \sqrt{s} \cdot \frac{1}{\sqrt{p^2q^2r}} \cdot \frac{N_b}{N_b+N_d+N_f}$ and $2^{m_b-n+2r_f}/(p^2q^2r)$.

Taking Advantage of the Structure of WARP

Reduction of m_b. To understand the first point, we look at a simple example that considers $N_b = 3$ rounds of key recovery prepended to the distinguisher, where α, the top difference of the boomerang distinguisher, has only two active nibbles, in position 1 and 4 (see the bottom of Figure 7.5).

To determine the value of r_b (the number of active bits in the plaintext structures), an attacker starts by propagating backwards the difference α to know which nibbles might be active and which are inactive for sure. This process is rather straightforward, and in our example it returns $r_b = 32$ active nibbles (denoted in green in Figure 7.5).

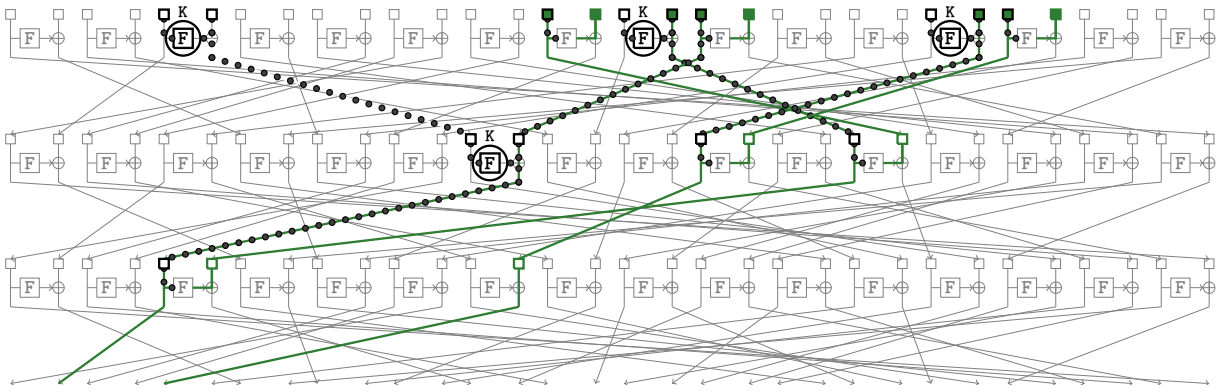


Figure 7.5: *Determining the required key bits to apply [Zha+20] over 3 rounds.*

The next step is the determination of the key bits that are required to compute M^2 from M^1. In the description given in [Zha+20], the attacker starts from M^1, computes partially the state at the input of the boomerang distinguisher so that she can add α to it, and decrypts the result to get M^2. Put differently, it can be seen as guessing the necessary key bits to uniquely determine the difference to add to M^1 to get M^2 knowing that after N_b rounds the internal states differ of α, which can also be seen as being able to uniquely propagate α backwards.

Consequently, the attacker starts from α, and takes note of the information needed: in round 2,

the exact difference at the output of the second S-box has to be uniquely determined, so the input *value* of this S-box is needed. This is denoted by the dashed lines in Figure 7.5. Knowing this value in round 2 implies that the two inputs of the xor number 6 of round 1 are known, which in particular forces to make a guess on the key nibble added after the S-box. The rest of the backward propagation is processed similarly, and we obtain that 4 nibbles of key are required in total. In the case where round keys are added before the S-boxes, this computation would have return a total of 10 nibbles of key (we consider here that the round keys are independent).

Improved filtering process. For some specific shapes of output difference δ', the attacker is able to increase the number of bits on which she looks for collision in step number 2(c) of the attack. An example of this is given in Figure 7.6 with $N_f = 2$: by counting the number of active nibbles at the output, we obtain $r_f = 21 \times 4 = 84$, which means that the hash table is used to look for collisions over $2 \times (128 - 84) = 88$ bits.

In our example, the F functions number 4 to 9, 12 and 13 in the last round all have a similar difference pattern where the input of the S-box ($X[i, 2k]$) is active while the right part ($X[i, 2k+1]$) is not.

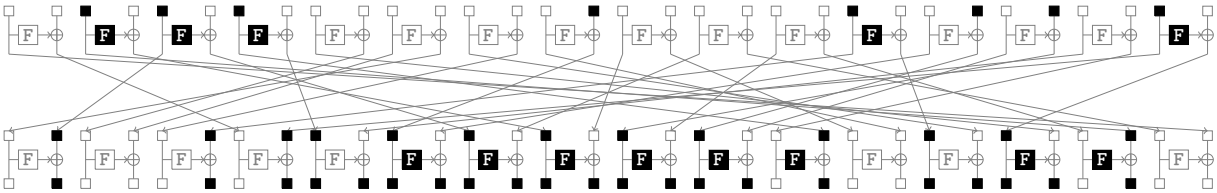


Figure 7.6: *Example of difference propagation over $N_f = 2$ rounds.*

The inactivity of $X[i, 2k + 1]$ can be translated into the equality

$$F(C^1[2k]) \oplus C^1[2k + 1] \oplus F(C^3[2k]) \oplus C^3[2k + 1] = 0.$$

Given that F is the application of an S-box followed by a sub-key addition it can be simplified into: $S(C^1[2k]) \oplus C^1[2k + 1] = S(C^3[2k]) \oplus C^3[2k + 1]$ which does not depend on a secret value. Thus, the idea is to simply add as index the value of $S(C[2k]) \oplus C[2k + 1]$ when building the H table of Step-2.(c), in addition to the value of the bits where the difference is expected to be 0 in the ciphertexts. In our example, it means that we are colliding on 32 additional bits, and thus that the filter for quartets is of size $2 \times (128 - 52) = 152$ bits.

Model for Searching a Rectangle Attack

In this subsection, we briefly go over the main characteristics of the model searching for the rectangle attack. The detailed model is given in Model B.1 in Appendix B. It takes as input the number of rounds covered by the distinguisher and by the prepended and appended rounds of key recovery (respectively N_d, N_b and N_f) and returns the complexities of the best attack that can be found.

The intermediate values that are needed to evaluate the time and data complexity of the attack

are m_b, r_f, and the probability of the distinguisher (previously denoted p^2q^2r). This latter is determined with the same constraints as in the model searching for the best distinguisher, and we simply add constraints to model the additional N_b and N_f rounds.

- r_b and r_f are computed by propagating with probability one the difference at the input and at the output of the boomerang distinguisher, see constraints (FA6) and (FA10). To take into account the above described trick on the filtering process, we define r_f as the number of active nibbles *entering* the last round.
- The data complexity is computed so that s right quartets are found, see constraint (FA3). To make sure that it does not exceed was is available, one may add a constraint stating that $\frac{\sqrt{s}\cdot 2^{n/2}}{\sqrt{p^2q^2r}} < 2^n$.
- As it is impossible to compute the cluster at this stage, we introduce a variable $cluster_{gain}$ which is set by the attacker to represents the expected gain obtained with clusters. Its value is precisely computed afterwards, once a solution to the model is obtained.
- The time complexity is computed as the maximum between the two most expensive stages detailed in section 7.1.1, see constraint (FA3). Again, one may add a constraint saying that the resulting time complexity has to be smaller than the cost of an exhaustive search of the key.
- Constraint (FA5) makes the link between the $\texttt{known}$ variables and the $\texttt{guess}_{\texttt{key}}$ variables.
- We take into account the simple key schedule of WARP to precisely compute the value of m_b. We start by determining the states that have to be known in value (denoted *known* in the model, constraints (FA7), (FA8) and (FA9)) and then link them to the keys and to m_b, taking into account the key schedule (constraints (FA4) and (FA5)).

All the values (except t which is related to the distinguisher probability and $cluster_{gain}$ which is first only approximated) can be computed during Step-1. As a result all the constraints are computed in Step-1 and only the constraints that imply $2t$ or t are modeled in Step-2.

Results: A 26-round Attack on WARP

We apply the previous model to search for rectangle attacks on WARP with various values for the parameters N_b, N_d and N_f. As the execution time rapidly increases with the number of rounds, we added another constraint (proposed in [DDV20]) which consists in using the bounds obtained for differential distinguishers. The idea is that the upper trail (resp. lower trail) cannot be better than a differential trail, *i.e.* the number of active S-boxes in the upper trail (resp. lower trail) cannot be lower than the minimal number of active S-boxes of a differential trail ($optimal_{diff}$). To implement this idea for WARP we use the lower bound of the number of active S-boxes computed in [Ban+20].

The best attack we found covers 26 rounds of WARP based on a $N_d = 22$ rounds distinguisher, $N_b = 1$ round added before and $N_f = 3$ rounds added after (Appendices B.4.1 and B.4.2). The

model took 6 days to solve this instance, and returned the following values: $m_b = 0$, $r_b = 72$, $r_f = 60$ and a distinguisher probability of 2^{-128}. This search was made by assuming that s is equal to 4 and that the value of the cluster gains would be the ones given in 7.2, so in the case of a 22-round distinguisher equal to a factor of 2^{12}.

We next run the cluster search on the 22-round distinguisher. We obtained a probability approximation of $2^{-111.2}$ (so with a cluster gain a bit larger than what was expected), resulting in the associated attack having a data complexity of $2^{120.6}$ messages, and a time complexity of little less than 2^{116} encryptions when following the key recovery method introduced by Zhao *et al.* [Zha+20].

The success probability of the attack is equal to 97.67%.

On the Impact of the Key Addition Position

We now briefly discuss a variant of WARP with a round key addition made before the S-box application, and consider an attack using the same 22-round distinguisher, and the same number of rounds N_b and N_f added before and after the distinguisher. The different round structure changes the value of m_b and r_f, as the improvements discussed in Section 7.3 cannot be applied anymore. m_b increases (from 0) to 32, while r_f is now equal to 88 instead of 60. The data complexity of an attack with such parameters would still be $2^{120.6}$, but the dominating factor of the time complexity becomes $2^{191.2}$.

This example shows the importance of the key addition position and of the techniques discussed in Section 7.3 that save a factor of $2^{75.3}$ in the time complexity.

7.4 Conclusion

In this Chapter, we propose the adaptation of two recent techniques to the case of Feistel ciphers to find boomerang distinguishers and rectangle attacks. Our analysis reveals a 23-round distinguisher and a 26-round attack of WARP, beating by 2 rounds the recent results of [TB21b]. Our code is public and can be used as a basis to attack other Feistel ciphers, and we actually demonstrate its versatility by providing results for TWINE and LBlock–s (see Appendix B.2).

Secondarily, while studying WARP we show how to take advantage of the key addition position to reduce the complexity of the attack. In our specific case, this design decision allows to reduce by a factor of 2^{75} the time complexity of the attack in comparison to a variant of WARP that would have the key addition positioned before the S-box (and thus would have the complementation property).

Outlooks and Conclusion

Our main issues in this work were to improve the scaling of CP solvers, which was done in Chapters 4 and 5, and improving existing attacks, which was done in Chapters 4, 6 and 7.

In Chapter 4 we extended the work of [Gér+20] to the Rijndael case, which required extending the generation of redundant equations to the different cases of the `KeySchedule` as well as adapting the `ShiftRows` operation to take into account the different shifts. We have simplified the equations in Step-2 by using the information extracted from Step-1 in a more efficient way. We also improved the interleaving of the two computational steps by using the minimum and maximum bounds of the two steps to:

1. stop the search at the earliest when the solution is proven to be the optimal solution;
2. speed up the search in step 2 by prohibiting the search for solutions worse than the best known solution.

These improvements allowed us to find two new bounds for the complexity of attacks on the $\mathsf{Rijndael}_{128-224}$ and $\mathsf{Rijndael}_{160-256}$ instances.

This chapter highlights a technique to speed up the search. Many papers using declarative approaches in the differential cryptanalysis community use the two-step method: truncated differential characteristic search and then differential characteristic optimization by using two different solvers. In most cases, the first step is done by a SAT or (M)ILP solver and the second by a (M)ILP or CP solver. Our method, being solver agnostic, can be adapted to many configurations currently in use, which would improve the solving process speed for many problems.

In chapter 5, we created a new global constraint *AbstractXOR* to model truncated differential features more easily. This global constraint strongly improves Choco's performance for Step-1 and makes Choco competitive with **SAT**. We point out that the xor operator is one of the most commonly used operators in encryption and is the operator generally used for calculating differentials. Therefore a correct modelling of this operator is an important issue. However, global constraints have some disadvantages. The main drawback is the loss of efficiency of the DomOverWDeg heuristic due to the standardization of the scores of the different variables. This drawback can be mitigated by adopting certain techniques such as implementing custom heuristics that take into account the structure of the problem underlying the global constraint. Furthermore, a better quality of filtering in the first step of the computation of a differential feature can have a significant impact since it can avoid the search for impossible concrete solutions (a case where a truncated solution is considered valid even if it has no concrete instantiation).

