Université Paris Diderot
(Paris 7)

École Normale Supérieure
Équipe Crypto

Thèse de doctorat

Cryptanalyse de primitives symétriques basées sur le chiffrement AES

Spécialité : Informatique

présentée et soutenue publiquement le 24 septembre 2013 par

Jérémy Jean

pour obtenir le grade de

Docteur de l'Université Paris Diderot

devant le jury composé de

Directeur de thèse :	Pierre-Alain Fouque	(Université de Rennes 1, France)
Rapporteurs :	Anne Canteaut	(INRIA, France)
	Henri Gilbert	(ANSSI, France)
Examinateurs :	Arnaud Durand	(Université Paris Diderot, France)
	Franck Landelle	(DGA, France)
	Thomas Peyrin	(Nanyang Technological University, Singapour)
	Vincent Rijmen	(Katholieke Universiteit Leuven, Belgique)

Remerciements

Je souhaite remercier toutes les personnes qui ont contribué de près ou de loin à mes trois années de thèse. Je remercie Arnaud Durand, Franck Landelle, Thomas Peyrin et Vincent Rijmen qui ont accepté de participer à ma soutenance de fin de thèse. Je remercie chaleureusement Anne Canteaut en Henri Gilbert pour avoir accepté la lourde tâche de rapporteur et pour avoir relu très attentivement ce manuscrit. Je les remercie sincèrement pour leur efficacité et leur bonne humeur quotidienne.

Je remercie également les organismes qui ont financé mes travaux de recherche et mes nombreux déplacements durant ces trois années : la DGA et le projet ANR Saphir II qui ont cofinancé ma bourse de thèse, l'ENS et l'INRIA pour l'environnement matériel ainsi que NTU pour les séjours que j'ai eu la chance de réaliser à Singapour.

Je suis extrêmement reconnaissant à Pierre-Alain pour avoir suivi mes travaux de recherche tout en me laissant une très grande liberté d'action. Merci à lui de m'avoir transmis son expérience de la recherche et pour sa disponibilité quand j'en ai eu besoin. Je remercie également Thomas Peyrin pour toutes les collaborations que nous avons pu avoir, et pour m'accueillir en tant que postdoc dans son équipe du NTU à Singapour en 2014. Au-delà de son enthousiasme scientifique sans limite, je le remercie également en tant qu'ami et plus spécialement pour m'avoir fait découvrir Catane.

Au cours de ces trois années, j'ai eu la chance de côtoyer de nombreuses personnes du domaine de la cryptographie, avec qui j'ai eu plaisir à travailler pour apporter quelques contributions scientifiques. Je remercie donc tous mes coauteurs : Patrick Derbez, Pierre-Alain Fouque, María Naya-Plasencia, Ivica Nikolić, Thomas Peyrin, Martin Schläffer, Lei Wang et Shuang Wu. De plus, je souhaite remercier toutes les personnes avec qui j'ai pu échanger de manière plus informelle et qui ont indirectement contribué à enrichir mes connaissances, et en particulier Orr Dunkelman, Henri Gilbert, Antoine Joux, Christian Rechberger, Yu Sasaki et Adi Shamir.

Je tiens également à saluer tous les thésards et chercheurs que j'ai eu le plaisir de croiser à l'ENS ou ailleurs : Aurélie, Aurore, Céline, Charles, Christina, Damien, Gaëtan, Jean-Christophe, Léo, Michel, Miriam, Olivier, Mario, Rodolphe, Tancrède, Vadim, Yannick et Yuanmi. Je remercie plus particulièrement David Pointcheval pour sa direction de l'équipe Crypto de l'ENS et qui a été directeur de thèse officiel pendant ma première année de thèse.

Cette thèse n'aurait pas été possible sans le soutien administratif de l'ENS. Je remercie particulièrement Joëlle, Lise-Marie, Michelle et Valérie pour leur efficacité. Je salue également tout le service informatique pour la mise à disposition et l'installation des machines et serveurs qui m'ont beaucoup servi.

Enfin, je remercie toute ma famille et tous mes proches pour leur soutien durant ces trois années de thèse. Merci beaucoup à tous ceux qui ont relu et corrigé des bouts de ce manuscrit. Au-delà de la thèse, merci beaucoup à mes parents pour leurs encouragements et leur confiance sans limite, et merci à ma sœur Amélie et à mes petits frères Benjamin et Sébastien. Finalement, je ne pourrais jamais assez remercier Ève pour son soutien au quotidien. Merci à toi pour avoir supporté mes longues soirées et week-ends de cryptanalyse et de rédaction. Un grand merci aussi pour accepter le défi de retourner vivre à l'étranger : sans toi ce départ n'aurait jamais pu être le même.

À Alice, Bob et Ève.

Table des matières

1 Introduction **1**
- 1.1 Histoire de la cryptographie 1
- 1.2 La cryptographie aujourd'hui 3
  - 1.2.1 Généralités 3
  - 1.2.2 Cryptographie asymétrique 5
    - 1.2.2.1 Algorithme de chiffrement 5
    - 1.2.2.2 Signature électronique 5
  - 1.2.3 Cryptographie symétrique 6
    - 1.2.3.1 Algorithmes de chiffrement par bloc 6
    - 1.2.3.2 Algorithmes de chiffrement par flot 7
    - 1.2.3.3 Fonctions de hachage 7
    - 1.2.3.4 Code d'authentification de message (MAC) 7
  - 1.2.4 Notions de cryptanalyse 8
- 1.3 Fonctions de hachage 9
  - 1.3.1 Fonctions de hachage cryptographique 10
  - 1.3.2 Paradoxe des anniversaires 11
  - 1.3.3 Modes opératoires 13
    - 1.3.3.1 Construction de Merkle-Damgård 13
    - 1.3.3.2 Attaque par extension (*extension attack*) 13
    - 1.3.3.3 Attaque par multicollisions 14
    - 1.3.3.4 Construction wide-pipe 14
    - 1.3.3.5 Construction sponge 15
- 1.4 Algorithmes de chiffrement par bloc 16
  - 1.4.1 Définition 16
  - 1.4.2 Construction itérée 17
  - 1.4.3 Modes opératoires 18
  - 1.4.4 Fonction de compression 19
  - 1.4.5 Cryptanalyse des algorithmes de chiffrement par bloc 20
    - 1.4.5.1 Attaque par le milieu 21
    - 1.4.5.2 Distance d'unicité 22
    - 1.4.5.3 Modèles d'attaquants 22

2 Présentation des Travaux **25**
- 2.1 Présentation des travaux 25
- 2.2 Liste des publications 31

3 Differential Cryptanalysis . . . **33**
3.1 Preliminaries . . . 34
3.1.1 Differentials . . . 34
3.1.2 Iterated functions . . . 36
3.1.3 Differential characteristics . . . 37
3.2 Block ciphers . . . 39
3.2.1 Basic key recovery attack . . . 39
3.2.1.1 Efficiency evaluation . . . 41
3.2.1.2 Improved variants . . . 42
3.2.2 Resistance against differential cryptanalysis . . . 42
3.3 Markov Ciphers . . . 43
3.4 Other forms of differential cryptanalysis . . . 46
3.4.1 Truncated differential cryptanalysis . . . 46
3.4.2 Impossible differential cryptanalysis . . . 47
3.4.2.1 Applications . . . 47
3.4.2.2 Resistance against impossible differential cryptanalysis . . . 48
3.4.3 Boomerang attack . . . 48
3.4.3.1 Improvements . . . 50
3.4.3.2 Applications . . . 51
3.4.4 Related-key attacks . . . 51
3.5 Hash functions . . . 54
3.5.1 Generalities . . . 54
3.5.2 Rebound attack . . . 55

4 Description of the AES and Cryptanalytic Results . . . **57**
4.1 The AES competition . . . 57
4.2 Description of the AES block cipher . . . 58
4.2.1 Key scheduling algorithms . . . 59
4.2.2 Round function . . . 61
4.2.3 The substitution layer . . . 62
4.2.4 The permutation layer . . . 64
4.2.4.1 ShiftRows . . . 64
4.2.4.2 MixColumns . . . 65
4.3 AES-like permutations . . . 66
4.4 Notable cryptanalytic results . . . 67
4.4.1 Square attack . . . 67
4.4.1.1 Attack on 4 rounds . . . 69
4.4.1.2 Attack on 5 rounds . . . 69
4.4.1.3 Attack on 6 rounds . . . 70
4.4.1.4 Extensions to the larger AES variants . . . 71
4.4.2 Improved square attack with partial sums . . . 71
4.4.2.1 First improvement . . . 71
4.4.2.2 Second improvement . . . 72
4.4.2.3 Extension to more rounds . . . 74
4.4.2.4 The herd attack . . . 74
4.4.3 Collision attack . . . 75

4.4.3.1 Distinguishers . . . 75
4.4.3.2 Collision attack on 4 rounds . . . 77
4.4.3.3 Extension to 7 rounds . . . 77
4.4.4 Impossible differential attack . . . 79
4.4.4.1 Bahrak and Aref attack . . . 80
4.4.4.2 Improved variants . . . 82
4.4.5 Related-key attacks . . . 84
4.4.5.1 Related-key boomerang attack on 7-round AES-128 . . . 85
4.4.6 Summary of all the attacks . . . 86

5 AES in the Secret-Key Model . . . 89
5.1 A class of attacks against AES . . . 90
5.1.1 Initial attacks . . . 90
5.1.2 Generalizations . . . 90
5.1.3 Attack framework . . . 92
5.1.3.1 Attack by Demirci and Selçuk . . . 92
5.1.3.2 Attack by Dunkelman, Keller and Shamir . . . 93
5.1.4 Improvements . . . 94
5.2 New attacks on 7-round AES . . . 95
5.2.1 Generalities . . . 95
5.2.2 Efficient tabulation . . . 95
5.2.3 A simple attack . . . 98
5.2.3.1 Precomputation phase . . . 98
5.2.3.2 Online phase . . . 98
5.2.4 Efficient Attack . . . 100
5.2.5 Key recovery . . . 102
5.3 Extensions to 8 and 9 rounds . . . 104
5.3.1 Attack on 8-round AES-192 . . . 104
5.3.2 Attack on 8-round AES-256 . . . 108
5.3.3 Attack on 9-round AES-256 . . . 109

6 AES in the Related-Key Model . . . 111
6.1 Generalities . . . 112
6.1.1 Motivations . . . 112
6.1.2 Graph traversal algorithms . . . 113
6.1.3 Structural evaluation . . . 116
6.2 Definitions . . . 118
6.2.1 Substitution-Permutation Network . . . 118
6.2.2 Truncated and actual differences . . . 119
6.2.2.1 The substitution layer . . . 119
6.2.2.2 The permutation layer for AES-like ciphers . . . 120
6.3 Related-key differential characteristics . . . 120
6.3.1 Differential characteristic search . . . 120
6.3.2 Precomputation phase . . . 122
6.3.2.1 The graph G_{BC} . . . 123
6.3.2.2 The graph G_{KS} . . . 124

6.3.3 Online phase . . . 124
6.4 Enhanced Markov process . . . 127
6.4.1 The Markov assumption and actual differences . . . 127
6.4.2 Block cipher state compression . . . 128
6.4.3 Evaluating the number of nodes/edges of G_{BC} and G_{KS} . . . 128
6.4.3.1 Number of nodes . . . 128
6.4.3.2 Number e_{BC} of edges in G_{BC} . . . 129
6.4.3.3 Number e_{KS} of edges in G_{KS} . . . 129
6.4.4 More complete Markov process . . . 130
6.4.4.1 New state compression . . . 131
6.4.4.2 Representation of truncated subkeys . . . 131
6.4.5 Explanations . . . 132
6.5 Applications to SPN and AES-128 . . . 133
6.5.1 Structural evaluation of SPN AES-like ciphers . . . 133
6.5.1.1 Complexity evaluation . . . 135
6.5.2 Differential characteristics results for AES-128 . . . 135

7 AES in the Open-Key Model . . . 143
7.1 Generalities . . . 143
7.1.1 Motivations . . . 143
7.1.2 Rebound technique . . . 145
7.1.3 Limited-birthday distinguisher . . . 146
7.2 Known-key model . . . 149
7.2.1 Distinguishers for 7 rounds . . . 149
7.2.1.1 Integral distinguisher . . . 149
7.2.1.2 Rebound attack . . . 151
7.2.1.3 Improved distinguisher: start-from-the-middle technique . . . 153
7.2.2 Distinguisher for 8 rounds . . . 155
7.2.2.1 Fully-active characteristic . . . 155
7.2.2.2 Non-fully-active characteristic . . . 158
7.3 Chosen-key model . . . 161
7.3.1 Distinguisher for 7-round AES . . . 162
7.3.1.1 Distinguishing algorithm for AES-128 . . . 162
7.3.1.2 Experimental verification . . . 166
7.3.1.3 Success probability . . . 166
7.3.1.4 Extension to 7-round AES-256 . . . 167
7.3.1.5 Extension to 8-round AES-256 . . . 167
7.3.2 Distinguisher for 8-round AES . . . 168
7.3.2.1 Distinguishing algorithm for AES-128 . . . 168
7.3.2.2 Experimental verification . . . 172
7.3.2.3 Extension to 9-round AES-256 . . . 172
7.3.3 Distinguisher for 9-round AES-128 . . . 173
7.3.3.1 Distinguishing algorithm . . . 173
7.3.3.2 Generic case . . . 176

8 Improved Rebound Algorithms . . . 179

8.1 Description of some AES-like primitives . . . 180
8.1.1 Description of Grøstl . . . 180
8.1.2 Description of PHOTON . . . 182
8.1.3 Description of LED . . . 183
8.1.4 Description of Whirlpool . . . 184
8.2 Improved Inbound Part . . . 185
8.2.1 Fully-active truncated differential characteristic . . . 186
8.2.1.1 The truncated differential characteristic . . . 186
8.2.1.2 Finding a conforming pair . . . 187
8.2.1.3 Comparison with the ideal case . . . 191
8.2.2 Non-fully-active truncated differential characteristic . . . 192
8.2.2.1 The generic truncated characteristic . . . 192
8.2.2.2 Finding a conforming pair . . . 193
8.2.2.3 Comparison with ideal case . . . 195
8.2.3 Application to Grøstl-256 permutations . . . 197
8.2.3.1 Three fully-active states . . . 197
8.2.3.2 Non-fully-active characteristic . . . 198
8.2.4 Distinguisher for 10-round Grøstl-512 . . . 198
8.2.4.1 The truncated differential characteristic . . . 198
8.2.4.2 Finding a conforming pair . . . 200
8.2.4.3 Comparison with ideal case . . . 204
8.2.5 Distinguishers for reduced PHOTON permutations . . . 205
8.3 Improved Outbound Part . . . 205
8.3.1 Multiple limited-birthday and generic complexity . . . 206
8.3.1.1 Structures of input data . . . 208
8.3.1.2 Generic algorithm . . . 209
8.3.2 Truncated characteristic with relaxed conditions . . . 210
8.3.2.1 Relaxed 9-round distinguisher for AES-like permutation . . . 210
8.3.2.2 Comparison with ideal case . . . 212
8.3.3 Applications . . . 212
8.3.3.1 AES . . . 212
8.3.3.2 ECHO . . . 214
8.3.3.3 Grøstl . . . 215
8.3.3.4 LED . . . 216
8.3.3.5 PHOTON . . . 216
8.3.3.6 Whirlpool . . . 217

9 Rebound Attacks on ECHO Hash Function . . . 219
9.1 Description of ECHO . . . 219
9.1.1 Original description . . . 220
9.1.2 Alternative description . . . 222
9.1.2.1 Super-SBox . . . 222
9.1.2.2 SuperMixColumns . . . 223
9.1.3 Current cryptanalysis . . . 226
9.2 Attacks on ECHO-256 . . . 227
9.2.1 Collision attack on the 4-round compression function . . . 227

9.2.1.1 Truncated differential characteristic . . . 228
9.2.1.2 Super-SBox sparse differentials . . . 228
9.2.1.3 Finding a message pair . . . 230
9.2.1.4 Step 1 - Partial first inbound . . . 231
9.2.1.5 Step 2 - Second inbound . . . 232
9.2.1.6 Step 3 - Merging the two inbounds . . . 234
9.2.1.7 Step 4 - Reaching the collision . . . 234
9.2.1.8 Experimental verification . . . 237
9.2.2 Collision attack on the 4-round hash function . . . 237
9.2.3 Collision attack on the 5-round hash function . . . 239
9.2.3.1 The truncated differential characteristic . . . 239
9.2.3.2 Colliding subspace differences . . . 240
9.2.3.3 High-level outline of the attack . . . 242
9.2.3.4 Details of the attack . . . 243
9.2.4 Distinguisher for the 7-round compression function . . . 247
9.2.4.1 Finding pairs between S6 and S23 . . . 249
9.2.4.2 Finding pairs between S30 and S47 . . . 250
9.2.4.3 Merging solutions . . . 250
9.2.5 Collision attack on the 6-round compression function . . . 252
9.2.6 Chosen-salt attacks on the compression function . . . 252
9.2.6.1 The truncated differential characteristic . . . 252
9.2.6.2 Outline of the attack . . . 254
9.2.6.3 Finding right pairs . . . 254
9.2.6.4 Chosen-salt collision attack for 6 rounds . . . 256
9.2.6.5 Chosen-salt distinguisher for 7 rounds . . . 256

Conclusions . . . 259

Bibliography . . . 261

Appendices . . . 275

A Advanced Encryption Standard . . . 277
A.1 AES S-Box lookup table . . . 277
A.2 PRESENT S-Box lookup table . . . 277
A.3 `Whirlpool` S-Box lookup table . . . 277

List of Figures . . . 278

List of Tables . . . 282

CHAPITRE 1

Introduction

1.1 Histoire de la cryptographie

Anciennement considérée comme un art, la cryptographie s'intègre dans le domaine plus général de la cryptologie, désormais reconnue comme science à part entière. Étymologiquement, la cryptologie est la science du secret, dans laquelle la cryptographie s'intéresse aux méthodes de protection de messages ou documents de types très variés. Parallèlement à la conception de méthodes de protection, on trouve également la *cryptanalyse* qui consiste à casser ces codes secrets, c'est-à-dire à se positionner comme l'ennemi qui cherche à retrouver le message original. Les applications de la cryptographie dans la vie courante sont diverses et adaptables à tout type de scénario où deux individus sont capables d'échanger des informations. Le but premier de cette science est de donner la possibilité aux individus de communiquer d'une manière protégée. À cette fin, on suppose que deux personnes disposent d'un *canal* de communication quelconque par lequel un échange d'informations est possible. Le canal de communication est considéré dans le domaine public et est donc accessible à tout le monde. Il peut être de nature différente suivant le scénario, mais on peut imaginer qu'il s'agisse d'un courrier postal, d'un courrier électronique, d'un câble téléphonique, d'ondes radios, d'une petite annonce dans un journal, etc.

Les premières utilisations connues de la cryptographie remontent à l'Antiquité, où la plus anciennetrace de message chiffré a été retrouvéee sur une table en argile sur les bords du Tigre en Irak. Un potier babylonien y avait gravé la recette de son vernis, voulant garder secrète la clé de sa réussite. La méthode de chiffrement utilisée a été découverte par des archéologues, et consistait simplement à supprimer certaines lettres de la recette et à remplacer certains mots par d'autres. Au fil des années, les motivations militaires ont conduit les Hommes à développer de nouvelles méthodes de chiffrement plus robustes afin d'éviter que les tactiques ou plans de bataille ne tombent dans les mains de l'ennemi. Les Spartiates ont ainsi inventé le premier dispositif militaire connu : la scytale, ou bâton de Plutarque. La scytale en elle-même est un bâton de bois, dont le diamètre est connu uniquement de l'émetteur et du destinataire du message. Pour chiffrer un message, on enroule un étroit morceau de papyrus autour de la scytale pour ensuite y écrire le message. Ainsi, seul le destinataire capable d'enrouler à nouveau le papyrus autour de sa scytale peut lire le message original. Un attaquant qui parviendrait à récupérer un message chiffré et qui devinerait la manière de chiffrer devrait alors essayer plusieurs diamètres de bâtons différents pour retrouver le message original.

Ces méthodes basiques de chiffrement ne garantissent pas que le message soit lu uniquement par le destinataire légitime, mais elles ont probablement été utilisées pendant de nombreuses années. Il a fallu attendre l'époque de Jules César, vers 50 av. J.-C., pour voir apparaître de véritables systèmes cryptographiques. Le plus célèbre d'entre eux est le chiffre de César, qui consiste simplement à décaler les lettres d'un message de trois positions vers la droite dans l'alphabet latin. La lettre A est ainsi transformée en D, le B en E, etc. Cette nouvelle méthode de chiffrement est dite de *substitution*, car chaque lettre du message est remplacée par une autre (voir Figure 1.1). On peut bien sûr utiliser d'autres valeurs que trois pour le décalage : il y en a 25

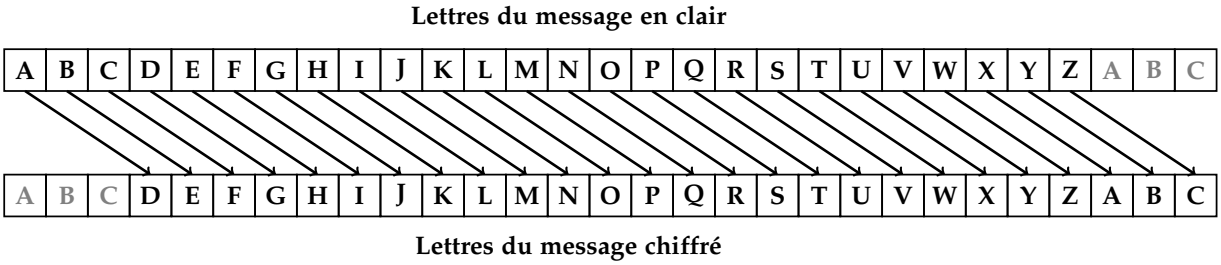


Figure 1.1: Chiffre de César : les lettres du message en clair sont décalées de trois positions dans l'alphabet.

pertinentes, étant donné que décaler les lettres de 26 positions ne modifie pas le message initial. D'autres manières similaires font leur apparition au fil des années, et on peut les classer dans deux grandes catégories : les substitutions monoalphabétiques ou polyalphabétiques. Le chiffre de César est une substitution monoalphabétique car tous les A du message seront toujours remplacés par D, mais on peut citer par exemple le chiffre de Vigenère (1586) dans lequel on ne se contente pas que d'un seul décalage comme pour César, mais de plusieurs. Ainsi, le A n'est pas forcément transformé en la même lettre : on parle de substitution polyalphabétique.

La cryptanalyse de ces schémas de chiffrement plus complexes repose principalement sur l'analyse des fréquences du texte chiffré. Il a fallu attendre le IXe siècle pour que le mathématicien arabe Al-Kindi s'intéresse à la répétition des lettres dans le message codé par le chiffre de César et s'aperçoive qu'elle correspond exactement à celle du message original. En effet, le chiffrement étant monoalphabétique, le nombre de E dans le message original est le même que le nombre de H dans le message chiffré. En français, le E étant la lettre la plus fréquente, il est facile de déduire quelle est la lettre correspondante dans le message chiffré, et ainsi de retrouver tout le message original. Le chiffre de Vigenère peut également être cassé en procédant à une analyse des fréquences comme l'a fait le mathématicien anglais Charles Babbage en 1854, dont les résultats ont été publiés en 1863 par le major Friedrich Kasiski. Le chiffre de Vigenère diffère de celui de César uniquement par le nombre (inconnu) de décalages : dès lors que ce nombre est connu, le même raisonnement que pour le chiffre de César peut s'appliquer. Kasiski montre dans ses travaux une manière efficace de récupérer ce nombre en analysant les répétitions de groupes de lettres dans le message chiffré. Plus récemment au XXe siècle, des techniques de statistiques utilisent l'*indice de coïncidence* pour retrouver la même valeur et également casser le code.

Le bond technologique suivant survient au début du XXe siècle pendant la Première et la Deuxième Guerres mondiales. Les besoins militaires des différentes armées de protéger leurs communications ont permis de voir l'apparition de machines spécialement conçues pour

le chiffrement et le déchiffrement. Elles utilisent les mêmes procédés connus de substitution, mais d'une manière plus complexe : on peut citer par exemple la machine Enigma, la C-36, la machine de Lorenz, la *Geheimfernschreiber*, etc. La machine allemande Enigma utilise notamment plusieurs rotors qui agissent indépendamment comme de simples chiffres de substitution, mais qui sont utilisés les uns à la suite des autres, et tournent comme le font les aiguilles d'une montre après que chaque lettre ait été traitée. Chaque camp développant ses propres mécanismes secrets de chiffrement, il était aussi naturel d'essayer d'attaquer les machines adverses. La machine Enigma a ainsi été décryptée par les britaniques, grâce à la Pologne en 1932 qui a partagé ses découvertes avec ses alliés peu de temps avant l'invasion du pays par les Allemands en 1939. Cette percée a permis aux Alliés de lire de nombreux messages allemands interceptés et aurait ainsi écourté la guerre de plusieurs mois.

Vers le milieu du XXe siècle, la cryptographie est devenue beaucoup plus mathématique et a été grandement facilitée par l'apparition des premiers ordinateurs. Cette cryptographie moderne est initiée par le travail de Claude Shannon en 1948 sur la *théorie mathématique de l'information*, sur laquelle repose la cryptographie moderne. En 1948, il montre également que même sur un canal véhiculant l'information de manière très altérée il est possible d'ajouter de la redondance afin que le message initial puisse être reconstruit après transmission. Enfin, en 1949, Shannon apporte la première preuve théorique de confidentialité, en lien avec la perfection du code de Vernam [Sha49], qui est une méthode théorique impossible à casser.

Aujourd'hui, la cryptographie fournit des méthodes de chiffrement afin de pallier d'une manière relativement sûre tous les défauts des méthodes exposées précédemment. La théorie mathématique donne de nombreux outils aux cryptographes pour développer des codes suffisamment robustes afin d'éviter que des cryptanalystes ne les cassent trop rapidement. En effet, même si l'on sait théoriquement comment créer des codes parfaitement sûrs, il est quasiment impossible de les utiliser efficacement en pratique. Des contraintes physiques, de temps ou d'argent imposent de recourir à des outils imparfaits qui bénéficient néanmoins d'une confiance théorique suffisante.

1.2 La cryptographie aujourd'hui

1.2.1 Généralités

À l'heure où les ordinateurs et les processeurs sont omniprésents, il devient facile d'incorporer les possibilités offertes par la cryptographie dans notre quotidien. Parmi ces capacités, on considère naturellement la *confidentialité* des données, mais dans beaucoup de cas, elle est insuffisante. L'*authentification* des personnes échangeant de l'information peut être bien plus importante, par exemple lorsque l'on considère un ordre de virement sur un compte bancaire, l'accès à un réseau internet ou téléphonique, l'accès à un coffre-fort, etc. Ces notions n'appellent pas nécessairement à une protection des données, mais plutôt à assurer à l'un des participants (ou à tous) qu'il dialogue avec la bonne personne et non avec un imposteur. Une troisième notion vient s'ajouter à la confidentialité et à l'authentification : l'*intégrité*. Les participants à une conversation veulent s'assurer que les messages qu'ils reçoivent sont bien ceux qui ont été envoyés par leurs interlocuteurs légitimes, et qu'ils n'ont pas été modifiés pendant la

transmission. La banque a par exemple besoin de vérifier que l'ordre de virement qu'elle reçoit d'une personne n'a pas été modifié par un attaquant pour en changer le compte créditeur.

La cryptographie moderne suit un certain nombre de principes fondamentaux, énoncés pour la première fois par Auguste Kerckhoffs en 1883 dans [Ker83]. Le plus important de tous décrit un système cryptographique comme un algorithme public, qui ne doit utiliser qu'une petite information secrète que l'on appelle la *clef*. Suivre ce principe a l'avantage de ne pas rendre inutilisable l'algorithme choisi si la clef est récupérée par l'ennemi. Avoir un algorithme secret ralenti théoriquement l'attaquant, mais ne doit pas être la seule source de secret du système. En effet, il est préférable d'utiliser un algorithme public car celui-ci peut être étudié et critiqué ouvertement, notamment par le monde académique.

De manière plus formelle, on considère généralement deux personnes (Alice et Bob) qui s'échangent des messages sur un canal non sécurisé (voir Figure 1.2). On suppose donc qu'un attaquant (Ève) est capable soit d'écouter le canal (attaquant passif), soit de le modifier (attaquant actif).

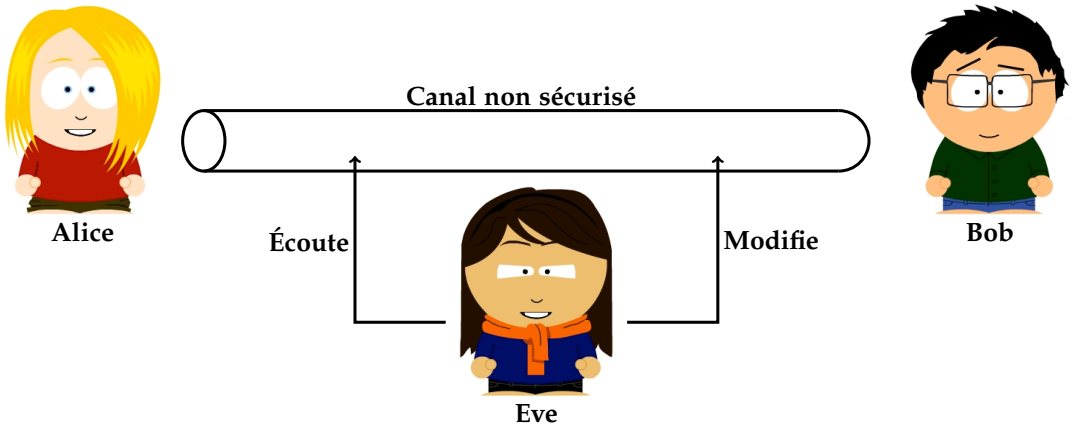


Figure 1.2: Canal de communication : Alice et Bob s'échangent des messages via le canal écouté par Ève.

Les notions précédentes se traduisent alors par :

- **La confidentialité.** Le message ne peut pas être lu par Ève.
- **L'authenticité.** Bob reçoit le message venant d'Alice et est convaincu qu'elle en est bien l'auteur. Si Ève envoie un message à Bob en se faisant passer pour Alice, il le détectera.
- **L'intégrité.** Si Ève modifie le message original envoyé par Alice, Bob le détectera.

Pour répondre à ces problèmes généraux, on distingue deux grandes familles d'outils cryptographiques : les systèmes à *clef secrète* et les systèmes à *clef publique*. Le premier système est le plus ancien et celui exposé jusqu'à présent dans ce manuscrit. Par exemple, la clef du chiffre de César est le décalage utilisé, et celle du chiffre de Vigenère est la connaissance de tous les décalages. Tous ces cryptosystèmes font l'hypothèse que les utilisateurs qui communiquent se sont mis d'accord au préalable sur une clef secrète à utiliser. Parfois, l'échange de cette clef ne peut être fait de manière sûre, et on se ramène au problème d'établir une communication sécurisée entre deux individus. Dans le deuxième cas, la cryptographie à base de clef publique permet de pallier ce paradoxe. Chaque utilisateur dispose de deux clefs : une clef publique qu'il n'est pas nécessaire de protéger au sens de la confidentialité et doit être transmise au

destinataire, et une clef privée qui doit être connue d'une et une seule personne. Alice et Bob peuvent alors échanger de manière sécurisée sans avoir à échanger de clef secrète au préalable : Alice utilise la clef publique de Bob pour chiffrer le message qu'elle souhaite lui envoyer, et à sa réception, Bob se sert de sa propre clef secrète pour pouvoir lire le message original. Si sa clef est restée secrète, il est en théorie le seul à pouvoir lire le message d'Alice.

En pratique, les deux systèmes sont utilisés conjointement : on se sert généralement d'un algorithme à clef publique pour échanger un secret entre les deux parties, puis ce secret sert de clef dans le deuxième algorithme à clef secrète qui est plus rapide à utiliser. Nous détaillons maintenant les primitives de ces deux grandes familles.

1.2.2 Cryptographie asymétrique

Par cryptographie asymétrique, on désigne la cryptographie à clef publique qui nécessite une paire de clefs par personne : une qui est publique et utilisée par tous, et une autre qui est privée, uniquement connue et utilisée par une personne. Le terme *asymétrique* est justifié par les processus de chiffrement et de déchiffrement qui utilisent des clefs différentes, ce qui n'est pas le cas pour la cryptographie symétrique. Pour construire ces algorithmes, des problèmes mathématiques réputés difficiles sont utilisés, ce qui permet de garantir que casser le mode d'utilisation de la primitive revient à résoudre ces problèmes difficiles. Ceci étant supposé impossible en temps raisonnable, on estime alors que les algorithmes sont sûrs. Les possibilités offertes par ces mécanismes sont nombreuses, mais on peut les classer principalement dans deux grandes catégories : les systèmes de chiffrement et les signatures électroniques.

1.2.2.1 Algorithme de chiffrement

Pour assurer la confidentialité du message qu'Alice envoie à Bob, Alice commence par récupérer la clef publique de Bob (dans un annuaire, un site Internet, ou en demandant directement à Bob par exemple) et l'utilise pour chiffrer le message qu'elle souhaite lui envoyer. Pour Alice, le message chiffré qu'elle envoie à Bob n'a plus aucune signification : Bob est en effet le seul à pouvoir le déchiffrer avec la clef privée. Le lien entre sa clef publique et privée permet de déchiffrer le message. À l'heure actuelle, les problèmes durs sur lesquels reposent les algorithmes de chiffrement asymétriques sont la difficulté de factoriser le produit de deux nombres premiers (système RSA [RSA78]), le logarithme discret (système ElGamal [ElG85], système de Schnorr [Sch89]) ou encore le problème du plus court vecteur dans un réseau (système NTRU [HPS98, SSNO12]).

1.2.2.2 Signature électronique

Les signatures électroniques assurent les mêmes fonctions que les signatures classiques sur papier : elles authentifient l'auteur d'un message ou d'un document. Pour signer un message qu'elle envoie à Bob, Alice utilise sa propre clef privée pour créer la signature, l'apposer sur le message puis elle envoie le tout à Bob. À la réception du message, Bob peut utiliser la clef publique d'Alice et s'assurer que le message qu'il a reçu a bien été signé par Alice, car elle est la seule à connaître la clef privée associée à la clef publique qu'il a utilisée pour la vérification. Les problèmes difficiles pour les algorithmes de signatures sont les mêmes que ceux pour les

algorithmes de chiffrement : le système RSA, ou encore DSA (*Digital Signature Algorithm*) qui utilise une variante de ElGamal.

Pour des raisons pratiques d'efficacité, il est préférable d'utiliser un algorithme symétrique pour le chiffrement. On se sert généralement d'un algorithme à clef publique pour effectuer l'échange du secret nécessaire entre les deux parties. L'échange de clefs le plus utilisé est Diffie-Hellman [DH76], du nom de leurs auteurs Whitfield Diffie et Martin Hellman, et est basé sur le problème difficile du logarithme discret. En terme de cryptanalyse, le principal problème des modes d'utilisation des algorithmes à clefs publiques est l'attaque par le milieu : si l'adversaire Ève arrive à remplacer la clef publique du destinataire légitime par la sienne, elle pourra faire croire aux deux individus honnêtes qu'ils communiquent entre eux, alors qu'elle agira comme relais et pourra lire tous les message échangés (voir Figure 1.3).

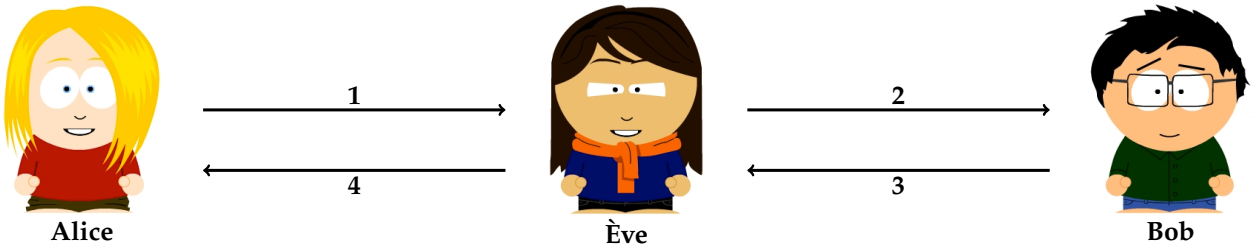


Figure 1.3: Attaque par le milieu : Ève agit comme intermédiaire entre Alice et Bob pour lire leurs communications.

1.2.3 Cryptographie symétrique

Par cryptographie symétrique, on fait référence à la cryptographie à clef secrète, les mécanismes de chiffrement et de déchiffrement utilisant la même clef pour les deux interlocuteurs Alice et Bob. Au sein de ce domaine, on distingue plusieurs familles de primitives qui permettent de répondre à différents besoins. Premièrement, on trouve les *algorithmes de chiffrement* qui permettent de transformer un message en clair en un message chiffré pour en assurer la confidentialité. On dispose ensuite de *fonctions de hachage*, qui sont utilisées dans beaucoup de domaines de la cryptographie et permettent par exemple de construire une empreinte d'un message pour en attester une certaine l'intégrité. Enfin, pour garantir l'authenticité de l'origine d'un message, on peut utiliser les *codes d'authentification de message* (MAC).

1.2.3.1 Algorithmes de chiffrement par bloc

Cette primitive de chiffrement est la plus répandue et consiste en un algorithme déterministe paramétré par une clef et qui travaille sur un groupe de bits de taille fixe appelé bloc. Pour chiffrer (ou déchiffrer) un message de longueur arbitraire, on utilise un *mode opératoire* qui permet de lier les résultats de chiffrements des blocs successifs et ainsi produire le chiffrement du message complet. En 1977, le *National Bureau of Standards* (désormais le *National Institute of Standards and Technology*, NIST) annonce le DES comme standard de chiffrement [DES77] : il s'agit d'un algorithme intialement développé par IBM qui utilise le réseau de Horst Feistel. Vingt ans plus tard, et après de nombreuses attaques publiées sur le DES, le NIST lance le concours

AES (*Advanced Encryption Standard*) qui sélectionne un nouveau standard de chiffrement en 2000 [AES01]. Complètement différent de son prédécesseur le DES, ce sont les cryptographes belges Joan Daemen et Vincent Rijmen qui remportent le concours avec leur soumission `Rijndael`, parmi les 15 autres algorithmes proposés [DR02].

1.2.3.2 Algorithmes de chiffrement par flot

Un schéma de chiffrement par flot utilise une clef secrète pour produire une séquence de caractères (pseudo-)aléatoires aussi longue que le message à chiffrer. Contrairement au chiffrement par bloc, cet algorithme n'a pas besoin de découper le message clair pour le chiffrer : on considère le clair en séquence pour opérer un OU exclusif (XOR) entre le caractère du message et le caractère de la séquence pseudo-aléatoire, ce qui donne le caractère du chiffré. Moins utilisés en pratique et moins étudiés, ils restent cependant très rapides à exécuter par leur construction. Faisant suite au concours AES, le projet eSTREAM s'est achevé en 2008 pour sélectionner de nouveaux algorithmes dans un portfolio. L'algorithme de chiffrement par flot le plus répandu actuellement est `RC4` par Ron Rivest, mais on peut également citer `Trivium`, `Grain`, `Salsa20`, ou encore `A5/1`, `A5/2`, etSNOW3G qui sont utilisés dans les communications mobiles.

1.2.3.3 Fonctions de hachage

Une fonction de hachage est une primitive cryptographique qui doit se comporter comme une fonction aléatoire. En particulier, elle doit être à sens unique, c'est-à-dire qu'il doit être *difficile* de l'inverser. Ces fonctions ne demandent aucun secret et fournissent une empreinte relativement courte d'un message, ce qui permet d'en donner une certaine preuve d'intégrité après transmission dans le canal non sécurisé (voir Figure 1.4). Bien qu'utilisées dans tous les domaines de la cryptographie, symétrique comme asymétrique, elles rentrent généralement dans la catégorie des primitives à clef secrète : elles utilisent des mécanismes similaires aux algorithmes de chiffrement, voire sont construites grâce à eux. C'est le cas pour beaucoup de fonctions de hachage comme `MD4`, `MD5`, `SHA-1`, `SHA-2`, etc. En 2008, le NIST a lancé une nouvelle compétition afin de définir un nouveau standard de hachage. La fonction `Keccak` de Guido Bertoni, Joan Daemen, Michaël Peeters, et Gilles Van Assche a remporté la compétition en octobre 2012, devenant ainsi le nouveau standard `SHA-3`.

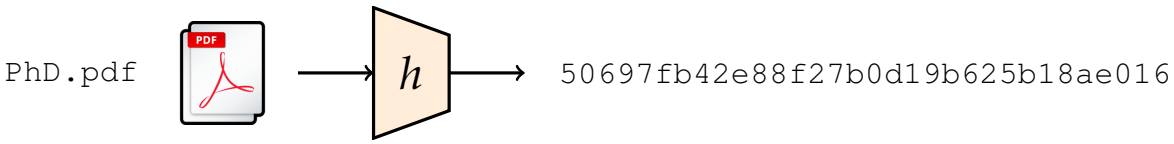


Figure 1.4: Fonction de hachage : la fonction h fournit une empreinte courte d'un document.

1.2.3.4 Code d'authentification de message (MAC)

Un MAC consiste en une petite information ajoutée à un message en clair qui permet d'en vérifier à la fois l'intégrité et la provenance. Un algorithme MAC s'utilise avec une clef secrète partagée entre Alice et Bob. Alice se sert de la clef pour créer le MAC et l'envoie avec le message à Bob. Celui-ci le recalcule avec sa copie de la clef et vérifie que le message n'a ni été modifié,

ni créé par un imposteur ne connaissant pas la clef secrète. Les algorithmes MAC ressemblent fortement aux fonctions de hachage mais demandent des notions de sécurité relativement différentes : il est possible de construire un algorithme MAC à partir d'une fonction de hachage (c'est le cas de HMAC) ou d'un algorithme de chiffrement par bloc (comme OMAC ou CBC-MAC).

Dans cette thèse, je me suis intéressé aux fonctions de hachage et aux algorithmes symétriques de chiffrement par bloc. Avant de les décrire plus en détails dans les sections suivantes, nous détaillons quelques notions de cryptanalyse.

1.2.4 Notions de cryptanalyse

Le rôle du cryptanalyste n'a pas changé suivant les évolutions de la cryptographie, mais les moyens qu'il a à sa disposition sont plus performants. En outre, les schémas auxquels il s'attaque sont complètement différents des simples algorithmes de substitutions utilisés dans la première ère de la cryptographie. Dans le cadre de la simple substitution du chiffre de César par exemple, dès lors que le cryptanalyste intercepte un message chiffré, il n'a plus qu'à essayer les 25 décalages possibles pour retrouver le message original en clair. Cette méthode de *force brute* consiste à essayer tous les scénarios possibles selon le mécanisme de chiffrement jusqu'à obtenir le résultat ; dans ce cas, le message en clair connu. Une méthode plus raffinée consiste à analyser la répartition des fréquences des lettres pour obtenir directement le bon décalage, et aboutir au même résultat. Cette méthode est plus rapide dans le sens où l'on n'essaye *que* le bon décalage, et non les 26 possibles. Avec les notions de clefs présentées précédemment pour les schémas de chiffrement, on peut constater qu'une manière naïve de récupérer le message original est d'essayer toutes les clefs possibles sur un message intercepté pour retrouver le message en clair : cette méthode porte le nom de *recherche exhaustive* ou force brute et s'applique à tous les schémas de chiffrement.

Du point de vue de l'attaquant, on distingue quatre scénarios d'*attaque* différents suivant l'information que cet attaquant est capable de récupérer. Dans tous les cas, le but du cryptanalyste est de retrouver la clef utilisée pour le chiffrement des messages qu'il aura intercepté. Par attaque, on entend une tentative de cryptanalyse d'un schéma donné qui utilise de l'information dans l'un des cas suivants :

- **Chiffré seul (*ciphertext only*).** L'attaquant a en sa possession uniquement un ou plusieurs messages chiffrés, mais ne détient aucune information sur les messages en clair correspondants. En pratique, ce scénario est le plus courant.
- **Clair connu (*known plaintext*).** Dans ce cas, le cryptanalyste a non seulement accès aux messages chiffrés, mais également aux messages en clair correspondants. On parle de paires clairs/chiffrés.
- **Clair choisi (*chosen plaintext*).** En donnant plus de puissance à l'attaquant, on lui permet de chiffrer les messages qu'il souhaite avec une clef qui lui est inconnue, et son but est d'en déterminer la valeur. Bien que ce scénario avantage grandement l'attaquant, on le retrouve dans diverses implémentations pratiques telles que dans les cartes à puce. La clef secrète est protégée physiquement, mais on peut demander le chiffrement de messages par la carte et en récupérer les chiffrés.
- **Chiffré choisi (*chosen ciphertext*).** De la même manière, l'attaquant peut dans ce cas demander le déchiffrement de messages quelconques et obtenir leur déchiffrement par la

clef secrète.

Ces différents modèles d'attaquant permettent de créer des classes d'attaques distinctes, qui doivent être envisagées lors du déploiement des cryptosystèmes. Dans tous les cas, en suivant les principes de Kerckhoffs, l'attaquant est toujours supposé connaître le mécanisme de chiffrement.

Lors de la conception des algorithmes de chiffrement, les cryptographes estiment que la meilleure attaque dont dispose le cryptanalyste est la recherche exhaustive de la clef. Si tel est le cas, alors le schéma est sûr, sinon il est considéré comme cassé. Ainsi, la recherche exhaustive de la clef dans le cas du chiffre de César n'est pas une attaque puisque toutes les clefs sont essayées. Par contre, l'analyse fréquentielle constitue la meilleure attaque connue puisqu'une seule clef est essayée. Cette différence entre les systèmes primitifs, comme les chiffres de César ou de Vigenère, et les méthodes modernes donne une toute autre vision de la cryptanalyse. En effet, dans les anciens systèmes, le nombre de clefs possibles est relativement petit en comparaison à la puissance de calcul offerte par les ordinateurs. Cette avancée technologique permet d'effectuer la recherche exhaustive de manière beaucoup plus rapide, et impose donc aux cryptographes de concevoir des algorithmes sûrs dont le nombre de clefs ne pourra pas être atteint par l'ordinateur.

L'ancien standard de chiffrement par bloc DES considère des clefs de 56 bits, ce qui signifie qu'au plus 2^{56} clefs existent. Ainsi, la recherche exhaustive demande une puissance de calcul équivalente à $2^{56} = 72\,057\,594\,037\,927\,936$ chiffrements DES. En termes de comparaison, on estime qu'à l'heure actuelle une machine ordinaire d'un particulier effectue 2^{32} calculs élementaires en quelques secondes, et que les agences gouvernementales les plus puissantes atteignent largement 2^{56} calculs, mais pas 2^{80}, soit 80 bits de sécurité. Avec l'AES comme nouveau standard de chiffrement, ainsi que la majorité des algorithmes de chiffrement publiés depuis, on dispose au minimum de 128 bits de sécurité, ce qui donne une certaine marge de confiance vis-à-vis de la recherche exhaustive.

1.3 Fonctions de hachage

On appelle fonction de hachage h une fonction mathématique qui associe un message de longueur arbitraire à une sortie de taille fixe notée n :

$$\begin{array}{rcl} h : \{0,1\}^* & \longrightarrow & \{0,1\}^n \\ m & \longrightarrow & h(m). \end{array}$$

Le paramètre d'entrée de la fonction, représenté comme une séquence de bits, est appelé *message*. L'image de ce message par h est appelé *haché* (ou *empreinte*). Ce dernier étant de petite taille, il permet de représenter le message associé d'une manière compacte et donc de l'identifier rapidement.

Bien que les fonctions de hachage cryptographique soient présentées dans la prochaine section comme un moyen d'obtenir des notions de sécurité informatique, leurs usages dépassent le domaine de la cryptographie : on peut citer par exemple le cas de l'algorithmie. La majorité des utilisations des fonctions de hachage repose sur une représentation courte et de taille fixe de n'importe quel message. La structure de données appelée *table de hachage* associe une clef à

un élément de nature quelconque. C'est le cas d'un annuaire téléphonique où les clefs seraient les initiales des personnes, et les éléments seraient les prénoms et les numéros de téléphone associés (voir Figure 1.5). Dans une telle structure, les clefs sont les hachés des éléments à stocker, ou d'une partie des valeurs à enregistrer, comme les initiales des personnes dans l'annuaire. Une table de hachage se présente le plus souvent comme un tableau, et permet d'effectuer des recherches en temps quasi constant, d'où son utilité.

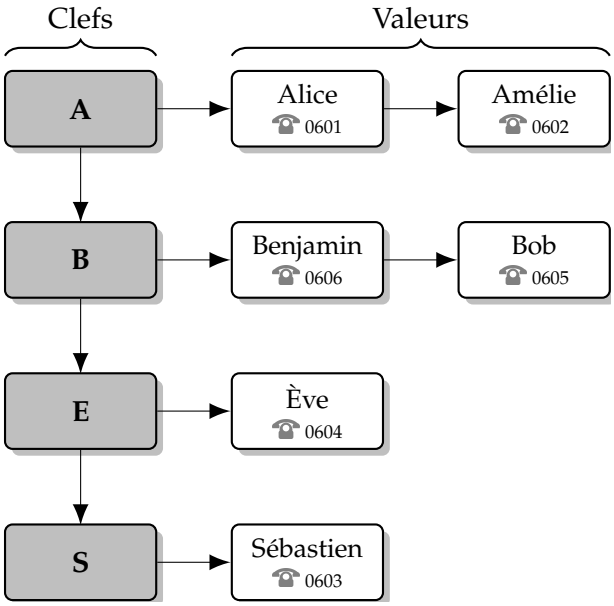


Figure 1.5: Table de hachage : un annuaire stocke les paires clé/élément comme nom/téléphone.

Selon les utilisations, la complexité de la fonction de hachage peut varier. On peut par exemple vouloir une fonction qui *répartit* de manière uniforme les sorties pour des entrées tirées de manière aléatoire. En effet, pour reprendre l'exemple de l'annuaire téléphonique, si plusieurs personnes partagent le même initiale, il faut alors stocker les éléments (prénoms et numéros) au même endroit dans la structure. Ceci pourrait être évité avec une fonction plus complexe, et garantirait que les données soient bien équilibrées au sein de la table. Ainsi, savoir construire une fonction de hachage vérifiant certaines propriétés est essentiel.

1.3.1 Fonctions de hachage cryptographique

Les fonctions de hachage cryptographiques ont beaucoup d'applications en sécurité de l'information, et en particulier dans le cas des signatures électroniques, des MACs, et d'autres formes d'authentification. Dès lors qu'elles interviennent dans des mécanismes ou protocoles de cryptographie, il est essentiel que ces fonctions vérifient des propriétés plus fortes que les fonctions de hachage classiques. En particulier, on attend d'une fonction de hachage cryptographique qu'elle soit difficile à inverser et que trouver deux messages ayant le même haché le soit également. Pour une fonction de hachage normale, utilisée par exemple dans une table de hachage, ces deux propriétés ne sont pas primordiales. Dans la suite, une fonction de hachage mentionnée sans précision supplémentaire sera supposée cryptographique.

Plus en détail, les trois propriétés fondamentales qu'une fonction de hachage h doit vérifier sont les suivantes :

- **Résistance en préimage.** Cette propriété traduit le caractère à sens unique de la fonction, c'est-à-dire que h doit être difficile à inverser. Formellement, étant donnée une image y, le problème est de trouver un message x tel que $h(x) = y$.
- **Résistance en collision.** Le problème associé consiste à trouver deux message x et x' tels que les images par h soient égales : $h(x) = h(x')$. Si ce problème est dur pour h, alors on dit que h résiste aux collisions.
- **Résistance en seconde-préimage.** Étant donné un message x, il doit être difficile de trouver un autre message x' différent de x tel que les deux images par h soient égales, i.e. $h(x) = h(x')$.

Dans le cas particulier des fonctions de hachage, la *difficulté* des problèmes présentés précédemment est liée à l'ensemble de sortie de la fonction de hachage h, i.e. au nombre de valeurs possibles que peut prendre cette fonction. Si la fonction de hachage h est supposée parfaite et sans meilleure attaque que la recherche exhaustive, alors elle s'approche suffisamment bien d'une fonction aléatoire pour qu'une image y donnée soit atteinte pour un message aléatoire avec probabilité 2^{-n}. On s'attend ainsi à devoir essayer 2^n messages x différents avant d'en trouver un tel que $h(x) = y$ (voir Table 1.1). Il en est de même pour la seconde préimage.

En terme de réduction, on peut montrer que la résistance en collision implique la résistance en seconde préimage. En effet, on suppose que h est résistante aux collisions et que x est un message quelconque d'image $y = h(x)$. Si h n'est pas résistante en seconde préimage, alors on peut trouver un deuxième message x' différent de x tel que $h(x) = h(x')$, ce qui contredit la résistance en collision.

Dans la section suivante, nous détaillons le *paradoxe des anniversaires* qui explique la complexité $2^{n/2}$ dans le cas de l'attaque en collision.

Résistance	Entrée	Sortie	Contrainte	Complexité
Préimage	y	x	$h(x) = y$	2^n
Collision	-	(x, x')	$h(x) = h(x')$	$2^{n/2}$
Seconde-préimage	x	x'	$h(x) = h(x')$	2^n

Table 1.1: Complexité des attaques génériques sur une fonction de hachage $h : \{0,1\}^* \to \{0,1\}^n$.

1.3.2 Paradoxe des anniversaires

Etant donnée une fonction de hachage $h : \{0,1\}^* \to \{0,1\}^n$, le principe des tiroirs de Dirichlet (en anglais, *pigeonhole principle*) permet d'affirmer qu'il existe des collisions dans h (voir Figure 1.6). Formellement, cela signifie qu'il est impossible de trouver une application $h : \{0,1\}^* \to \{0,1\}^n$ qui soit injective, et donc il existe deux messages différents x et x' tels que leur image par h soient égales, i.e. $h(x) = h(x')$.

Le problème des anniversaires consiste à choisir k éléments *avec* remise dans un ensemble qui en contient n, et à mesurer à partir de quand un élément déjà pioché l'est à nouveau. Le problème a initialement été présenté en 1939 par le mathématicien Richard von Mises qui se

demandait combien de personnes il fallait réunir pour que la probabilité que deux d'entre eux partagent le même jour d'anniversaire soit d'au moins 0.5. Avec une fonction de hachage h, cela revient à trouver quel est le nombre minimal de messages à considérer afin qu'au moins deux de leurs images soient égales, avec une probabilité plus grande que p. De manière théorique, pour une fonction dont le nombre d'images possibles est N, on peut prouver qu'il faut en moyenne $\sqrt{\pi N/2}$ messages avant que les images ne collisionnent (voir par exemple [vOW99, Appendix A]). Ainsi, une fonction de hachage avec des empreintes de n bits offre une sécurité $2^{n/2}$ en résistance aux collisions.

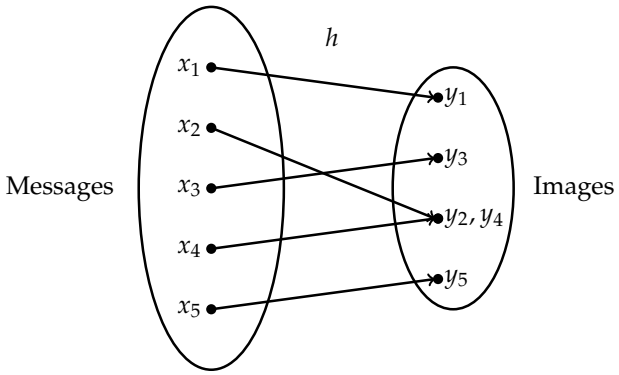


Figure 1.6: Vue schématique d'une fonction de hachage h : l'ensemble des images étant plus petit que l'ensemble des messages, il y a nécessairement des collisions dans h : e.g. $h(x_2) = h(x_4)$.

Pour trouver une collision dans $h : \{0,1\}^* \to \{0,1\}^n$, l'attaque générique la plus efficace repose sur le paradoxe des anniversaires (*birthday attack*). Par attaque *générique*, on entend une attaque qui s'applique de la même manière quelle que soit la fonction que l'on se donne. Cet algorithme utilise simplement le paradoxe des anniversaires exposé précédemment en retenant dans une liste l'ensemble des messages x tirés aléatoirement et leurs images jusqu'à que deux d'entre elles soient égales. Il faut alors attendre d'avoir environ $2^{n/2}$ éléments dans la liste avant d'obtenir la collision, ce qui donne une attaque en $2^{n/2}$ en temps et en mémoire. Cette attaque a été décrite indépendamment par Merkel et Yuval [Yuv79, Mer90] et permet par exemple de montrer une attaque sur un schéma de signatures électroniques utilisant une fonction de hachage.

L'algorithme précédent peut être amélioré pour diminuer la complexité mémoire de $2^{n/2}$ à une taille constante et négligeable. On peut par exemple utiliser l'algorithme Rho de Pollard [Pol75] pour ne stocker qu'une nombre fini et maîtrisé de points particuliers. De manière plus générale, on peut ramener le problème de trouver la collision à une détection de cycle en introduisant une application pour simuler une marche aléatoire dans l'ensemble des images de h. On peut par exemple citer les algorithmes de Floyd [Flo67, Knu97] qui mime la progression du lièvre et de la tortue, ou bien l'algorithme de Brent [Bre80] qui améliore légèrement celui de Floyd. Pour réduire la complexité en temps, il est possible de paralléliser la recherche de collision sur plusieurs processeurs : Van Oorschot et Wiener proposent une méthode pour atteindre un gain linéaire en temps dans [vOW99].

1.3.3 Modes opératoires

Dans la pratique, une fonction avec un domaine de définition infini comme une fonction de hachage est difficile à manipuler. Plutôt que de considérer le message en entier, on préfère ainsi le morceler en plusieurs blocs de taille fixe de $\mathcal{K}$ bits, et traiter ces blocs par une fonction plus simple. Cette idée a initialement été publiée par Michael O. Rabin en 1978 [Rab78]. La *fonction de compression* $f : \{0,1\}^{\mathcal{K}} \times \{0,1\}^n \to \{0,1\}^n$ qui traite les blocs devient donc la brique de base : elle prend en entrée le résultat du bloc précédent et le bloc courant (voir Figure 1.7).

Une fonction de hachage est ainsi séparée en deux entités, la première étant la fonction de compression f, et la deuxième le *mode opératoire* ou algorithme d'*extension de domaine*, qui décrit la manière de lier les résultats intermédiaires de la transformation interne. Dans les paragraphes suivants, nous décrivons des constructions concrètes, l'un des objectifs étant de conserver les propriétés cryptographiques de la fonction de compression f.

1.3.3.1 Construction de Merkle-Damgård

La construction de Rabin qui découpe le message en blocs de tailles fixes a été prouvée par Merkle et Damgård : si le message est morcelé correctement et traité par une fonction de compression résistante aux collisions, alors la fonction de hachage l'est également [Mer90, Dam90]. Afin de s'assurer que la longueur du message m soit un multiple de la taille du bloc,

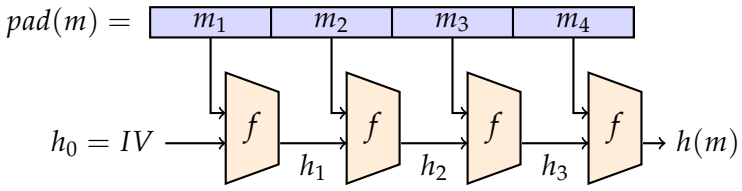


Figure 1.7: Construction de Merkle-Damgård : le message m avec le padding $pad(m)$ est morcelé en quatre blocs, et la fonction de compression f chaine les résultats intermédiaires pour produire $h(m)$.

on ajoute le bit 1 à la fin du message, suivi d'autant de bits à 0 que nécessaire pour atteindre le nouveau message $pad(m)$ qui a une longueur du multiple voulu (*padding*). De plus, on renforce généralement cette complétion par l'ajout d'un bloc supplémentaire au message pour représenter la longueur réelle du message traité (*Merkle-Damgård strengthening*). Cela permet, par exemple, d'éviter des attaques en collision avec des messages de tailles différentes, des attaques avec des messages très longs, ou encore la recherche de points fixes dans la fonction de compression f.

1.3.3.2 Attaque par extension (*extension attack*)

Bien que cette construction soit utilisée dans les fonctions de hachage `MD5`, `SHA-1`, `SHA-2`, on remarque que le chaînage des blocs permet de déduire le haché du message $pad(x)||y$ uniquement avec $h(x)$, sans la connaissance de x. De la même manière, on peut calculer le haché d'un message m' uniquement avec la connaissance du haché $h(k||m)$ d'un message m connu et d'une clef k inconnue de longueur connue. Dans ce cas, la construction de Merkle-Damgård permet d'obtenir la valeur de chaînage $\alpha = pad(k||m)$ après le traitement du message. Même sans connaître la clef k, il est donc possible de calculer le haché $h(m')$ pour $m' = pad(k||m)||x$.

Cette utilisation d'une fonction de hachage avec une clef secrète intervient dans le calcul d'un MAC, et une attaque d'extension comme celle-ci permet de signer un message particulier sans connaître la clef. Ce problème peut être contourné en utilisant $h(k||h(k||m))$ plutôt que $h(k||m)$ ou $h(m||k)$. C'est le cas de HMAC qui fait deux évaluations de h pour protéger la clef k par l'attaque d'extension.

1.3.3.3 Attaque par multicollisions

En terme de sécurité, une fonction de hachage h sur n bits doit être résistante aux simples collisions, i.e. trouver (x, x') tels que $x \neq x'$ et $h(x) = h(x')$. De manière plus forte, une attaque en multicollisions consiste à trouver un ensemble $X = \{x_i\}$ de messages deux à deux distincts tels que toutes les images $h(x_i)$ soient égales. Obtenir un ensemble X de cardinal t vérifiant ces propriétés pour une fonction aléatoire requiert l'évaluation de la fonction $2^{n \cdot t/(t-1)}$ fois. En 2004, Antoine Joux montre dans [Jou04] que la construction d'un tel ensemble pour une fonction de hachage basée sur Merkle-Damgård demande en réalité beaucoup moins d'efforts. Il remarque que si l'on obtient t collisions dans la fonction de compression f, alors il est possible de construire 2^t collisions dans la fonction de hachage. Avec les notations de la Figure 1.8,

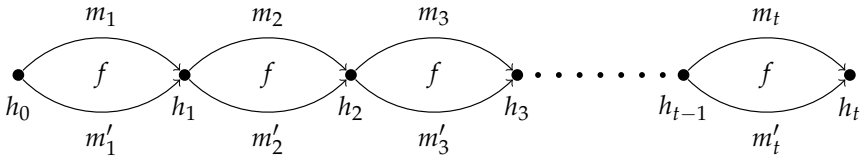


Figure 1.8: Construction de multicollisions dans une fonction de hachage basée sur Merkle-Damgård.

si l'attaquant connaît t collisions dans f, c'est-à-dire des blocs de messages $m_0, \ldots, m_t$ tels que $f(m_i, h_i) = f(m'_i, h_i) = h_{i+1}$, pour $0 \leq i < t$, alors il est possible de construire 2^t messages de t blocs en considérant tous les chemins possibles du graphe entre h_0 et h_t. Tous ces messages partagent alors le même haché par la fonction de hachage h construite sur le schéma de Merkle-Damgård. Ce résultat surprenant se réduit à la recherche de t collisions dans f, et demande donc uniquement $t \cdot 2^{n/2}$ évaluations de f.

1.3.3.4 Construction wide-pipe

Pour pallier les deux principaux points faibles de la construction de Merkle-Damgård, Stefan Lucks a proposé la construction *wide pipe* [Luc04]. Des attaques précédents, on constate que la sortie de la fonction de hachage est aussi grande que la taille de l'état interne, à savoir n bits. Avec la construction *wide-pipe*, on double la taille de l'état interne, pour que la sortie de la fonction en soit moitié moins grande : si on veut n bits de sortie, l'état interne est sur $2n$ bits. En contrepartie, on est contraint de rajouter une application finale pour compresser la dernière valeur de chaînage de $2n$ bits aux n bits voulus.

En termes de sécurité, la réduction classique de la fonction de hachage vers la fonction de compression reste identique, mais la complexité des attaques est désormais beaucoup élevée. Pour obtenir une collision dans la fonction de compression, il faut désormais évaluer $2^{2n/2} = 2^n$ messages, et non plus $2^{n/2}$. Pour la résistance en préimage, la complexité reste la même. Doubler la taille de l'état interne de la fonction de hachage a cependant un inconvénient : la fonction

de compression doit manipuler plus de données, et peut difficilement être plus efficace. En revanche, Nandi et Paul ont montré qu'il est possible d'adapter la construction [NP10a] pour atteindre de meilleures performances.

1.3.3.5 Construction sponge

Un autre type de construction qui a été largement popularisé par l'adoption de Keccak comme nouveau standard de hachage est la construction éponge, ou *sponge construction* (voir Figure 1.9).

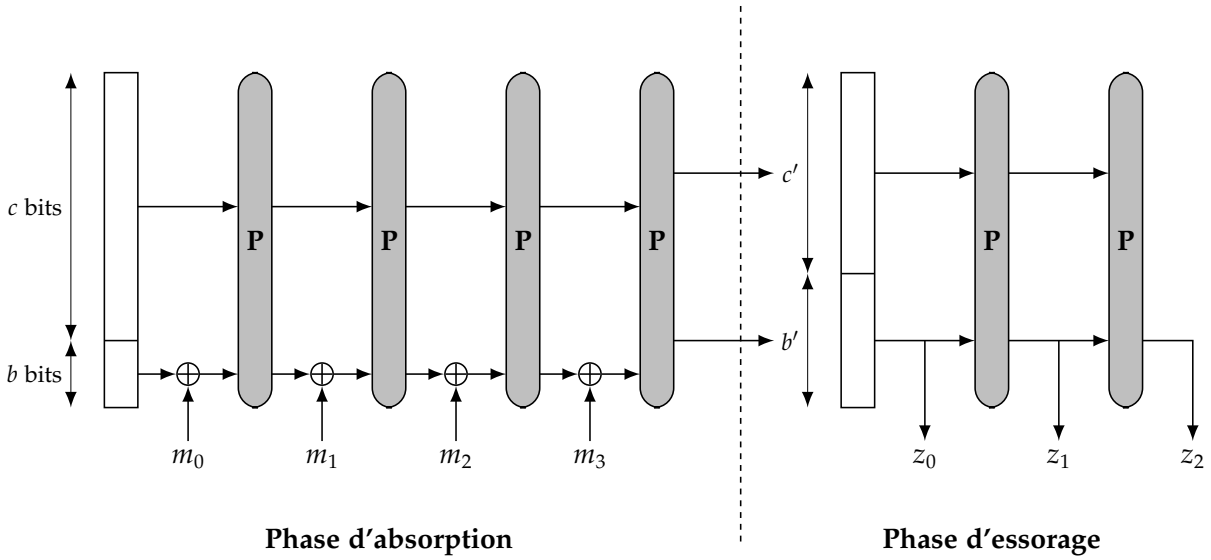


Figure 1.9: La construction éponge (*sponge*). Les paramètres (b, c) déterminent la phase d'absorption, et les paramètres (b', c') la phase d'essorage.

Ce type d'algorithme peut avoir a des utilisations très diverses, étant donné qu'il peut prendre n'importe quelle taille de message en entrée et qu'il peut produire autant de bits de sortie que nécessaire. On peut, par exemple, citer des applications à tous les niveaux de la cryptographie, que ce soit pour des fonctions de hachage, des MACs, des algorithmes de chiffrement par blocs ou par flots. On doit principalement ce mode opératoire à Joan Daemen qui l'avait déjà introduit dans plusieurs primitives, comme la fonction de hachage cryptographique RADIOGATÚN [BDAP06], qui elle-même améliorait la fonction de hachage PANAMA [DC98].

Contrairement aux fonctions de compression qui servent de briques de base aux fonctions de hachage présentées jusqu'à présent, la construction *sponge* utilise une permutation, sans procéder à une compression. Nous l'avons notée P dans la Figure 1.9. L'utilisation de cette construction se déroule en deux temps : d'abord une phase d'*absorption*, pendant laquelle le message est progressivement intégré à l'état interne, puis une phase d'*essorage*, où l'on génère autant de bits de sortie que nécessaire. Avant de procéder à la première étape, le message m à intégrer est ajusté à une longueur multiple de la taille de bloc (*padding*) pour former $\text{pad}(m)$, puis il est découpé en t blocs de mêmes tailles $m_1 || \cdots || m_t$. Chacun de ces m_i est intégré à l'état interne par un XOR, puis un appel à P remplace l'état interne actuel s par $P(s)$. Cette phase d'absorption se répète tant qu'il reste des blocs m_i à traiter.

On commence ensuite à construire la sortie de la primitive en prenant autant de blocs de l'état interne que nécessaire et en les séparant par de nouvelles applications de la permutation P.

Les paramètres utilisés dans la construction *sponge* dépendent du niveau de sécurité souhaité pour la primitive. On peut par exemple montrer [BDPV08] que la résistance en collision d'une fonction éponge produisant n bits de sortie et paramétrée par c bits de capacité est $\min(2^{n/2}, 2^{c/2})$ évaluations de la fonction. L'utilisateur peut librement ajuster le niveau de sécurité en choisissant une valeur de c appropriée, tout en gardant un bon compromis sécurité/efficacité. En effet, plus la valeur de c est grande, plus le niveau de sécurité sera important, mais moins la fonction globale sera performante. En revanche, lorsque une forte sécurité n'est pas la principale fonctionnalité demandée pour une primitive, comme toutes les fonctions à bas coût (*lightweight*), on peut obtenir de très bonnes performances avec cette construction *sponge*. On peut par exemple citer QUARK [AHMNP10], PHOTON [GPP11] et SPONGENT [BKL+11].

1.4 Algorithmes de chiffrement par bloc

1.4.1 Définition

Un algorithme de chiffrement par bloc constitue la principale méthode de chiffrement offerte par la cryptographie. Il se classe dans la catégorie symétrique. Cependant, il existe une deuxième méthode : le chiffrement par flot. La différence entre ces deux méthodes repose principalement sur le fait de découper ou non le message d'entrée. Dans le cas d'un algorithme de chiffrement par flot, le message n'est pas découpé, mais on lui applique, bit par bit (ou octet par octet), un masque similaire au code de Vernam (*one-time pad*). Cette séquence de masques représente les bits (ou octets) de sortie de l'algorithme de chiffrement par flot, et doit être la plus aléatoire possible.

Dans le cas d'un algorithme de chiffrement par bloc, le message d'entrée est découpé en blocs de taille fixe de n bits (par exemple $n = 128$ bits ou 16 octets), qui sont tous traités simultanément par l'algorithme. Un tel algorithme est déterministe et applique une permutation fixée par le biais d'une clef K de k bits. En effet, un algorithme de chiffrement par bloc définit une famille de permutations indépendantes paramétrées par une clef :

$$\begin{aligned} \mathcal{E} : \{0,1\}^k \times \{0,1\}^n &\longrightarrow \{0,1\}^n \\ (K, m) &\longrightarrow E(K, m). \end{aligned}$$

Pour échanger un message de manière secrète, Alice et Bob se mettent d'accord sur une clef secrète K à utiliser, ce qui leur fixe une permutation $\mathcal{E}_K$ dans la famille $\mathcal{E}$. Ils peuvent l'utiliser simplement pour chiffrer tous leurs blocs de messages m en $\mathcal{E}_k(m)$. Si l'adversaire Eve parvient à intercepter un ou plusieurs blocs de messages chiffrés, elle ne peut pas retrouver les messages en clair étant donné qu'elle ne connaît pas la permutation utilisée, c'est-à-dire la clef. Cet adversaire ne peut alors utiliser que la technique de la force brute pour essayer toutes les 2^k permutations possibles jusqu'à ce que le bloc de message déchiffré ait un sens.

1.4.2 Construction itérée

En théorie, il existe $2^n!$ permutations sur n bits, et donc $(2^n!)^{2^k}$ familles de permutations avec des clefs de k bits. Un algorithme de chiffrement est dit idéal si les $2^n!$ permutations possibles sont équiprobables. Or, dans la pratique, il est impossible de satisfaire cette propriété. On préfère donc utiliser des algorithmes non idéaux, mais que l'on sait représenter d'une manière compacte, et donc calculer de manière efficace.

Afin de construire des algorithmes de chiffrement par bloc, on privilégie l'utilisation de plusieurs applications successives d'une fonction cryptographiquement peu sûre, mais dont la répétition apporte la sécurité. On doit cette idée à Shannon [Sha49], qui en reprenant les concepts historiques de substitution et de transposition introduit les notions générales de *confusion* et de *diffusion*. La confusion, qui est liée aux substitutions des schémas plus anciens, tend à dissimuler le plus possible les liens existant entre la clef, le message en clair et le message chiffré. La diffusion correspond à la forte dépendance qui doit être présente entre les différents états intermédiaires du calcul. L'intuition de base serait, par exemple, que tous les caractères du message chiffrés dépendent de *tous* les caractères du message d'entrée. Ainsi, même si deux messages en clair sont très proches et ne diffèrent par exemple que d'un seul caractère, alors les deux messages chiffrés n'auraient aucun point commun. On parle également d'effet d'avalanche lorsqu'une différence d'un seul bit entre deux messages entraine d'importantes différences dans les messages chiffrés associés.

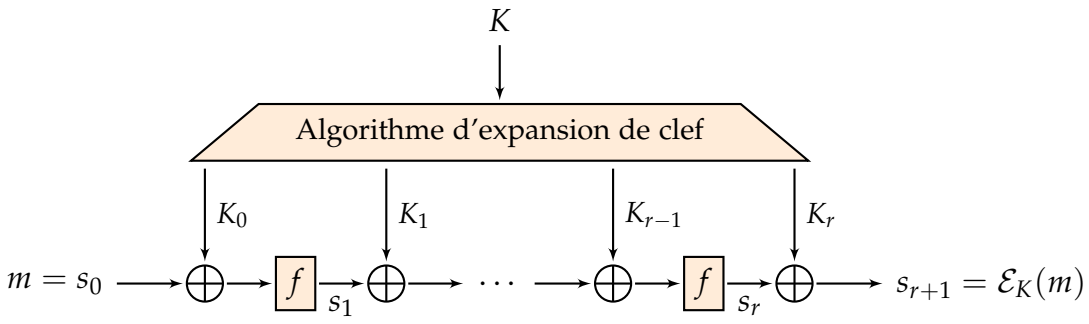


Figure 1.10: Algorithme de chiffrement par bloc itéré à clefs alernantes : la fonction f est appliquée r fois pour apporter la confusion et la diffusion dans l'algorithme. Les sous-clefs sont introduites entre chaque application de la fonction f.

En pratique, pour construire un algorithme de chiffrement par bloc à la fois compact et efficace, on utilise la construction itérée, représentée sur la Figure 1.10. Un bloc de message m initialise la variable s_0 et on applique r fois une fonction f plus simple pour produire le bloc de message chiffré $\mathcal{E}_K(m)$. La clef secrète K mentionnée précédemment, qui sélectionne une permutation $\mathcal{E}_K$ parmi la famille des 2^k permutations $\mathcal{E}$, est d'abord étendue à $r+1$ sous-clefs $K_0, \ldots, K_r$, qui sont ensuite intégrées au sein du calcul. Dans le cas général, chaque sous-clef k_i définit une fonction f_{k_i} paramétrée par k_i. Dans le cas particulier de la Figure 1.10, nous présentons le cas du chiffrement par bloc où les sous-clefs sont insérées par l'opération XOR entre chaque application de la même fonction f. La grande majorité des algorithmes de chiffrement par bloc utilisés actuellement suit ce principe de conception, comme par exemple le DES ou l'AES.