Simplifying the modelling also allows the use of more complex attack models by reducing the risk of errors.

A possible improvement of our approach would be to generalize the modelling of the XOR operator in order to better capture the linear and non-linear operators in the equation system. In the case of Rijndael, for example, it could be interesting to model the multiplication in the Galois field in order to decompose `MixColumns` and thus allow *AbstractXOR* to check more advanced equations. It would also be interesting to add inference rules when the relations between binary sequences have specific properties, *e.g.* transitivity, symmetry, reflexivity.

In Chapter 6 we developed a tool for searching for boomerang distinguisher applied to Rijndael encryption. To do so, we merged the Boomerang model of *Delaune et al.* and the model of Chapter 4. This allows us to create new attacks for the 9 over 11 rounds of the $\mathsf{Rijndael}_{128-160}$ instance.

In Chapter 7 we adapted the *Delaune et al.* model to Feistel's schemes. Indeed, the model of *Delaune et al.* was designed to work on SPNs, which assumes that the non-linear operations are reversed during decryption. In the case of Feistel's schemes, the functions are not inverted, which implies that the calculation of probabilities in the boomerangs is different. We therefore used the work of *Lallemand et al.* to generalize the *Delaune et al.* model to support Feistel schemes as well. Our model was mainly developed for WARP, but being adaptable to other Feistel schemes. So we also applied it to Twine and LBlock-s ciphers. We have also taken automation to the next level as we have integrated the calculation of the complexity of attacks into the model instead of doing this step separately. This model was able to find a new state-of-the-art distinguisher for 23 rounds on WARP and a new state-of-the-art attack on 26 rounds on WARP. We also find new best distinguishers for 15 and 16 rounds of Twine and 16 rounds of LBlock-s.

This chapter shows how to apply some of the techniques developed for SPNs to Feistel schemes. In addition, we show how to integrate the `KeyRecovery` into the model in order to optimize the total complexity of the attack and not only the probability of the distinguisher which is only one parameter among others.

As this thesis shows, modelling cryptanalysis problems is both complex and tedious. While attacks can be applied to a wide range of ciphers, each cipher has specific operations that require the complete rewriting of a new model.

One track for further research would be to integrate the attacks presented in this thesis into automatic template generation mechanisms such as [Lib+21] (see Figure 7.7). The main idea is to represent attacks and ciphers independently. The attacks are represented by models applied to an abstract graph. Ciphers are represented by translating their specification into a specific graph. The solution search then consists in solving the attack model applied to the specific graph related to the cipher. This would make it easier to test new attacks on existing ciphers. Conversely, when developing new ciphers, it would then be possible to test more quickly whether the specifications are resistant to already known attacks.

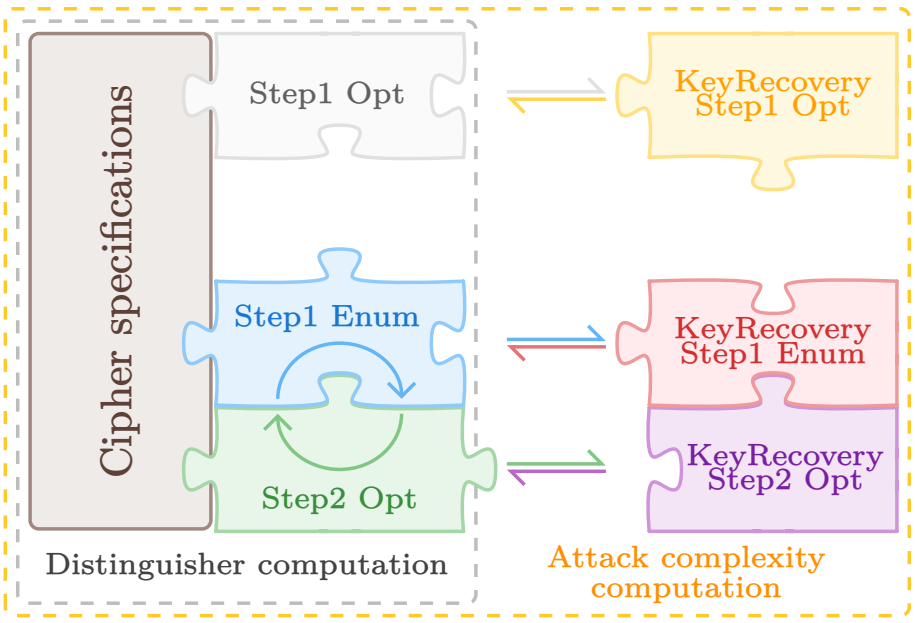


Figure 7.7: *The generic representation of an attack search using a declarative approach. The encryption specification must be translated into a mathematical model. This model must be adapted to the attack search, for example by removing constants in the case of differential analysis or by removing operations that involve the key in the case of single-key attacks. The attack itself must be modelled and adapted to the cipher,* e.g. *a boomerang attack is not represented the same way in the case of an SPN and a Feistel cipher. Finally, the search may involve more or less steps and include more or less automation,* e.g. *by including the* `KeyRecovery` *part and the attack complexity computation.*

Bibliography

[01] *Advanced Encryption Standard (AES)*. National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce. Nov. 2001 (cit. on pp. 21, 73).

[77] *Data Encryption Standard*. National Bureau of Standards, NBS FIPS PUB 46, U.S. Department of Commerce. Jan. 1977 (cit. on p. 34).

[Abd+17] Ahmed Abdelkhalek, Yu Sasaki, Yosuke Todo, Mohamed Tolba, and Amr M. Youssef. "MILP Modeling for (Large) S-boxes to Optimize Probability of Differential Characteristics". In: *IACR Trans. Symm. Cryptol.* 2017.4 (2017), pp. 99–129. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i4.99-129 (cit. on p. 42).

[Bal+13] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. "Adding Virtualization Capabilities to the Grid'5000 Testbed". In: *Cloud Computing and Services Science*. Ed. by Ivan I. Ivanov, Marten van Sinderen, Frank Leymann, and Tony Shan. Vol. 367. Communications in Computer and Information Science. Springer International Publishing, 2013, pp. 3–20. ISBN: 978-3-319-04518-4. DOI: 10.1007/978-3-319-04519-1_1 (cit. on p. 83).

[Ban+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. "Midori: A Block Cipher for Low Energy". In: *ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. LNCS. Springer, Heidelberg, Nov. 2015, pp. 411–436. DOI: 10.1007/978-3-662-48800-3_17 (cit. on pp. 26, 27, 106).

[Ban+20] Subhadeep Banik, Zhenzhen Bao, Takanori Isobe, Hiroyasu Kubo, Fukang Liu, Kazuhiko Minematsu, Kosei Sakamoto, Nao Shibata, and Maki Shigeri. "WARP : Revisiting GFN for Lightweight 128-Bit Block Cipher". In: *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*. Ed. by Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn. Vol. 12804. Lecture Notes in Computer Science. Springer, 2020, pp. 535–564. DOI: 10.1007/978-3-030-81652-0_21. URL: https://doi.org/10.1007/978-3-030-81652-0%5C_21 (cit. on pp. 30, 124, 125, 141).

[BDK05] Eli Biham, Orr Dunkelman, and Nathan Keller. "Related-Key Boomerang and Rectangle Attacks". In: *EUROCRYPT 2005*. Ed. by Ronald Cramer. Vol. 3494. LNCS. Springer, Heidelberg, May 2005, pp. 507–525. DOI: 10.1007/11426639_30 (cit. on p. 113).

[Bei+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. "The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS". In: *CRYPTO 2016, Part II*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. LNCS. Springer, Heidelberg, Aug. 2016, pp. 123–153. DOI: 10.1007/978-3-662-53008-5_5 (cit. on p. 111).

[Bel53] Giovan Battista Bellaso. *La Cifra del Sig.* 1553 (cit. on pp. 1, 7).

[Bes+04] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. "The Tractability of Global Constraints". In: *CP*. Vol. 3258. Lecture Notes in Computer Science. Springer, 2004, pp. 716–720 (cit. on pp. 49, 102).

[Bie] Armin Biere. *Lingeling*. URL: https://github.com/arminbiere/lingeling (cit. on pp. 52, 106, 136).

[Bih94] Eli Biham. "New Types of Cryptanalytic Attacks Using Related Keys". In: *Journal of Cryptology* 7.4 (Dec. 1994), pp. 229–246. DOI: 10.1007/BF00203965 (cit. on p. 34).

[BK09] Alex Biryukov and Dmitry Khovratovich. "Related-Key Cryptanalysis of the Full AES-192 and AES-256". In: *ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. LNCS. Springer, Heidelberg, Dec. 2009, pp. 1–18. DOI: 10.1007/978-3-642-10366-7_1 (cit. on pp. 123, 125).

[BN10] 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: *EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 322–344. DOI: 10.1007/978-3-642-13190-5_17 (cit. on pp. 42, 73).

[Bou+04] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. "Boosting Systematic Search by Weighting Constraints". In: *ECAI*. IOS Press, 2004, pp. 146–150 (cit. on pp. 102, 103, 106).

[Bou+20] Hamid Boukerrou, Paul Huynh, Virginie Lallemand, Bimal Mandal, and Marine Minier. "On the Feistel Counterpart of the Boomerang Connectivity Table (Long Paper)". In: *IACR Trans. Symm. Cryptol.* 2020.1 (2020), pp. 331–362. ISSN: 2519-173X. DOI: 10.13154/tosc.v2020.i1.331-362 (cit. on pp. 5, 10, 123, 126, 168).

[BS91] Eli Biham and Adi Shamir. "Differential Cryptanalysis of DES-like Cryptosystems". In: *CRYPTO'90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Vol. 537. LNCS. Springer, Heidelberg, Aug. 1991, pp. 2–21. DOI: 10.1007/3-540-38424-3_1 (cit. on pp. 18, 34, 35, 39, 113, 123).

[BW99] Alex Biryukov and David Wagner. “Slide Attacks”. In: *FSE'99*. Ed. by Lars R. Knudsen. Vol. 1636. LNCS. Springer, Heidelberg, Mar. 1999, pp. 245–259. DOI: 10.1007/3-540-48519-8_18 (cit. on p. 20).

[Chr10] Jan Pelzl (auth.) Christof Paar. *Understanding cryptography: a textbook for students and practitioners*. 1st ed. Springer-Verlag Berlin Heidelberg, 2010. ISBN: 3642041000; 9783642041006; 3642446493; 9783642446498; 3642041019; 9783642041013 (cit. on p. 18).

[Cid+18] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. “Boomerang Connectivity Table: A New Cryptanalysis Tool”. In: *EUROCRYPT 2018, Part II*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10821. LNCS. Springer, Heidelberg, Apr. 2018, pp. 683–714. DOI: 10.1007/978-3-319-78375-8_22 (cit. on pp. 111, 113, 123, 126).

[CR15] Anne Canteaut and Joëlle Roué. “On the Behaviors of Affine Equivalent Sboxes Regarding Differential and Linear Attacks”. In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 45–74. DOI: 10.1007/978-3-662-46800-5_3 (cit. on p. 119).

[CS14] Geoffrey Chu and Peter J. Stuckey. *Chuffed solver description*. Available at http://www.minizinc.org/challenge2014/description_chuffed.txt. 2014 (cit. on p. 116).

[DDV20] Stéphanie Delaune, Patrick Derbez, and Mathieu Vavrille. “Catching the Fastest Boomerangs Application to SKINNY”. In: *IACR Trans. Symm. Cryptol.* 2020.4 (2020), pp. 104–129. ISSN: 2519-173X. DOI: 10.46586/tosc.v2020.i4.104-129 (cit. on pp. 5, 10, 64, 65, 67, 113–115, 119, 123, 124, 127, 133, 141).

[DKS10] Orr Dunkelman, Nathan Keller, and Adi Shamir. “A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony”. In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 393–410. DOI: 10.1007/978-3-642-14623-7_21 (cit. on pp. 64, 125).

[DKS14] Orr Dunkelman, Nathan Keller, and Adi Shamir. “A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony”. In: *Journal of Cryptology* 27.4 (Oct. 2014), pp. 824–849. DOI: 10.1007/s00145-013-9154-9 (cit. on p. 64).