1.4.3 Modes opératoires

Tout comme pour les fonctions de hachage, construire une famille de permutations pour chiffrer des blocs de message est nécessaire, mais il est également primordial de savoir gérer des messages de plus d'un bloc. Il faut alors *lier* les blocs de message en clair et chiffrés les uns avec les autres dans un mode opératoire.

Dans un paragraphe précédent, nous proposions de simplement utiliser une permutation secrètement choisie $\mathcal{E}_K$ par Alice et Bob via une clef K pour qu'ils remplacement chacun des blocs de messages m par leur image $\mathcal{E}_K(m)$ (voir Figure 1.11). Ce mode opératoire porte le

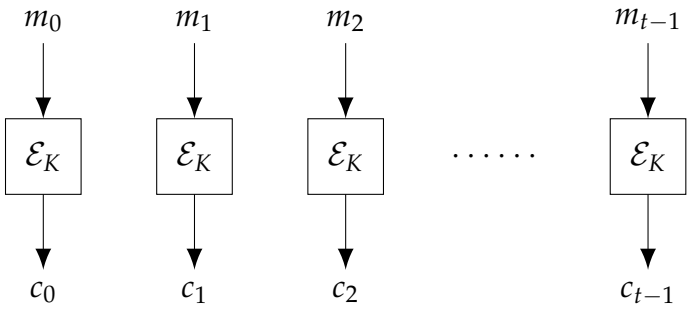
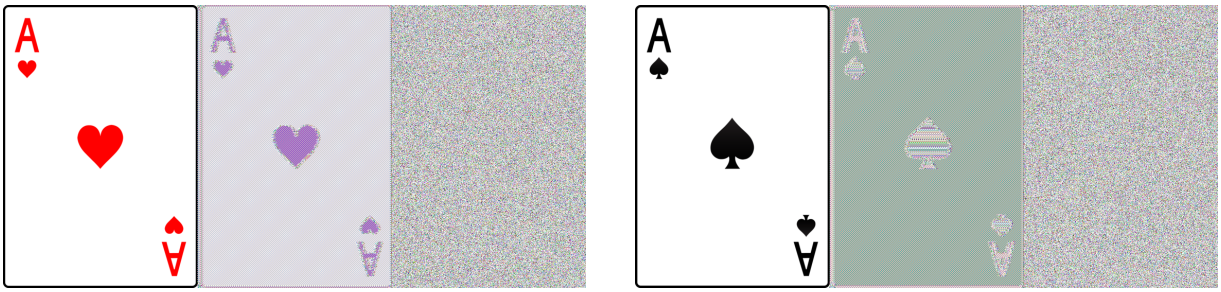


Figure 1.11: Le mode ECB (*Electronic Codebook*). Les t blocs de messages $m_0, \ldots, m_{t-1}$ sont traités indépendamment par la permutation $\mathcal{E}_K$.

nom de *Electronic Codebook* (ECB) et ne devrait jamais être utilisé sur des applications sensibles. En effet, faire une simple substitution de m par $\mathcal{E}_K(m)$ a la forte propriété que des blocs de messages m_1 et m_2 égaux auront le même bloc de message chiffré. En chiffrés seuls, dès lors qu'une forte structure existe dans les messages en clair, plus ils sont longs, plus la chance que ce phénomène arrive sera grande. Ainsi, un attaquant pourra voir des répétitions dans les blocs de messages chiffrés, et ainsi récupérer de l'information, ce qui voulait être évité. L'exemple le plus connu et le plus flagrant est le chiffrement d'une image par ECB. Prenons, par exemple, des images représentant des cartes à jouer, et chiffrons-les avec AES-128 en mode ECB. Puisque



(a) Clair. **(b)** AES-ECB. **(c)** AES-CBC. **(d)** Clair. **(e)** AES-ECB. **(f)** AES-CBC.

Figure 1.12: Démonstration des modes ECB et CBC sur des images présentants une forte structure. L'AES-128 a été utilisé comme algorithme de chiffrement.

qu'il y a très peu de pixels différents dans les images en clair, il y a également très peu de blocs de messages différents passés au mode ECB. Par conséquent, beaucoup de blocs m sont chiffrés

par le même bloc $\mathcal{E}_K(m)$ pour une clef donnée, et on visualise directement cet effet dans les images chiffrées.

Pour pallier ce point faible, il existe plusieurs autres modes, comme le mode de chaînage CBC (*Cipher-Block Chaining*). Le mécanisme est décrit sur la Figure 1.13 et le résultat sur l'exemple précédent est montré sur la Figure 1.12. Le chaînage entraîne une dépendance entre

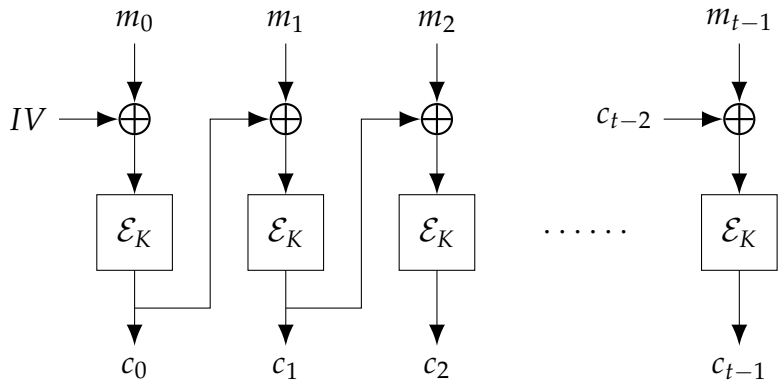


Figure 1.13: Le mode CBC (*Cipher-block chaining*). Les t blocs de messages $m_0, \ldots, m_{t-1}$ sont traités en chaîne par la permutation $\mathcal{E}_K$ pour former la séquence de blocs chiffrés $c_0, \ldots, c_{t-1}$.

un bloc chiffré et tous les blocs chiffrés précédents, ce qui permet de rendre aléatoire les entrées séquentielles de la permutation $\mathcal{E}_K$. Il existe plusieurs autres modes opératoires pour les algorithmes de chiffrement par bloc. On peut par exemple citer le chiffrement à rétroaction d'entrée (*Cipher Feedback*, CFB) ou de sortie (*Output Feedback*, OFB), le mode compteur (*counter*, CTR), etc.

1.4.4 Fonction de compression

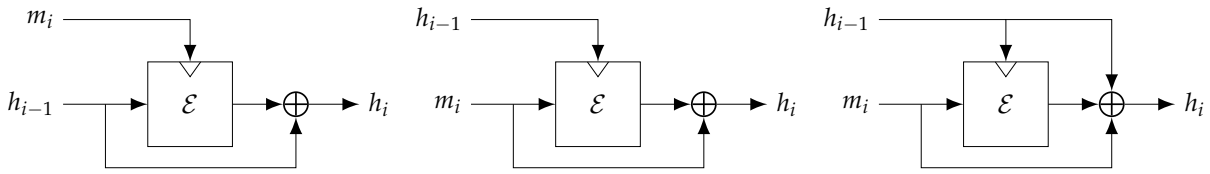
Dans la Section 1.3, nous avons décrit les fonctions de hachage et leurs propriétés, mais nous n'avons pas proposé de manière de les construire. Beaucoup de fonctions de hachage se basent en réalité sur un algorithme de chiffrement par bloc pour former leur fonction de compression. Rappelons qu'une fonction de compression f prend deux paramètres en entrée : la valeur de chaînage courante h et le bloc de message à intégrer m. La valeur renvoyée $f(h, m)$ est une nouvelle valeur de chaînage h' pour la prochaine application de f.

Il existe des schémas célèbres qui permettent de construire f grâce à un algorithme de chiffrement par bloc $\mathcal{E} : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n$ comme vu précédemment. La manière d'utiliser $\mathcal{E}$ doit cependant vérifier certains principes des fonctions de hachage, et en particulier la résistance en préimage. On doit les premières constructions à Davies et Meyer (Figure 1.14a), et à Matyas, Meyer et Oseas (Figure 1.14b). Dans le cas de Davies-Meyer, on construit la fonction de compression f avec :

$$f_{DM}(h, m) = \mathcal{E}_{m_i}(h_{i-1}) \oplus h_{i-1},$$

alors que la construction Matyas-Meyer-Oseas inverse les deux entrées :

$$f_{MMO}(h, m) = \mathcal{E}_{h_{i-1}}(m_i) \oplus m_i.$$



(a) Davies-Meyer. **(b)** Matyas-Meyer-Oseas. **(c)** Miyaguchi-Preneel.

Figure 1.14: Modes opératoires classique pour fonctions de compression à base d'un algorithme de chiffrement par bloc.

Ces deux méthodes de construction offrent une forte résistance face aux préimages : étant donné y, il est difficile de trouver (h, m) tels que $f(h, m) = y$. En revanche, on peut très simplement créer des point fixes dans la fonction de compression. En effet, si l'on impose $f(h, m) = h$ dans Davies-Meyer par exemple, alors on déduit que $\mathcal{E}_m(h) = 0$, et pour n'importe quel message m, il suffit alors de choisir $h = \mathcal{E}_m^{-1}(0)$.

Une autre variante de construction a par la suite été proposée par Miyaguchi et Preneel (Figure 1.14c) : elle étend celle de Matyas-Meyer-Oseas en ajoutant la valeur de chaînage précédente dans le XOR final. Toutes les variantes possibles de construction ont été étudiées par Preneel, Govaerts et Vandewalle dans [PGV94] : on les surnomme désormais les constructions PGV du nom de leurs auteurs. En combinant les entrées entre elles (ou non), on peut construire 2^6 constructions différentes dont beaucoup ne conviennent pas pour une fonction de compression car elles ne dépendent par exemple pas de h_{i-1}. Cependant, 12 sont prouvées sûres : on retrouve, par exemple, les trois constructions de Davies-Meyer, Matyas-Meyer-Oseas et Miyaguchi-Preneel.

1.4.5 Cryptanalyse des algorithmes de chiffrement par bloc

Pour attaquer des schémas de chiffrement par bloc, il existe des méthodes désormais classiques issues d'attaques spécifiques à certaines méthodes de chiffrement. Toutes ces méthodes convergent vers un seul but : exhiber une propriété non triviale de l'algorithme de chiffrement, et si possible l'utiliser pour récupérer la valeur secrète utilisée lors d'un (ou de plusieurs) chiffrement(s). D'une manière générale, comme on l'a déjà mentionné, une attaque consiste en un algorithme plus efficace que la recherche exhaustive (de la clef), à savoir une technique moins coûteuse que la recherche des 2^k clefs de k bits.

Dans cette thèse, nous nous intéressons exclusivement à des attaques sur des primitives symétriques, et en particulier au standard de chiffrement actuel : l'AES. Nous détaillerons beaucoup plus certaines des attaques classiques utilisées actuellement, mais on peut par exemple citer l'attaque par le milieu qui a été découverte alors que l'on cherchait une manière de renforcer la sécurité du DES. Celui-ci était le standard de chiffrement entre 1977 et 1999 et permet de chiffrer avec des clefs de 56 bits. Suivant les cas, on peut vouloir utiliser plus que 56 bits de clefs, et il est alors naturel de se demander comment étendre DES à des clefs plus grandes. Une première idée consiste à utiliser deux clefs K_1 et K_2 et d'appliquer deux fois le DES à un message m successivement avec les deux clefs pour le transformer en son chiffré $c = \mathrm{DES}_{K_2}(\mathrm{DES}_{K_1}(m))$. Cette méthode semble apporter une sécurité de $2 \times 56 = 112$ bits avec

ces deux clefs, or le mécanisme appelé `Double-DES` est vulnérable à une technique triviale d'attaque par le milieu (*meet-in-the-middle*).

1.4.5.1 Attaque par le milieu

Supposons qu'un adversaire ait obtenu un message m et son chiffré c, et qu'il sache que le `Double-DES` a été utilisé. Plutôt que de faire une recherche exhaustive sur les 2^{112} clefs (K_1, K_2) possibles, il va procéder en deux temps. Dans un premier temps, il énumère toutes les 2^{56} clefs K_1 possibles, et stocke le chiffrement du message m de 64 bits sous la clef K_1 par DES dans une table T (voir Figure 1.15). Cette table contient des éléments de la forme (e, K_1), où

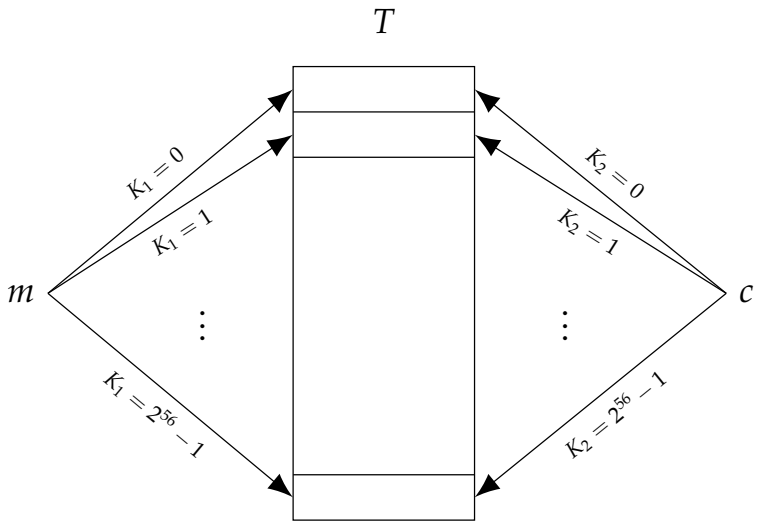


Figure 1.15: Attaque par le milieu. L'attaquant récupère les clefs K_1 et K_2 plus rapidement que la recherche exhaustive en utilisant une mémoire de $2^{|K_1|}$ éléments.

$e = \mathrm{DES}_{K_1}(m)$ et est stockée en classant les éléments e par ordre croissant. Notons au passage que construire cette table T demande 2^{56} évaluations du DES et une mémoire capable de stocker les 2^{56} éléments. Dans un deuxième temps maintenant, il va partir du message chiffré c de 64 bits et énumérer toutes les 2^{56} valeurs possibles pour la deuxième clef K_2. Pour chacune de ces valeurs, il calcule le déchiffrement de c par la clef K_2 et obtient $\mathrm{DES}^{-1}_{K_2}(c)$. Pour ces valeurs obtenues en séquence pour chaque nouvelle tentative de K_2, on obtient en moyenne $2^{56}/2^{64} = 2^{-8}$ élément e de la table T. En effet, nous stockons 2^{56} valeurs dans T, vues comme des variables aléatoires sur 64 bits suivant la loi uniforme, et nous sélectionnons un élément par 56 bits de $\mathrm{DES}^{-1}_{K_2}(c)$.

Ainsi, pour une valeur de K_2 et avec probabilité 2^{-8}, la table T suggère un élément (e, K_1) qui peut correspondre à une valeur correcte pour K_1. Cette suggestion construit une paire de clef (K_1, K_2) qu'il faut tester sur une nouvelle paire clair/chiffré (m', c') que l'adversaire est supposé avoir. L'information fournie par cette deuxième paire permet de supprimer toutes les suggestions (K_1, K_2) sauf les valeurs correctes qui vont également vérifier $c' = \mathrm{DES}_{K_2}(\mathrm{DES}_{K_1}(m'))$. Finalement, avec deux paires claire/chiffré, 2×2^{56} évaluations du DES et en utilisant une mémoire de 2^{56} éléments, nous avons retrouvé les 112 bits de clef. Cette attaque classique contredit l'intuition qui pousse à croire que doubler la taille des clefs double la sécurité de la méthode de chiffrement. Pour répondre au problème intial, il est toutefois possible d'augmenter

la sécurité du DES, mais il faut alors considérer trois clefs (K_1, K_2, K_3) et chiffrer m en

$$c = \mathrm{DES}_{K_3}(\mathrm{DES}^{-1}_{K_2}(\mathrm{DES}_{K_1}(m'))).$$

On appelle cette primitive le `Triple-DES`. L'appel central consiste en un déchiffrement pour des raisons de compatibilité avec le simple DES : en effet, prendre $K_1 = K_2 = K_3$ revient à faire un simple DES, ce qui peut être utile dans certains cas.

1.4.5.2 Distance d'unicité

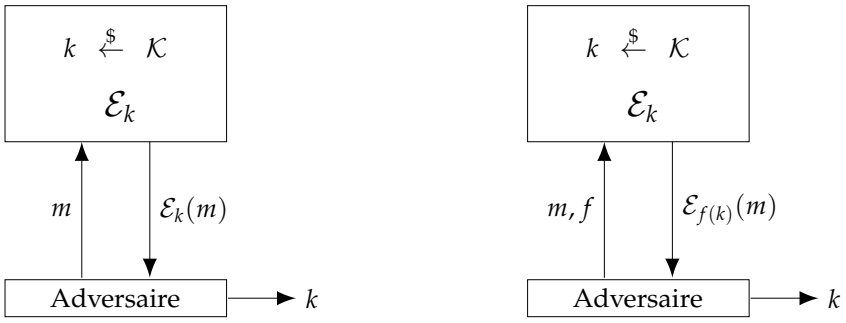
Dans l'attaque par le milieu présentée dans la section précédente, nous avons vu que l'adversaire a eu besoin de deux couples clair/chiffré pour mener à bien son attaque. D'une manière générale, on parle de *distance d'unicité* pour mesurer la quantité d'information nécessaire à cet adversaire pour isoler théoriquement et sans ambiguité la seule clef secrète possible. Cette information est mesurée en termes de paires clair/chiffré et rapporte un comportement en moyenne.

Formellement, la distance d'unicité d se rapproche de l'entropie de Shannon h et peut se définir de manière inconditionnelle par le nombre minimal de messages chiffrés c_i à connaître pour réduire l'entropie de la clef secrète K à zéro :

$$d = \min_t h\Big(K \,\Big|\, c_1, \ldots, c_t\Big) = 0.$$

1.4.5.3 Modèles d'attaquants

Lorsque l'on considère un attaquant face à un algorithme de chiffrement par bloc, il convient de modéliser ses capacités afin de borner les requêtes qu'il peut faire. Nous avons déjà évoqué le modèle à clairs connus, à clairs choisis ou encore à chiffrés choisis, qui sont des scénarios précisant des contraintes sur l'une des deux entrées de l'algorithme de chiffrement, à savoir le message. Cependant, il est également possible de préciser les capacités de l'adversaire vis-à-vis de la seconde entrée de l'algorithme : la clef.

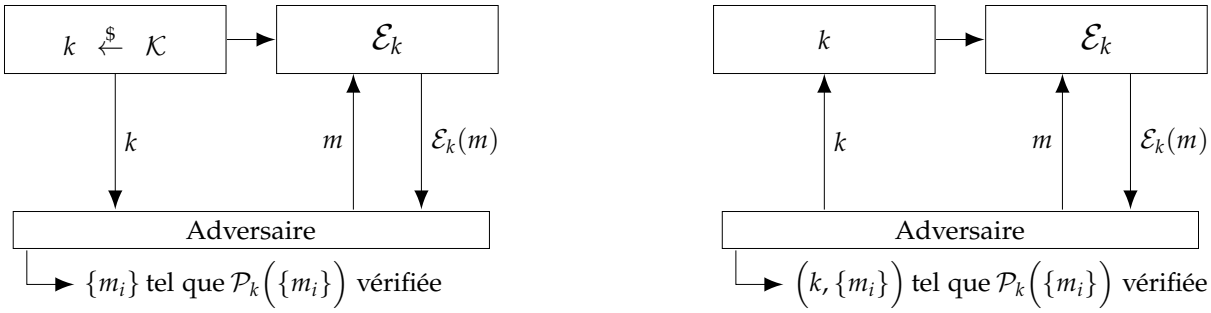


(a) Modèle standard **(b)** Modèle à clefs reliées.

Figure 1.16: Modèles d'attaques en clef secrète : le modèle standard (a) et le modèle en clefs reliées (b) .

Il existe plusieurs modèles de ce point de vue, dont le plus pertinent en pratique est le modèle standard (Figure 1.16a) dans lequel l'adversaire n'a aucune influence sur la clef secrète. Celle-ci est choisie aléatoirement dans l'espace $\mathcal{K}$ de toutes les clefs possibles, et sert lors

du chiffrement des messages reçus ou choisis par l'adversaire. Un deuxième modèle plus souple et moins réaliste dans la pratique est le modèle en clefs reliées (Figure 1.16b). Dans ce cas, l'adversaire ne connaît toujours pas la clef secrète utilisée, mais il peut demander le chiffrement de messages avec k ou avec n'importe quelle clef reliée à k par une certaine relation f. L'ensemble des clefs k' qui sont en relation avec la clef secrète k est également secret, i.e. l'adversaire ne connait aucune clef k', mais il peut cependant obtenir plus d'informations sur k dans ce modèle que dans le modèle standard. La relation f peut varier, et nous considérons un cas particulier dans le chapitre 6.



(a) Modèle en clef connue.

(b) Modèle en clef choisie.

Figure 1.17: Modèles d'attaques en clef ouverte : le modèle en clef connue (a) et le modèle en clef choisie (b). La propriété $\mathcal{P}_K$ dépend de la clef k, et prend en entrée un ensemble de un ou plusieurs messages.

Un autre ensemble de scénarios a été récemment proposé par Knudsen et Rijmen et consiste en un modèle complètement ouvert où l'on donne la clef à l'adversaire. Son but n'est plus de la retrouver comme dans les modèles précédents, mais d'exhiber une propriété non triviale $\mathcal{P}_k$ sur la primitive lorsque tous les paramètres sont connus, y compris la clef k. On distingue plus particulièrement deux cas : le modèle en clef connue (Figure 1.17a) où l'on donne explicitement une valeur de k à l'adversaire, ce qui lui fixe totalement la permutation $\mathcal{E}_k$ à attaquer. Un deuxième modèle moins contraint consiste à lui laisser spécifier une valeur particulière de la clef k (Figure 1.17b) pour lui faciliter la recherche de valeurs satisfaisant $\mathcal{P}_k$. Nous détaillons ces deux modèles dans le chapitre 7. Dans le chapitre suivant, je détaille les contributions de cette thèse, chapitre par chapitre.

CHAPITRE 2

Présentation des Travaux

2.1 Présentation des travaux

Mes travaux durant cette thèse se sont principalement articulés autour de la compétition SHA-3. Cette compétition a été annoncée par le *National Institute of Standards and Technology* (NIST) à la fin de l'année 2007 et s'est terminée 5 ans plus tard en octobre 2012 par la victoire de Keccak. Au début de ma thèse, j'ai été amené à étudier la fonction de hachage ECHO, également soumise à cette compétition. ECHO a atteint la deuxième étape du processus de sélection, mais n'a pas été choisie pour la finale. Mes premiers travaux ont permis d'appliquer l'attaque par rebond sur cette fonction, pour montrer plusieurs attaques sur des versions réduites. Ensuite, j'ai continué l'étude des fonctions de hachage de la compétition SHA-3, en particulier Grøstl, qui a été sélectionnée pour la finale. Comme ECHO, cette fonction réutilise les idées de l'AES comme base de la permutation interne de la fonction de compression. Là encore, l'attaque par rebond peut s'appliquer et il est possible d'améliorer grandement les résultats précédents. Finalement, dans la dernière partie de ma thèse, j'ai dévié du domaine des fonctions de hachage pour m'intéresser à l'AES comme algorithme de chiffrement. J'ai pu l'étudier dans trois modèles différents : le modèle standard, où l'on cherche à récupérer la clef secrète, le modèle à clefs reliées et le modèle ouvert, où la clef est supposée connue de l'adversaire.

Chronologiquement, mes travaux présentent une progression des fonctions de hachage vers les algorithmes de chiffrement par bloc. Cependant, dans ce manuscrit, j'ai préféré commencer par décrire mes travaux sur l'AES, étant donné que les fonctions de hachage que j'ai étudiées réutilisent ses concepts structurels. Dans la suite de ce chapitre, je décris plus en détail chacune de mes contributions.

Chapitre 3

Dans ce chapitre, je reviens sur la cryptanalyse différentielle sur laquelle toute ma thèse repose. J'y rappelle les principales définitions et notations usuelles utilisées dans la littérature, ainsi que plusieurs résultats classiques. En particulier, je décris les concepts de *différentielle* et de *caractéristique différentielle* qui sont à la base de la cryptanalyse différentielle. Je détaille comment ce type d'attaque s'applique aux algorithmes généraux de chiffrement par blocs, et plus en détail à ceux qui définissent un processus de Markov.

J'aborde l'attaque différentielle basique de recouvrement de clef, qui demande à un adversaire de construire une différentielle de forte probabilité sur $n-1$ tours pour récupérer efficacement la sous-clef introduite au n-ème tour. Ensuite, je rappelle des variantes de cette technique, telles que la cryptanalyse en différentielle impossible, la cryptanalyse en différence tronquée ou encore l'attaque boomerang. Je précise également le modèle à clefs reliées, qui autorise l'adversaire à observer le chiffrement de messages avec plusieurs clefs secrètes différentes, mais qui vérifient une certaine relation.

Enfin, je discute comment ces techniques peuvent être appliquées aux fonctions de hachage. En effet, contrairement aux algorithmes de chiffrement, celles-ci n'ont aucun paramètre secret et on peut alors utiliser les techniques de cryptanalyse différentielle différemment pour mettre en défaut certaines propriétés cryptographiques requises. Dans la dernière section de ce chapitre, je décris le principe général de l'attaque par rebond, méthode utilisée à de nombreuses reprises dans les chapitres de ce manuscrit.

Chapitre 4

Le quatrième chapitre est consacré à la description du standard actuel de chiffrement : l'AES (*Advanced Encryption Standard*). Je détaille les spécifications exactes des trois versions de l'AES, et généralise sa structure pour définir la notion de permutation AES-*like*, qui est utilisée par la suite pour pouvoir regrouper plusieurs primitives par une même analyse.

La deuxième moitié du chapitre décrit les principales attaques qui ont été publiées sur l'AES depuis sa présentation en 1997 par Joan Daemen et Vincent Rijmen. Je ne décris pas *toutes* les attaques existantes sur l'AES dans ce manuscrit, mais je me suis concentré sur les plus importantes. Je rappelle en particulier l'attaque intégrale de Knudsen qui a été réutilisée de nombreuses fois par la suite sur beaucoup d'autres algorithmes, et qui permet de casser 6 tours d'AES. Ensuite, je montre comment la technique des sommes partielles de Ferguson et al. permet d'optimiser les calculs de l'attaque intégrale, puis comment Henri Gilbert et Marine Minier ont ajouté un tour dans le distingueur intégral. Par la suite, je rappelle les principales attaques en différentielle impossible sur l'AES qui ont été pendant longtemps les meilleures attaques existantes. Finalement, je donne un aperçu des attaques récentes en clefs reliées de Biryukov et al.

Chapitre 5

Dans ce chapitre, l'AES est étudié dans le modèle standard où un attaquant essaye de récupérer la clef secrète utilisée lors du chiffrement. Il est classique en cryptanalyse de ne pas s'attaquer à la primitive complète, mais plutôt de considérer des versions réduites qui sont plus simples à analyser. Ainsi, dans ce chapitre, je détaille des attaques de recouvrement de clef sur des versions réduites de l'AES, en ne considérant par exemple que 7 des 10 tours de l'AES-128. Ce chapitre a largement été inspiré par l'article [DFJ13] que j'ai écrit avec Patrick Derbez et Pierre-Alain Fouque, publié à EUROCRYPT 2013.

Dans ce chapitre, nous montrons comment généraliser et améliorer les attaques sur l'AES de Demirci et Selçuk [DS08] et de Dunkelman, Keller et Shamir [DKS10]. En 2008, Demirci et Selçuk ont montré comment monter une attaque de type *meet-in-the-middle* en tabulant

complètement le comportement de 4 tours d'AES. En 2010, Dunkelman, Keller et Shamir ont raffiné cette méthode en proposant d'ajouter une propriété pour ces 4 tours. En effet, ils procèdent à la même tabulation que Demirci et Selçuk pour les tours du milieu, mais imposent une propriété différentielle supplémentaire pour les éléments ajoutés à la table. Cette propriété repose sur une caractéristique différentielle particulière, qui réduit l'entropie possible pour les tours considérés. Leur attaque propose une manière d'énumérer les éléments à stocker dans les tables, mais la procédure peut être améliorée. En effet, leur méthode n'est pas optimale et nous montrons comment gagner un facteur 2^{48} dans la taille de la table à stocker. Cette amélioration permet d'obtenir les meilleures attaques connues sur 7 tours de toutes les versions de l'AES et d'atteindre une attaque sur 9 des 14 tours d'AES-256.

Chapitre 6

Dans ce chapitre, je me suis intéressé aux conséquences du modèle en clefs reliées sur la structure de l'AES. Dans ce modèle, on suppose que l'adversaire est capable de chiffrer ou de déchiffrer des messages qu'il choisit sur un ensemble de clefs reliées par une certaine relation publique. Ainsi, contrairement au modèle standard où l'adversaire n'observe le chiffrement qu'avec une unique clef secrète k, il peut par exemple choisir de chiffrer un même message avec deux clefs k et k', telles que k et k' soient reliées par une différence δ connue : $k \oplus k' = \delta$. Plusieurs types de relations sont possibles, mais dans cette thèse, je me suis limité aux relations différentielles du type $k \oplus k' = \delta$, qui permettent à l'adversaire d'introduire des différences dans la clef.

Bien que ce modèle soit moins pertinent dans la pratique que le modèle standard, il offre cependant de fortes indications sur la qualité de l'algorithme d'expansion de clef utilisé dans les algorithmes de chiffrement par bloc. Dans le cas particulier de l'AES, le mécanisme d'expansion de clef repose sur une procédure ad hoc sans véritable justification théorique, et qui permet de construire des attaques sur les versions complètes de l'AES-192 et l'AES-256 [BKN09, BK09]. Dans ce chapitre, je m'intéresse aux conséquences de ce modèle sur l'AES-128. J'ai pour cela développé un algorithme de recherche des meilleures caractéristiques différentielles en clefs reliées sur un nombre arbitraire de tours d'AES-128. Ce chapitre est le résultat d'une collaboration avec Pierre-Alain Fouque et Thomas Peyrin et a été publié dans l'article [FJP13a] de Crypto 2013, et dans une version longue dans [FJP13b].

Nous montrons dans ce chapitre que l'algorithme de recherche des meilleurs caractéristiques différentielles peut se réduire à un algorithme de plus court chemin dans un graphe acyclique orienté particulier qui représente les transitions différentielles par tour d'AES. Par une variante de l'algorithme de Dijkstra, il est possible d'énumérer efficacement tous les meilleurs chemins dans ce graphe et donc les caractéristiques en différentielles tronquées qui possèdent la meilleure probabilité. Nous détaillons ensuite comment cette représentation définit en réalité un processus de Markov auquel il est possible d'ajouter de l'information pour trouver les caractéristiques différentielles avec des différences instantiées. Cette recherche permet de trouver les probabilités exactes des meilleures caractéristiques différentielles sur l'AES-128.

De plus, les résultats de cette recherche nous permettent de procéder à une analyse structurelle de l'AES-128. Cette analyse a pour but d'estimer la sécurité de la structure de l'AES lorsque les composants principaux tels que la S-Box et la couche de diffusion linéaire ne sont

pas spécifiés. Ainsi, la résistance à la cryptanalyse différentielle est exprimée en fonction de plusieurs paramètres, ce qui permet par exemple d'affirmer qu'il est impossible d'obtenir la résistance en cryptanalyse différentielle dans le modèle à clefs reliées sans prendre en compte l'instantiation de la S-Box. En effet, nous détaillons dans ce modèle une attaque structurelle sur les 10 tours de l'AES-128. De même, dans le cas où l'AES-128 est utilisé dans un mode tel que Davies-Meyer pour définir une fonction de compression, prouver la résistance à la cryptanalyse différentielle nécessite de fournir l'instantiation de la S-Box et de la couche linéaire. Ces résultats théoriques ne remettent pas en cause la sécurité de l'AES mais donnent une indication de sa qualité structurelle.

Chapitre 7

Dans ce chapitre, j'ai poursuivi l'étude récente de l'AES dans le modèle à clef connue ou choisie. Dans ce modèle, on suppose que l'adversaire a connaissance de la clef utilisée lors du chiffrement ou du déchiffrement (clef connue), voire même qu'il est capable de choisir sa valeur (clef choisie). Ce modèle a initialement été introduit par Lars Knudsen et Vincent Rijmen [KR07] mais ne représente pas de risques pratiques très importants. Il permet en revanche d'évaluer les capacités de l'adversaire dans un modèle ouvert qui se rapproche du domaine des fonctions de hachage.

Je rappelle tout d'abord les principaux algorithmes qui permettent de distinguer l'AES d'une permutation aléatoire dans ce modèle. Ces résultats se basent sur l'attaque par rebond [MRST09] et exhibent des propriétés non aléatoires sur des versions de l'AES réduites jusqu'à 8 tours lorsque l'attaquant connaît la valeur de la clef secrète. Le travail que j'ai effectué dans cette direction a été publié à INDOCRYPT 2012 dans l'article [DFJ12a] écrit avec Patrick Derbez et Pierre-Alain Fouque et montre comment il est possible d'étendre les résultats précédents au cas où l'adversaire peut choisir la valeur de la clef.

Avant ce travail, on ne connaissait pas de méthode qui permettait de tirer parti des degrés de liberté supplémentaires introduits lors du passage du modèle en clef connue au modèle en clef choisie. Dans ce chapitre, nous montrons que si l'adversaire a cette capacité, alors il peut contrôler un tour de plus dans la phase non probabiliste de l'attaque par rebond. Ces nouvelles techniques consistent principalement à réordonner les étapes des attaques précédentes afin d'obtenir des attaques optimales dans la classe d'attaques considérée.

Dans ce modèle à clef choisie, nous montrons également le premier distingueur pour 9 tours d'AES-128, en tant qu'application de l'algorithme de recherche présentée au chapitre précédent. Le problème de montrer une propriété non triviale sur 9 tours d'AES-128 était ouvert depuis longtemps dans la communauté de la cryptographie symétrique, et nous montrons qu'il existe un algorithme capable de générer une paires de clefs et une paire de messages vérifiant toutes les deux certaines relations plus rapidement que pour une famille de permutations aléatoires. Ce résultat a été publié comme application de l'article [FJP13a] de CRYPTO 2013.

Chapitre 8

Étant donné que beaucoup de nouvelles attaques sur les fonctions de hachage utilisent la nouvelle technique de l'attaque par rebond, j'ai dédié tout le huitième chapitre à l'analyse de

cette méthode. Ce travail a été fait en collaboration avec María Naya-Plasencia et Thomas Peyrin et a fait l'objet de deux articles : le premier a été publié à FSE 2012 [JNPP12], et le deuxième est actuellement en soumission.

Ce chapitre analyse successivement les deux parties de l'attaque par rebond. Cette technique s'inscrit dans le cadre de la cryptanalyse différentielle et donne un moyen de trouver une paire de messages qui satisfait une caractéristique différentielle donnée. Elle consiste en une première partie non probabiliste qui trouve efficacement des paires de messages pour la section centrale de la caractéristique, puis en une deuxième partie, qui énumère ces paires tant qu'aucune ne vérifie une certaine propriété. Cette deuxième phase est probabiliste et la caractéristique choisie doit donc faire en sorte que la probabilité de succès soit la plus grande possible.

Nous étudions dans un premier temps la phase non probabiliste, et nous abordons ensuite la phase probabiliste. Dans ces deux études, nous nous concentrons sur la principale cible vis-à-vis de ces attaques, à savoir l'AES. La première étape consiste à générer le plus de paires vérifiant la partie centrale de la caractéristique le plus efficacement possible. Idéalement, nous voudrions générer une paire en une opération. Plusieurs algorithmes présentés dans le chapitre 7 permettent d'atteindre cet objectif, mais ils sont limités à seulement deux tours d'AES. Avant notre travail, la question de savoir gérer un tour de plus pour contrôler trois tours centraux était un problème ouvert. Nous montrons dans ce chapitre que pour une permutation à base d'AES quand l'état est au moins de taille 8, il existe un algorithme "efficace" générant les paires voulues pour les tours du milieu de la caractéristique. La complexité amortie de cet algorithme est moins bonne que le coût idéal d'une opération par paire, mais nous montrons néanmoins qu'il est possible de gagner un tour supplémentaire dans les distingueurs précédents. La nouvelle question ouverte est désormais de savoir s'il est possible d'atteindre une meilleure complexité amortie.

Dans un deuxième temps, nous nous intéressons à la phase probabiliste de l'attaque par rebond en supposant connu l'algorithme générant les paires pour la section centrale. En considérant une permutation basée sur l'AES, la phase probabiliste consiste à attendre l'annulation de différences dans le **MixColumns**. Pour l'AES par exemple, on attend généralement que quatre différences se transforment en une seule différence à une position spécifiée à l'avance par la caractéristique. Pour augmenter la probabilité, nous proposons de relaxer la position de cette différence. Ainsi, dans le cas de l'AES, la probabilité est par exemple multipliée par quatre. Le gain global reste marginal, mais améliore cependant toutes les attaques sur des permutations à bases d'AES utilisant l'attaque par rebond. Hormis ces résultats, la contribution théorique de ce chapitre consiste en une évaluation de la complexité générique dans le cas d'une permutation aléatoire. Nous proposons une borne inférieure basée sur l'algorithme du *limited-birthday* pour borner la complexité générique de ce problème que nous avons baptisé *multiple limited-birthday*. De plus, dans le cas des permutations basées sur l'AES, nous présentons un algorithme qui résout ce problème plus rapidement que la borne générique.

Chapitre 9

Dans ce dernier chapitre, j'étudie la fonction de hachage ECHO, qui a été soumise à la compétition SHA-3 du NIST par Henri Gilbert et al. en 2007. Cette fonction de hachage utilise l'AES comme composant interne pour créer un état de 16 états d'AES. La fonction de compres-

sion applique 8 tours d'une permutation ressemblant à l'AES ce qui permet alors d'appliquer des techniques d'analyse similaires. Le point de départ de mon travail a été une erreur dans une attaque publiée par Martin Schläffer à SAC 2010 et a initialement consisté à la corriger. L'erreur a été de négliger la résolution d'un système linéaire issu d'un réordonnancement des opérations de la permutation interne d'ECHO. Ce système est de la forme $\mathbf{M} \times X = A$ où certains paramètres sont fixés, mais où la matrice $\mathbf{M}$ a une forme très particulière et ne garantit pas que toutes les valeurs des paramètres fassent que le système ait des solutions. En effet, je montre dans ce chapitre que pour des valeurs aléatoires des paramètres, le système a une solution avec probabilité seulement 2^{-128}. Cependant, je montre qu'il est possible de corriger l'attaque afin de trouver des solutions au même système linéaire. Ceci donne une attaque en collision pratique sur la fonction de compression d'ECHO réduite à quatre tours. Ce résultat a été publié dans l'article [JF11] avec Pierre-Alain Fouque à FSE 2011.

Dans un deuxième temps, j'ai continué à travailler sur ECHO en collaboration avec María Naya-Plasencia et Martin Schläffer pour étendre les résultats précédents en prenant en compte la correction sur le système linéaire. Ce travail a donné plusieurs résultats qui utilisent tous la même technique. Nous avons utilisé l'attaque par rebond deux fois à deux endroits différents dans une longue caractéristique différentielle sur la fonction de compression. Il est possible d'utiliser une attaque avec plusieurs rebonds sur ECHO étant donné le grand nombre de degrés de liberté dont on dispose pour cette fonction. Nous avons obtenu plusieurs résultats : une attaque en collision sur 4 tours de la fonction de hachage, une attaque en collision sur 5 tours de la fonction de compression, un distingueur sur 7 tours de la fonction de compression, une attaque en collision sur 6 tours de la fonction de compression ainsi que des attaques lorsque l'attaquant peut choisir un paramètre additionnel : le sel. Toutes ces attaques ont été publiées dans un article [JNPS11a] à SAC 2011 ainsi que dans une version étendue de cet article sur l'ePrint [JNPS11b].

2.2 Liste des publications

[JF11] Practical Near-Collisions and Collisions on Round-Reduced ECHO-256 Compression Function
Jérémy Jean et Pierre-Alain Fouque
FSE 2011

[JNPS11a] Improved Analysis of ECHO-256
Jérémy Jean, María Naya-Plasencia et Martin Schläffer
SAC 2011

[JNPS11b] **ePrint** — Version longue de l'article de **SAC 2011**

[JNPP12] Improved Rebound Attack on the Finalist Grøstl
Jérémy Jean, María Naya-Plasencia et Thomas Peyrin
FSE 2012

[DFJ12a] Faster Chosen-Key Distinguishers on Reduced-Round AES
Patrick Derbez, Pierre-Alain Fouque et Jérémy Jean
INDOCRYPT 2012

[JNP+13] Security Analysis of PRINCE
Jérémy Jean, Ivica Nikolić, Thomas Peyrin, Lei Wang et Shuang Wu
FSE 2013

[DFJ13] Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting
Patrick Derbez, Pierre-Alain Fouque et Jérémy Jean
EUROCRYPT 2013

[DFJ12b] **ePrint** — Version longue de l'article d'**EUROCRYPT 2013**

[JNPP13b] Multiple Limited-Birthday Distinguishers and Applications
Jérémy Jean, María Naya-Plasencia et Thomas Peyrin
SAC 2013

[JNPP13c] **ePrint** — Version longue de l'article de **SAC 2013**

[FJP13a] Structural Evaluation of AES and Chosen-Key Distinguisher of 9-round AES-128
Pierre-Alain Fouque, Jérémy Jean et Thomas Peyrin
CRYPTO 2013

[FJP13b] **ePrint** — Version longue de l'article de **CRYPTO 2013**

[JNPP13a] Improved Cryptanalysis of AES-like Permutations
Jérémy Jean, María Naya-Plasencia et Thomas Peyrin
Journal of Cryptology (*accepté, à paraître*)

CHAPTER 3

Differential Cryptanalysis

Contents

3.1 Preliminaries **34**
3.1.1 Differentials 34
3.1.2 Iterated functions 36
3.1.3 Differential characteristics 37
3.2 Block ciphers **39**
3.2.1 Basic key recovery attack 39
3.2.2 Resistance against differential cryptanalysis 42
3.3 Markov Ciphers **43**
3.4 Other forms of differential cryptanalysis **46**
3.4.1 Truncated differential cryptanalysis 46
3.4.2 Impossible differential cryptanalysis 47
3.4.3 Boomerang attack 48
3.4.4 Related-key attacks 51
3.5 Hash functions **54**
3.5.1 Generalities 54
3.5.2 Rebound attack 55

Differential cryptanalysis has been introduced in 1990 by Eli Biham and Adi Shamir in [BS91a, BS91b] with DES-like cryptosystems as a direct application. This technique is statistical and basically observes the propagation of differences between two inputs through a cryptosystem and tries to predict its behavior with high probability. The method assumes nothing about the actual values of the two messages, but the attack still needs to impose a fixed difference between two plaintexts, which makes it a chosen-plaintext attack.

With primitives which are not fully protected against this strategy, we can deduce nontrivial properties and eventually retrieve the key in the case of some block ciphers. For instance, the same authors show in [BS93] how to attack the full 16 rounds of DES, which was the current standard of encryption by the time of the publication in 1992. Later, different techniques like linear cryptanalysis [Mat94a] have been published to attack DES, but differential cryptanalysis opened a wide line of research in the cryptanalysis of both block ciphers and hash functions. In 1994, Don Coppersmith has revealed that IBM and the NSA were well

aware of the differential cryptanalysis technique, as the DES has been designed to be resistant to it. The discovery was decided not be published, as Coppersmith mentions in [Cop94]:

> “After discussions with NSA, it was decided that disclosure of the design considerations would reveal the technique of differential cryptanalysis, a powerful technique that could be used against many ciphers. This in turn would weaken the competitive advantage the United States enjoyed over other countries in the field of cryptography.”

In the first section of this chapter, we recall the intuition behind the differential cryptanalysis technique and we precise some definitions that may help the reader for the rest of this document.

3.1 Preliminaries

Attacks built using a differential cryptanalysis usually depend heavily on the targeted primitive and its inner structure. Indeed, the cryptanalyst needs to study carefully how an input difference propagates within the system and tries to predict its output value. The term difference could have several meanings and also depends of the target. When we usually consider the XOR difference for many primitives like DES in [BS93], other types of differences have been considered, as modular difference for MD4 in the work of Dobbertin in [Dob96], signed differences in all the so-called Wang's attacks [WY05, WYY05b, WYY05a, WLF+05] on the MD5 family of hash functions, or truncated differences introduced by Knudsen in [Knu94] to analyze byte-oriented primitives. In the following, we consider the most frequent difference which is the XOR difference. We denote $a \oplus b$ the XOR difference between the two elements a and b, but we could adapt $\oplus$ to any kind of difference.

3.1.1 Differentials

Let $F : \{0,1\}^n \to \{0,1\}^m$ be a function with n-bit inputs and m-bit outputs used in a cryptosystem. We make the weak assumption that F is not linear, as if it were, we probably could not use it either as a block cipher or as a compression function. Hence, F being non-linear, a given input difference δ to F could lead to an output difference Δ. That is, we might find two inputs elements a and b such that $a \oplus b = \delta$ and $F(a) \oplus F(b) = \Delta$. We give the following Definition 3.1.

Definition 3.1 (Differential, [LMM91]). A *differential* $\mathcal{D}$ in a function $F : \{0,1\}^n \to \{0,1\}^m$ is a pair composed of an input difference δ and an output difference Δ, and we denote it $\delta \to \Delta$. Moreover, if there exists a pair (a, b) of input elements to F such that $a \oplus b = \delta$ and $F(a) \oplus F(b) = \Delta$, we say that (a, b) verifies $\mathcal{D}$, conforms to it or is a right pair.

An important notion about a differential $\delta \to \Delta$ is the number of pairs that verify it. That is, we are interested in the cardinality of the set

$$E_{\delta \to \Delta} = \left\{ (a, b) \in 2^n \times 2^n \;/\; a \oplus b = \delta \text{ and } F(a) \oplus F(b) = \Delta \right\}. \tag{3.1}$$

Since F is not linear, we cannot deduce easily $|E_{\delta \to \Delta}|$, but if n and m are small enough, we can go through all the 2^n possible inputs for a fixed δ, and check for the differences between

them. Doing so for all δ, allows to construct the *Difference Distribution Table* (DDT), which can be precomputed for small non-linear components as we will discuss in the next chapters.

From the number of pairs $|E_{\delta\to\Delta}|$ that conform to the differential $\delta \to \Delta$, we deduce its probability.

Definition 3.2 (Differential probability). Let $\delta \to \Delta$ be a differential for F. Its *differential probability* $DP^F(\delta, \Delta)$ is defined by:

$$DP^F(\delta, \Delta) = \frac{|E_{\delta\to\Delta}|}{2^n}. \tag{3.2}$$

The differential probability measures the chance that a random pair (a, b) with input difference δ results in a pair with difference Δ through F. That is:

$$DP^F(\delta, \Delta) = 2^{-n} \cdot \Big|\big\{x / F(x) \oplus F(x \oplus \delta) = \Delta\big\}\Big|. \tag{3.3}$$

In terms of probabilities, $DP^F(\delta, \Delta)$ measures the chance that the output difference Δ_o reaches Δ, given that the input difference Δ_i equals δ:

$$DP^F(\delta, \Delta) = \mathbf{Pr}(\Delta_o = \Delta | \Delta_i = \delta). \tag{3.4}$$

We are generally interested in the maximal value that DP^F can reach: the *maximal differential probability*.

Definition 3.3 (Maximal Differential Probability). Let F be a non-linear function. We define the *maximal differential probability* by the maximal value for $DP^F(\delta, \Delta)$ when the input difference δ ranges over all the non-zero possible differences. We denote it p^F_{max}:

$$p^F_{max} = \max_{\delta\neq 0,\Delta} DP^F(\delta, \Delta) \tag{3.5}$$

This notion is particularly useful for small non-linear components (S-Boxes) that bring non-linearity in almost all block ciphers.

Definition 3.4 (Impossible differential). Let $\delta \to \Delta$ be a differential. If $DP^F(\delta, \Delta) = 0$, we say that the differential $\delta \to \Delta$ is an *impossible differential* for F.

The differential cryptanalysis of the function F starts by searching for a differential $\mathcal{D} = \delta \to \Delta$ that holds with very high or very low probability. Then, the adversary asks the encryption of many chosen-plaintext pairs related by the fixed difference δ and computes the differences in the corresponding outputs through F. Finally, he tries to detect statistical patterns in their distribution, which is easier if $\mathcal{D}$ has been chosen to hold with very high or very low probability.

This basic strategy allows to detect a statistical bias in F and to distinguish it from random. Additionally, it is possible to apply similar techniques to block ciphers to perform key recovery attacks, or also to hash functions to find collisions. In the latter case, F is a compression function with $n > m$, and the adversary has to find a pair of inputs (a, b) such that $a \oplus b = \delta \neq 0$ and $F(a) \oplus F(b) = \Delta = 0$, i.e. he is looking for a right pair for the differential $\delta \to 0$, for some non-zero difference δ.

3.1.2 Iterated functions

In practice, symmetric primitives such as block ciphers or compression functions F are built upon a smaller building block f that is iterated a certain amount of times. We refer to such a function f as a *round function* and the number r of iterations heavily depends on the targeted security of F, the structure of f, its differential properties, its algebraic degree, etc. The round function f is usually cryptographically weak, but its iterations bring security to F.

The function f generally takes two inputs: the first one being the current state of computation, and the second one a round-dependent parameter called *round-key*. Round keys may be obtained by the expansion of a master secret k with an optional expansion algorithm: $k \to (k_0, \ldots, k_{r-1})$, but we can also consider the case where all the $k_0, \ldots, k_{r-1}$ are independent and completely specified.

Formally for a non-negative $n < r$, we write $f(s_n, k_n) = s_{n+1}$ the function that transforms the state s_n in one round into the state s_{n+1}, using round-key k_n. Initially, state s_0 is set to the input value and state s_n is the output of the n-th round. In the case of a compression function, there is no key involved, and the values k_i are public.

Definition 3.5 (Key-alternating primitive). Let F be a function that iterates a round function f. We say that F is a key-alternating primitive when the general form $f(s_n, k_n) = s_{n+1}$ for $n < r$ becomes $f(s_n \oplus k_n) = s_{n+1}$, where the current state s_n and the incoming round-key k_n are XORed prior to the application of the round function f. Moreover, a final round-key k_r is added after the r applications of f to produce the output value.

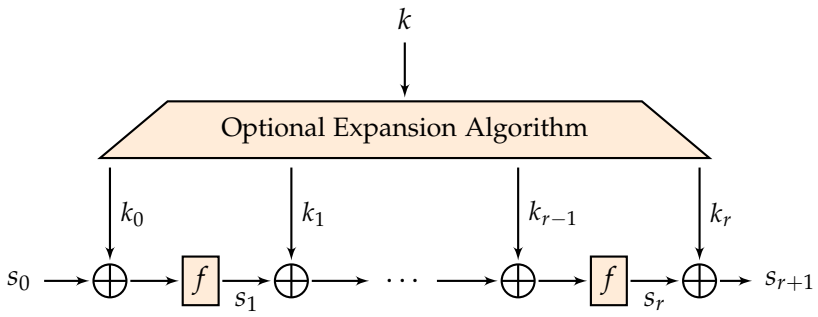


Figure 3.1: Key-alternating block cipher: the function f is applied r times, surrounded by subkey mixing operations.

In the above example of Figure 3.1, we make the assumption that the primitive is a key-alternating function (Definition 3.5). This construction is used in almost all modern algorithms like AES [AES01], PRESENT [BKL+07], SQUARE [DKR97], `Serpent` [BAK98], etc. and generalizes the work initiated by Even and Mansour in [EM97, EM91] which consists of only one application of f surrounded by two independent key additions (see Figure 3.2).

The construction of Even-Mansour is attractive by its minimal character: two keys xored around a single and publicly known permutation f. This simple block cipher offers $n/2$-bit security with $2n$-bit keys, and an attack from 1991 by Daemen in [Dae91] shows that this security bound is tight. The idea by Daemen is simple: the attacker asks for the encryption of D chosen-plaintext pairs with a fixed input difference Δ and collects the corresponding

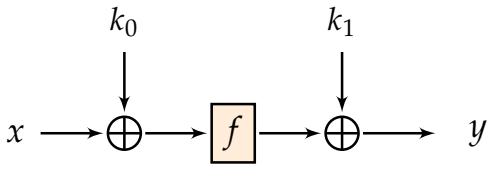


Figure 3.2: Even-Mansour construction: $y = f(x \oplus k_0) \oplus k_1$.

ciphertexts. He then does the same for T chosen plaintexts through the public permutation f. If D and T verify $DT > 2^n$, the birthday paradox ensures with high probability that there exists two pairs (m_1, m_2) and (m_3, m_4) such that we get a collision on the n-bit output difference:

$$\Big(f(m_1 \oplus k_0) \oplus k_1\Big) \oplus \Big(f(m_2 \oplus k_0) \oplus k_1\Big) = f(m_3) \oplus f(m_4).$$

Since f is known, we know the input and output values to f and to the cipher, which allows to deduce the two keys k_0 and k_1. Indeed, the collision on the output difference suggests the equality of the two pairs

$$\Big(f(m_3),\ f(m_4)\Big) \quad \text{and} \quad \Big(f(m_1 \oplus k_0),\ f(m_2 \oplus k_0)\Big),$$

which means that $f(m_3)$ equals either $f(m_1 \oplus k_0)$ or $f(m_2 \oplus k_0)$. In terms of inputs, this implies $m_3 = m_1 \oplus k_0$ or $m_4 = m_2 \oplus k_0$. As the inputs m_1, m_2, m_3 and m_4 are known, we can recover k_0, and therefore k_1 by a similar process. Consequently, this procedure suggests a pair of keys (k_0, k_1) matching an actual encryption, which can easily be verified with another plaintext/ciphertext pair.

This gives an attack in $2^{n/2}$ encryptions of chosen plaintexts, using $T = D = 2^{n/2}$ memory. In 2000, Biryukov and Wagner [BW00] have improved the chosen-plaintext attack into a known-plaintext attack with the same complexity. Their technique makes use of an advanced slide attack, and provides an attack in $2^{n/2}$ data and memory, with no tradeoff possible. More recently, Dunkelman, Keller and Shamir have shown in [DKS12] how to improve this slide attack with any number D of known plaintexts, and a time complexity of T computations that match $DT = 2^n$. Additionally, they revisit the simplified Even-Mansour scheme using $k_0 = k_1$, which makes the block cipher extremely simple (Figure 3.3) with the same n-bit security. This simplification has already been discussed by Kilian and Rogaway in [KR96, KR01].

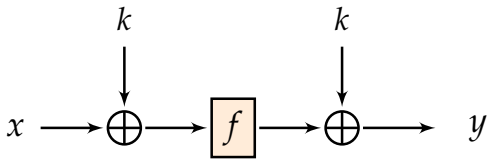


Figure 3.3: Even-Mansour simplified construction: $y = f(x \oplus k) \oplus k$.

3.1.3 Differential characteristics

In all the sequel, we assume that the studied primitive F iterates a round-function f as described in the previous section. When we refer to a difference, we mean a difference in the whole input

state seen as a pair (s_n, k_n).

Definition 3.6 (Differential characteristic). Let f be a round function iterated r times in F. A *differential characteristic* on $r' \in \{1, \ldots, r\}$ rounds in F is a sequence of $r'+1$ differences, denoted $\delta_0 \to \cdots \to \delta_{r'}$.

In this context, the probability of differential on F can be approximated more easily by *differential characteristics*, which consider consecutive one-round differentials in f. From the point of view of graph theory, one may think of a differential characteristic as a path in a graph where the nodes would be the differences between the rounds of F, and the edges the one-round differentials in f. Hence, a differential would consist of *all* the paths from a source node δ to a sink node Δ (see Figure 3.4).

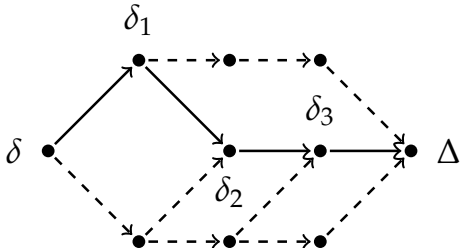


Figure 3.4: Differential and differential characteristic: example of a 4-round differential $\delta \to \Delta$ composed of 5 differential characteristics with a highlighted one: $\delta \to \delta_1 \to \delta_2 \to \delta_3 \to \Delta$.

We say approximation since a differential characteristic from δ to Δ could represent a differential with more than one differential characteristic. While in general there are many differential characteristics within a differential, in some cases, we can find one with a differential probability much higher than all the others. We can thus approximate the differential by this particular characteristic, which represents almost the whole differential probability of the differential itself.

The reason we use characteristics is also that they are easier to construct and handle. In particular, if we assume the independence of all the one-round elementary differentials composing a differential characteristic, we can multiply their respective probability to get an approximation of the differential probability of the differential. The assumption of independence does not hold in practice, but the computed values for cryptographic primitives are usually close enough to the real ones. This is actually the case when we consider the differential probability of a key-alternating cipher averaged over the subkeys.

The reasoning to count the number of right pairs applies in the same manner for differential characteristics, and this leads to a similar formula for the differential probability of a characteristic.

Definition 3.7 (Differential probability). Let $\delta_0 \to \cdots \to \delta_r$ be a differential characteristic for F. Its *differential probability* $DP^F(\delta_0 \to \cdots \to \delta_r)$ is defined by:

$$DP^F(\delta_0 \to \cdots \to \delta_r) = \frac{|E_{\delta_0 \to \cdots \to \delta_r}|}{2^n}, \tag{3.6}$$

where $E_{\delta_0 \to \cdots \to \delta_r}$ denotes the set of all right pairs for the differential characteristic $\delta_0 \to \cdots \to \delta_r$ similarly as in Equation 3.1 for differentials.

Proposition 3.1 (Differential probability). *Let $\delta_0 \to \cdots \to \delta_r$ be a differential characteristic for F. If we assume the independence of the r elementary differential probabilities, we can compute the differential probability $DP^F(\delta_0 \to \cdots \to \delta_r)$ of the differential characteristic by:*

$$DP^F(\delta_0 \to \cdots \to \delta_r) = \prod_{i=0}^{r-1} DP^f(\delta_i \to \delta_{i+1}). \tag{3.7}$$

We elaborate on this independence in the following Section 3.3 on Markov ciphers.

Proposition 3.2 (Differential probability). *The differential probability of the r-round differential $\delta \to \Delta$ equals the sum of all the differential probabilities of the r-round differential characteristics whose first difference is δ and ending difference is Δ. That is:*

$$DP^F(\delta \to \Delta) = \sum_{\delta_1, \ldots, \delta_{r-1}} DF^F(\delta \to \delta_1 \to \cdots \to \delta_{r-1} \to \Delta). \tag{3.8}$$

Obviously, since a differential consists of a set of differential characteristics, its differential probability can only be greater than or equal to the one of a particular differential characteristic with the same input and output differences (Corollary 3.3).

Corollary 3.3. *Let $\delta_0 \to \delta_1 \to \cdots \to \delta_{r-1} \to \delta_r$ be a differential characteristic for F. The differential probability of the differential $\delta_0 \to \delta_r$ is at least the differential probability of the differential characteristic $\delta_0 \to \delta_1 \to \cdots \to \delta_{r-1} \to \delta_r$, that is:*

$$DP^F(\delta_0 \to \delta_r) \geq DP^F(\delta_0 \to \delta_1 \to \cdots \to \delta_{r-1} \to \delta_r). \tag{3.9}$$

Definition 3.8 (Expected differential probability, EDP). If a function F is parameterized by a key k, we consider the differential probability $DP^F[k](\delta_0 \to \delta_r)$ in a straightforward manner. Then, we define the *expected differential probability* $EDP(\delta_0 \to \delta_r)$ of the differential $\delta_0 \to \delta_r$ as the mean value of $DP^F[k](\delta_0 \to \delta_r)$ where the key parameter k follows the uniform distribution when $k \in \mathcal{K}$:

$$EDP(\delta_0 \to \delta_r) = \mathbb{E}\Big[DP^F[k](\delta_0 \to \delta_r),\, k\Big] = 2^{-|\mathcal{K}|} \cdot \sum_{k \in \mathcal{K}} DP^F[k](\delta_0 \to \delta_r).$$

3.2 Block ciphers

3.2.1 Basic key recovery attack

In the case of block ciphers, we are usually interested in recovering the secret key used: we assume that the adversary can either get plaintext/ciphertext pairs (known-plaintext attack), or can ask the encryption of plaintexts of his choice and he receives the corresponding ciphertexts (chosen-plaintext attack).

As discussed in the preliminaries (Section 3.1), differential cryptanalysis can be used to exhibit some statistical bias on some block ciphers. Indeed, a particular differential $\mathcal{D}$ can hold with a differential probability higher than the expected average one. Such a differential can

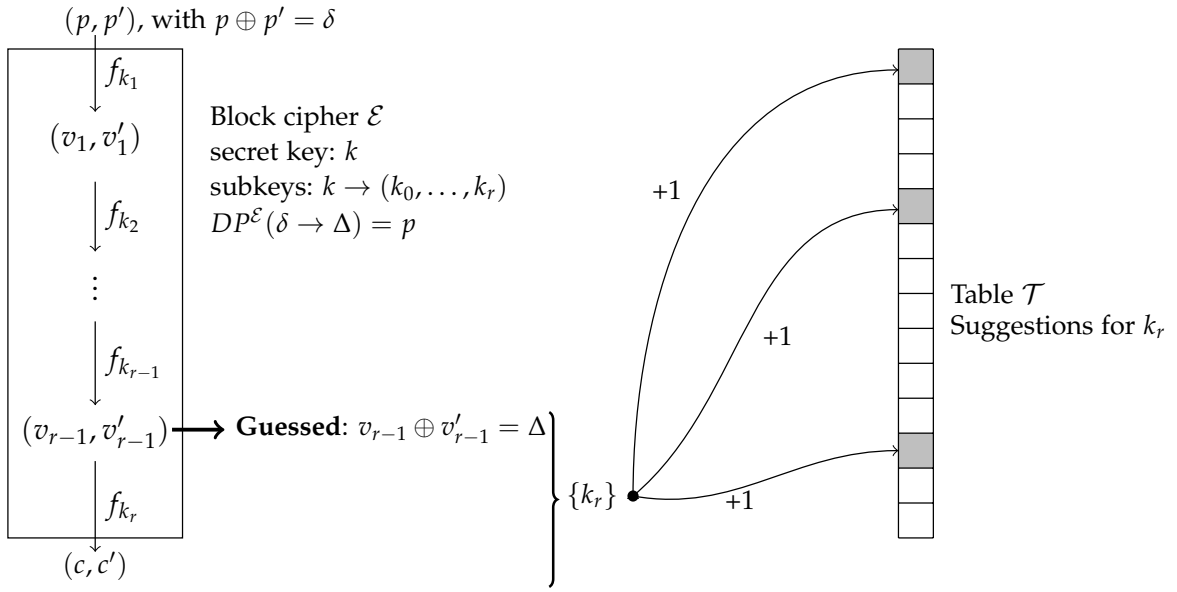


Figure 3.5: Key recovery and differential cryptanalysis: the adversary uses the high-probability differential $\delta \to \Delta$ to get suggestions for the last subkey k_r.

be found by a specific analysis of the targeted cipher, and if it holds for any key, then we can mount a basic key recovery attack (see Figure 3.5).

Assume that the differential $\mathcal{D} = \delta \to \Delta$ holds with high probability on $r-1$ rounds on a block cipher $\mathcal{E}$ with r rounds. Following the previous notations, the round function f which takes as input the current state of computation and a round-key parameter k is denoted by f_k. On Figure 3.5, the encryption of the pair (v_i, v_i') with $f_{k_{i+1}}$ is denoted

$$(v_i, v_i') \xrightarrow{f_{k_{i+1}}} (v_{i+1}, v_{i+1}'), \tag{3.10}$$

where (v_{i+1}, v_{i+1}') is the result of the one-round encryption through $f_{k_{i+1}}$.

The adversary $\mathcal{A}$ is supposed to have oracle access to a black box instantiating $\mathcal{E}$ under a secret key k that he wants to recover. First, $\mathcal{A}$ prepares a set of plaintext pairs that all verify the input difference δ of $\mathcal{D}$, and then he asks the encryption of all the plaintexts p_i under the secret key k to get their respective ciphertexts.

For all the plaintext pairs, the adversary *assumes* that the differential $\mathcal{D}$ has been verified for the first $r-1$ rounds and analyzes the differential transition in the very last round. From the fixed input difference Δ to the last round and the known difference $c \oplus c'$ in the ciphertext, $\mathcal{A}$ gets suggestions for the values of the very last subkey of the cipher. Many subkeys k_r are not possible as they do not connect the input and output of the last round. Hence, this filter allows to throw away many wrong values for k_r. The adversary stores this information by keeping a mapping $\mathcal{T}$ between possible key values for the last subkey k_r and how many times they are suggested.

In the end, when all the pairs have been exhausted, the adversary takes the value with the highest counter value in the table $\mathcal{T}$: if the differential $\mathcal{D}$ has been chosen with a high enough

differential probability, then the correct value for that subkey should be retrieved. Depending on the block cipher $\mathcal{E}$, the key recovery attack stops here if we can deduce the master key k from k_r, or the adversary needs to finish the recovery by exhausting the remaining unknown key bits in k. Alternatively, he can also mount the same attack on all rounds but the last one.

We note that the generic algorithm that we have described implicitly considers that the adversary can attack efficiently one round of the cipher, namely the last one, but the same strategy could apply if the adversary knew any efficient algorithms to handle more than one round in the end. Also, everything could easily transpose to a chosen-ciphertext attack where the adversary recovers< the initial round key(s).

3.2.1.1 Efficiency evaluation

We note that the data complexity, i.e. the number of chosen plaintexts required in this attack is closely related to the quantity of information needed to distinguish a distribution from the uniform distribution. Moreover, the same strategy can be applied if the differential $\mathcal{D}$ holds with very low probability: in that case, the adversary needs to consider the reversed table $\mathcal{T}$ by examining the key values k_r with the lowest counters.

To perform a more precise evaluation of the data and time complexities of this attack, we first note that the adversary needs at least $1/p$ chosen-plaintext pairs if he wants at least one to verify the $(r-1)$-round differential holding with probability p. A rough estimation of the data complexity D by Lai, Massey and Murphy in [LMM91] gives $D = \mathcal{O}(1/p)$, but if we denote $\tilde{p}$ the average differential probability depending on the block size, the data complexity has been evaluated more precisely by Blondeau, Gérard and Tillich in [BGT11] as $\mathcal{O}(\frac{1}{p\ln(p/\tilde{p})-p})$. Moreover, since the targeted round key k_r is the last one, its right value should be suggested more often than the other. To handle this notion, we usually introduce the *signal-to-noise ratio*, which measures how a particular event occurs among many others. It is defined as the ratio of the probability $p_{correct}$ that the right value k_r is suggested and the probability $p_{incorrect}$ that it is not. We usually denote it:

$$S/N \stackrel{\text{def}}{:=} \frac{p_{correct}}{p_{incorrect}}. \tag{3.11}$$

These two probabilities are expressed in terms of parameters depending on the targeted cipher. First, l the number of bits of k_r involved in the last round that we are trying to recover. Then, we introduce the probability q that a random pair is indeed used as a starting point for the attack. This probability verifies $p \leq q$ since we have more pairs than only the right ones ($2^{-p} \geq 2^{-q}$), because we can sometimes directly discard the pair from its output difference: its form or its value may not be consistent with the output difference of the differential. We call those pairs the *wrong pairs*. Hence, the error probability that a consistent pair never suggests the correct value k_r equals $q - p \geq 0$. For all the D tested pairs, we thus expect $D(q-p)$ of them to be completely useless. By denoting σ the average number of suggested values for k_r for a single pair of D, we deduce that the probability that the correct k_r will not get suggested is:

$$p_{incorrect} = \frac{(q-p)\sigma + p(\sigma-1)}{2^l - 1} = \frac{q\sigma - p}{2^l - 1}. \tag{3.12}$$

Indeed, if the pair is a wrong pair (probability $q-p$) then *all* the values suggested (σ) are wrong, but if it is a right pair (probability p) then all but one values are not the correct ones,

as the correct value is suggested exactly once. Finally, there are a total of $2^l - 1$ wrong values among the total 2^l possible values for the l bits of k_r.

As for $p_{correct}$, the probability that a pair suggests the right value for k_r, $p_{correct} = p$, the probability that the pair verifies the fixed differential characteristic in the first $r-1$ rounds. All in all, the signal-to-noise ratio equals

$$S/N = \frac{p(2^l - 1)}{q\sigma - p}, \tag{3.13}$$

and is compared to 1 for the relevance of the differential attack with the chosen parameter values.

If $S/N = 1$, then the proportion of right pairs is the same as wrong pairs: therefore, no statistical advantage can be exploited to distinguish the correct value for k_r. On the other hand, if $S/N \neq 1$, then the adversary can take advantage of the statistical bias introduced by the differential. If $S/N \geq 1$, the correct guess lies in the suggested values with the highest counters in the table. If $S/N \leq 1$, it has been suggested less than the others, so that the adversary finds it in the lowest counters.

Obviously, the greater the distance $|S/N - 1|$ is for the selected values of the parameters (p, q, l, σ), the more successful the attack. An extreme case occurs when $S/N = 0$, which are called *impossible differential attacks* (see Section 3.4.2). In theory, Lai, Massey and Murphy show in [LMM91] that the differential attack implicitly makes the *hypothesis of stochastic equivalence*, that we describe in the next section.

3.2.1.2 Improved variants

We note that the strategy exposed previously can be improved in several ways, according to the cipher. For instance, if the key size of k_r is slightly shorter than the one of the full secret key k, then Biham and Shamir propose in [BS93] a memoryless variant of the attack. The basic idea is to perform an exhaustive search on the remaining key bits as soon as a value for k_r is suggested. The table $\mathcal{T}$ becomes useless as the guessed key material is checked on the fly.

This attack is a chosen-plaintext attack as the adversary explicitly constructs pairs of inputs that verify a certain difference δ, but with an increase of the data complexity, this attack also applies in the known-plaintext model. In that case, the method is exactly the same, except that the adversary waits for plaintext pairs to create the wanted difference δ. Consequently, for a set of N plaintexts where we can construct $\binom{N}{2}$ pairs, the correct difference occurs with probability $1/2^n$, where n is the bit length of one plaintext. Therefore, the adversary needs to increase the number of pairs $\binom{N}{2}/2^n$ that verify the input difference δ for the attack to be successful.

There are several other improvements over this method, but we now consider a natural question: how to construct a block cipher resistant to this class of attacks?

3.2.2 Resistance against differential cryptanalysis

Since the cryptanalysis of the DES, the National Institute of Standards and Technology (NIST) announced in 1997 an international competition to find a new standard for encryption called

Advanced Encryption Standard (AES). The resistance against differential cryptanalysis has been one of the main motivation in the design of the winning algorithm: `Rijndael`. Its authors Joan Daemen and Vincent Rijmen describe for instance in [DR02] how they manage to construct a block cipher resisting to both linear and differential cryptanalysis. We briefly recall their idea.

As presented before, to mount a differential attack, the adversary needs to find a differential that holds with high probability. The methodology starts by finding a high-probability differential characteristic, which ensures a high-probability differential thanks to Theorem 3.2. Consequently, if one can find a way to upper bound the probability of *all* differential characteristics, one could directly deduce the worst impact of any differential attacks on the block cipher. Indeed, in the security proofs of block ciphers against differential cryptanalysis we usually conjecture that the high probability differentials have only one high probability differential characteristic.

In the design of `Rijndael`, Daemen and Rijmen propose a byte-oriented block cipher based on a Substitution-Permutation Network (SPN) such that any 4-round differential characteristic has a probability at most $2^{-6\times 25} = 2^{-150}$. The inner structure of the permutation ensures a very fast and complete diffusion after only two rounds. This means that if a single bit difference is introduced in the input of the round function, only two iterations are required to make the two outputs completely different. The theoretical point that justifies this diffusion relies on coding theory: the internal state fits into a square matrix over the finite field $\mathrm{GF}(2^8)$ and is multiplied by a Maximum Distance Separable (MDS) matrix to ensure maximal diffusion. The differential properties of this component allows to lower bound the number of active S-Boxes in the cipher, e.g. 25 active S-Boxes for 4 rounds. Additionally, the chosen S-Box has differential properties which provide the maximal differential probability for any r-round differential characteristic.

The block cipher accepts different key sizes, and for each of them, the given bound proves that all differential attacks have a running time greater than the trivial exhaustive search for the secret key. For instance, the variant `AES-128`, which uses 128-bit keys has 10 rounds, such that the probability p of any differential attack is upper bounded by $p \leq 2^{-300}$, which is more than enough to ensure the security up to 2^{128}. Obviously, this strategy does *not* provide absolute security, but only resistance to attacks in this class of differential cryptanalysis.

Additionally in [DR06b, DR06a], the same authors present a strong analysis of two rounds of their cipher and derive precise bounds for differential probabilities. Their analysis has then been refined by Keliher and Sui in [KS07] where it is shown that the maximal expected differential probability (MEDP) that we can reach for the AES S-Box is exactly 53×2^{-34}. Similar analyses have been conducted for 4-round AES, as in [DLP+09, RTV13].

3.3 Markov Ciphers

In a seminal work, Lai, Massey and Murphy in [LMM91] revisit the differential cryptanalysis and introduce the class of *Markov ciphers*. We recall here their main contributions. We denote the difference in variable x by Δx. In the following the probabilities are taken over the assumed uniform distribution of the key space. We suppose all the subkeys to be uniformly distributed in the the key space, and are independent for one another. While this is not the case in practice

for block ciphers, we detail this hypothesis in the end of this section.

Definition 3.9 (Markov Cipher, [LMM91]). A *Markov cipher* is an iterated cipher for which the average difference propagation probability over one round is independent of the input, i.e. for all α and β

$$\mathbf{Pr}\left(\Delta s_{n+1} = \beta | \Delta s_n = \alpha \cap s_n = \gamma\right) \tag{3.14}$$

is independent of γ. Stated otherwise, an r-round iterated block cipher is a *Markov cipher* if the sequence of output differences after each of the r rounds $(\Delta_0, \Delta_1, \ldots, \Delta_r)$ seen as discrete random variables forms a Markov chain, that is:

$$\mathbf{Pr}\left(\Delta_r = \omega_r | \Delta_{r-1} = \omega_{r-1} \cap \cdots \cap \Delta_0 = \omega_0\right) = \mathbf{Pr}\left(\Delta_r = \omega_r | \Delta_{r-1} = \omega_{r-1}\right). \tag{3.15}$$

The motivation behind the introduction of this class of ciphers is to provide a more precise evaluation of differential probabilities for a theoretical analysis of differential cryptanalysis. In particular, the notion of independence of the round keys allows to compute the exact probability of a differential characteristic.

Theorem 3.1. (Differential probabilities) *Let F be a Markov cipher with round function f, and $\delta_0 \to \cdots \to \delta_r$ a r-round differential characteristic in F. Then, we can compute the probability of the differential characteristic by:*

$$DP^F(\delta_0 \to \cdots \to \delta_r) = \prod_{i=0}^{r-1} DP^f(\delta_i \to \delta_{i+1}), \tag{3.16}$$

and the probability of the differential $\delta_0 \to \delta_r$ by:

$$DP^F(\delta_0 \to \delta_r) = \sum_{\delta_1, \ldots, \delta_{r-1}} \prod_{i=0}^{r-1} DP^f(\delta_i \to \delta_{i+1}). \tag{3.17}$$

Theorem 3.2. *If we assume the independence of the round keys, a key-alternating cipher (Definition 3.5) is a Markov cipher.*

To prove this theorem, we use the following result in probability theory.