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. “A machine program for theorem-proving”. In: *Commun. ACM* 5.7 (1962), pp. 394–397. DOI: 10.1145/368273.368557. URL: https://doi.org/10.1145/368273.368557 (cit. on p. 48).

[Don+21] Xiaoyang Dong, Lingyue Qin, Siwei Sun, and Xiaoyun Wang. *Key Guessing Strategies for Linear Key-Schedule Algorithms in Rectangle Attacks*. Cryptology ePrint Archive, Report 2021/856. https://eprint.iacr.org/2021/856. 2021 (cit. on pp. 119, 126).

[DP60] Martin Davis and Hilary Putnam. “A Computing Procedure for Quantification Theory”. In: *J. ACM* 7.3 (1960), pp. 201–215. DOI: 10.1145/321033.321034. URL: http://doi.acm.org/10.1145/321033.321034 (cit. on p. 48).

[DR06] Joan Daemen and Vincent Rijmen. “Understanding Two-Round Differentials in AES”. In: *SCN 06*. Ed. by Roberto De Prisco and Moti Yung. Vol. 4116. LNCS. Springer, Heidelberg, Sept. 2006, pp. 78–94. DOI: 10.1007/11832072_6 (cit. on p. 119).

[DR99] Joan Daemen and Vincent Rijmen. “AES proposal: Rijndael”. In: (1999) (cit. on pp. 21, 114).

[Dwo01] Morris J Dworkin. *Sp 800-38a 2001 edition. recommendation for block cipher modes of operation: Methods and techniques*. 2001 (cit. on p. 19).

[Ehr+78] William F Ehrsam, Carl HW Meyer, John L Smith, and Walter L Tuchman. *Message verification and transmission error detection by block chaining*. US Patent 4,074,066. Feb. 1978 (cit. on p. 19).

[Fei73] Horst Feistel. “Cryptography and Computer Privacy”. In: *Scientific American* 228.5 (May 1973), pp. 15–23. DOI: 10.1038/scientificamerican0573-15 (cit. on p. 21).

[Fei74] Horst Feistel. *Block Cipher Cryptographic System*. US Patent 3,798,359. 1974. URL: https://patentimages.storage.googleapis.com/c4/82/27/f8da3eaeb64bdf/US3798359.pdf (cit. on p. 29).

[FJP13] Pierre-Alain Fouque, Jérémy Jean, and Thomas Peyrin. “Structural Evaluation of AES and Chosen-Key Distinguisher of 9-Round AES-128”. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 183–203. DOI: 10.1007/978-3-642-40041-4_11 (cit. on pp. 42, 73, 79).

[FT87] Michael L. Fredman and Robert Endre Tarjan. “Fibonacci heaps and their uses in improved network optimization algorithms”. In: *J. ACM* 34.3 (1987), pp. 596–615 (cit. on p. 49).

[Gér+18] David Gérault, Pascal Lafourcade, Marine Minier, and Christine Solnon. “Revisiting AES related-key differential attacks with constraint programming”. In: *Inf. Process. Lett.* 139 (2018), pp. 24–29. DOI: 10.1016/j.ipl.2018.07.001. URL: https://doi.org/10.1016/j.ipl.2018.07.001 (cit. on p. 42).

[Gér+20] David Gérault, Pascal Lafourcade, Marine Minier, and Christine Solnon. “Computing AES related-key differential characteristics with constraint programming”. In: *Artif. Intell.* 278 (2020). DOI: 10.1016/j.artint.2019.103183. URL: https://doi.org/10.1016/j.artint.2019.103183 (cit. on pp. 5, 10, 53, 55, 57, 59, 61–64, 73–76, 78, 79, 82–84, 91, 107, 108, 110, 143).

[Gér18] David Gérault. “Security analysis of contactless communication protocols. (Analyse de sécurité des protocoles de communication sans contact)”. PhD thesis. University of Clermont Auvergne, Clermont-Ferrand, France, 2018. URL: https://tel.archives-ouvertes.fr/tel-02536478 (cit. on pp. 94, 106).

[GM08] Samuel Galice and Marine Minier. “Improving Integral Attacks Against Rijndael-256 Up to 9 Rounds”. In: *AFRICACRYPT 08*. Ed. by Serge Vaudenay. Vol. 5023. LNCS. Springer, Heidelberg, June 2008, pp. 1–15 (cit. on pp. 74, 114).

[GMS16] David Gérault, Marine Minier, and Christine Solnon. “Constraint Programming Models for Chosen Key Differential Cryptanalysis”. In: *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*. Ed. by Michel Rueher. Vol. 9892. Lecture Notes in Computer Science. Springer, 2016, pp. 584–601. DOI: 10.1007/978-3-319-44953-1_37. URL: https://doi.org/10.1007/978-3-319-44953-1_37 (cit. on pp. 4, 9, 42, 76).

[Gom58] Ralph Gomory. “Essentials of an algorithm for integer solutions to linear programs”. In: *Recent Advances in Mathematical Programming* (1958), pp. 269–302 (cit. on p. 49).

[Gur22] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2022. URL: https://www.gurobi.com (cit. on p. 52).

[HBS21] Hosein Hadipour, Nasour Bagheri, and Ling Song. “Improved Rectangle Attacks on SKINNY and CRAFT”. In: *IACR Trans. Symm. Cryptol.* 2021.2 (2021), pp. 140–198. ISSN: 2519-173X. DOI: 10.46586/tosc.v2021.i2.140-198 (cit. on p. 123).

[Hey02] Howard M. Heys. “A TUTORIAL ON LINEAR AND DIFFERENTIAL CRYPTANALYSIS”. en. In: *Cryptologia* 26.3 (July 2002), pp. 189–221. ISSN: 0161-1194, 1558-1586. DOI: 10.1080/0161-110291890885. URL: http://www.tandfonline.com/doi/abs/10.1080/0161-110291890885 (visited on 04/01/2022) (cit. on pp. 39–41).

[HNE22] Hosein Hadipour, Marcel Nageler, and Maria Eichlseder. *Throwing Boomerangs into Feistel Structures: Application to CLEFIA, WARP, LBlock, LBlock-s and TWINE*. Cryptology ePrint Archive, Report 2022/745. https://eprint.iacr.org/2022/745. 2022 (cit. on p. 125).

[Jea15] Jérémy Jean. *Iterated construction for a block cipher*. 2015. URL: https://www.iacr.org/authors/tikz/ (cit. on p. 21).

[Kas63] Friedrich Wilhelm Kasiski. *Die Geheimschriften und die Dechiffrir-Kunst*. 1863 (cit. on pp. 1, 7).

[Ker83a] Auguste Kerckhoffs. “La cryptographie militaire”. In: *Journal des sciences militaires*. Vol. 9. Jan. 1883, pp. 5–39 (cit. on pp. 1, 7).

[Ker83b] Auguste Kerckhoffs. “La cryptographie militaire”. In: *Journal des sciences militaires*. Vol. 9. Feb. 1883, pp. 161–191 (cit. on pp. 1, 7).

[KLT15] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. “Observations on the SIMON Block Cipher Family”. In: *CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. LNCS. Springer, Heidelberg, Aug. 2015, pp. 161–185. DOI: 10.1007/978-3-662-47989-6_8 (cit. on p. 42).

[Knu95] Lars R. Knudsen. "Truncated and Higher Order Differentials". In: *FSE'94*. Ed. by Bart Preneel. Vol. 1008. LNCS. Springer, Heidelberg, Dec. 1995, pp. 196–211. DOI: 10.1007/3-540-60590-8_16 (cit. on pp. 36, 53, 74).

[KY21] Manoj Kumar and Tarun Yadav. "MILP Based Differential Attack on Round Reduced WARP". In: *Security, Privacy, and Applied Cryptography Engineering - 11th International Conference, SPACE 2021, Kolkata, India, December 10-13, 2021, Proceedings*. Ed. by Lejla Batina, Stjepan Picek, and Mainack Mondal. Vol. 13162. Lecture Notes in Computer Science. Springer, 2021, pp. 42–59. DOI: 10.1007/978-3-030-95085-9_3. URL: https://doi.org/10.1007/978-3-030-95085-9%5C_3 (cit. on p. 125).

[LD10] Ailsa H. Land and Alison G. Doig. "An Automatic Method for Solving Discrete Programming Problems". In: *50 Years of Integer Programming*. Springer, 2010, pp. 105–132 (cit. on p. 49).

[Le +13] Vianney Le Clément de Saint-Marcq, Pierre Schaus, Christine Solnon, and Christophe Lecoutre. "Sparse-Sets for Domain Implementation". In: *CP workshop on Techniques foR Implementing Constraint programming Systems (TRICS)*. Uppsala, Sweden, Sept. 2013, pp. 1–10. URL: https://hal.archives-ouvertes.fr/hal-01339250 (cit. on p. 101).

[Lib+21] Luc Libralesso, François Delobel, Pascal Lafourcade, and Christine Solnon. "Automatic Generation of Declarative Models For Differential Cryptanalysis". In: *CP*. Vol. 210. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 40:1–40:18 (cit. on pp. 136, 144).

[Liu+19] Ya Liu, Yifan Shi, Dawu Gu, Bo Dai, Fengyu Zhao, Wei Li, Zhiqiang Liu, and Zhiqiang Zeng. "Improved impossible differential cryptanalysis of large-block Rijndael". In: *Sci. China Inf. Sci.* 62.3 (2019), 32101:1–32101:14. DOI: 10.1007/s11432-017-9365-4. URL: https://doi.org/10.1007/s11432-017-9365-4 (cit. on pp. 74, 114).

[LMR22] Virginie Lallemand, Marine Minier, and Loïc Rouquette. "Automatic Search of Rectangle Attacks on Feistel Ciphers: Application to WARP". In: *IACR Trans. Symmetric Cryptol.* 2022.2 (2022), pp. 113–140. DOI: 10.46586/tosc.v2022.i2.113-140. URL: https://doi.org/10.46586/tosc.v2022.i2.113-140 (cit. on pp. 6, 11).

[LW66] E. L. Lawler and D. E. Wood. "Branch-and-Bound Methods: A Survey". In: *Oper. Res.* 14.4 (1966), pp. 699–719 (cit. on p. 49).

[Min17] Marine Minier. "Improving impossible-differential attacks against Rijndael-160 and Rijndael-224". In: *Des. Codes Cryptogr.* 82.1-2 (2017), pp. 117–129. DOI: 10.1007/s10623-016-0206-7. URL: https://doi.org/10.1007/s10623-016-0206-7 (cit. on pp. 74, 114).

[Mod80] DES Modes. "of Operation". In: *ANSI X9* 52 (1980) (cit. on p. 19).

[Mou+11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. "Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming". In: *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*. Ed. by Chuankun Wu, Moti Yung, and Dongdai Lin. Vol. 7537. Lecture Notes in Computer Science. Springer, 2011, pp. 57–76. DOI: 10.1007/978-3-642-34704-7_5. URL: https://doi.org/10.1007/978-3-642-34704-7_5 (cit. on pp. 4, 9, 42).

[MP13] Nicky Mouha and Bart Preneel. "A Proof that the ARX Cipher Salsa20 is Secure against Differential Cryptanalysis". In: *IACR Cryptol. ePrint Arch.* (2013), p. 328. URL: http://eprint.iacr.org/2013/328 (cit. on p. 42).

[MSR14] Marine Minier, Christine Solnon, and Julia Reboul. "Solving a Symmetric Key Cryptographic Problem with Constraint Programming". In: *ModRef 2014, Workshop of the CP 2014 Conference*. Lyon, France, Sept. 2014, p. 13. URL: https://hal.inria.fr/hal-01092574 (cit. on pp. 57, 59).

[Mur11] Sean Murphy. "The Return of the Cryptographic Boomerang". In: *IEEE Trans. Inf. Theory* 57.4 (2011), pp. 2517–2521 (cit. on pp. 123, 125).

[MVV18] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2018 (cit. on p. 19).

[Net+07] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. "MiniZinc: Towards a Standard CP Modelling Language". In: *CP*. Vol. 4741. Lecture Notes in Computer Science. Springer, 2007, pp. 529–543 (cit. on pp. 116, 136).

[NP07] Jorge Nakahara Jr. and Ivan Carlos Pavão. "Impossible-Differential Attacks on Large-Block Rijndael". In: *ISC 2007*. Ed. by Juan A. Garay, Arjen K. Lenstra, Masahiro Mambo, and René Peralta. Vol. 4779. LNCS. Springer, Heidelberg, Oct. 2007, pp. 104–117 (cit. on pp. 74, 114).

[Pap94] Christos H. Papadimitriou. *Computational complexity*. Reading, Mass: Addison-Wesley, 1994. ISBN: 978-0-201-53082-7 (cit. on pp. 47, 50).

[PFL16] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. "Choco solver documentation". In: *TASC, INRIA Rennes, LINA CNRS UMR* 6241 (2016) (cit. on pp. 102, 106, 114, 116, 118, 133, 136).

[PP10] Christof Paar and Jan Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. en. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. ISBN: 978-3-642-44649-8 978-3-642-04101-3. DOI: 10.1007/978-3-642-04101-3. URL: http://link.springer.com/10.1007/978-3-642-04101-3 (visited on 03/04/2022) (cit. on p. 18).

[PR91] Manfred Padberg and Giovanni Rinaldi. “A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems”. In: *SIAM Rev.* 33.1 (1991), pp. 60–100. DOI: 10.1137/1033004. URL: https://doi.org/10.1137/1033004 (cit. on p. 49).

[Qin+21] Lingyue Qin, Xiaoyang Dong, Xiaoyun Wang, Keting Jia, and Yunwen Liu. “Automated Search Oriented to Key Recovery on Ciphers with Linear Key Schedule”. In: *IACR Trans. Symm. Cryptol.* 2021.2 (2021), pp. 249–291. ISSN: 2519-173X. DOI: 10.46586/tosc.v2021.i2.249-291 (cit. on pp. 124, 137, 139).

[Rég94] Jean-Charles Régin. “A Filtering Algorithm for Constraints of Difference in CSPs”. In: *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1*. Ed. by Barbara Hayes-Roth and Richard E. Korf. AAAI Press / The MIT Press, 1994, pp. 362–367. URL: http://www.aaai.org/Library/AAAI/1994/aaai94-055.php (cit. on pp. 49, 51).

[Rou+22] Loïc Rouquette, David Gerault, Marine Minier, and Christine Solnon. “And Rijndael? Automatic Related-key Differential Analysis of Rijndael”. In: *AfricaCrypt 2022 - 13th International Conference on Cryptology AfricaCrypt*. Fes, Morocco, July 2022. URL: https://hal.archives-ouvertes.fr/hal-03671013 (cit. on pp. 6, 11).

[RS19] Loïc Rouquette and Christine Solnon. “Contrainte globale abstractXOR : résultats de complexité et algorithmes de propagation”. In: *Journées Francophones de Programmation par Contraintes (JFPC)*. Albi, France, 2019. URL: https://hal.archives-ouvertes.fr/hal-02147858 (cit. on pp. 6, 11).

[RS20] Loïc Rouquette and Christine Solnon. “abstractXOR: A global constraint dedicated to differential cryptanalysis”. en. In: *Principles and Practice of Constraint Programming*. Ed. by Helmut Simonis. Vol. 12333. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 566–584. ISBN: 978-3-030-58474-0 978-3-030-58475-7. DOI: 10.1007/978-3-030-58475-7_33. URL: https://link.springer.com/10.1007/978-3-030-58475-7_33 (visited on 04/30/2021) (cit. on pp. 6, 11).

[RS22] Loïc Rouquette and Christine Solnon. “Utilisation de la Programmation par contrainte appliquée à la cryptanalyse différentielle”. In: *23ème congrès annuel de la Société Française de Recherche Opérationnelle et d’Aide à la Décision*. INSA Lyon. Villeurbanne - Lyon, France, Feb. 2022. URL: https://hal.archives-ouvertes.fr/hal-03595449 (cit. on pp. 6, 11).

[RVW06] Francesca Rossi, Peter Van Beek, and Toby Walsh, eds. *Handbook of constraint programming*. en. 1st ed. Foundations of artificial intelligence. OCLC: ocm70408044. Amsterdam ; Boston: Elsevier, 2006. ISBN: 978-0-444-52726-4 (cit. on pp. 46–48).

[Sel08] Ali Aydin Selçuk. “On Probability of Success in Linear and Differential Cryptanalysis”. In: *Journal of Cryptology* 21.1 (Jan. 2008), pp. 131–147. DOI: 10.1007/s00145-007-9013-7 (cit. on pp. 89, 91, 120, 127).

[Sha49] C. E. Shannon. “Communication Theory of Secrecy Systems*”. en. In: *Bell System Technical Journal* 28.4 (Oct. 1949), pp. 656–715. ISSN: 00058580. DOI: 10.1002/j.1538-7305.1949.tb00928.x. URL: https://ieeexplore.ieee.org/document/6769090 (visited on 02/09/2022) (cit. on pp. 3, 7, 20).

[Shi+07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. “The 128-Bit Blockcipher CLEFIA (Extended Abstract)”. In: *FSE 2007*. Ed. by Alex Biryukov. Vol. 4593. LNCS. Springer, Heidelberg, Mar. 2007, pp. 181–195. DOI: 10.1007/978-3-540-74619-5_12 (cit. on pp. 30–32).

[SN14] Hadi Soleimany and Kaisa Nyberg. “Zero-correlation linear cryptanalysis of reduced-round LBlock”. In: *Des. Codes Cryptogr.* 73.2 (2014), pp. 683–698 (cit. on pp. 32, 33, 168).

[SNC09] 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, Swansea, UK, June 30 - July 3, 2009. Proceedings*. Ed. by Oliver Kullmann. Vol. 5584. Lecture Notes in Computer Science. Springer, 2009, pp. 244–257. DOI: 10.1007/978-3-642-02777-2_24. URL: https://doi.org/10.1007/978-3-642-02777-2_24 (cit. on pp. 4, 9).

[SS96] João P. Marques Silva and Karem A. Sakallah. “GRASP - a new search algorithm for satisfiability”. In: *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, November 10-14, 1996*. Ed. by Rob A. Rutenbar and Ralph H. J. M. Otten. IEEE, 1996, pp. 220–227. DOI: 10.1109/ICCAD.1996.569607. URL: https://doi.org/10.1109/ICCAD.1996.569607 (cit. on p. 48).

[SS99] João P. Marques Silva and Karem A. Sakallah. “GRASP: A Search Algorithm for Propositional Satisfiability”. In: *IEEE Trans. Computers* 48.5 (1999), pp. 506–521. DOI: 10.1109/12.769433. URL: https://doi.org/10.1109/12.769433 (cit. on p. 48).

[Sun+14] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. “Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-Oriented Block Ciphers”. In: *ASIACRYPT 2014, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. LNCS. Springer, Heidelberg, Dec. 2014, pp. 158–178. DOI: 10.1007/978-3-662-45611-8_9 (cit. on p. 42).

[Sun+17] Siwei Sun, David Gerault, Pascal Lafourcade, Qianqian Yang, Yosuke Todo, Kexin Qiao, and Lei Hu. “Analysis of AES, SKINNY, and Others with Constraint Programming”. In: *IACR Trans. Symm. Cryptol.* 2017.1 (2017), pp. 281–306. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i1.281-306 (cit. on p. 42).

[Sut63] Ivan E. Sutherland. *Sketchpad, A Man-Machine Graphical Communication System*. Outstanding Dissertations in the Computer Sciences. Garland Publishing, New York, 1963 (cit. on pp. 3, 9).

[Suz+13] Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. "TWINE : A Lightweight Block Cipher for Multiple Platforms". In: *SAC 2012*. Ed. by Lars R. Knudsen and Huapeng Wu. Vol. 7707. LNCS. Springer, Heidelberg, Aug. 2013, pp. 339–354. DOI: 10.1007/978-3-642-35999-6_22 (cit. on pp. 31, 32, 135, 168).

[SWW18] Ling Sun, Wei Wang, and Meiqin Wang(66). "More Accurate Differential Properties of LED64 and Midori64". In: *IACR Trans. Symm. Cryptol.* 2018.3 (2018), pp. 93–123. ISSN: 2519-173X. DOI: 10.13154/tosc.v2018.i3.93-123 (cit. on p. 42).

[TB21a] Je Sen Teh and Alex Biryukov. "Differential Cryptanalysis of WARP". In: *IACR Cryptol. ePrint Arch.* (2021), p. 1641. URL: https://eprint.iacr.org/2021/1641 (cit. on p. 125).

[TB21b] Je Sen Teh and Alex Biryukov. *Differential Cryptanalysis of WARP*. Cryptology ePrint Archive, Report 2021/1641. https://eprint.iacr.org/2021/1641. 2021 (cit. on pp. 142, 168).

[Tod+17] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. "Cube Attacks on Non-Blackbox Polynomials Based on Division Property". In: *CRYPTO 2017, Part III*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. LNCS. Springer, Heidelberg, Aug. 2017, pp. 250–279. DOI: 10.1007/978-3-319-63697-9_9 (cit. on p. 111).

[Wag99] David Wagner. "The Boomerang Attack". In: *FSE'99*. Ed. by Lars R. Knudsen. Vol. 1636. LNCS. Springer, Heidelberg, Mar. 1999, pp. 156–170. DOI: 10.1007/3-540-48519-8_12 (cit. on pp. 37, 38, 113, 123).

[Wan+13] Qingju Wang, Dawu Gu, Vincent Rijmen, Ya Liu, Jiazhe Chen, and Andrey Bogdanov. "Improved Impossible Differential Attacks on Large-Block Rijndael". In: *ICISC 12*. Ed. by Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon. Vol. 7839. LNCS. Springer, Heidelberg, Nov. 2013, pp. 126–140. DOI: 10.1007/978-3-642-37682-5_10 (cit. on pp. 74, 114).

[Wan+15] Qingju Wang, Zhiqiang Liu, Deniz Toz, Kerem Varici, and Dawu Gu. "Related-key rectangle cryptanalysis of Rijndael-160 and Rijndael-192". In: *IET Inf. Secur.* 9.5 (2015), pp. 266–276. DOI: 10.1049/iet-ifs.2014.0380. URL: https://doi.org/10.1049/iet-ifs.2014.0380 (cit. on pp. 74, 114).

[WZ11] Wenling Wu and Lei Zhang. "LBlock: A Lightweight Block Cipher". In: *ACNS 11*. Ed. by Javier Lopez and Gene Tsudik. Vol. 6715. LNCS. Springer, Heidelberg, June 2011, pp. 327–344. DOI: 10.1007/978-3-642-21554-4_19 (cit. on pp. 32, 33, 168).

[ZDJ19] Boxin Zhao, Xiaoyang Dong, and Keting Jia. "New Related-Tweakey Boomerang and Rectangle Attacks on Deoxys-BC Including BDT Effect". In: *IACR Trans. Symm. Cryptol.* 2019.3 (2019), pp. 121–151. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i3.121-151 (cit. on pp. 124, 137).

[Zha+08] Lei Zhang, Wenling Wu, Je Hong Park, Bon Wook Koo, and Yongjin Yeom. “Improved Impossible Differential Attacks on Large-Block Rijndael”. In: *ISC 2008*. Ed. by Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee. Vol. 5222. LNCS. Springer, Heidelberg, Sept. 2008, pp. 298–315 (cit. on pp. 74, 114).

[Zha+14] Lei Zhang, Wenling Wu, Yanfeng Wang, Shengbao Wu, and Jian Zhang. “Lac: A lightweight Authenticated Encryption Cipher”. In: *Published to the CAESAR competition* (2014), p. 21 (cit. on pp. 32, 33).

[Zha+20] Boxin Zhao, Xiaoyang Dong, Willi Meier, Keting Jia, and Gaoli Wang. “Generalized related-key rectangle attacks on block ciphers with linear key schedule: applications to SKINNY and GIFT”. In: *Des. Codes Cryptogr.* 88.6 (2020), pp. 1103–1126 (cit. on pp. 119, 126, 139, 142, 167).

[ZK16] Neng-Fa Zhou and Håkan Kjellerstrand. “The Picat-SAT Compiler”. In: *PADL*. Vol. 9585. Lecture Notes in Computer Science. Springer, 2016, pp. 48–62 (cit. on pp. 52, 106, 114, 116, 133, 136).

[ZMI90] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. “On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses”. In: *CRYPTO’89*. Ed. by Gilles Brassard. Vol. 435. LNCS. Springer, Heidelberg, Aug. 1990, pp. 461–480. DOI: 10.1007/0-387-34805-0_42 (cit. on pp. 30–32).