Proposition 3.3. *Let A, B and C three possible events. We have:*

$$\mathbf{Pr}\left(A \cap B | C\right) = \mathbf{Pr}\left(A | B \cap C\right) \times \mathbf{Pr}\left(B | C\right). \tag{3.18}$$

Proof. From the definition of the conditional probability $\mathbf{Pr}(A|B) = \frac{\mathbf{Pr}(A \cap B)}{\mathbf{Pr}(B)}$, we can write:

$$\begin{aligned}
\mathbf{Pr}\left(A \cap B | C\right) &= \frac{\mathbf{Pr}\left(A \cap B \cap C\right)}{\mathbf{Pr}\left(C\right)} \\
&= \frac{\mathbf{Pr}\left(A | B \cap C\right) \times \mathbf{Pr}\left(B \cap C\right)}{\mathbf{Pr}\left(C\right)} \\
&= \frac{\mathbf{Pr}\left(A | B \cap C\right) \times \mathbf{Pr}\left(B | C\right) \times \mathbf{Pr}\left(C\right)}{\mathbf{Pr}\left(C\right)} \\
&= \mathbf{Pr}\left(A | B \cap C\right) \times \mathbf{Pr}\left(B | C\right),
\end{aligned}$$

which concludes the proof. ■

Proof. (of Theorem 3.2) Let F be a key-alternating cipher with round function f. From Definition 3.5, we have $f(s_n \oplus k_n) = s_{n+1}$, where s_n is the current m-bit state input, k_n the m-bit round key input and s_{n+1} the m-bit output state. We denote $x_n = s_n \oplus k_n$.

To prove that F is a Markov cipher, it is sufficient to show that

$$\mathbf{Pr}\left(\Delta s_{n+1} = \beta | \Delta s_n = \alpha \cap s_n = \gamma\right) \tag{3.19}$$

is independent of the input value s_n; that is, given (α, β), the probability is constant and independent of γ. Due to the linear key addition around the $\oplus$ operation, we have $\Delta s_n = \Delta x_n$. Hence:

$$\begin{aligned}
&\mathbf{Pr}\left(\Delta s_{n+1} = \beta | \Delta s_n = \alpha \cap s_n = \gamma\right) \\
&= \sum_x \mathbf{Pr}\left(\Delta s_{n+1} = \beta \cap x_n = x | \Delta x_n = \alpha \cap s_n = \gamma\right) \\
&= \sum_x \mathbf{Pr}\left(\Delta s_{n+1} = \beta | x_n = x \cap \Delta x_n = \alpha \cap s_n = \gamma\right) \mathbf{Pr}\left(x_n = x | \Delta x_n = \alpha \cap s_n = \gamma\right),
\end{aligned}$$

from the previous Theorem 3.3. Then, we observe that the second term of the product $\mathbf{Pr}\left(x_n = x | \Delta x_n = \alpha \cap s_n = \gamma\right)$ is actually equal to $\mathbf{Pr}\left(k_n = x \oplus \gamma\right)$. Finally, the first term reduces to $\mathbf{Pr}\left(\Delta s_{n+1} = \beta | x_n = x \cap \Delta x_n = \alpha\right)$ since the two conditions $x_n = x$ and $\Delta x_n = \alpha$ actually set the two inputs of f. Consequently, we have:

$$\mathbf{Pr}\left(\Delta s_{n+1} = \beta | \Delta s_n = \alpha \cap s_n = \gamma\right) = 2^{-m} \cdot \sum_x \mathbf{Pr}\left(\Delta s_{n+1} = \beta | x_n = x \cap \Delta x_n = \alpha\right),$$

because we make the assumption that the round keys variable are uniformly distributed. Finally, the probability (3.19) is independent of γ and F is Markov. ■

Theorem 3.4. AES *with independent round keys is a Markov cipher.*

Proof. AES is a key-alternating cipher, so from Theorem 3.2, if we assume that the round keys are independent and not all generated from a secret key through a key scheduling algorithm, then AES is a Markov cipher. ■

Remark. Theoretically, the majority of block ciphers does not conform to the hypothesis of independent round-keys since they are usually derived from a single master key k through a key schedule algorithm. Still, in practice, this assumption makes sense unless the keys from k are derived in a very basic way, and the practical evaluations confirm it [BS93]. Consequently, one can assume that DES and AES are Markov ciphers.

Remark. Additionnaly, it is also of interest to study the behaviour of reduced-round AES when the key is fixed. This has been done for instance by Daemen and Rijmen in [DR06b, DR06a, DR07, DR09], and they show that there exists some particular characteristics that reach a differential probability way above the maximal expected differential probability, averaged on all the keys.

In a differential attack, the adversary usually interacts with a black box outputting ciphertexts encrypted with a *single* secret key. But the offline phase of the differential cryptanalysis which finds a high-probability differential makes the choice of the differential based on an average behavior over all keys uniformly distributed. For the differential attack to work, Lai, Massey and Murphy show in [LMM91] that the adversary implicitly makes the following assumption (Property 3.5) that ensures all keys to behave the same way.

Property 3.5 (Hypothesis of stochastic equivalence, [LMM91]). *For virtually all high-probability r-round differentials $\delta \to \Delta$,*

$$\mathbf{Pr}_{\mathcal{P},\mathcal{K}}\left(\Delta_r = \Delta | \Delta_0 = \delta\right) \approx \mathbf{Pr}_{\mathcal{P}}\left(\Delta_r = \Delta | \Delta_0 = \delta \cap k = z\right)$$

holds for a substantial fraction of the key values z, where $\mathbf{Pr}_{\mathcal{P},\mathcal{K}}$ averages on both the uniform distributions of input space $\mathcal{P}$ and key space $\mathcal{K}$, whereas $\mathbf{Pr}_{\mathcal{P}}$ averages only on the input space.

Remark. This assumption holds for the majority of block ciphers designed for cryptographical applications, but we may construct an example such that Property 3.5 does not hold. Anyway, such a block cipher probably should not be used in any actual cryptosystem.

3.4 Other forms of differential cryptanalysis

3.4.1 Truncated differential cryptanalysis

Another form of differential cryptanalysis has been introduced by Lars Knudsen in [Knu94] and is called *truncated differential cryptanalysis*. The technique is a generalization of the differential cryptanalysis framework in the sense that the adversary follows differences that are *partially* determined, rather than fully determined for a classical differential attack. The attack actually considers the *presence* of differences between two variables of the primitive.

Formally, we define the truncation as a non-injective function $\Delta \to \bar{\Delta}$, which associates the actual difference Δ to

$$\bar{\Delta} = \begin{cases} 0 \text{ if } \Delta = 0 \\ 1 \text{ if } \Delta \neq 0. \end{cases}$$

Therefore, a difference is reduced to a 1-bit information. Depending on the structure of the primitive, we may want to decompose the difference Δ into a vector of differences $(\Delta^i)_i$ such that it makes sense for the cipher. For instance, we usually consider the 16 differences of the AES state separately, such that an actual difference $\Delta \in \{0,1\}^{128}$ equals $\Delta = (\Delta^i)_{i=0,\ldots,15}, \Delta_i \in \{0,1\}^8$, and is truncated as $\bar{\Delta} = (\bar{\Delta}^i)_{i=0,\ldots,15}$ with $\Delta_i \in \{0,1\}$. Here, the truncated difference $\bar{\Delta}$ is reduced to 16 bits.

In general, truncated differential attacks apply more efficiently on block ciphers where the round function considers parts of the state together, like bytes or nibbles, rather than bit by bit. This way, one can represent a truncated difference in a single part, say a byte, and trace its evolution throughout the round function applications. Block ciphers successfully attacked with

truncated cryptanalysis include for instance Skipjack [Ski98], where the 31-round impossible differential from [BBS05] relies on two truncated differentials, and the authors in [KRW99] show that there exist a 24-round truncated differential that holds with probability 1.

In the next chapters of this document, we use truncated cryptanalysis on several byte-oriented primitives like the AES block cipher (Chapter 7), and the hash functions Grøstl (Chapter 8) and ECHO (Chapter 9).

3.4.2 Impossible differential cryptanalysis

Another form of differential cryptanalysis is called impossible differential cryptanalysis and uses the differentials that hold with probability zero (see Definition 3.4). This is the complete opposite of the previously described strategy that relies on high-probability differentials: here, the adversary takes advantage on impossible events.

In detail, the attack methodology starts by finding an r-round differential $\delta \rightarrow \Delta$ such that $DP^F(\delta \rightarrow \Delta) = 0$. Then, he uses it in the online phase by querying chosen-plaintext pairs with input difference δ, and checks whether the differential has been followed in the first r rounds. This test is performed by guessing key material in the last round and by partially decrypting the resulting ciphertexts. If the key guess suggests that the impossible differential has been followed, then the adversary can confidently discard the key guesses and try another one. By removing all wrong key guesses, the only valid key remains.

3.4.2.1 Applications

The impossible differential attack has been first used by Lars Knudsen on the 128-bit block cipher DEAL [Knu98], which has been submitted to the AES competition. At the Rump Session of CRYPTO 1998, Eli Biham, Alex Biryukov and Adi Shamir introduce the term *impossible differential* and show a method to construct an impossible differential on $2k$ rounds from two k-round differentials holding with probability 1. The first one transforms δ into δ_0 in the forward direction, while the second one transforms Δ into δ_1 in the backward direction. If the two differences δ_0 and δ_1 present incompatibility, then the $2k$-round differential $\delta \rightarrow \Delta$ is an impossible differential. This method has been called *miss-in-the-middle*.

The same authors also present an application of the *miss-in-the-middle* technique to IDEA [LM90], which leads to an attack on 3.5 rounds of the cipher (out of 8.5) with 2^{35} chosen plaintexts and 2^{64} steps, and another attack on 4.5 rounds with 2^{64} plaintexts and 2^{112} steps. This is no longer the best available cryptanalysis of reduced variants of IDEA. Additionally, they apply the technique to Skipjack [Ski98] which is an unbalanced Feistel network developed by the NSA and published in 1998. The result in [BBS99, BBS05] shows an impossible differential attack on 31 rounds (out of 32) of Skipjack to recover the 80-bit secret key, and later, new attacks on this cipher complete the impossible cryptanalysis, for instance [KRW99, RW03].

This strategy has been applied to many block ciphers, and it has been the best attack against 7-round AES-128 for some time, until the improved meet-in-the-middle attack from [DFJ13] (see Chapter 5). Previous impossible attacks [BA08, LDKK08, MDRMH10] use truncated differences (see Section 3.4.1) to show a 4-round impossible differential and extend it to a

7-round attack (see Section 4.4.4.1 for a description of this attack).

3.4.2.2 Resistance against impossible differential cryptanalysis

In a previous paragraph, we say that resisting to differential attack can be achieved by upper bounding the maximal differential probability of the differentials. However, if a differential has a zero-probability, it is subject to impossible differential cryptanalysis: hence, we also need to lower bound the differential probabilities of the differentials. Therefore, if we can lower bound all differential probabilities of the differentials for any number of rounds of a given cipher by $p_{min} > 0$, then we can guarantee that no impossible differential exists. Contrary to upper bounding the probability, this task is much more difficult, as we not only need to bound the probability of differential characteristics, but the probability of *differentials*, which are less convenient to handle.

Provable security against differential cryptanalysis has been a very investigated line of research and still remains of importance matters for designers. In the next section, we recall a particular class of ciphers that allows to deduce bounds on the differential properties.

3.4.3 Boomerang attack

Another differential cryptanalysis technique is due to David Wagner and has been published in 1998 in [Wag99]: it is called the *boomerang attack*. In the original differential attack, one differential is used to detect a statistical correlation between inputs and outputs. Then, the impossible differential technique introduces the miss-in-the-middle technique to combine two differentials with probability 1 to get a longer impossible differential. In the boomerang strategy, Wagner also proposes to combine two differentials to mount a chosen-plaintext and (adaptive) chosen-ciphertext attack. The powerful consequence of this technique is the ability to attack a cipher in a differential manner even when there is no high-probability differential on the whole cipher, but only on parts of it.

The strategy starts by breaking the cipher $\mathcal{E}$ into two parts $\mathcal{E}_0$ and $\mathcal{E}_1$, with $\mathcal{E} = \mathcal{E}_1 \circ L \circ \mathcal{E}_0$ and an optional linear of affine mapping L (see Figure 3.6). Then, the adversary finds one differential on each $\mathcal{E}_0$ and $\mathcal{E}_1$: since $\mathcal{E}_0$ and $\mathcal{E}_1$ are basically half the size of $\mathcal{E}$, it should be easier to find high-probability differentials on them. Then, to connect the two parts and deduce an attack of the full $\mathcal{E}$, he tries to construct a quartet element (X_1, X_2, X_3, X_4) that verifies a certain relation in the middle on the primitive. Here, the pair of values of the original differential attack is replaced by a quartet in the boomerang attack.

In the sequel, we assume we know one differential $\Delta \to \Delta^*$ in $\mathcal{E}_0$ and an other differential $\nabla \to \nabla^*$ in $\mathcal{E}_1^{-1}$. Let p and q be the two respective probabilities. Then, the adversary applies the following steps:

1. Pick a random pair of inputs (X_1, X_2) such that the difference $X_1 \oplus X_2 = \Delta$.
2. Ask the encryption of the two inputs and store their respective outputs in Y_1 and Y_2. With probability p, the differential in the first half $\mathcal{E}_0$ is verified so that:

$$\mathbf{Pr}\Big(\mathcal{E}_0(X_1) \oplus \mathcal{E}_0(X_2) = \Delta^*\Big) = p.$$

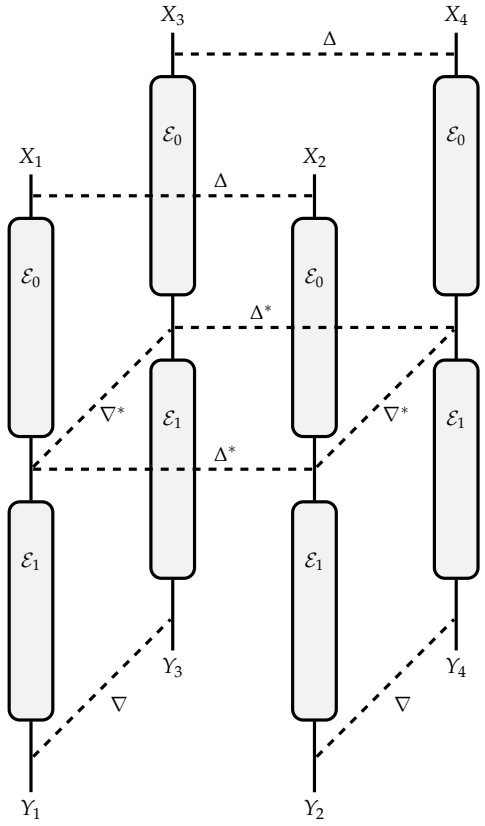


Figure 3.6: The boomerang attack: the adversary tries to find a quartet (X_1, X_2, X_3, X_4) that verifies a certain relation half-way through the cipher. This figure assumes that the affine layer L in the middle is the identity.

3. With the received values Y_1 and Y_2 compute the translation by ∇ to get: $Y_3 = Y_1 \oplus \nabla$ and $Y_4 = Y_2 \oplus \nabla$.
4. Ask the decryption of Y_3 and Y_4 and store their respective inputs in X_3 and X_4. With probability q, each of the two differentials in $\mathcal{E}_1^{-1}$ are verified independently; that is:

$$\mathbf{Pr}\left(\mathcal{E}_1^{-1}(Y_1) \oplus \mathcal{E}_1^{-1}(Y_3) = \mathcal{E}_1^{-1}(Y_2) \oplus \mathcal{E}_1^{-1}(Y_4) = \nabla^* \middle| Y_1 \oplus Y_3 = Y_2 \oplus Y_4 = \nabla\right) = q^2.$$

5. Check whether the difference $X_3 \oplus X_4 = \Delta$. If it does, then the differential holds and (X_1, X_2, X_3, X_4) is a valid boomerang quartet. Together with the previous probabilities, this occurs when the differential $\Delta^* \to \Delta$ holds in $\mathcal{E}_0^{-1}$ with (X_3, X_4).

In the end, the total probability that the quartet is a valid boomerang quartet is p^2q^2, since the two differentials in $\mathcal{E}_0$ and $\mathcal{E}_1$ need to be verified twice each. Consequently, if we can find two differentials with probabilities p and q that cover the full cipher and verify $(pq)^{-2} < 2^n$ where n is the bit size of the key, then this gives an attack faster than exhaustive search.

We note that the same strategy applies when one considers an additional linear or affine transformation L in the middle on the cipher: the linear relation still holds in that case. Moreover, we remark that the previous computations assume the differentials to be regular differentials, and not truncated differentials. Indeed, for a regular differential $\mathcal{D}$, the differential probability p is the same whether we consider $\mathcal{D}$ or $\mathcal{D}^{-1}$ in the other direction. This is not

the case for truncated differentials, where we would need to consider a probability $\overrightarrow{p}$ for the forward direction, and $\overleftarrow{p}$ for the backward direction.

3.4.3.1 Improvements

There have been several improvements over this method. A major one is the *amplified boomerang* attack published in [KKS00], and also described under the name *rectangle attack* in [BDK01], which turns the boomerang attack into a more classical chosen-plaintext attack. The key principle is to perform requests *only* to the encryption oracle, whereas the original boomerang attack queries both the encryption and decryption oracles.

In the original attack, the adversary performs $O\left((pq)^{-2}\right)$ chosen-plaintext and chosen-ciphertext queries to the oracles, whereas the amplified attack uses an amplificated event in the middle, which reduces the data complexity to $O\left((pq)^{-1} \cdot 2^{n/2}\right)$ chosen-plaintext queries. This drop comes from the linear relation verified by a quartet in the middle.

Namely, the adversary prepares a set of D pairs of plaintexts with chosen input difference Δ, so that a random pair from D verifies the first differential $\Delta \to \Delta^*$ in $\mathcal{E}_0$ with probability p. Consequently, there are about $2 \times \binom{Dp}{2}$ pairs of pairs $((X_1, X_2), (X_3, X_4))$ such that the two pairs verify the differential in $\mathcal{E}_0$. Since $X_1 \oplus X_2 = X_3 \oplus X_4 = \Delta^*$, the linear relation in the middle is verified as soon as $X_1 \oplus X_3 = \nabla^*$, which occurs with probability 2^{-n}. Indeed, the last equality $X_2 \oplus X_4 = \nabla^*$ is automatically verified by linearity. Therefore, there are about $2 \times \binom{Dp}{2} \times 2^{-n}$ pairs of pairs with the correct difference ∇^* for the two pairs at the input of the second differential $\nabla^* \to \nabla$ in $\mathcal{E}_1$. In the end, we expect

$$2 \times \binom{Dp}{2} \times 2^{-n} \times q^2 \approx D^2 \frac{(pq)^2}{2^n}$$

right quartets for the boomerang attack.

This gives a distinguishing attack for the block cipher $\mathcal{E}$ when we are able to produce more right quartets for the block cipher than for a random permutation. In that latter case, we expect about $D^2/2^{2n}$ quartets since we have two pairs each constrained on a particular n-bit output difference ∇. So, if

$$\frac{D^2(pq)^2}{2^n} > \frac{D^2}{2^{2n}},$$

we can attack $\mathcal{E}$, which happens as soon as $pq > 2^{-n/2}$.

Additionally, in the original description of the boomerang attack [Wag99], Wagner notices that the middle differences Δ^* and ∇^* actually does not need to be specified: the success of the attack is independent of their values. Therefore, a right quartet can be obtained as long as the middle linear relations hold:

$$\begin{cases} \mathcal{E}_0(X_1) \oplus \mathcal{E}_0(X_2) = \mathcal{E}_0(X_3) \oplus \mathcal{E}_0(X_4) \\ \mathcal{E}_0(X_1) \oplus \mathcal{E}_0(X_3) = \mathcal{E}_0(X_2) \oplus \mathcal{E}_0(X_4). \end{cases}$$

This remark directly affects the probability of the differentials in $\mathcal{E}_0$ and $\mathcal{E}_1$ as we can consider sums of differential probabilities: p and q become respectively $\widehat{p}$ and $\widehat{q}$, sometimes

called *amplified probabilities*:

$$\widehat{p} = \sqrt{\sum_{\Delta^*} \left(DP^{\mathcal{E}_0}(\Delta \to \Delta^*)\right)^2} \qquad \text{and} \qquad \widehat{q} = \sqrt{\sum_{\nabla^*} \left(DP^{\mathcal{E}_1}(\nabla^* \to \nabla)\right)^2}. \tag{3.20}$$

With this remark, the data complexity of the original boomerang attack becomes $O\left((\widehat{p}\,\widehat{q})^{-2}\right)$, and the data complexity of the amplified boomerang attack $O\left((\widehat{p}\,\widehat{q})^{-1} \times 2^{-n}\right)$. We summarize the data complexities and conditions on p and q to get the attacks in Table 3.1.

Type	Data	Condition	Reference
Boomerang	$(pq)^{-2}$	$(pq)^{-2} < 2^n$	[Wag99]
Amplified boomerang	$(pq)^{-1} \times 2^{-n}$	$pq > 2^{-n/2}$	[KKS00, BDK01]

Table 3.1: Complexity of boomerang and amplified boomerang attacks: p and q are the differential probabilities of the differential in $\mathcal{E}_0$ and $\mathcal{E}_1$, respectively, and can be replaced by $\widehat{p}$ and $\widehat{q}$ defined in Equation 3.20.

3.4.3.2 Applications

There have been several applications of the boomerang attack to different block ciphers in the standard model or in the related-key model, and also an extension of the technique to the hash function domain.

The block cipher COCONUT98 [Vau98, Vau03] by Serge Vaudenay has been the first cryptanalyzed by the boomerang attack. In the original boomerang paper [Wag99], Wagner describes a method using 2^{16} adaptively chosen plaintexts and ciphertexts to break this cipher in about 2^{38} computations with an overwhelming probability of success. This result is possible despite the "proof" of resistance of COCONUT98 against differential cryptanalysis based on decorrelation theory [Vau03] since we are using quartets, and not pairs of messages.

Several other block ciphers have been broken or partially broken by the boomerang attack and its variants, as non-exhaustively: AES in [GL08], MARS and Serpent in [KKS00], an improvement for Serpent in [BDK01] and [BDK02], SC2000 in [BDK02], full KASUMI in [BDK05b] which is used in 3GPP for mobile communications, PRINCE in [JNP+13].

3.4.4 Related-key attacks

In [Bih93, Bih94], Eli Biham introduces the concept of related-key attacks. In this model, the adversary is given access to a blackbox construction to which he can query a particular message under several keys related in some ways. For instance, he could get the result of the encryption of message m by the secret key k he is trying to recover and a translated key $k \oplus \delta$ for some known (or chosen) difference δ. In practice, this model is obviously less threatening than the standard model, but it can still be quite devastating when the keys used in a protocol are not properly generated, when the adversary can encrypt under a subset of unknown keys, etc.

The most notorious example is probably the case of the Wired Equivalent Privacy (WEP) algorithm used in the encryption of WiFi wireless network communications. The underlying

encryption algorithm is the widely used RC4 stream cipher algorithm: a well-known security requirement for stream ciphers is to never use the same keystream twice. Indeed, encrypting two messages m and m' with the *same* key k produces two ciphertexts c and c' respectively, such that the input and the output differences are equal: $m \oplus m' = c \oplus c'$. This property is true for any stream cipher, and if the mistake is done, we can run basic statistical tests as exposed in the introduction. To prevent this from happening, the WEP includes a 24-bit initialization vector (IV) to randomize the behavior of the cipher, and this short value is concatenated with the WEP key. By the birthday paradox, we expect that for every set of 2^{12} encrypted packets, at least two share the same IV and thus the full two keys are equal. We can thus run devastating attacks like [TWP07] at that point to recover the plaintexts and eventually the WEP key. This is an example of a design practically attacked by related keys, but this could be prevented as done for instance in the WPA by refreshing the keys used in the protocol, or by using a block cipher like AES.

Theoretically, the relation in the keys can be anything from a constant difference between two subkeys, a set of keys sharing the same truncated differences, a slid pair in the key scheduling algorithm, etc. In more detail, a slid pair of keys for a step $\mathcal{KS}$ of the key schedule would consist in the pair (k_0, k'_0) such that:

$$\begin{aligned} k_0 \xrightarrow{\mathcal{KS}} k_1 \xrightarrow{\mathcal{KS}} k_2 \xrightarrow{\mathcal{KS}} \dots \xrightarrow{\mathcal{KS}} k_n \\ k'_0 \xrightarrow{\mathcal{KS}} k'_1 \xrightarrow{\mathcal{KS}} \dots \xrightarrow{\mathcal{KS}} k'_{n-1} \xrightarrow{\mathcal{KS}} k'_n, \end{aligned}$$

where $k'_i = k_{i+1}$, for $i \in \{0, \dots, n-1\}$. We note that this observation is only possible when the $\mathcal{KS}$ function is the same at each step of the key derivation, which is usually not the case in modern block ciphers where we usually introduce different round-dependent constants at each application.

Formally, when the adversary in the standard model can only query the oracle using the secret key k, in the related-key model, he can now query under keys $f_1(k), \dots, f_t(k)$ for some relations f_i chosen by the adversary. At first, the concept of related keys has been introduced to analyze the security provided by the key scheduling algorithms of block ciphers. Indeed, all the subkeys used in a block cipher are generally derived from a single master secret key, and for efficiency matters, we want the derivation $\mathcal{KS}$ to be fast, which can make a quite weak key scheduling algorithm.

In Chapter 6, we revisit this approach while targeting SPN ciphers, and in particular the AES-128. Indeed, the key schedule algorithm of AES is somewhat acknowledged to be the weakest point of the cipher, so that an adversary can take advantage of it in the related-key model. On the larger variants of the AES, there are two known related-key attacks of the full AES-192 and AES-256, where we allow the adversary to introduce (and control) differences in the (sub)keys. These results have been published in two major works, in [BK09] by Alex Biryukov and Dmitry Khovratovich, and in [BKN09] by Alex Biryukov, Dmitry Khovratovich and Ivica Nikolić. We note that these two results have been later renamed *related-subkey attacks* in [BDK+10] since the relation in the keys is verified on a particular subkey, and not on the original master key at the input of the key scheduling algorithm.

In comparison to the standard model, it is hard to give a formalization of the related-key model to provide a formal definition of the security against related-key attacks. In [BK03],

Bellare and Kohno give an intuition of the difficulty of the problem: they show that there exist key relations for which we can find trivial attacks for any block ciphers. While we could investigate a way to somehow restrict the set of possible relations in the key, we cannot completely ignore the definition of the targeted cipher, as the key relation may depend on the cipher definition, as in [BKN09, BK09].

Related-key boomerang attacks

We have mentioned the boomerang attack in the previous section, which considers the encryption decryption of a four-message structure under a single secret key. The concept can easily be generalized to related keys where the adversary encrypts the four messages under two, four or more related subkeys (see Figure 3.7). The concept has first been introduced by

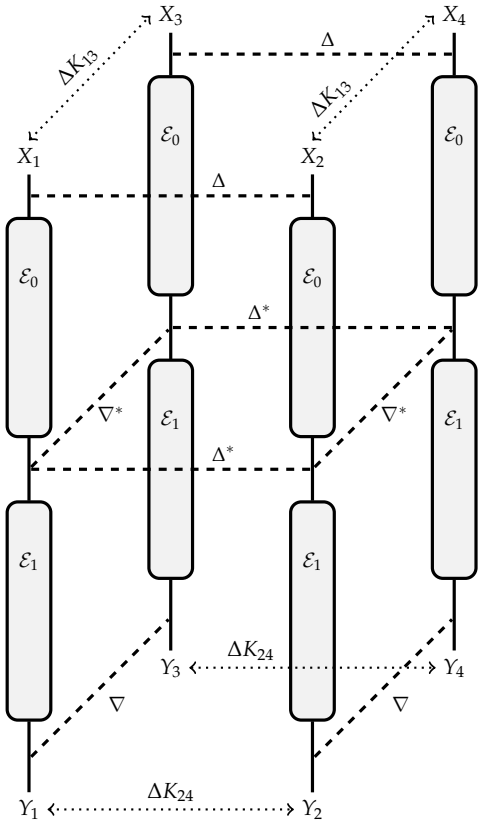


Figure 3.7: The related-key boomerang attack.

Biham, Dunkelman and Keller in [BDK05a], and then applied to several ciphers like KASUMI in [BDK05b], AES in [GL08] and later on the largest versions of the AES as we already mentioned in [BKN09, BK09].

The methodology to find an input structure for the related-key boomerang attack is closely related to the original boomerang attack: we do not discuss it in this document and refer the reader to the mentioned works.

3.5 Hash functions

In this section, we give an overview of some cryptanalytic techniques used for attacks on hash functions. We note that there is a vast literature on the subject, which would probably need a book by itself to be properly summarized. Instead, we have chosen to discuss *some* general ideas (Section 3.5.1) and the recent rebound strategy (Section 3.5.2).

3.5.1 Generalities

Since most of the current hash functions are based on block ciphers by using a mode like Davies-Meyer (DM) or Matyas-Meyer-Oseas (MMO), we are tempted to reuse the previous ideas on block ciphers and apply them to hash functions. While this can be done, there are several differences that help the adversary: first of all, there is no secret in a hash function, which means that we either know or control the key input to the underlying block cipher. The adversary can exploit this by choosing the best possible key input or select the one that suits best his purpose.

A general idea that we have developed for block ciphers is the concept of differential characteristic: almost all the attacks against hash functions start by devising a high-probability differential characteristic. If the hash function or its compression function uses an internal block cipher, we can use the freedom in the key bits to select the characteristic with the highest differential probability as it is normally key-dependent.

Another consequence of the lack of secret parameter in the hash function is the total transparency of computations that goes with it. In particular, we do not need to *guess* key material to check whether an event has occurred during the computation, like the integral property for instance, we can directly check for the event and avoid extra computations if it does not hold (early-abort principle). Additionally, we have mentioned earlier the unicity distance for block cipher that gives an estimation of the quantity of information required to determined uniquely and without ambiguity the secret key. Here, this does not make sense as we need to verify a relation *only once*, e.g. find two inputs sharing the same image. All in all, while the cryptanalysis of block ciphers and hash functions use similar technique, the respective problems are completely different from one another, and one could say that all one can do when a secret is involved can also be done when none is.

In the case of hash functions, the attacker starts by finding a differential characteristic that holds with high probability. The probabilities are generally computed whether conditions are verified along the characteristic. For hash functions from the SHA family like `MD4`, `MD5`, `SHA-1`, these equations are generally bit conditions and can be enforced with various techniques like message modification techniques. This topic goes beyond the scope of this document, but we emphasize that the goal of the attacker here is to *partially* determine a set of bits during the hashing process. In the end, he tries to fulfill the remaining conditions by finding values for the unset bits.

3.5.2 Rebound attack

In this section, we give a high-level introduction to the rebound attack. It has been first proposed by Mendel, Rechberger, Schläffer and Søren S. Thomsen at FSE 2009 in [MRST09]. This attack is categorized in the class of differential attacks, as an optimized version specifically dedicated to hash functions. While differential cryptanalysis for block ciphers discussed in Section 3.2 aims at recovering the secret of the cryptosystem, here the point of the adversary is to contradict one of the requirements of the hash function, e.g. the collision or the preimage resistance.

In the rebound attack, the targeted function f is decomposed into three parts:

$$f = f_{fw} \circ f_{in} \circ f_{bw},$$

where f can either be a block cipher, a compression function or a permutation depending on the context (see also Figure 3.8). Then, the adversary tries to find high-probability differentials in f_{fw} for the forward part and in f_{bw} for the backward part. They are connected in the inbound part in f_{in}. Now, as we consider a secretless primitive or assuming the key is in the public domain, we can consume the freedom degree coming from the message and/or key input to fulfill the strong constraints at the merging point in f_{in}.

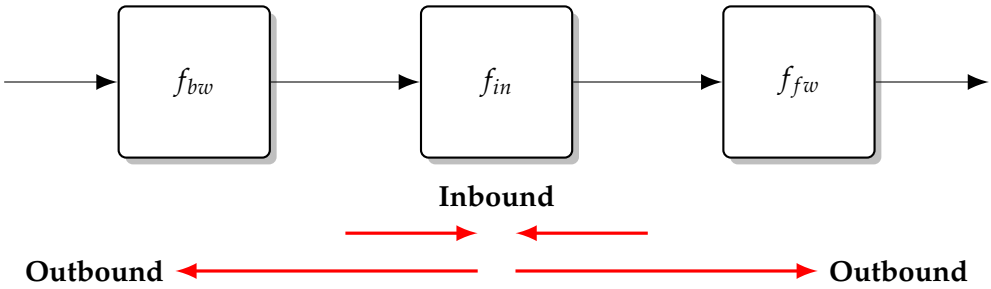


Figure 3.8: Overview of the rebound attack: we start in the middle in the inbound phase to end in the two outbound phases.

As in classical differential attacks on hash functions, we first need to construct a high-probability differential for this scenario, but we now allow the characteristic to have a low probability in the middle. Indeed, this part lies in the inbound phase where we start by consuming lots of freedom degrees to generate efficiently partial conforming pairs of messages. Consequently, we can rather concentrate our efforts on the outer parts so that the backward and forward outbound phases would be verified with the highest possible probabilities. However, it is to be noted that the purpose of the rebound attack is to delegate the efficient generation of the message pairs to the inbound phase where we can consume many freedom degrees and to exhaust them into the probabilistic outbound phases to filter out a right pair for the full characteristic. This mechanism therefore assumes that the inbound *can* generate enough starting points for the outbound phases, namely name the probability $p_{bw} \cdot p_{fw}$ of the two outbound phases verifies $p_{bw} \cdot p_{fw} > 1/N$, where N is the number of pairs the inbound can generate. This simply means that there is a non-negative amount of freedom degrees for the complete characteristic.

The rebound attack has been published at the very beginning of the SHA-3 competition and has first been applied in [MRST09] to reduced version of the `Whirlpool` and `Grøstl`

hash functions, both based on the AES. The analysis of Whirlpool has later be refined in [LMR+09] where the full compression function is attacked by a distinguishing algorithm in 2^{188} computations. For this attack, the rebound strategy applies once in the message and once in the key schedule, which allows to connect efficiently the two parts and construct efficiently a near-collision input to the compression function. This is possible in Whirlpool as the main permutation is almost the same as the permutation in the key schedule. In Chapter 8, we show how to increase the probability of the outbound phases to decrease the time complexity of this attack on Whirlpool.

In some cases, it is possible to extend the general framework of the rebound attack of Figure 3.8 by considering more than one inbound phase. Naturally, this assumes there are enough freedom degrees to do so, but we may isolate independent parts of the primitive, construct a differential characteristic matching the behavior of the rebound, and run the two or more inbounds. This strategy has been applied for instance to the LANE hash function in [MNPN+09] or in [JNPS11a] for ECHO as we describe in Chapter 9.

At first, the rebound attack has focused on the AES-like designs like AES, Grøstl and Whirlpool, probably because truncated differential characteristics can be handled easily for such designs and they apply perfectly to the rebound technique. However, similar ideas can be carried out on the completely different ARX designs like it has been done on Luffa in [KNPRS10] or on Skein in [KNR10].

Throughout this document, we revisit various aspects of the rebound technique and apply it to different primitives. In Chapter 7, we start by describing the basic application of one inbound and two inbound phases in a keyless AES settings with 7 rounds. The key is assumed to be known, and the rebound attack can find an input pair to the cipher more efficiently than for a random permutation. We present this technique for AES-like permutations in a very general context to apply it easily on larger functions like Grøstl or ECHO, and present all the published algorithms improving the capacity of the rebound. Finally, in the last Chapter 8, our goal is twofold: first, we detail how we can extend the inbound phase by one more round in some instances of AES-like permutations, and then we show how to increase the probability of the outbound phases to decrease the overall time complexity requires by the rebound attack.

CHAPTER 4

Description of the AES and cryptanalytic Results

Contents

4.1 The AES competition 57
4.2 Description of the AES block cipher 58
4.2.1 Key scheduling algorithms 59
4.2.2 Round function 61
4.2.3 The substitution layer 62
4.2.4 The permutation layer 64
4.3 AES-like permutations 66
4.4 Notable cryptanalytic results 67
4.4.1 Square attack 67
4.4.2 Improved square attack with partial sums 71
4.4.3 Collision attack 75
4.4.4 Impossible differential attack 79
4.4.5 Related-key attacks 84
4.4.6 Summary of all the attacks 86

In this section, we describe the structure of all the three variants of the AES. We start in Section 4.1 by recalling the history of the selection of the AES in the competition launched by the NIST. Then in Section 4.2, we precise the complete description of the AES-128, AES-192 and AES-256, which only differ by their key scheduling algorithms. Finally, we give some of the main cryptanalysis results that have been published against this cipher (Section 4.4).

4.1 The AES competition

On January 2, 1997, the National Institute of Standards and Technology of the United States (NIST) has announced a will to replace the current standard of encryption DES. Indeed, the 16 rounds of the DES has been analyzed successfully since the publication of the differential attack [BS93] and the linear attack [Mat94a]. Moreover, the NIST assesses that 56-bit keys are

no longer sufficient to thwart the growing threats of cryptanalysis and affordable computing power.

Rather than imposing the AES as a new publicly available algorithm designed by its own services like it has been done for its predecessor, the NIST calls for participations [AES97] under an open process where the interested parties are asked to propose their own block cipher for public evaluation. The submissions are requested to offer 128-bit blocks and key sizes of 128, 192 and 256 bits, at least. This process starts in early 1998 when the NIST chooses 15 candidates entering the first of the three rounds of the selection. In alphabetic order, the NIST selects the following submissions for public evaluation by the cryptographic community:

- CAST-256 (Adams et al.)
- CRYPTON (Lim)
- DEAL (Knudsen)
- DFC (Stern et al.)
- E2 (NTT)
- FROG (Georgoudis et al.)
- HPC (Schroeppel)
- LOKI97 (Brown)
- MAGENTA (Jacobson et al.)
- MARS (IBM)
- RC6 (Rivest et al.)
- Rijndael (Rijmen and Daemen)
- SAFER+ (Massey)
- Serpent (Anderson et al.)
- Twofish (Schneier).

Besides security, an important feature required by the NIST is good performances on major platforms: for those two reasons, the NIST reduces in August 1999 the possible candidates to only five so-called AES finalists: MARS, RC6, Rijndael, Serpent, and Twofish. Finally, in October 2000, the NIST has declared Rijndael as the winner of the competition and begins the process of standardization. To date, the Belgium block cipher designed by Joan Daemen and Vincent Rijmen is commonly referred to as AES and is the current standard of encryption since 2001 by FIPS PUB 197 [AES01].

4.2 Description of the AES block cipher

The Advanced Encryption Standard (AES) [AES01] is an iterated block cipher based on a Substitution-Permutation Network that can be instantiated using three different key sizes: 128, 192, and 256 bits. The block size for all the variants is the same and is 128-bit large. The authors Joan Daemen and Vincent Rijmen have reused their earlier design strategies of the SHARK [RDP+96] and SQUARE [DKR97] block ciphers, which was one of the few ciphers having keys larger than 128 bits.

This modern design breaks with the Feistel family by introducing a byte-oriented structure, with a more advanced mathematical background. Namely, the internal state of the primitive is seen as a 4×4 matrix of bytes[1] as values in the finite field $\mathrm{GF}(2^8)$, which is defined via the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ over $\mathrm{GF}(2)$. Therefore, any byte value in the interval $[0, 2^8 - 1]$ can be represented as the polynomial in $\mathrm{GF}(2^8)$: for instance, the byte 3 is represented by $x + 1$ and 0x42 as hexadecimal representation of 66 is represented by $x^6 + x$. During the encryption process that we describe in the next paragraphs, the operations on the

[1] We recall that one byte commonly equals 8 bits, and this is the case in this document.

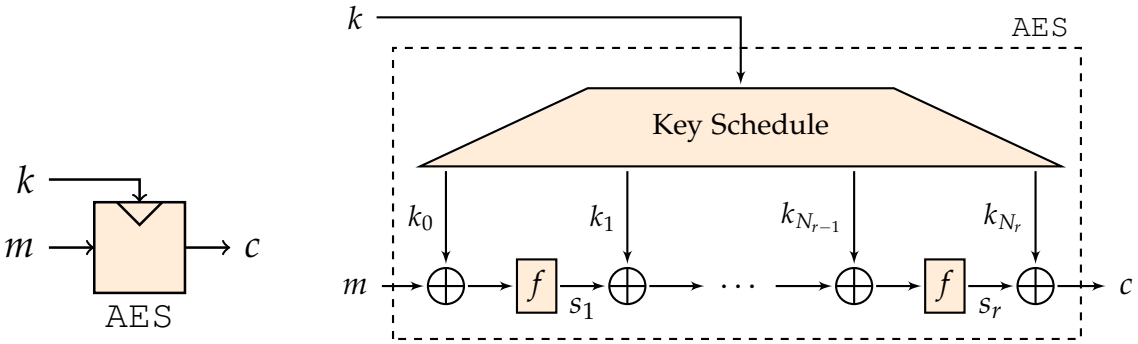
state matrix are performed with the field arithmetic. In the sequel, we use an ordering of the 16 bytes inside the matrix as mentioned on Figure 4.1, and we denote by $x[j]$ the j-th byte of this ordering for a given AES state x, or alternatively if needed $x[i,j]$ or $x_{i,j}$, $0 \leq i,j \leq 3$ the matrix element at index (i,j) in the state.

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

Figure 4.1: Ordering of the bytes in one AES state.

For a long message m, a block cipher mode transforms m into N chunks of same size as the block size, $m \to (m_0, \ldots, m_{N-1})$, and links their results to construct the encrypted version $\text{AES}_k(m)$ of m using the key k. The NIST suggests several modes depending on the scenario: six confidentiality modes [Dwo01a] (ECB, CBC, OFB, CFB, CTR, and XTS-AES), one authentication mode [Dwo01d] (CMAC), and five combined modes for confidentiality and authentication [Dwo01c, Dwo01b] (CCM, GCM, KW, KWP, and TKW). In the sequel, we ignore the mode of operation and concentrate on the primitive of encryption.

On Figure 4.2, we show a schematic view of the AES encryption process. Like any block cipher, it takes two inputs: the key k and a message m to encrypt, and outputs the corresponding ciphertext c encrypted under key k: $c = \text{AES}_k(m)$. In the next two sections, we describe the two main components: the key schedule (Section 4.2.1) and the round function f (Section 4.2.2).



(a) Black-box view. **(b)** Complete AES encryption.

Figure 4.2: Encryption with AES: the key k is fed to the key schedule to produce the subkeys, and the input message m is encrypted using the N_r rounds of the round function f. The ciphertext c corresponds to $c = \text{AES}_k(m)$.

4.2.1 Key scheduling algorithms

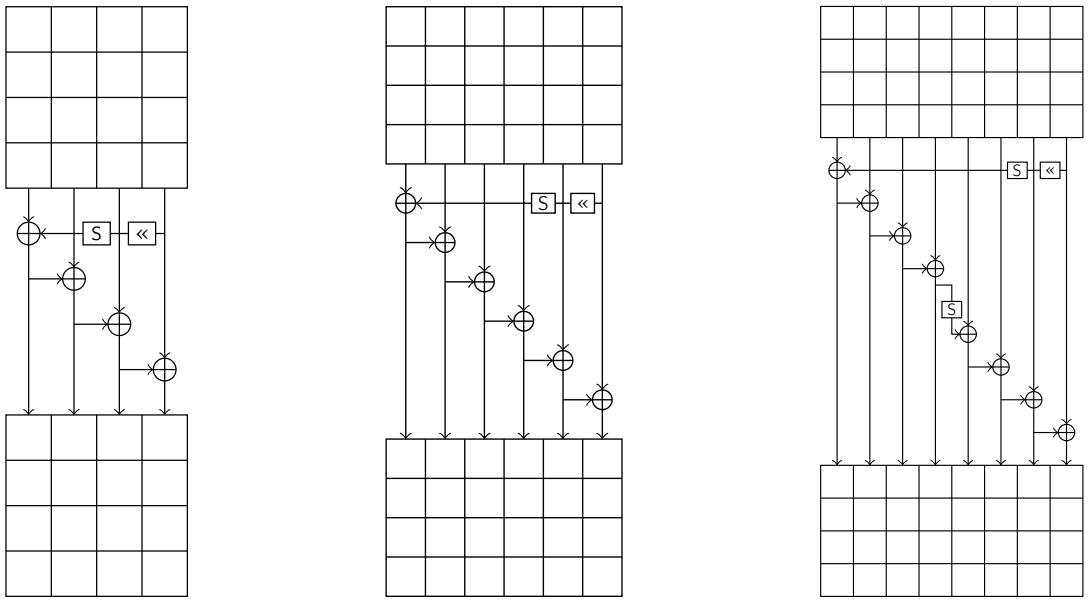
The AES comes in three variants, which only differ by their key schedules. We recall that a key scheduling algorithm consists of an algorithm that *expands* an input key k into a sequence

of $N_r + 1$ subkeys $k_0, \ldots, k_n$. It is sometimes called *key expansion*. We note that the AES is a key-alternating cipher (see Section 3.1.2).

Once the subkeys are generated, they are incorporated into the block cipher to combine the original secret k and the message to encrypt: in the AES, this is achieved by XORing one subkey between each of the N_r applications of the round function. Consequently, the subkeys are as large as the block size, that is 128 bits.

The algorithm to expand the original secret k depends on the variant: for AES-128, the key is exactly as large as the block size, so the initial subkey is fixed to the master key: $k_0 \leftarrow k$. For the largest variants AES-256, the master key k is initially stored into the two first subkeys: $(k_0, k_1) \leftarrow k$. For the AES-192, the key size is 1.5 the block-size, so we use $(k_0, \text{hi}(k_1)) \leftarrow k$, where $\text{hi}(x)$ is the highest part of the state, that is bytes 0 to 7 from Figure 4.1.

Then, to produce the following subkeys, we use the algorithms represented schematically on Figure 4.3. These diagrams show one iteration of the key schedule; therefore, to produce all the subkeys, we need to run it 10 times for AES-128, 7 times for AES-192 and 6 times for AES-256.



(a) AES-128. **(b)** AES-192. **(c)** AES-256.

Figure 4.3: Key schedules of the three variants of the AES: AES-128, AES-192 and AES-256. On the three diagrams, one cell represents one byte, the « performs a rotation upwards by one cell of the whole 4-byte column, and S is the AES non-linear S-Box.

As we describe in the next section, the design of the round function of AES is completely justified to achieve some resistance against known class of attacks, but on the other hand, the key schedules are somewhat ad-hoc and do not rely on particular properties. Constructing a key schedule both efficient and provable secure has been an open problem for a long time, but the authors of the AES did not truly investigate this direction. They have chosen an algorithm adapted from the one of SQUARE but not totally linear as its predecessor. It is widely acknowledged that the key schedule of the AES is the weakest point of its design, while the

round function has been very strongly and securely designed.

4.2.2 Round function

First, the plaintext initializes the internal state matrix, and then, the encryption process applies N_r times a round function, where N_r depends of the version of AES: $N_r = 10$ for AES-128, $N_r = 12$ for AES-192 and $N_r = 14$ for AES-256. Each of the N_r AES round (Figure 4.4)

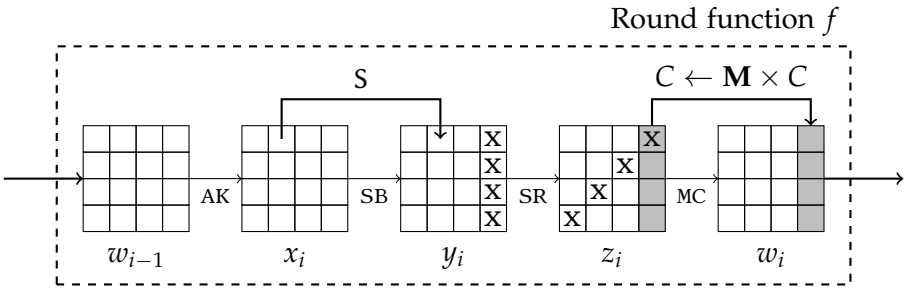


Figure 4.4: One round of AES: description of the round function f

applies four operations to the state matrix (except the last one where we omit the **MixColumns**):

- **AddRoundKey** (AK) adds the next 128-bit subkey to the state,
- **SubBytes** (SB) applies the same 8-bit to 8-bit bijective S-Box S 16 times in parallel on each byte of the state,
- **ShiftRows** (SR) shifts the i-th row left by i positions,
- **MixColumns** (MC) replaces each of the four column C of the state by $\mathbf{M} \times C$ where $\mathbf{M}$ is a constant 4×4 maximum distance separable matrix over $GF(2^8)$.

After the N_r-th round has been applied, the last subkey k_{N_r} is added to the internal state to produce the ciphertext c (see Figure 4.2).

In the remaining of this document, we may need to refer to certain states during the encryption. As depicted on Figure 4.4, we use: w_{i-1} for the state before the i-th key addition, x_i the state after it, y_i the state after the S-Box applications and z_i the state before the **MixColumns** operation.

We note that the **AddRoundKey** follows the **MixColumns** operation, and since they are both linear, their order can be changed. Sometimes, we thus may swap these two operations and refer to the equivalent added subkey by $u_i = \mathbf{M}^{-1}(k_i)$.

The AES is a Substitution-Permutation Network where the substitution step is instantiated by the **SubBytes** operation that introduces non-linearity in the cipher, and the permutation phase is the composition of **MixColumns** and **ShiftRows**. We give more details of both the S and P layers in the next two sections.

4.2.3 The substitution layer

In the AES, the substitution is instantiated by the bijective S-Box S: it is an 8-bit S-Box carefully chosen to bring as much security as possible. We give in Appendix A.1 its full specifications, seen as a lookup table of 2^8 entries, and we give here the formal definition. This part of the cipher needs to be non-linear, that is why the main operation involved in S is the multiplicative inverse in the field $\mathrm{GF}(2^8)$. For an element $x \in \mathrm{GF}(2^8) \setminus \{0\}$, we note $x^{-1} = (x_0, \ldots, x_7)$ its multiplicative inverse, i.e. $x \times x^{-1} = 1$, seen as a vector in the base field $\mathrm{GF}(2)$. To complete this definition, we assume that 0 is sent to itself through the inverse mapping. Then, this vector is changed into the final output by an affine transformation $g : x \to A \times x \oplus b$ given below:

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \xrightarrow{g} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}}_{x} \oplus \underbrace{\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}}_{b} \tag{4.1}$$

The main reason for this choice is the quasi-optimal resistance to linear and differential cryptanalysis in the set of all the 8-bit S-Boxes. We focus here on differential cryptanalysis, and recall the following Proposition 4.1.

Proposition 4.1 (Maximal differential probability of S). *The* AES *S-Box has maximal differential probability* $p_{max} = 2^{-6}$.

For cryptographic applications, the designers of block ciphers want to minimize the maximal differential probability (see Definition 3.3) that basically expresses how strong the S-Box is against differential cryptanalysis. Ideally, one wants this probability to be as small as possible: it cannot be strictly smaller than $2^{-(n-1)}$ since an entry in the DDT[2] is necessarily even. Indeed, for given non-zero differences Δ_i and Δ_o the equation

$$\mathrm{S}(x) \oplus \mathrm{S}(x \oplus \Delta_i) = \Delta_o, \tag{4.2}$$

has either no solution, or an even number of solutions: as soon as $(x, x \oplus \Delta_i)$ verifies Equation (4.2), it is also the case for $(x \oplus \Delta_i, x)$. In the case of the AES, $n = 8$ would mean $p^{\mathrm{S}}_{max} \geq 2^{-7}$. This bound is not tight for the AES as it only reaches $2^{-(n-2)}$, which makes $p^{\mathrm{S}}_{max} = 2^{-6}$.

There have been a lot of research work done in finding good non-linear functions, called *almost perfect non-linear* (APN) functions, see for instance [Nyb91, Nyb93, MS90, Pie90, BCCLC06, BCC10]. The ideal case $2^{-(n-1)}$ is (almost) the perfect case where all differences behave the same way: there is no difference which appears more often than the others so that a differential cryptanalysis would be much harder to mount. Until the work by Dillon et al. in [BDMW10],

[2] Difference Distribution Table.

no APN permutation was known in even dimension, and this is still an open problem to find one for higher dimensions. We give their result as the following permutation in GF(2^6):

$$\pi_6 = \begin{pmatrix} 0 & 54 & 48 & 13 & 15 & 18 & 53 & 35 \\ 25 & 63 & 45 & 52 & 3 & 20 & 41 & 33 \\ 59 & 36 & 2 & 34 & 10 & 8 & 57 & 37 \\ 60 & 19 & 42 & 14 & 50 & 26 & 58 & 24 \\ 39 & 27 & 21 & 17 & 16 & 29 & 1 & 62 \\ 47 & 40 & 51 & 56 & 7 & 43 & 44 & 38 \\ 31 & 11 & 4 & 28 & 61 & 46 & 5 & 49 \\ 9 & 6 & 23 & 32 & 30 & 12 & 55 & 22 \end{pmatrix},$$

which has the particularity that for any non-zero α and β in GF(2^6), the equation

$$\pi_6(x) \oplus \pi_6(x \oplus \alpha) = \beta,$$

has at either 0 or 2 solutions, and nothing else. We now state a more general open problem.

Open problem 1. *Find an APN permutation π_n in an even-sized field GF(2^n) for $n > 6$. That is, find a permutation π_n that reaches maximal differential probability $p_{max}^{\pi_n} = 2^{-(n-1)}$.*

Still, for the AES, $p_{max}^{S} = 2^{-(n-2)}$ is good enough to ensure security against basic differential cryptanalysis beyond exhaustive search. For cryptanalysis applications, we also note Theorem 4.2 which provides an important property of the AES S-Box.

Theorem 4.2 (Differential Property of S). *Given Δ_i and Δ_o two non-zero differences in GF(2^8), the equation*

$$S(x) \oplus S(x \oplus \Delta_i) = \Delta_o, \tag{4.3}$$

has either zero, two of four solutions. This property also applies to S^{-1}.

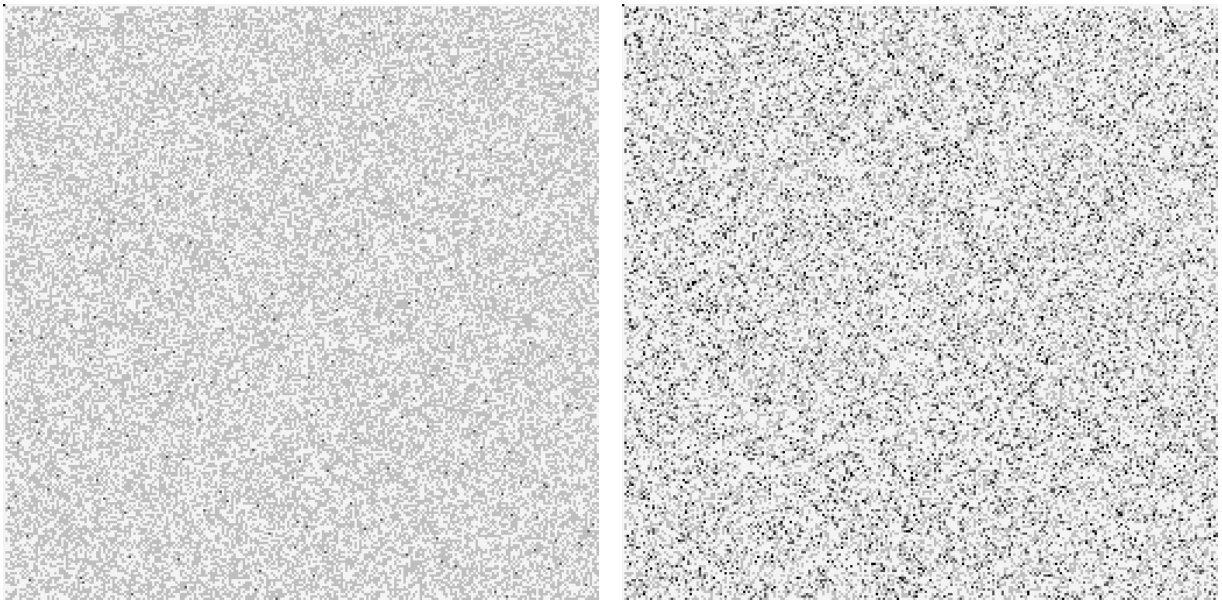
Proof. To prevent the prediction of the propagation of differences in the AES, the S-Box S as been chosen so that almost all differences behave equivalently: none is significantly more frequent than the others.

The number of solutions $N(\Delta_i, \Delta_o)$ of the 8-bit Equation (4.3) is almost constant for any choice of Δ_i and Δ_o. Like for any permutation, we get on solution to this equation on average over all choices of differences Δ_i and Δ_o, but for S there are zero or two, and more rarely four solutions. In detail, for a fixed Δ_i, among the $2^8 - 1$ possible Δ_o, there are $2^7 - 2$ of them for which $N(\Delta_i, \Delta_o) = 0$, another $2^7 - 1$ so that $N(\Delta_i, \Delta_o) = 2$ and the remaining one gives $N(\Delta_i, \Delta_o) = 4$. Due to symmetry, an even number of solutions means that both x and $x \oplus \Delta_i$ are valid.

All in all, if both input and output differences Δ_i and Δ_o are known, then the values are also known. This property allows to deduce the values from the knowledge of the differences. ■

For a more visual representation of the DDT, we print the two following images (Figure 4.5). Each image represents the 256×256 matrix corresponding to the DDT where the coefficients

are non-negative integers. The pixel at row α and column β has a color which is an affine function of the number of solutions of Equation (4.3) when $\Delta_i = \alpha$ and $\Delta_o = \beta$. That is, the darker the pixel is, the greater is $N(\Delta_i, \Delta_o)$. On the left, the DDT of S show that there are very few (Δ_i, Δ_o) such that $N(\Delta_i, \Delta_o) = 4$: there are actually exactly one per row, if we discard the first one. The same reasoning applies on columns. On the right, we have generated the DDT for a randomly draw permutation π among the 256! existing ones. With no surprise, we see that the differential properties of π are not as strong as S. Nevertheless, the AES structure in the standard model can be proven as secure against classical differential attacks even when we consider a random S-Box like π. Still, there could be other valid attacks that S prevents.



(a) AES S-Box. **(b)** Random permutation π.

Figure 4.5: Visual representation of the DDT of the AES S-Box ((a), left) and the DDT of a randomly drawn permutation π ((b), right). Each pixel at coordinate (i, j) is indexed on the gray scale which represents the increasing number of solutions $N(i, j)$ of Equation (4.3): [0, 2, 4, 6+].

4.2.4 The permutation layer

In each round (Figure 4.4), the substitution layer is followed by the permutation layer to mix the values within the state. In the AES, this phase is linear and composed of two transformations: **ShiftRows** and **MixColumns**.

4.2.4.1 ShiftRows

The first one is the **ShiftRows** operation that acts on the rows by changing the positions of the bytes in the state, but does not modify their values. Namely, the rows of the 4×4 matrix are rotated: row i is rotated left by i positions, so that the first row remains unchanged. This linear operation can be seen as a matrix multiplication $\mathbf{P_{SR}} \times v(y_i)$ where the 16-byte state y_i is

considered as a vector $z_i = v(y_i)$ of 16 elements in GF(2^8) and $\mathbf{P_{SR}}$ is a 16×16 permutation matrix from GF(2).

In terms of indexes in the state (Figure 4.1), the byte permutation can be expressed as:

$$SR = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 0 & 5 & 10 & 15 & 4 & 9 & 14 & 3 & 8 & 13 & 2 & 7 & 12 & 1 & 6 & 11 \end{pmatrix}, \tag{4.4}$$

or as explained before, in terms of matrix multiplication by the matrix:

$$\mathbf{P_{SR}} = \begin{pmatrix} 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}. \tag{4.5}$$

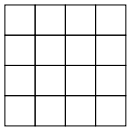
4.2.4.2 MixColumns

The second linear transformation is called **MixColumns**, which acts on the columns. The operation is a matrix multiplication in GF(2^8) between a constant matrix $\mathbf{M}$ specified in the AES standard and the state matrix z_i. The matrix $\mathbf{M}$ is the circulant and invertible matrix $\mathbf{M} = \text{cir}(2, 3, 1, 1)$:

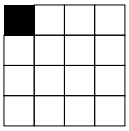
$$\mathbf{M} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \quad \text{with:} \quad \mathbf{M}^{-1} = \begin{pmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{pmatrix}, \tag{4.6}$$

and has been chosen for its good diffusion properties. It is a Maximum Distance Separable (MDS) matrix that ensures a distance at least 5 between any input/output pairs to the matrix. In particular, if two inputs x and x' differ in n_i position, then the multiplication ensures that $\mathbf{M}x$ and $\mathbf{M}x'$ differ in at least n_o positions. The MDS property consists in $n_i + n_o \geq 5$ for any inputs with $n_i \neq 0$, otherwise $n_i = 0$ imposes $n_o = 0$.

In terms of truncated difference (Section 3.4.1), we denote a difference by a non-white cell, and no difference by a white cell, see Figure 4.6. Following the MDS property, the byte difference from Figure 4.6b the **MixColumns** makes the difference spread to 4 bytes at its output.



(a) No difference.



(b) One truncated difference in byte 0.

Figure 4.6: Truncated difference representation in AES: no difference on the left, one truncated difference on the right.

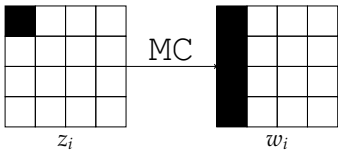


Figure 4.7: Transition $1 \rightarrow 4$ in the **MixColumns**: the truncated difference in z_i is expanded to 4 truncated differences in w_i with the **MixColumns** operation.

4.3 AES-like permutations

In this section, we introduce a generalization of the AES round permutation to encompass more that just the AES. Indeed, there are many symmetric primitives that reuse the design of the AES with slightly different components. To include all these designs in a single parameterized one, we introduce the concept of AES-like permutations.

We keep the notations from the AES to name the substitution and permutations layers, but we generalize the square state to a size $t \times t$ and the width of the cell to c bits. In the case of the AES, we then have $t = 4$ and $c = 8$. The non-linear S-Box is still denoted by S, but we do not necessarily refer to the one of the AES. However, we assume that the generic S verifies the same good properties as the AES S-Box, in particular the differential properties stated in Theorem 4.2. We generalize them in Definition 4.1.

Definition 4.1. We say that S generalizes the AES S-Box to c bits if for any random non-zero Δ_o, the set $\{\Delta_i \mid \exists x, S(x) \oplus S(x \oplus \Delta_i) = \Delta_o\}$ has exactly $2^{c-1} - 1$ elements.

The **ShiftRows** operation does not necessarily shift the cells in a row the same way the AES does, but we assume that all columns depend on each row, which is possible because of the square geometry. The **MixColumns** transformation is supposed to be MDS, where the MDS bound generalizes to: $n_i + n_o \leq t + 1$ if n_i and n_o are the number of active cells at the input (respectively the output) of the **MixColumns**. The main consequence of the generalized diffusion layer mimics the one from the wide trail strategy on the AES: we get full diffusion after two rounds of the permutation when a single bit is flipped at the input. We note that if the state were not square, the full diffusion could only be reached after three rounds.

To give an example, we can consider the case of an AES-like permutation with $t = 8$ and depict it in Figure 4.8. This example encompasses primitives like `Grøstl` or `Whirlpool`, and we analyze this deeper in Section 8.2.

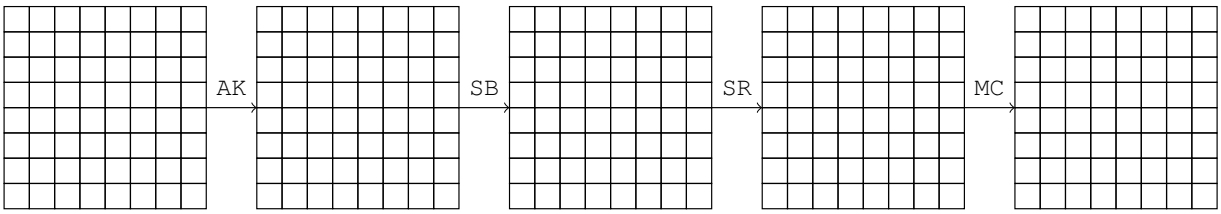


Figure 4.8: Generalized round function of an AES-like permutation with $t = 8$.

4.4 Notable cryptanalytic results

In this section, we give a non-exhaustive list of cryptanalytic results on the AES or its building blocks that have been published since the presentation of `Rijndael`. They include nontrivial properties of the round function with the *square attack* [DKR97, KW02], and an improvement with the *partial sum* technique [FKL+00], a collision attack on 7 rounds [GM00], impossible differential attacks on 7 rounds [BA08, LDKK08, MDRMH10], related-key attacks of the full version of both AES-192 and AES-256 [BKN09, BK09]. Another major work includes the ASIACRYPT 2010 paper by Dunkelman, Keller and Shamir [DKS10] where they improve single-key attacks on AES-192 and AES-256 with advanced meet-in-the-middle techniques. As we improve their results in [DFJ12b], we dedicate the complete Chapter 5 to this analysis and recall theirs.

4.4.1 Square attack

In this section, we recall the square attack, also known as *integral attack* or sometimes *saturation attack*. Lars Knudsen originally develops this technique against the block cipher SQUARE to break up to 6 rounds of the cipher [DKR97].

The attack is a chosen plaintext attack and aims at recovering the secret key k. It uses the algebraic structure of the block cipher and works independently of both the bijective S-Box and the key scheduling algorithm. The basic observation is summarized in Theorem 4.1

Definition 4.2 (δ**-set).** Let a δ-set be a set of 2^8 AES states that assume all values on a particular byte (called *active byte*) and are constant on the 15 other bytes (called *passive bytes*).

Property 4.1. *Let $\mathcal{S}$ be a δ-set. Consider the encryption of each element of this δ-set with three rounds of* AES*. We have the property:*

$$\forall j \in [0, \ldots, 15], \quad \bigoplus_{x \in \mathcal{S}} x[j] = 0. \tag{4.7}$$

We say that byte j is balanced *after three rounds.*

Proof. Let $\mathcal{S}$ be a δ-set. As described previously, one round of AES in composed of several transformations.

First, **AddRoundKey** and **SubBytes** apply a bijective transformation (key addition and S-Box S, respectively) to all the bytes of the state, so that a δ-set is transformed into an other

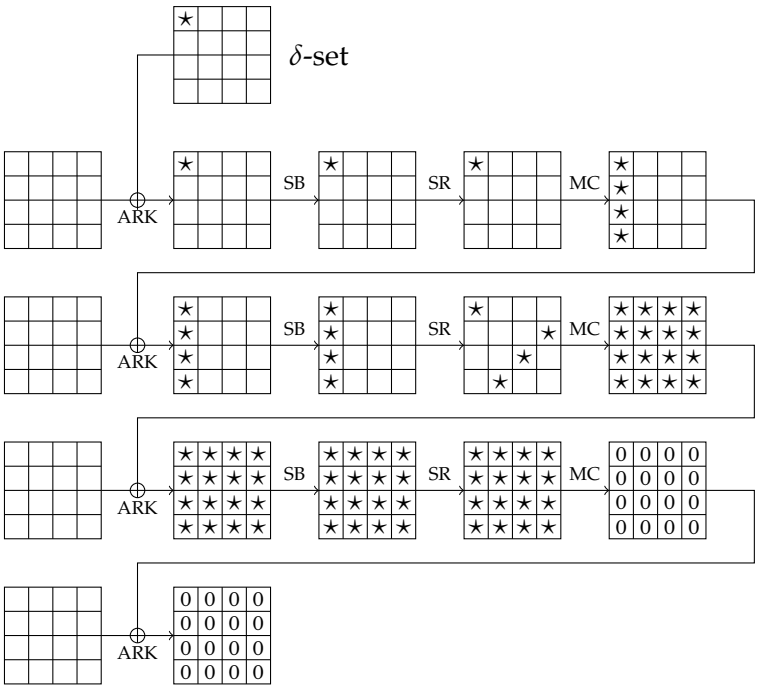


Figure 4.9: Integral distinguisher on 3-round AES: a byte marked by $\star$ is active, a white byte is passive and a byte marked by 0 is balanced.

δ-set. Then, the **ShiftRows** operation shifts the bytes the same way in all the elements of $\mathcal{S}$, so that the resulting set is still a δ-set. Finally, the last operation is the **MixColumns** which applies the MDS matrix, and transforms the δ-set to a set where all the four bytes of one column assume all values.

In the second round, the first **AddRoundKey** and **SubBytes** keep the structure of the δ-set and the **ShiftRows** spreads the four active bytes of the column to one active byte per column. Consequently, the **MixColumns** behaves the same way as in the first round: it makes all 16 bytes of the state to assume all values.

In the third round, the first **AddRoundKey** and **SubBytes** still maintain this property, as well as the **ShiftRows**. Let $Z = \{z^k\}$ be the set of 2^8 inputs of the third **MixColumns** and $W = \{w^k\}$ their respective outputs. To show that the bytes of W are balanced, we compute the sum for a particular position $[i, j]$:

$$\begin{aligned}
\bigoplus_k w^k[i,j] &= \bigoplus_k \textbf{MixColumns}\left(\left[z^k_{0,j},\, z^k_{1,j},\, z^k_{2,j},\, z^k_{3,j}\right]^{\mathrm{T}}\right) \\
&= \bigoplus_k \left(2z^k_{0,j} \oplus 3z^k_{1,j} \oplus z^k_{2,j} \oplus z^k_{3,j}\right) \\
&= 2 \times \bigoplus_k z^k_{0,j} \oplus 3 \times \bigoplus_k z^k_{1,j} \oplus \bigoplus_k z^k_{2,j} \oplus \bigoplus_k z^k_{3,j}.
\end{aligned}$$

The coefficients in the matrix multiplication depend on the position $[i, j]$, but in any case, since

all the bytes of Z assume all values, the four sums equal zero, so that

$$\forall (i,j), \quad \bigoplus_k w^k[i,j] = 0,$$

which proves that all the bytes after 3 rounds of AES of a δ-set are balanced. ■

This property actually describes a distinguisher on 3 rounds of AES, and can be used to construct a key recovery attack on 4, 5 and 6 rounds.

4.4.1.1 Attack on 4 rounds

The attack for 4 rounds is trivial and is a direct application of the distinguisher (Figure 4.10): the adversary begins by constructing a δ-set and asks its encryption.

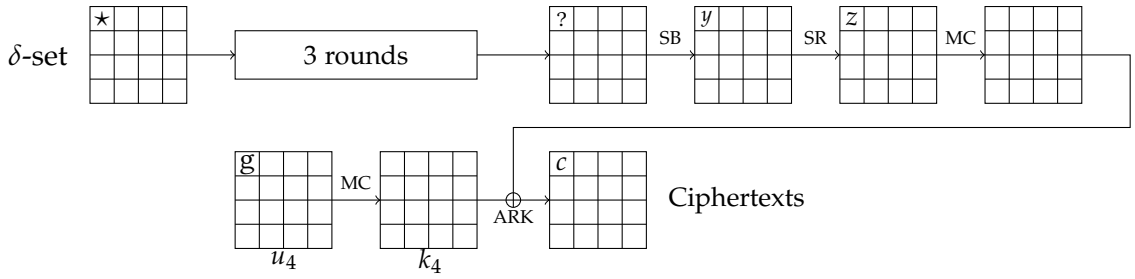


Figure 4.10: Integral attack on 4-round AES: the last equivalent subkey u_4 is recovered from the encryption of a δ-set.

Then, he recovers the bytes of the last equivalent subkey u_4 one by one by guessing them and decrypting the 2^8 ciphertexts to check whether the balanced property is verified: it must be for the correct guess, and could be for a wrong one. Suppose he guesses the key byte to the value g, he then decrypts the corresponding bytes in all the ciphertexts, until $S^{-1}(y)$ and check:

$$\bigoplus_c S^{-1}(y) = \bigoplus_c (S^{-1} \circ SR^{-1})(c+g) = 0. \tag{4.8}$$

This test removes about $1/256$ key value for that byte, and the adversary performs the algorithm for the remaining 15 bytes. Therefore, with two δ-sets, the adversary can shrink the guesses to the correct value and recover the secret key with an overwhelming probability. The data complexity amounts to $2 \cdot 2^8$ chosen plaintexts and the attack runs in $2 \cdot 2^8 \cdot 2^8 \cdot 16 = 2^{21}$ XOR operations and table lookups as we perform 2^8 of them for one guess and one δ-set. As we usually measure an attack in terms of encryption, we assume that a full encryption is composed of 2^6 similar simple operations. This makes the time complexity equivalent to 2^{15} encryptions.

4.4.1.2 Attack on 5 rounds

The attack on 4 rounds uses the previous one as a building block by adding one round at the end (Figure 4.11). To apply the 4-round attack, the adversary first guesses four bytes of the last equivalent subkey u_5 and decrypts a full (shifted) column of the ciphertexts through one round. This allows to learn the values of one column after the fourth round, and the situation is then

comparable to the 4-round attack. He guesses an additional byte in u_4 and performs the check on the corresponding balanced byte. This step can be repeated four times for the four values, and then he needs to guess another column of u_5.

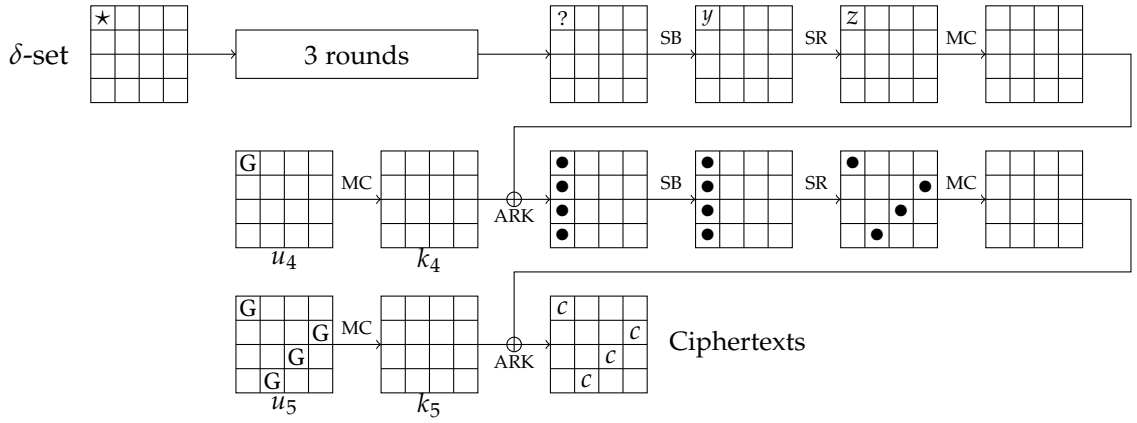


Figure 4.11: Integral attack on 5-round AES: the last equivalent subkey u_5 is recovered from the encryption of a δ-set.

The check on the bytes still removes $1/256$ of the $2^{8\times(4+1)} = 2^{40}$ guessed values, and we repeat this procedure 4 times for the 4 (shifted) columns in the ciphertexts. Consequently, with five or six δ-sets, we should successfully determine the correct secret key: the data complexity amounts to $5 \cdot 2^8$ chosen plaintexts. The number of computations for a single δ-set equals $2^{32} \cdot (2^8 \cdot 4 + 4 \cdot 2^8 \cdot (2^8))$ XOR operations or table lookups, which makes a total time complexity of approximately 2^{52} simple operations, or 2^{46} encryptions.

4.4.1.3 Attack on 6 rounds

We cannot add one more round at the end since two rounds of AES are sufficient to provide a complete diffusion, but we can extend by one round at the beginning (Figure 4.12).

The strategy is to construct a structure of chosen plaintexts that contains several δ-sets after one round and to apply the 5-round attack. Namely, if the initial structure $\mathcal{S}$ contains 2^{32} chosen plaintexts where the main diagonal assumes all the possible values, the structure can be seen as a set of 2^{24} δ-sets of size 2^8 each.

To identify them, the original attack [DKR97] starts by guessing the 4 diagonal bytes of k_0 and selects 2^8 plaintexts from the structure $\mathcal{S}$ such that they result in a δ-set after one round. From there, the previous 5-round attack can be applied by guessing 5 more bytes and summing over the 2^8 partially decrypted ciphertexts to check the balanced state byte. As we guess a total of 9 key bytes, we need about 10 δ-sets to remove all the wrong key guesses: the data complexity therefore amounts to 2^{32} chosen plaintexts. The time complexity is equivalent to

$$2^{32} \cdot 10 \cdot (4 \cdot 2^{32} \cdot 2^8 \cdot 4 + 4 \cdot 2^8 \cdot 2^8) \approx 2^{77}$$

simple operations as we guess 4 bytes in k_0 to construct 10 δ-sets with negligible cost compare to the remaining operations. Then, we sequentially guess 4 times 4 bytes of u_6 and partially decrypt the 2^8 ciphertexts through one round, guess the key byte from u_5 for the 4 positions and compute the XOR of the 2^8 values. Therefore, the time complexity is about 2^{71} encryptions.

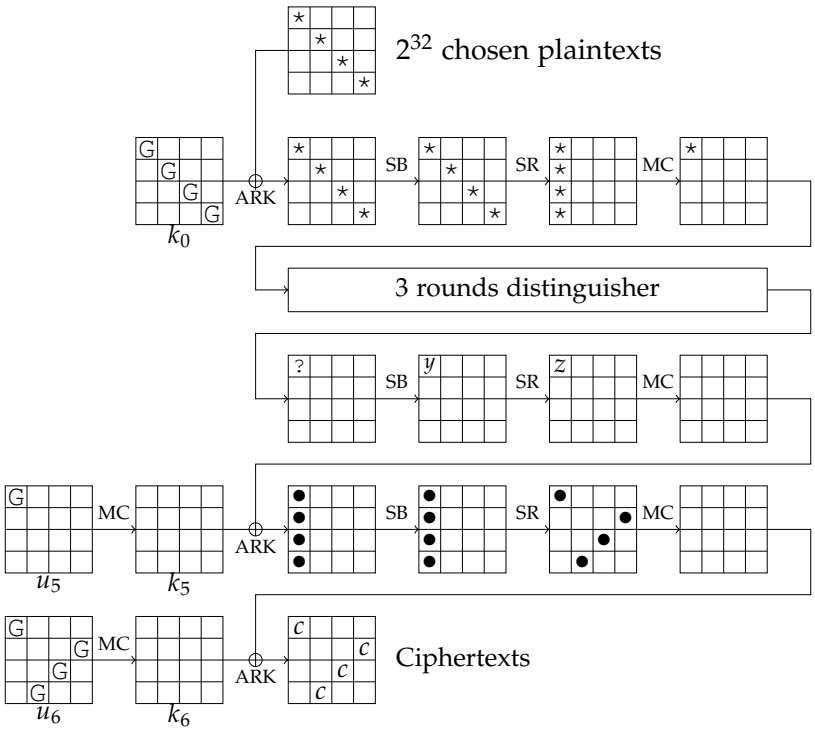


Figure 4.12: Integral attack on 6-round AES: the last equivalent subkey u_6 is recovered from the encryption of a δ-set.

4.4.1.4 Extensions to the larger AES variants

We note that the described attacks work for any key sizes, but we can extend the 6-round technique to 7 rounds by adding one round in the end, and guessing completely its corresponding subkey k_7. This has been first described by Ferguson et al. in [FKL+00]. Naively doing so increase the time complexity by a factor $2^{8 \cdot 16} = 2^{128}$, but we can remark linear relation in the bytes from the key schedule in both AES-192 and AES-256 variants.

Namely, for AES-192, we can spare the guess of 2 bytes in u_6 and one byte in u_5, which makes the time complexity grows to $2^{71+(16-3)\times 8} = 2^{175}$ encryptions, and requires 2^{32} chosen plaintexts.

For AES-256, the key scheduling algorithm aligns the subkeys a bit differently, and we cannot deduce as many bytes from k_7. We can only deduce one byte from u_5, which makes the time complexity equivalent to$2^{71+(16-1)\times 8} = 2^{191}$ encryptions.

4.4.2 Improved square attack with partial sums

4.4.2.1 First improvement

A first improvement of the square attack reduces the time complexity by a factor of 2^8. The observation has been made in [FKL+00] where Ferguson et al. consider the structure $\mathcal{S}$ as a whole, which contains 2^{24} unknown δ-sets. This spares the 4 guesses from k_0.

They observe that even if we cannot distinguish which plaintext belongs to which δ-set, since we no longer known the 4 corresponding key bytes of k_0, we can still apply the previous 5-round attack. Indeed, considering the 2^{32} ciphertexts to perform the check is equivalent at performing the sum of all the check sums for each of the 2^{24} δ-sets, which is true because $0 \oplus 0 = 0$. In the end, we save a factor 2^{32} by not guessing 4 bytes in k_0, but we need to compute the XOR of 2^{32} rather than 2^8. In total, we save a factor 2^8 in the time complexity.

4.4.2.2 Second improvement

In the same paper [FKL+00], Ferguson et al. also note that we can further improve this result by a factor 2^{28}. The final computed value and the approach to get the data is exactly the same as before, but they observe a way to group the computations to reduce the number of operations performed.

Their method is called *partial sums* as the technique to group the computations relies on partial summations. Namely, in the previous attack, we compute a sum over all the $2^{24} \times 2^8 = 2^{32}$ ciphertexts c_i that depends on 9 parameters:

- 4 bytes from the ciphertexts c_i that we denote by $c_{i,0}$, $c_{i,1}$, $c_{i,2}$ and $c_{i,3}$,
- 4 key guesses of u_6 that we note k_0, k_1, k_2, k_3,
- and an additional guess noted k_4 from u_5.

The sum σ computed to check the balanced state equals:

$$\sigma \stackrel{\text{def}}{=} \bigoplus_i \mathrm{S}^{-1}\Big(L_{k_0,k_1,k_2,k_3,k_4}(c_{i,0}, c_{i,1}, c_{i,2}, c_{i,3})\Big), \tag{4.9}$$

with:

$$L_{k_0,k_1,k_2,k_3,k_4}(c_{i,0}, c_{i,1}, c_{i,2}, c_{i,3}) = \\ \mathrm{S}_0\left(c_{i,0} \oplus k_0\right) \oplus \mathrm{S}_1\left(c_{i,1} \oplus k_1\right) \oplus \mathrm{S}_2\left(c_{i,2} \oplus k_2\right) \oplus \mathrm{S}_3\left(c_{i,3} \oplus k_3\right) \oplus k_4, \tag{4.10}$$

where $\mathrm{S}_0, \ldots, \mathrm{S}_3$ are bijective applications of the form $x \to \alpha \times \mathrm{S}^{-1}(x)$ for a fixed non-zero α. The inner sums in Equation (4.9) that involves the S_i come from the **MixColumns** application of the penultimate round.

Naively iterating over the different values for the 9-byte parameter requires 2^{72} operations to evaluate all the sums (see Figure 4.13). Basically, for each ciphertext c_i, we loop over the 2^{40} values for the 5 key guesses, and compute the value of the function $L_{k_0,k_1,k_2,k_3,k_4}(c_{i,0}, c_{i,1}, c_{i,2}, c_{i,3})$ denoted $\sigma_{i,4}$. Once we have done that, we have reduced the list of 2^{32} ciphertexts $(c_i)_i$ to a list of 2^{32} bytes, and we sum their images by S^{-1}. This requires $2^{32} \cdot 2^{40} = 2^{72}$ operations.

More efficiently, we start by guessing two values in u_6, say k_0 and k_1 (see Figure 4.14). For each ciphertext c_o, we can evaluate the partial sum:

$$\sigma_{i,1} \stackrel{\text{def}}{=} \mathrm{S}_0\left(c_{i,0} \oplus k_0\right) \oplus \mathrm{S}_1\left(c_{i,1} \oplus k_1\right). \tag{4.11}$$

That way, we replace the list of 2^{32} ciphertexts by a list of tuples

$$\left(\sigma_{i,1},\ c_{i,2},\ c_{i,3}\right), \tag{4.12}$$

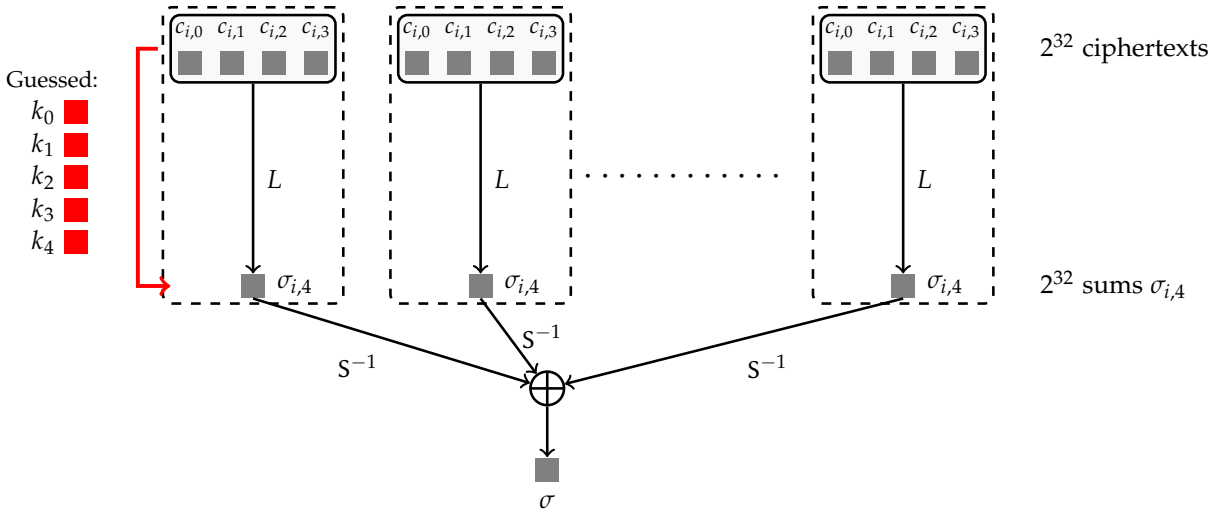


Figure 4.13: Basic summation for the integral attack: for each ciphertext, the guessed values in the subkeys are used to evaluate L and produce the final sum σ.

that contains the partial sum right behind the non-linear S where to perform the check for the guesses k_0 and k_1.

We note that for a single ciphertext c_i, the guesses k_0 and k_1 can take up to 2^{16} values, but the sum $\sigma_{i,1}$ only 2^8. Moreover, for that ciphertext, we are interested in the XOR of all the values, so it is sufficient to know whether a particular value appear in the summation; that is counting the occurrence modulo 2. This trick allows to compress the 2^{32} possible tuples $(\sigma_{i,1}, c_{i,2}, c_{i,3})$ for all the k_0, k_1 for a single ciphertext to only 2^{24}. Therefore, the initial list of 2^{32} ciphertexts is reduced to a list of 2^{24} tuples, and this has cost $2^{32} \times 2^{16} = 2^{48}$ operations (Figure 4.14).

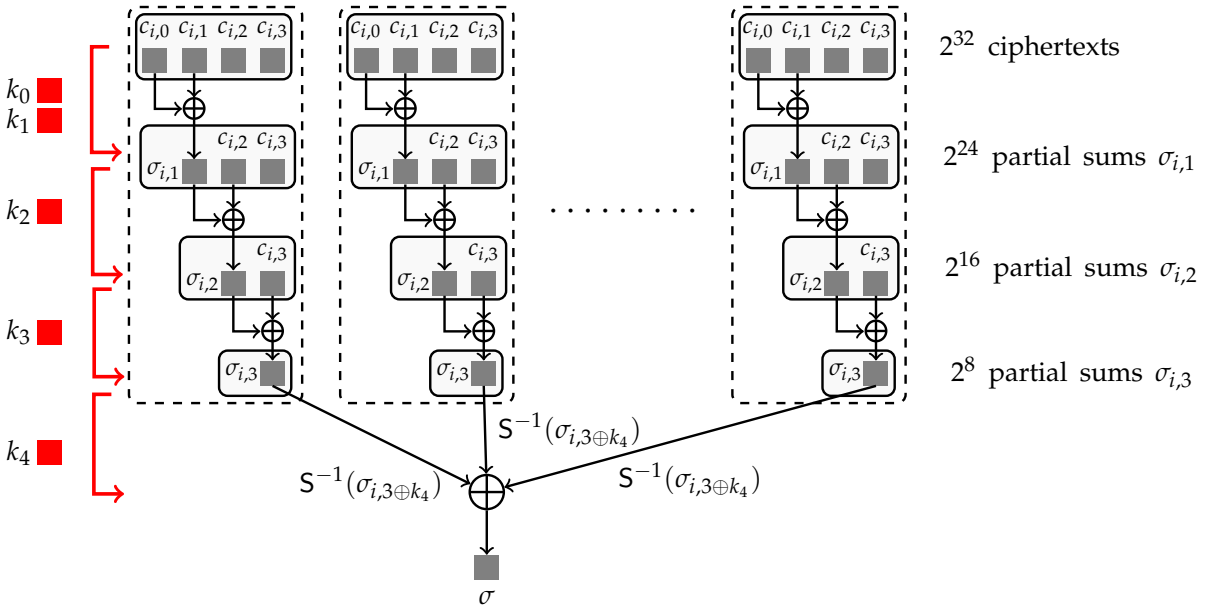


Figure 4.14: Improved summation for the integral attack: the key bytes are progressively guessed while the input data structure is being compressed.

Then, we continue by guessing the third byte k_2 and the 2^{24} constructed tuples: the time complexity equals $2^{24} \times 2^{24} = 2^{48}$, and we can perform the same computation. From a guess k_2 and a tuple $(\sigma_{i,1}, c_{i,2}, c_{i,3})$, we construct the new tuples

$$(\sigma_{i,2},\ c_{i,3}) = (\sigma_{i,1} \oplus \mathrm{S}_2\left(c_{i,2} \oplus k_2\right),\ c_{i,3}),$$

which can be compressed to 2^{16} possible values. We continue by guessing k_3, and we construct the values

$$\sigma_{i,3} = \sigma_{i,2} \oplus \mathrm{S}_3\left(c_{i,3} \oplus k_3\right)$$

at a cost of $2^{32} \times 2^{16} = 2^{48}$ to guess 4 bytes from u_6 and the 2^{16} previous tuples. Finally, we are left with a structure of 2^8 possible values for the partial sum $\sigma_{i,3}$

$$\sigma_{i,3} = \mathrm{S}_0\left(c_{i,0} \oplus k_0\right) \oplus \mathrm{S}_1\left(c_{i,1} \oplus k_1\right) \oplus \mathrm{S}_2\left(c_{i,2} \oplus k_2\right) \oplus \mathrm{S}_3\left(c_{i,3} \oplus k_3\right), \tag{4.13}$$

and the final guess k_4 allows to construct the final sum in Equation (4.9) in time $2^{40} \times 2^8 = 2^{48}$, needed to guess the 5 key bytes and the 2^8 values in the previous structure.

In the end, this technique computes the sum to check the balanced state of a single byte in the square attack in 2^{48} simple computations and 2^{32} memory units to store the data and the intermediate sums.

4.4.2.3 Extension to more rounds

Extending the partial sum technique to 7 rounds can be done in a similar way as the extension of the original square attack: we begin by guessing completely the 16 bytes of the very last subkey, and then, the technique applies on a similar equation as Equation (4.9), but with more variables.

For AES-192, the 16 guesses from k_7 linearly gives 3 bytes in the two previous subkeys. The partial sum technique compresses the initial structure of 2^{32} elements into 2^{24} counters: for each guess of k_7, precomputed tables can perform this operation in 2^{32} table lookups. As before, each compression step is done in the same amount of time, here $2^{128+32} = 2^{160}$ computations. In terms of AES encryptions, we estimate this to 2^{155} encryptions.

For AES-256, guessing k_7 provides only one additional byte in k_5 such that the first step in the partial sum technique requires $2^{128+32} = 2^{160}$ computations, but then we need to add the two guesses which increases the complexity to 2^{176} computations, or in total about 2^{172} encryptions.

4.4.2.4 The herd attack

In the same paper, Ferguson et al. [FKL+00] show that it is possible to reach an attack with almost the full codebook and a time complexity of 2^{120} encryptions. The main point is similar to the one the partial sum technique: if we have the 2^{128} ciphertexts corresponding to 2^{120} δ-sets, then the sum over them all to check the square property should yield zero, as it is done over 2^{32} ciphertexts for the partial sum.

However, this property would be verified for any bijective application and therefore leaks no information about the key. To apply it, we can partition the input data to 2^{120} structures

of 2^8 plaintexts called *herds*. Now, summing over a herd should yields zero for a correct key guess, but is unlikely to happen for a random application.

For 7 rounds, this "attack" can be performed in about 2^{120} encryptions using 2^{64} memory and all the 2^{128} plaintexts. We note that we could discuss whether this result actually describes an attack in the sense that the adversary knows the full codebook, which gives him the ability to encrypt and decrypt any messages for that secret key and already requires 2^{128} computations and memory to retrieve. We can extend this to 8 rounds with 2^{188} encryptions for AES-192 and 2^{204} for AES-256.

4.4.3 Collision attack

We describe here the collision attack presented at the AES conference in 2000 by Henri Gilbert and Marine Minier [GM00]. They revisit the basic integral distinguisher for 3 rounds of AES to describe a nontrivial property over 4 rounds, and apply it to get a collision attack on reduced variants of AES.

4.4.3.1 Distinguishers

Similarly as the square property stated in Theorem 4.1, we can formulate their result in the two following properties.

Property 4.2 ([GM00]). *Let $\mathcal{S}$ be a δ-set active in one byte α. Consider the encryption of each element of $\mathcal{S}$ with 3 rounds of* AES *with no whitening subkey. The function that maps α to any byte in the output state is fully determined by 9 byte parameters.*

Proof. Let consider the encryption of a δ-set over 3 rounds of AES with unknown subkeys $k_i, \ldots, k_{i+3}$ as shown on Figure 4.15. We show how to express the bytes $x_3[0]$ as a function of $x_0[0]$ depending of constant bytes.

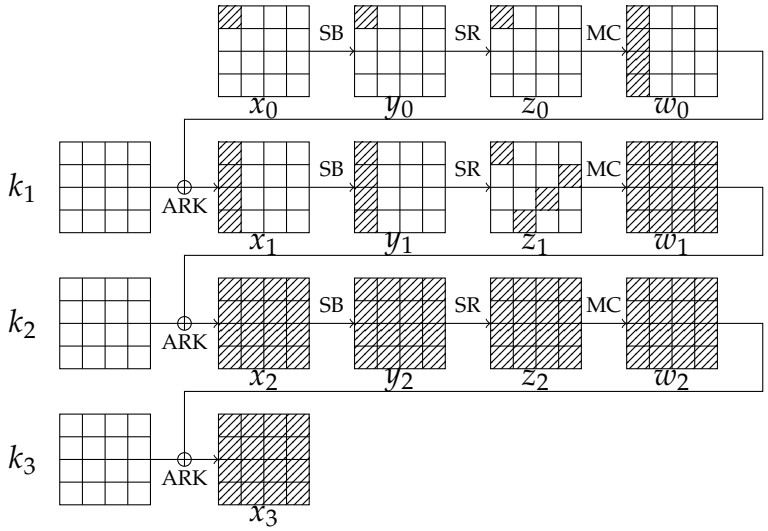


Figure 4.15: Functional distinguisher from Gilbert and Minier for 3 rounds of AES.

We can trace the encryption of a plaintext x_0 in the δ-set in terms of equations dependent of byte variables. After one round, we can write the variables w_0 in terms of affine functions of $y_0[i] = \mathtt{S}(x_0[i])$. Namely, we have:

$$\begin{aligned}
w_0[0] &= 02 \cdot \mathtt{S}(x_0[0]) \oplus L_0\Big(\mathtt{S}(x_0[5]), \mathtt{S}(x_0[10]), \mathtt{S}(x_0[15])\Big),\\
w_0[1] &= 01 \cdot \mathtt{S}(x_0[0]) \oplus L_1\Big(\mathtt{S}(x_0[5]), \mathtt{S}(x_0[10]), \mathtt{S}(x_0[15])\Big),\\
w_0[2] &= 01 \cdot \mathtt{S}(x_0[0]) \oplus L_2\Big(\mathtt{S}(x_0[5]), \mathtt{S}(x_0[10]), \mathtt{S}(x_0[15])\Big),\\
w_0[3] &= 03 \cdot \mathtt{S}(x_0[0]) \oplus L_3\Big(\mathtt{S}(x_0[5]), \mathtt{S}(x_0[10]), \mathtt{S}(x_0[15])\Big),
\end{aligned}$$

where $L_0, \ldots, L_3$ are linear functions whose coefficients depend on the matrix $\mathbf{M}$. The remaining variables $w_0[j]$ for $j \geq 4$ are independent of the active byte α. We define the added constants in the previous equations XOR-ed with the key bytes $k_0[0,1,2,3]$ by c_i and consider the first in x_1:

$$\begin{aligned}
x_1[0] &= 02 \cdot \mathtt{S}(x_0[0]) \oplus c_0\\
x_1[1] &= 01 \cdot \mathtt{S}(x_0[0]) \oplus c_1\\
x_1[2] &= 01 \cdot \mathtt{S}(x_0[0]) \oplus c_2\\
x_1[3] &= 03 \cdot \mathtt{S}(x_0[0]) \oplus c_3
\end{aligned}$$

so that the effect of the second round on those bytes leads to the following expression for the diagonal bytes of w_1:

$$\begin{aligned}
w_1[0] &= 02 \cdot \mathtt{S}\Big(02 \cdot \mathtt{S}(x_0[0]) \oplus c_0\Big) \oplus 03 \cdot \mathtt{S}(x_1[5]) \oplus 01 \cdot \mathtt{S}(x_1[10]) \oplus 01 \cdot \mathtt{S}(x_1[15]),\\
w_1[5] &= 01 \cdot \mathtt{S}\Big(03 \cdot \mathtt{S}(x_0[0]) \oplus c_3\Big) \oplus 01 \cdot \mathtt{S}(x_1[4]) \oplus 02 \cdot \mathtt{S}(x_1[9]) \oplus 03 \cdot \mathtt{S}(x_1[14]),\\
w_1[10] &= 02 \cdot \mathtt{S}\Big(01 \cdot \mathtt{S}(x_0[0]) \oplus c_2\Big) \oplus 03 \cdot \mathtt{S}(x_1[7]) \oplus 01 \cdot \mathtt{S}(x_1[8]) \oplus 01 \cdot \mathtt{S}(x_1[13]),\\
w_1[15] &= 01 \cdot \mathtt{S}\Big(01 \cdot \mathtt{S}(x_0[0]) \oplus c_1\Big) \oplus 01 \cdot \mathtt{S}(x_1[6]) \oplus 02 \cdot \mathtt{S}(x_1[11]) \oplus 03 \cdot \mathtt{S}(x_1[12]),
\end{aligned}$$

that we can reformulate by introducing constants c_i, $i \geq 4$, for the three last terms of each line. That is:

$$\begin{aligned}
x_2[0] &= 02 \cdot \mathtt{S}\Big(02 \cdot \mathtt{S}(x_0[0]) \oplus c_0\Big) \oplus c_4,\\
x_2[5] &= 01 \cdot \mathtt{S}\Big(03 \cdot \mathtt{S}(x_0[0]) \oplus c_3\Big) \oplus c_5,\\
x_2[10] &= 02 \cdot \mathtt{S}\Big(01 \cdot \mathtt{S}(x_0[0]) \oplus c_2\Big) \oplus c_6,\\
x_2[15] &= 01 \cdot \mathtt{S}\Big(01 \cdot \mathtt{S}(x_0[0]) \oplus c_1\Big) \oplus c_7.
\end{aligned}$$

The same analysis can be performed for the other diagonal bytes, but here we want to get the expression of $x_3[0]$ which lies in the first column of x_3 and therefore only depends on the four diagonal bytes of x_2.

As we can express $x_3[0]$ with the **MixColumns** coefficients by:

$$x_3[0] = k_3[0] \oplus 02 \cdot \mathtt{S}(x_2[0]) \oplus 03 \cdot \mathtt{S}(x_2[5]) \oplus 01 \cdot \mathtt{S}(x_2[10]) \oplus 01 \cdot \mathtt{S}(x_2[15]),$$

we can also deduce its dependence to $x_0[0]$ from the above equations. Indeed, $x_3[0]$ depends on $x_0[0]$ and the parameters $c_0, \ldots, c_3$ from the first round, $c_4, \ldots, c_7$ from the second round and $k_3[0]$. Consequently, the mapping $x_0[0] \to x_3[0]$ is fully determined by the 9 key-dependent constants $(c_0, \ldots, c_7, k_3[0])$. ■

The second property is a direct consequence of the first one, and provide a distinguisher for 4 rounds of AES by giving a way to test whether the 3 first rounds verify the 3-round property.

Property 4.3 ([GM00]). *Let $\mathcal{S}$ be a δ-set active in a single byte b. Consider the encryption of each element of $\mathcal{S}$ with 4 rounds of* AES*. The mapping*

$$b \to S^{-1}\Big(14 \cdot x_4[0] \oplus 11 \cdot x_4[1] \oplus 13 \cdot x_4[2] \oplus 09 \cdot x_4[3] \oplus k_4[0]\Big) \tag{4.14}$$

is fully determined by 9 byte parameters depending on the key and inactive bytes of $\mathcal{S}$.

4.4.3.2 Collision attack on 4 rounds

From the proof of Theorem 4.2, Gilbert and Minier note in [GM00] that the four first constants $c_0, \ldots, c_3$ are independent of the bytes $x_0[1]$, $x_0[2]$ and $x_0[3]$, but this is not the case for the four other constants $c_4, \ldots, c_7$.

Consequently if we assume that the values for $(c_4, \ldots, c_7)$ are uniformly distributed over the choices of $(x_0[1], x_0[2], x_0[3])$, we expect that two elements in a list of 2^{16} elements $(x_0[1], x_0[2], x_0[3])$ would produce a collision in the 32-bit value $(c_4, \ldots, c_7)$. That is, by the birthday paradox, we expect to find $(x_0[1], x_0[2], x_0[3]) \neq (x'_0[1], x'_0[2], x'_0[3])$ such that $(c_4, \ldots, c_7) = (c'_4, \ldots, c'_7)$ which incidentally produces the same mapping $x_0[0] \to x_3[0]$.

From Theorem 4.2, we see that a collision occurs as soon as the linear combination of $x_4[0], \ldots, x_4[3]$ in the inner expression of Equation (4.14) collides. That is, if for two different choices $(x_0[1], x_0[2], x_0[3])$ and $(x'_0[1], x'_0[2], x'_0[3])$, the equation

$$\begin{aligned} &14 \cdot x_4[0] \oplus 11 \cdot x_4[1] \oplus 13 \cdot x_4[2] \oplus 09 \cdot x_4[3] \\ &= 14 \cdot x'_4[0] \oplus 11 \cdot x'_4[1] \oplus 13 \cdot x'_4[2] \oplus 09 \cdot x'_4[3] \end{aligned} \tag{4.15}$$

holds, then we detect in round 4 that the collision in the two mappings $x_0[0] \to x_3[0]$ and $x'_0[0] \to x'_3[0]$ occurs in round 3.

4.4.3.3 Extension to 7 rounds

To apply this technique to 7 rounds of AES, we can prepend one round at the beginning exactly in the same way as it has been done for the square attack (Section 4.4.1.3), at the expense of guessing the four corresponding values in the first subkey.

Then, we also add two more rounds at the end. To check for the collision with the Equation (4.15), we use a meet-in-the-middle strategy to decrease the time complexity by guessing fewer bytes while using memory to store the intermediate computations. Namely, we rewrite Equation (4.15), and we note that x_4 is pushed to x_5 since we have added one round at

the beginning:

$$\begin{aligned} &14 \cdot \Big(x_5[0] \oplus x'_5[0]\Big) \oplus 11 \cdot \Big(x_5[1] \oplus x'_5[1]\Big) \\ &= 13 \cdot \Big(x_5[2] \oplus x'_5[2]\Big) \oplus 09 \cdot \Big(x_5[3] \oplus x'_5[3]\Big) \end{aligned} \tag{4.16}$$

The strategy to apply a key recovery attack with the 4-round distinguisher starts by encrypting the 2^{32} chosen plaintexts, and continue by guessing the four diagonal key bytes of k_0 (see Figure 4.16). Using precomputation, we can store in small tables the δ-set along

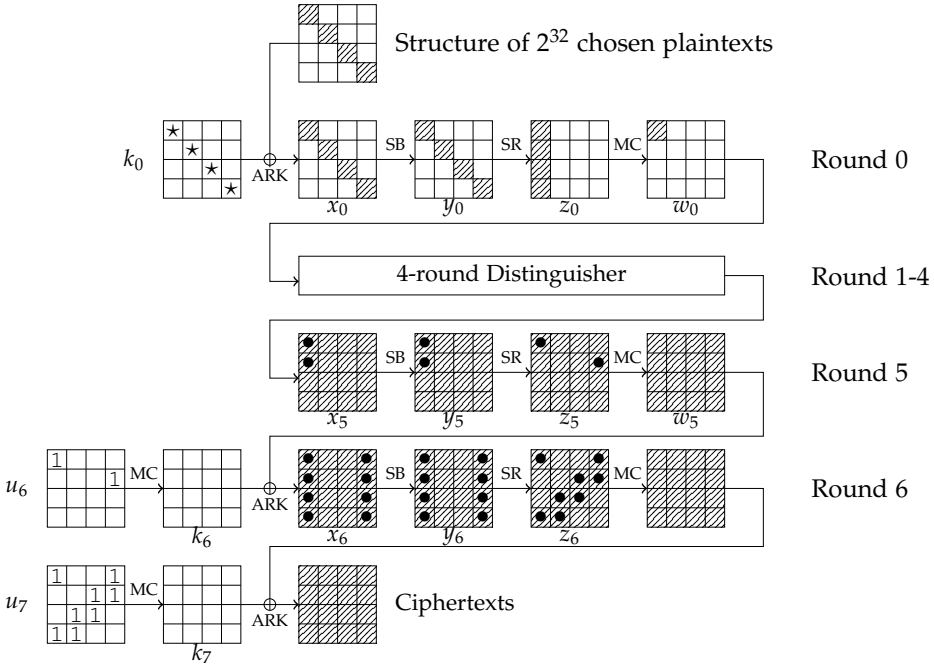


Figure 4.16: Gilbert and Minier attack on 7-round AES based on the 4-round collision distinguisher: the figure shows the first half of the guessing process to perform the meet-in-the-middle. Hatched bytes have a non-zero difference.

with the constant values of $x_1[1,2,3]$ to retrieve efficiently the plaintexts to use. Then, for each ciphertext pairs, we guess the 10 bytes marked by 1 in Figure 4.16 that allows to partially decrypt the pair until $x_5[0]$ and $x_5[1]$. From those two values, we can compute the left-hand side of Equation (4.16) and store this value in a table T along with the guessed material. Repeat this procedure for the 8 other bytes of u_7 and the two bytes $u_6[7]$ and $u_6[13]$ to partially decrypt $x_5[2]$ and $x_5[3]$ and compute the right-hand side of Equation (4.16).

This whole procedure takes about $2^{32} \times 2^{24} \times 2^{80} \times 2^8 \approx 2^{144}$ simple operations and requires 2^{32} chosen plaintexts and 2^{80} memory units to perform the meet-in-the-middle. In the end, this attack retrieves completely the last subkey k_7, and we can apply a more efficient attack on 6 rounds to retrieve the remaining key material if we are targeting either AES-192 or AES-256. We can for instance use the previously described attack by Ferguson et al. with partials sums (see Section 4.4.2).

In the case where the block cipher is AES-128, Gilbert and Minier discuss in [GM00, Min02] a technique to make the exhaustive search more efficient by using large precomputed tables to

decrease the number of operations to test a single key. Anyway, this attack remains marginal for AES-128 as the time complexity is very close to 2^{128} and the memory requirements are impractical.

4.4.4 Impossible differential attack

Impossible differential cryptanalysis is a form of differential cryptanalysis that collects information on the secret key from impossible events during the encryption of particular pairs of plaintexts (see Section 3.4.2). In the case of the AES, we usually start by finding an impossible differential characteristic with the miss-in-the-middle technique by concatenating two differential characteristics that hold with probability one. For instance, the truncated differential characteristic in Figure 4.17 is impossible. This particular impossible differential characteristic

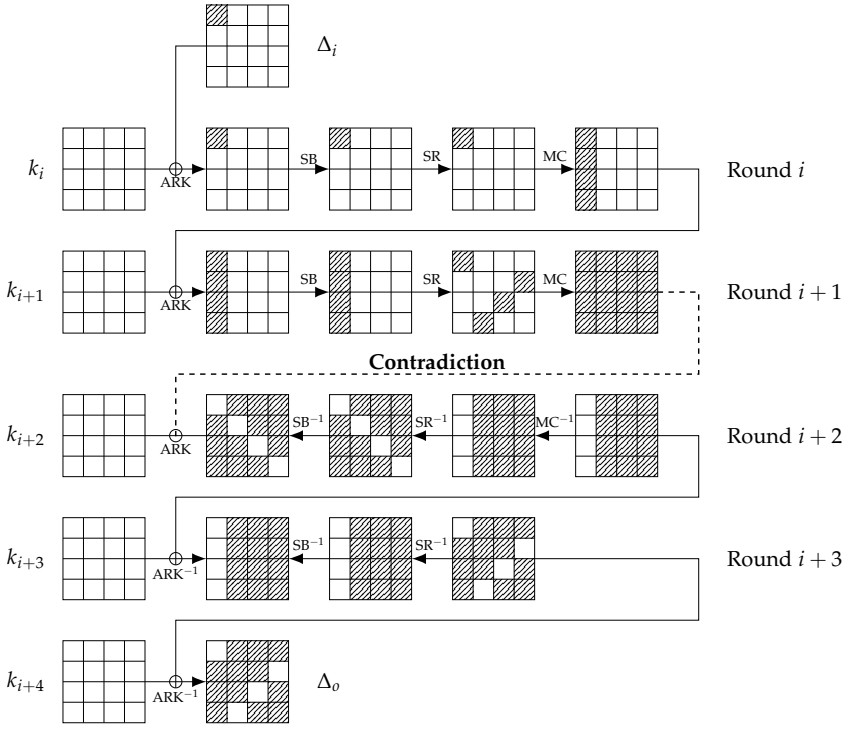


Figure 4.17: Impossible differential characteristic on 4-round AES used in several impossible differential attacks. Hatched bytes have a non-zero difference.

actually relies on a more general impossibility result for 4 rounds of AES without the last **MixColumns** that we state in the following Property 4.4. This property is the basis for almost all the known impossible differential results on the AES.

Property 4.4. *Let (x, y) a pair of input* AES *states with difference $\Delta_i = x \oplus y$. We consider the encryption of x and y through 4 rounds of* AES *while omitting the last* ***MixColumns*** *operation, and denote Δ_o the output difference. If we have:*

1. *Δ_i is active in only one byte position,*
2. *and Δ_o is inactive at least in a shifted column,*

then the differential $\Delta_i \rightarrow \Delta_o$ over 4 rounds of AES *is impossible.*

Proof. The strategy follows the miss-in-the-middle (Section 3.4.2) argument that consists in concatenating two differential characteristics that hold with probability one.

In the forward direction, we start round i with a single byte non-zero difference. Consequently, after two full rounds we end up with probability one to a difference active in all the 16 bytes. This follows the design strategy of the AES.

In the backward direction, the four possible patterns of inactive differences lead to a fully inactive column at the beginning of round $i+3$, at least. Backtracking one more round keeps the inactivity in those four bytes through the **MixColumns** operation of round $i+2$ so that a diagonal is fully inactive at the beginning of round $i+2$.

Therefore, the input difference to round $i+2$ should be fully active from the forward direction and inactive in at least one diagonal from the backwards argument. This contradiction concludes the proof for the impossible differential $\Delta_i \to \Delta_o$. ■

4.4.4.1 Bahrak and Aref attack on 7-round AES-128

We now describe the original attack of Bahrak and Aref from [BA08] which gives the basics of impossible differential attacks on AES.

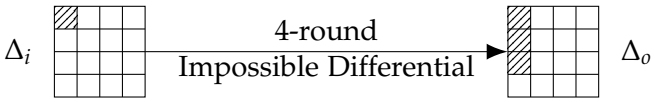


Figure 4.18: Impossible differential on 4-round AES used in the Bahrak and Aref attack.

It relies on Theorem 4.4 by considering the impossible differential of Figure 4.18: they integrate this 4-round characteristic in a 7-round one by appending 2 rounds where the last one omits the **MixColumns** operation, and prepending one round (Figure 4.19).

The technique starts by considering N_s structures of 2^{32} chosen plaintexts such that the main diagonal assumes all the possible values. We specify the value for N_s as soon as we have evaluated the conditions for the attack to succeed. One structure allows to construct $N_s \times \binom{2^{32}}{2} \approx N_s \times 2^{63}$ pairs, which can be filtered after encryption to remove all the pairs whose ciphertext differences are not inactive of the 8 bytes at positions 1, 2, 4, 5, 11, 14 and 15. This is a 64-bit condition which leaves only $N_s \times 2^{-1}$ pairs of data.

Then, we guess the 4 bytes $k_7[12, 9, 6, 3]$ from the last subkey k_7 and partially decrypt the $N_s \times 2^{-1}$ pairs through round 6. From the difference in the last column in state x_6, we can compute the difference in the same column in z_5 using the linearity of the **MixColumns**. Therefore, we can remove all the pairs that are not inactive on bytes 12, 14 and 15 on z_5. A random pair passes this test with probability $(2^{-8})^3$ so only $N_s \times 2^{-25}$ pairs survive.

We continue by guessing the 4 bytes $k_7[8, 5, 2, 15]$ and by partially decrypting the corresponding column in the remaining pairs. With the same probability as before, only one byte is active in the first column of z_5. Consequently, there are 2^{-49} remaining pairs that verify all the conditions.

Next, we guess two more bytes in the equivalent subkey u_6, bytes $u_6[0]$ and $u_6[13]$, and partially decrypt the two active bytes at the beginning of round 5. From this, we compute

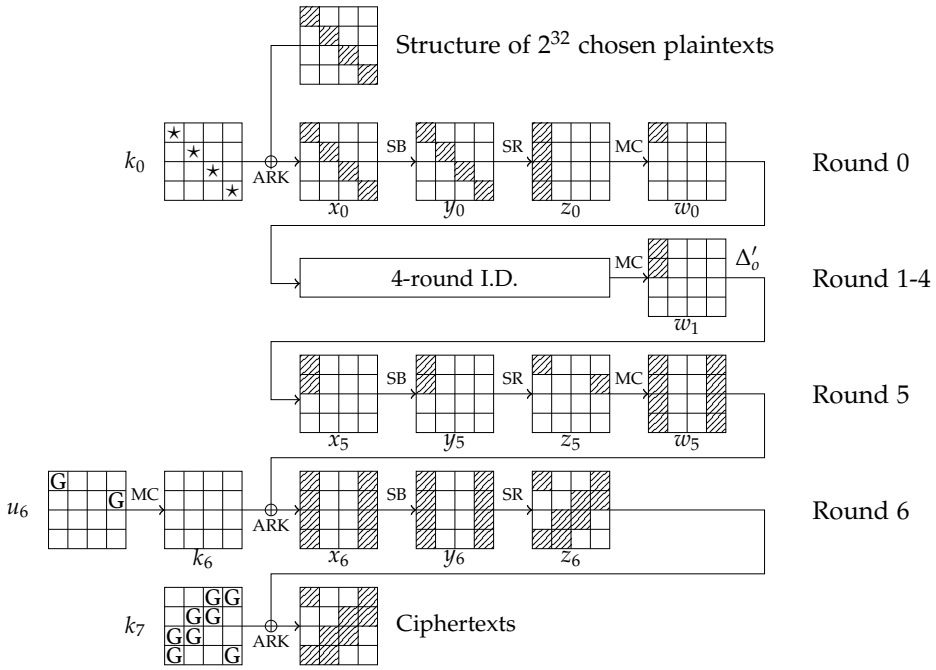


Figure 4.19: Differential characteristic used in the Bahrak and Aref attack on 7 rounds AES-128.

the Δ'_o difference in state w_1 and check if $\mathbf{M}^{-1}(\Delta'_o)$ is inactive in at least one byte as the output difference of the impossible differential from Figure 4.18. This is true with probability $\binom{4}{1} \cdot 2^8 = 2^{-6}$, which makes the number of valid pairs until that point to $N_s \times 2^{-55}$.

Now, we use the impossible differential to remove 32 bits of key from the initial subkey k_0. Namely, for each value on the bytes marked by $\star$ in k_0 that change the plaintext difference on one remaining pairs to a single active byte at the input of round 1, we discard this value with probability 1. To save time, we can precompute a hash table of 2^{40} elements indexed by the difference in the plaintext: for a 32-bit difference, we directly get the list of 2^8 wrong key candidates. Therefore, by exhausting the $N_s \times 2^{-55}$ remaining pairs and the 2^8 wrong key values for each pair, we expect

$$2^{32} \times (1 - 2^{-32})^{N_s \times 2^{-55} \times 2^8}$$

guesses from k_7, u_6 and k_0 to remain. Consequently, wrong values for the 32 bits of k_0 are expected to remain with probability

$$p = 2^{4+4+2} \times 2^{32} \times (1 - 2^{-32})^{N_s \times 2^{-55} \times 2^8}$$

since we guess $4 + 4$ bytes in k_7 and 2 bytes in u_6. To choose the correct number of structures N_s, we want $p \ll 1$ while minimizing N_s to reach the smallest data complexity. The smallest N_s that verifies this is $N_s \approx 2^{85.5}$, which makes an attack in data complexity $2^{85.5+32} = 2^{117.5}$ chosen plaintexts, and a time complexity equivalent to

$$\underbrace{2^{117.5}}_{\text{Data encryption}} + \underbrace{2^{84.5} \times 2^{32}}_{\text{4 guesses in } k_7} + \underbrace{2^{60.5} \times 2^{64}}_{\text{8 guesses in } k_7} + \underbrace{2^{36.5} \times 2^{80}}_{\text{All 10 guesses}} + \underbrace{2^{30.5} \times 2^{88}}_{\text{Discarding values}} \approx 2^{124.5}$$

one-round encryptions, so approximately $2^{124.5}/7 \approx 2^{121}$ AES encryptions. The memory complexity requires about 2^{109} bytes to store the discarded values.

4.4.4.2 Improved variants

The presented impossible differential attack from Bahrak and Aref has been improved in [LDKK08] by Lu et al., and later by Mala et al. in [MDRMH10].

First improvement

In [LDKK08], Lu, Dunkelman, Keller and Kim notice that the differential property of the AES S-Box (that we have recalled in Theorem 4.2) can be used to improve the key guesses. Namely, for a given ciphertext pair, we know the output difference so we can compute the differences in the last column of y_6 at the output of the last **SubBytes** layer. If we know the differences at its input in x_6 in the same 4 bytes, we can use Theorem 4.2 to deduce the values for the last column of x_6 and incidentally compute the bytes at positions $12, 9, 6, 3$ in k_7.

Therefore, in a precomputation phase, we exhaust the remaining $N_s \times 2^{-1}$ pairs of ciphertext (c_i, c_i') and for each, we guess the difference in the last column of x_6. It can only take $2^8 - 1$ values from the single difference in the column of z_5, so we get in average $2^8 - 1$ key suggestions for $k_7[12, 9, 6, 3]$ that we store in a table: $T[k_7[12, 9, 6, 3]] = (c_i, c_i')$. This whole procedure is performed in $S \times 2^{-1} \times 2^8$ operations, and allows to get the key suggestions in amortized cost 1 in this part of the attack.

In their paper, Lu et al. show that the number N_s of structures to use is reduced to $N_s \approx 2^{80.2}$, which makes an attack with data complexity $2^{80.2+32} = 2^{112.2}$. The main consequence of this is the smaller number of discarded key material per ciphertext pairs. As a minor improvement, they suggest to derive more impossible differentials by swapping positions of some differences, but nevertheless, too many key candidates are left after the attack, so that there is a final step of exhaustive search to recover the complete secret key.

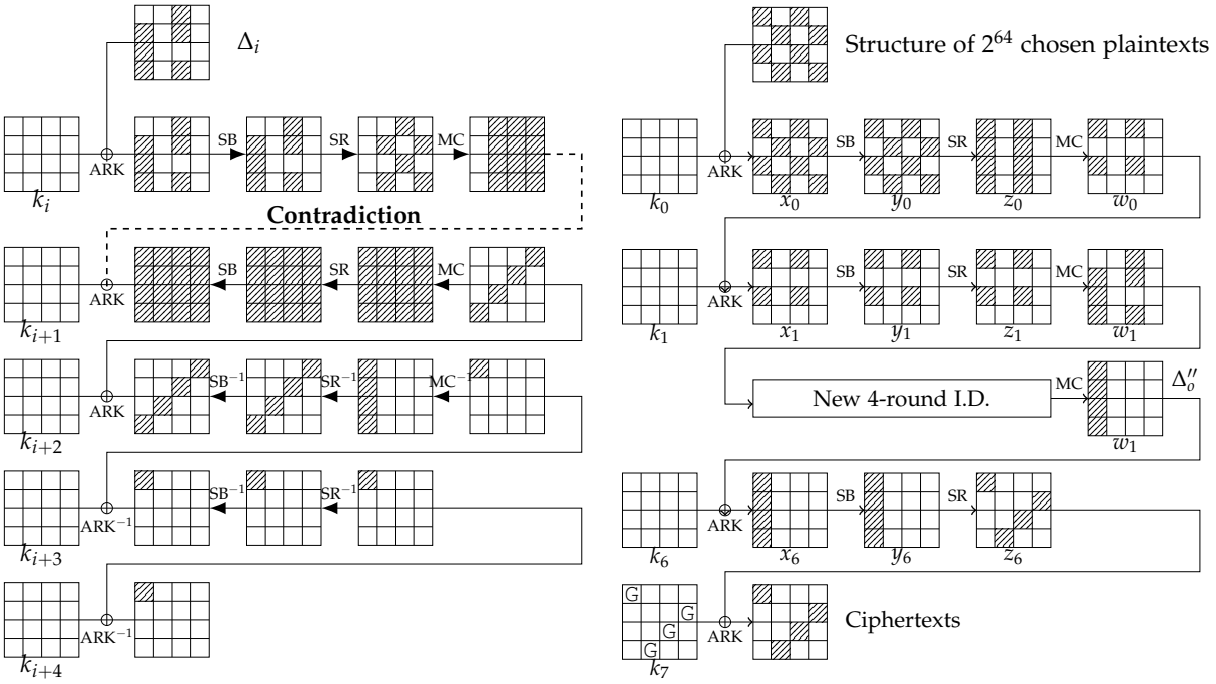
Additionally, they make an observation that uses the key schedule of AES-128 to reduce the number of key candidates among the successive applications of the modified impossible differentials used in the attack. They use the same key byte positions in different attacks to reduce the possible entropy of those bytes.

The time complexity of this modified version of the attack is equivalent to $2^{117.2}$ AES encryptions, and is dominated by the repetition of the simple attack using the multiple impossible differentials.

Second improvement

In [MDRMH10], Mala, Dakhilalian, Rijmen, and Modarres-Hashemi show that the strategy devised by Lu et al. using the key schedule of AES-128 can be adapted more efficiently with a single run of the attack, without repeating it with other impossible differentials. Their main point switches the order the two probability-one differential characteristics in the miss-in-the-middle construction of the impossible differential on 4 rounds (see Figure 4.20a).

This variant constructs N_s structures of chosen plaintexts with 2^{64} elements in each structure. The elements are such that diagonals 0 and 2 assume all the 2^{64} possible values while the others are constant. Similarly, the ciphertexts are efficiently filtered through the 96-bit filter imposed by



(a) New 4-round Impossible Differential (ID). **(b)** 7-round differential characteristic.

Figure 4.20: On the left, (a): Impossible differential characteristic on 4 rounds AES used in the most efficient impossible differential attack on AES-128 (to date). A hatched byte is an active byte and white bytes are inactive. On the right, (b): the full 7-round differential characteristic used in the attack with the (a) one plugged with $i = 2$.

the desired output difference pattern. Consequently, we are left with $N_s \times \binom{2^{64}}{2} \times 2^{-96} \approx N_s \times 2^{31}$ pairs.

Then, they also use the differential property of the AES S-Box stated in Theorem 4.2 in the same way as the previous attack from Lu et al.. Since they swapped the order of the two small characteristics, they use the same argument as Lu et al. to precompute the possible key candidates, but they perform this computation for the first subkey k_0. Since the difference in the main diagonal of y_0 can only take $(2^8 - 1)^2$ values from the two active differences in w_0, one store about 2^{16} suggestions for $k_0[0, 5, 10, 15]$ for each plaintext pairs. Consequently, for a single guess of $k_0[0, 5, 10, 15]$, one expect $N_s \times 2^{31+16-32} = N_s \times 2^{15}$ pairs of plaintexts to match the pattern of differences in $w_0[0, 1, 2, 3]$.

For each of the $N_s \times 2^{15}$ pairs, by precomputing the possible values for the second active diagonals, we can directly deduce the approximately 2^{16} corresponding values for $k_0[8, 13, 2, 7]$. Here, we know the values for bytes $k_0[0, 2, 5, 7, 8, 10, 13, 15]$, and we can use the key scheduling algorithm of AES-128 (see Figure 4.3a) to deduce the values of two more bytes in the next subkey, namely $k_1[0]$ and $k_0[2]$. With precomputed data over the second round and independently for the two active diagonals in x_1, we can deduce the values of $k_1[8]$ and $k_1[10]$, for each of the 4 possible positions for the inactive bytes in w_1: $(0, 10)$, $(1, 11)$, $(2, 8)$ or$(3, 9)$. The time complexity to get here amounts approximately to $2^{32} \times S \times 2^{15} \times 2^{16} \times 4$ computations, and we have $2^{16} \times 4$ suggestions for the 10 bytes $k_0[0, 2, 5, 7, 8, 10, 13, 15]$ and $k_1[0, 2, 8, 10]$.

From the other side, for each of the $N_s \times 2^{15}$ ciphertext pairs, we can apply Theorem 4.2 to get a suggestion for key bytes $k_7[0,7,10,13]$. Indeed, the difference Δ_o'' in x_6 can only take about $\binom{4}{2} \times (2^8 - 1) \approx 2^{10}$ values since we want the output difference $\mathbf{M}^{-1}(\Delta_o'')$ of the impossible differential to be active in a single byte. Therefore, we can precompute the possible values of $k_7[0,7,10,13]$ for a given output difference in the ciphertexts and get them in amortized cost 1. Hence, we get 2^{10} suggestions for those 4 bytes for each pair.

All in all, for each pair and from both sides we get $2^{16} \times 4 \times 2^{10}$ suggestions for the 14 independent bytes $k_0[0,2,5,7,8,10,13,15]$, $k_1[8,10]$ and $k_7[0,7,8,13]$ such that the input and output differences of the impossible differential of Figure 4.20a are verified. We deduce that all those $N_s \times 2^{15} \times 2^{16} \times 4 \times 2^{10}$ suggestions are wrong.

For a single pair among the $N_s \times 2^{15}$, each guess of $k_0[0,5,10,15]$ removes about 2^{28} values among the space of $2^{8\times(14-4)} = 2^{80}$ possible values for the targeted key bytes. The probability that a wrong value remains in the suggestions is therefore $1 - 2^{28}/2^{8\times 10} \approx 1 - 2^{-52}$. This means that about

$$2^{8\times 14}(1 - 2^{-52})^{N_s \times 2^{15}}$$

values for the 14 independent bytes remain after all the $N_s \times 2^{15}$ pairs have been processed.

We note that for each suggestion, the bytes suggested in k_0 allow to deduce more bytes from the key schedule of AES-128. Indeed, the key schedule equations in the AES-128 are strongly linear and we can deduce $k_0[4]$ and $k_0[6]$. Consequently, the entropy of the secret key is reduced from 128 bits to only 24 bits from all the linear equations between the known bytes of the subkeys.

There are several attacks that lie on the same curve depending on the choice of the parameters: we mention only the one chosen by [MDRMH10] which reaches a data complexity of $2^{106.2}$ chosen plaintexts with $N_s = 2^{42.2}$ initial structures, a time complexity equivalent to $2^{110.2}$ AES encryptions and a memory of $2^{90.2}$ AES states.

4.4.5 Related-key attacks

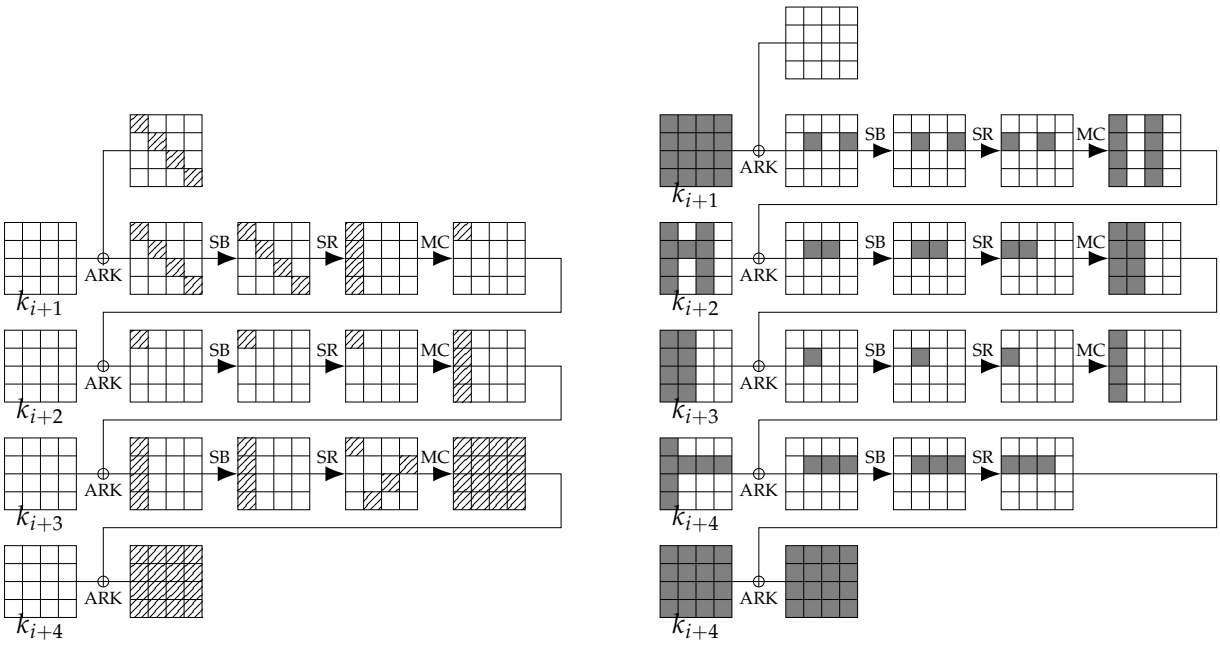
In [Bih93, Bih94], Eli Biham introduces the concept of related-key attacks, which is another model of differential cryptanalysis that allows the adversary to observe the encryption of different plaintexts under different keys. The set of keys is initially unknown to the adversary, but he knows that a certain mathematical relation holds between them. Like in the standard model, his goal is also to retrieve one or several of those keys.

All the three variants of the AES have been analyzed in this model: the full AES-192 and AES-256 are subject to related-key boomerang attacks while there exists a related-key boomerang attack on AES-128 reduced to 7 rounds. The results on AES-192 and AES-256 have started in [BDK05a] and [KHP07] where 10-round rectangle attacks were presented. The first result on the full 14 rounds of AES-256 has been published at CRYPTO 2009 by Biryukov, Khovratovich, and Nikolić in [BKN09]. Later, at ASIACRYPT 2009, Biryukov and Khovratovich have improved this by attacking the full 12 rounds of AES-192 in [BK09]. Finally, in [BN10] at EUROCRYPT 2010, Biryukov and Nikolić have developed an automatic tool to search for the best differential characteristics on AES and other block ciphers and have improved the results

on AES-192 and AES-256. Moreover, they give a 7-round related-key boomerang attack on AES-128.

4.4.5.1 Related-key boomerang attack on 7-round AES-128

The complete description of this attack has been done by Biryukov and Nikolić and can be found in [BN10]. As in a boomerang attack (see Section 3.4.3), they use two differential characteristics to cover the 7 rounds of the cipher: the first one with 3 rounds for the top part with no difference in the key, and the second one with 3 rounds for the bottom with two related-key keys, extended by one additional round in the end. The model assumes that the adversary can query the two oracles with the two related keys. The two differential characteristics used are represented on Figure 4.21.



(a) Top characteristic.

(b) Bottom characteristic.

Figure 4.21: Two differential characteristics used in the 7-round related-key boomerang attack on AES-128: top part in (a) and bottom part in (b). Hatched bytes have non-zero truncated difference, and gray bytes have known non-zero difference.

The first differential on Figure 4.21a is a truncated differential that holds with probability $4 \times 2^{-24} = 2^{-22}$ as the position of the active difference in the second round can be placed on four different bytes. The second one on Figure 4.21b is a fully instantiated differential characteristic that has 5 active S-Boxes in the three first rounds in the state part, and 1 active S-Box in the corresponding keys. This second characteristic is the result of the search conducted in [BN10] and in Chapter 6 as it is the best related-key differential characteristic for 4 rounds of AES-128. We note that by extending the 3-round characteristic by one round, the ciphertext difference only depends of the last 3 active difference in the state, and the active S-Box in the last subkey. Consequently, this difference can only take about $(2^7 - 1)^4 \approx 2^{28}$ since for a fixed input difference, the AES S-Box can only reach $2^7 - 1$ output differences.

The attack starts by requiring the encryption under key k of a structure of 2^{32} chosen plaintexts P_i that only differ by their main diagonal. This structure is thus transformed into ciphertexts C_i. By guessing 31 bits of key material allowing to bypass the last added round in the bottom characteristic, we can construct ciphertext pairs with correct output difference and query the corresponding plaintexts. If one pair has the correct truncated difference in the plaintext, the 31-bit guess is correct and we can finish by exhaustively search for the remaining ones. For a pair of ciphertext, the second characteristic has a probability of $2^{-6\times 3} \times 2^{-7} = 2^{-25}$ to be verified, assuming the 31-bit guess is correct. Indeed, the last added round is guessed and we then need to pass $1+2$ S-Boxes backwards with probability $p_{max} = 2^{-6}$ and the last two only need to have equal difference in the beginning for the two ciphertext pairs. Consequently, the two pairs conform to the characteristic with probability $(2^{-25})^2 = 2^{-50}$. Finally, the second pair for the top characteristic shares the same truncated difference in the plaintext with probability 2^{-24}. In total, we then find a boomerang quartet with probability $2^{-22-50-24} = 2^{-96}$, so that we need about 2^{33} structures of 2^{32} plaintexts before finding one right quartet. The data complexity of the attack consists of the encryption of $2^{32+33} = 2^{65}$ chosen plaintexts and $2^{31+65} = 2^{96}$ adaptively chosen ciphertexts.

This procedure finds a right quartet and incidentally retrieve 31 bits of the secret key, and we can finish the key recovery by performing an exhaustive search of the remaining 97 bits in 2^{97} computations.

4.4.6 Summary of all the attacks

In the following two tables, we report the major cryptanalytic results on the three versions of the AES. First, in Table 4.1 in the secret-key model, and then in Table 4.2, in the related-key model. As we consider the open-key model in Chapter 7, we recall there the results in that model.

Table 4.1: Best cryptanalytic results on reduced AES variants in the secret-key model.

Version	Rounds	Data	Time	Memory	Technique	Reference
128	6	2^{32}	2^{71}	2^{32}	Square	[DKR97]
	6	2^{32}	2^{48}	2^{32}	Partial Sums	[FKL+00]
	7	2^{128}	2^{120}	2^{64}	Herd	[FKL+00]
	7	2^{32}	2^{128}	2^{32}	Collision	[GM00]
	7	$2^{112.2}$	$2^{117.2}$	$2^{112.2}$	ID	[LDKK08]
	7	$2^{106.2}$	$2^{110.2}$	$2^{90.2}$	ID	[MDRMH10]
	7	2^{116}	2^{116}	2^{116}	MITM	[DKS10]
	7	$\mathbf{2^{105}}$	$\mathbf{2^{99}}$	$\mathbf{2^{90}}$	**MITM**	**Chapter 5**
	7	$\mathbf{2^{97}}$	$\mathbf{2^{99}}$	$\mathbf{2^{98}}$	**MITM**	**Chapter 5**
192	6	2^{32}	2^{71}	2^{32}	Square	[DKR97]
	6	2^{32}	2^{48}	2^{32}	Partial Sums	[FKL+00]
	7	2^{32}	2^{175}	2^{32}	Square	[DKR97]
	7	2^{32}	2^{155}	2^{32}	Partial Sums	[FKL+00]
	7	2^{128}	2^{120}	2^{64}	Herd	[FKL+00]
	7	2^{32}	2^{144}	2^{32}	Collision	[GM00]
	7	2^{116}	2^{116}	2^{116}	MITM	[DKS10]
	7	$\mathbf{2^{99}}$	$\mathbf{2^{99}}$	$\mathbf{2^{96}}$	**MITM**	**Chapter 5**
	8	2^{128}	2^{188}	2^{64}	Herd	[FKL+00]
	8	2^{113}	2^{172}	2^{129}	MITM	[DKS10]
	8	$\mathbf{2^{113}}$	$\mathbf{2^{172}}$	$\mathbf{2^{82}}$	**MITM**	**Chapter 5**
	8	$\mathbf{2^{107}}$	$\mathbf{2^{172}}$	$\mathbf{2^{96}}$	**MITM**	**Chapter 5**
256	6	2^{32}	2^{71}	2^{32}	Square	[DKR97]
	6	2^{32}	2^{48}	2^{32}	Partial Sums	[FKL+00]
	7	2^{32}	2^{191}	2^{32}	Square	[DKR97]
	7	2^{32}	2^{172}	2^{32}	Partial Sums	[FKL+00]
	7	2^{128}	2^{120}	2^{64}	Herd	[FKL+00]
	7	2^{32}	2^{144}	2^{32}	Collision	[GM00]
	7	2^{116}	2^{116}	2^{116}	MITM	[DKS10]
	7	$\mathbf{2^{99}}$	$\mathbf{2^{98}}$	$\mathbf{2^{96}}$	**MITM**	**Chapter 5**
	8	2^{128}	2^{204}	2^{64}	Herd	[FKL+00]
	8	2^{113}	2^{196}	2^{129}	MITM	[DKS10]
	8	$\mathbf{2^{113}}$	$\mathbf{2^{196}}$	$\mathbf{2^{82}}$	**MITM**	**Chapter 5**
	8	$\mathbf{2^{107}}$	$\mathbf{2^{196}}$	$\mathbf{2^{96}}$	**MITM**	**Chapter 5**
	9	$\mathbf{2^{120}}$	$\mathbf{2^{203}}$	$\mathbf{2^{203}}$	**MITM**	**Chapter 5**

CP: Chosen-plaintext. ID: Impossible Differential. MITM: Meet-in-the-Middle.

Table 4.2: Best cryptanalytic results on reduced AES variants in the related-key model.

Version	Rounds	Data	Time	Memory	Technique	Reference
128	5	2^{39}	2^{39}	2^{32}	Boomerang	[Bir04]
	6	2^{71}	2^{71}	2^{32}	Boomerang	[Bir04]
	7	2^{97}	2^{97}	2^{32}	Boomerang	[BN10]
192	9	2^{67}	2^{143}	2^{64}	Boomerang	[GL08]
	10	2^{125}	2^{182}	2^{64}	Rectangle	[KHP07]
	12	2^{123}	2^{176}	2^{48}	Boomerang	[BK09]
	12	2^{116}	2^{169}	2^{32}	Boomerang	[BN10]
256	9	2^{99}	2^{120}	2^{64}	Rectangle	[BDK05a, KHP07]
	10	2^{114}	2^{173}	2^{64}	Rectangle	[BDK05a, KHP07]
	14	2^{131}	2^{131}	2^{64}	Differential	[BKN09]
	14	$2^{99.5}$	$2^{99.5}$	2^{56}	Boomerang	[BK09]

CHAPTER 5

AES in the Secret-Key Model

Contents

5.1 A class of attacks against AES **90**
5.1.1 Initial attacks 90
5.1.2 Generalizations 90
5.1.3 Attack framework 92
5.1.4 Improvements 94
5.2 New attacks on 7-round AES **95**
5.2.1 Generalities 95
5.2.2 Efficient tabulation 95
5.2.3 A simple attack 98
5.2.4 Efficient Attack 100
5.2.5 Key recovery 102
5.3 Extensions to 8 and 9 rounds **104**
5.3.1 Attack on 8-round AES-192 104
5.3.2 Attack on 8-round AES-256 108
5.3.3 Attack on 9-round AES-256 109

In this chapter, we are interested in the security of the AES versions in the classical setting. Namely, we have access to an oracle that encrypts and decrypts message blocks of our choice with a secret key. Our goal as an adversary is to recover the secret key of k bits faster than the exhaustive search; i.e. with (significantly) less than 2^k encryptions.

The content of this chapter is largely inspired from the article [DFJ13] published at EUROCRYPT 2013 and co-authored with Patrick Derbez and Pierre-Alain Fouque. The research line behind this work goes back to the square attack (Section 4.4.1) and the Gilbert and Minier attack (Section 4.4.3) on `Rijndael`, but most importantly to the work by Demirci and Selçuk in [DS08], and Dunkelman, Keller and Shamir in [DKS10]. To provide a full understanding of our work, we begin by giving an introduction on these pioneering cryptanalytic results on reduced variants of the AES in Section 5.1, and we continue by describing our improvements in the next sections Section 5.2 and Section 5.3.

5.1 A class of attacks against AES

5.1.1 Initial attacks

We recall that the first attack on AES is the SQUARE attack, proposed by Daemen, Knudsen and Rijmen on the SQUARE block cipher [DKR97]. If we encrypt a δ-set (Definition 4.2) by 3 rounds of `Rijndael`, the sum of each byte of the 256 ciphertexts equals zero. This distinguishing property can be used to mount efficient attacks up to 6 rounds. The first attack has a time complexity of 2^{72} encryptions and requires 2^{32} messages, and it has been improved by Ferguson et al. to 2^{46} operations in [FKL+00]. We refer to Chapter 4 for a complete description of these attacks.

Then, Gilbert and Minier show in [GM00] that this property can be made more precise using functions of the active byte, which allows to build a distinguisher on 3 rounds. The main idea is to consider the set of functions mapping one active byte to one byte after 3 rounds. This set depends on 9 one-byte parameters so that the whole set can be described using a table of 2^{72} entries of a 256-byte sequence $(f(0), \ldots, f(255))$. Their attack allows to break 7 rounds of AES with a marginal time complexity over exhaustive search.

5.1.2 Generalizations

This idea has been generalized at FSE 2008 by Demirci and Selçuk in [DS08] using meet-in-the-middle techniques, whereas Gilbert and Minier used collision between the functions. More specifically, they show that on 4 rounds, the value of each byte of the ciphertext can be described by a function of the active byte parameterized by 25 in [DS08] and 24 8-bit parameters in [DTCB09] (see Figure 5.1). The last improvement is due to the observation that the 25th

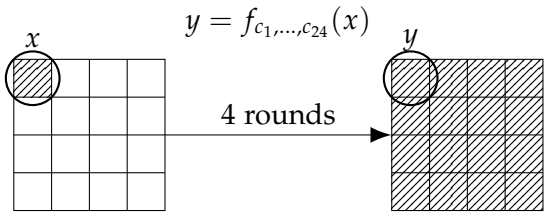


Figure 5.1: Byte y at the output can be expressed as a function of byte x at the input that depends on the 24 byte parameters $c_1, \ldots, c_{24}$.

parameter is a key byte which is constant for all functions. Consequently, by considering

$$\Big(f(0) - f(0), f(1) - f(0), \ldots, f(255) - f(0)\Big)$$

we can use only 24 parameters. The main drawback of the meet-in-the-middle attack is the large memory requirement. Indeed, the basic attack only works for the 256-bit version of the AES and then Demirci and Selçuk have to use a time/memory tradeoff to extend the attack for the 192-bit AES version.

At ASIACRYPT 2010, Dunkelman, Keller and Shamir develop in [DKS10] many new ideas to solve the memory problems of the Demirci and Selçuk attacks. First of all, they show

that instead of storing the whole sequence, we can only store the associated multiset, i.e. the unordered sequence with multiplicity rather than the ordered sequence. This reduces the table by a factor 4 and spares one guess during the attack. The second and main idea is the *differential enumeration* which allows to reduce the number of parameters that describe the set of functions from 24 to 16. However, to reduce this number, they rely on a special property on a truncated differential characteristic. The idea consists in using a truncated differential characteristic whose probability is not too small. The property of this characteristic is that the set of functions from one state to the state after 4 rounds can only take a restricted number of values, which is much smaller than the number of all functions. The direct consequence is an increase of the amount of needed data, but the memory requirement is reduced to 2^{128} and the same analysis also applies to the 128-bit version. However, even though many tradeoffs could be used, this attack is not better than the best attack at that time, which is the impossible differential attack from Lu, Dunkelman, Keller and Kim [LDKK08] (see Chapter 4 for its description).

The attack devised by Dunkelman, Keller and Shamir uses ideas from the classes of differential and meet-in-the-middle attacks. We refer to Section 3.2.1 for a more complete overview of key-recovery differential attacks, and Section 1.4.5.1 for an introduction to meet-in-the-middle attacks.

In the first stage of differential attacks, we need to find a differential characteristic with high or low probability covering many rounds (see Section 3.2). Then, in the online stage, the adversary asks for the encryption of many pairs: for each pair, he tries to decrypt by guessing the last subkey and if the differential characteristic is followed, then the adversary increases the counter of the associated subkey. If the probability of the characteristic is high enough, then the counter corresponding to the right secret-key would be among the higher counters. In some case, it is also possible to add some rounds at the beginning by guessing part of the first subkeys.

In [DKS10], Dunkelman et al. propose a novel differential attack. Instead of increasing a counter once a pair is found, the adversary uses another test to eliminate the wrong guesses of the first or last subkeys. This test decides with probability one whether the middle rounds are covered with the differential. The idea is that the middle rounds follow a part of the differential and the function f that associates each byte of the input state to one byte of the output state can be stored efficiently. Demirci and Selçuk propose to store in a table the function with no differential characteristic, which turns out to be much larger that this one. Consequently, in Dunkelman et al.'s attack, the adversary guesses key material in the first and last subkeys and looks for a pair that follows the beginning and last rounds of the differential characteristic. Once such a pair is found, the adversary takes one of the messages that follows the characteristic and constructs a structure to encrypt which is related to a δ-set for the intermediate rounds. From the encryption of this set, he can decrypt the last rounds and check whether the encryption of this δ-set belongs to the table. If this is the case, then the part of the first and last subkeys are correct and an exhaustive search on the other parts of the key allows to find the full key.

To construct the table, the idea is similar to the attack. We need to find a pair of messages that satisfies the truncated differential characteristic. Then, we take one message in the pair and we compute the function f. Dunkelman et al. use a rebound technique to find the pair that follows the characteristic, and in our work, we adapt the strategy described in Section 7.3.1 for the chosen-key distinguisher on AES reduced to 7 rounds.

5.1.3 Attack framework

We present here a unified view of the previously known meet-in-the-middle (MITM) attacks on AES [GM00, DS08, DKS10], where n rounds of the block cipher can be split into three consecutive parts of n_1, n_2 and n_3 rounds, $n = n_1 + n_2 + n_3$, such that a particular set of messages may verify a certain property that we denote ★ in the sequel in the n_2 middle rounds (Figure 5.2).

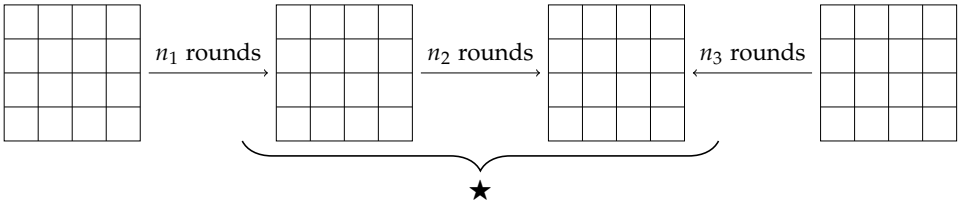


Figure 5.2: General scheme of the meet-in-the-middle attack on AES, where some messages in the middle rounds may verify a certain ★ property used to perform the meet-in-the-middle.

The general attack uses three successive steps:

Precomputation phase

1. In this phase, we build a lookup table T containing all the possible sequences constructed from a δ-set such that one message verifies the ★ property.

Online phase

2. Then, in the online phase, we need to identify a δ-set containing a message m verifying the wanted property.
3. Finally, we partially decrypt the associated δ-set through the last n_3 rounds and check whether it belongs to T.

The two steps of the online phase require to guess some key bytes while the goal of this attack is to filter some of their values. In the best case, only the right ones should pass the test.

We now precise how the Demirci and Selçuk attack and the Dunkelman, Keller and Shamir attack fit into this framework.

5.1.3.1 Attack by Demirci and Selçuk

The starting point is to consider the set of functions

$$f : \{0,1\}^8 \to \{0,1\}^8$$

that maps a byte of a δ-set to another byte of the state after four AES rounds. A convenient way is to view f as an ordered byte sequence $(f(0), \ldots, f(255))$ so that it can be represented by 256 bytes. The crucial observation made by the generalizing Gilbert and Minier attack is that this set is tiny since it can be described using 25 byte-parameters ($2^{25 \cdot 8} = 2^{200}$) compared with the set of all functions of this type that counts as many as $2^{8 \cdot 2^8} = 2^{2048}$ elements. Considering the differences

$$\Big(f(0) - f(0), f(1) - f(0), \ldots, f(255) - f(0)\Big)$$

rather than values, the set of functions can be described by 24 parameters. Dunkelman et al. identify these parameters as follows:

- the full state x_3 of message 0,
- four bytes of state x_2 of message 0,
- four bytes of subkey k_3.

The four bytes of the state x_2 only depend on the column of z_1 where the active byte of the δ-set is located; for instance, if it is column 0, then those bytes are $x_2[0,1,2,3]$. Similarly, the four bytes of k_3 depend on the column of x_5 where the byte we want to determine is located; as an example, if it is column 0, then those bytes are $k_3[0,5,10,15]$.

In their attacks [DS08], Demirci and Selçuk use the ★ property that does not filter any message. Consequently, they do not require to identify a particular message m. The data complexity of their basic attack is very small and around 2^{32} chosen plaintexts. However, since there is no particular property, the size of the table T is very large and the basic attack only works for the AES-256. To mount an attack on the AES-192, they consider some time/memory tradeoff. More precisely, the table T does not contain all the possible states, but only a fraction α. Consequently, a specific δ-set may not be in the table T, so that we have to wait for this event and redo the attack $O(1/\alpha)$ times on average. The attack becomes probabilistic and the memory requirement makes the attack possible for AES-192. The consequence of this advanced version, which also works for AES-256, is that the amount of data increases a lot. The time and memory requirements of the precomputation phase are due to the construction of table T that contains messages for the $n_2 = 4$ middle rounds, which counts as many as $2^{8 \cdot 24} = 2^{192}$ ordered sequences of 256 bytes.

Finally, it is possible to remove from each function some output values. Since we know that these functions can be described by the key of 24 or 32 bytes, one can reduce T by a factor 10 or 8 by storing only the first differences. Such an observation has been used by Wei et al. in [WLH11].