Appendix

A Rijndael

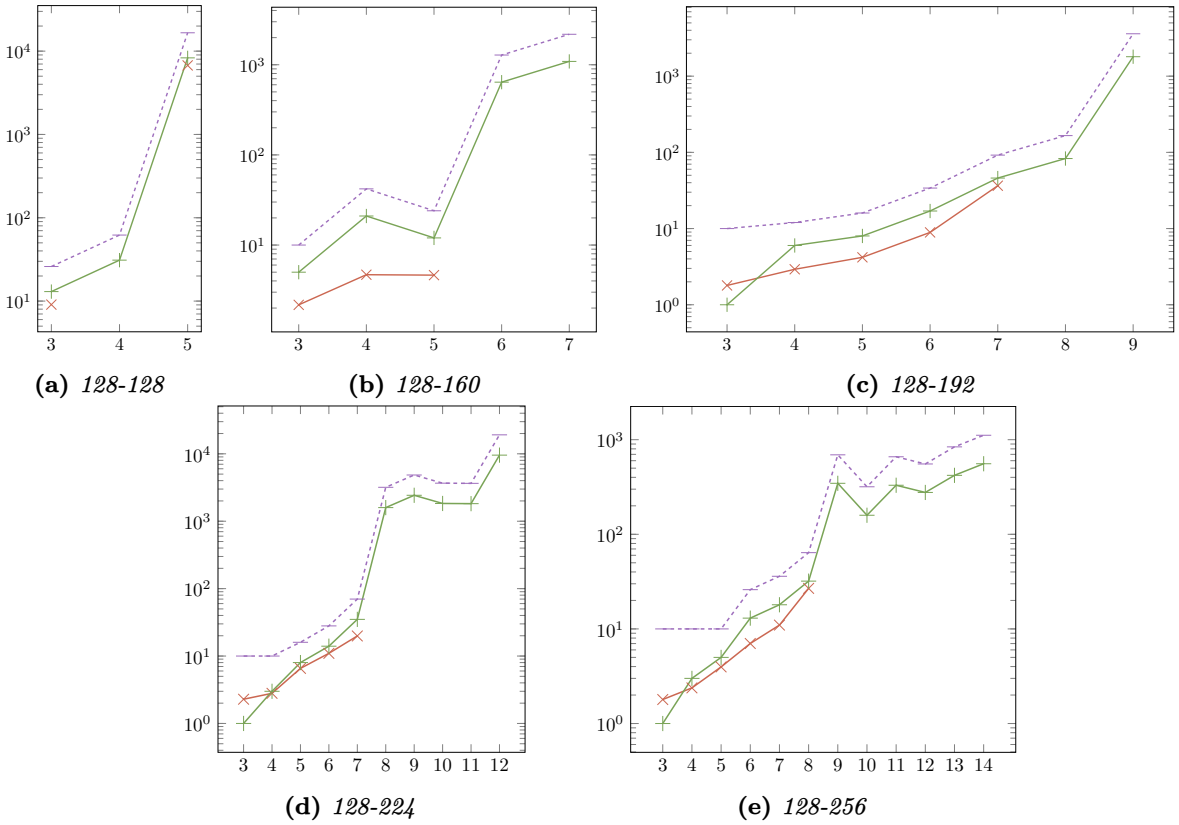
A.1 Round constants

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
00	01	02	04	08	10	20	40	80	1B	36	6C	D8	AB	4D	9A	2F	5E	BC	63	C6	97	35	6A	D4	B3	7D	FA	EF	C5	91	39

Figure A.1: *Round constants (in hexadecimal) for the key generation of* Rijndael

A.2 *Global*$_{GAC}$ vs our **SAT** / **CP** model on Rijndael

A.2.1 Results for Block length = 128

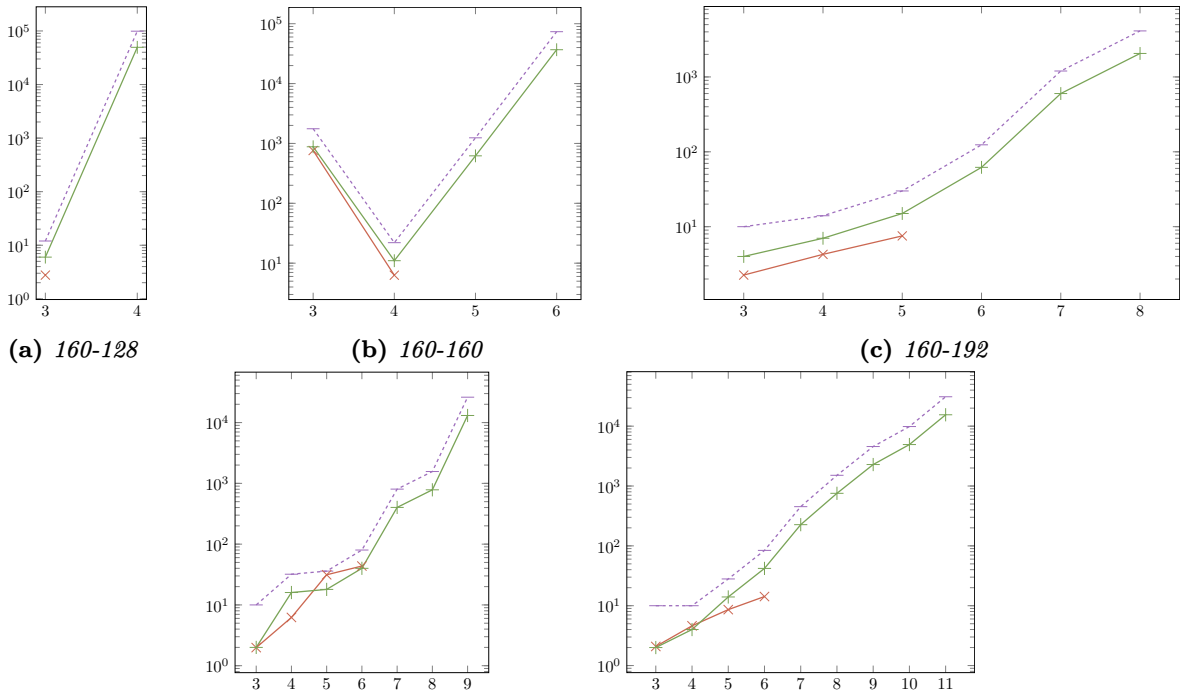


(a) *128-128* **(b)** *128-160* **(c)** *128-192*

(d) *128-224* **(e)** *128-256*

Figure A.2: *Solving time for* Global$_{GAC}$ *(full CP) and* ***SAT / CP*** *models on* Rijndael$_{128-K_{len}}$. *Each instance is denoted* $128 - K_{len}$ *where* K_{len} *is the key size in bits. The* Global$_{GAC}$ *model (Chapter 5) is in red (-×-), the* ***SAT / CP*** *model (Chapter 4) is in green (-+-) and the time limit (*2× *the time of SAT / CP model) is in purple (- -). The* x *axis is the number of rounds and the* y *axis is the time in seconds, measures that are above the time limit are not depicted.*

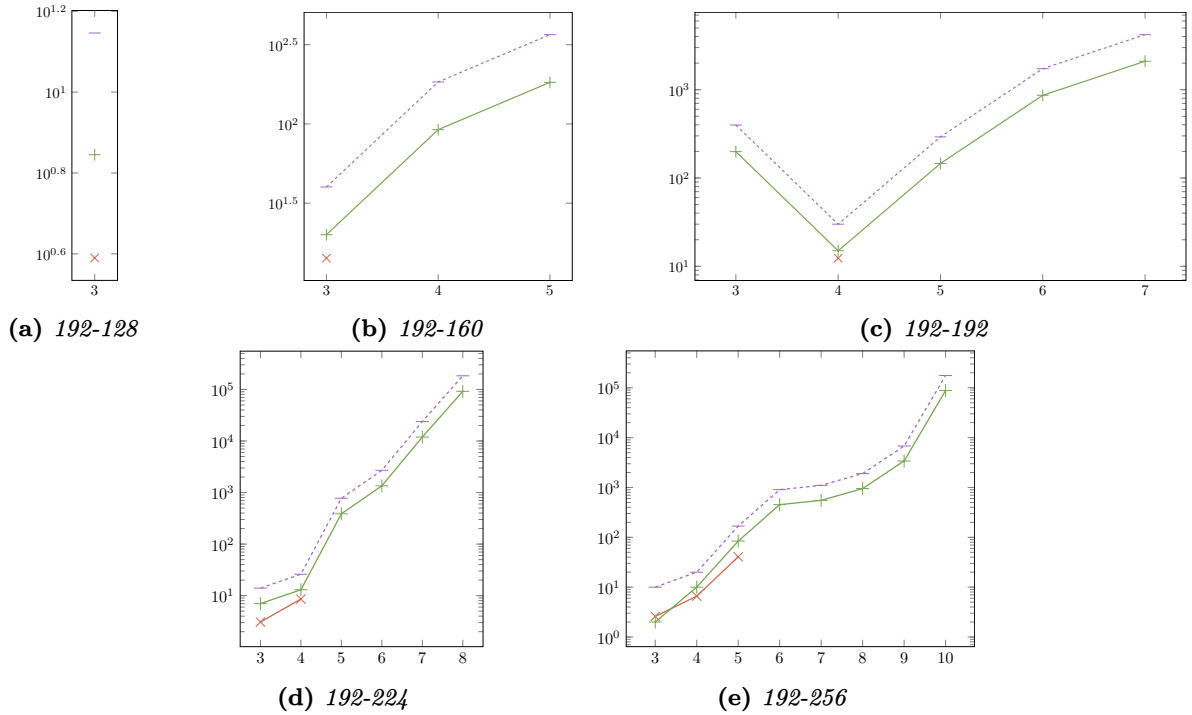
A.2.2 Results for Block length = 160 and 192



(a) *160-128* **(b)** *160-160* **(c)** *160-192*

(d) *160-224* **(e)** *160-256*

Figure A.3: *Solving time for* Global_{GAC} *(full CP) and* ***SAT / CP*** *models on* $\text{Rijndael}_{160-K_{len}}$. *Each instance is denoted* $160 - K_{len}$ *where* K_{len} *is the key size in bits.*

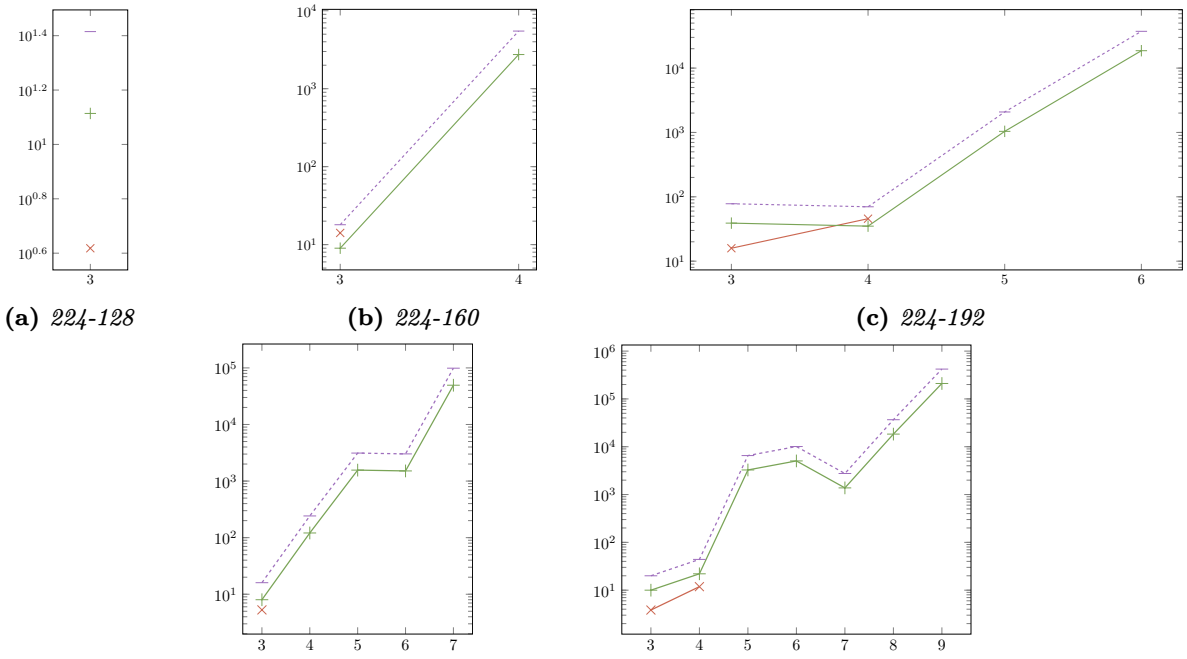


(a) *192-128* **(b)** *192-160* **(c)** *192-192*

(d) *192-224* **(e)** *192-256*

Figure A.4: *Solving time for* Global_{GAC} *(full CP) and* ***SAT / CP*** *models on* $\text{Rijndael}_{192-K_{len}}$. *Each instance is denoted* $192 - K_{len}$ *where* K_{len} *is the key size in bits.*

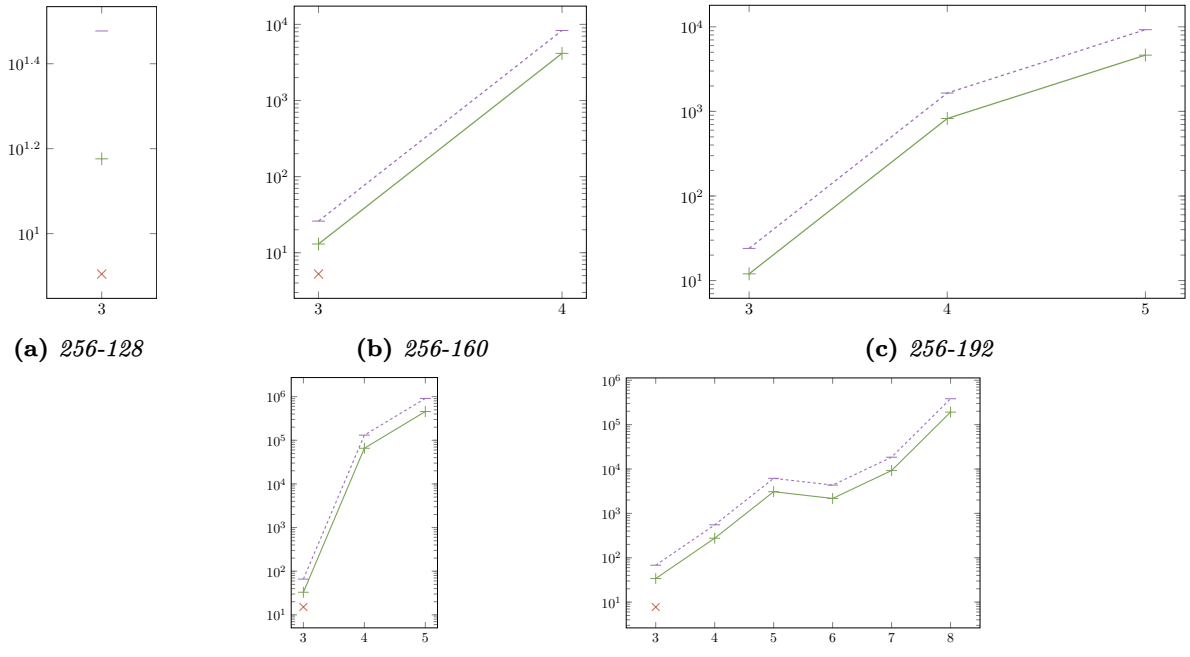
A.2.3 Results for Block length = 224 and 256



(a) *224-128* **(b)** *224-160* **(c)** *224-192*

(d) *224-224* **(e)** *224-256*

Figure A.5: *Solving time for* Global_{GAC} *(full CP) and* ***SAT / CP*** *models on* $\text{Rijndael}_{224-K_{len}}$. *Each instance is denoted* $224 - K_{len}$ *where* K_{len} *is the key size in bits.*



(a) *256-128* **(b)** *256-160* **(c)** *256-192*

(d) *256-224* **(e)** *256-256*

Figure A.6: *Solving time for* Global_{GAC} *(full CP) and* ***SAT / CP*** *models on* $\text{Rijndael}_{256-K_{len}}$. *Each instance is denoted* $256 - K_{len}$ *where* K_{len} *is the key size in bits.*

A.3 Related-key boomerang distinguisher computation time

K_{len}	N_r	D_r	obj_{s1}	obj_{s2}	optimality
128	3	1	2^{0}	2^{-6}	×
128	4	2	2^{0}	2^{-5}	×
128	5	3	2^{0}	2^{0}	✓
128	6	4	2^{-12}	2^{-20}	×
128	7	5	2^{-59}		
160	3	1	2^{0}	2^{-29}	×
160	4	2	2^{0}	2^{0}	✓
160	5	3	2^{0}	2^{0}	✓
160	6	4	2^{0}	2^{0}	✓
160	7	5	2^{-18}	2^{-35}	×
160	8	6	2^{-56}	2^{-104}	×
160	9	7	2^{-84}	2^{-95}	×
192	3	1	2^{0}	2^{0}	✓
192	4	2	2^{0}	2^{-5}	×
192	5	3	2^{0}	2^{0}	✓
192	6	4	2^{0}	2^{0}	✓
192	7	5	2^{0}	2^{0}	✓
192	8	6	2^{-12}	2^{-12}	✓
192	9	7	2^{-47}	2^{-51}	×
192	10	8	2^{-76}	2^{-136}	×
192	11	9	2^{-119}		
224	3	1	2^{0}		
224	4	2	2^{0}		
224	5	3	2^{0}		
224	6	4	2^{0}		
224	7	5	2^{0}		
224	8	6	2^{-24}		
224	9	7	2^{-47}		
224	10	8	2^{-78}		
224	11	9	2^{-90}		
224	12	10	2^{-155}		
256	3	1	2^{0}		
256	4	2	2^{0}		
256	5	3	2^{0}		
256	6	4	2^{0}		
256	7	5	2^{0}		
256	8	6	2^{-12}		
256	9	7	2^{-35}		
256	10	8	2^{-48}		
256	11	9	2^{-64}		
256	12	10	2^{-88}		
256	13	11	2^{-114}		

(a) $C_{len} = 128$

K_{len}	N_r	D_r	obj_{s1}	obj_{s2}	optimality
128	3	1	2^{0}	2^{-12}	×
128	4	2	2^{0}	2^{-5}	×
128	5	3	2^{0}	2^{0}	✓
128	6	4	2^{-12}	2^{-18}	✓
160	3	1	2^{0}	2^{-6}	×
160	4	2	2^{0}	2^{0}	✓
160	5	3	2^{0}	2^{0}	✓
160	6	4	2^{-11}	2^{-12}	✓
160	7	5	2^{-41}	2^{-45}	×
160	8	6	2^{-118}		
192	3	1	2^{0}		
192	4	2	2^{0}		
192	5	3	2^{0}		
192	6	4	2^{0}		
192	7	5	2^{0}		
192	8	6	2^{-47}		
192	9	7	2^{-102}		
224	3	1	2^{0}		
224	4	2	2^{0}		
224	5	3	2^{0}		
224	6	4	2^{0}		
224	7	5	2^{0}		
224	8	6	2^{-12}		
224	9	7	2^{-42}		
224	10	8	2^{-114}		
256	3	1	2^{0}		
256	4	2	2^{0}		
256	5	3	2^{0}		
256	6	4	2^{0}		
256	7	5	2^{0}		
256	8	6	2^{-18}		
256	9	7	2^{-36}		
256	10	8	2^{-87}		
256	11	9	2^{-120}		

(b) $C_{len} = 160$

K_{len}	N_r	D_r	obj_{s1}	obj_{s2}	optimality
128	3	1	2^{0}	2^{0}	✓
128	4	2	2^{0}	2^{0}	✓
128	5	3	2^{0}	2^{0}	✓
128	6	4	2^{-36}	2^{-36}	✓
160	3	1	2^{0}	2^{-6}	×
160	4	2	2^{0}	2^{0}	✓
160	5	3	2^{0}	2^{0}	✓
160	6	4	2^{-6}	2^{-6}	✓
160	7	5	2^{-46}	2^{-73}	×
192	3	1	2^{0}		
192	4	2	2^{0}		
192	5	3	2^{0}		
192	6	4	2^{0}		
192	7	5	2^{-28}		
192	8	6	2^{-84}		
224	3	1	2^{0}		
224	4	2	2^{0}		
224	5	3	2^{0}		
224	6	4	2^{0}		
224	7	5	2^{-29}		
224	8	6	2^{-54}		
256	3	1	2^{0}		
256	4	2	2^{0}		
256	5	3	2^{0}		
256	6	4	2^{0}		
256	7	5	2^{-6}		
256	8	6	2^{-35}		

(c) $C_{len} = 192$

K_{len}	N_r	D_r	obj_{s1}	obj_{s2}	optimality
128	3	1	2^{0}		
128	4	2	2^{0}		
128	5	3	2^{0}		
128	6	4	2^{-24}		
160	3	1	2^{0}		
160	4	2	2^{0}		
160	5	3	2^{0}		
160	6	4	2^{0}		
160	7	5	2^{-54}		
192	3	1	2^{0}		
192	4	2	2^{0}		
192	5	3	2^{0}		
192	6	4	2^{0}		
192	7	5	2^{-36}		

(d) $C_{len} = 224$

Table A.1: *The probabilities found for the different versions of* Rijndael. *Each table represents a variant* C_{len} *of* Rijndael. N_r *is the number of rounds,* D_r *is the number of rounds for which we compute the probability of distinction* ($D_r = N_r - 2$). obj_{s1} *is the upper bound found by* Step1-Opt. obj_{s2} *is the best probability found with Step2-Opt.* obj_{s2} *is not given either when we did not perform the computation or when the computation did not finish. The optimality column indicates whether the algorithm has found* (✓) *the optimal bound (complete search) or not* (×) *(incomplete search).*

A.4 Related-Key Boomerang Distinguisher on 9 rounds for Rijndael$_{128-160}$

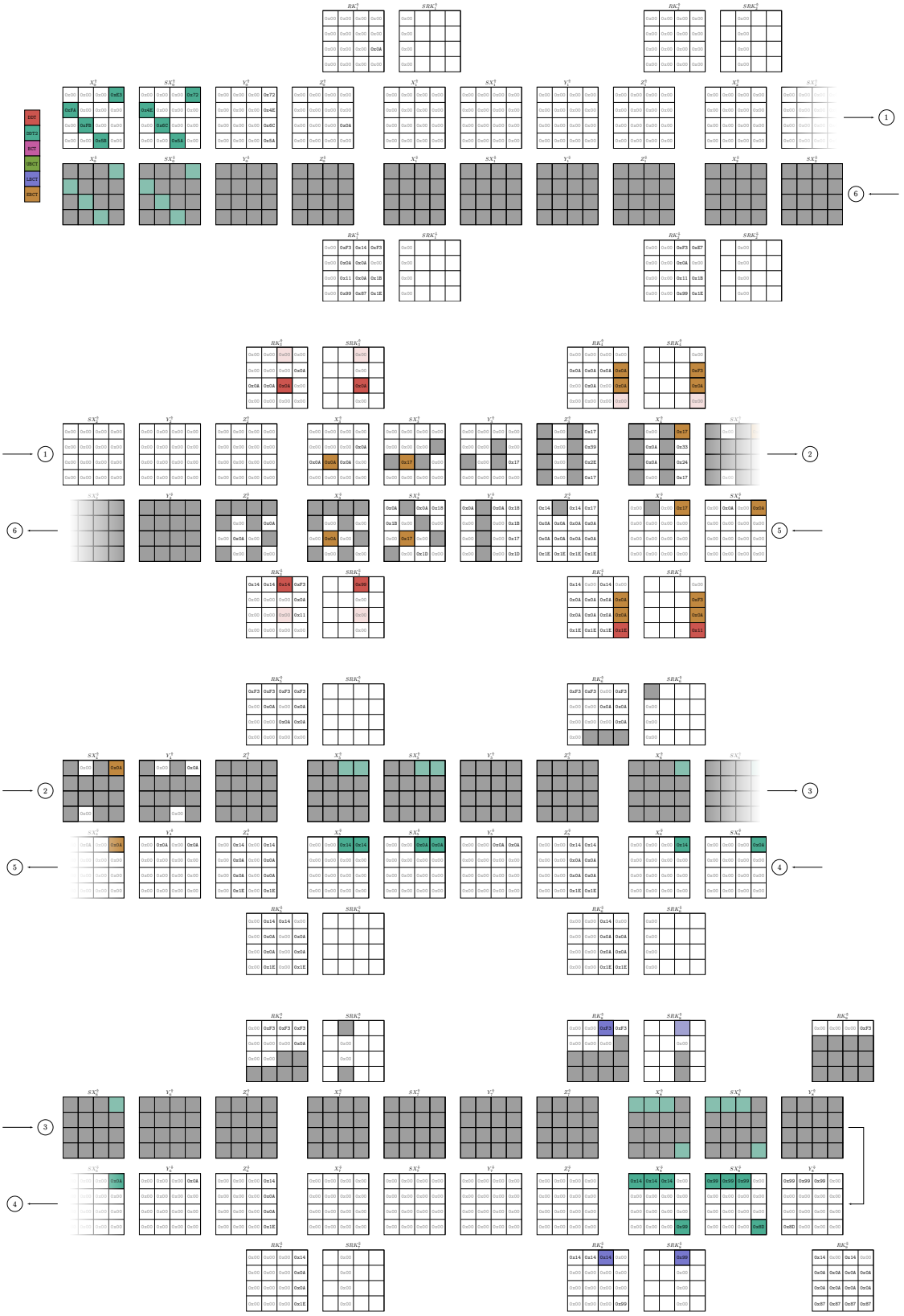


Figure A.7: *The* Rijndael$_{128-160}$ *9 rounds attack. The distinguisher works for rounds 1 to 8 and has a probability of* 2^{-56} *for the encryption part and of* 2^{-39} *for the* `KeySchedule`.