5.1.3.2 Attack by Dunkelman, Keller and Shamir

In [DKS10], Dunkelman, Keller and Shamir introduced two new improvements to further reduce the memory complexity of [DS08]. The first one uses multisets, behaving as unordered sequences, and despite the information lost, the authors show it is still enough so that the attack succeeds. The second improvement uses a particular 4-round differential characteristic (Figure 5.3) to reduce the size of the precomputed lookup table T, at the expense of trying more pairs of messages to expect at least one to conform to the truncated characteristic.

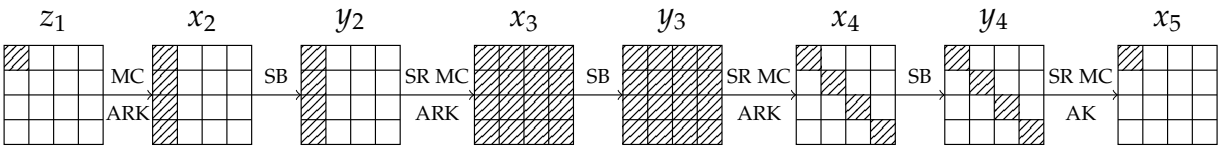


Figure 5.3: The four middle rounds used in the 7-round attack from [DKS10]. Dashed bytes are active, others inactive.

The main idea of the differential characteristic is to fix the values of as many state-bytes

as possible to a constant. Assume now we have a message m such that we have a pair (m, m') that satisfies the whole 7-round differential characteristic and our goal is to recover the key. Contrary to classical differential attacks, where the adversary guesses some bytes of the last subkey and eliminates the wrong guess, the smart idea of Dunkelman et al. is to use a table to recover the right key more efficiently. Usually, differential attacks do not use memory to recover the key or to find the right pair. The attack principle consists in constructing the δ-set from m which can be made since we already have to guess some key bytes to check if the pair (m, m') has followed the right differential characteristic. Then, the table allows to identify the right key from the encryption of the δ-set.

It is now easy to see that the differential characteristic can be described using only 16 bytes. The states x_3 and y_3 can only take 2^{32} possible differences each, so that the number of solutions for these two states is 2^{64}. We also have the 4 key-bytes of u_2 and the 4 key-bytes of k_3 corresponding to the active bytes of Figure 5.3 in states z_2 and x_4.

5.1.4 Improvements

Dunkelman et al. show that by using a particular 4-round differential characteristic with a not too small probability, the active states in the middle of the characteristic can only take 2^{64} values. In their characteristic, they also need to consider the same 8 key bytes[1] as Demirci and Selçuk. They claim:

> “In order to reduce the size of the precomputed table, we would like to choose the δ-set such that several of these parameters will equal to predetermined constants. Of course, the key bytes are not known to the adversary and thus cannot be “replaced” by such constants.”

In this chapter, we show that it is possible to enumerate the whole set of solutions more efficiently than by taking all the values for the key bytes such that every value of these bytes are possible. We show that the whole set can take only 2^{80} values with this efficient enumeration technique. Of course, it might be possible to improve this result to 2^{64} but not any further since the key bytes may take all the 2^{64} possible values. Using the same ideas, we show that it is possible to have an efficient enumeration for a 5-round differential characteristic which allows us to mount an attack on 9 rounds for AES-256. The bottleneck of the attack is no longer the memory, but the time and data complexities.

The main technical contribution of this work shows that the number of parameters describing the functions can be further reduced from 16 to 10. The resulting attack on AES-128 is now more efficient than the impossible differential attack by Lu, Dunkelman, Keller and Kim from [LDKK08] and is the most efficient attack on this variant of the AES.

[1] Those are $u_2[0, 7, 10, 13]$ and $k_3[0, 5, 10, 15]$.

5.2 New attacks on 7-round AES

5.2.1 Generalities

In the following sections, we use δ-sets as input structured messages, and we compress this representation by introducing the associated multiset as described in the following Definition 5.1. We provide an almost optimal way to represent them in Proposition 5.1.

Definition 5.1 (Multisets of bytes). A multiset generalizes the set concept by allowing elements to appear more than once. A multiset is a set combined by multiplicity for each of its elements. A set can be written as a multiset where all the elements have multiplicity one.

Proposition 5.1. *We can represent a multiset of 256 bytes on* 512 *bits.*

Proof. From the point of view of information theory, as there are about

$$\left(\!\!\binom{2^8}{2^8}\!\!\right) = \binom{2^8+2^8-1}{2^8} \approx 2^{506.17}$$

multisets of 256 elements from $\mathrm{GF}(2^8)$, we are able to represent them on 512 bits. Here is one way of doing it for a given multiset M. In the sequel, we consider that $M = \{x_1^{n_1}, \ldots, x_m^{n_m}\}$, with $\sum_{i=1}^{m} n_i = 256$, that we may represent by

$$\underbrace{x_1\, x_1\, x_1\, x_1}_{n_1} \,\Big|\, \underbrace{x_2\, x_2\, x_2}_{n_2} \,\Big|\, \ldots \,\Big|\, \underbrace{x_m\, x_m\, x_m\, x_m\, x_m}_{n_m}, \tag{5.1}$$

where the distinct elements are the m elements x_i, which appear each with multiplicity n_i. In M, the order of the elements is undetermined.

Consider the set $S = \{x_1, \ldots, x_m\}$ deduced from M by deleting any repetition of element in M. As there are at most 256 elements in S, we can encode whether $e \in \mathrm{GF}(2^8)$ belongs to S in a 256-bit number s by a 1-bit flag at the position e seen as an index in $[0, \ldots, 255]$ in s. Then, to express the repetition of element, we sort M using the natural order in the integers and consider the sequence of multiplicity of each distinct element: if $x_1 < \cdots < x_m$, then we consider the sequence $n_1, \ldots, n_m$. We use a second 256-bit number t to store the sequence $(\sum_{j=1}^{i} n_j)_i$ seen as indexes in t, which actually encodes the positions of the vertical separators in the multiset representation of Equation 5.1. The 512-bit element (s, t) then represents the multiset M. ■

Finally, we note that we measure memory complexities of our attacks in number of 128-bit AES blocks and time complexities in terms of AES encryptions.

In the following, we use the notation ExhaustiveSearch to refer to the final exhaustive search for the remaining key bytes when the attack is finished. Indeed, the attack we describe recovers *some* bytes of the subkeys, and to complete to key recovery and actually find the missing bytes, we perform a basic exhaustive search implementing by this procedure.

5.2.2 Efficient tabulation

As in the previous results, our attack also uses a large memory lookup table constructed in the precomputation phase, and used in the online phase. Dunkelman, Keller and Shamir

showed that if a message m belongs to a pair of states conforming to the truncated differential characteristic of Figure 5.3, then the multiset of differences $\Delta x_5[0]$ obtained from the δ-set constructed from m in x_1 can only take 2^{128} values, because 16 of the 24 parameters used to build the multisets can take only 2^{64} values instead of 2^{128}. We make the following Proposition 5.2 that reduces the size of the table by a factor 2^{48}.

Proposition 5.2. *If a message m belongs to a pair of states conforming to the truncated differential characteristic of Figure 5.3, then the multiset of differences $\Delta x_5[0]$ obtained from the δ-set constructed from m in x_1 can only take 2^{80} values. More precisely, the 24 parameters (which are state bytes of m) can take only 2^{80} values in that case. Conversely, for each of these 2^{80} values, there exists a tuple (m, m', k) such that m is set to the chosen value and the pair (m, m') follows the truncated characteristic.*

Proof. The proof uses rebound-like arguments borrowed from the hash function cryptanalysis domain [MRST09]. Let (m, m') be a right pair. We show in the following how the knowledge of 10 particular bytes restricts the values of the 24 parameters used to construct the multisets, namely:

$$x_2[0,1,2,3],\ x_3[0,\ldots,15],\ x_4[0,5,10,15]. \tag{5.2}$$

In the sequel, we use the state names mentioned in Figure 5.4. The 10 bytes

$$\Delta z_1[0],\ x_2[0,1,2,3],\quad \Delta w_4[0],\ z_4[0,1,2,3]. \tag{5.3}$$

can take as many as 2^{80} possible values, and for each of them, we can determine the values of all the differences shown on Figure 5.4: linearly in x_2, applying the S-Box to reach y_2, linearly for x_3 and similarly in the other direction starting from z_4. By the differential property of the AES S-Box (Theorem 4.2), we get on average one value for each of the 16 bytes of state x_3. In fact, only 2^{64} values of the 10 bytes lead to a solution for x_3 but for each value, there are 2^{16} solutions for x_3. From the known values around the two **AddRoundKey** layers of rounds 3 and 4, this suggests four bytes of the equivalent subkey $u_2 = \mathsf{MC}^{-1}(k_2)$ and four others in subkey k_3: those are $u_2[0]$, $u_2[7]$, $u_2[10]$, $u_2[13]$ and $k_3[0]$, $k_3[5]$, $k_3[10]$, $k_3[15]$; they are marked by a bullet (•) in Figure 5.4.

The converse is now trivial: the only difficulty is to prove that for each value of the 8 key bytes, there exists a corresponding master key. This actually gives a chosen-key distinguisher for 7 rounds of AES, as it has been done in [DFJ12a] (see Section 7.3).

To construct the multiset for each of the 2^{80} possible choices for the 10 bytes from Equation 5.3, we consider all the $2^8 - 1$ possible values for the difference $\Delta y_1[0]$, and propagate them until x_5. This leads to a multiset of $2^8 - 1$ differences in $\Delta x_5[0]$. Finally, as the AES S-Box behaves as a permutation over $\mathrm{GF}(2^8)$, the sequence in $\Delta y_1[0]$ allows to derive the sequence in $\Delta x_1[0]$.

Note that in the present case where there is a single byte of difference between m and m' in the state x_1, both messages belongs to the same δ-set. This does not hold if we consider more active bytes as we do in the following Section 5.3. We describe in an algorithmic manner this proof in Algorithm 5.1. ■

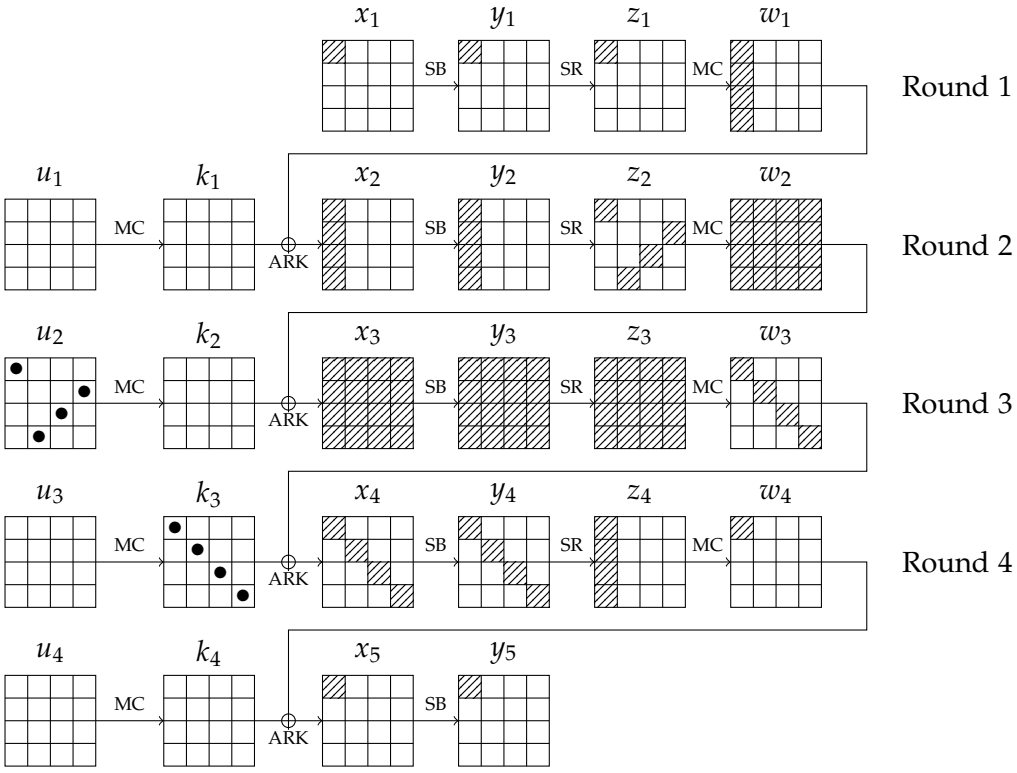


Figure 5.4: Truncated differential characteristic used in the middle of the 7-round attacks on AES. A hatched byte denotes a non-zero difference, whereas a white cell has no difference.

Algorithm 5.1 – Construction of the tables.

function ConstructTable(i, j)
  $b_i \leftarrow i - 4(i \mod 4) \mod 16$. ▷ *Retrieving the right positions*
  $c_i \leftarrow \lfloor b_i/4 \rfloor$. ▷ *because of the* **ShiftRows**.
  $c_j \leftarrow \lfloor j/4 \rfloor$.
  Empty a lookup table T.
  Guess values of the 5 bytes $\Delta z_1[b_i]$, $x_2[4c_i]$, $x_2[4c_i+1]$, $x_2[4c_i+2]$, $x_2[4c_i+3]$.
    Deduce differences in Δx_3.
    Guess values of the 5 bytes $\Delta w_4[j]$, $z_4[4c_j]$, $z_4[4c_j+1]$, $z_4[4c_j+2]$, $z_4[4c_j+3]$.
      Deduce differences in Δy_3.
      Use the property of the AES S-Box to deduce the values in x_3 and x'_3.
      Deduce $\mathrm{SR}^{-1}(u_2)[4c_i]$, $\mathrm{SR}^{-1}(u_2)[4c_i+1]$, $\mathrm{SR}^{-1}(u_2)[4c_i+2]$, $\mathrm{SR}^{-1}(u_2)[4c_i+3]$.
      Deduce $\mathrm{SR}(k_3)[4c_j]$, $\mathrm{SR}(k_3)[4c_j+1]$, $\mathrm{SR}(k_3)[4c_j+2]$, $\mathrm{SR}(k_3)[4c_j+3]$.
      Empty a multiset M.
      for all the differences $\Delta z_1[b_i]$ **do**
        Obtain a column x_2, and then a state x_3.
        Add $\Delta x_5[j]$ to M.
      Add M to the lookup table T.
  return T of size $\approx 2^{80}$.

5.2.3 A simple attack

5.2.3.1 Precomputation phase

In the precomputation phase of the attack, we build the lookup table that contains the 2^{80} multisets for difference Δx_5 by following the proof of Proposition 5.2 and Algorithm 5.1. This step is performed by first iterating on the 2^{80} possible values for the 10 bytes of Equation 5.3 and for each of them, we deduce the possible values of the 24 original parameters. Then, for each of them, we construct the multiset of $2^8 - 1$ differences. Using the differential property of the AES S-Box (Theorem 4.2), we can count exactly the number of multisets that are computed:

$$2^{80} \times \left(4 \times \frac{2^8 - 1}{(2^8 - 1)^2} + 2 \times \frac{(2^8 - 1)(2^7 - 1 - 1)}{(2^8 - 1)^2}\right)^{16} \approx 2^{80.09}. \tag{5.4}$$

Finally, the lookup table of the $2^{80.09}$ possible multisets that we simplify to 2^{80} requires about 2^{82} 128-bit blocks to be stored. To construct the table, we have to perform 2^{80} partial encryptions on 256 messages, which we estimate to be equivalent to 2^{84} encryptions.

5.2.3.2 Online phase

The online phase splits into three parts: the first one finds pairs of messages that conform to the truncated differential characteristic of Figure 5.5, which embeds the previous 4-round characteristic in the middle rounds (round 1 to round 5). The second step uses the found pairs to create a δ-set, test them against the precomputed table and retrieve the secret key in a final phase.

To generate one pair of messages conforming to the 7 full-rounds of this characteristic where there are only four active bytes in both the plaintext and the ciphertext differences, we prepare a structure of 2^{32} plaintexts where the diagonal assumes all the possible 2^{32} values, and the remaining 12 bytes are fixed to some constants. Hence, each of the $2^{32} \times (2^{32} - 1)/2 \approx 2^{63}$ pairs we can generate satisfies the plaintext difference. Among the 2^{63} corresponding ciphertext pairs, we expect $2^{63} \cdot 2^{-96} = 2^{-33}$ to verify the truncated difference pattern. Finding *one* such pair then requires 2^{33} structures of 2^{32} messages and $2^{32+33} = 2^{65}$ encryptions under the secret key. Using this secret key, the probability that the whole truncated characteristic of Figure 5.5 is verified is $2^{-2\times3\times8} = 2^{-48}$ because of the two $4 \to 1$ transitions in the **MixColumns** of rounds 0 and 5. By repeating the previous procedure to find 2^{48} pairs, one is thus expected to verify the full 7-round characteristic.

All in all, we ask the encryptions of $2^{48+65} = 2^{113}$ messages to find 2^{48} pairs of messages. Note that we do not have to examine each pair in order to find the right one. Indeed, if a pair verifies the full 7-round characteristic, then the ciphertext difference has only four active bytes. Thus, we can store the structures in a hash table indexed by the 12 inactive bytes to get the right pairs in average time of one computation.

For each of the 2^{48} pairs, we get $2^{8\times(8-2\times3)} \cdot 2^8 = 2^{24}$ suggestions for the 9 key bytes marked by $\star$ in Figure 5.5:

$$k_{-1}[0, 5, 10, 15],\ u_5[0],\ u_6[0, 7, 10, 13]. \tag{5.5}$$

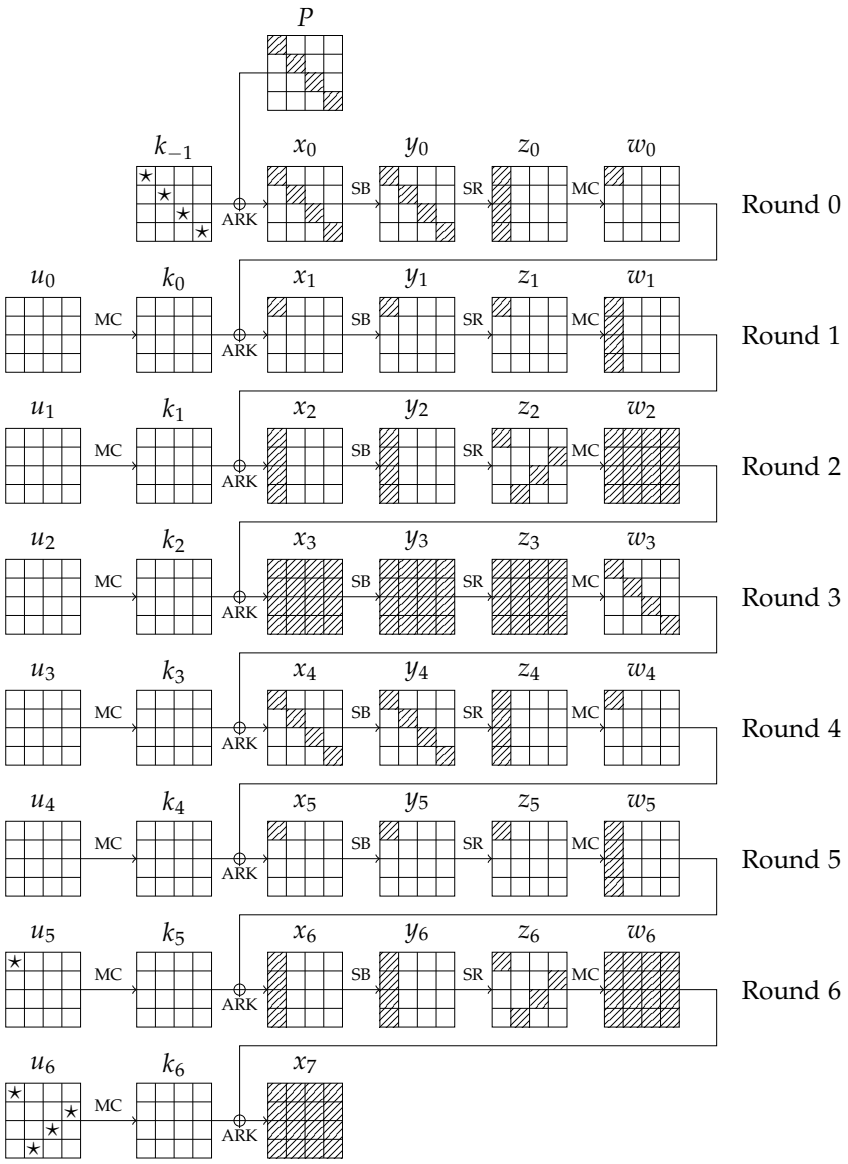


Figure 5.5: Complete 7-round truncated differential characteristic used in the simple attack.

Indeed, there are 2^8 possibilities for the bytes from k_{-1} since the pair of diagonals in x_0 need to be active only in w_0 after the **MixColumns** operation. Among the 2^{32} possible values for those bytes, only $2^{32} \times 2^{-24} = 2^8$ verify the truncated pattern. The same reasoning applies for $u_6[0, 7, 10, 13]$, and the last byte $u_5[0]$ can take all the 2^8 values.

For all the 2^{24} possibilities, we construct a δ-set to use the precomputed table. To do so, we partially encrypt the diagonal of one message, using the four known bytes from k_{-1} and consider the $2^8 - 1$ possible non-zero differences for $\Delta x_1[0]$. This gives one set of 2^8 plaintexts, whose corresponding ciphertexts may be partially decrypted using the four known bytes from u_6 and the one from u_5. Once decrypted, we can construct the multiset of differences for Δx_5 and check if it lies in the precomputed lookup table. If not, we can discard the subkey with certainty. On the other hand, the probability for a wrong guess to pass this test is smaller than

$2^{80} \cdot 2^{-467.6} = 2^{-387.6}$ so, as we try at most $2^{48} \cdot 2^{24} = 2^{72}$ multisets, only the right subkey should verify the test. Note that the probability is $2^{-467.6}$ (and not $2^{-506.17}$) because the number of ordered sequences associated to a multiset is not constant.

We summarize the above description in the following Algorithm 5.2, where the initial call to the function ConstructTable$(0,0)$ constructs the lookup table for Δx_1 and Δx_5 both at position zero (Figure 5.4) and is defined in Algorithm 5.1.

Algorithm 5.2 – A simple attack on 7-round AES.

```
1:  function SimpleAttack
2:    T_{0,0} ← ConstructTable(0,0).                    ▷ Algorithm 5.1
3:    while true do                                     ▷ 2^81 times on average
4:      Ask for a structure S of 2^32 plaintexts P_m
          where bytes in diagonals 0 assume all values.
5:      Empty a hash table T of list of plaintexts.
6:      for all corresponding ciphertexts C_m do
7:        index ← MC^{-1}(C_m)[1,2,3,4,5,6,8,9,11,12,14,15].
8:        for all P ∈ T[index] do
9:          Consider the pair (P, P_m).                 ▷ ∼ 2^-33 pairs
10:         for all k_{-1}[0,5,10,15] s.t. Δw_0[1,2,3] = 0 do   ▷ ∼ 2^8 times
11:           Construct δ-set D from P.                 ▷ D ∈ S
12:           for all u_6[0,7,10,13] s.t. Δz_5[1,2,3] = 0 do
13:             Decrypt column 0 of x_6 for D.
14:             for all u_5[0] do                       ▷ 2^8 times
15:               Decrypt byte 0 of x_5 for D.
16:               Construct multiset M of Δx_5.
17:               if M ∈ T_{0,0} then
18:                 return ExhaustiveSearch()
19:       T[index] ← T[index] ∪ {P_m}.
```

To evaluate the complexity of the online phase of the simple attack, we count the number of AES encryptions. First, we ask the encryption of 2^{113} chosen plaintexts, so that the time complexity for that step is already 2^{113} encryptions. Then, for each of the 2^{48} found pairs, we perform 2^{24} partial encryptions/decryptions of a δ-set. We evaluate the time complexity of this part to $2^{48+24+8} \cdot 2^{-5} = 2^{75}$ encryptions since we can do the computations in a good ordering as shown in Algorithm 5.2.

All in all, the time complexity is dominated by the 2^{113} initial encryptions, the data complexity equals 2^{113} chosen plaintexts, and the memory complexity is 2^{82} since it requires to store 2^{80} multisets.

5.2.4 Efficient Attack

Unlike the previous attacks where the bottleneck complexity is the memory, our attack uses a smaller table which makes the time complexity to find the pairs the dominating one. Therefore,

we would like to decrease the time spent in that phase. The natural idea is to find a new property ★ for the four middle rounds that can be checked more efficiently.

To do so, we reuse the idea of Dunkelman et al. from [DKS10], which adds an active byte in the second round of the differential characteristic. The sequence of active bytes becomes:

$$8 \xrightarrow{R_0} 2 \xrightarrow{R_1} 4 \xrightarrow{R_2} 16 \xrightarrow{R_3} 4 \xrightarrow{R_4} 1 \xrightarrow{R_5} 4 \xrightarrow{R_6} 16, \tag{5.6}$$

with the constraint that the two active bytes of the second round belong to the same diagonal to be transformed in a column in the next round (see Figure 5.6).

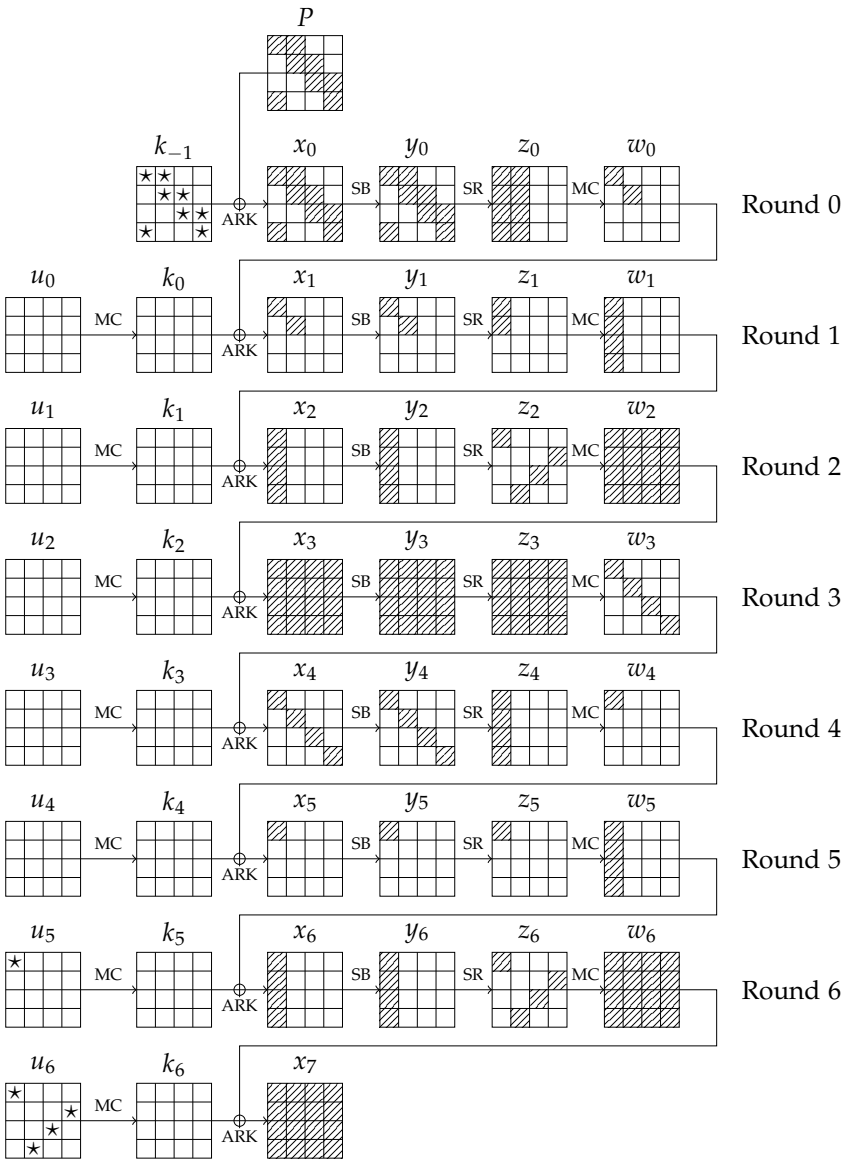


Figure 5.6: Complete 7-round truncated differential characteristic used in the efficient attack.

As a consequence, it is now easier to find pairs conforming to that truncated differential characteristic. Indeed, the size of the structure of plaintexts may take as many as 2^{64} different values, so that we can construct at most $2^{64} \cdot (2^{64} - 1)/2 = 2^{127}$ pairs from each structure.

Therefore, it is enough to ask the encryption of $2^{8\cdot3\cdot3}/2^{127-8\cdot12} = 2^{41}$ structures to get 2^{72} pairs with the desired output difference pattern, and expect one to conform to the 7-round characteristic of Figure 5.6.

Consequently in this new setting, we only need 2^{105} chosen plaintexts. In return, the number of pairs that the adversary has to consider is increased by a factor 2^{24} and so is the time complexity. Furthermore, we now need 11 parameters to generate the 24 parameters of the precomputed table, increasing the memory requirement by a factor 2^8. These parameters are the previous 10 ones and the difference in the second active byte of z_A.

All in all, the time complexity of this attack is $2^{75+24} = 2^{99}$ encryptions, the data complexity is 2^{105} chosen plaintexts and the memory requirement is $2^{82+8} = 2^{90}$ 128-bit blocks.

Note that the time spent on one pair is the same for both the simple attack and the efficient one. Indeed, let K be the key bytes needed to construct the multiset. We suppose that we have a set of pairs such that one follows the differential. To find it, and incidentally some key-byte values, we proceed as follows: for each pair (m, m'), enumerate all possible values of K such that (m, m', K) have a non-zero probability to follow the differential. For each of them, construct the corresponding multiset from m or m'. If it belongs to the table, then we expect that it follows the differential characteristic since the table has been constructed that way. Otherwise, we know with probability 1 that either the pair (m, m') does not satisfy the characteristic, or the guessed value from K is wrong.

Assuming that the bytes 0, 3, 4, 5, 9, 10, 14 and 15 of diagonals 0 and 1 of the structure of plaintexts range over all the values, the two differences in the first state of the second round can take four different positions: $(0,5)$, $(1,6)$, $(2,7)$ and $(3,4)$. Similarly, the position of the active byte in the penultimate round is not constrained; it can be placed anywhere on the 16 positions. We can also consider the opposite: one active byte at the beginning, and two active bytes in the end. These possibilities actually define tweaked versions of the property ★ and allows to trade some time for memory: with less data, we can check more tables for the same final probability of success. Namely, by storing $4 \times 16 + \binom{4}{2} \times 4 = 2^8$ tables to cover all the cases by adapting the proof of Proposition 5.2, the encryption of $2^{41}/2^8 = 2^{33}$ structures of 2^{64} plaintexts suffices to expect a hit in one of the 2^8 tables. Therefore, the memory complexity reaches 2^{98} AES blocks and the time complexity remains unchanged since we analyze 2^8 times less pairs, but the quantity of work to check *one* pair is multiplied by the same factor. We describe this efficient attack in an algorithmic manner in Algorithm 5.3.

5.2.5 Key recovery

In this section, we present an efficient way to turn this distinguisher into a key recovery attack. First, let us summarize what the adversary has in his possession at the end of the efficient attack: a pair (m, m') following the truncated differential characteristic, a δ-set containing m, the knowledge of 9 key bytes and the corresponding multiset for which we found a match in the precomputed table. Thus, there are still 2^{56}, 2^{120} or 2^{184} possible keys, if we consider AES-128, AES-192 or AES-256 respectively. As a consequence, performing an exhaustive search to find the missing key bytes would drastically increase the complexity of the whole attack, except for the 128-bit version. Even in that case, it seems nontrivial to recover the 2^{56}

Algorithm 5.3 – An efficient attack on 7-round AES.

function EFFICIENTATTACK
  for all $(i,j) \in \{0,\dots,3\} \times \{0,\dots,15\}$ **do** ▷ *Construction of the* 2^6 *Tables*
    $T_{i,j} \leftarrow$ CONSTRUCTTABLE2(i,j).
  while *true* **do** ▷ 2^{35} *times on average*
    Ask for a structure S of 2^{64} plaintexts P_m
      where bytes in diagonals 0 and 1 assume all values.
    for all $k \in \{0,\dots,3\}$ **do** ▷ *Position of the non-zero column of* Δx_6
      Empty a hash table T of list of plaintexts.
      for all corresponding ciphertexts C_m **do**
        $index \leftarrow (\mathsf{SR}^{-1} \circ \mathsf{MC}^{-1}(C_m))[\{0,\dots,15\} - \{4k,\dots,4k+3\}]$.
        for all $P \in T[index]$ **do**
          Consider the pair (P, P_m). ▷ 2^{33} *pairs by structure on average*
          for all $(i,l_j) \in \{0,\dots,3\} \times \{0,\dots,3\}$ **do**
            $j \leftarrow 4k - 3l_j \mod 16$. ▷ *Assume* mod *give a positive result.*
            ONLINEPHASE$\left((P,P_m), i, j, T_{i,j}, S\right)$.
        $T[index] \leftarrow T[index] \cup \{P_m\}$.

function ONLINEPHASE$((m,m'), i, j, T, S)$
  $b_j \leftarrow (j - 4 \times (j \mod 4)) \mod 16$. ▷ *Retrieving the right positions*
  $c_j \leftarrow \lfloor b_j/4 \rfloor$. ▷ *because of the* **ShiftRows**.
  $Col_j \leftarrow \{4c_j,\dots,4c_j+3\}$
  for all $k_{-1}[0,5,10,15]$ s.t. $\Delta w_0[\{0,\dots,3\} - \{i\}] = 0$ **do**
    Construct δ-set D from m.
    for all $\mathsf{SR}(u_6)[Col_j]$ s.t. $\Delta z_5[Col_j - \{j\}] = 0$ **do**
      Decrypt column c_j of x_6 for D.
      for all $u_5[b_j]$ **do**
        Decrypt byte j of x_5 for D.
        Construct multiset M of Δx_5.
        if $M \in T$ **then**
          return EXHAUSTIVESEARCH()

possible keys in less than 2^{96}, as the 9 key bytes do not belong to the same subkey.

A natural way to recover the missing bytes would be to replay the efficient attack by using different positions for the input and output differences. Unfortunately, this increases the complexity, and it would also interfere with the trade-off since we could not look for all the possible positions of the differences anymore.

We propose a method that recovers the two last subkeys in a negligible time compared to the 2^{99} encryptions of the efficient attack. First, the adversary guesses the 11 parameters used to build the table of multisets, computes the value the corresponding 24 parameters and keeps the only one used to build the checked multiset. In particular, he obtains the value of all the intermediate state x_3 and one column of x_2. As a consequence, and for any position of the active byte of x_5, the Demirci and Selçuk original attack may be performed really quickly. Indeed, among the 9 (resp. 24) bytes to guess to perform the online (resp. offline) phase, at

least 4 (resp. 20) are already known and the data needed is also in his possession. Finally, the adversary replays this attack for each position of the active byte of x_5 and thus retrieves the two last subkeys.

5.3 Extensions to 8 and 9 rounds

We can extend the simple attack on the AES presented Section 5.2.3 to an 8-round attack for both 192- and 256-bit versions by adding one additional round at the end (see Figure 5.7). The main consequence of this extension is to destroy the linearity existing in the subspace of differences in the ciphertexts. Now, we apply a non-linear layer on it so that we get pairs of ciphertexts that are fully active on the 16 bytes, and the differences are completely independent. This attack is schematized on Figure 5.7. Due to the high complexities, the exhaustive search happens to be faster to recover the key in the case of AES-128.

The main difficulty compared to the previous attack is that we cannot apply a first step to the structure to filter the wrong pairs. Indeed, now for each pair from the structure, there exists at least one key such that the pair follows the differential characteristic. Then our goal is to enumerate, for each pair and as fast as possible, the key bytes needed to identify a δ-set and construct the associated multiset assuming that the pair is a right one.

The main idea to do so is the following: if there is a single non-zero difference in a column of a state before (resp. after) the **MixColumns** operation, then the difference on same column in the state after (resp. before) can only assume $2^8 - 1$ values among all the $(2^8 - 1)^4$ possible ones. Combining this with the key schedule equations and with the differential property of the AES S-Box (Theorem 4.2), this leads to an attack requiring 2^{113} chosen plaintexts, 2^{82} 128-bit blocks of storage and a time complexity equivalent to 2^{172} (resp. 2^{196}) encryptions on AES-192 (resp. AES-256).

To reach this time complexity, the position of the output active byte must be chosen carefully. The position of the input active byte for both the pair and the δ-set must be identical, as well as the output active byte of the pair and the byte that is to be checked. Then, the output difference must be located at position 1, 6, 11 or 12 in the case of AES-192. As for the AES-256, it can be located anywhere, except on bytes 0, 5, 10 and 15. Finally, in both cases, the position of the input difference does not matter.

5.3.1 Attack on 8-round AES-192

Assume the positions of the input and output active bytes of the 4-round differential between rounds 1 and rounds 5 are respectively 0 and 1. In the first stage of the attack, we ask for the encryption of 2^{81} structures of 2^{32} plaintexts. This allows the construction of

$$2^{81} \cdot \binom{2^{32}}{2} \approx 2^{144}$$

pairs, and for each of them, we apply the following procedure, which enumerates the 2^{24} possible values for the key bytes in about 2^{24} simple operations. Those key bytes are required to identify a δ-set at the input of AES-192 to perform the attack.

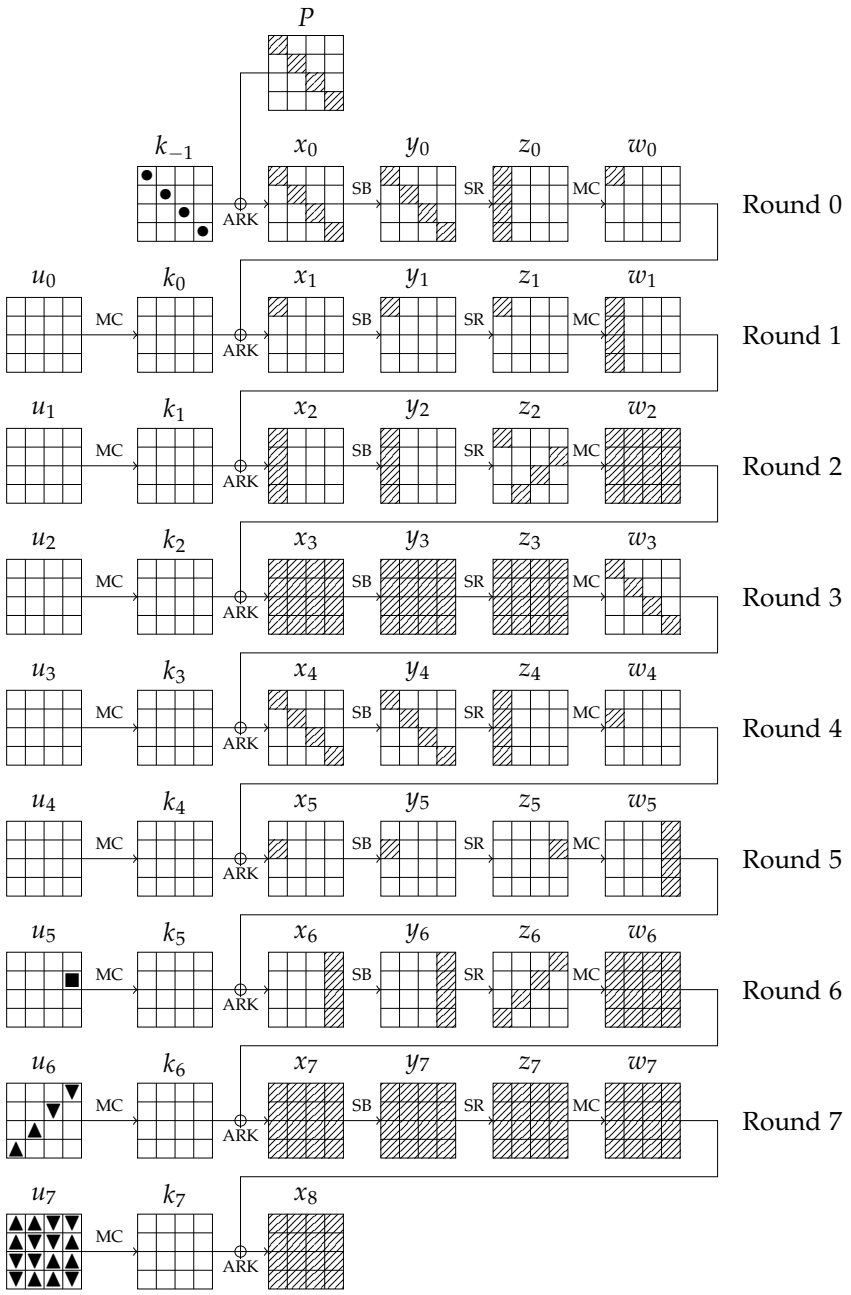


Figure 5.7: Complete 8-round truncated differential characteristic used in the 8-round attacks on AES-192 and AES-256.

For the current pair (see Figure 5.7), the input difference is known and in particular the input difference in the main diagonal. By guessing the byte difference in byte 0 of w_0, we learn the difference at the input and the output of the first **SubBytes**. With Theorem 4.2, we deduce the paired values for those bytes, which linearly give the four diagonal bytes of the first subkey: $k_{-1}[0, 5, 10, 15]$. We mark those bytes by a • on Figure 5.7. We store this 32-bit value in a table T_{-1} at the index $k_{-1}[15]$. Since we add one element to T_{-1} for each guess Δw_0, we expect this table to contain 2^8 elements.

Then, for the same pair, we guess the differences in the last column of x_6, that is the four bytes $\Delta x_6[12, 13, 14, 15]$. This step performs a meet-in-the-middle technique to recover the possible values for the key bytes such that the pair follows the differential characteristic of Figure 5.7.

Step 1. On the one hand, we guess the two bytes $\Delta z_6[3]$ and $\Delta z_6[6]$, and linearly compute the differences in the left half of $\Delta w_6 = \Delta x_7$. Since the pair is fixed, we know its output difference and incidentally the difference Δy_7 by linearity. Therefore, the input and output differences of the left halves of Δx_7 and Δy_7 are known, so Theorem 4.2 also applies and determines the paired values for these bytes. Again, the linear **AddRoundKey** determines 8 bytes of the equivalent subkey u_7, namely $u_7[0, 1, 4, 7, 10, 11, 13, 14]$. Additionally, as we guessed $\Delta z_6[3, 6]$, we also deduce the values of bytes $u_6[3]$ and $u_6[6]$. We marked those 10 bytes by ▲ on Figure 5.7. We store those values of subkeys u_6 and u_7 in a table $T_{6,7}$ at index $i_{6,7}$ that we define later. The table $T_{6,7}$ has an expected size of 2^{16} elements.

Step 2. On the other hand, we guess the two other active differences in Δz_6, that is $\Delta z_6[9]$ and $\Delta z_6[12]$ and linearly compute the differences in the right half of $\Delta w_6 = \Delta x_7$. Similarly, we determine values of the 8 other bytes of the equivalent subkey u_7, namely $u_7[2, 3, 5, 6, 8, 9, 12, 15]$ and also the 2 bytes $u_6[9]$ and $u_6[12]$ for the same reasons as before. Now, we can use a particularity of the key scheduling algorithm specific to AES-192 (see Figure 5.8) allowing to match the bytes from this step with one entry of table $T_{6,7}$ stored in the previous step. The idea is to take advantage of the linear equations of the key schedule, by expressing $u_6[4]$ and $u_6[6]$ as linear functions of bytes from step 1 and 2 to perform a meet-in-the-middle. Namely, from the **MixColumns** inverse operation, we have

$$\begin{aligned} u_6[4] &= 14k_6[4] \oplus 13k_6[5] \oplus 11k_6[6] \oplus 9k_6[7] \\ u_6[6] &= 11k_6[4] \oplus 9k_6[5] \oplus 14k_6[6] \oplus 13k_6[7] \end{aligned}$$

and the key schedule equations for AES-192 (see Figure 5.8) give:

$$\begin{aligned} k_6[4] &= k_7[8] \oplus k_7[12] \\ k_6[5] &= k_7[9] \oplus k_7[13] \\ k_6[6] &= k_7[10] \oplus k_7[14] \\ k_6[7] &= k_7[11] \oplus k_7[15], \end{aligned}$$

which leads to:

$$u_6[4] = 14k_7[8] \oplus 14k_7[12] \oplus 13k_7[9] \oplus 13k_7[13] \oplus 11k_7[10] \oplus 11k_7[14] \oplus 9k_7[11] \oplus 9k_7[15], \quad (5.7)$$

$$u_6[6] = 11k_7[8] \oplus 11k_7[12] \oplus 9k_7[9] \oplus 9k_7[13] \oplus 14k_7[10] \oplus 14k_7[14] \oplus 13k_7[11] \oplus 13k_7[15]. \quad (5.8)$$

For simplicity, we assume that we get bytes from k_7 rather than bytes from u_7. To consider the exact case, we would need to write four more linear relations from the inverse **MixColumns** operation to express $k_7[8], \ldots, k_7[11]$ in terms of $u_7[8], \ldots, u_7[11]$. We can reorder the variables so that the left-hand sides only contain variables from step 1, and the right-hand sides from step 2:

$$\begin{aligned} u_6[4] \oplus 14k_7[8] \oplus 13k_7[9] \oplus 14k_7[12] \oplus 9k_7[15] &= 11k_7[10] \oplus 9k_7[11] \oplus 13k_7[13] \oplus 11k_7[14] \\ 11k_7[8] \oplus 9k_7[9] \oplus 11k_7[12] \oplus 13k_7[15] &= 14k_7[10] \oplus 13k_7[11] \oplus 9k_7[13] \oplus 14k_7[14] \oplus u_6[6]. \end{aligned}$$

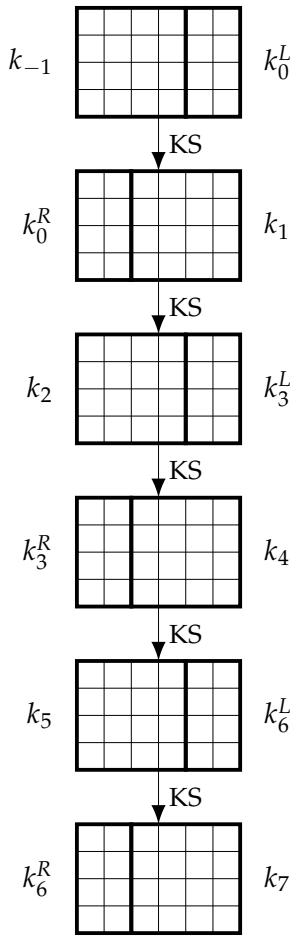


Figure 5.8: Five steps of the key scheduling algorithm of AES-192. We denote k^L the left part of subkey k, and k^R its right part. The subkeys k_i, $0 \leq i \leq 7$ are 128-bit large and incorporated in the 8-round encryption depicted on Figure 5.7.

We note that the same separation of variables applies with variables from u_7. From this, we select an entry of the previously stored table $T_{6,7}$ at the index defined by the two previous left-hand sides

$$\Big(u_6[4] \oplus 14k_7[8] \oplus 13k_7[9] \oplus 14k_7[12] \oplus 9k_7[15],\ 11k_7[8] \oplus 9k_7[9] \oplus 11k_7[12] \oplus 13k_7[15]\Big)$$

and retrieve the full subkey k_7 in one operation. The right-hand side linear relations of the meet-in-the-middle in equations (5.7) and (5.8) define the index $i_{6,7}$ to store the entries in table $T_{6,7}$ in step 1. This whole step is therefore done in 2^{16} simple operations, basically by enumerating the values of the two guessed bytes $\Delta z_6[9]$ and $\Delta z_6[12]$.

Step 3. The knowledge of k_7 allows to compute the last column of k_5 and u_5 by inverting the last key schedule step (Figure 5.8). From Proposition 5.3 found by Dunkelman et al., we can also compute $k_{-1}[\cdot, 3]$, that is bytes $k_{15}[12, 13, 14, 15]$.

Proposition 5.3 (Key bridging, [DKS10]). *By the key schedule of* AES-192*, the knowledge of columns* 0, 1, 3 *of the subkey* k_7 *allows to deduce column* 3 *of the whitening key* k_{-1}.

Now, with byte $k_{-1}[15]$, we can use table T_{-1} to select one entry and recover the four diagonal bytes $k_{-1}[0,5,10,15]$ of k_{-1}.

Finally, for each of the 2^{144} pairs and for each of the 2^{24} subkeys corresponding to one pair, the adversary identifies the δ-set and verifies whether the corresponding multiset belongs to the precomputed table. This part of the attack is the same as the previous attacks described for 7 rounds.

Thus, the time complexity is equivalent to $2^{144} \cdot 2^{24} \cdot 2^{8} \cdot 2^{-4} = 2^{172}$ encryptions to enumerate the pair, the key bytes, construct the δ-set and check the precomputed table for the multiset. The data complexity comes from the encryption of 2^{113} chosen plaintexts, and the memory requirements amount to 2^{80} multisets, that is 2^{82} blocks of 128 bits.

5.3.2 Attack on 8-round AES-256

In the case of the 256-bit version, the procedure is very similar. The only difference comes from the key scheduling algorithm which prevent u_6 to be deduced from u_7 (see Figure 5.9). Indeed,

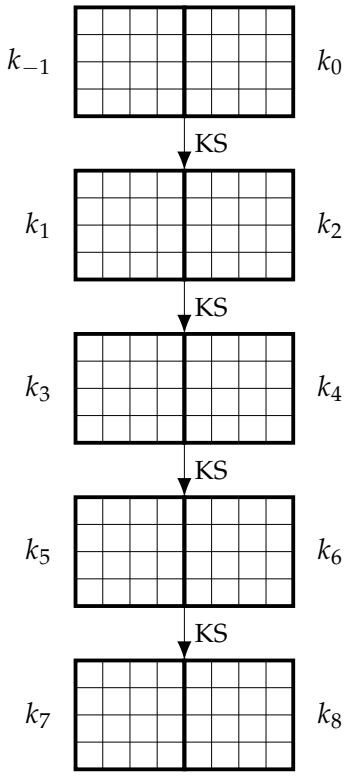


Figure 5.9: Four steps of the key scheduling algorithm of AES-256.

k_6 and k_7 are independent and the only property we can use is the following one.

Proposition 5.4. *By the key schedule of* AES-256*, the knowledge of the subkey k_7 allows to linearly deduce columns 1, 2 and 3 of k_5.*

Because of the independence of k_6 and k_7, we cannot apply the meet-in-the-middle strategy

on the two linear relations coming from the key schedule as before, so steps 1 and 2 collapse into a single one. First, we guess the four differences in Δz_6 at once, and deduce the paired state in x_7 from Theorem 4.2 since we know the ciphertext difference for the current pair. Then, we deduce the full subkey k_7 and incidentally u_7, and also the four diagonal bytes of u_6. This has been done in 2^{32} simple operations. As the two remaining required bytes $u_5[13]$ and $k_{-1}[15]$ are independent of the current ones, we guess them and then we can construct the δ-set and finish the attack as before. Then, there are $2^{32+8+8} = 2^{48}$ possible values for the required key bytes and we enumerate them in 2^{48} simple operations.

Consequently, the straightforward application of the previous algorithm runs in time equivalent to $2^{144} \cdot 2^{48} \cdot 2^8 \cdot 2^{-4} = 2^{196}$ encryptions, but it is possible to save some data in exchange for memory by considering several characteristics in parallel. We can bypass the fact that all the positions for the output active byte do not lead in the same complexity by performing the check on y_5 instead of x_5. This is done by just adding one parameter to the precomputed table and increases its size by a factor 2^8, from 2^{80} to 2^{88} multisets. Then, we can look for all the $4 \cdot 16 = 2^6$ differentials in parallel on the same structure.

All in all, the data complexity and the memory requirement become respectively $2^{113}/2^6 = 2^{107}$ chosen plaintexts and $2^{88+2} \times 2^6 = 2^{96}$ 128-bit blocks, while the time complexity remains equivalent to 2^{196} encryptions.

5.3.3 Attack on 9-round AES-256

The 8-round attack on AES-256 can be extended to an attack on 9-round by adding one round right in the middle (see Figure 5.10). This only increases the memory requirements: the time and data complexities remain unchanged. More precisely, the number of parameters needed to construct the precomputed table turns out to be $24 + 16 = 40$, but they can only assume $2^{8\times(10+16)} = 2^{208}$ different values. By adding the complete 128-bit subkey k_3 to the precomputed table, we can extend the number of covered rounds in the offline phase from 4 to 5. Now, the multisets are constructed in y_6 from a δ-set in x_1.

All in all, without the previous tradeoff, the data complexity of the attack stays at 2^{113} chosen plaintexts, the time complexity remains 2^{196} encryptions and the memory requirement reaches about 2^{210} 128-bit blocks to store the large precomputed table.

To reduce its complexity, we can cover only a fraction 2^{-7} of the possible multisets stored in the precomputed table. In return, the data and time complexities are increased by a factor 2^7 by replaying the attack several times. This way, we reach balanced complexities with $2^{113} \times 2^7 = 2^{120}$ chosen plaintexts, $2^{196} \times 2^7 = 2^{203}$ encryptions and a storage of $2^{210}/2^7 = 2^{203}$ blocks of 128 bits.

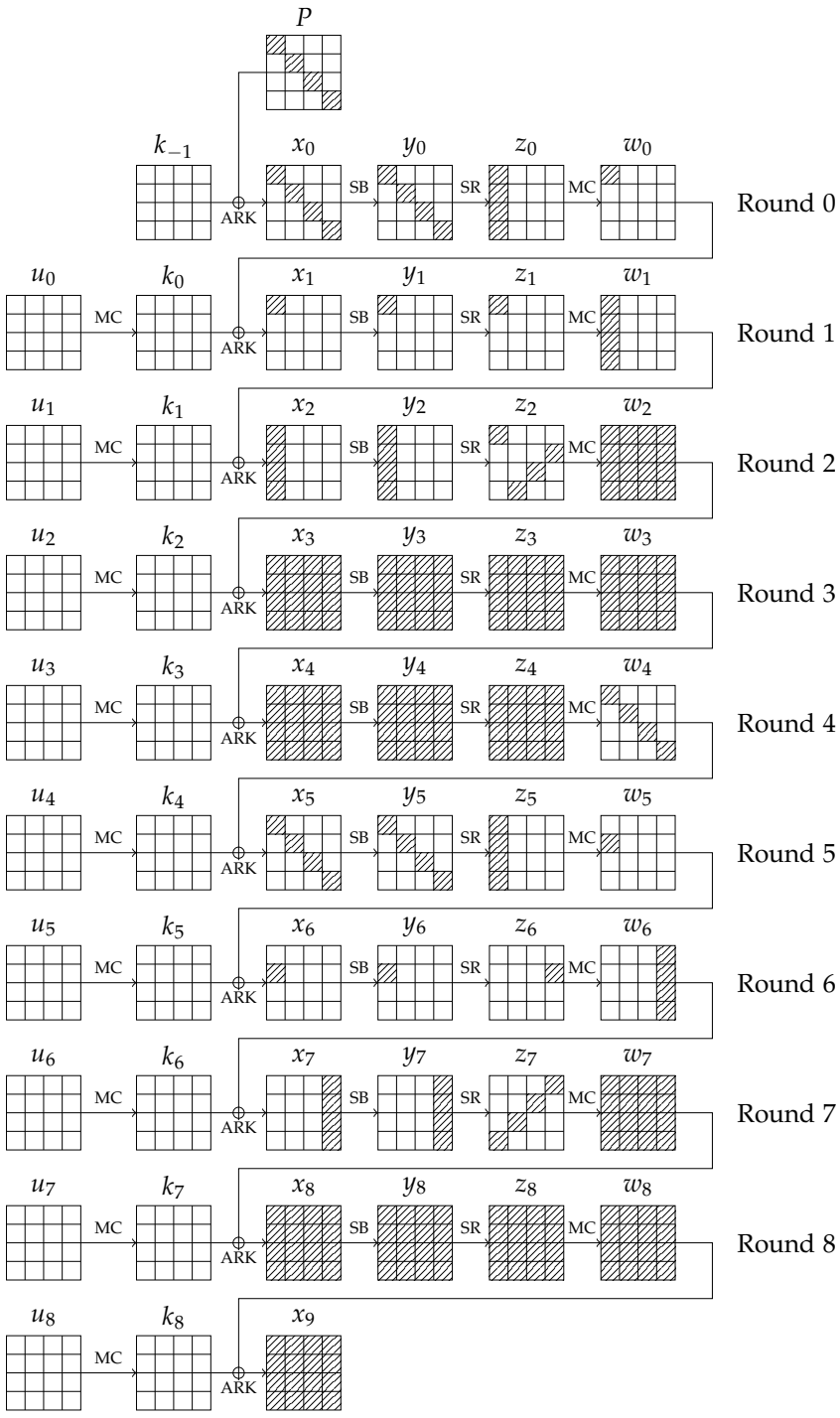


Figure 5.10: Complete 9-round truncated differential characteristic used in the 9-round attack on AES-256.

CHAPTER 6

AES in the Related-Key Model

Contents

6.1 Generalities **112**
6.1.1 Motivations 112
6.1.2 Graph traversal algorithms 113
6.1.3 Structural evaluation 116
6.2 Definitions **118**
6.2.1 Substitution-Permutation Network 118
6.2.2 Truncated and actual differences 119
6.3 Related-key differential characteristics **120**
6.3.1 Differential characteristic search 120
6.3.2 Precomputation phase 122
6.3.3 Online phase 124
6.4 Enhanced Markov process **127**
6.4.1 The Markov assumption and actual differences 127
6.4.2 Block cipher state compression 128
6.4.3 Evaluating the number of nodes/edges of G_{BC} and G_{KS} 128
6.4.4 More complete Markov process 130
6.4.5 Explanations 132
6.5 Applications to SPN and AES-128 . . . . . **133**
6.5.1 Structural evaluation of SPN AES-like ciphers 133
6.5.2 Differential characteristics results for AES-128 135

In this chapter, we consider the resistance of the structure of the AES in the related-key model where the adversary can encrypt plaintexts or decrypt ciphertexts under a set of keys related by a known relation. By *structure*, we mean the design of the AES, where the building blocks have unspecified or unknown values, namely the S-Box or the MDS matrix. An example of relation can for instance be the case where the adversary has access to two encryption oracles using keys k and k' such that $k' = k \oplus \delta$ for a non-zero known difference δ.

Our work is actually more general as it captures not only AES but also any SPN ciphers. In detail, we develop an efficient and generic algorithm that computes all the best differential characteristics for general SPN ciphers, including the ones conforming to the AES structure. We

recall that a bound on the differential probability of the differential characteristic for a cipher give an estimation of its resistance to differential attacks (see Chapter 3). We show how we can reduce this problem to the shortest path problem in a special class of directed acyclic graphs.

The conclusions of our analysis that we detail in the following sections consist of impossibility results. The resistance of the AES structure against differential cryptanalysis in the related-key model *cannot* be proven unless we provide the instantiation of the S-Box. We consider several scenarios like the classical related-key model, and also the hash function setting where the adversary can control the key bits. In the latter scenario where both the key schedule and the message can be considered somewhat independently by the adversary, we prove the impossibility to ensure the resistance of the AES structure against differential cryptanalysis unless the instantiations of the S-Box and the linear layer are provided.

This chapter is largely inspired from an article co-authored with Pierre-Alain Fouque and Thomas Peyrin that has been published at CRYPTO 2013 in [FJP13a]. We begin this chapter by a motivation section where we also introduce the literature and previous algorithms in this research line. This particularly includes other automatic analyses and search algorithms for differential characteristics.

6.1 Generalities

6.1.1 Motivations

Block ciphers and hash functions are among the most important primitives in cryptography and while their respective goals are different, they are related in many ways. For example, most compression functions, which can in turn be used to define a hash function, are built upon an internal block cipher thanks to classical constructions such as Davies-Meyer (DM), Matyas-Meyer-Oseas (MMO) or Miyaguchi-Preneel (MP) [ISO04, MMO85, PGV94] (see also Section 1.4).

One of the main differences between the two families of primitives is that in the case of the block cipher, the key input is unknown and uncontrolled by the attacker, whereas for the compression function, the attacker has full control on the key schedule (generally called message expansion in that context). Yet, the so-called *related-key* attack scenario [Bih93, BDK08] is interesting for both cases. This model allows the attacker to incorporate differences not only in the plaintext or ciphertext input, but also in the key input. While less relevant in practice than the classical single-key model, it is important to analyze block ciphers in the light of related-key attacks since the secret keys are often updated in security protocols or differences can be incorporated using fault attacks (see for instance the WEP case detailed in Section 3.4.4).

Moreover, related-key attacks are also very important when the block cipher is used as inner primitive of a hash function, and in that setting one can even consider the known-key or chosen-key models where the attacker is given knowledge or complete control of the key and his goal is to exhibit some non-ideal property of the primitive (see the following Chapter 7).

To measure the resistance of a primitive against differential cryptanalysis, we often consider differential characteristics rather than the more general concept of differentials. In that sense,

avoiding high-probability related-key differential characteristics is one of the goal of the key schedule, and so far various directions have been investigated to construct this component. This has been taken into account in the conception of the Piccolo block cipher [SIH+11] or in the Whirlpool hash function [BR00], the latter proposing to use the same AES-like permutation for both the internal permutation and the message expansion part, leading to a strong key schedule in terms of number of S-Box calls, but quite slow as it represents about half of the total amount of computations. As a complete opposite, the designers of the LED block cipher [GPPR11] have chosen to use no key schedule at all, at the expense that an important number of rounds is required. These two functions can both provide provable security with regard to related-key differential attacks, but they also both suffer from efficiency issues.

In general, see for example AES or PRESENT [BKL+07], key schedules are built by using an ad-hoc and relatively light function that is quite different from the main permutation, in a hope that this will avoid any correlation between the two components and enforce low-probability related-key differential characteristics. However, because of the heuristic design process and the difficulty of the task, no real security argument is given and this can eventually lead to security issues [BK09, BN10]. To help designers and cryptanalysts, many automated differential analyses have already been applied to various primitives [BN11, WY05, MP08, DR06c, WYY05a, Leu12, BN10, KBN09, BDF11].

The AES block cipher is currently the most interesting candidate to scrutinize with regard to related-key, chosen-key attacks or when used as a black-box in cryptosystems. Indeed, during the NIST SHA-3 hash function competition, many candidates [BBG+09, GKM+11, BD09] reused some components from the AES. Moreover, previous cryptanalytic results in the related-key model showing attacks on the full versions of AES-192 and AES-256 [BK09, BKN09] have been discovered, which give less confidence in the key scheduling algorithms of these two versions and somehow draw attention on the one of AES-128.

While basic differential and linear attacks against the AES in the single-key scenario seem to be mastered since the design of the cipher focuses in particular to resist to those class of attacks, provable security against related-key attacks seems more complex to tackle. In this chapter, we consider the related-key model applied to SPN ciphers and in particular to the generalized design of the AES that we already presented as AES-like permutation in Section 4.3. Our work is then completed by an analysis in the known-key and chosen-key models in the following Chapter 7.

We now recall the related works done in automated analysis of block ciphers against differential cryptanalysis (Section 6.1.2), and then we discuss the *structural* aspect of block cipher design and analysis (Section 6.1.3).

6.1.2 Graph traversal algorithms

As the algorithm we develop uses graph-based techniques for differential characteristics search, we recall here a fraction of the literature in the same domain.

In [Mat94b], Matsui proposes an algorithm to find the best differential characteristics for DES. The strategy to find the best one on n rounds first starts by computing the best ones on 1 to $n-1$ rounds. The algorithm works by induction and can be seen as a tree traversal in

a depth-first manner, where the tree represents all the possible differential characteristics in the cipher layered by round (see Figure 6.1). The nodes represent the actual differences and

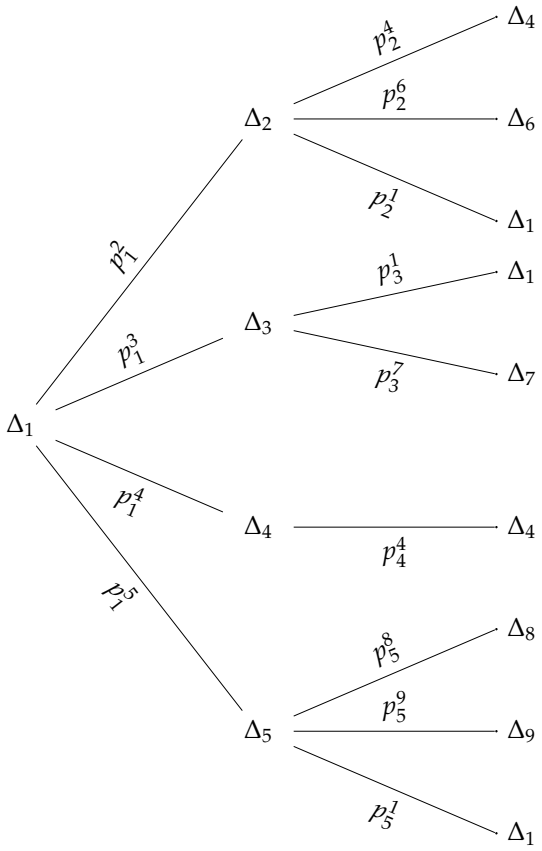


Figure 6.1: Example of a tree representing all differential characteristics for 2 rounds of a SPN cipher. The nodes are values of difference, and p_i^j is the differential probability of the 1-round differential $\Delta_i \to \Delta_j$.

the edges the possible transitions between them, and are labeled by their probabilities. One differential characteristic is a path in this tree, for example $\Delta_1 \to \Delta_3 \to \Delta_7$, and its probability equals the product of all traversed edges, e.g. $p_1^3 p_3^7$. We are looking for the path with the highest probability in this tree. The knowledge of the previous best characteristics, i.e. up to some depth in the tree, allows pruning during the procedure like the A^* heuristic [HNR68]: the target value being known (the exhaustive search bound), we can reduce the possibilities for each one-round transition. Using such an algorithm, the complexity is exponential in the number of nodes in the tree, and therefore in the block-size and the number of rounds, except if the pruning is very efficient.

In modern byte-oriented ciphers, designers ensure there is a fast diffusion and that all actual differential transitions occur with (almost) the same probability: all differences become equivalent. This is for instance the case in AES as the S-Box has maximal differential probability 2^{-6}. Consequently, Matsui's search algorithm becomes less efficient since there is no dominant characteristic. Biryukov and Nikolić propose in [BN10] to restrict the search to truncated differences to decrease the number of edges in the tree. They also introduce a nice representation of truncated differences to consider the branching (combinatorial explosion of differences) in

the key schedule. Their work replaces the actual differences located at the nodes of the previous tree representation of Matsui of Figure 6.1 by truncated differences, and the branching is expressed by the degree of each nodes in the tree.

In this chapter, we present a new algorithm to perform the same search by changing the tree representation from the previous works into a graph: the nodes and edges have the same signification as before, but we merge all the nodes representing the same differences after the same number of rounds into a single one (see Figure 6.2). Matsui's tree encodes all the paths of

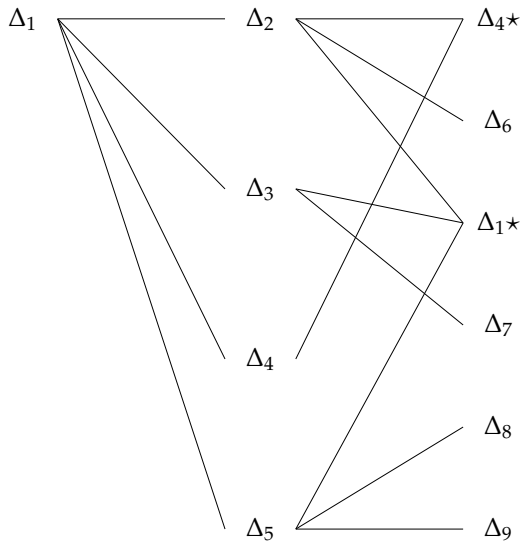


Figure 6.2: Graph representation of the tree from Figure 6.1. Equal differences on a layer have been grouped into a singe node: we mark by a $\star$ the nodes that represent several nodes in the previous tree.

our graph, from the source node to any nodes in the last layer. Consequently, the numbers of nodes and edges decrease a lot and become linear in the number of rounds. Finding the best differential characteristics is reduced to a shortest path problem: we want all the shortest paths in this graph to get all the differential characteristics with the highest probabilities.

We use a variant of Dijkstra algorithm combined with the A^* heuristic [HNR68] to explore a kind of graph product in a breadth-first manner. Our algorithm uses a dynamic programming method, which has been considered too costly in terms of memory in [BN10]. This approach solves the problem of finding the best related-key characteristics using graph algorithms in polynomial time in the number of rounds and exponential in the state size, whereas the previous best known ways are exponential in both parameters using Matsui's algorithm variants. We note that the search in [BN10] has been made possible thanks to an extreme pruning in the AES tree.

The algorithm computes the probabilities of the differential characteristics by multiplying the differential probabilities of each traversed edge in a path. As we explain in more detail in the following, this is based on a Markov process and the same graph-based problem can be reformulated as a matrix exponentiation. The matrix itself is the transition matrix of the Markov chain that defines the differential probabilities of *all* one-round differentials. The matrix therefore describes one-round transitions in the cipher, and its n-th power corresponds to the application of n rounds of the cipher. If we evaluate input vectors of the form $[1, 0, \ldots, 0]^{\mathrm{T}}$

where all but one coordinate are zero, which corresponds to an input active difference, we can compute the probabilities of all differential characteristics and incidentally find the best ones. However, the size of the matrix makes the exponentiation algorithms less efficient that our graph-based solution, but the core idea is the same.

6.1.3 Structural evaluation

By structural evaluation, we mean the domain of cryptography that analyzes a cryptosystem in terms of generic constructions using black-box elements. We are interested in how the building blocks of the primitives interact together, while "ignoring their semantic definitions as particular functions" as in meet-in-the-middle attacks.

In this line of research, a major result is the conception of `Rijndael`, or how to construct a block cipher provably resistant to differential attacks. Daemen and Rijmen show in [DR02] a lower bound B_r on the number of active S-Boxes for any differential characteristic on r rounds of `Rijndael`, when no difference is introduced in the key. For an S-Box with maximal differential probability p_{max}, this result allows to upper bound the probability of success of any differential attack on r rounds by $p_{max}^{B_r}$. For k-bit keys block ciphers, the resistance to differential cryptanalysis means $p_{max}^{-B_r} > 2^k$, which gives a criteria on r and p_{max} for the security of the cipher.

In [BS10], Biryukov and Shamir analyze the SASAS construction, alternating five layers of non-linear S and affine A functions. In their article, they consider generalized SPN structures

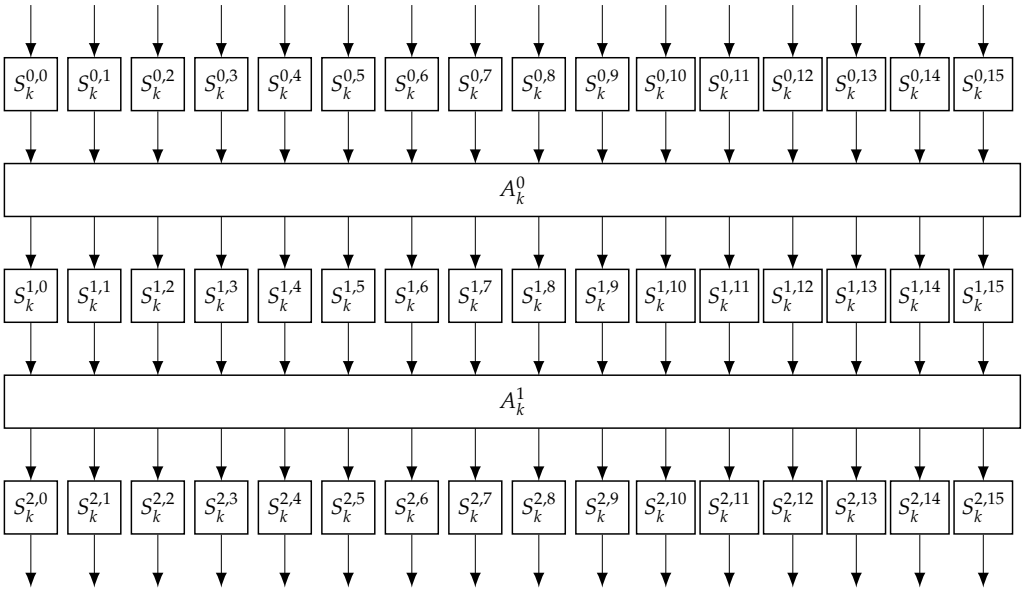


Figure 6.3: SASAS construction: 5 layers with alternating Substitution (S) and affine layers (A), which are not necessarily the same and may depend on the secret key k. The figure takes the AES as example: an arrow is 8-bit wide and there are 16 of them.

where the permutation layer is replaced by an affine mapping, and its diffusion property is assumed to bring complete diffusion in a single step. This compares for instance with the two rounds needed to bring full diffusion in the AES. Their motivation has been greatly

inspired from the square attack on the `Rijndael` cipher (see Section 4.4.1), but the weakened assumptions on the diffusion and the possibly different and key-dependent (thus unknown) S layers intuitively reduce the strength of the square attack. They show that five such rounds are vulnerable to a very efficient structural attack, even though the adversary does not know anything about the inner structure of both S and A. With the same parameters as the AES, the attack requires 2^{16} chosen plaintexts and 2^{28} simple operations to recover the full secret key k of 128 bits.

In a related work, Eli Biham studies a similar generic construction proposed by Patarin and Goubin in [PG97] called $2R$. The design also has five rounds, and simply interchanges the S and A layers (see Figure 6.4); we call it $ASASA$. However, the assumption on the S-Boxes is completely different: while Biryukov and Shamir assume bijective mappings, the design from Patarin and Goubin instantiates the non-linear S-Boxes by quadratic polynomials, making the S layer non-invertible. In his work [Bih00], Biham describes a generic attack for this kind of

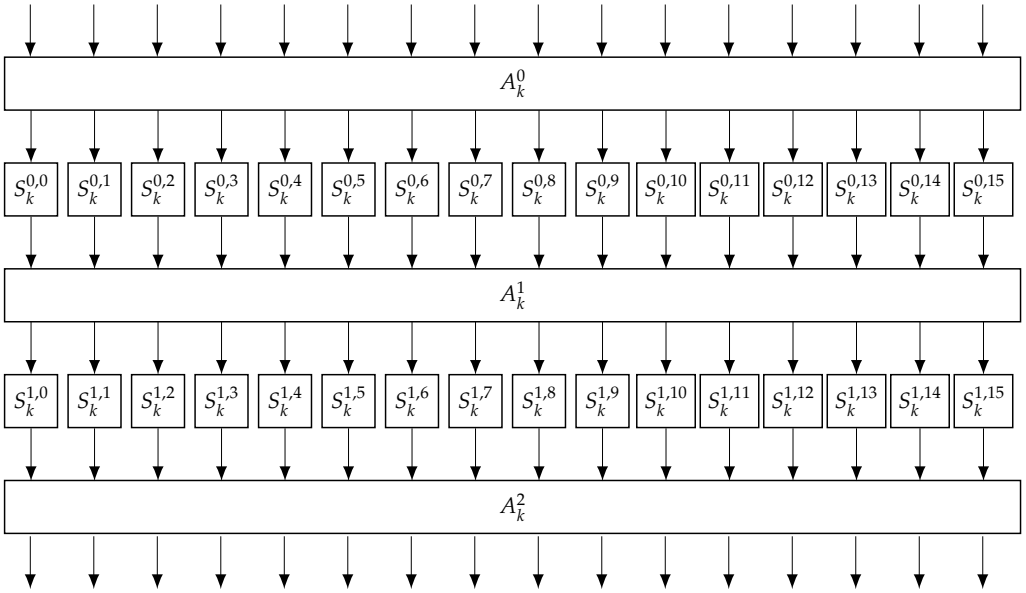


Figure 6.4: ASASA construction: 5 layers with alternating Substitution (S) and affine layers (A), which are not necessarily the same and may depend on the secret key k. The figure takes the AES as example: an arrow is 8-bit wide and there are 16 of them.

design. The structural weakness comes from the existence of collisions created by the S-Boxes, and can be exploited by the birthday paradox. Because of the birthday bound, the complexities of Biham's attack reach about 2^{60} data and simple operations. This attack does not apply to the invertible case analyzed by Biryukov and Shamir [BS10] as no collision can arise from them.