Appendix

B WARP

B.1 Model Searching for Rectangle Attacks with the Technique of [Zha+20]

New variables

$$\begin{aligned} 2t &= obj{-}cluster_{gain} \quad \triangleright \text{ we pose that: } p^2q^2r = 2^{-2t} \\ t &= \lceil 2t/2 \rceil \\ rb &= \sum_{k=0}^{31} \Delta_{X_{\text{up}}}[0,k] \times 4 \\ rf &= \sum_{k=0}^{31} \Delta_{X_{\text{lo}}}[N_r - 1, k] \times 4 \end{aligned} \tag{FA1}$$

$$data = \sigma + 64 + t \tag{FA2}$$

$$\begin{aligned} t_1 &= \sigma + 64 + t + mb - 4 \quad \triangleright \text{ we pose that: } s = 2^{2\sigma} \\ t_2 &= 2\sigma + mb - 128 + 2rf + 2t \\ time &= \max(t_1, t_2) \end{aligned} \tag{FA3}$$

$$mb = \sum_{alt \in \{0,1\}} \sum_{k=0}^{15} \texttt{guess}_{\texttt{key}}[alt, k] \times 4 \tag{FA4}$$

New constraints

$\forall k \in [0; \texttt{BR}/2[,$

$$\begin{aligned} \texttt{guess}_{\texttt{key}}[0,k] = \#\{\texttt{known}[i, 2\times k+1] \mid i \in [0; N_b[\ \wedge \ i \mod 2 = 0\} \geq 1 \wedge \\ \texttt{guess}_{\texttt{key}}[1,k] = \#\{\texttt{known}[i, 2\times k+1] \mid i \in [0; N_b[\ \wedge \ i \mod 2 = 1\} \geq 1 \end{aligned} \tag{FA5}$$

$\forall i \in [0; N_b[, \quad \forall k \in [0; \texttt{BR}/2[,$

$$(\Delta_{X_{\text{up}}}[i+1, \pi_{odd}[k]] \vee \Delta_{X_{\text{up}}}[i, 2\times k]) \implies \Delta_{X_{\text{up}}}[i, 2\times k+1] \tag{FA6}$$

$\forall i \in [0; N_b - 1[, \quad \forall k \in [0; \texttt{BR}/2[,$

$$\texttt{known}[i+1, \pi_{odd}[k]] \implies (\texttt{known}[i, 2\times k] \wedge \texttt{known}[i, 2\times k+1]) \tag{FA7}$$

$\forall i \in [0; N_b[, \quad \forall k \in [0; \texttt{BR}/2[,$

$$\Delta_{X_{\text{up}}}[i+1, \pi_{even}[k]] \implies \texttt{known}[i, 2\times k] \tag{FA8}$$

$\forall k \in [0; \texttt{BR}/2[$

$$\texttt{known}[N_b - 1, 2\times k+1] = \texttt{false} \tag{FA9}$$

$\forall i \in [N_b + N_d, Nr[, \quad \forall k \in [0; \texttt{BR}/2[,$

$$\Delta_{X_{\text{lo}}}[i, 2\times k+1] \vee \Delta_{X_{\text{lo}}}[i, 2\times k] \implies \Delta_{X_{\text{lo}}}[i+1, \pi_{odd}[k]] \tag{FA10}$$

Model B.1: *Attack extension for the attack technique of Zhao* et al. *[Zha+20].*

B.2 Application of our Technique to TWINE and LBlock–s

To illustrate the flexibility of our tool, this section reports the results obtained when applying it to two well-known Feistel ciphers, TWINE and LBlock–s. TWINE [Suz+13] is a 64-bit block cipher with a Type-II **GFN** structure and LBlock–s is used in the authenticated encryption LAC [**zhang2014lac**] submitted to the `CAESAR` competition. LBlock–s is a simplified version of the original cipher LBlock [WZ11] which uses only one S-box instead of the 8 original ones and admits 16 rounds or 32 rounds according to where it is used in LAC. It is also a 64-bit cipher and it could also be represented as a Type-II **GFN** as shown in [SN14]. This is that representation that we used for our models. Then, for those two ciphers, we apply our method for computing the boomerang clusters and the results are summed up in Table B.1.

Cipher	Distinguishers	Rounds	Probability	Ref.
TWINE	Boomerang distinguisher	15	$2^{-58.92}$	[TB21b]
TWINE	Boomerang distinguisher	16	$2^{-61.62}$	[TB21b]
TWINE	Boomerang Distinguisher + Clustering	15	$2^{-47.7}$	This PhD thesis
TWINE	Boomerang Distinguisher + Clustering	16	$2^{-59.8}$	This PhD thesis
LBlock-s	Boomerang distinguisher	15	$2^{-58.64}$	[TB21b]
LBlock-s	Boomerang Distinguisher + Clustering	16	$2^{-56.14}$	[Bou+20]
LBlock-s	Boomerang Distinguisher + Clustering	16	$2^{-54.8}$	This PhD thesis

Table B.1: *Summary of the results for computing the best boomerang clusters for* TWINE *and* LBlock–s.

B.3 23-round Boomerang distinguisher on WARP

B.3.1 Upper trail

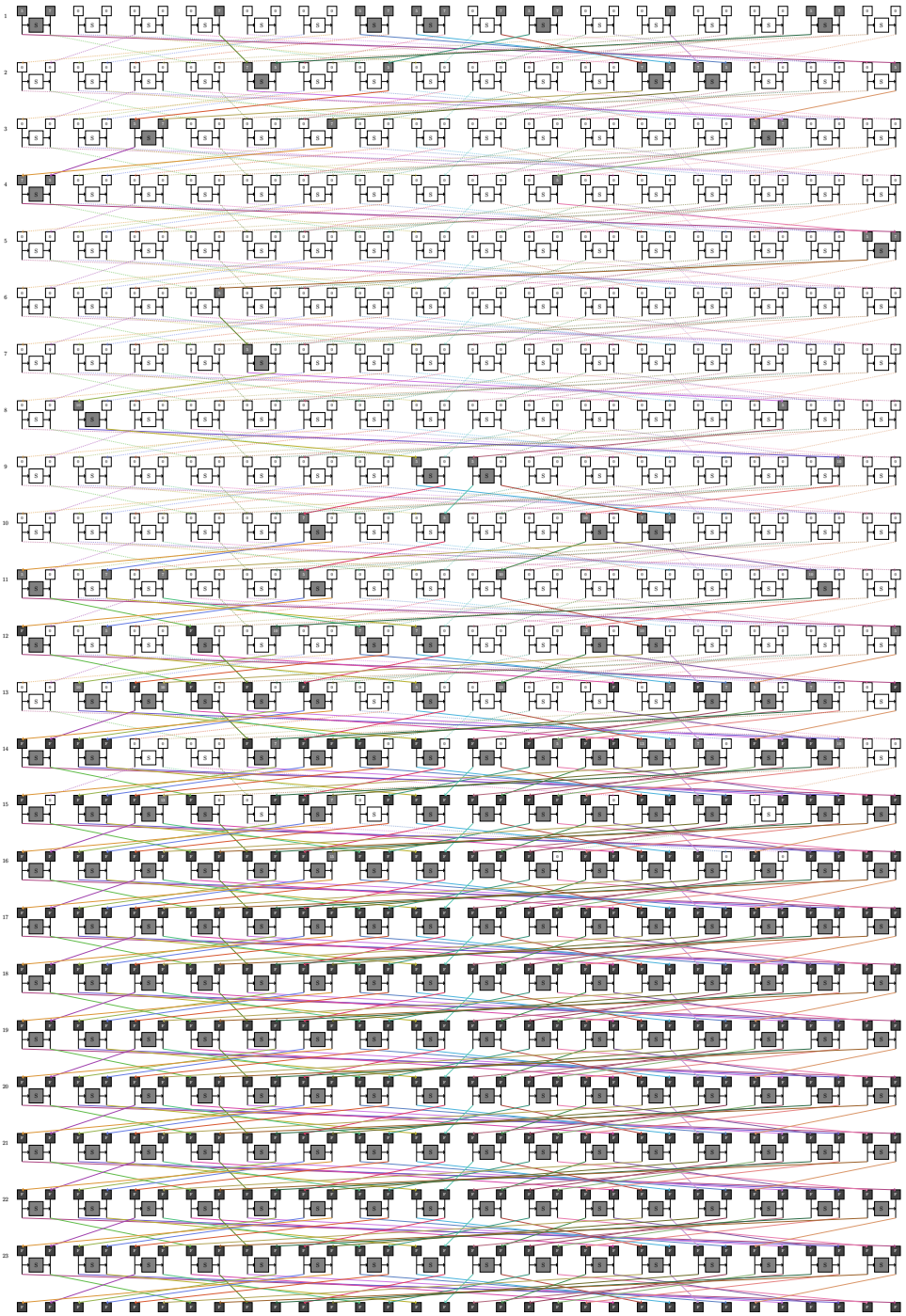


Figure B.1: *Upper trail of the 23-round Boomerang distinguisher on* WARP.

B.3.2 Lower trail

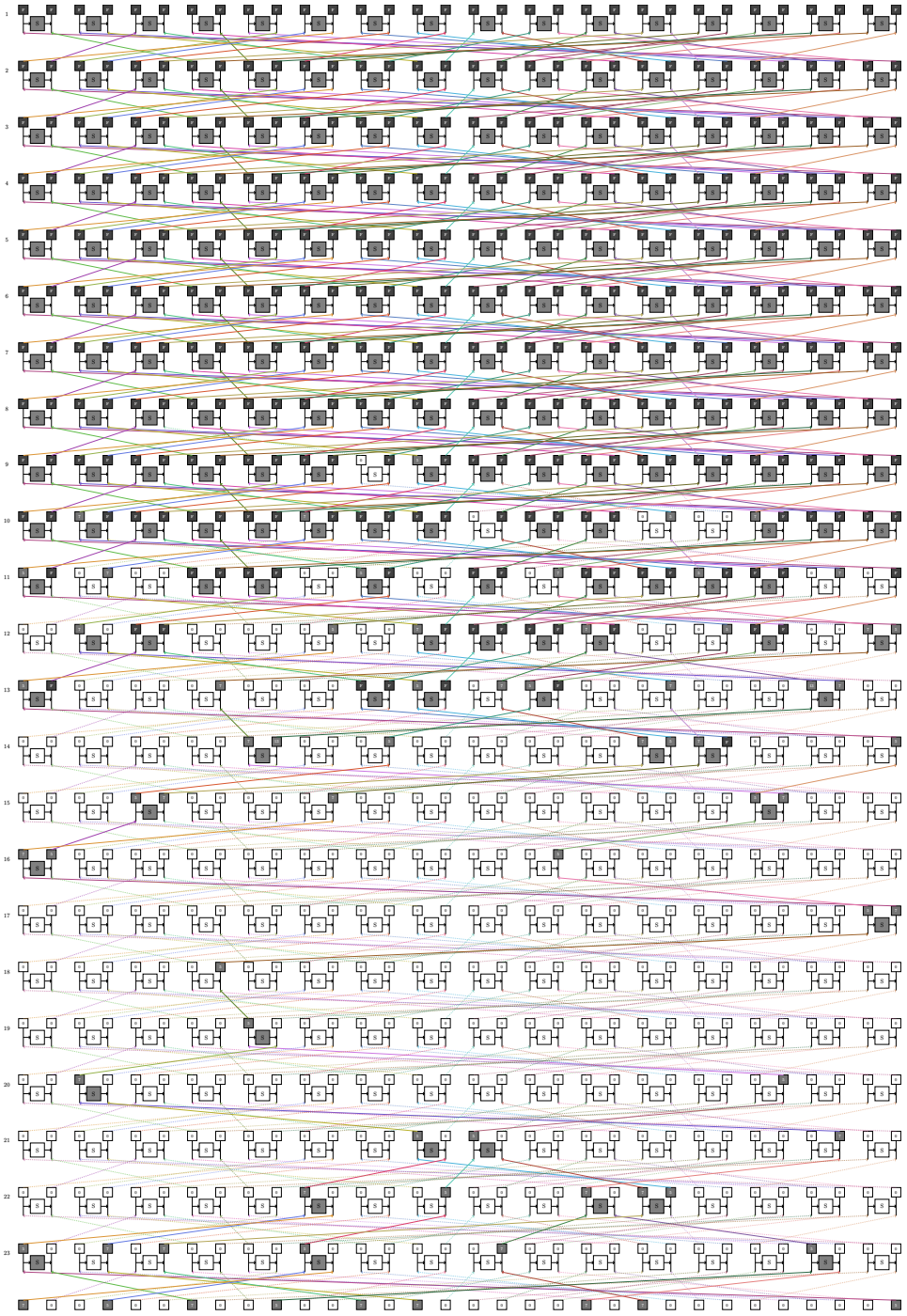


Figure B.2: *Lower trail of the 23-round Boomerang distinguisher on* WARP.

B.4 26-round Rectangle attack on WARP

B.4.1 Upper trail

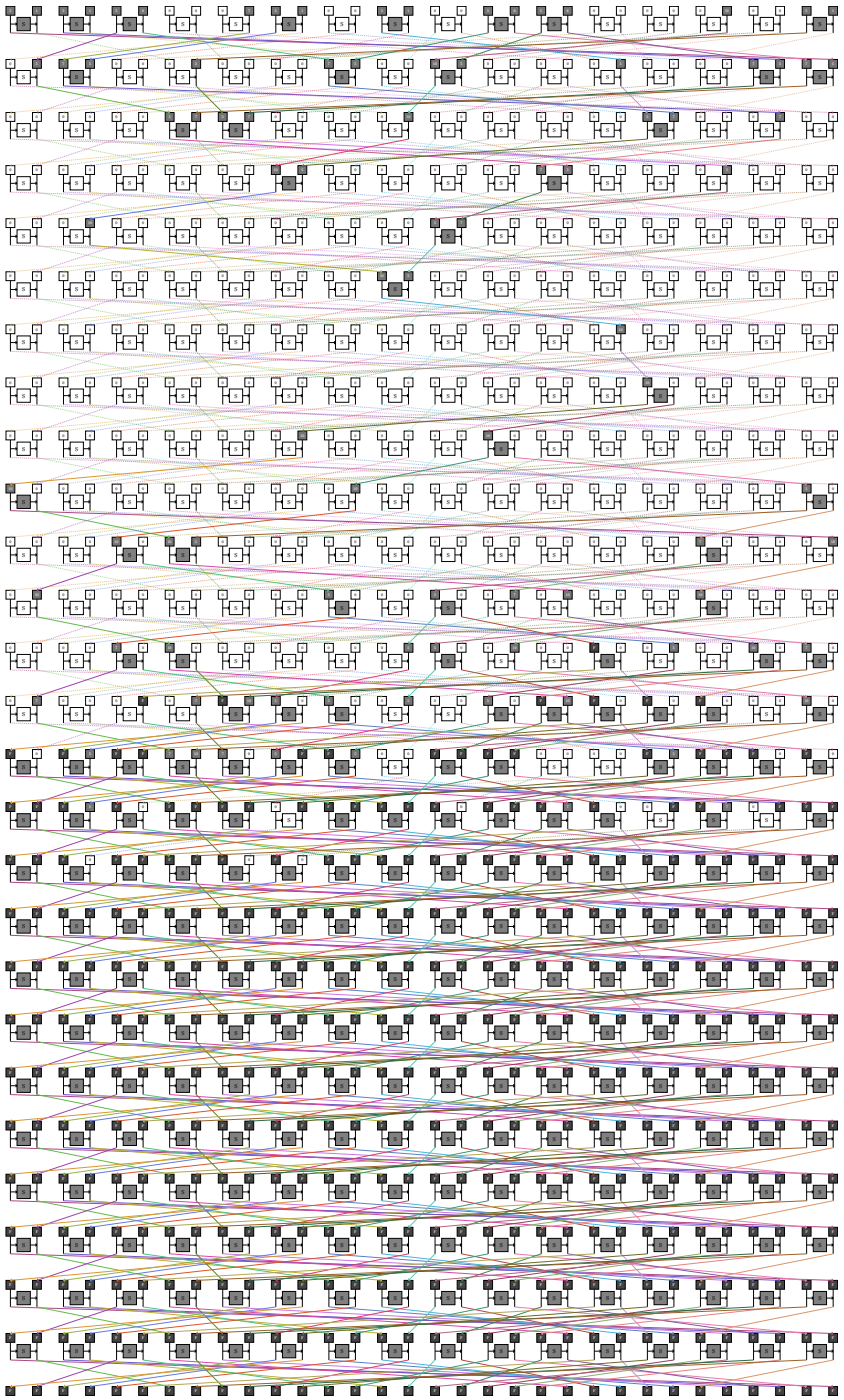


Figure B.3: *Upper trail of the 26-round Rectangle attack on* WARP.

B.4.2 Lower trail

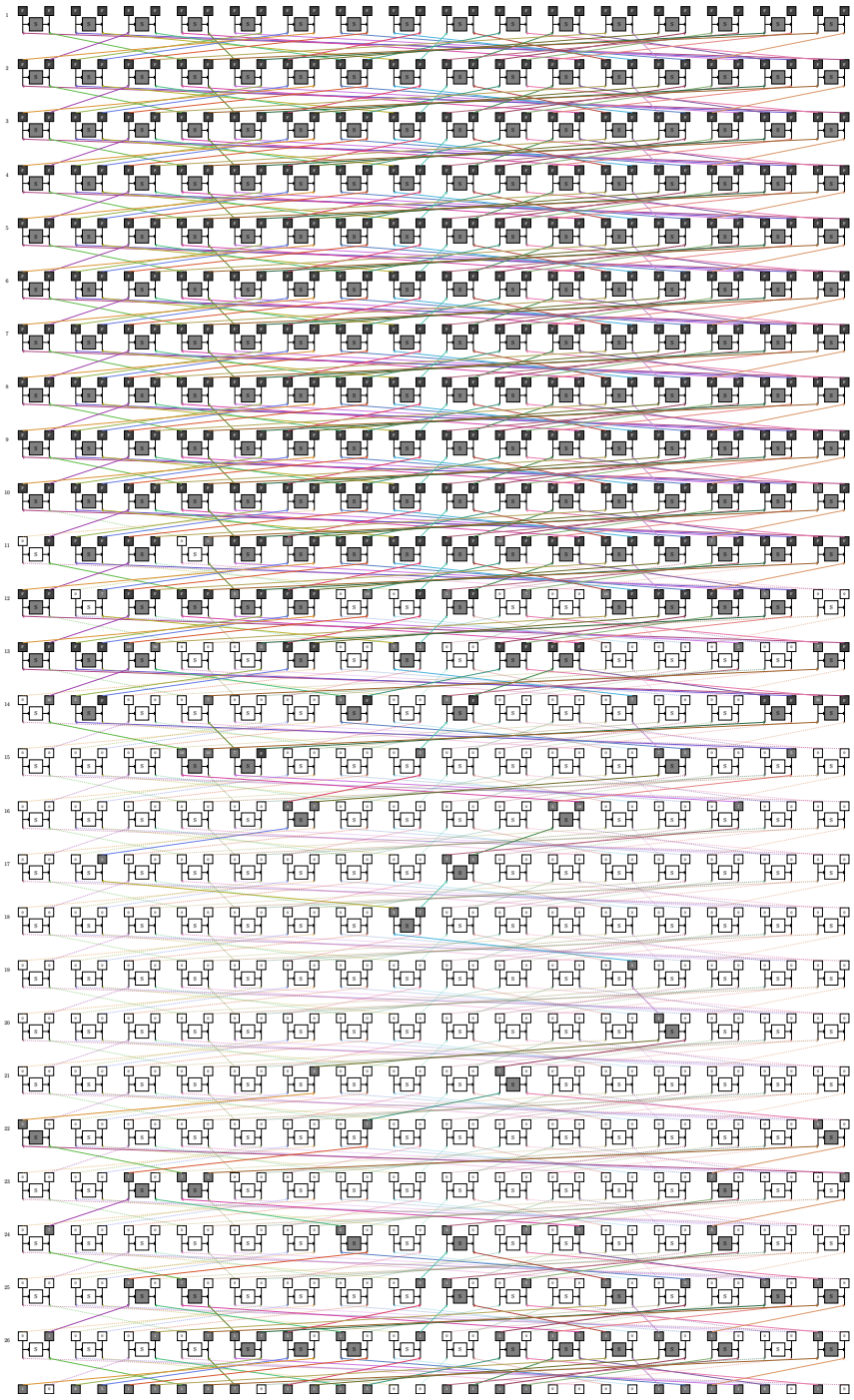


Figure B.4: *Lower trail of the 26-round Rectangle attack on* WARP.

FOLIO ADMINISTRATIF

THESE DE L'INSA LYON MEMBRE DE L'UNIVERSITÉ DE LYON

NOM : ROUQUETTE
(avec précision du nom de jeune fille, le cas échant)

DATE de SOUTENANCE : 15/11/2022

Prénoms : Loïc, Roger

TITRE : Improving scalability and reusability of differential cryptanalysis models using constraint programming.

NATURE : Doctorat

Numéro d'ordre : 2022 ISAL0094

Ecole doctorale: Informatique et Mathématiques

Spécialité : Informatique

RESUME :
Dans cette thèse, nous proposons de nouveaux outils de modélisation destinés à la cryptanalyse différentielle à l'aide de la programmation par contraintes. En effet, de récentes publications ont démontré l'efficacité des approches déclaratives dans la modélisation et la résolution de problèmes cryptographiques. Dans cette thèse, nous étendons la recherche automatique de caractéristiques différentielles du chiffrement standard AES à sa version non standard, à savoir Rijndael. Cette extension s'accompagne d'une nouvelle procédure de recherche permettant de mieux interconnecter les différentes étapes de la recherche et ainsi améliorer les temps de calcul, cette première modélisation a permis de trouver une nouvelle attaque sur une des instances de Rijndael. Dans un second temps, nous introduisons la contrainte globale "AbstractXOR" permettant de modéliser de façon plus précise un ensemble d'équations XOR dans le cas de la recherche d'une différentielle tronquée. Nous prouvons que la vérification de cette contrainte est un problème NP-Complet et nous décrivons un algorithme permettant de vérifier en temps polynomial la satisfaction de cette contrainte sur une relaxation du problème. Par la suite, nous adaptons la modélisation des attaques Boomerangs de Delaune et al. au chiffrement Rijndael en intégrant la partie non linéaire du KeySchedule. Cette modélisation nous a permis de trouver deux nouvelles attaques sur ce chiffrement. Pour finir, nous montrons qu'il est possible d'adapter la modélisation de Delaune et al. aux chiffrements Feistel. Nous appliquons cette adaptation aux chiffrements WARP, Twine et LBlock-s, ce qui nous a permis de trouver de nouveaux distingueurs pour les trois chiffrements. Nous montrons également comment intégrer la phase KeyRecovery au processus d'optimisation afin d'améliorer la complexité globale de l'attaque, ce qui nous a permis de trouver une nouvelle attaque sur 26 tours de WARP.

MOTS-CLÉS :
Differential cryptanalysis, Single-Key, Related-Key, Boomerang Attack, Retangle Attack, AES, Rijndael, Midori, WARP, Twine, LBlock-s, Constraint Programming, CP, SAT, Global Constraint

Laboratoire (s) de recherche : LIRIS, UMR 5205 CNRS / CITI, INRIA

Directeur de thèse : SOLNON Christine, MINIER Marine

Président de jury : LECOUTRE Christophe

Composition du jury :
- DEQUEN Gilles
- BOURA Christina
- SOLNON Christine
- NAYA PLASENCIA Maria
- DERBEZ Patrick
- MINIER Marine