The remaining of this chapter introduces definitions and a study of the structural resistance of generic Substitution-Permutation Networks in the related-key model. Ideally, we would want a lower bound on the number of active S-Boxes in the related-key model for any number of rounds, so that we could upper bound the differential probability of *any* differential characteristics in this model, and consequently lower bound the complexities of any differential attacks in this general setting. Unfortunately, contrary to the single-key model, it is not possible to prove anything by hand on the key schedule resistance in the same vein as [DR02]. This

is due to the ad-hoc nature of the key schedule design which makes a formal analysis much harder than the one of the inner permutation. To tackle this interesting problem anyway, we build a tool to enumerate and analyze the differential characteristics.

6.2 Definitions

6.2.1 Substitution-Permutation Network

To keep our reasoning as general as possible, we give a generic description of Substitution-Permutation Network (SPN) ciphers. We consider that the block ciphers studied here take as input a plaintext or ciphertext of size n bits, and a key of size k bits. The cipher is composed of R successive applications of a round function, and we denote respectively s_i and k_i the successive internal states of the block cipher and the key schedule after the i-th round. The state s_0 is initialized with the input plaintext and k_0 with the input key. One round i is itself composed of three layers:

- a key extraction and incorporation layer (AK) where a n-bit round-key rk_{i-1} is extracted from k_{i-1} and added to s_{i-1},
- a block cipher permutation layer $\mathcal{BC}$ that updates the n-bit current state of the block cipher after addition of the subkey, i.e. $s_i = \mathcal{BC}(s_{i-1} \oplus rk_{i-1})$,
- a key schedule transformation layer $\mathcal{KS}$ that updates the k-bit current state of the key schedule, i.e. $k_i = \mathcal{KS}(k_{i-1})$,

The final ciphertext is then defined as $s_R \oplus rk_R$.

Definition 6.1. (SPN cipher) Let $\mathcal{E}$ be a block cipher whose internal state is viewed as a t_{BC}-cell vector (where $t_{\mathsf{BC}} = \frac{n}{c}$), each cell representing a c-bit word, and the key schedule as a t_{KS}-cell vector (where $t_{\mathsf{KS}} = \frac{k}{c}$). The block cipher $\mathcal{E}$ is called an SPN cipher when its round function $\mathcal{BC}$ is made of a linear permutation P and a non-linear permutation S, with $\mathcal{BC} = \mathsf{P} \circ \mathsf{S}$, the latter applying one or distinct b-bit S-Boxes to every cell.

In the particular case of AES-like ciphers, the internal state of $\mathcal{BC}$ implements an AES-like permutation (see Section 4.3 for more details), and can therefore be viewed as a square matrix of c-bit cells with t rows and t columns ($n = t^2 \cdot c$). Then, the linear layer is itself composed of the **ShiftRows** transformation (SR), that moves each cell by x positions to the left in its own row, and the **MixColumns** transformation (MC), that linearly mixes all the columns of the matrix separately. Overall, for AES-like ciphers we have $\mathcal{BC} = \mathsf{P} \circ \mathsf{S} = \mathsf{MC} \circ \mathsf{SR} \circ \mathsf{S}$ (Figure 6.5).

We also study the more particular AES-like ciphers that have a key schedule internal state that can be viewed as a c-bit cell matrix of t rows and t columns for a $(t^2 \cdot c)$-bit key (t rows and $1.5t$ columns for a $(1.5t^2 \cdot c)$-bit key or t rows and $2t$ columns for a $(2t^2 \cdot c)$-bit key) and whose key schedule layer $\mathcal{KS}$ is the direct generalization of the AES key schedules to the dimension t (see Section 4.2.1). We denote this class of ciphers total-AES-like ciphers, which encompasses all versions of AES. We note that the constant addition of the RCON values in the key schedule does not affect our reasoning: that is why we omit it in the sequel.

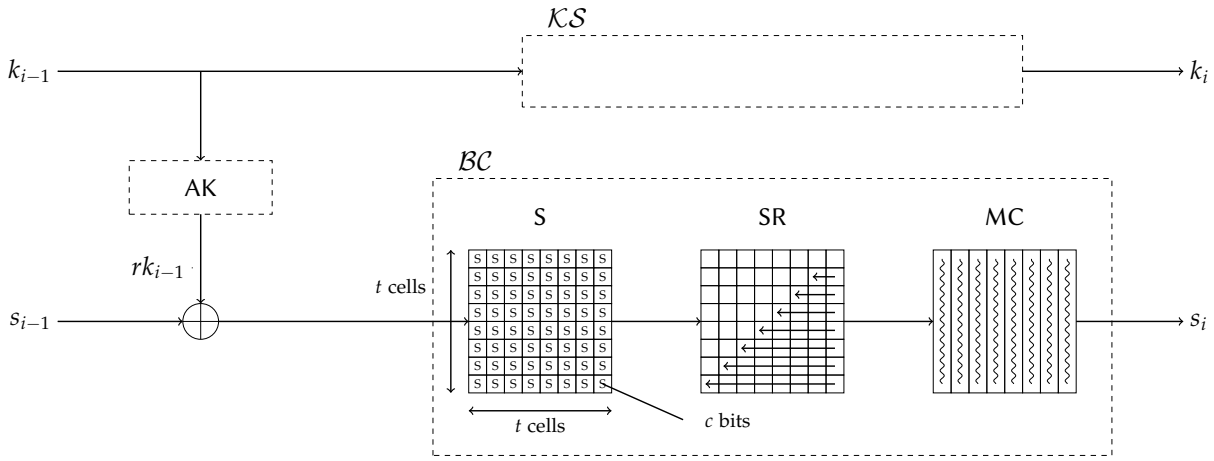


Figure 6.5: One round of the generic SPN and AES-like ciphers.

6.2.2 Truncated and actual differences

We are interested in differential attacks [BS91a, BS91b] (see Chapter 3) and usually, in this scenario the attacker looks for the bitwise difference between two state values. However, here we also consider truncated differential attacks [Knu94]. That is, for a state of differences, we only consider the *presence* of differences in every cell, regardless of their actual values. To distinguish between the two, we call the former *actual* differences and the latter *truncated* differences.

Definition 6.2 (Differences). Let $A = [A_{i,j}]$ and $B = [B_{i,j}]$ two states. We denote their **truncated difference** by $\Delta = [\Delta_{i,j}]$ with $\Delta_{i,j} = 1$ if and only if $A_{i,j} \neq B_{i,j}$ (active cell), and $\Delta_{i,j} = 0$ otherwise (inactive cell). We denote their **actual difference** by $\delta = [\delta_{i,j}]$ with $\delta_{i,j} = A_{i,j} \oplus B^{i,j}$.

First, we analyze the effect of the cipher transformations on the both truncated differences and the actual differences.

6.2.2.1 The substitution layer

One can easily check that the substitution layer S has no effect on the truncated difference of a cell: a cell remains in the same active/inactive situation after application of the bijective transformation. However, S has an effect on the actual difference of every active cells. This effect can be visualized by the differential distribution table (DDT) of the S-Box. More precisely, for each possible pair $(\delta_{in}, \delta_{out})$ of actual differences on the input/output of the S-Box, the table gives the number $\mathrm{DDT}(\delta_{in}, \delta_{out}) = x$ of values X that validate this differential transition, i.e.:

$$\mathrm{S}(X) \oplus \mathrm{S}(X \oplus \delta_{in}) = \delta_{out}.$$

Alternatively, $x/2^c$ represents the differential probability of the transition. An important criteria that can be derived from this table is the maximal differential probability p_{max}, which is the highest possible differential probability when $\delta_{in} \neq 0$ and $\delta_{out} \neq 0$. For example, the S-Box implemented in AES has maximal differential probability $p_{max} = 2^{-6}$.

In order to measure the quality of a truncated differential characteristic, we use the classical counting of the number of active S-Boxes appearing in the characteristic, and we denote it $|\cdot|$.

Definition 6.3. Let $v = [\Delta^i]$ be a vector of truncated differences. The **weight** of v is the number of active differences in v: $\sum_{\Delta^i \neq 0} 1$. We denote it $|v|$ and generalize the notion to any matrix v.

6.2.2.2 The permutation layer for AES-like ciphers

Since the SR layer only moves the cells around, it only changes the active/inactive cells positions in the internal state, but not their number. The same reasoning applies to the actual differences.

The MC transformation being linear, the effect on the values and the actual differences is the same and therefore for each column of the internal state, the output actual differences are simply deduced by the application of the MC linear matrix. Concerning the truncated differences, the effect depends on the branching number B_{MC} of the MC matrix. The branching number is the minimum amount of active cells one can get on both the input and the output of the matrix, excluding the case when there are both null. This measure of the diffusion is crucial for the security of many cryptography primitives and, in general, the MC matrix is Maximum Distance Separable (MDS), that is $B_{MC} = t + 1$ is maximal. A valid truncated differential transition forcing i cells to be inactive on the output happens with probability $2^{-c \cdot i}$.

6.3 Related-key differential characteristics

In this section, we explain the inner workings of our generic related-key differential characteristic search tool for SPN ciphers. As a first step, we model the problem by assuming that the cipher round function is a Markov process in regard to the truncated differential characteristic search (Section 6.3.1). This allows to reduce the problem to a shortest path search in a special $(r+1)$-equipartite directed acyclic graph, for which we provide a simple yet powerful algorithm. The precomputation phase of the process is devoted to building the graphs on which we work on (Section 6.3.2), while the online phase looks for the shortest paths (Section 6.3.3). Finally, we explain how to tweak the Markov assumption in order to find not only the best truncated differential characteristics, but also the actual difference ones (in Section 6.4).

6.3.1 Differential characteristic search

When an attacker considers truncated differentials, he accepts to loose some information (the actual values of the differences) in order to make the analysis simpler. In general, when dealing with truncated differentials for SPN ciphers, most of the attacks actually maintain implicitly more information than just the presence or absence of difference in a cell. For example, in the case of the AES-128, the truncated differential characteristics found verify the linear conditions imposed by the key schedule of the cipher. Therefore, the characteristic actually contains more information than just active/inactive cells.

We describe a first algorithm that generates for any number of rounds *all* the related-key truncated differential characteristics for SPN ciphers, whose number of active S-Boxes is minimal. This analyzes the structure of the cipher in regard to the resistance against related-key

attacks. We make a simple assumption: we would like the search to be a Markov process. More precisely, we assume that the possible differential transitions through a round from one truncated state to another one does not depend on previous round transitions. If we stick to the real definition of truncated differentials (i.e. without implicit conditions contained), then this assumption is verified for SPN ciphers: knowing the truncated input difference of one round represents all the information needed in order to deduce the possible output ones. We discuss in Section 6.4 how to adapt ourselves to the case of actual differences.

Graph modeling

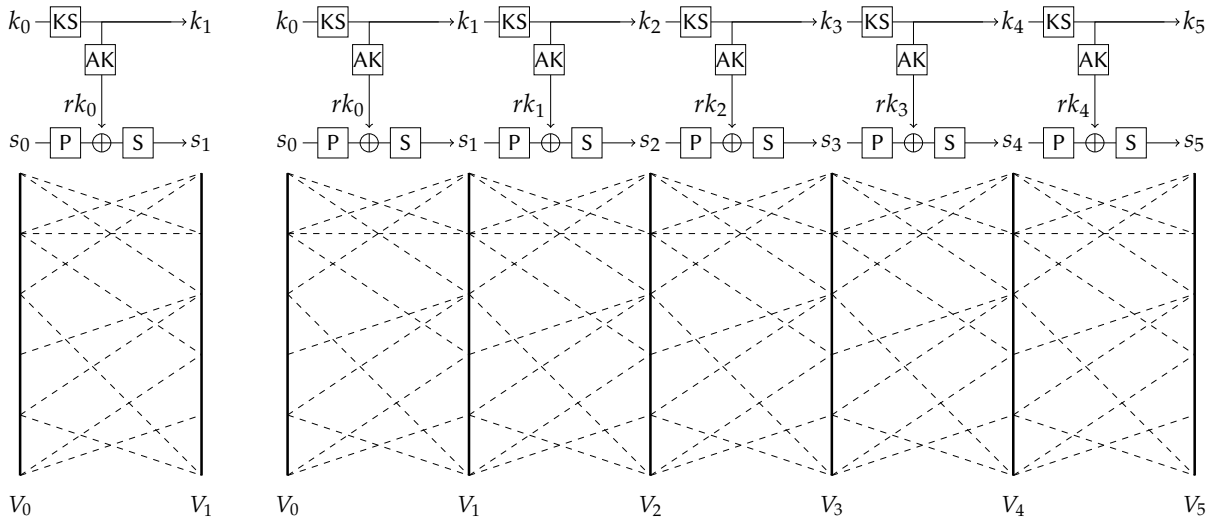
In order to find the best r-round related-key truncated differential characteristics, we use a graph modeling of the problem. Let G be the 2-equipartite directed acyclic graph of all the possible one-round transitions. Thus, all the best r-round related-key truncated differential characteristics correspond to all the shortest paths in the $(r+1)$-equipartite directed acyclic graph G_r built by concatenating r copies of G (see Figure 6.6). Namely, denoting $G = (V_0, V_1 ; E_{0,1})$ the 2-equipartite graph linking with one cipher round a state in set V_0 to a state in set V_1 using some edge in set $E_{0,1}$, we build the graph G_r representing r rounds of the cipher by $G_r = (V_0, \ldots, V_r ; E)$ such that for all i, the subgraph $(V_i, V_{i+1} ; E_{i,i+1})$ is equal to G. Note that all edges are oriented from V_i to V_{i+1}, and that V_i and V_{i+1} contain the same number of nodes.

The nodes of the graph stand for all the possible pairs $(\Delta_{\mathsf{KS}}, \Delta_{\mathsf{BC}})$ where Δ_{KS} represents the truncated difference in the key schedule state and Δ_{BC} represents the truncated difference in the block cipher state. Since we have $2^{t_{\mathsf{KS}}}$ possible values Δ_{KS} and $2^{t_{\mathsf{BC}}}$ possible values Δ_{BC}, all V_i in the graph are composed of $2^{(k+n)/c}$ nodes. The edges correspond to a possible one-round related-key truncated differential characteristic from the input to the output vertex and in the worst case where all differential transitions are possible, we have $2^{2(k+n)/c}$ edges. A path in G_r is defined as a sequence of $r+1$ nodes, one in each of the V_i.

Instead of viewing one round with the normal SPN layers ordering AK, S and P, we prefer to slightly shift the window to the left: P, AK and S, the input key of this new window is the one that has been incorporated into the block cipher state during the previous round (see Figure 6.6). Then, the cost of the transitions are not associated to the vertices, but to the output nodes. Indeed, the number of active cells in the output node represents the number of active S-Boxes during this round[1].

We denote C_{BC} (resp. C_{KS}) the total number of active S-Boxes in the internal permutation part of the block cipher (resp. in the key schedule part) in the whole characteristic. Depending on the situation considered, one might want to minimize $C_{\mathsf{BC}} + C_{\mathsf{KS}}$ for classical scenarios, or instead $\max\{C_{\mathsf{BC}}; C_{\mathsf{KS}}\}$ for hash function settings, where the key schedule and the block cipher parts can be attacked independently. In the case of minimizing $\max\{C_{\mathsf{BC}}; C_{\mathsf{KS}}\}$, the algorithm will not be able to find the shortest path since we loose the total order. However, in most ciphers, $C_{\mathsf{BC}} > C_{\mathsf{KS}}$ and therefore, we minimize only C_{BC}. For the best found paths, if $\max\{C_{\mathsf{BC}}; C_{\mathsf{KS}}\} = C_{\mathsf{BC}}$ is verified, we are ensured that we indeed found one of the best paths minimizing $\max\{C_{\mathsf{BC}}; C_{\mathsf{KS}}\}$.

[1] To be able to associate the number of active S-Boxes in the key schedule to the output node as well, we make the weak assumption that one round of the key schedule is composed of an S-Box and a linear layer at most.



(a) Graph G. **(b)** Graph G_5.

Figure 6.6: Examples of simplified versions of the two graphs G and G_5. Variables s_i and k_i represent the current internal permutation/key state respectively, while rk_i stands for the subkey generated during the round.

Theorem 6.1. (Search algorithm) *Let $\mathcal{E}$ be a SPN cipher on n-bit blocks using a k-bit internal state in the key schedule. Both states are viewed as vectors of b-bit cells. There exists an algorithm $\mathcal{A}$ with a theoretical time complexity of $\mathcal{O}(r \cdot 2^{(2n+k)/c})$ that finds all the best characteristics on r rounds of $\mathcal{E}$.* ■

We emphasize that algorithm $\mathcal{A}$ will find all the shortest paths in G_r representing the differential transitions of r rounds of $\mathcal{E}$. Moreover, we note that the time complexity of $\mathcal{A}$ can be greatly reduced with heuristics.

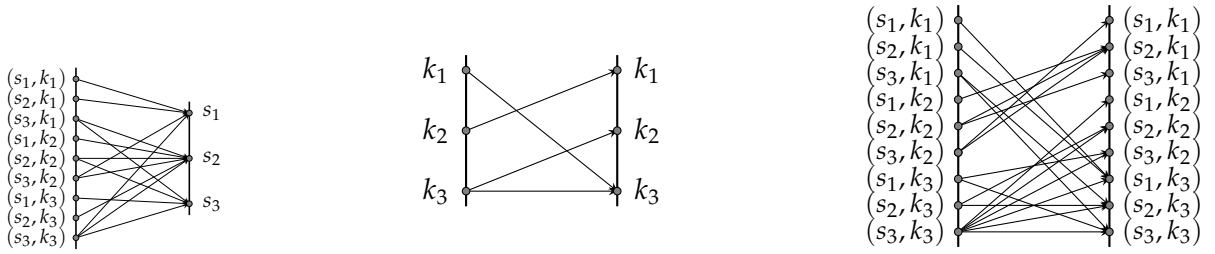
We describe in the next two sections our tool that searches for the best r-round related-key truncated differential characteristics. The precomputation phase (Section 6.3.2) of the tool constructs the graph G from which one can virtually build G_r. During the online phase (Section 6.3.3), the tool looks for the best possible related-key truncated differential characteristics on r rounds by searching for the cheapest paths of size r in the graph G_r.

6.3.2 Precomputation phase

The precomputation phase builds the graph G. It can be built and stored efficiently by observing its inner structure: the block cipher internal state output depends only on the block cipher internal state input and the incoming subkey (deduced by the extraction phase from the key schedule internal state input), while the key schedule internal state output depends only on the key schedule internal state input. Therefore, G can actually be described as a special product of two smaller graphs G_{BC} and G_{KS} (see Figure 6.7), such that an edge $(s_i, k_j) \to (s_{i'}, k_{j'})$ exists in G if and only if $k_j \to k_{j'}$ exists in G_{KS} and $(s_i, k_j) \to s_{i'}$ exists in G_{BC}.

On the one hand, G_{BC} on Figure 6.7a is a bipartite directed acyclic graph whose input nodes are all the possible block cipher internal state and subkey pairs, and whose output nodes are all

the possible block cipher internal states. The edges represent input nodes that can be mapped to output nodes through a valid differential transition. On the other hand, G_{KS} on Figure 6.7b is a 2-equipartite directed acyclic graph, whose input and output nodes are all the possible key schedule internal states. The edges represent input nodes that can be mapped to output nodes through a valid differential transition.



(a) Graph G_{BC}. **(b)** Graph G_{KS}. **(c)** Graph G.

Figure 6.7: Example of graph product to build G, with three possible internal states s_1, s_2, s_3 and three possible key states k_1, k_2, k_3, where (s_i, k_j) represents a node. An edge $(s_i, k_j) \rightarrow (s_{i'}, k_{j'})$ exists in G if and only if $k_j \rightarrow k_{j'}$ exists in G_{KS} and $(s_i, k_j) \rightarrow s_{i'}$ exists in G_{BC}.

This observation slightly reduces the amount of computation/memory to build/store G: the number of vertices in G_{BC} is $v_{\mathsf{BC}} = 2^{t_{\mathsf{BC}}+t_{\mathsf{KS}}} + 2^{t_{\mathsf{BC}}}$ and the number of vertices in G_{KS} is $v_{\mathsf{KS}} = 2 \times 2^{t_{\mathsf{KS}}}$. This has to be compared with the $2 \times 2^{t_{\mathsf{BC}}+t_{\mathsf{KS}}}$ nodes in G. For example, in the particular case of the AES-128, this trick reduces the number of nodes from 2^{33} in G to $v_{\mathsf{BC}} = 2^{32} + 2^{16}$ in G_{BC} and 2^{17} in G_{KS} and mainly allows to apply an *early-abort* approach to prune edges in G in the online phase. More importantly, the total number of edges shrinks considerably from $e_{\mathsf{BC}} \cdot e_{\mathsf{KS}}$ to $e_G = e_{\mathsf{BC}} + e_{\mathsf{KS}}$, which equals to $2^{33.6} + 2^{22.15}$ in the case of AES-128 (these edge numbers are explained in Section 6.4.3).

At last, since G_r is the concatenation of r instances of G, we only need to store G to run computations on G_r and this further saves roughly a factor r. In the case where the cipher rounds are not the same, we may need to store x different instances of G for x distinct types.

6.3.2.1 The graph G_{BC}

The graph G_{BC} can be built by repeating the three following steps for all the $2^{t_{\mathsf{BC}}}$ possible truncated differences Δ_{in} on the input and all the $2^{t_{\mathsf{BC}}}$ possible truncated differences Δ_{out} on the output.

1. Compute all the possible truncated differences Δ_x that can be obtained from Δ_{in} through the P layer (on the backward direction, a truncated difference Δ_{out} stays the same when inverting the S layer).
2. For every Δ_x found, compute all the possible truncated differences Δ_k on the key state that can be obtained from $\mathsf{AK}^{-1}(\Delta_x \oplus \Delta_{out})$.
3. For every Δ_k found, add an edge in G_{BC} from input node (Δ_k, Δ_{in}) to output node Δ_{out} if none exists.

The time complexity to build G_{BC} depends on the average branching B_{P} of the P layer and on the average branching B_{xor} of the subkey XORing layer. It amounts to $2^{2t_{\mathsf{BC}}} \cdot B_{\mathsf{xor}} \cdot B_{\mathsf{P}}$

operations. The memory cost to store G_{BC} corresponds to the number of edges e_{BC} of G_{BC} and is upper bounded by $2^{2t_{\mathrm{BC}}} \cdot B_{\mathrm{xor}} \cdot B_{\mathrm{P}}$ since one operation on step 3 adds at most one edge. In the following, we denote $\mathrm{succ}_{\mathrm{BC}}(s,k)$ the set of successors of the state s in the graph G_{BC} using the key k.

To be able to evaluate properly the time and memory complexity of the related-key differential characteristics search tool, we give an estimation of the average branching factor B_{xor} in the XOR operation in the following Theorem 6.2. As an example, we estimate B_{P} in the case of AES-128 in Section 6.4.3.2.

Theorem 6.2. *The average branching B_{xor} in the XOR key addition is:*

$$B_{xor} = \sum_{z=0}^{t_{BC}} \sum_{i=0}^{t_{BC}} \sum_{j=0}^{t_{BC}} \frac{\binom{t_{BC}}{i}}{2^{t_{BC}}} \cdot \frac{\binom{t_{BC}}{j}}{2^{t_{BC}}} \cdot P_{\mathrm{and}}(t_{BC}, i, j, z) \cdot 2^z.$$

Proof. We denote $X[i]$ the i-th bit of the word X, and $\mathrm{HAM}(X)$ the Hamming weight of the word X. We recall from [ABNP+11] that for two random k-bit words A and B of Hamming weight a and b respectively, the probability that $\mathrm{HAM}(A \wedge B) = i$, where $\wedge$ stands for the bitwise AND function, is given by the formula

$$P_{\mathrm{and}}(k,a,b,i) = \frac{\binom{a}{i}\binom{k-a}{b-i}}{\binom{k}{b}} = \frac{\binom{b}{i}\binom{k-b}{a-i}}{\binom{k}{a}}. \tag{6.1}$$

All the branching in the $\oplus$ operation between two words A and B comes from the active bits in $A \wedge B$, and we have $2^{\mathrm{HAM}(A \wedge B)}$ possibilities. Therefore, we can deduce that

$$B_{\mathrm{xor}} = \sum_{z=0}^{t_{\mathrm{BC}}} \sum_{i=0}^{t_{\mathrm{BC}}} \sum_{j=0}^{t_{\mathrm{BC}}} \frac{\binom{t_{\mathrm{BC}}}{i}}{2^{t_{\mathrm{BC}}}} \cdot \frac{\binom{t_{\mathrm{BC}}}{j}}{2^{t_{\mathrm{BC}}}} \cdot P_{\mathrm{and}}(t_{\mathrm{BC}}, i, j, z) \cdot 2^z. \tag{6.2}$$

In the case of AES-128, this gives us $B_{\mathrm{xor}} = 2^{5.15}$. ■

6.3.2.2 The graph G_{KS}

The graph G_{KS} is built by simply going through all the $2^{t_{\mathrm{KS}}}$ possible key schedule internal state input truncated differences, checking which output truncated differences can be obtained through the KS layer and adding edges in G_{KS} accordingly[2]. The time and memory complexities depend on the average branching B_{KS} of the KS layer and amounts to $2^{t_{\mathrm{KS}}} \cdot B_{\mathrm{KS}}$ operations. The number of edges e_{KS} of G_{KS} equals $e_{\mathrm{KS}} = 2^{t_{\mathrm{KS}}} \cdot B_{\mathrm{KS}}$. In the sequel, we denote $\mathrm{succ}_{\mathrm{KS}}(k)$ the set of successors of the key k in graph G_{KS}. We evaluate B_{KS} in the case of AES-128 in Section 6.4.3.3.

6.3.3 Online phase

The online phase finds all the shortest paths in G_r with at most

$$r \cdot \left(\frac{v_G}{2} \cdot \log(\frac{v_G}{2}) + e_G \right)$$

[2]We assume that the key schedule is simple: given a truncated difference on the input, one can find each reachable truncated output difference in constant time. This assumption is weaker than the one from footnote 1, and verified by most ciphers since a very complex key schedule would make the whole primitive inefficient anyway.

Algorithm 6.1 – Search all the shortest paths in G_r.

```
1:  function SEARCH(G_r)
2:    Copy all nodes of G_r in a new graph G*_r
3:    for all v ∈ V_0, c(v) ← |v|          ▷ Initialize the starting nodes at their weight.
4:    for all v ∈ V_1, ..., V_r, c(v) ← ∞  ▷ The other nodes are not reachable yet.
5:    SORTLIST(V_0)                        ▷ Sort by cost c(v) of the nodes.
6:    for i = 1 → r do                     ▷ Loop over the r rounds.
7:      for all v' ∈ V_i, by increasing c(v') do   ▷ This ordering ensures the minimization.
8:        for all v ∈ succ(v') do
9:          α ← c(v') + |v|
10:         if c(v) = ∞ then               ▷ If the node v have not been visited yet,
11:           c(v) ← α                     ▷ we update its cost,
12:           Add the edge v' → v to G*_r  ▷ and we add the associated edge to G*_r.
13:         else if c(v) = α then          ▷ If we can reach it at the same cost,
14:           Add the edge v' → v to G*_r  ▷ also add the edge to G*_r.
15:     SORTLIST(V_i)                      ▷ Sort the next nodes by increasing costs.
16:   return G*_r                          ▷ Return the graph of shortest paths.
```

computations and memory $r \cdot e_G$ linear in the number of rounds r. This is possible because G_r is a vertex-weighted directed acyclic graph. Since the edges have a constant weight (the number of active S-Boxes, i.e. the weights, are on the nodes and not the edges), the function we want to minimize for each node $v \in V_i$, $i \in [1, r]$ is:

$$|v| + \min_{v' \in \text{pred}(v)} \left(c(v')\right), \tag{6.3}$$

where $\text{pred}(v) \subseteq V_{i-1}$ is the set of all predecessors of v and $c(v')$ represents the cost of the shortest path to v'. In other words, assuming that we know the shortest path costs to all the nodes $v' \in V_{i-1}$, we find the shortest path to any $v \in V_i$ by choosing the predecessor of v with the minimal cost.

This can easily be done by creating a list containing all the nodes $v' \in V_{i-1}$ sorted increasingly according to the cost of their shortest path $c(v')$. Then, starting from the cheapest v' and ending to the most expensive one, we set the shortest path cost of all the successors v of v' to $|v| + c(v')$ if and only if the cost of v was not set yet (see Algorithm 6.1). This is an improvement over the simple shortest path computation in a directed acyclic graph using a topological order since we can take advantage of the vertex-weighted property.

In practice, we iteratively build a simpler vertex-weighted directed acyclic graph G_r^* from G_r (all the nodes are the same, but with less edges), for which each node $v \in V_i$ has a cost equal to the cost of the shortest path to v in G_r, and an edge leading to $v \in V_i$ represents one of the shortest paths to v (see Figure 6.8).

At this point, in the graph G_r^* the costs assigned to all the nodes v in V_r represent the cost of the shortest path to v in G_r. If v_G represents the number of vertices and e_G the number of

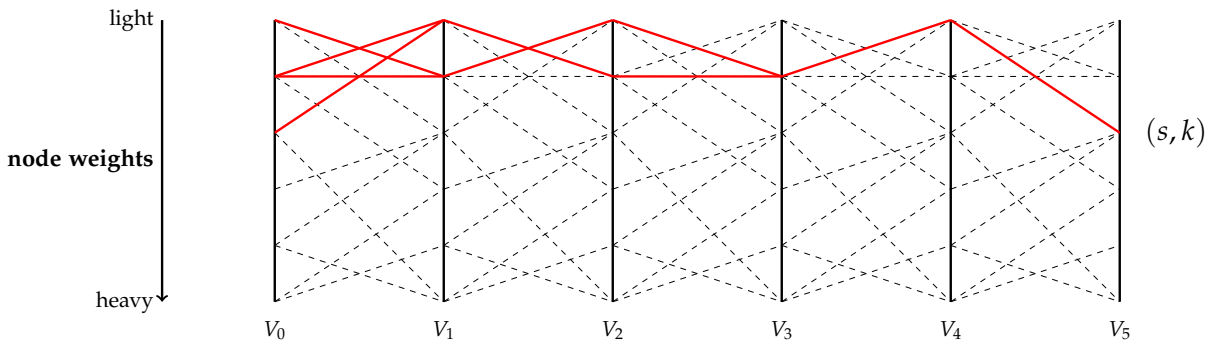


Figure 6.8: Example of shortest paths in G_5. The dashed edges form an example of a simplified G_5. The thick edges describe paths in the subgraph G_5^* that are shortest paths in G_5 to node (s, k). All the nodes in G_5^* are sorted according to their weight, the top being the cheapest ones.

edges in the graph G, then the complexity of the shortest path search is about

$$r \cdot \left(\frac{v_G}{2} \cdot \log(\frac{v_G}{2}) + e_G \right)$$

operations: the $\frac{v_G}{2} \cdot \log(\frac{v_G}{2})$ term comes from the construction of the sorted list of the nodes at each round, and the e_G term is the number of edges visited during each round as we visit all of them. Note that this is an upper bound on the complexity since we do not need to go through all $\frac{v_G}{2}$ nodes every rounds, but only a subset of them, and we may cut some edges among all the e_G ones. The term $e_G = e_{\mathsf{BC}} + e_{\mathsf{KS}}$ dominates the complexity, and since $e_{\mathsf{BC}} \gg e_{\mathsf{KS}}$, it can be approximated by the number $e_{\mathsf{BC}} \leq 2^{(n+k)/c} \times 2^{n/c}$ of edges in G_{BC}. Hence, the total time complexity is $O(r \cdot 2^{(2n+k)/c})$ for r rounds.

In order to get *all* the shortest paths in G_r, we need to store at each node $v \in V_i$ not only the first shortest path found to v but all of them (lines 13 and 14 in Algorithm 6.1). In general, this number is very small and never exceeds the total number of shortest paths anyway. In the worst case where all paths are the shortest, it amounts to the total number of edges $r \cdot e_G$.

As explained previously, in practice we do not use the graph G directly, but the two separate graphs G_{BC} and G_{KS}. We can adapt the Algorithm 6.1 for this setting: in order to build G_r^*, we replace the **for all** loop of line 8 that iterates over all nodes $v' = (s_i, k_i) \in V_i$ by two **for all** loops that describe all $k_{i+1} \in \mathrm{succ}_{\mathsf{KS}}(k_i)$ and all $s_{i+1} \in \mathrm{succ}_{\mathsf{BC}}(s_i, k_{i+1})$.

In [Mat94b], Matsui introduces an argument equivalent to the A^* optimization for path-finding or graph traversal algorithms [HNR68] that allows to prune the majority of the edges of G and to avoid the evaluation of many sets of successors. If we know the costs c_k of all k-round characteristics, $1 \leq k \leq n-1$, and we target an n-round characteristic of cost at least c_n, then we can consider only the nodes from V_0 that have a cost at most $c_n - c_{n-1}$, and the ones in V_1 that have a cost at most $c_n - c_{n-2}$. Intuitively, after one round has been passed, we know that we paid *at least* c_1, and since there are $n-1$ remaining rounds to pass, we will need to pay *at least* c_{n-1}. In terms of intervals of costs, for each of the V_i, we only need to consider nodes that have costs in $[c_i, c_n - c_{n-i}]$, $0 \leq i \leq n$ assuming $c_0 = 0$. To take advantage of the A^* heuristic, we sort the sets of successors in both graphs, so that we can perform an extreme pruning of the edges whenever the updated costs exceed the current interval, in an early-abort manner.

In the next section, we detail how to extend this algorithm to the case of AES-like ciphers, and we then give the consequences of the search for this class of ciphers in Section 6.5.

6.4 Enhanced Markov process

In this section, we study the special SPN case of AES-like ciphers, where the P layer is composed of SR and MC (see Section 6.2). In this situation, we are able to compress the states by making some observations on one AES-like round. This saves a significant amount of computations and memory. Moreover, we also evaluate the number of nodes and edges of the graph G_{BC} (and of G_{KS} in the case of total-AES-like ciphers) so as to be able to estimate precisely the time and memory complexities.

6.4.1 The Markov assumption and actual differences

Our search algorithm (Section 6.3) only works because we place ourselves in a Markov process scenario. Depending on the analysis we want to conduct on the studied block cipher, we may want more than just pure truncated differentials. This scenario indeed gives a structural evaluation of the cipher in regard to the related-key model, but we may want to instantiate the truncated differences into actual differences. With our current approach, the pure truncated characteristics found may not be valid since the Markov process did not propagate some constraints along the rounds, which includes equality conditions between actual differences, or their difference, or linear relations, etc. For example, choosing a subkey transition in one round of the AES-128 key schedule affects the possible choices for the next subkeys because of its strong linearity (see Figure 6.9). To address this deeper analysis, we propose two fundamentally different approaches.

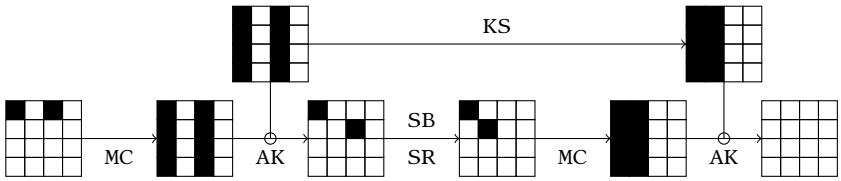


Figure 6.9: Example of linear incompatibility in the case of AES-128: the linearity of the key schedule imposes all the active columns $[a, b, c, d]^T$ to be equal, which contradicts $\mathbf{M} \cdot [x, 0, 0, 0]^{\mathrm{T}} \oplus [x', 0, 0, 0]^{\mathrm{T}} = \mathbf{M} \cdot [y, 0, 0, 0]^{\mathrm{T}} \oplus [0, y', 0, 0]^{\mathrm{T}}$ in the first key addition (AK).

The first one is the filtering. By starting with all the best pure truncated differential characteristics we have found with the search algorithm, we filter them one by one, until we reach one that fulfills all the implicit necessary conditions imposed by the block cipher. Depending on the studied block cipher, we may not find one with the minimal cost: this method is not guaranteed to find the best differential characteristics with all the extra conditions.

The second method is to consider the same algorithm, but propagating all the information such that the Markov assumption is verified again. Then, we can directly verify the implicit conditions and eventually be sure that the search finds valid characteristics. In return, the overall search introduces more complex operations since the graphs G_{BC} and G_{KS} are bigger.

A mix of the two methods seems to be the best strategy in practice and we give in the following the detailed study for the case of AES-128. With this more complete Markov process, the overall complexity of the algorithm is the same as before, except that we perform extra computations for each visited edges to check for linear consistency by solving small linear systems, which may all be precomputed.

6.4.2 Block cipher state compression

In the case of AES-like ciphers, the search space can be drastically reduced thanks to some observations on the round function. We introduce a new compressed view of the block cipher state.

Definition 6.4 (Compressed state). Let Δ be a state of truncated differences considered as a d-column square matrix $\Delta = [\Delta^{\cdot,1}, \ldots, \Delta^{\cdot,d}]$. We denote $\overline{\Delta}$ the image of Δ by the non-injective function:

$$\Delta \longrightarrow \overline{\Delta} = \left[\left|\overline{\Delta}^{\cdot,1}\right|, \ldots, \left|\overline{\Delta}^{\cdot,d}\right| \right], \text{ where: } \quad \forall j,\ \overline{\Delta}^{\cdot,j} = \left|\Delta^{\cdot,j}\right|.$$

We call $\overline{\Delta}$ the **compressed state** of Δ.

A compressed state as defined above only describes the number of active cells there are in each column of the block cipher internal state. The motivation to introduce such a compressed representation lies in the MDS property of the underlying matrix $\mathbf{M}$ of the MC layer. Indeed, to get the possible output truncated patterns, we only need the number of active cells on the input, i.e. the weight of that column.

Theorem 6.3. *Let $\mathcal{E}$ be an AES-like cipher with n-bit blocks using a k-bit internal state in the key schedule. Both states are viewed as vectors of c-bit cells. With state compression, the time complexity of Algorithm 6.1 to find all the best differential characteristics for r round of $\mathcal{E}$ becomes*

$$\mathcal{O}\left(r \cdot 2^{\sqrt{\frac{n}{c}}\log_2(\frac{n}{c})+\frac{n}{c}}\right),$$

with a small hidden constant factor. ■

6.4.3 Evaluating the number of nodes/edges of $G_{\mathbf{BC}}$ and $G_{\mathbf{KS}}$

We now evaluate the characteristics of the two graphs G_{BC} and G_{KS} in the case of AES-like block ciphers, and give numerical example of the particular case of AES-128.

6.4.3.1 Number of nodes

With this new compressed representation, it is easy to see that the internal state can now take $(t+1)^t$ possible values instead of $2^{t_{\mathrm{BC}}}$ and G_{BC} now contains $(t+1)^t \times (2^{t_{\mathrm{KS}}}+1)$ nodes. The graph G_{KS} is not affected by the compression of states since we only alter the ones in the message part.

6.4.3.2 Number e_{BC} of edges in $G_{\mathbf{BC}}$

The average branching factor B_{P} of the P layer for AES-like ciphers corresponds to the one of the **MixColumns** layer: for a single column with $i \in [1, t]$ active differences in its input, we may choose the location of j active difference in its output, with $j \in [t+1-i, t]$ to respect the MDS property of the underlying $t \times t$ matrix $\mathbf{M}$. Alternatively, we may choose the location of $j \in [0, i-1]$ inactive differences, which leads to the following formula of B_{P} for the t columns where the leading 1 comes from the full inactive column:

$$B_{\mathsf{P}} = \left(2^{-t}\left(1 + \sum_{i=1}^{t}\binom{t}{i}\sum_{j=0}^{i-1}\binom{t}{j}\right)\right)^{t} = \left(1 + \sum_{i=1}^{t}\sum_{j=0}^{i-1}\binom{t}{i}\binom{t}{j}\right)^{t}. \tag{6.4}$$

In the case of AES-128, we obtain an average branching of $B_{\mathsf{P}} = 2^{2.55}$ for one column of the P layer. However, to estimate the number e_{BC} of edges in G_{BC}, we have to consider not only the P layer but also the subkey XOR layer that comes right after it. We cannot use the value we previously computed for B_{P} since in Theorem 6.2, we assumed that the two words XORed were random. However, in the current situation, the hamming weight of the column words arriving from the P layer presents a strong bias towards higher values: this is due to the branching effect that tends to populate more dense than sparse words. Therefore, we need to tweak the formula in order to take in account the hamming weight probability of the column words A and B involved:

$$B_{\mathsf{xor}} = \left(\sum_{z=0}^{t}\sum_{a=0}^{t}\sum_{b=0}^{t}\mathbf{Pr}[\mathrm{HAM}(A)=a]\cdot\mathbf{Pr}[\mathrm{HAM}(B)=b]\cdot P_{\mathsf{and}}(t,a,b,z)\cdot 2^{z}\right)^{t}, \tag{6.5}$$

where

$$\mathbf{Pr}[\mathrm{HAM}(B)=b] = \binom{t}{b}/2^{t} \tag{6.6}$$

since the subkey column words are not biased, and P_{and} is defined in the proof of Theorem 6.2. The hamming weight probabilities $\mathbf{Pr}[\mathrm{HAM}(A)=a]$ concerning the column words coming from the P layer are computed by

$$\mathbf{Pr}[\mathrm{HAM}(A)=a] = \frac{\sum_{i=1}^{t}\binom{t}{i}\binom{t}{a}\cdot 1_{a<i}}{1+\sum_{i=1}^{t}\sum_{j=0}^{i-1}\binom{t}{i}\binom{t}{j}}. \tag{6.7}$$

In the case of AES-128, the total branching of both P layer and XOR layer amounts to $2^{16.88}$. One can see that the branching here is very strong compared to the number of nodes in the graph G_{BC}, which indicates that this bipartite graph is dense. Therefore, we can instead upper bound the number of edges e_{BC} by reasoning on the number of nodes $e_{\mathrm{BC}} \leq (t+1)^{td}\cdot 2^{t_{\mathrm{BC}}}$, since two nodes cannot share more than one edge. This gives $e_{\mathrm{BC}} \leq 2^{34.6}$ for AES-128, which is very close to the reality since in practice, we measure $2^{33.6}$ edges for G_{BC}.

6.4.3.3 Number e_{KS} of edges in $G_{\mathbf{KS}}$

In the case of total-AES-like ciphers where we generalize the key scheduling algorithms, we can compute an estimation of the average branching factor B_{KS} of the KS layer to evaluate the

number of edges in G_{KS}. Remember that any S-Box application has no effect on the truncated differentials search branching, so we only need to consider the XOR operations.

In order to obtain this estimation, we model the total-AES-like ciphers key schedule as the following operations:

$$A'_0 = A_0 \oplus R_0; \quad A'_1 = A'_0 \oplus R_1; \quad \cdots \quad A'_{t-1} = A'_{t-2} \oplus R_{t-1}, \tag{6.8}$$

where all words represent a t-bit key state column and A_0 and all R_i are random t-bit numbers. If we denote B^i_{KS} the average branching concerning the i-th operation (i.e. column), then we have

$$B_{\mathsf{KS}} = \prod_{i=0}^{t-1} B^i_{\mathsf{KS}}. \tag{6.9}$$

Note that it is easy to evaluate the average branching B^0_{KS} of the first operation, but hard to do for the remaining ones. Indeed, in the first operation we consider that both A_0 and R_0 are random t-bit numbers, but for the next operations the words $A'_0, \ldots, A'_{t-2}$ are not random looking values anymore as their hamming weight are slightly biased towards higher values due to the effect of the branching in the previous operation. This is the very same problem that appear for combining the branching of the P and the XOR layer in the previous section.

We then use the same formula of Equation (6.5) to compute B^i_{KS}, remarking that A_0 and R_i are considered as not biased. This means that

$$\mathbf{Pr}[\mathrm{HAM}(A_0) = b] = \mathbf{Pr}[\mathrm{HAM}(R_i) = b] = \frac{\binom{t}{b}}{2^t}, \tag{6.10}$$

while the biased probabilities $\mathbf{Pr}[\mathrm{HAM}(A'_i) = a]$ are computed with:

$$\mathbf{Pr}[\mathrm{HAM}(A'_i) = a] = \tag{6.11}$$
$$\left(\sum_{z=0}^{t} \sum_{a=0}^{t} \sum_{b=0}^{t} \sum_{y=z}^{2z} \mathbf{Pr}[\mathrm{HAM}(A'_{i-1}) = a] \cdot \mathbf{Pr}[\mathrm{HAM}(R_i) = b] \cdot P_{\mathrm{and}}(t,a,b,z) \cdot \binom{z}{y-z} \cdot \frac{1_{i+j-y=a}}{2^z} \right)^t.$$

We can now estimate the number of edges $e_{\mathsf{KS}} = 2^{t_{\mathsf{KS}}} \cdot B_{\mathsf{KS}}$ in G_{KS}. For AES-128, we obtain an average branching for the KS layer of $B_{\mathsf{KS}} = 2^{6.15}$. Our model and our assumptions seems to be sound since in practice, we measure an average branching of about $B_{\mathsf{KS}} = 2^{6.22}$. Overall, building/storing G_{KS} during the precomputation phase should not require more than $e_{\mathsf{KS}} = 2^{22.15}$ computations and memory.

6.4.4 More complete Markov process

The related-key differential characteristics outputted by the algorithm from Section 6.3 are valid only when one deals with truncated differences and these characteristics give an indication on the structural security provided by the AES-128 key schedule. However, it turns out that none of them can be instantiated with actual differences, because of inconsistencies in some linear constraints. Therefore, we apply the techniques proposed in Section 6.4.1: at the cost of a

bigger graph G to handle, we first add some more information in the Markov process both on the representation of the key schedule state and the internal permutation state, and we then filter the best characteristics obtained and hope to find one that can be instantiated with actual differences.

6.4.4.1 New state compression

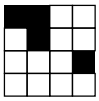
In order to look for actual differences characteristics for AES-128, we slightly reduce the state compression used for AES-like ciphers presented in Definition 6.4 to decrease the compression factor.

Definition 6.5 (Semi-compressed state). Let Δ be a state of truncated differences considered as a d-column square matrix $\Delta = [\Delta^{\cdot,1}, \ldots, \Delta^{\cdot,t}]$. We denote $\tilde{\Delta}$ the image of Δ by the non-injective function:

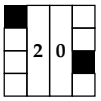
$$\Delta \longrightarrow \tilde{\Delta} = \left[|\tilde{\Delta}^{\cdot,1}|, \ldots, |\tilde{\Delta}^{\cdot,d}| \right], \text{ where: } \quad \forall j,\ \tilde{\Delta}^{\cdot,j} = \begin{cases} \Delta^{\cdot,j} & \text{if } |\Delta^{\cdot,j}| = 1, \\ |\Delta^{\cdot,j}| & \text{otherwise.} \end{cases}$$

We call $\tilde{\Delta}$ the **semi-compressed state** of Δ.

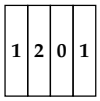
A semi-compressed state as defined above only describes the number of active cells there are in each column of the block cipher internal state, and in the event that there is only one, keep tracks of its position (see example in Figure 6.10). The stored position in the particular case of a single input active cell provides the additional information needed by the Markov process to construct actual differences characteristics without inconsistencies.



(a) Truncated state.



(b) Semi-compressed state.



(c) Compressed state.

Figure 6.10: Example of compressed truncated state (c) and semi-compressed truncated state (b) from a truncated state (a).

6.4.4.2 Representation of truncated subkeys

Keeping in mind that the unique t^2-bit truncated difference information is not enough for the Markov process to find actual differential characteristics, we provide a more complete coding of the subkeys. Namely, we introduce a representation that allows to encode some particular cases of linear constraints between the differences, which are needed later to solve systems of linear equations.

Definition 6.6 (Subkey representation). Let $\delta = [\delta^{x,y}]$ be the actual difference in a subkey k and Δ its truncated counterpart, where the t actual differences on each column j are related by a possibly empty system of linear equations $\mathcal{S}_j$. If the following, we note $\cong$ the equivalence between two systems of equations. We call **semi-truncated difference** of the key k, and denote

it $\tilde{k}$, the t-column square matrix such that:

$$\tilde{k} = \left(b, \left[\tilde{k}^{\cdot,j}\right]\right),$$

$$\text{where:} \quad \forall j,\ \tilde{k}^{\cdot,j} = \begin{cases} x & \text{if } \mathcal{S}_j \cong \mathbf{M}x = 0, & \text{with } |x| = 1, & \text{(type 1)} \\ (x,y) & \text{if } \mathcal{S}_j \cong \mathbf{M}x \oplus y = 0, & \text{with } |x| = |y| = 1, & \text{(type 2)} \\ \Delta^{\cdot,j} & \text{otherwise.} & & \text{(type 3)} \end{cases}$$

and b may be $\top$ if and only if all columns of the same type are equal in actual differences; it always equals to $\bot$ otherwise. Additionally, we call an **extended state** a couple $(\tilde{s}, \tilde{k})$ of a block cipher semi-compressed state $\tilde{s}$ and a key schedule semi-truncated difference $\tilde{k}$ and denote $\left|(\tilde{s}, \tilde{k})\right|$ its weight.

This definition behaves as a trade-off between the actual differences, which amounts to a total of $2^{c \cdot t^2}$ different differences but keeps the whole relations between the differences, and the truncated differences which compress to the minimum information on each difference, where there are only 2^{t^2} of them.

As an example, in the case $t = 4$, the two linear systems $\mathcal{S}_1 : \mathbf{M}[a,0,0,0]^{\mathrm{T}} = 0$ and $\mathcal{S}_2 : \mathbf{M}[0,0,0,b]^{\mathrm{T}} = 0$ falls into the type 1 category but results in two different columns, where we store $[a,0,0,0]^{\mathrm{T}}$ and $[0,0,0,b]^{\mathrm{T}}$ respectively, or equivalently the position of the only active difference, 0 and 3. In this case, the bit b cannot give a relation between a and b since the two columns are not of the same type; but if $\mathcal{S}_2$ were $\mathcal{S}_2 : \mathbf{M}[c,0,0,0]^{\mathrm{T}} = 0$, it could, which would mean that $a = c$ in terms of actual differences.

6.4.5 Explanations

We explain here the choice of the extra information added in the Markov in comparison to our preliminary tool. Namely, we keep more information in some special cases to avoid loosing information of those particular values. In the block cipher, the columns of weight exactly 1 are stored uncompressed (Definition 6.4); in the key schedule, we encode the position of active differences in two particular cases (types 1 and 2, Definition 6.6).

Those enhanced representations come into the picture when iterating over the sets of successors in the two graphs G_{BC} and G_{KS}. To construct the set $\mathrm{succ}_{\mathrm{KS}}(k)$ of successors of a semi-truncated key k, we consider in a very straightforward manner the sum of two columns and deduce which one(s) can be reached trough the key schedule algorithm.

In the graph G_{BC}, we want to find the set $\mathrm{succ}_{\mathrm{BC}}(s,k)$ of successors of a semi-compressed state s and a semi-truncated difference k. To do so, we first construct the set of all truncated state after the MC layer and check which truncated state s' can be obtained after the $\mathsf{AK}(k)$. For each of those s', we may write a homogeneous system of linear equations corresponding to the two linear AES transformations MC and AK, using the additional information on the columns of k to check whether the transition is indeed valid. If the input semi-compressed state s is associated to n actual truncated state, then we write n systems and check if at least one has non-trivial solutions. In practice, we precompute all the possible systems.

Consequently, the cost update function of the Markov process is done as before, with extra checks on the transitions/edges available with the added information at each node. This

enhanced Markov process thus leads to graphs with more nodes than the one for pure truncated differentials, but proportionally, there are fewer edges because of the tighter transition function.

6.5 Applications to SPN and AES-128

6.5.1 Structural evaluation of SPN AES-like ciphers

We present here the results on the structural evaluation of the AES-like ciphers in regard to the related-key model, which provides an estimation of the security provided by their key schedule. Namely, we ignore the semantic definition of the S-Box and the MDS matrix, and we are only interested in how they can interact in the related-key setting.

The results are measured in terms of number of active S-Boxes as in [DR02], and presented in Table 6.1. Lines 2 and 3 of the table provide the minimum number of active S-Boxes (line 2) for any number of rounds when implementing an AES-like cipher with the same size as AES-128, and the number of truncated characteristics that reach that bound (line 3). In that case, we count the number of active S-Boxes in both the state and the subkeys, whereas in lines 4 and 5 of Table 6.1, we consider the case of the hash function setting where the block cipher and its key schedule can be considered somewhat independently by an adversary.

Rounds	**1**	**2**	**3**	**4**	**5†**	**6**	**7**	**8†**	**9**	**10†**
$\mathbf{min}(C_{KS}+C_{BC})$	0	1	3	9	11	13	15	**21**	23	**25**
Truncated Char. ($\log_2$)	–	4.52	6.58	10.46	5.00	13.26	16.17	21.34	14.90	21.38
$\mathbf{min}(\mathbf{max}(C_{BC},C_{KS}))$	0	1	3	6	7	9	11	14	15	**17**
Truncated Char. ($\log_2$)	–	4.00	10.00	11.73	10.00	18.92	23.64	>30	>30	>30

Table 6.1: For the AES-128 cipher on r rounds, this table shows: (1) the minimal number $C_{KS}+C_{BC}$ of active S-Boxes in **both** the key schedule C_{KS} and in the block cipher C_{BC} achievable in truncated differential characteristics; and (2), the same figures for the minimal number $\max(C_{BC},C_{KS})$ for the hash function setting. Lines 3 and 5 count the number of distinct truncated characteristics that reach that bound. † For $r \in \{5,8,10\}$, see the following figures for the characteristics.

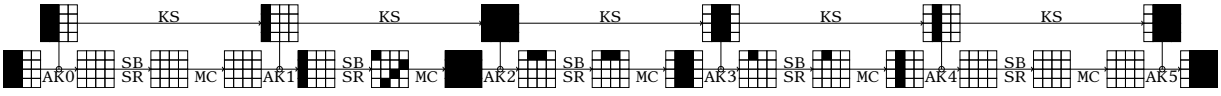


Figure 6.11: Best 5-round truncated differential characteristic for AES-128 with 11 active S-Boxes.

From these results, we can now state the following impossibility result (Theorem 6.4) on the structure of AES-128.

Theorem 6.4. *It is impossible to prove the security of the full* AES-128 *against related-key differential attacks without considering the differential property of the S-Box when two keys verify a certain relation. It is impossible to prove the security of the full* AES-128 *in the hash function setting without considering both the differential property of the S-Box and the P layer.*

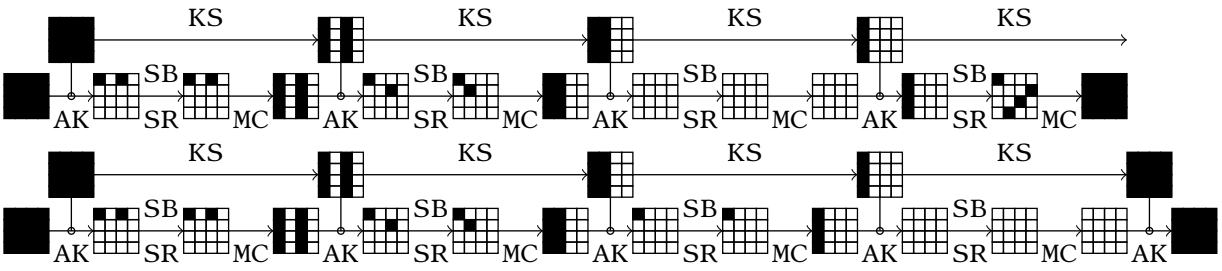


Figure 6.12: Best 8-round truncated differential characteristic for AES-128 with 21 active S-Boxes.

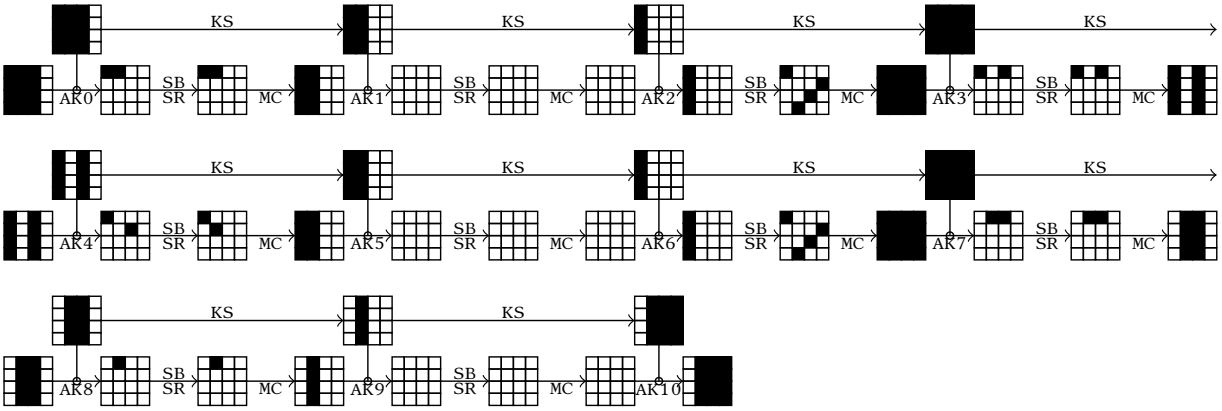


Figure 6.13: Best 10-round truncated differential characteristic for AES-128 with 25 active S-Boxes.

In particular, the results in the related-key model extends the famous result by Daemen and Rijmen on the AES structure in the single-key model from [DR02]. With no difference in the key, they have shown that there are 55 actives S-Boxes for 10-round AES, which makes the cipher resistant to differential cryptanalysis as long as the S-Box verifies $p_{max} \leq 2^{-3}$. In the related-key setting, we show that the S-Box needs to verify at least $p_{max} \leq 2^{-6}$.

Proof. First, in the case where we consider related-key attacks where two keys are related if their difference verify a certain relation (line 2), we remark that for 10 rounds there exists a truncated differential characteristic counting only 25 S-Boxes. As we discussed before, this means that a differential analysis would run in p_{max}^{-25} operations. Consequently, the structure of AES-128 on its own is not enough to prove the resistance to related-key attacks for any ciphers in this class, we at least need to add a criteria on the S-Box via p_{max}.

Second, with an S-Box on c bits ($c = 8$ in the AES), the maximal theoretical p_{max} that can be obtained is $2^{-(c-1)}$: consequently, the largest number of rounds that our structural analysis could attack for AES-like ciphers is 7 rounds. Indeed, for 7 rounds, the 15 active S-Boxes give a differential analysis in $p_{max}^{-15} \geq 2^{105}$, which might be smaller than 2^{128}. We note that we do not know how to construct an almost-perfect permutation on c bits acting as an S-Box with the maximal differential probability $2^{-(c-1)}$. We recall the open problem previously defined in Open Problem 1, which challenges to find an almost-perfect permutation in $\mathrm{GF}(2^c)$, with $c > 6$. The S-Box chosen in the AES implements a composition of an affine transformation on the inverse mapping, and reaches $p_{max} = 2^{-(c-2)}$. With that weaker S-Box, the largest number of rounds that our structural analysis could attack is 8 rounds. Indeed, for 8 rounds, the 21

active S-Boxes give a differential analysis in $p_{max}^{-21} \geq 2^{126}$, which might be smaller than 2^{128}.

When we instantiate the P layer by the one of the AES-128, we observe that none of the $2^{16.17}$ characteristics found on 7 rounds by our search algorithm nor the $2^{21.34}$ ones for 8 rounds can be instantiated due to linear constraints coming from the key schedule. This means that proving or disproving the security of the AES-128 in the related-key setting needs to consider both the differential properties of the S-Box and the linear equations of the P layer.

From an instantiated P layer, we can write a system of linear equations $\mathcal{Q}$ whose solutions are the values of all the truncated differences of the characteristic. Therefore, choosing P such that $\mathcal{Q}$ can be made inconsistent on a small number of rounds brings more security than a random P. Our tool can be used to write this system of linear equations for any truncated differential characteristic.

Finally, for 10 rounds in the hash function setting, there exists characteristics with only 17 active S-Boxes in the internal state part and 8 in the key schedule part (Figure 6.13). For the AES-128, in the best case, the differential probability equals $2^{-6 \cdot 17} = 2^{-102}$. In this setting, the adversary is supposed to have full control over the input of both the key schedule and the block cipher, that is why we considered $\max(C_{BC}, C_{KS})$ as an objective function to minimize in our search algorithm of Section 6.3. As the previous structural results, this also means that one cannot prove the security of the full AES-128 against differential cryptanalysis in that setting by only analyzing its structure. ■

6.5.1.1 Complexity evaluation

Our tool has found those results for any number of rounds in a few seconds on a single regular processor. We note that the minimal characteristics in the single-key scenario are also found quasi-instantaneously. As a practical evaluation of the number of operations in terms of number of costs update (line 9 of Algorithm 6.1), we measured at most $2^{21.31}$ updates in this case, for the analysis on 10 rounds.

6.5.2 Differential characteristics results for AES-128

Theorem 6.5. *After 6 rounds, there is no related-key differential characteristic for* AES-128 *with a probability higher than* 2^{-128}.

The related-key differential characteristics presented in the previous section are valid only when one deals with truncated differences, and these characteristics give an indication on the structural security provided by the AES-128 key schedule. However, due to the choice of the P layer in AES-128, it turns out that none of them can be instantiated with actual differences, because of inconsistencies in some linear constraints. To overcome this difficulty, we implement the ideas exposed in Section 6.4, and at the cost of a bigger graph G to handle, we first add some more information in the Markov process both on the representation of the key schedule state and the internal permutation state, and we then filter the best characteristics obtained and hope to find one that can be instantiated with actual differences. Our algorithm performs a search fundamentally different from [BN10], but it finds again and more efficiently the same results.

By a system resolution, we show that from a truncated differential characteristic, we can decide whether it can be instantiated with actual differences, and even find an associated differential characteristic with the greatest probability. To do so, we need to write down the system of linear equations which exists from the cipher definition. In the case of the AES, there are lots of linear constraints in the key schedule, and others at the **MixColumns** layer. To express those equations, we choose a set of independent variables $\mathcal{B}$ such that any actual difference of the differential characteristic can be written as a linear combination of variables from $\mathcal{B}$. In the case of the AES, we can write all the equations with a basis $\mathcal{B}$ of variables from the key schedule; for example, the $t^2 - t$ last cells from the first subkey k_0, and each cell of the first column that goes out of the S-Box in the following subkeys, $k_1, \ldots, k_r$ (see Figure 6.14).

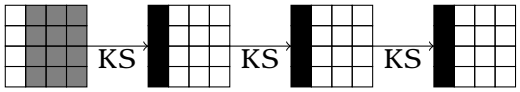


Figure 6.14: Variables from the key scheduling algorithm used in the system resolution in the case of AES-128 for $r = 3$ key schedule steps ($t = 4$). The ones used in k_0 (resp. $k_i, i > 0$ are in colored in gray (resp. black).

Once the system $\mathcal{S}$ of linear equations has been written, we apply the Gauss-Jordan elimination algorithm to transform it into reduced row echelon form and compute a basis of its kernel. We note that we want more than a non-trivial solution to the system; namely, we want each subsystem of $\mathcal{S}$ corresponding to each round to have non-trivial solutions. This is taken care of by the enhanced Markov process that we introduced to deal with actual differences. In the event that the nullity ν of $\mathcal{S}$ of the system is not null, we can get as many as $2^{c \cdot \nu}$ different possibilities to set the values for the actual differences of the truncated differential characteristic and any of them would conform to all the linear constraints.

In a second step, we need to take care of the non-linear constraints; namely, that each pair of input/output actual differences $(\delta_{in}, \delta_{out})$ of the S-Box provide a non-null entry in the DDT. From the system $\mathcal{S}$, we can write each δ_{in} and δ_{out} as a linear combinations of variables from the basis $\mathcal{B}$ and gather all the *different* transitions in a set $\mathcal{D} = \left\{(\delta_{in}^k \rightarrow \delta_{out}^k)\right\}$. Depending on the truncated differential characteristic, there may be several transitions which are equal: in the end, there are $|\mathcal{D}|$ different ones. Finally, we enumerate all the elements of the previously computed kernel to find one which validates all the transitions in $\mathcal{D}$.

Furthermore, each pair $(\delta_{in}^k, \delta_{out}^k) \in \mathcal{D}$ with a certain repetition α_k in the characteristic goes along with a certain probability p_k (depending on the DDT), which contributes to the probability p of the final differential characteristic:

$$p = \prod_{\mathcal{D}} p_k^{\alpha_k}.$$

Thus, if there are several kernel elements that validate all the transitions of $\mathcal{D}$, then we may prefer the one that maximize p.

As an example, our tool has found again the best truncated differential characteristic on 5 rounds of AES-128 with 17 active S-Boxes, and also found how to achieve the greatest probability 2^{-105} by instantiating the differences. This has to be compared with the upper bound of $2^{-6 \cdot 17} = 2^{-102}$ given in [BN10] since in the best case, all the AES active S-Boxes have maximal differential probability 2^{-6}. Trying all the possible differences that instantiate this

truncated differential characteristic, we show that we cannot reach that bound, but we can only set 15 out of 17 S-Boxes to the maximal differential probability (see Figure 6.19 for the actual differential characteristic).

The following Table 6.2 reports the best related-key characteristics found by our tool on AES-128 up to 5 rounds, with their respective highest achievable probabilities. From 6 rounds, exhaustive search runs faster that any related-key attacks in this class.

Rounds	**1**	**2**	**3**	**4**	**5**
$\mathbf{min}(C_{KS} + C_{BC})$	0	1	5	13	17
$\mathbf{max}\log_2(p)$	0	-6	-31	-81	-105
Figure/Table ref.	–	6.15/6.3	6.16/6.4	6.17, 6.18	6.19/6.6

Table 6.2: For the AES-128, related-key attacks are faster than exhaustive search only up to 5 rounds. Our tool retrieved the previous known results and provide the real differential characteristics with maximum probability.

Complexity evaluation

For 5 rounds, the online phase of our tool performs $2^{35.36}$ cost updates (line 9 of Algorithm 6.1, for the transition between V_0 and V_1), which takes about one hour in a parallelized version of the shortest path finding algorithm on a 12-core machine. The precomputation step completes in half an hour on this machine and needs 60GB of memory to store precomputed tables, notably the G_{KS} graph.

Best characteristic on 2 rounds

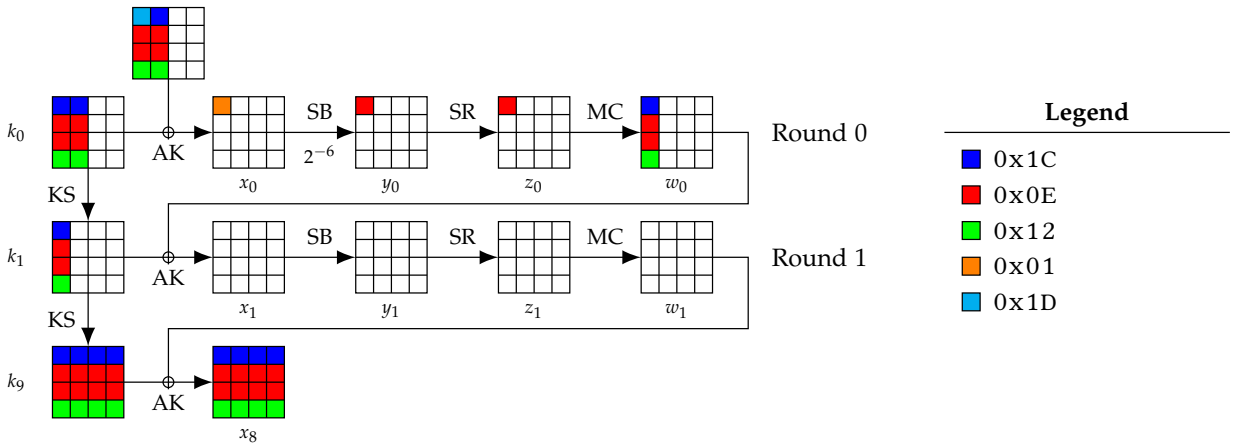


Figure 6.15: The best differential characteristic on two rounds of AES-128, which has a probability $p = 2^{-6}$. There are $2^8 - 1$ similar characteristics with the same differential probability. See also Table 6.3.

Table 6.3: The best differential characteristic on two rounds of AES-128, which has a probability $p = 2^{-6}$. The two lines for state differences are respectively the input difference after key addition and the output difference. See also Figure 6.15.

Round	State differences	Key differences
Plaintext	1D0E0E12 1C0E0E12 00000000 00000000	
0	01000000 00000000 00000000 00000000 1C0E0E12 00000000 00000000 00000000	1C0E0E12 1C0E0E12 00000000 00000000
1	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	1C0E0E12 00000000 00000000 00000000
Ciphertext	1C0E0E12 1C0E0E12 1C0E0E12 1C0E0E12	1C0E0E12 1C0E0E12 1C0E0E12 1C0E0E12

Best characteristic on 3 rounds

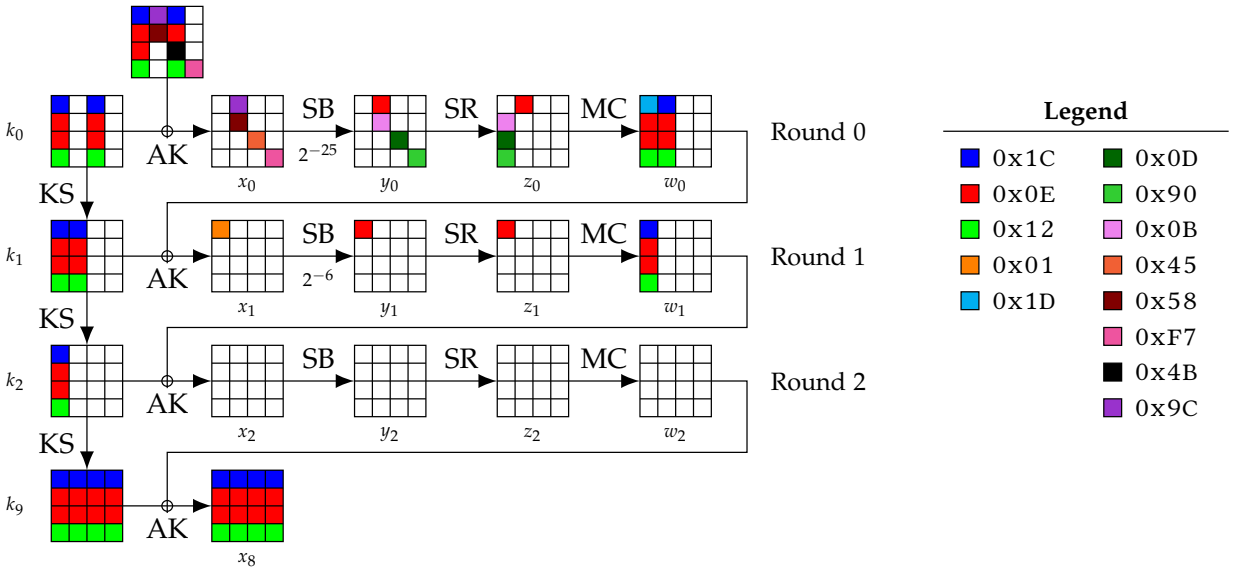


Figure 6.16: The best differential characteristic on three rounds of AES-128, which has a probability $p = 2^{-31}$. See also Table 6.4.

Table 6.4: The best differential characteristic on three rounds of AES-128, which has a probability $p = 2^{-31}$. The two lines for state differences are respectively the input difference after key addition and the output difference. See also Figure 6.16.

Round	State differences	Key differences
Plaintext	1C0E0E12 B3580000 1C0E4B12 000000F7	
0	00000000 B3580000 00004500 000000F7 1D0E0E12 1C0E0E12 00000000 00000000	1C0E0E12 00000000 1C0E0E12 00000000
1	01000000 00000000 00000000 00000000 1C0E0E12 00000000 00000000 00000000	1C0E0E12 1C0E0E12 00000000 00000000
2	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	1C0E0E12 00000000 00000000 00000000
Ciphertext	1C0E0E12 1C0E0E12 1C0E0E12 1C0E0E12	1C0E0E12 1C0E0E12 1C0E0E12 1C0E0E12

Table 6.5: Example of a pair of messages (m, m') that conforms to the 3-round truncated differential characteristic for AES-128 of Figure 6.16 with a *second* set of differences that reaches the same differential probability. The lines in this array contains the values of two subkeys and internal states before entering the corresponding round, as well as their differences. Note that discarding the first round provide a test vector for the differential characteristics of Figure 6.15.

Round	k	k'	$k \oplus k'$
0	95220EA1 3C000000 F5416D13 3E000000	AD3E1285 3C000000 CD5D7137 3E000000	381C1C24 00000000 381C1C24 00000000
1	F7416D13 CB416D13 3E000000 00000000	CF5D7137 F35D7137 3E000000 00000000	381C1C24 381C1C24 00000000 00000000
2	96220E70 5D636363 63636363 63636363	AE3E1254 5D636363 63636363 63636363	381C1C24 00000000 00000000 00000000
3	69D9F58B 34BA96E8 57D9F58B 34BA96E8	51C5E9AF 0CA68ACC 6FC5E9AF 0CA68ACC	381C1C24 381C1C24 381C1C24 381C1C24
Round	m	m'	$m \oplus m'$
Init.	5970F4AD 572C52B7 F3C5C241 6CB59500	616CE889 3C0052B7 CBD97165 6CB5953F	381C1C24 6B2C0000 381CB324 0000003F
0	CC52FA0C 6B2C52B7 0684AF52 52B59500	CC52FA0C 000052B7 06840052 52B5953F	00000000 6B2C0000 0000AF00 0000003F
1	E8000000 00000000 00000000 00000000	EA000000 00000000 00000000 00000000	02000000 00000000 00000000 00000000
2	1EB99500 3E000000 00000000 00000000	1EB99500 3E000000 00000000 00000000	00000000 00000000 00000000 00000000
End	28AB87DB EE0824E3 7D6104A1 08B3C0BE	10B79BFF D61438C7 457D1885 30AFDC9A	381C1C24 381C1C24 381C1C24 381C1C24

Best characteristics on 4 rounds

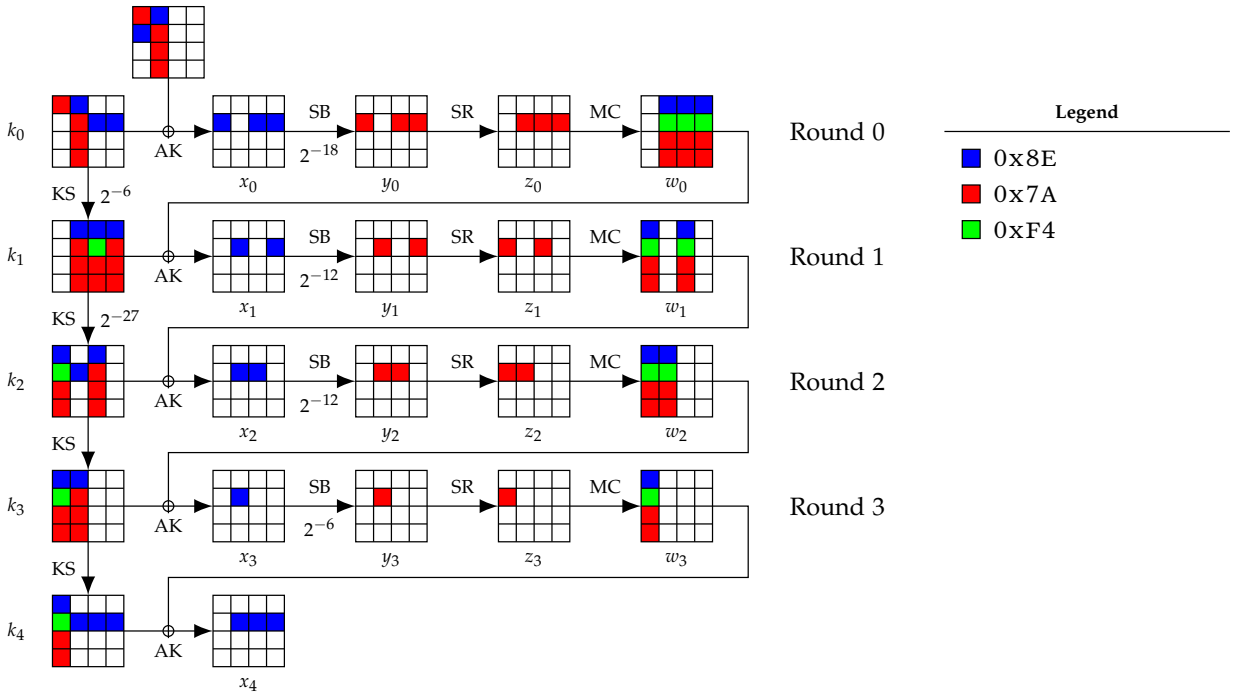


Figure 6.17: The first best differential characteristic on four rounds of AES-128, which has a probability $p = 2^{-81}$. These four rounds are the four **first** ones of the 5-round best differential characteristic (see Figure 6.19 and Table 6.6).

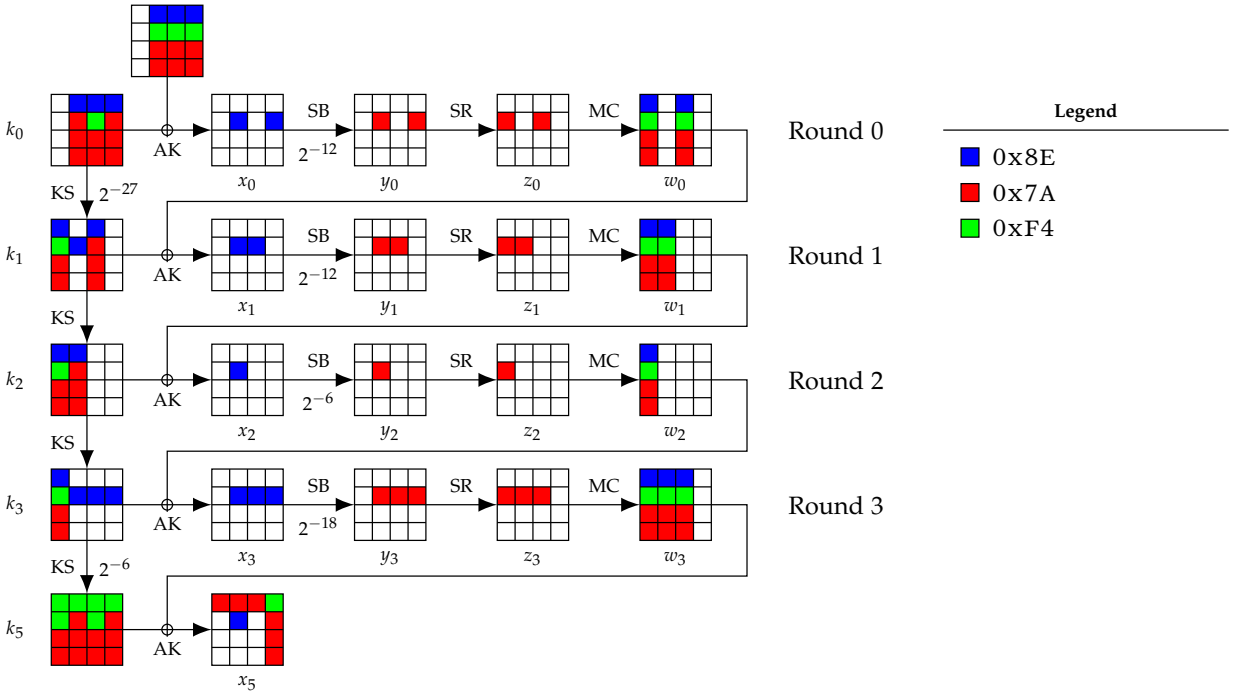


Figure 6.18: The second best differential characteristic on four rounds of AES-128, which has a probability $p = 2^{-81}$. These four rounds are the four **last** ones of the 5-round best differential characteristic (see Figure 6.19 and Table 6.6).

Best characteristic on 5 rounds

Table 6.6: The best differential characteristic on five rounds of AES-128, which has a probability $p = 2^{-105}$. The two lines for state differences are respectively the input difference after key addition and the output difference. See also Figure 6.19.

Round	State differences	Key differences
Plaintext	7A8E0000 8E7A7A7A 00000000 00000000	
0	008E0000 00000000 008E0000 008E0000 00000000 8EF47A7A 8EF47A7A 8EF47A7A	7A000000 8E7A7A7A 008E0000 008E0000
1	00000000 008E0000 00000000 008E0000 8EF47A7A 00000000 8EF47A7A 00000000	00000000 8E7A7A7A 8EF47A7A 8E7A7A7A
2	00000000 008E0000 008E0000 00000000 8EF47A7A 8EF47A7A 00000000 00000000	8EF47A7A 008E0000 8E7A7A7A 00000000
3	00000000 008E0000 00000000 00000000 8EF47A7A 00000000 00000000 00000000	8EF47A7A 8E7A7A7A 00000000 00000000
4	00000000 008E0000 008E0000 008E0000 8EF47A7A 8EF47A7A 8EF47A7A 00000000	8EF47A7A 008E0000 008E0000 008E0000
Ciphertext	7A000000 7A8E0000 7A000000 F47A7A7A	F4F47A7A F47A7A7A F4F47A7A F47A7A7A

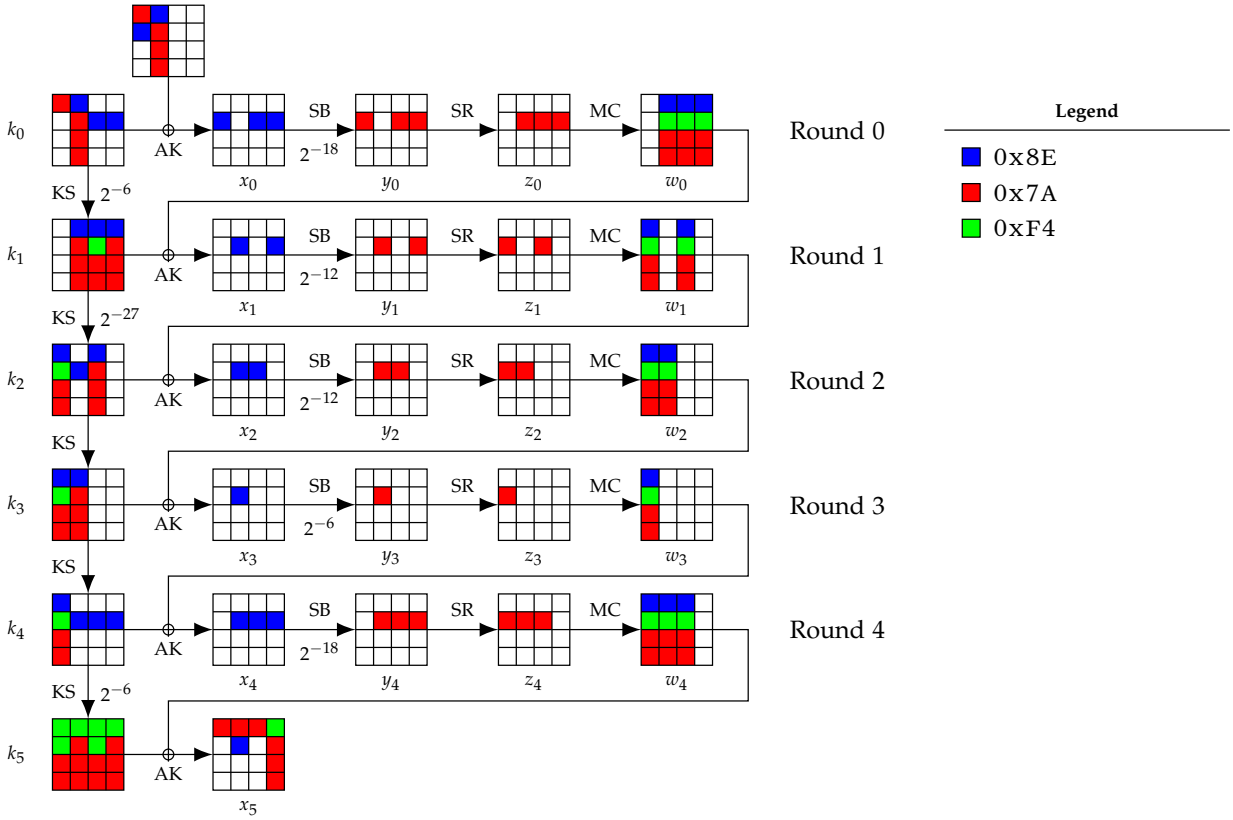


Figure 6.19: The best differential characteristic on five rounds of AES-128, which has a probability $p = 2^{-105}$. See also Table 6.6.

Table 6.7: Example of a pair of messages (m, m') that conforms to the 5-round truncated differential characteristic for AES-128 of Figure 6.19. The lines in this array contains the values of the two subkeys and internal states before entering the corresponding round, as well as their differences. Note that discarding the first or the last round provide a test vector for the differential characteristics of Figure 6.17 and Figure 6.18.

Round	k	k'	$k \oplus k'$
0	6D387102 D0C52A0F 854283FB 208E76EE	17387102 5EBF5075 85CC83FB 200076EE	7A000000 8E7A7A7A 008E0000 008E0000
1	750059B5 A5C573BA 2087F041 000986AF	750059B5 2BBF09C0 AE738A3B 8E73FCD5	00000000 8E7A7A7A 8EF47A7A 8E7A7A7A
2	764420D6 D381536C F306A32D F30F2582	F8B05AAC D30F536C 7D7CD957 F30F2582	8EF47A7A 008E0000 8E7A7A7A 00000000
3	047B33DB D7FA60B7 24FCC39A D7F3E618	8A8F49A1 59801ACD 24FCC39A D7F3E618	8EF47A7A 8E7A7A7A 00000000 00000000
4	01F59ED5 D60FFE62 F2F33DF8 2500DBE0	8F01E4AF D681FE62 F27D3DF8 258EDBE0	8EF47A7A 008E0000 008E0000 008E0000
5	724C7FEA A4438188 56B0BC70 73B06790	86B80590 5039FBF2 A244C60A 87CA1DEA	F4F47A7A F47A7A7A F4F47A7A F47A7A7A

Round	m	m'	$m \oplus m'$
Init.	65380101 FDA4FF6F D0424BEF 7A8E35D8	1FB60101 73DE8515 D0424BEF 7A8E35D8	7A8E0000 8E7A7A7A 00000000 00000000
0	08007003 2D61D560 5500C814 5A004336	088E7003 2D61D560 558EC814 5A8E4336	008E0000 00000000 008E0000 008E0000
1	D2D342E8 CA8E7146 E79EA6D7 3B8E48F9	D2D342E8 CA007146 E79EA6D7 3B0048F9	00000000 008E0000 00000000 008E0000
2	91367406 EF8E3E84 9D8E980B 2BD1EE66	91367406 EF003E84 9D00980B 2BD1EE66	00000000 008E0000 008E0000 00000000
3	4331727A 1E004722 172C7D6A B8EE10F1	4331727A 1E8E4722 172C7D6A B8EE10F1	00000000 008E0000 00000000 00000000
4	CE92FA3E B10007E0 A200CBA6 0D002D37	CE92FA3E B18E07E0 A28ECBA6 0D8E2D37	00000000 008E0000 008E0000 008E0000
End	5FBA1C3F E08C4C0F 4BDA87A9 F6890230	25BA1C3F 9A024C0F 31DA87A9 02F3784A	7A000000 7A8E0000 7A000000 F47A7A7A

CHAPTER 7

AES in the Open-Key Model

Contents

7.1 Generalities **143**
7.1.1 Motivations 143
7.1.2 Rebound technique 145
7.1.3 Limited-birthday distinguisher 146
7.2 Known-key model **149**
7.2.1 Distinguishers for 7 rounds 149
7.2.2 Distinguisher for 8 rounds 155
7.3 Chosen-key model **161**
7.3.1 Distinguisher for 7-round AES 162
7.3.2 Distinguisher for 8-round AES 168
7.3.3 Distinguisher for 9-round AES-128 173

In this chapter, we discuss the security of AES in the recent open-key model. This framework has been introduced by Knudsen and Rijmen at ASIACRYPT 2007 in [KR07] and considers the block cipher where the key acts as an additional input parameter, which might be constant or not. Consequently, the goal of the adversary is not to recover the bits of the key but rather to disclose a nontrivial property of the structure of the family of block ciphers.

In the following sections, we analyze an AES-like permutation as defined in Section 4.3, and we suppose the key to be either known (Section 7.2) or can be chosen (Section 7.3). The results on AES variants reduced to 7 and 8 rounds in the chosen-key setting have been described in the paper [DFJ12a] written with Patrick Derbez and Pierre-Alain Fouque and have been published at INDOCRYPT 2012. The 9-round distinguisher for AES-128 has been published at CRYPTO 2013 in [FJP13a] in a paper co-authored with Pierre-Alain Fouque and Thomas Peyrin.

7.1 Generalities

7.1.1 Motivations

In this chapter, we describe several published results that distinguish AES-like permutations in the open key models where the adversary either knows the key used in the encryption process

or is given access to the complete family of permutations implemented by the considered block cipher; that is, he can choose the key.

The motivation behind this model is twofold. First, we intuitively give more confidence in a block cipher that has been analyzed in both the standard model and in the open key model. If we can somehow show that the block cipher has no particular weakness when an adversary knows the key or can select some particular keys to pinpoint a certain class of weak keys, then we have more confidence in it when it is used in a more classical way.

The term *weakness* happens to be hard to define formally as there is no secret involved in the process. In the more usual standard model, a key is uniformly and secretly drawn to be used by the block cipher, and the adversary aims at recovering it or at least tries to build a distinguishing algorithm. In the known-key model, there is no secret, and the goal of the adversary is to prove that a structural property can be verified or computed for the block cipher when the same property for an ideal permutation could only be achieved after a significant computational effort.

We can for instance examine the block cipher DES when the key is not specified: the adversary considers the block cipher family seen as a set of 64-bit permutations indexed by a 56-bit key parameter. A well-known structural property existing in the DES is the complementation property that a random 64-bit permutation does not have with overwhelming probability. Note that this property also yields to a related-key distinguisher. Namely, for any message m and any key k, the knowledge of the ciphertext $\mathrm{DES}_k(m)$ allows to compute the ciphertext corresponding to the plaintext $\overline{m}$ under key $\overline{k}$, where $\overline{x}$ is the bitwise complementary of x:

$$\forall (k, m), \quad \mathrm{DES}_{\overline{k}}(\overline{m}) = \overline{\mathrm{DES}_k(m)}.$$

Consider now the following problem: find a pair of keys (k, k') and a pair of inputs (m, m') such that $k' = \overline{k}$, $m' = \overline{m}$ and $\mathrm{DES}_{k'}(m') = \overline{\mathrm{DES}_k(m)}$. For a random 64-bit permutation, this property would be verified with probability 2^{-64} as a valid pair of keys $(k, \overline{k})$ and pair of inputs $(m, \overline{m})$ would result in complementary outputs with probability 2^{-64}. Consequently, the cost the adversary needs to pay to find such inputs is approximately 2^{64} encryption queries to the oracle implementing the family of block ciphers. In the case of DES, the structural property allows to find it in constant time, and for any choice of (k, m). Indeed, for a (k, m), we get $c = \mathrm{DES}_k(m)$, and we know with probability 1 that $\mathrm{DES}_{\overline{k}}(\overline{m})$ produces $\overline{c}$.

More formally, this kind of property is called an *evasive property*: it can be checked easily but cannot be satisfied with oracle accesses $\mathcal{O}$ to a permutation and its inverse with a non-negligible probability. Formally, Canetti, Goldreich and Halevi define a property as evasive in [CGH98, CGH04] by a binary relation $\mathcal{R}$ when it is computationally hard to find a value x such that the pair $(x, \mathcal{O}(x))$ is in $\mathcal{R}$.

In the literature, we find examples of evasive properties that answer to particular problems. In the seminal work by Knudsen and Rijmen [KR07], the problem is to find a k-sum of plaintext/ciphertext pairs $(x, \pi(x))$ for the permutation π of the known-key AES (Section 7.2.1.1). In [GP10], Gilbert and Peyrin consider the problem of finding two inputs to a permutation π that collide on some prescribed bits such that their images by π also collide on some other bits (Section 7.1.3). Another example can be found in the work by Biryukov, Khovratovich and

Nikolić in [BKN09] that finds differential q-multicollisions for a given block cipher, and then apply this to AES-256.

A second motivation for the open-key model is related to the hash function domain. Indeed, the open-key framework fills the gap between the standard model for block ciphers and the security notions for hash functions where no key is involved. The relations between block-ciphers and hash functions have always been strong since there are many known and provably secure ways to turn a block cipher into a compression function, like the PGV hash functions [PGV94, BRS02, BRSS10]. The intuition is that the adversary controls both the key input and the message input of the block cipher when we consider a hash function or a compression function. Therefore, it is important to carefully study the block cipher designs in the open-key model to prevent an oblivious use in a compression function.

7.1.2 Rebound technique

We have already recalled the rebound technique in Section 3.5.2, but we present here how it has been applied to AES-like permutations in the literature. We describe all the published results with the unified view of Figure 7.1 on an AES-like permutation. The rebound technique splits the algorithm in two parts: the inbound phase and the probabilistic outbound phase. For any size t of the internal state, the inbound phase requires to enumerate the solutions of the middle rounds efficiently to test them against the outbound phase. We note that whenever we fix the input and output differences of the inbound phase, there is exactly one solution if the middle rounds are fully active, which is the case for most of the published results that we detail in the following. We ideally want to find an algorithm that runs in constant amortized cost to only pay the workfactor of the outbound phase.

One of the major general improvements to decrease the time complexity of the procedure consists in including a probabilistic event of the outbound phase into the inbound to control it. This considerably reduces the number of solutions to generate for the middle round since the probability of the outbound is significantly increased, but in return the algorithm for the middle rounds is more complex.

In the following Table 7.1, we summarize the main results published for AES-like permutations in the open-key model.

Characteristic		Known-key			Chosen-key		
Inbound	**Total**	**Ref.**	**Section**	**Cond.**	**Ref.**	**Section**	**Cond.**
1R	7R	[MRST09]	7.2.1.2	$t \geq 4$	[DFJ12a]	7.3.1	AES
		[MPRS09]	7.2.1.3	$t \geq 4$			
2R	8R	[GP10]	7.2.2.1	$t \geq 4$	[DFJ12a]	7.3.2	AES
		[SLW+10]	7.2.2.2	$t \geq 5$			
3R	9R	[JNPP12]	8.2	$t \geq 8$		—	

Table 7.1: Advances done in inbound phases of rebound-based algorithms for AES-like permutations. The variable t represents the size of the square state of the permutation. We give the conditions required for the algorithms to work. The chosen-key setting in particular requires a key schedule algorithm in addition to the AES-like permutation.

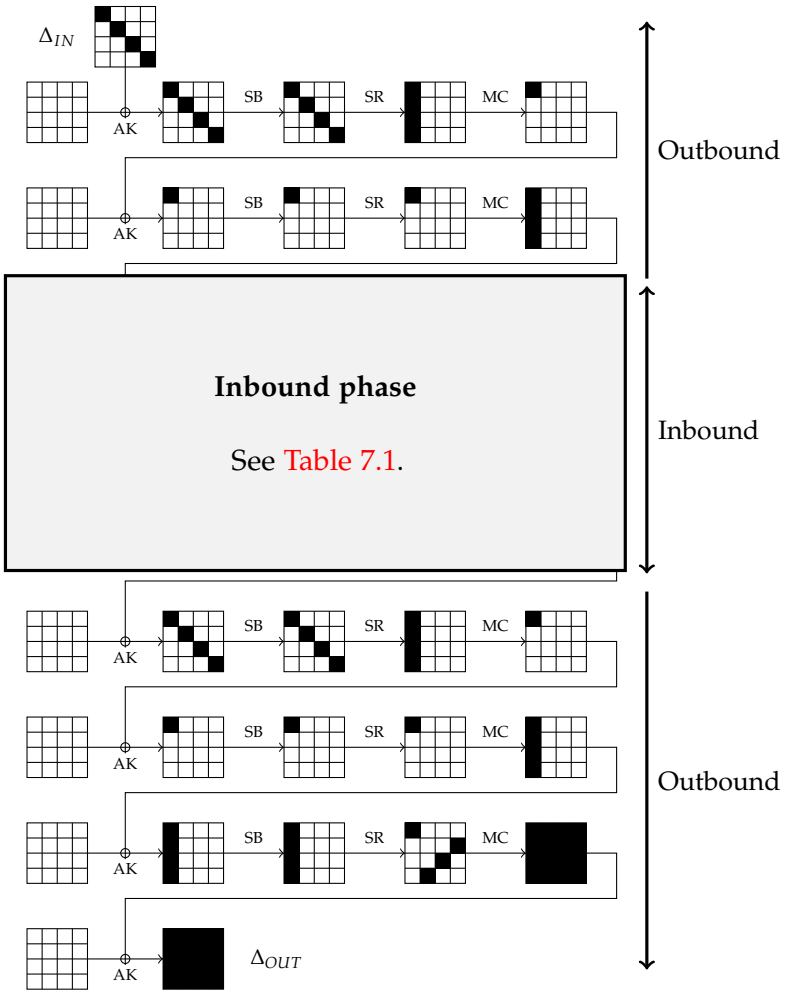


Figure 7.1: General framework to apply the rebound technique to an AES-like permutation (here, $t = 4$).

7.1.3 Limited-birthday distinguisher

In this section, we precise one of the main type of distinguishers we are using. It has been called limited birthday distinguishers and has been introduced by Henri Gilbert and Thomas Peyrin in [GP10]. We are interested in the kind of distinguishers where the attacker is challenged to find a pair of inputs whose difference is constrained in a predefined input subspace, such that the ciphertext difference lies in another predefined subspace (Problem 7.1).

Problem 7.1 ([GP10]). *Given a permutation π and two subspaces E_{in} and E_{out}, find a pair of inputs (x, x') such that $x \oplus x' \in E_{in}$ and $\pi(x) \oplus \pi(x') \in E_{out}$.*

We note that the two subspaces E_{in} and E_{out} actually define truncated differences, as they represent a set of differences. Consequently, Problem 7.1 can be reformulated as follows: find a pair of inputs (x, x') such that the input difference $x \oplus x'$ belongs to the truncated difference E_{in}, and that the output difference $\pi(x) \oplus \pi(x')$ belongs to the truncated difference E_{out}.

In the sequel, we denote $n_i = \dim(E_{in})$ and $n_o = \dim(E_{out})$ (see Figure 7.2). This problem generalizes the concept of collision in a function where we want the output space to be empty ($n_o = 0$) and we do not constrain the input at all. By the birthday effect on an n-bit function, we know we only need to consider about $2^{n/2}$ random elements before two of them share the same image by π. This strategy thus finds a pair of inputs in $2^{n/2}$ computations in the case $n_o = 0$.

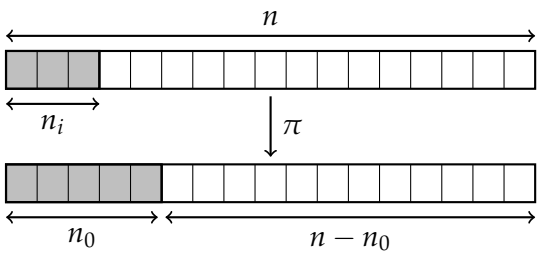


Figure 7.2: Generalized collision problem for the limited birthday problem. Assuming $n_i \leq n_o$, the attacker searches for a pair of inputs to the random permutation π differing in n_i known cell positions such that the output differs in n_o known cell positions. A gray cell indicates a byte with a non-zero difference.

More importantly, this property assumes that it is possible to find $2^{n/2}$ different inputs. In the generalized case of Problem 7.1, we restrict the number of possible input pairs. In particular, for $n_i = n$, we can construct as many as $\binom{2^n}{2} \approx 2^{2n-1}$ pairs, whereas for any n_i this is reduced to

$$2^{n-n_i}\binom{2^{n_i}}{2} \approx 2^{n-n_i} \cdot 2^{2n_i-1} \approx 2^{n+n_i-1}.$$

We obtain this by considering all $\binom{2^{n_i}}{2}$ pairs in the input subspace E_{in} and then restart with different bits at the $n - n_i$ inactive positions. Consequently, as $2^{n+n_i-1} \ll 2^{n/2}$, we cannot hope to find a collision on $n_o = n$ output bits with the input data. That is why Gilbert and Peyrin have dubbed the algorithm *limited* birthday, for the restricted amount of freedom at the input of permutation. To overcome this difficulty, we create more freedom by considering new values of the constant bits, but this results in independent birthday structures which cannot be combined to take advantage of the birthday effect.

In the case of an AES-like permutation, we can actually pack c bits together due to the byte-oriented structure of the primitive. Consequently, the cells depicted on Figure 7.2 actually represent c bits, and can take as many as 2^c different values. Therefore, we have $n = ct^2$, and both n_i and n_o range in the interval between 0 and t^2. To state the time complexity of the algorithm in the limited case, we need to consider different ranges for the input parameters n_i and n_o to determine whether the original birthday is more efficient. Without loss of generality, we assume in the following that $n_i \leq n_o$: the attacker thus considers π rather than its inverse π^{-1}, as it is easier to collide on $t^2 - n_o$ cells than on $t^2 - n_i$.

In the event that $t^2 - n_o < 2n_i$, the original birthday effect applies easily as we have enough freedom at the input to reach a collision on $t^2 - n_o$ cells. We actually need to compute the image by π of $2^{c\cdot(t^2-n_o)/2} < 2^{c\cdot n_i}$ input elements randomly drawn in E_{in}, which is feasible in time $2^{c\cdot(t^2-n_o)/2}$.

On the other hand, if $t^2 - n_o \geq 2n_i$, then $t^2 - n_o - 2n_i$ of the $t^2 - n_o$ cells do not have a zero-difference at the output. Hence, we need to restart the birthday paradox process about $2^{c\cdot(t^2-n_o-2n_i)}$ times, which costs $2^{c\cdot(t^2-n_o-2n_i)} \cdot 2^{c\cdot n_i} = 2^{c\cdot(t^2-n_o-n_i)}$ in total.

We summarize the time complexity in the following Theorem 7.1.

Theorem 7.1. *Let $n_i = \dim(E_{in})$ and $n_o = \dim(E_{out})$. There exists an algorithm called* limited-birthday *algorithm that solves Problem 7.1 for an* AES*-like permutation in time $LB(n_i, n_o)$ where:*

$$\log_{2^c}\Big(LB(n_i,n_o)\Big) = \begin{cases} (t^2-n_o)/2 & \text{if } t^2 < 2n_i + n_o, \\ n_i & \text{if } t^2 = 2n_i + n_o, \\ t^2 - n_i - n_o & \text{if } t^2 > 2n_i + n_o. \end{cases}$$

As an example, we give the following chart (Figure 7.3) which lists all the complexities $LB(n_i, n_o)$ for all n_i and n_o in $[0, t^2]$ when we fix the parameters $t = 4$ and $c = 8$ like the AES.

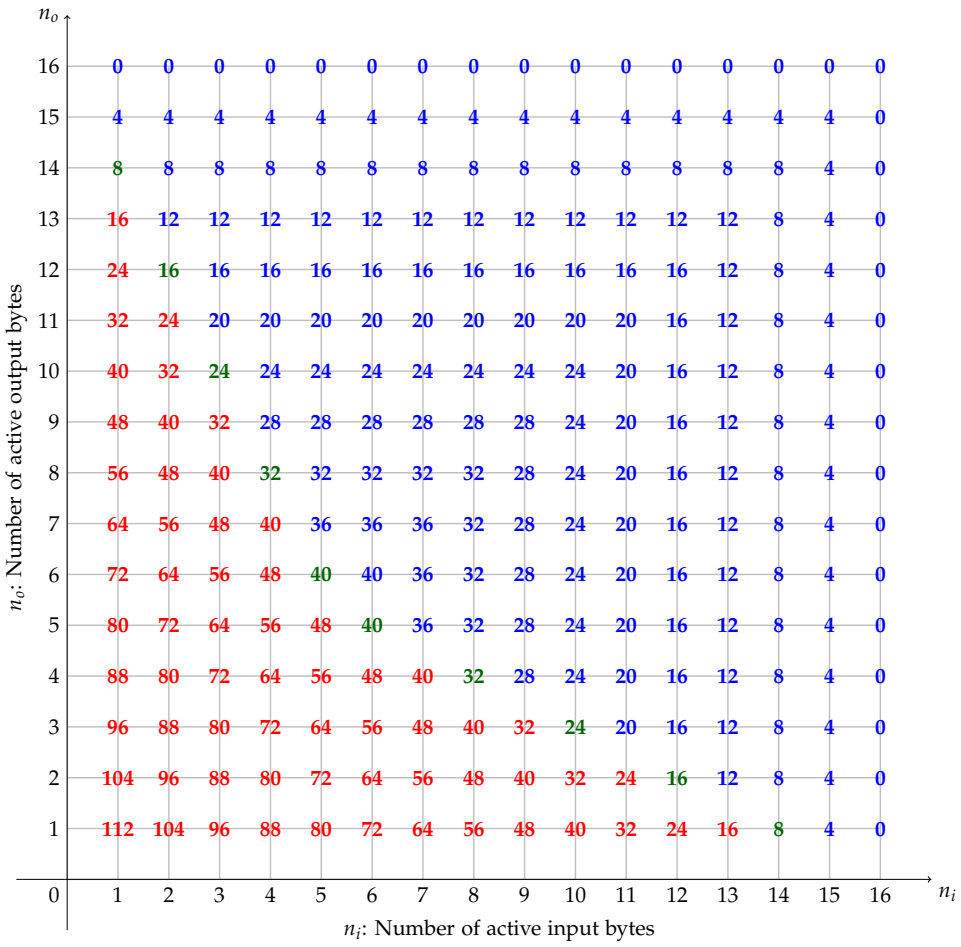


Figure 7.3: Chart of the $\log_2$ time complexities for the limited-birthday algorithm in the case of an AES-like permutation with $t = 4$ and $c = 8$. Numbers in blue correspond to the original birthday algorithm while red ones refer to the limited case. Green ones are the border cases.

7.2 Known-key model

7.2.1 Distinguishers for 7 rounds

We present here the three main results published presenting distinguishers for 7 rounds of an AES-like permutation in the known-key model. By chronological order, the first one (Section 7.2.1.1) is the original algorithm from Knudsen and Rijmen in [KR07] where they present the open-key model. As an application, they show that we can use the integral property of 3 and 4 rounds AES to distinguish it from a random permutation.

Then, the second distinguisher (Section 7.2.1.2) by Mendel, Rechberger Schläffer and Thomsen in [MRST09] uses the rebound strategy to find a pair of inputs to the AES-like permutation more efficiently than for a random permutation. Finally, the last algorithm (Section 7.2.1.3) by Mendel, Peyrin, Rechberger and Schläffer in [MPRS09] improves the previous result by increasing significantly the probability of the outbound phase.

7.2.1.1 Integral distinguisher

In [KR07], Knudsen and Rijmen propose an integral distinguisher based on their earlier work with the square attack [DKR97] (see Section 4.4.1). In the known-key model, we assume that the key is a parameter known to the adversary so that he can start the algorithm by choosing values right in the middle on the AES computation, and not necessarily at one end, in the plaintext or the ciphertext.

The integral distinguisher propagates a δ-set from the middle to both ends of the AES-like permutation (see Figure 7.4) to produce a set $\mathcal{P}$ of plaintexts that verify the balance property such that their corresponding ciphertexts $\mathcal{C}$ also verify the same property.

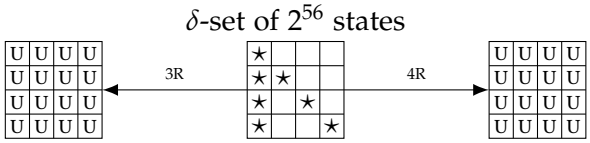
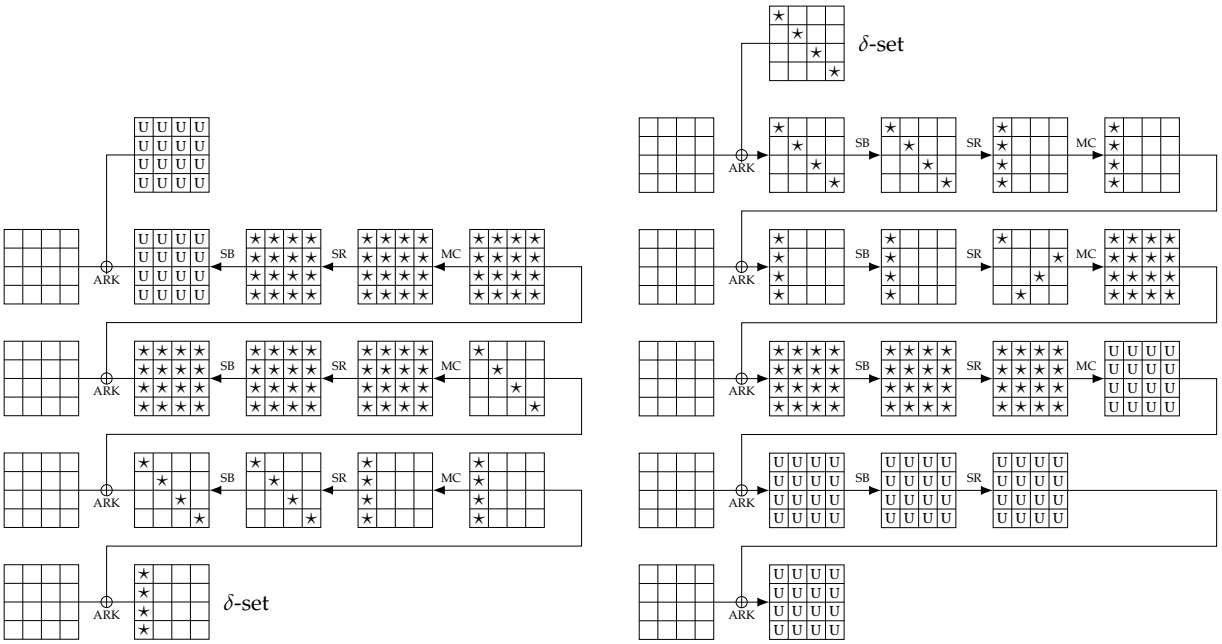


Figure 7.4: The full 7-round integral distinguisher for an AES-like permutation (example with $t = 4$). A byte marked by $\star$ assumes all 2^8 values, a white byte is passive and a byte marked by U is balanced.

The forward part of the distinguisher consists of a 4-round integral distinguisher where we omit the last **MixColumns** application (see Figure 7.5b). When going backwards, the balance property is still verified because all the operations of the AES round function are bijective.

To get a δ-set that propagates from the middle, we consider the structure of $2^{c\cdot(2t-1)}$ states of Figure 7.4 in the middle of the computation: forwards, the integral distinguisher of Figure 7.5b applies, and the one from Figure 7.5a for the backward direction. The $t-1$ additional active cells for both sides do not change the balance property because it boils down to the same summation several time and produces zero since $0 \oplus 0 = 0$.

As a result the time complexity of this algorithm is equivalent to $2^{c\cdot(2t-1)}$ encryptions with small memory requirements.



(a) Backward integral distinguisher (3 rounds). **(b)** Forward integral distinguisher (4 rounds).

Figure 7.5: Integral distinguisher for 7 rounds of an AES-like permutation ($t = 4$): a byte marked by $\star$ is active, a white byte is passive and a byte marked by U is balanced. On the left, (a) shows the backward part, while the forward part is on (b).

Generic complexity

Showing that the above algorithm actually results in a distinguisher also includes an analysis of the generic scenario. We need to show that this algorithm is faster for AES than for a random permutation.

Namely, we are given a random permutation f on ct^2 bits, and our goal is to find a set $\mathcal{P}$ of inputs $\{x_i\}$ such that $\sum_{i=1}^{k} = f(x_i) = 0$. This problem is related to the k-sum problem or generalized birthday problem. It has been studied by David Wagner in [Wag02] where he proposes an algorithm to solve it that works in time

$$\mathcal{O}\left(k \cdot 2^{\frac{ct^2}{1+\log_2(k)}}\right).$$

In our case, we have $ct^2 = 128$ and $k = 2^{c\cdot(2t-1)} = 2^{56}$ which gives a time complexity of approximately $2^{58.2}$ encryptions. Obviously, this is a rough estimate as the big-O notation masks small constants and memory accesses can be costly, but anyway the integral algorithm distinguishes the AES-like permutation from a random permutation. We also note that the XHASH attack due to Bellare and Micciancio in [BM97] might provide an improvement over the generalized birthday algorithm as it has been noted in [AKK+10, BDPA10].

7.2.1.2 Rebound attack

In this section, we give a more efficient known-key distinguisher for 7 rounds of an AES-like permutation. The algorithm uses the rebound technique [MRST09] that we have already presented in Section 3.5.2 originally introduced for hash functions. Indeed, the reason why we can apply it to hash functions is the lack of secret key, but the known-key model effect is the same.

The application of the technique is basic: we generate many solutions for the middle rounds 2 and 3 at a small cost (inbound phase) and exhaust them in the probabilistic filters of the outer rounds (outbound phase): **MixColumns** $4 \rightarrow 1$ transitions in rounds 1 and 5 (see Figure 7.6).

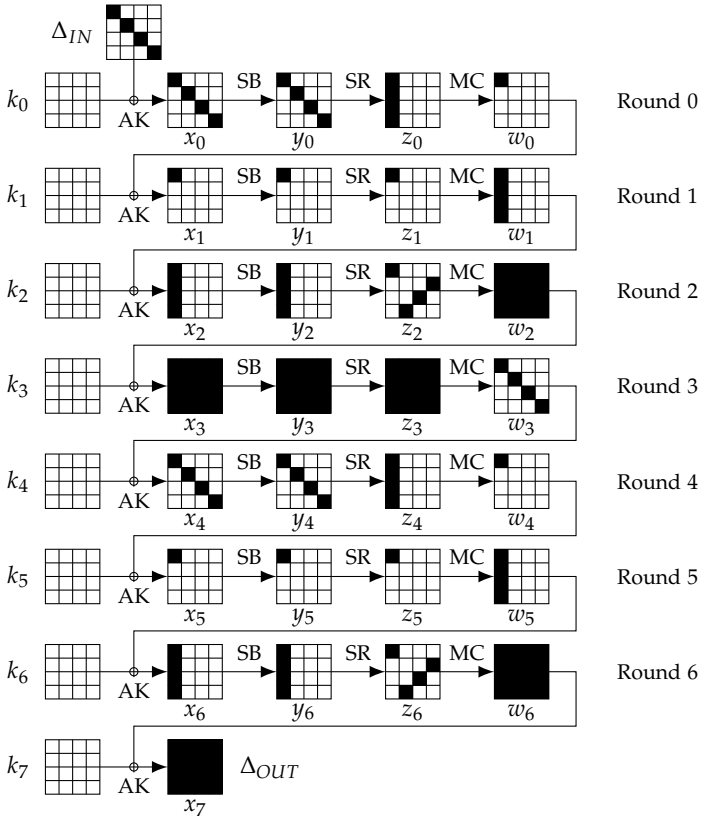


Figure 7.6: Truncated differential characteristic with $t = 4$ uses in the rebound distinguisher for 7 rounds of the known-key AES-like permutation. A black cell has non-zero difference while white cells are inactive.

The strategy starts by randomizing the difference Δy_2 in state y_2 and the difference Δw_3 in state w_3. From y_2 to x_3, the operations are linear (**ShiftRows**, **MixColumns** and **AddRoundKey**) so that we can propagate the randomized difference at the input of the S-Box of round 3. Then, starting from w_3 and going backwards we can do the same and deduce the values of the differences in Δy_3.

Consequently, the input and output differences of the **SubBytes** layer of round 3 are completely determined. From the differential property of AES S-Box S that we recalled in Theorem 4.2, we know that the t^2 parallel equations $\mathrm{S}(x) \oplus \mathrm{S}(x \oplus \Delta_i) = \Delta_o$ for a Δ_i at the input in Δx_3 and a Δ_o at the output in Δy_3 all have one solution with probability $1/2$. More

specifically, this equation at the state level $S(x_3) \oplus S(x_3 \oplus \Delta x_3) = \Delta y_3$ has 2^{t^2} solutions x_3 with probability 2^{-t^2}, for fixed Δx_3 and Δy_3. This means that we have to restart 2^{t^2} times the randomization of the differences in Δy_2 and Δw_3 before getting 2^{t^2} solutions.

Therefore, for all the $(2^c - 1)^{2t} \approx 2^{2ct}$ guessed differences in Δy_2 and Δw_3, we expect to have 2^{2ct} paired values that conform to the truncated differential characteristic of Figure 7.6 reduced to rounds 2 to 3. While solving this inbound phase, we get pairs of states that match the middle rounds, and we can directly test them against the outer rounds to check whether the truncated differential transitions $t \to 1$ hold in the two **MixColumns**.

For a single pair, the probability of each of the two transitions equals $2^{-c\cdot(t-1)}$ as we require only the top cell to remain active at the output of the transition. This event is a $c \cdot (t-1)$-bit filter which makes the total probability of the outbound phase $2^{-2c\cdot(t-1)}$. By exhausting only $2^{2c\cdot(t-1)}$ of the 2^{2ct} possible pairs, we then expect to get one that conforms to the whole 7-round characteristic. This algorithms cost $2^{2c\cdot(t-1)}$ computations per final pair, and we can get as much as $2^{2ct-2c\cdot(t-1)} = 2^{2c}$.

We note that this algorithm implicitly assumes that $t^2 \leq 2ct$, i.e. $t \leq 2c$, as we obviously need more freedom degree in the randomization of the two initial differences to hit values that lead to the 2^{t^2} solutions around the **SubBytes** layer. We emphasize that we need 2^{t^2} computations to get the 2^{t^2} solutions. In the case of the AES, the condition is verified since $c = 8$ and $t = 4$, and the algorithm has a time complexity of 2^{48} simple operations. This algorithm requires to store about 2^{2c} elements in memory to represent the DDT of the S-Box, which amounts to 2^{16} in the case of the AES.

Generic complexity

In comparison, finding two inputs that verify the same criterion for a randomly drawn ct^2-bit permutation π requires approximately

$$LB(t,t) = 2^{ct^2-2ct} = 2^{ct(t-2)}$$

computations by the limited-birthday algorithm (Theorem 7.1). Indeed, the same property consists in finding a pair of inputs to π such that the truncated input difference equals Δ_{IN} and the truncated output difference equals Δ_{OUT} (Figure 7.7), where there are $c \cdot t$ active bits at both the input and the output.

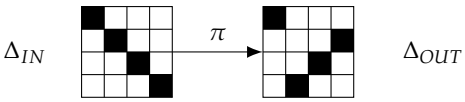


Figure 7.7: Differential properties required on the wanted pair for the generic scenario for a random permutation π.

We note that while the difference in the outputs of Figure 7.6 differs in all the t^2 cells, the actual differences are not *all* independent. The last **MixColumns** application being linear, we can invert it to recover the 4 independent variables from Δz_6 that actually makes a 4-dimensional output subspace of differences. This is the reason why we consider an output dimension of $c \cdot t$ rather than $c \cdot t^2$.

Consequently, the algorithm based on the rebound strategy is faster than the generic solution to the problem if the following inequality holds:

$$2(t-1) \leq t(t-2),$$

which is independent of c and true as soon as $t \geq 2+\sqrt{2}$, that is $t \geq 4$.

7.2.1.3 Improved distinguisher: start-from-the-middle technique

The way to handle the pairs in the previous algorithm is a direct application of the rebound technique, but in that case, we can adapt it to get a significant improvement in the time complexity. This trick has been published in [MPRS09] and basically consists in absorbing the first of the two probabilistic $t \to 1$ transitions in the inbound phase. This way, we control exactly how the solutions for the middle rounds behave through one transition and we only need to pay the second one. Concretely, this makes a distinguisher requiring about $2^{c(t-1)}$ simple computations and $2^{t^2}2^{2c}$ memory units to store the DDT of the S-Box and the lists needed in the inbound phase. We detail how in the following.

The algorithm uses the same truncated differential characteristic (Figure 7.6) and also starts by randomizing the differences $\Delta x_4 = \Delta w_3$ in w_3. We note that we can have as many as $(2^c-1)^t \approx 2^{ct}$ starting points here. Again, the difference propagates linearly backwards up to Δy_3. At this point, we do not perform the rebound by randomizing the other side of the **SubBytes** layer, but rather we store *all* the valid input differences for Δx_3. That is, for each of the t^2 output cell differences Δ in Δy_3, we consider the list

$$\mathcal{S}_\Delta = \Big\{(x, x\oplus\delta) \mid \mathrm{S}(x)\oplus\mathrm{S}(x\oplus\delta) = \Delta\Big\},$$

which contains all the paired values that transform one input difference δ to the output difference Δ. From the assumed differential property of the S-Box, these sets contain 2^c pairs. For clarity, we denote $L_{i,j}$ the lists that contains the elements of $\mathcal{S}_{\Delta y_3[i,j]}$.

We note that while the lists contain 2^c different pairs, the 2^c differences between each pairs are not all different: this is a consequence of the symmetry in the equation $\mathrm{S}(x)\oplus\mathrm{S}(x\oplus\delta)=\Delta$, where both $(x, x\oplus\delta)$ and $(x\oplus\delta, x)$ belong to the lists, but the difference of the two are δ. Consequently, the lists inherently represent only $2^{c-1}-1$ different differences each.

Now, in state x_3 we need to propagate and filter lists of paired values to find one pair in each of the t^2 lists that conforms to the truncated pattern up to x_0. To filter the pairs in the **MixColumns** transition of round 2, we consider t lists of one column at a time. For the first column for instance, the lists are $L_{0,0}, \ldots, L_{t-1,0}$ which make as many as $(2^c)^t = 2^{ct}$ possible candidates, when we want only a fraction $1/2^{c\cdot(t-1)}$ of them since we need only the top cell to be active in that column of z_2. To find them all, we guess the difference $\Delta z_2[0,0]$ and compute its image $\mathbf{M}\times\Delta z_2[\star,0]$ by the **MixColumns** and lookup the lists for the t differences. We expect each difference to lie in the t lists with probability $(2^{c-1}-1)/(2^c-1) \approx 2^{-1}$ so that we deduce about $(2^c-1)\times 2^{-t} \approx 2^{c-t}$ paired elements for the first column of z_2 that verify the truncated pattern.

By doing so for the t columns in parallel, we end up in z_2 with lists of 2^{c-t} paired elements per column, and we can propagate them backwards up to x_2 into the t diagonals of the state, by

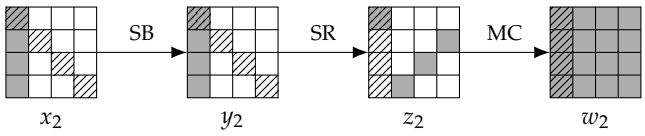


Figure 7.8: Round 2 of the characteristic of Figure 7.6 ($t = 4$). A cell in white is inactive, a cell in gray is active and hatched bytes trace the evolution of the elements of one list from w_2 to x_2.

passing the values through the S-Box (see Figure 7.8). The differences in Δx_2 get randomized, but any tuple of t paired values in the t lists at this point of the algorithm makes a valid pair of states, in the sense that the truncated differential characteristic from x_2 to z_4 is already verified.

Now, the improvement in comparison to the classical rebound strategy consists in controlling the $t \to 1$ transition of round 1 by carefully choosing the values that make the transition valid. Again, we guess the difference $\Delta y_1 = \Delta z_1$ in round 1 and compute its image by **MixColumns**:

$$\mathbf{M} \times \Big[\Delta z_1[0,0],\, 0,\, \ldots,\, 0\Big]^{\mathrm{T}} = \Big[\Delta w_1[0,0],\, \ldots \Delta w_1[t-1,0]\Big]^{\mathrm{T}}.$$

Performing exactly the same method between z_1 and w_1 as we have done to match the lists between z_2 and w_2 does not work since the lists are smaller. Indeed, they contain about 2^{c-t} elements by assuming $t \leq c$ so we find the t differences $\Delta w_1[*,0]$ in each of the t lists with probability $2^{c-t}/2^c = 2^{-t}$. Consequently, the total probability would be 2^{-t^2} for the t lists. Hopefully, we still have some freedom degree left: the critical point is the small size of the lists, but we can introduce new values in the lists that are completely independent of the present ones. Namely, in the previous step between z_2 and w_2, we match differences and build lists of size 2^t, but for each found t-tuple of paired values, we can build up to $2^t - 1$ *different* tuples that all share the same difference by swapping the elements of the pairs. This increases the size of the t lists to $2^{c-t} \cdot (2^t - 1) \approx 2^c$ elements that get randomized in the corresponding lists in the diagonals of x_2 with the effect of S^{-1}. Therefore, we now find the differences $\Delta w_1[*,0]$ in the t list with probability close to 1 with the corresponding paired t-tuple of values.

All in all, we start by guessing the differences in w_3, and by operations on lists of sizes at most 2^c, we manage to construct a pair of states that conform to the truncated differential characteristic of Figure 7.6 between x_0 and y_4. To end the algorithm and finally find a pair of states that matches the whole characteristic, we repeat this procedure about $2^{c\cdot(t-1)}$ times as in the rebound technique to pass the $t \to 1$ probabilistic transition of round 5 that holds with probability $2^{-c\cdot(t-1)}$. We thus expect to find one after $2^{c\cdot(t-1)}$ repetitions and we note that we can produce about

$$(2^c - 1)^t \times 2^{-c\cdot(t-1)} \approx 2^c$$

solutions in time $2^{c+c\cdot(t-1)} = 2^{ct}$, since we can pick $(2^c - 1)^t$ different starting points for Δw_3.

We recall from the description that this algorithm assumes implicitly that $t \leq c$, which would otherwise make the lists empty halfway in the process. In the case of the AES, this condition is also verified as $t = 4$ and $c = 8$ and produces an algorithm with a memory of about 2^{16} elements, and a running time of 2^{24} computations.

Generic complexity

In comparison, finding two inputs that verify the same criterion for a randomly drawn ct^2-bit permutation π requires approximately

$$LB(t,t) = 2^{ct^2-2ct} = 2^{ct(t-2)}$$

computations by the limited-birthday algorithm (Theorem 7.1). The justification on the output difference from the previous algorithm still holds.

Consequently, the algorithm based on the rebound strategy is faster than the generic solution to the problem if $t \leq c$ and if the following inequality holds:

$$t - 1 \leq t(t-2),$$

which is independent of c and true as soon as $t \geq \frac{3+\sqrt{5}}{2}$, that is $t \geq 3$.

7.2.2 Distinguisher for 8 rounds

7.2.2.1 Fully-active characteristic

At FSE 2010 in [GP10], Henri Gilbert and Thomas Peyrin introduce a way to extend the known-key result on 7 rounds of an AES-like permutation to 8 rounds. Their algorithm is also based on the rebound technique but they manage to cover one more round in the inbound phase. The outbound phase remains the same as the original result [MRST09] by Mendel et al., which has a probability $2^{-2c(t-1)}$ of two $t \to 1$ transitions, but the inbound phase covers x_2 in round 2 to w_4 in round 4. The truncated differential characteristic extended by one round from the middle (round 3) is depicted on Figure 7.9.

Except the limited-birthday distinguisher, the main contribution of their paper revisits the 2-round non-linear construction called **Super-SBox**, which has already been discussed several times in the design of the AES by its authors [DR01, DR02, DR06b]. The novelty is to see it as building to perform cryptanalysis rather that proving bounds on the differential probabilities of 2-round differentials for uniformly drawn keys. We now recall their main results in our generalized framework.

Generic Super-SBox construction

As shown on Figure 7.10, we can rewrite the successive transformations of two rounds of an AES-like permutation and permute the order of the two first **SubBytes** and **ShiftRows** operations. This introduces the non-linear **Super-SBox** which applies on t columns of ct bits of the state in parallel.

Therefore, the **Super-SBox** is a keyed operation that acts as a non-linear permutation of $c \cdot t$ bits. It is defined as the composition of the four operations:

$$\textbf{Super-SBox}_k \stackrel{\text{def}}{=} \textbf{SubBytes} \circ \textbf{AddRoundKey}(\mathbf{k}) \circ \textbf{MixColumns} \circ \textbf{SubBytes}.$$

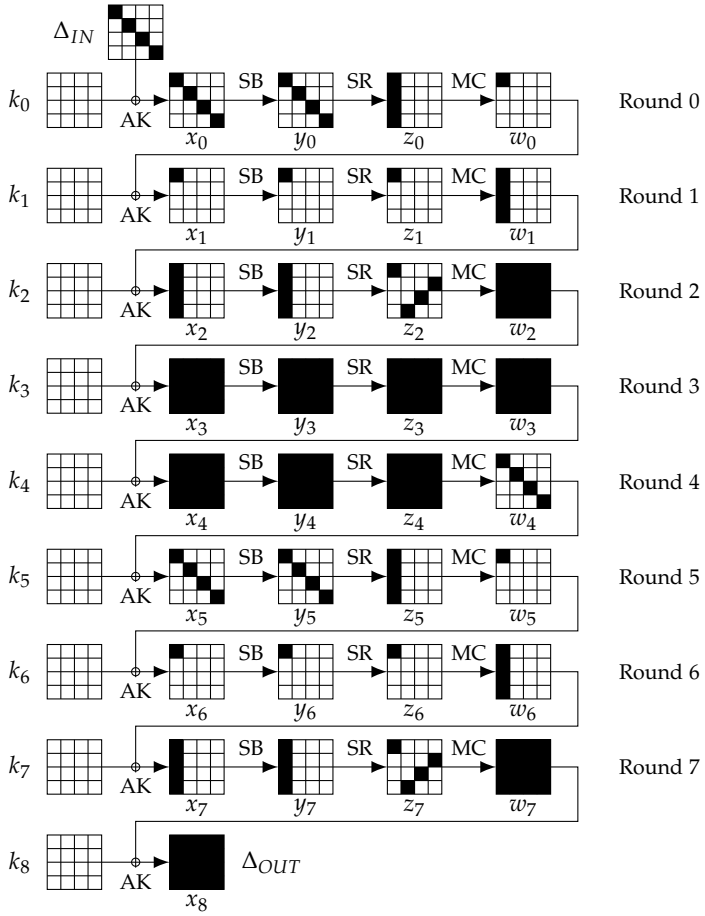


Figure 7.9: Truncated differential characteristic uses in the rebound distinguisher for 8 rounds of the known-key AES-like permutation (with $t = 4$). A black cell has non-zero difference while white cells are inactive.

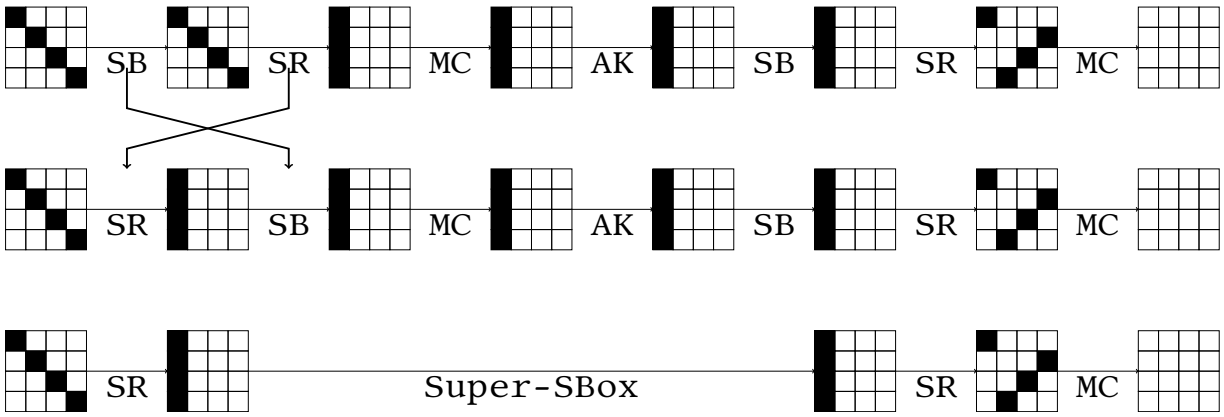


Figure 7.10: The **Super-SBox** construction: we obtain it by swapping the order of the operations in two rounds of the AES-like permutation. The black cells trace the evolution of the cells that go through *one* **Super-SBox**. There are t in parallel: one for each column ($t = 4$ in this example).

The keyed parameter k is a column of the subkey added in the inner **AddRoundKey**. In the current model, the key of the AES-like permutation is assumed to be known so each of the t **Super-SBox** permutation is fixed.

Distinguishing algorithm

The strategy of the distinguishing algorithm devised in [GP10] mimics the rebound technique from [MRST09], but the rebound itself is not performed on the c-bit S-Box on one round, but on the $c \times t$-bit **Super-SBox** on two rounds. With the same differential properties as S, the **Super-SBox** can also construct solutions for the middle rounds from round 2 to round 4 (inbound) and then we propagate them through the same probabilistic filters (outbound phase) that hold with probability $2^{-2c(t-1)}$.

While we could construct the difference distribution table (DDT) for the **Super-SBox**, it would require 2^{2ct} computations and $ct \times 2^{2ct}$ bits to store in memory. This would give the solutions to the **??** for any (Δ_i, Δ_o) in one memory access, but together with the cost of the outbound phase, the algorithm would run in $2^{2ct} + 2^{2c(t-1)}$, and that can easily be improved by not computing the large DDT that is the bottleneck.

Instead, the beginning of the algorithm is to randomize differences in y_2 (see Figure 7.9). From this, we can again compute linearly the difference $\Delta w_2 = \Delta x_3$ at the input of the **Super-SBox**. However now, we do not continue in guessing differences at the output of the **Super-SBox**, but we consider each of the t **Super-SBoxes** in parallel and enumerate all the $2^{ct}/2 = 2^{ct-1}$ input pairs of *values*. The factor 2 is explained by the redundancy of the pairs if we consider all the 2^{ct} values. Since the input difference has been fixed, one value is sufficient to compute the input pairs to the t parallel **Super-SBoxes**. So in 2^{ct-1} simple computations, we compute in parallel the output pairs of the t **Super-SBoxes** in y_4 and thus in z_4, and store each pair in a table T_i for the i-th **Super-SBox** indexed by their output differences. The memory requirements for this step are therefore $t \cdot 2^{ct-1}$ words of ct bits.

With the t precomputed tables, we can now randomize the difference Δw_4, compute backwards its image by **MixColumns** in z_4 and check the tables to find a match. As each table contains 2^{ct-1} pairs distributed in about 2^{ct} buckets, we expect to find a match in each of the t tables with a probability very close to $1/2$. As we have about $(2^c - 1)^t \approx 2^{ct}$ possible differences Δw_3, we expect that every 2^t try, we get 2^t solutions so that the precomputed tables allows to construct 2^{ct} solutions for the middle rounds.

Consequently, for a single input difference in y_2 and all the $(2^c - 1)^t$ differences in w_4, we try 2^{ct} pairs of states in the outbound phase that holds with probability $2^{-2c(t-1)}$. Thus, we expect to find a solution with probability $2^{ct-2c(t-1)} = 2^{-c(t-2)}$, which means that we need to repeat this entire procedure about $2^{c(t-2)}$ times with a new difference Δy_2. In total, the time complexity amounts to $2^{c(t-2)+ct} = 2^{2c(t-1)}$ operations, which can also be seen with the amortized cost 1 of the inbound phase that provides one solution with an average cost of one computation. Between the different attempts of Δy_2, the $t \cdot 2^{ct-1}$ words of the memory can be erased to be reused.

Generic complexity

As before, this algorithm is faster than the generic solution to the problem requiring about $LB(t,t) = 2^{ct(t-2)}$ operations if the following inequality holds:

$$2(t-1) \leq t(t-2),$$

which is independent of c and true as soon as $t \geq 2+\sqrt{2}$, that is $t \geq 4$.

7.2.2.2 Non-fully-active characteristic

At ASIACRYPT 2010, Sasaki et al. show in [SLW+10] how to decrease significantly the time complexity of the previous 8-round known-key distinguisher for larger values of t. We describe their main result with the use of the truncated differential characteristic depicted in Figure 7.11 when $t = 4$.

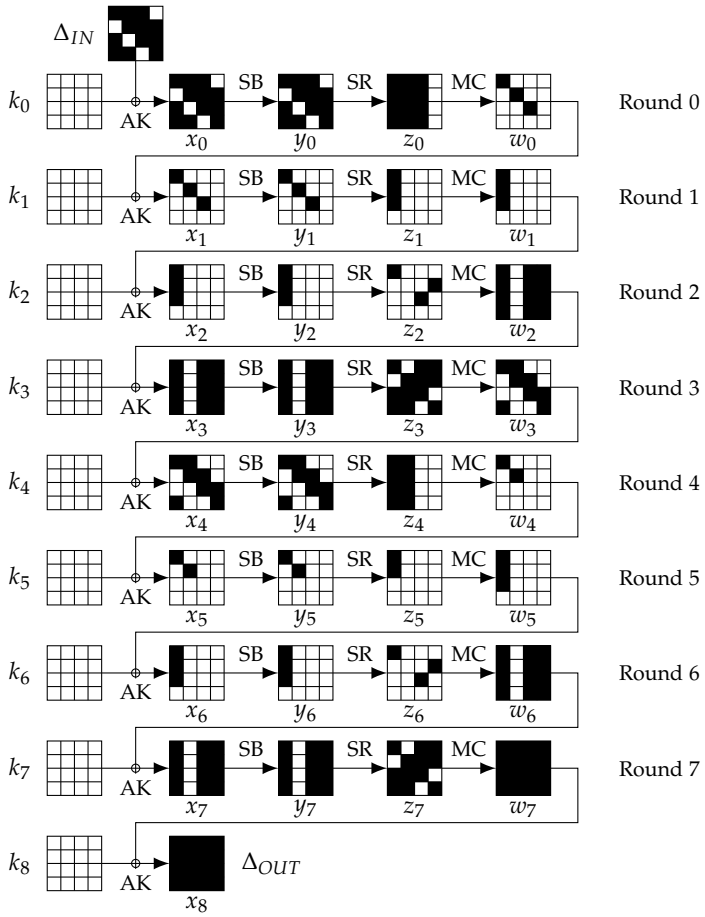


Figure 7.11: Non-fully-active truncated differential characteristic used in the improved rebound distinguisher for 8 rounds of the known-key AES-like permutation (with $t = 4$ and $s = 1$). A black cell has non-zero difference while white cells are inactive.

The main difference with previous work relies in the new truncated characteristic which does not include a fully active state of t^2 cells in the middle. This particular state acts as the key point to all the previous work since this is where we perform the rebound. Here, we still

use the rebound strategy, but the inbound phase becomes cheaper as the middle states are less active.

The non-fully-active characteristic is parameterized by a parameter s that describes the number of non-active "columns" in state w_2, and the number of non-active cells in the active column of z_1. On Figure 7.11, we have printed the characteristic with the value $s = 1$. We also note that in round 1, the **MixColumns** transition is a $t - s \to t - s$ transition, whereas in round 5, it is a $s + 1 \to t - s$ such that the MDS bound is tight ($s + 1 + t - s = t + 1$). The first transition implies $2(t - s) \geq t + 1$, i.e. $2s + 1 \leq t$ for the characteristic to be valid, otherwise the MDS property would be violated.

To apply the rebound technique, we split the characteristic in two parts: the inbound phase from y_2 to w_4, and the outbound phase that propagates outwards from those states. With the parameter s, the probability of the outbound becomes

$$2^{-c \cdot s} \cdot 2^{-c \cdot s} = 2^{-2cs},$$

by multiplying the probabilities of the two **MixColumns** transitions in rounds 1 and 5 that both require to cancel s cells of c bits. Consequently, the inbound *only* needs to generate 2^{cs} pairs for the middle rounds. This has to be compared with the $2^{2c(t-1)}$ requirements of the original rebound algorithm.

Non-fully-active Super-SBox

This new characteristic allows to consider different patterns for the **Super-SBox** construction, see Figure 7.12. Namely, from Figure 7.11 we see four difference patterns with different active positions at the input/output of the **Super-SBox**. This constrained situation has $2^{c \cdot t + c \cdot (t-s)} =$

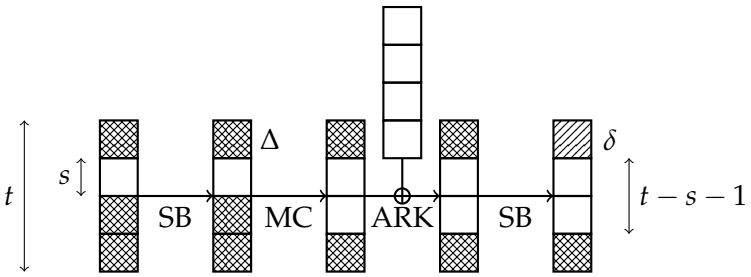


Figure 7.12: Doubly hatched cells have known non-zero difference, simply hatched cells have unknown non-zero differences, and white cells are inactive. The key is assumed to be known.

$2^{c \cdot (2t-s)}$ possible pairs of inputs and the probability that one hits a valid pattern at the output is $2^{-c \cdot (t-(s+1))}$. We therefore expect to find

$$2^{c \cdot (2t-s)} \times 2^{-c \cdot (t-(s+1))} = 2^{c \cdot (t+1)}$$

pairs that verify the truncated pattern.

To generate the pairs in amortized cost of 1 computation, we begin by randomizing all but one differences at the input and output of the considered **Super-SBox**. For instance, we let unconstrained the top difference at the output, but we set all the other differences to random

non-zero values (see Figure 7.12). We note that there are now exactly $t-s+t-(t-s-1)-1 = t$ fixed differences, so we now expect about 2^c pairs to verify all the conditions, and we can repeat this step about $(2^c-1)^t \approx 2^{ct}$ times with different values for the differences. The goal of the subsequent steps is to construct a table T indexed by the 2^c-1 values for the remaining difference δ at the output such that $T[\delta]$ contains a valid pair conforming to the differential in the **Super-SBox**.

We guess the value of the difference Δ after the first **SubBytes** layer. There are 2^c-1 possible values, and for each of them we compute a small linear system to deduce all the differences inside the **Super-SBox**. Indeed, the inner **MixColumns** operation has only 1 freedom degree for the differences. This is a consequence of the tightness of MDS bound: $t-s+s+1 = t+1$ for the transition $t-s \to s+1$. Therefore, the $t+1$ differences in the **Super-SBox** are linear functions of Δ, so we determine their values.

For all the active transitions with fixed output differences around the S-Box S or its inverse S^{-1} at both **SubBytes** layers, we find a matching pairs of values in small precomputed tables of 2^c elements. We observe that there are t such transitions, so that the probability that a match is found for all of them is 2^{-t}. In return, as soon as one pair is found for the t transitions, we can swap the elements of the pairs to construct a set of about 2^t solutions. So in average, we assume that we have one solution for any value of the difference Δ, but this behavior intrinsically assumes that $c \geq t$ otherwise we would not end up with t valid pairs even once.

At this point, all the doubly hatched cells of Figure 7.12 have known values and differences. We now determine the remaining differences. To do so, we observe that there are s unset values at the input on the inactive cells, and exactly s known values among the active cells at the output. We can thus solve a linear system of equations on the values to compute the missing ones at the input and propagate them at the output. After this step, everything is fixed, and we can compute the output difference δ and store the pair of inputs in $T[\delta]$.

From fixed differences in t cells at the input/output, this algorithm finds about 2^c pairs conforming the differential in 2^c computations and about 2^c memory units in small precomputed tables.

Distinguishing algorithm

To apply this efficient algorithm to the characteristic of Figure 7.11, we start by randomizing the values of the differences in y_2 which propagate linearly to x_3 at the input of the **Super-SBoxes**. We also randomize all but one of the differences in w_4 to reach exactly the requirements of the previous algorithm where all but one difference on each column of y_4 are active. There are $2^{c\cdot(t-s)}$ ways to randomize the input, and $2^{c\cdot s}$ for the output, so 2^{ct} in total.

For any $2^x \leq 2^{ct}$ different starting points of the differences in y_2 and w_4, we generate about $2^{x+c\cdot(t+1)}$ pairs for the middle rounds in $2^{x+c\cdot(t+1)}$ operations. As stated previously, we need to generate 2^{2cs} pairs for the middle rounds to pass the outbound phase in the outer rounds, which imposes

$$s \leq \frac{t+1}{2} \tag{7.1}$$

and a time complexity of 2^{2cs} computations to generate the 2^{2cs} pairs for a success probability

approximately 1 of the whole algorithm. We note that the previous $2s - 1 \leq t$ constraint from Equation 7.1 is weaker that the required condition $2s + 1 \leq t$ for the characteristic not to violate the MDS bounds. Additionally, as we said, we need $t \leq c$, which is true for all known AES-like permutations.

Generic complexity

As a comparison, the generic complexity is given by the time complexity of the limited-birthday algorithm in an equivalent setting: $LB\left(t \cdot (t-s), t \cdot (t-s)\right)$. With Equation 7.1, we can ensure that we have enough freedom degrees at the input to perform a simple birthday to reach a collision on $c \cdot t \cdot s$ bits in the output. Indeed, $s \leq (t+1)/2$ ensures that we can get as many as $2^{c \cdot t \cdot (t-s)} \geq 2^{c \cdot t \cdot (t-1)/2}$ input pairs, which allow to find a collision on at least $c \cdot t \cdot (t-1)$ output bits. If $s \geq 1$, then $t \cdot (t-1) \geq t \cdot (t-s)$, which confirms that we can collide on $c \cdot t \cdot s$ prescribed bits. We do not consider the case $s = 0$, since it does not allow to distinguish anything.

Hence, the time complexity of the generic algorithm equals

$$LB\Big(t \cdot (t-s), t \cdot (t-s)\Big) = 2^{c \cdot (t^2 - t \cdot (t-s))/2} = 2^{c \cdot t \cdot s/2},$$

and the algorithm described is a distinguishing algorithm as soon as

$$2 \cdot c \cdot s < \frac{c \cdot t \cdot s}{2},$$

which is true as soon as $t \geq 5$ for integer values of $s \in \left[1, \frac{t-1}{2}\right]$.

7.3 Chosen-key model

In this section, we switch to a slightly different model where the adversary can also use the bits of the key to exhibit nontrivial properties of the permutation. We call this the *chosen-key model* and it can actually be declined in two forms whether we allow the key to have differences or not. If there are differences, it is linked to the related-key model and we can cite the results from the previous Chapter 6 (see also [BKN09]), and if the key does not have differences, we call it single-chosen-key model.

In the next two sections, we present an improvement over the known-key methods to distinguish 7 and 8 rounds of AES by using the additional freedom brought by the key bits. Before this work, it was unknown how to actually use this extra freedom to decrease the complexities of the known-key algorithms. These two results have been published in a paper co-authored by Patrick Derbez and Pierre-Alain Fouque in [DFJ12a] at INDOCRYPT 2012. The main point of these two results consists in controlling one more transition $4 \rightarrow 1$ in the **MixColumns** that normally lies in the outbound phase. This way, we control it and gain a workfactor of 2^{24} in the time complexity.

While we generalize the known-key algorithms of the previous section to AES-like permutations, it is less obvious to do the same here, as the subkeys are linked by the key schedule equations. For instance, if the algorithm sets values for some part of a particular subkey, it may

impose strong constraints on other values for different subkeys, which we have to consider. Consequently in the following, we choose not to make the permutation as general as possible, but we consider explicitly the AES, and in particular AES-128 and AES-256.

The third result that we present here is the first known distinguisher for 9 rounds of AES-128 in any model. This one considers chosen and related keys to construct a pair of keys and a pair of inputs that verify a certain property which is about 2^{13} harder to find for a random permutation. This result has been published in the CRYPTO 2013 paper [FJP13a].

7.3.1 Distinguisher for 7-round AES

7.3.1.1 Distinguishing algorithm for AES-128

We consider the 7-round truncated differential characteristic of Figure 7.13 (same one as Figure 7.6), where the differences in both the plaintext and the ciphertext lie in subspaces of dimension four. Indeed, the output difference lies in a subspace of dimension four since all the operations after the last **SubBytes** layer are linear. With respect to the description of the distinguisher (Section 7.1.3), the time complexity to find a pair of messages that conforms to those patterns in a family of pseudo-random permutations is $LB(4,4) = 2^{64}$ computations.

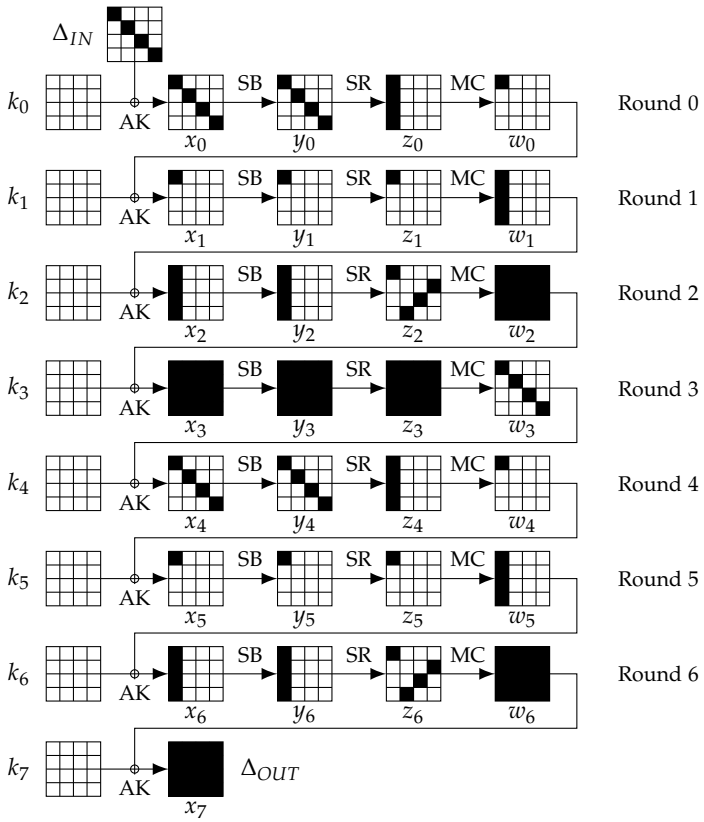


Figure 7.13: Truncated differential characteristic with $t = 4$ uses in the rebound distinguisher for 7 rounds of the known-key AES-like permutation. A black cell has non-zero difference while white cells are inactive.

The following of this section describes a way to build a key and a pair of messages that conform to the restrictions of the characteristic with 2^8 computations using a memory complexity of 2^8 bytes. This complexity has to be compared to 2^{16} computations, which is the minimal time complexity expected for a straightforward application of the rebound attack on the **SubBytes** layer of the AES (Section 7.2.1.2). Indeed, we need to repeat at least 2^{16} times a randomized phase and this cannot be decreased. In the following, we proceed slightly differently to reach a solution in 2^8 computations by using the key bits.

In terms of freedom degrees, we begin by estimating the number of solutions that we expect to verify the truncated differential characteristic. There are 16 bytes in the first message, 4 more independent ones in the second message and 16 others in the key: that makes 36 freedom degrees at the input. On a 36-byte random input, the probability that the truncated differential characteristic is followed depends on the amount of freedom degrees that we loose in probabilistic transitions within the **MixColumns** transitions:

- 3 in round 0 to pass one $4 \rightarrow 1$ transition,
- 12 in round 3 to pass four $4 \rightarrow 1$ transitions,
- 3 again in round 4 for the last $4 \rightarrow 1$ transition.

In total, we thus expect

$$2^{8\times(16+4+16)}\, 2^{-8\times(3+12+3)} = 2^{8\times 18}$$

triplets (m, m', k) composed by a pair (m, m') of messages and a key k to conform to the truncated differential characteristic of Figure 7.13. Hence, we have 18 freedom degrees left to find such a triplet.

First, we observe that whenever we find such a solution for the middle rounds (round 1 to round 4), we are ensured that all the rounds are covered in the complete truncated differential characteristic due to an outward propagation occurring with probability 1. This can be compared with the basic rebound technique and its improved version: in the original one (Section 7.2.1.2), the inbound phase controls none of the two $4 \rightarrow 1$ transitions and we need to perform 2^{48} computations in the outbound phase, then the improved algorithm (Section 7.2.1.3) includes one such transition in the inbound to make the outbound cost only 2^{24} computations. Here, we propose a way to control the two $4 \rightarrow 1$ transitions so that the outbound is verified with probability 1. Hence, our strategy focuses on the middle rounds.

To reduce the number of valid solutions, we begin by fixing some bytes (see Figure 7.14) to a random value: Δz_1 and $x_2[0, \ldots, 3]$. Therefore, we can deduce the values and differences in the first column of x_2 and y_2, as well as the difference Δx_3 by linearity.

Let $[\Delta_0, \Delta_1, \Delta_2, \Delta_3]^{\mathrm{T}}$ be the column-vector of deduced differences in Δy_2 and we note $\mathrm{diag}(\delta_0, \delta_1, \delta_2, \delta_3)$ the differences in the diagonal of Δx_4. Linearly, we can express the differences around the **SubBytes** layer of round 3 with these 8 variables (see Equation 7.2).

$$\Delta x_3 = \begin{pmatrix} 2\Delta_0 & \Delta_3 & \Delta_2 & 3\Delta_1 \\ \Delta_0 & \Delta_3 & 3\Delta_2 & 2\Delta_1 \\ \Delta_0 & 3\Delta_3 & 2\Delta_2 & \Delta_1 \\ 3\Delta_0 & 2\Delta_3 & \Delta_2 & \Delta_1 \end{pmatrix} \xrightarrow{\text{SB}} \begin{pmatrix} 14\delta_0 & 11\delta_1 & 13\delta_2 & 9\delta_3 \\ 13\delta_3 & 9\delta_0 & 14\delta_1 & 11\delta_2 \\ 14\delta_2 & 11\delta_3 & 14\delta_0 & 9\delta_1 \\ 13\delta_1 & 9\delta_2 & 14\delta_3 & 11\delta_0 \end{pmatrix} = \Delta y_3. \tag{7.2}$$

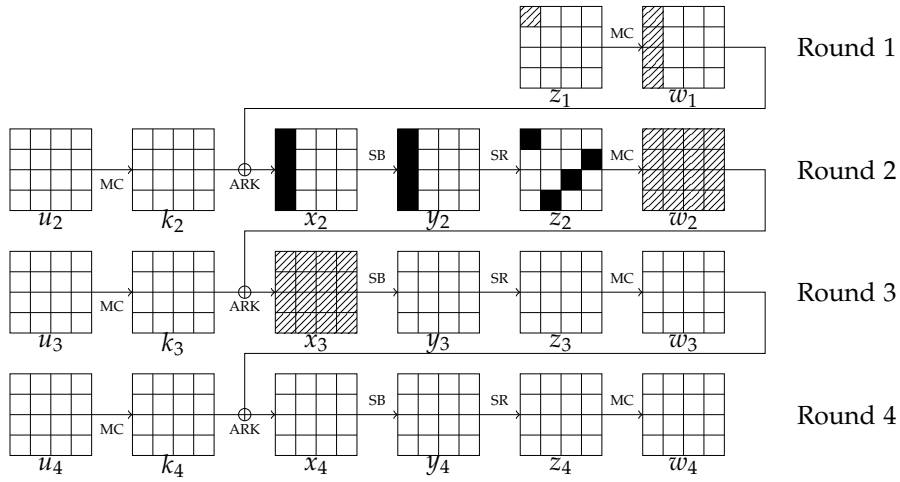


Figure 7.14: First step of the distinguishing algorithm. Black bytes have known values and differences, hatched bytes have known differences and white bytes have unknown values and/or differences.

As a consequence, from the differential properties of the AES S-Box (Theorem 4.2), for $i, j \in \{0, \ldots, 3\}$, Δ_j suggests $2^7 - 1$ different values for δ_i: we store them in the list $L_{i,j}$:

$$L_{i,j} = \left\{ \delta_i \;\middle/\; \Delta_j \to \delta_i \text{ is possible} \right\}. \tag{7.3}$$

By $\Delta \to \delta$ possible, we mean that the equation $\mathrm{S}(x) \oplus \mathrm{S}(x \oplus \Delta) = \Delta$ has at least two solutions; that is, the entry (Δ, δ) in the DDT of S is non-zero. Once the lists $L_{i,j}$ are constructed, we build the list L_i, for $i \in \{0, \ldots, 3\}$:

$$L_i = \bigcap_{j=0}^{3} L_{i,j} = \left\{ \delta_i \;\middle/\; \forall j \in \{0, \ldots, 3\},\ \Delta_j \to \delta_i \text{ is possible} \right\}. \tag{7.4}$$

Each $L_{i,j}$ being of size $2^7 - 1$, we expect each L_i to contain about 2^4 elements.

We continue by setting $\Delta x_4[0]$ to a random value drawn from L_0 and $x_4[0]$ to a random value, which allows to determine the value and difference in $y_4[0]$. Since the difference Δy_4 can only take $2^8 - 1$ values due to the **MixColumns** transition of round 4, we also deduce Δw_4 and the remaining differences in Δy_4. The knowledge of Δy_4 suggests about 2^7 possible values for δ_i. As before, we store them in lists called T_i, and we select a value for δ_i in $L_i \cap T_i$ (Figure 7.15). We expect each intersection to contain about 2^3 elements. More rigorously, if we assume that the lists $L_{i,j}$ and T_i are uniformly distributed, then the probability that L_0, $L_1 \cap T_1$, $L_2 \cap T_2$ and $L_3 \cap T_3$ are not empty is higher than 99.96%. We perform a detailed analysis of the success probability in the following Section 7.3.1.3. Finally, we compute the values in x_3 and in the diagonal of x_4.

We now need to find a key that matches the previous deduced values in the internal states. We have built a partial pair of internal states that conforms to the middle rounds, but it has fixed 8 bytes of constraints in the key. Namely, if we denote k_i the subkey introduced in round i and $u_i = \mathrm{MC}^{-1}(k_i)$, then both u_3 and k_4 have four known bytes (see Figure 7.16).

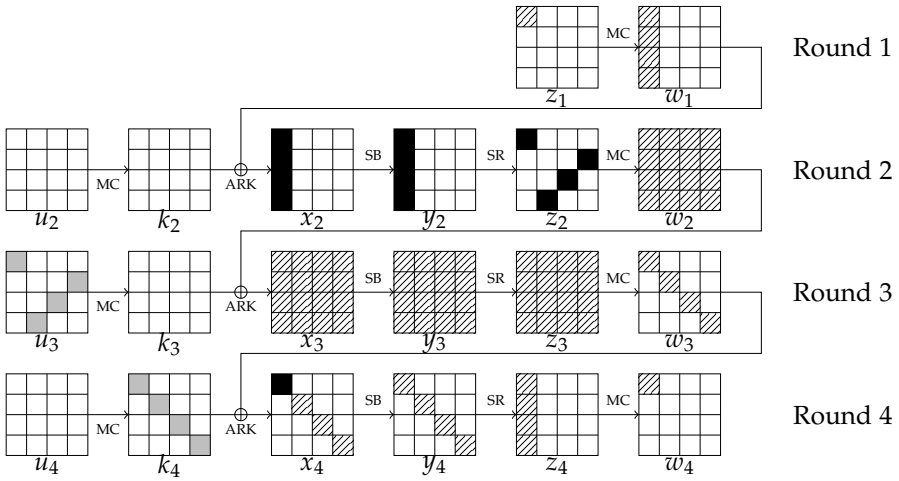


Figure 7.15: Second step of the distinguishing algorithm. Black bytes have known values and differences, gray bytes have known values, hatched bytes have known differences and white bytes have unknown values and/or differences.

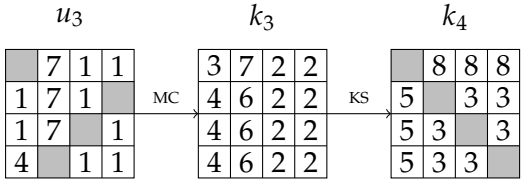


Figure 7.16: Generating a compatible key: gray bytes are known, and numbers indicate the order in which we guess or determine the bytes.

We start by fixing all the bytes marked by 1 in u_3 to random values: this allows to compute the values of all 2's in the two last columns of k_3. By the column-wise operations of the AES-128 key schedule, we can get the values of all bytes marked by 3. As for the 4's, we get them since there are four known bytes among the eight in the first columns of u_3 and k_3. Again, the key schedule gives the 5's and 6's, and the **MixColumns** the 7's. Finally, we determine values for all the bytes tagged by 8 from the key schedule equations. By inverting the key schedule, we are thus able to compute the master key k.

All in all, we start by getting a partial pair of internal states that conforms to the middle rounds, continue by deriving a valid key that matches the partial known bytes and determine the rest of the middle internal states to get the pair on input messages. The bottleneck of the time and memory complexity occurs when handling the lists of size at most 2^8 elements to compute intersections. Note that those intersections can be done in roughly 2^8 computations by representing lists by 256-bit numbers and then perform logical ANDs. We thus overcome the bottleneck time complexity of 2^{16} operations required by the rebound technique, but we cannot get any smaller than 2^8 computations, which actually equals 2^c computations with c the width of the S-Box.

In the end, we build a pair of messages (m, m') and a key k that conforms to the truncated differential characteristic of Figure 7.13 in 2^8 computations, where it requires 2^{64} computations

in the generic scenario. We note that among the 18 freedom degrees left for the attack, we used only 10 by setting 10 bytes to random values, so that we expect $2^{8\times 8} = 2^{64}$ solutions in total. All those solutions could be generated in 2^{64} computations by iterating over all the possibilities of the bytes marked by 1 in Figure 7.16.

7.3.1.2 Experimental verification

Since the complexity is more than practical, we have implemented this algorithm to verify that it indeed works. We have found for instance the following triplet (m, m', k) shown in Table 7.2.

Table 7.2: Example of a pair of messages (m, m') that conforms to the 7-round truncated differential characteristic for AES-128 of Figure 7.13. The master key found by the attack is: 93CA1344 10A7EBDF B659C8AF ECC59699. The lines in this array contain the values of two internal states before entering the corresponding round, as well as their difference.

Round	m	m'	$m \oplus m'$
Init.	E5FC5DFE 79A851F7 7EB9E366 51C3D9C5	F8FC5DFE 79C951F7 7EB96566 51C3D96E	1D000000 00610000 00008600 000000AB
0	76364EBA 690FBA28 C8E02BC9 BD064F5C	6B364EBA 696EBA28 C8E0ADC9 BD064FF7	1D000000 00610000 00008600 000000AB
1	65CC94D1 85BE1AD3 F3D75BF1 ACCBB8BD	8DCC94D1 85BE1AD3 F3D75BF1 ACCBB8BD	E8000000 00000000 00000000 00000000
2	E93319CD 88F41390 10623230 F66BFBAD	C92309FD 88F41390 10623230 F66BFBAD	20101030 00000000 00000000 00000000
3	89C79074 E09E6F44 F1DBAB2F F984FCC4	1404532A 09774F8D 24BF1AFA CD551921	9DC3C35E E9E920C9 D564B1D5 34D1E5E5
4	867A12E6 BF19139C 1C848362 400030D3	047A12E6 BF5B139C 1C847C62 400030D7	82000000 00420000 0000FF00 00000004
5	84606BEA 0E22D904 3BF29061 9F454807	4B606BEA 0E22D904 3BF29061 9F454807	CF000000 00000000 00000000 00000000
6	FF867544 274436AF 75ECC287 A6BF72F6	3C6A996B 274436AF 75ECC287 A6BF72F6	C3ECEC2F 00000000 00000000 00000000
End	C49E4CB3 0C944043 D5ED6D3B 247E3843	2563B1AF 68F0EC8B A6788B48 EEF27E05	E1FDFD1C 6464ACC8 7395E673 CA8C4646

7.3.1.3 Success probability

We propose here a formal analysis of the success probability of a single run of the algorithm. We are interested in the probability that the intersection of four or five subsets of $\{1, \ldots, 255\}$ each of size 128 is empty.

To evaluate it, let $\mathcal{P}$ denote the set of subsets $X \subset \{1, \ldots, 255\}$ such that $|X| = 128$. We also define:

$$T(n,k) := \{(X_1, \ldots, X_n) \in \mathcal{P}^n \mid |X_1 \cap \ldots \cap X_n| = k\} \qquad \text{for } n \geq 1, k \geq 0.$$

In others words, $|T(n,k)|/|\mathcal{P}^n|$ is the probability that the intersection of n elements from $\mathcal{P}$ has a size equal to k.

Property 7.1. *The cardinality of $T(n,k)$ satisfies the following recurrence relation:*

$$\begin{cases} \left|T(1,k)\right| & = \left|\mathcal{P}\right| \text{ if } k = 128, 0 \text{ otherwise} \\ \left|T(n+1,k)\right| & = \sum_{l=k}^{128} \left|T(n,l)\right| \binom{l}{k} \binom{255-l}{128-k} \end{cases} \qquad \text{for } n \geq 1, k \geq 0.$$

Proof. First, we note that we can partition $\mathcal{P}^n$ by the sets:

$$T(n,Y) := \{(X_1, \ldots, X_n) \in \mathcal{P}^n \mid X_1 \cap \ldots \cap X_n = Y\} \qquad \text{for any subset } Y \subset \{1, \ldots, 255\}.$$

Then, we have:

$$\begin{aligned}\left|T(n+1,k)\right| &= \sum_{Y} \left| \left\{ (X_1,\ldots,X_{n+1}) \in T(n,Y) \times \mathcal{P} \;\middle/\; \left|Y \cap X_{n+1}\right| = k \right\} \right| \\ &= \sum_{Y} \left|T(n,Y)\right| \times \left| \left\{ X \in \mathcal{P} \;\middle/\; \left|Y \cap X\right| = k \right\} \right|\end{aligned}$$

If we fix a set $Y \subset \{1,\ldots,255\}$, then a set $X \in \mathcal{P}$ such that $|X \cap Y| = k$ is obtained by choosing k elements in Y and $128-k$ elements in Y^c. As a consequence, we obtain:

$$\begin{aligned}\left|T(n+1,k)\right| &= \sum_{Y} \left|T(n,Y)\right| \binom{|Y|}{k} \binom{255-|Y|}{128-k} \\ &= \sum_{l=0}^{255} \binom{l}{k} \binom{255-l}{128-k} \sum_{|Y|=l} \left|T(n,Y)\right|.\end{aligned}$$

Finally, we remark that $\{T(n,Y)\}_{|Y|=l}$ is a partition of $T(n,l)$ and thus:

$$\left|T(n+1,k)\right| = \sum_{l=0}^{255} \binom{l}{k} \binom{255-l}{128-k} \left|T(n,l)\right|,$$

which ends the proof. ■

Using Maple, we found that the probability of failure of the distinguisher described in Section 7.3.1 equals:

$$\frac{T(4,0)}{|\mathcal{P}|^4} \times \left(\frac{T(5,0)}{|\mathcal{P}|^5}\right)^3 \approx 0.04\%.$$

7.3.1.4 Extension to 7-round AES-256

The first step of the attack described in the 7-round distinguisher on AES-128 (Section 7.3.1.1) still applies in the case of AES-256 since it does not involve the key schedule. Then, we can generate a compatible key easily since there are only two subkeys involved: we can just choose bytes of k_3 and k_4 as we want, except the imposed ones, and deduce the master key afterwards. This yields to a distinguisher with time and memory complexities around 2^8 for AES-256.

7.3.1.5 Extension to 8-round AES-256

We use a similar approach as the 7-round distinguisher on AES-128 of Section 7.3.1.1, but the truncated differential characteristic has one more fully active round in the middle (same one as Figure 7.9).

We begin by choosing values for Δz_1 and $x_2[0,\ldots,3]$. This allows to deduce Δx_2, Δy_2, and Δx_3. Then, we also set random values for Δw_5 and for the diagonal of x_5 to obtain both Δx_5 and Δy_4. Now, we find a value for Δx_4 that is compatible with Δx_3 and Δy_4. Indeed, we cannot take an arbitrary value for Δx_4 because the probability that it fits is very close to 2^{-32}. However, we can find a correct value with the following steps:

1. Store the $2^7 - 1$ possible values for $\Delta x_4[0]$ in a list L_0.

2. In a similar way, make lists L_1 with $\Delta x_4[1]$, L_2 with $\Delta x_4[2]$ and L_3 with $\Delta x_4[3]$.
3. Choose a value for $(x_3[0], x_3[5], x_3[10], x_3[15])$ and compute $\Delta x_4[0, \ldots, 3]$.
4. If $\Delta x_4[0, \ldots, 3]$ is not in $L_0 \times L_1 \times L_2 \times L_3$, then go back to step 3.

On average, we go back to the step 3 only $\left(2^{8-7}\right)^4 = 2^4$ times since lists are of size 2^7. In the same way, we can obtain values for the other columns of x_4.

At this point, we have computed actual values in all those internal states, and we need to generate a compatible key. Finding one can be done using the procedure described in Figure 7.17. Bytes tagged by 1 are chosen at random, odd steps use the key schedule equations and even steps the properties of **MixColumns**.

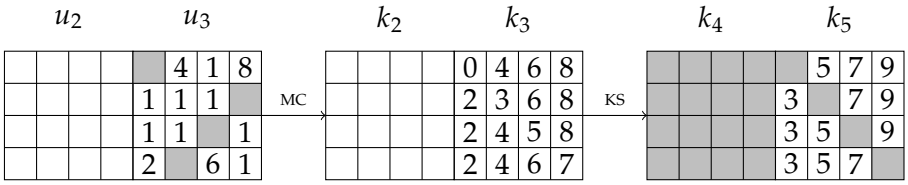


Figure 7.17: Generating a compatible key for AES-256: gray bytes are known, and numbers indicate the order in which we guess or determine the bytes.

7.3.2 Distinguisher for 8-round AES

7.3.2.1 Distinguishing algorithm for AES-128

We consider the 8-round truncated differential characteristic of Figure 7.18, where the states of differences in both the plaintext and the ciphertext lie in the same matrix subspaces of dimension four as before. Indeed, the output difference lies in a subspace of dimension four since all the operations after the last **SubBytes** layer are linear. Again, the distinguisher previously described (Section 7.1.3) claims that the time complexity to find a pair of messages that conforms to those patterns in a family of pseudo-random permutations requires $LB(4, 4) = 2^{64}$ computations.

The following of this section describes a way to build a key and a pair of messages that conform to this characteristic in time and memory complexity 2^{24}. We note that it is possible to optimize the memory requirement to 2^{16}. As in the previous section, there are 36 freedom degrees at the input, which shrink to 18 after the consideration of the truncated differential characteristic. Therefore, we also expect $2^{8\times 18}$ solutions in the end.

First of all, we observe that finding 2^{24} triplets (m, m', k) composed by a key and a pair of internal states that conform to the rounds 2 to 5 is sufficient since the propagation in the outward rounds is done with probability 2^{-24} due to the **MixColumns** transition of round 1. The following analysis consequently focuses of those four middle rounds.

In comparison with the previous algorithm for the known-key model due to Gilbert and Peyrin [GP10] described in Paragraph 7.2.2.1, we control one of the two $4 \rightarrow 1$ transitions in the inbound phase, such that we only need pass one at a cost of 2^{24} computations in the

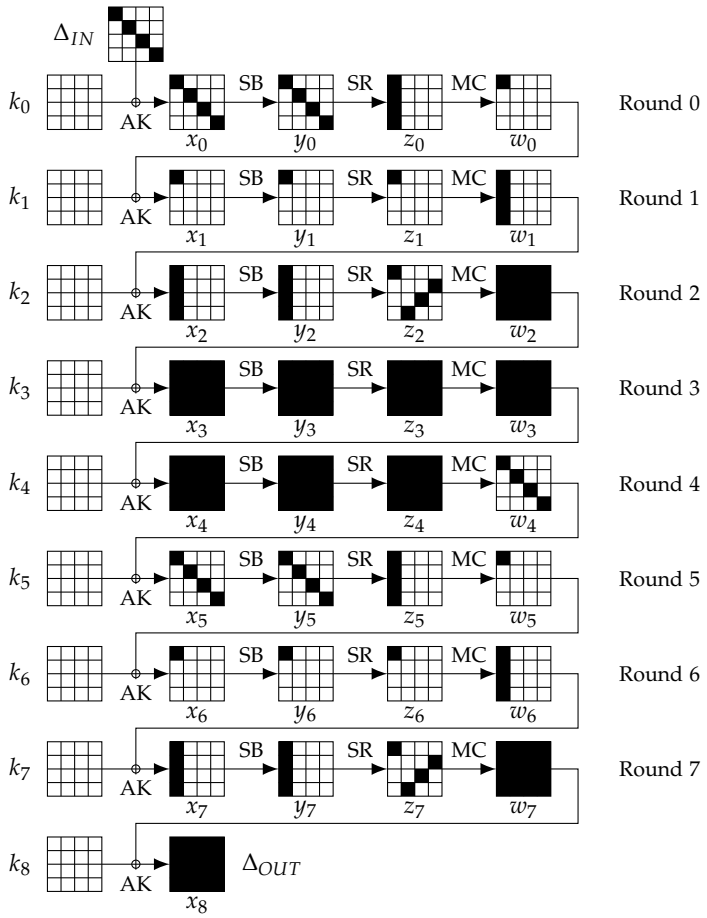


Figure 7.18: The 8-round truncated differential characteristic used to distinguish the AES-128. Black bytes are active, white bytes are not.

outbound phase. To date, it is still unknown how to push this even further by controlling the two transitions to get an 8-round distinguisher with a workfactor of roughly 2^8 operations.

We now describe an instance of a problem that we use as a building block in our algorithm, which is related to the keyed **Super-SBox** construction.

Problem 7.2. *Let a and b be two bytes. Given 32-bit input and output differences Δ_{in} and Δ_{out} of a $\boldsymbol{Super\text{-}SBox}_k$ for an unconstrained k, find all the pairs of 32-bit* AES*-columns (c, c') and keys k such that:*

i. $c + c' = \Delta_{in}$,
ii. $\boldsymbol{Super\text{-}SBox}_k(c) + \boldsymbol{Super\text{-}SBox}_k(c') = \Delta_{out}$,
iii. $\boldsymbol{Super\text{-}SBox}_k(c) = [a, b, \star, \star]^{\mathrm{T}}$.

Considering the key k known and the case where there is no restriction on the output bytes (*iii*), we would expect this problem to have one solution on average. Finding it would naively require 2^{32} computations by iterating over the 2^{32} possible inputs and check whether the output has the correct Δ_{out} known difference. The additional constraints on the two output

bytes reduce the success of finding a pair (c, c') of inputs to 2^{-16}, but if we allow the four bytes in the key k to be chosen, then we expect 2^{16} solutions to this problem.

To find all of them in 2^{16} computations, we proceed as follows (Figure 7.19): the two output bytes a and b being known, we can deduce the values of the two associated bytes before the last **SubBytes**, $\tilde{a}$ and $\tilde{b}$ respectively. We can also deduce the differences in those bytes since their output differences are known. Then, we guess the two unset differences at the input of the last **SubBytes**: the differences then propagate completely inside the **Super-SBox**. At both **SubBytes** layers, by the differential properties of the AES S-Box, we expect to find one value on average for each of the six unset transitions. Consequently, the input and output of the **AddRoundKey** operation are known, which determine the four bytes of k. In the end, we find the 2^{16} solutions of Problem 7.2 in 2^{16} computations.

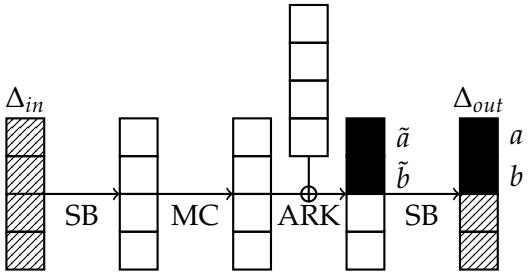


Figure 7.19: Black bytes have known values and differences, hatched bytes have known differences and white bytes have unknown values and/or differences.

To apply this strategy to the 8-round truncated differential characteristic of Figure 7.18, we start by randomizing the difference Δy_2, the difference Δw_5 and the values in the first column of w_5. Due to the linear operations involved, we deduce $\Delta x_3 = \Delta w_2$ from Δy_2 and Δy_4 from Δw_4. To use the previous algorithm, we randomize the values of the two first columns of w_4 (see Figure 7.20). Doing so, the four columns of y_4 are constrained on two bytes each and have fixed differences. Consequently, the four **Super-SBoxes** between x_3 and y_4 keyed by the four corresponding columns of k_4 conforms to the requirements of Problem 7.2. The positions of the known output bytes differ, but the strategy applies in the same way. In time and memory complexities 2^{16}, for $i \in \{0, 1, 2, 3\}$, we store the 2^{16} solutions for the i-th **Super-SBox** associated to the i-th column of x_4 in the list L_i.

We continue by observing that the randomization of the bytes in w_4 actually sets the value of two diagonal bytes in k_5 ($k_5[0]$ and $k_5[5]$), which imposes constraints on the elements in the lists L_i. We start by considering the 2^{16} elements of L_3, and for each of them, we learn the values $x_4[12, \ldots, 15]$ and $k_4[12, \ldots, 15]$. Due to the column-wise operations in the key schedule, we also deduce the value of $k_4[0]$. By filtering the elements of L_0 that share this value of $k_4[0]$, we are left with 2^8 elements for bytes $x_4[0, \ldots, 3]$ and $k_4[0, \ldots, 3]$. At this point, we constructed $2^{16+8} = 2^{24}$ solutions in 2^{24} computations that we store in a list $L_{0,3}$.

As $k_5[5]$ has been previously determined, we can deduce $k_4[5] = k_5[5] + k_5[1]$ from the AES key schedule for each entry of $L_{0,3}$. Again, this adds an 8-bit constraint on the elements of L_1: we expect 2^8 of them to match the condition on $k_4[5]$. In total, we could construct a list $L_{0,1,3}$ of size $2^{24+8} = 2^{32}$, whose elements would be the columns 0, 1 and 3 of x_4 and k_4, but as soon as we get 2^{24} elements in that list, we stop and discard the remaining possibilities.

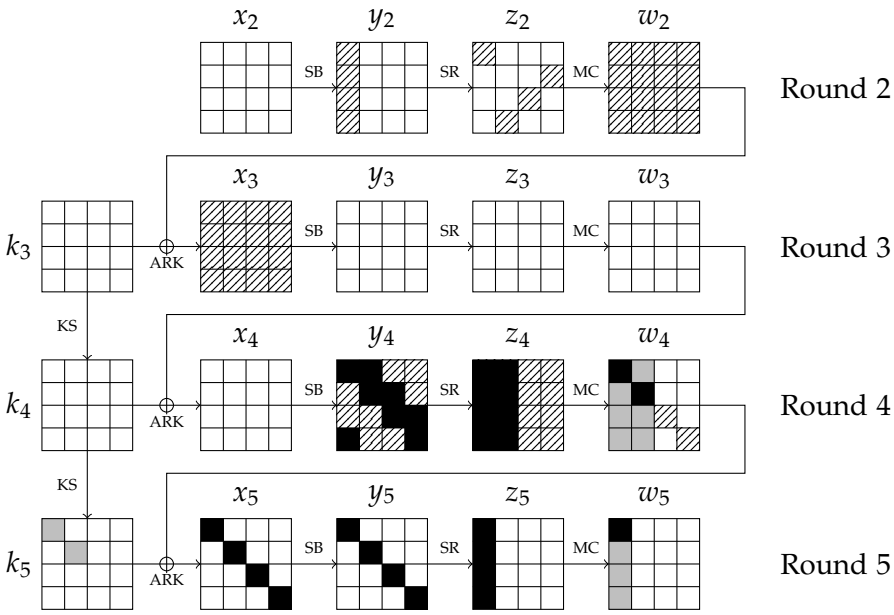


Figure 7.20: Black bytes have known values and differences, gray bytes have known values, hatched bytes have known differences and white bytes have unknown values and/or differences.

Finally, to ensure the correctness of the choice in the remaining column 2, we need to consider the subkey k_5 and the **MixColumns** operation in round 4. Indeed, as soon as we choose an element in both $L_{0,1,3}$ and L_2, x_4, k_4 and k_5 become fully determined, but we need to ensure that the values $x_5[10]$ and $x_5[15]$ are consistent with the known ones. In particular, for $x_5[10]$, we have:

$$\begin{aligned} k_4[10] + k_5[6] &= k_5[10] && (7.5)\\ &= w_4[10] + x_5[10] && (7.6)\\ &= z_4[8] + z_4[9] + 2z_4[10] + 3z_4[11] + x_5[10], && (7.7) \end{aligned}$$

and for $x_5[15]$:

$$\begin{aligned} k_4[11] + k_5[7] + k_4[15] &= k_5[11] + k_4[15] && (7.8)\\ &= k_5[15] && (7.9)\\ &= w_4[15] + x_5[15] && (7.10)\\ &= 3z_4[12] + z_4[13] + z_4[14] + 2z_4[15] + x_5[15], && (7.11) \end{aligned}$$

where (7.5), (7.8) and (7.9) come from the key schedule, (7.6) and (7.10) from the **AddRoundKey** and (7.7) and (7.11) use the equations from the **MixColumns**. Hence, for each element of $L_{0,1,3}$, we can compute:

$$\begin{aligned} \mathrm{S}(x_4[8]) + k_4[10] &:= x_5[10] + k_5[6] + \mathrm{S}(x_4[13]) + 2\,\mathrm{S}(x_4[2]) + 3\,\mathrm{S}(x_4[7]), && (7.12)\\ k_4[11] + 2\,\mathrm{S}(x_4[11]) &:= k_5[7] + k_4[15] + 3\,\mathrm{S}(x_4[12]) + \mathrm{S}(x_4[1]) + \mathrm{S}(x_4[6]) + x_5[15] && (7.13) \end{aligned}$$

and lookup in L_2 to find $2^{16}\,2^{-8\times 2} = 1$ element that match those two 8-bit conditions. We create the list L by adding the found element from L_2 to each entry of $L_{0,1,3}$.

All in all, in time and memory complexities 2^{24}, we build L of size 2^{24} and we now exhaust its elements to find one that verifies the 2^{-24} probability of the $4 \to 1$ backward transition in the **MixColumns** of round 1. Consequently, we expect to find a pair (m, m') of messages and a key k that conforms to the 8-round truncated differential characteristic of Figure 7.18 in 2^{24} computations, while it requires $LB(4,4) = 2^{64}$ computations in the ideal case.

Among the 18 available freedom degrees available to mount the attack, we use 17 of them, which means that we expect to have 2^8 solutions. We could have them in time 2^{32}, but since we discarded 2^8 elements in the algorithm described, we get only 1 in 2^{24} computations. We note that it is possible to gain a factor 2^8 in the memory requirements of our attack since we can implement the algorithm without storing the lists L_0, $L_{0,3}$ and $L_{0,1,3}$, by using hash tables for L_1, L_2 and L_3.

7.3.2.2 Experimental verification

We have also implemented this algorithm to verify that it indeed works, and we have found for instance the triplet (m, m', k) reported in Table 7.3.

Table 7.3: Example of a pair of messages (m, m') that conforms to the 8-round truncated differential characteristic for AES-128 of Figure 7.18. The master key found by the attack is: 98C45623 6CA00686 301E836D 614DFAB0. The lines in this array contains the values of two internal states before entering the corresponding round, as well as their difference.

Round	m	m'	$m \oplus m'$
Init.	9588B342 D43D04D4 AB298AE1 E43687DB	0B88B342 D46904D4 AB29D0E1 E4368728	9E000000 00540000 00005A00 000000F3
0	0D4CE561 B89D0252 9B37098C 857B7D6B	934CE561 B8C90252 9B37538C 857B7D98	9E000000 00540000 00005A00 000000F3
1	53FEBB0F 6BFF8E5E B471A8E3 1A2232A3	0EFEBB0F 6BFF8E5E B471A8E3 1A2232A3	5D000000 00000000 00000000 00000000
2	E9F44380 991A8ECB F7B18344 2C936CEB	65B2054A 991A8ECB F7B18344 2C936CEB	8C4646CA 00000000 00000000 00000000
3	2977F65C 3883EDEF 615D3C9E 5CE5384B	8F24A5A9 2398C0D9 10CEDEEF DFEEB0C3	A65353F5 1B1B2D36 7193E271 830B8888
4	BB1DB144 2BE947C3 5FCD89DF DF1CA0EB	82188658 42FFCAAE B337F0CA 09AB1513	3905371C 69168D6D ECFA7915 D6B7B5F8
5	C3E1961D 02A9713E 770A20D4 5470FA8F	8DE1961D 029B713E 770A3AD4 5470FA27	4E000000 00320000 00001A00 000000A8
6	D79D534C 33CC3861 76635DCD 548870C9	EB9D534C 33CC3861 76635DCD 548870C9	3C000000 00000000 00000000 00000000
7	D7F645C6 89358035 09847940 D831EFDE	0211A2F4 89358035 09847940 D831EFDE	D5E7E732 00000000 00000000 00000000
End	16E58308 DFD78F11 A8B05B9D C0A0363E	E49CFA83 D4DC9207 FC4CF3C9 9B3BF6FE	F279798B 0B0B1D16 54FCA854 5B9BC0C0

7.3.2.3 Extension to 9-round AES-256

We begin as in Section 7.3.2.1 by choosing the difference Δy_2, the difference Δw_6 and the values in the first column of w_6. Then, we deduce $\Delta w_2 = \Delta x_3$ from Δy_2 and Δy_5 from Δw_5. In addition, we set x_3 to a random value, which allows to determine Δx_4. In order to apply the result from Problem 7.2 again, we set the values in the two first columns of w_5 to random values.

As before, for $i \in \{0,1,2,3\}$, we store in the list L_i the 2^{16} possible values of the i-th column of x_5 and the i-th column of k_5. Unlike previously, we also obtain values of the i-th column of $\mathrm{SR}(k_4)$, but the scenario of the attack still applies. We start by observing that bytes of L_0 allow to compute $k_4[1]$ and $k_4[13]$, which are bytes of L_3. Thus, we can merge L_0 and L_3 in a list $L_{0,3}$ containing 2^{16} elements. Then, we construct the list $L_{0,2,3}$ containing 2^{24} elements of $L_{0,3} \times L_2$.

Finally, from bytes of $L_{0,2,3}$, we can compute:

$$3z_5[11] := k_4[2] + \mathrm{S}(k_5[15]) + k_4[6] + k_4[10] + z_5[8] + z_5[9] + 2z_5[10] + x_6[10], \quad (7.14)$$

$$z_5[14] + k_4[3] := \mathrm{S}(k_5[12]) + k_4[7] + k_4[11] + k_4[15] + 3z_5[12] + z_5[13] + 2z_5[15] + x_6[15]. \quad (7.15)$$

As a consequence, we expect only one element of L_1 to satisfy those two 8-bit conditions and so, we obtain 2^{24} solutions for the middle rounds. All in all, this yields a distinguisher with a time complexity around 2^{24} computations and memory requirements around 2^{16} using the same trick given in Section 7.3.2.1.

7.3.3 Distinguisher for 9-round AES-128

In this section, we show an algorithm to distinguish 9 rounds of AES-128 in the chosen-key model. A nontrivial result for AES-128 reduced to 9 rounds has been an open problem for a long time as we could not extend the 8-round distinguishers presented previously, nor the recent advances made on a larger geometry with the algorithm suggested in [JNPP12] and described later in Section 8.2.

The algorithm implemented by the distinguisher tries to find two related permutations AES_k and $\mathrm{AES}_{k'}$ and an input m and m' to each one such that the differences $m \oplus m'$ and $\mathrm{AES}_k(m) \oplus \mathrm{AES}_{k'}(m')$ verify specified relations. The relation between the two keys k and k' is $k \oplus k' = \delta$, where δ is a fixed and known difference (defined in Figure 7.21 and Table 7.4).

We achieve this result by considering the best 5-round differential characteristic that exists on AES-128 and propagating it backwards to reach 9 rounds. This reduced differential characteristic is not truncated and has been first described as a truncated one in [BN10]. Later, the tool we have developed to automatically search for related-key differential characteristics in AES-like permutations (Chapter 6) have rediscovered it and instantiated the truncated differences to construct an actual differential characteristic. This is the one we use in the following.

The 5 last rounds hence count 6 active S-Boxes in the key schedule part and 11 in the data part (rounds 4 to 8 in Figure 7.21). By the backward propagation in the key schedule, we reach a total of 15 active S-Boxes for the key schedule differential characteristic, whose probability equals 2^{-101}. Since we have 2^{128} possible key values, we expect about 2^{27} pairs of keys to conform to the differential characteristic in the key schedule. In the block cipher part, we prepend three rounds that we plan to control with an average cost of one computation using the **Super-SBox** technique [DR02, GP10], and one more round at the very beginning that we make as sparse as possible. The entire 9-round differential characteristic is depicted on Figure 7.21. We also represent the same characteristic without colors in Table 7.4.

7.3.3.1 Distinguishing algorithm

Once this differential characteristic settled, we find inputs that verify the whole characteristic (see Algorithm 7.1). We start by finding a pair of keys that conforms to the whole differential characteristic in the key schedule. There are 2^{27} expected such pairs of keys, and we can generate them at an average cost of one computation by picking random values satisfying all

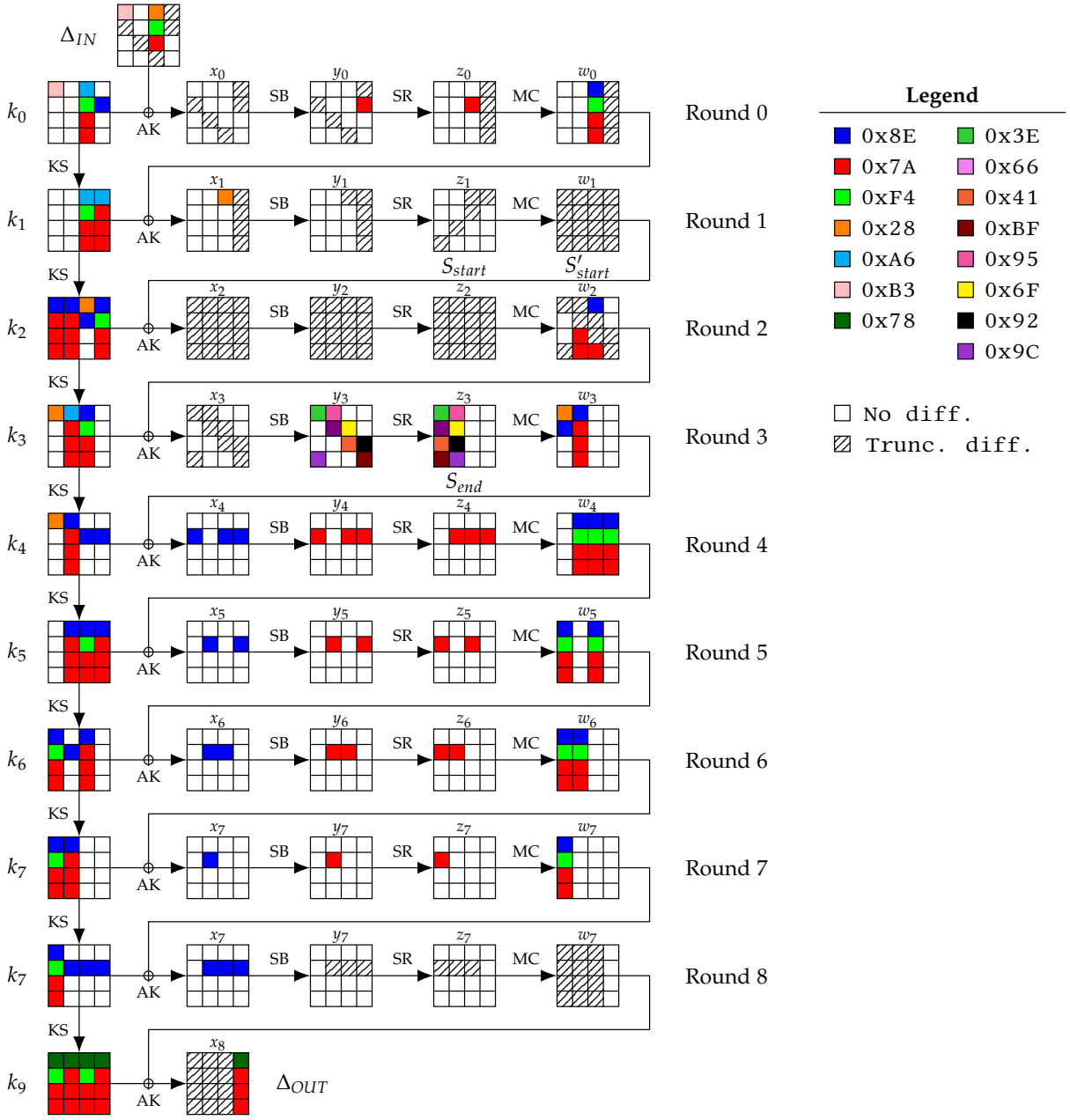


Figure 7.21: Differential characteristic of 9-round AES-128 used in the distinguisher. The colors map to actual values for the differences, whereas hatched bytes are truncated differences and white ones are inactive. See also Table 7.4.

the non-linear transitions and then efficiently solve a linear system to retrieve all the subkeys. We get for instance the pair of keys shown in Table 7.5 and our implementation confirms about 2^{27} are found.

For a pair of keys, we precompute the four arrays T_i that contain the paired values of the i-th **Super-SBox**. To construct the tables T_i, we iterate in parallel over the 2^{32} input values from state S_{end} that corresponds to the i-th **Super-SBox** and propagate the values backwards until S'_{start}. We note that the difference in S_{end} is completely determined by our differential

Table 7.4: Differential characteristic used in the distinguisher of 9 rounds of AES-128 in Figure 7.21. The known differences are represented by their values from 0x00 to 0xFF, and truncated differences as ??, since their values are unknown, but positive. The two lines for state differences are respectively the input difference after key addition and the output difference.

Round	State differences	Key differences
Plaintext	B3??0000 0000??00 28F47A?? ????0000	
1	00??0000 0000??00 000000?? ????0000 00000000 00000000 8EF47A7A ????????	B3000000 00000000 A6F47A7A 008E0000
2	00000000 00000000 28000000 ???????? ???????? ???????? ???????? ????????	00000000 00000000 A6F47A7A A67A7A7A
3	???????? ???????? ???????? ???????? ??0000?? ????7A7A 8E????7A 0000????	8E7A7A7A 8E7A7A7A 288E0000 8EF47A7A
4	??0000?? ????0000 00????00 0000???? 288E0000 8E7A7A7A 00000000 00000000	28000000 A67A7A7A 8EF47A7A 00000000
5	008E0000 00000000 008E0000 008E0000 00000000 8EF47A7A 8EF47A7A 8EF47A7A	28000000 8E7A7A7A 008E0000 008E0000
6	00000000 008E0000 00000000 008E0000 8EF47A7A 00000000 8EF47A7A 00000000	00000000 8E7A7A7A 8EF47A7A 8E7A7A7A
7	00000000 008E0000 008E0000 00000000 8EF47A7A 8EF47A7A 00000000 00000000	8EF47A7A 008E0000 8E7A7A7A 00000000
8	00000000 008E0000 00000000 00000000 8EF47A7A 00000000 00000000 00000000	8EF47A7A 8E7A7A7A 00000000 00000000
9	00000000 008E0000 008E0000 008E0000 ???????? ???????? ???????? 00000000	8EF47A7A 008E0000 008E0000 008E0000
Ciphertext	???????? ???????? ???????? 787A7A7A	78F47A7A 787A7A7A 78F47A7A 787A7A7A

Algorithm 7.1 – Distinguishing algorithm for 9 rounds of AES-128.

```
1:  function DISTINGUISHER()
2:    while True do
3:      Find (k, k ⊕ δ) conforming to the KS characteristic      ▷ Done in amortized cost 1
4:      for i ∈ {0, ..., 3} do                                  ▷ About 2^32 operations in parallel
5:        construct the array T_i of the i-th Super-SBox
6:      for all values of the 5 differences in S_start do       ▷ Done in 2^40 simple operations
7:        Use tables T_i to get a pair of messages (m, m')
            verifying the characteristic from S_start to S_end
8:      if backward transition not verified then continue       ▷ Verified with probability 2^-7
9:      if forward transitions not verified then continue       ▷ Verified with probability 2^-48
10:     return (k, m, m')                                       ▷ Returns after about 2^55 operations
```

characteristic. We store the pair in T_i indexed by its difference, so that this precomputation requires 2^{32} computations, a memory complexity of 2^{32}, and depends on the selected pair of keys. To simplify, we assume the differences in S'_{start} to be uniformly distributed so that each 32-bit difference appears once. While this is not the case in practice, the average cost to find one solution remains one, so it does not change the complexity estimation.

We continue by picking random values for the 5-byte differences after the second non-linear layer in S_{start}, which linearly fixes all the differences in S'_{start}. Note that we can repeat this

Table 7.5: Example of a pair of keys conforming to the differential characteristic of our 9-round distinguisher of AES-128 (Figure 7.21). There are about 2^{27} such pairs.

Round	k	k'	$k \oplus k'$
0	BD219F91 37EBDD3C 623F76DB 34AD0BBB	0E219F91 37EBDD3C C4CB0CA1 34230BBB	B3000000 00000000 A6F47A7A 008E0000
1	290A7589 1EE1A8B5 7CDEDE6E 4873D5D5	290A7589 1EE1A8B5 DA2AA414 EE09AFAF	00000000 00000000 A6F47A7A A67A7A7A
2	A40976DB BAE8DE6E C6360000 8E45D5D5	2A730CA1 3492A414 EEB80000 00B1AFAF	8E7A7A7A 8E7A7A7A 288E0000 8EF47A7A
3	CE0A75C2 74E2ABAC B2D4ABAC 3C917E79	E60A75C2 D298D1D6 3C20D1D6 3C917E79	28000000 A67A7A7A 8EF47A7A 00000000
4	47F9C329 331B6885 81CFC329 BD5EBD50	6FF9C329 BD6112FF 8141C329 BDD0BD50	28000000 8E7A7A7A 008E0000 008E0000
5	0F839053 3C98F8D6 BD573BFF 000986AF	0F839053 B2E282AC 33A34185 8E73FCD5	00000000 8E7A7A7A 8EF47A7A 8E7A7A7A
6	2EC7E930 125F11E6 AF082A19 AF01ACB6	A033934A 12D111E6 21725063 AF01ACB6	8EF47A7A 008E0000 8E7A7A7A 00000000
7	1256A749 0009B6AF AF019CB6 00003000	9CA2DD33 8E73CCD5 AF019CB6 00003000	8EF47A7A 8E7A7A7A 00000000 00000000
8	F152C42A F15B7285 5E5AEE33 5E5ADE33	7FA6BE50 F1D57285 5ED4EE33 5ED4DE33	8EF47A7A 008E0000 008E0000 008E0000
9	544F0772 A51475F7 FB4E9BC4 A51445F7	2CBB7D08 DD6E0F8D 83BAE1BE DD6E3F8D	78F47A7A 787A7A7A 78F47A7A 787A7A7A

part about $2^{8.5} = 2^{40}$ times. From the precomputed tables T_i, we find on average one pair of messages that verifies the middle rounds from S_{start} to S_{end}. The remaining of the algorithm is probabilistic: backwards, we expect a fraction of 2^{-7} pairs to pass the unique specified S-Box transition in the second round up to Δ_{IN}. Forwards, we expect a fraction of $2^{-6\times 8} = 2^{-48}$ to verify the 5 last rounds up to Δ_{OUT} (all 8 transitions have been chosen by our tool to be 8 times the same one with maximal probability $p_{max} = 2^{-6}$). Finally, we expect a fraction $2^{-7-48} = 2^{-55}$ of the pairs generated in the middle to propagate correctly forwards and backwards.

By repeating this process for all 2^{40} differences in S_{start} and for 2^{15} distinct pairs of keys, we expect to find a solution for the whole characteristic in $2^{15} \cdot (2^{32} + 2^{40}) \approx 2^{55}$ computations. Note that the remaining freedom degrees allow to get up to 2^{12} solutions in 2^{67} computations by exhausting the remaining 2^{12} valid pairs of keys.

7.3.3.2 Generic case

For an ideal cipher, the adversary faces a family $\{\pi_i, i \in \{0,1\}^{128}\}$ of random permutations. His goal is to find a key k and a pair of messages (m, m') such that: $m \oplus m' \in \Delta_{IN}$ and $\pi_k(m) \oplus \pi_{k\oplus\delta}(m') \in \Delta_{OUT}$, where δ, Δ_{IN} and Δ_{OUT} are specified in Figure 7.21. We recall them here in hexadecimal notations:

$$\delta = \begin{pmatrix} B3 & 00 & A6 & 00 \\ 00 & 00 & F4 & 8E \\ 00 & 00 & 7A & 00 \\ 00 & 00 & 7A & 00 \end{pmatrix}, \Delta_{IN} = \begin{pmatrix} B3 & 00 & 28 & ?? \\ ?? & 00 & F4 & ?? \\ 00 & ?? & 7A & 00 \\ 00 & 00 & ?? & 00 \end{pmatrix}, \Delta_{OUT} = \begin{pmatrix} ?? & ?? & ?? & 78 \\ ?? & ?? & ?? & 7A \\ ?? & ?? & ?? & 7A \\ ?? & ?? & ?? & 7A \end{pmatrix}.$$

The values marked by ?? are non-zero truncated differences.

On the output, we constrain each of the three independent active bytes in Δy_7 after the last non-linear layer of the last round to only 127 reachable difference values (since from a fixed input difference, only 127 output differences can be reached through the AES S-Box), and the **MixColumns** layer being linear we have $|\Delta_{OUT}| = 127^3 \approx 2^{21}$. On the input, 4 bytes in Δ_{IN} can take any difference value and 1 byte is constrained to only 127 reachable difference values, thus $|\Delta_{IN}| = 127 \cdot (2^8 - 1)^4 \approx 2^{39}$.

The best known way for the attacker to find (k, m, m') verifying those properties consists in applying the limited birthday algorithm [GP10]. The additional freedom left in choosing the

key bits does not help the attacker to find the actual pair of messages that verifies the required property, since the permutations π_k and $\pi_{k\oplus\delta}$ have to be chosen beforehand. All in all, the attacker has access to 39 bits of differences at the input and 21 bits at the output, for a pair of permutations on $ct^2 = 128$ bits. The limited birthday distinguisher on these permutations finds a solution in time

$$\max\left(\min\left(2^{IN/2}, 2^{OUT/2}\right), 2^{IN+OUT-n}\right),$$

with $IN = ct^2 - 39$ and $OUT = ct^2 - 21$, which gives a time complexity equivalent to 2^{68} encryptions.

Consequently, the algorithm we have presented acts as a distinguisher for the AES-128 reduced to 9 rounds 2^{13} times faster than the generic solution with a time complexity of 2^{55} computations. We note that while we can generate 2^{12} solutions with our algorithm in about 2^{67} computations and 2^{32} memory, it would require approximately $2^{12+68} = 2^{80}$ encryptions for the generic case.

CHAPTER 8

Improved Rebound Algorithms

Contents

8.1 Description of some AES-like primitives **180**
8.1.1 Description of Grøstl 180
8.1.2 Description of PHOTON 182
8.1.3 Description of LED 183
8.1.4 Description of Whirlpool 184
8.2 Improved Inbound Part **185**
8.2.1 Fully-active truncated differential characteristic 186
8.2.2 Non-fully-active truncated differential characteristic 192
8.2.3 Application to Grøstl-256 permutations 197
8.2.4 Distinguisher for 10-round Grøstl-512 198
8.2.5 Distinguishers for reduced PHOTON permutations 205
8.3 Improved Outbound Part **205**
8.3.1 Multiple limited-birthday and generic complexity 206
8.3.2 Truncated characteristic with relaxed conditions 210
8.3.3 Applications 212

In this chapter, we revisit the rebound attack strategy. The original technique has been published at FSE 2009 by Florian Mendel, Christian Rechberger, Martin Schläffer and Søren S. Thomsen in [MRST09]. This technique has been widely used as a very efficient cryptanalytic technique during the SHA-3 competition, and it has now entered the collection of classical attacks designers need to consider when evaluating the security of a new design.

The main motivation of this chapter is to estimate the limitations of the rebound technique, and to push its capacity as its maximum. Before the work done here, it was for instance an open problem whether we could apply the rebound technique to 9 rounds of an AES-like permutation, as the most efficient works only reached 8 rounds [GP10, SLW+10]. In Section 8.2 of this chapter, we show how to extend the inbound phase by one round to control a total of three rounds in the middle of a truncated differential characteristic, which makes the complete attack to reach 9 rounds. This work has been done in collaboration with María Naya-Plasencia and Thomas Peyrin, and has been published at the FSE 2012 conference [JNPP12]. It has been

solicited and accepted for publication in the Journal of Cryptology [JNPP13a]. The process of publication is still pending.

While Section 8.2 considers an improvement over the inbound part of the rebound strategy, Section 8.3 tackles the probabilistic outbound phase. We show how to increase the probability of the probabilistic filter so that the overall time complexity of the rebound attacks can be decreased. In return, the generic case serving as a comparison criterion needs to be changed as well, and the generic complexity also decreases. This work has also been done in collaboration with María Naya-Plasencia and Thomas Peyrin, and is currently in submission to a conference.

In all the subsequent sections of this chapter, we consider the generalized framework of AES-like permutations already presented in Section 4.3. Before introducing our improvements of the rebound technique, we start by giving the specifications of some AES-like primitives that are cryptanalyzed in this chapter.

8.1 Description of some AES-like primitives

8.1.1 Description of Grøstl

The hash function Grøstl-0 has been submitted to the SHA-3 competition under two different versions: Grøstl-0-256, which outputs a 256-bit digest and Grøstl-0-512 with a 512-bit one. For the final round of the competition, the candidate has been tweaked to Grøstl, with corresponding versions Grøstl-256 and Grøstl-512.

The Grøstl hash function handles messages by dividing them into blocks after some padding and uses them to update iteratively an internal state (initialized to a predefined IV) with a compression function. Messages are of maximal bit-length $2n \cdot (2^{64} - 1) - 64 - 1$ for Grøstl-n, with $n \in \{256, 512\}$. This compression function is itself built upon two different permutations, namely P and Q. Each of those two permutations reuses the well-understood wide-trail strategy of the AES. As an AES-like Substitution-Permutation Network, Grøstl enjoys a strong diffusion in each of the two permutations, and by its wide-pipe design, the size of the internal state is ensured to be at least twice as large as the final digest.

The compression function f_{256} of Grøstl-256 uses two 256-bit permutations P_{256} and Q_{256}, which are similar to the two 512-bit permutations P_{512} and Q_{512} used in the compression function f_{512} of Grøstl-512. More precisely, for a chaining value h and a message block m, the compression functions (Figure 8.1) produces the output:

$$f_{256}(h, m) = P_{256}(h \oplus m) \oplus Q_{256}(m) \oplus h, \tag{8.1}$$

$$\text{or:} \quad f_{512}(h, m) = P_{512}(h \oplus m) \oplus Q_{512}(m) \oplus h. \tag{8.2}$$

The internal states are viewed as matrices of bytes of size 8×8 for the 256-bit version and 8×16 for the 512-bit one. The permutations strictly follow the design of the AES (see Section 4.2 for the full specifications) and are constructed as N_r iterations of the composition of four basic transformations:

$$R \stackrel{\text{def}}{:=} \textbf{MixCells} \circ \textbf{ShiftBytes} \circ \textbf{SubBytes} \circ \textbf{AddRoundConstant}. \tag{8.3}$$

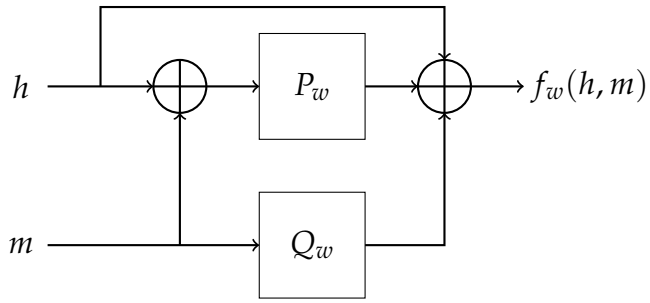


Figure 8.1: The compression function of Grøstl using the permutations P_w and Q_w, with $w \in \{256, 512\}$.

All the linear operations are performed in the same finite field $GF(2^8)$ as in the AES, defined via the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ over $GF(2)$. The **AddRoundConstant** (AC) operation adds a predefined round-dependent constant, which significantly differs between P and Q to prevent the internal differential attack [Pey10] that takes advantage of the similarities between P and Q. The **SubBytes** (SB) layer is the non-linear layer of the round function R and applies the same S-Box as in the AES to all the cells of the internal state. The **ShiftBytes** (Sh) transformation shifts cells in row i by $\tau_P[i]$ positions to the left for permutation P and $\tau_Q[i]$ positions for permutation Q. We note that τ also differs from P to Q to emphasize the asymmetry between the two permutations. Finally, **MixCells** (MC) is implemented in Grøstl by the **MixBytes** (Mb) operation that applies the circulant MDS constant matrix $\mathbf{M}$

$$\mathbf{M} = \text{circ}(2,2,3,4,5,3,5,7) = \begin{pmatrix} 2 & 2 & 3 & 4 & 5 & 3 & 5 & 7 \\ 7 & 2 & 2 & 3 & 4 & 5 & 3 & 5 \\ 5 & 7 & 2 & 2 & 3 & 4 & 5 & 3 \\ 3 & 5 & 7 & 2 & 2 & 3 & 4 & 5 \\ 5 & 3 & 5 & 7 & 2 & 2 & 3 & 4 \\ 4 & 5 & 3 & 5 & 7 & 2 & 2 & 3 \\ 3 & 4 & 5 & 3 & 5 & 7 & 2 & 2 \\ 2 & 3 & 4 & 5 & 3 & 5 & 7 & 2 \end{pmatrix}$$

independently to all the columns of the state. In Grøstl-256, there are $N_r = 10$ rounds, and

$$\tau_P = [0,1,2,3,4,5,6,7]$$
$$\text{and} \quad \tau_Q = [1,3,5,7,0,2,4,6],$$

whereas for Grøstl-512, there are $N_r = 14$ rounds and:

$$\tau_P = [0,1,2,3,4,5,6,11]$$
$$\text{and} \quad \tau_Q = [1,3,5,11,0,2,4,6].$$

Once all the message blocks of the padded input message have been processed by the compression function, a final output transformation is applied to the last chaining value h to produce the final n-bit hash value

$$h' = \text{trunc}_n(P(h) \oplus h),$$

where trunc_n only keeps the last n bits.

Compression function distinguisher

In this chapter, we mostly consider distinguishers of the inner permutations of particular symmetric primitives, however in the case of Grøstl, it is also interesting to look at not only P and Q, but also at the compression function f itself. For that matter, we can generate compression function input values (h, m) such that $\Delta_{IN} = m \oplus h$ belongs to a subset of size IN, and such that $\Delta_{IN} \oplus \Delta_{OUT} = f(h, m) \oplus f(m, h) \oplus h \oplus m$ belongs to a subset of size OUT. Then, one can remark that:

$$f(h, m) \oplus f(m, h) = P_{256}(h \oplus m) \oplus Q_{256}(m) \oplus P_{256}(m \oplus h) \oplus Q_{256}(h) \oplus h \oplus m, \quad (8.4)$$

$$f(h, m) \oplus f(m, h) = Q_{256}(m) \oplus Q_{256}(h) \oplus h \oplus m. \quad (8.5)$$

Hence, it follows that:

$$f(h, m) \oplus f(m, h) \oplus h \oplus m = Q_{256}(m) \oplus Q_{256}(h). \quad (8.6)$$

Since the permutation Q is supposed to have no structural flaw, the best known generic algorithm requires

$$\max\{\min\{\sqrt{2^n/IN}, \sqrt{2^n/OUT}\}, 2^n/(IN \cdot OUT)\}$$

operations to find a pair (h, m) of inputs such that $h \oplus m \in IN$ and $f(h, m) \oplus f(m, h) \oplus h \oplus m \in OUT$. The situation is exactly the same as the permutation distinguisher with permutation Q. Note that both IN and OUT are specific to our attacks.

We emphasize that even if trivial distinguishers are already known for the Grøstl compression function (e.g. , fixed-points), no distinguisher is known for the internal permutations. Moreover, our observations on the compression function use the differential properties of the internal permutations.

8.1.2 Description of PHOTON

PHOTON is a lightweight hash function family that is composed of an AES-like permutation in a sponge function mode as a domain extension algorithm (see Figure 8.2). It has been designed by Jian Guo, Thomas Peyrin and Axel Poschmann and has been presented at Crypto 2011 in [GPP11]. The security proof of sponge functions being directly based on the security of the internal permutation, it is important to study distinguishers for this component. Five distinct functions exist in the PHOTON family, all performing 12 rounds of an AES-like permutation, and having the set of parameters described in Table 8.1.

The nibble-sized cells for $c = 4$ uses the S-Box from PRESENT (see Appendix A.2), while the S-Box for the case $c = 8$ is the one from the AES. The diffusion layer is implemented by an MDS matrix like in the AES, but the coefficients of the matrix are not the same. Namely, the designers of PHOTON have investigated a new way to construct MDS matrices, which allow more efficient evaluations in hardware. In detail, they select the matrix as the t-th power of a

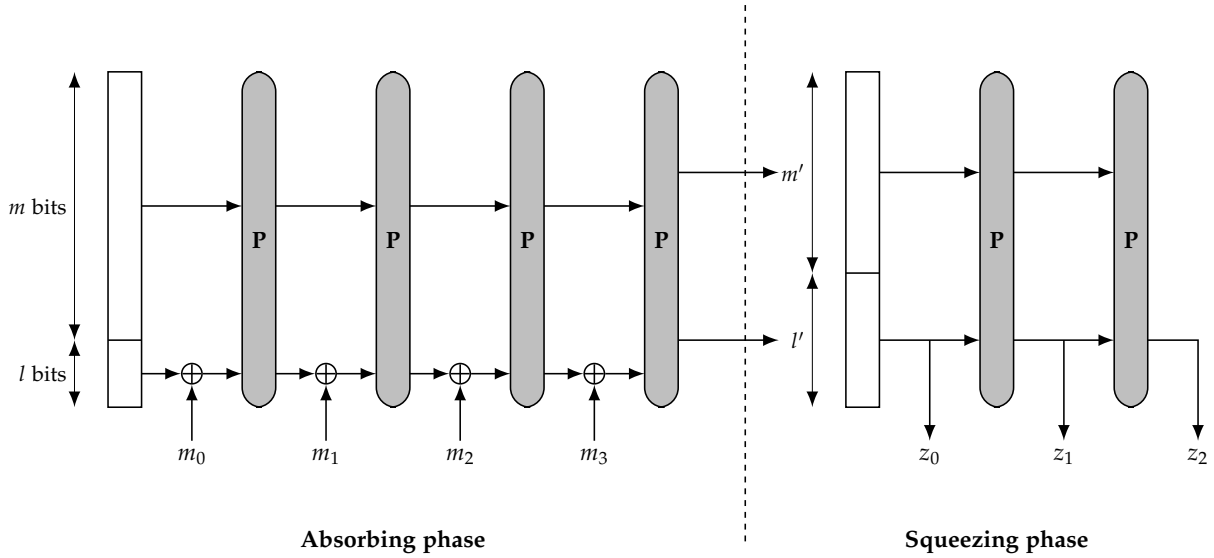


Figure 8.2: The sponge construction.

		Sponge				Perm.	
Name	h	l	m	l'	m'	t	c
PHOTON-80/20/16	80	80	20	84	16	5	4
PHOTON-128/16/16	128	128	16	128	16	6	4
PHOTON-160/36/36	160	160	36	160	36	7	4
PHOTON-224/32/32	224	224	32	224	32	8	4
PHOTON-256/32/32	256	256	32	256	32	6	8

Table 8.1: Variants of the PHOTON hash function family. For one variant, h denotes the output size in bits, l and m parameterizes the sponge absorbing phase, l' and m' the squeezing phase, and (t, c) are the two parameters of the AES-like permutation.

companion matrix:

$$\mathbf{A}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & \dots & 0 & c_0 \\ 1 & 0 & \dots & 0 & c_1 \\ 0 & 1 & \dots & 0 & c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & c_{t-1} \end{bmatrix},$$

by choosing the appropriate t coefficients $(c_0, \dots, c_{t-1})$ such that $\mathbf{A}^{\mathrm{T}}$ is MDS.

8.1.3 Description of LED

The LED block cipher has been designed by Jian Guo, Thomas Peyrin, Axel Poschmann and Matt Robshaw and has been first published at CHES 2011 in [GPPR11], and updated later in [GPPR12]. It is a lightweight block cipher based on an AES-like permutation with parameters

$t = 4$ and $c = 4$ for all its versions and comes in two versions which handle either 64- or 128-bit keys. A notable difference in comparison to the AES is its S-Box: it uses the one from PRESENT (see Appendix A.2). Moreover, the authors reused the diffusion layer design of the PHOTON hash function family.

The 64-bit version LED-64 is composed of 32 rounds, divided into 8 steps of 4 rounds each. Between two steps, a 64-bit secret key K is added to the internal state, without key schedule (see Figure 8.3).

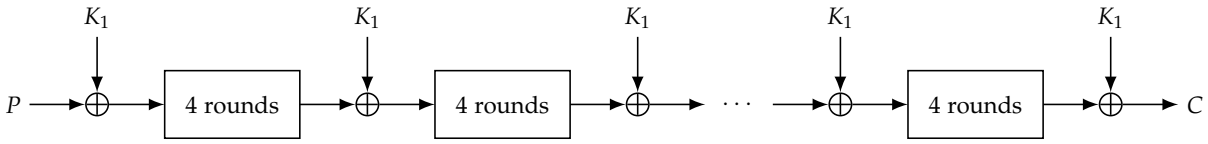


Figure 8.3: The LED-64 block cipher. There is only one 64-bit key denoted K_1.

One particularity of the LED block cipher is the absence of key scheduling algorithm. This makes the overall performance to increase a lot in comparison to the AES, but in return, it needs to have a lot more rounds to achieve enough security. The 64-bit version of the cipher adds the master key K_1 between each of the steps, and the 128-bit version has two 64-bit keys K_1 and K_2, which are introduced alternatively (see Figure 8.4).

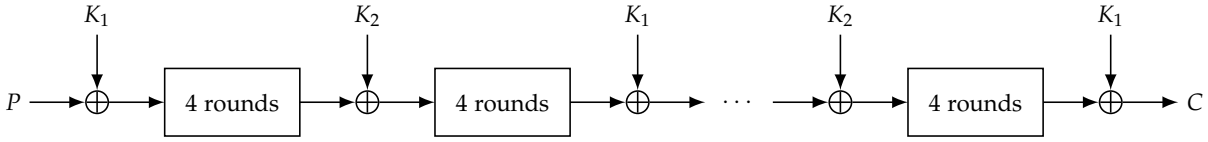


Figure 8.4: The LED-128 block cipher. There are two 64-bit keys denoted K_1 and K_2.

In total, the LED-64 internal permutation contains 32 rounds and the LED-128 internal permutation contains 48 rounds. To differentiate each round, constants are added with the **AddRoundKey** operation of the AES-like permutation: the constant added at round i is of the form:

$$\begin{bmatrix} \alpha_0 & \beta_{0,i} & 0 & 0 \\ \alpha_1 & \beta_{1,i} & 0 & 0 \\ \alpha_2 & \beta_{2,i} & 0 & 0 \\ \alpha_3 & \beta_{3,i} & 0 & 0 \end{bmatrix},$$

where $\beta_{0,i}, \ldots, \beta_{3,i}$ depend on the round i, which is not the case for the constants $\alpha_0, \ldots, \alpha_3$.

8.1.4 Description of Whirlpool

Whirlpool [BR00, BR11] is a 512-bit hash function whose compression function is built upon a block cipher E in a Miyaguchi-Preneel mode:

$$h(H, M) = E_H(M) \oplus M \oplus H.$$

This block cipher E uses two 10-round AES-like permutations with parameters $t = 8$ and $c = 8$, one for the internal state transformation and one for the key schedule. The first

permutation is fixed and takes as input the 512-bit incoming chaining variable, while the second permutation takes as input the 512-bit message block, and it round keys are the successive internal states of the first permutation.

We note that the diffusion layer in `Whirlpool` operates on the rows for the **MixColumns** layer that applies the MDS matrix, and on the columns for the **ShiftRows** operation. Even if the terms **MixColumns** and **ShiftRows** do not reflect the actual operations, we keep their names for the sake consistency with all the other notations used in this document. See the round function of Figure 8.5 for the three main operations.

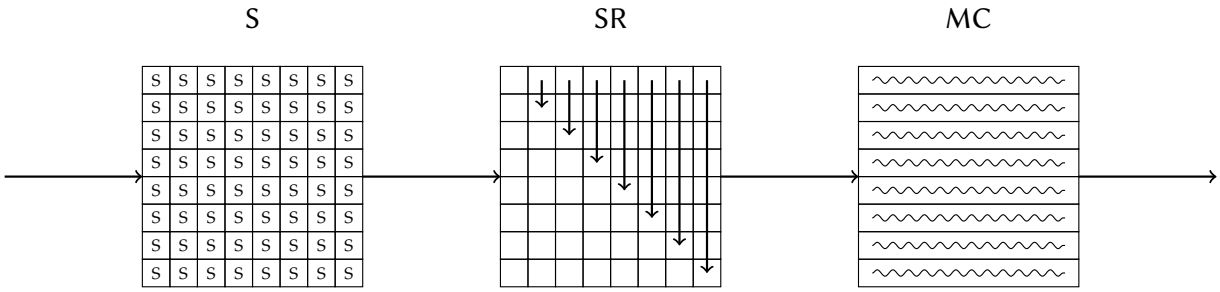


Figure 8.5: One round of the `Whirlpool` hash function inner permutation.

The non-linear S-Box of `Whirlpool` is reported as a lookup table in Appendix A.3. As for the linear MDS layer, the square matrix used is a circulant matrix as in the AES, but of size 8×8 defined by:

$$\mathbf{M} = \text{circ}(1,1,4,1,8,5,2,9) = \begin{pmatrix} 1 & 1 & 4 & 1 & 8 & 5 & 2 & 9 \\ 9 & 1 & 1 & 4 & 1 & 8 & 5 & 2 \\ 2 & 9 & 1 & 1 & 4 & 1 & 8 & 5 \\ 5 & 2 & 9 & 1 & 1 & 4 & 1 & 8 \\ 8 & 5 & 2 & 9 & 1 & 1 & 4 & 1 \\ 1 & 8 & 5 & 2 & 9 & 1 & 1 & 4 \\ 4 & 1 & 8 & 5 & 2 & 9 & 1 & 1 \\ 1 & 4 & 1 & 8 & 5 & 2 & 9 & 1 \end{pmatrix}.$$

We now describe the improvements we have found on the inbound phase on AES-like primitives.

8.2 Improved Inbound Part

AES-based functions and permutations have attracted of a lot of analysis in the recent years, mainly due to the SHA-3 hash function competition. In particular, the rebound attack (see Section 3.5.2) allows to break several proposals and many improvements/variants of this method have been published (Chapter 7).

Yet, it remained an open problem whether it would be possible to reach one more round with this type of technique compared to the state-of-the-art. We summarize the results in this research

field in the following Table 8.2, where we report the major works done in cryptanalyzing AES-based primitives with rebound-like arguments. Then, in the next two sections, we consider the problem of extending the inbound phase by one round, to tackle truncated differential characteristics of 9 rounds: Section 8.2.1 considers fully active characteristics in the three middle rounds, whereas Section 8.2.2 generalizes the case by adding parameters to capture non-fully-active characteristics, as Sasaki et al. have done in [SLW+10]. In both cases, we show an algorithm to "efficiently" generate pairs conforming to the middle rounds, which results in distinguishing algorithms under certain conditions for some AES-based primitives. As applications, we have focused on the two variants of the `Grøstl` hash functions, and present distinguishers for their reduced variants.

Target	Rounds	Time	Memory	Ideal	Reference
`Grøstl-256`	8 (dist.)	2^{112}	2^{64}	2^{384}	[GP10]
	8 (dist.)	2^{48}	2^{8}	2^{96}	[SLW+10]
	9 (dist.)	$\mathbf{2^{368}}$	$\mathbf{2^{64}}$	$\mathbf{2^{384}}$	Section 8.2.1, [JNPP13a]
	10 (zero-sum)	2^{509}	–	2^{512}	[BCD11]
`Grøstl-512`	7 (dist.)	2^{152}	2^{56}	2^{512}	[SLW+10]
	8 (dist.)	$\mathbf{2^{280}}$	$\mathbf{2^{64}}$	$\mathbf{2^{448}}$	Section 8.2.4, [JNPP13a]
	9 (dist.)	$\mathbf{2^{328}}$	$\mathbf{2^{64}}$	$\mathbf{2^{384}}$	Section 8.2.4, [JNPP13a]
	10 (dist.)	$\mathbf{2^{392}}$	$\mathbf{2^{64}}$	$\mathbf{2^{448}}$	Section 8.2.4, [JNPP13a]
PHOTON-224/32/32	8 (dist.)	2^{8}	2^{4}	2^{10}	[GPP11]
	9 (dist.)	$\mathbf{2^{184}}$	$\mathbf{2^{32}}$	$\mathbf{2^{192}}$	Section 8.2.5, [JNPP13a]

Table 8.2: Best attacks on the inner permutation of some targets where our analysis is applicable. By best analysis, we mean the ones on the highest number of rounds.

8.2.1 Fully-active truncated differential characteristic

In this section, we describe a distinguisher for 9 rounds of an AES-like permutation with certain parameters t and c. For the sake of clarity, we first describe the attack for a truncated differential characteristic with three fully active states in the middle, but we generalize our method in the next Section 8.2.2 by introducing a characteristic parameterized by variables controlling the number of active cells in some particular states.

8.2.1.1 The truncated differential characteristic

The truncated differential characteristic we use has the sequence of active cells

$$t \xrightarrow{R_1} 1 \xrightarrow{R_2} t \xrightarrow{R_3} t^2 \xrightarrow{R_4} t^2 \xrightarrow{R_5} t^2 \xrightarrow{R_6} t \xrightarrow{R_7} 1 \xrightarrow{R_8} t \xrightarrow{R_9} t^2, \tag{8.7}$$

where the size of the input and output difference subsets are both $IN = OUT = 2^{ct}$, since there are t active c-bit cells in the input of the truncated characteristic, and the t^2 active cells in the output are linearly generated from only t active cells. The actual truncated characteristic instantiated with $t = 8$ is described in Figure 8.6.

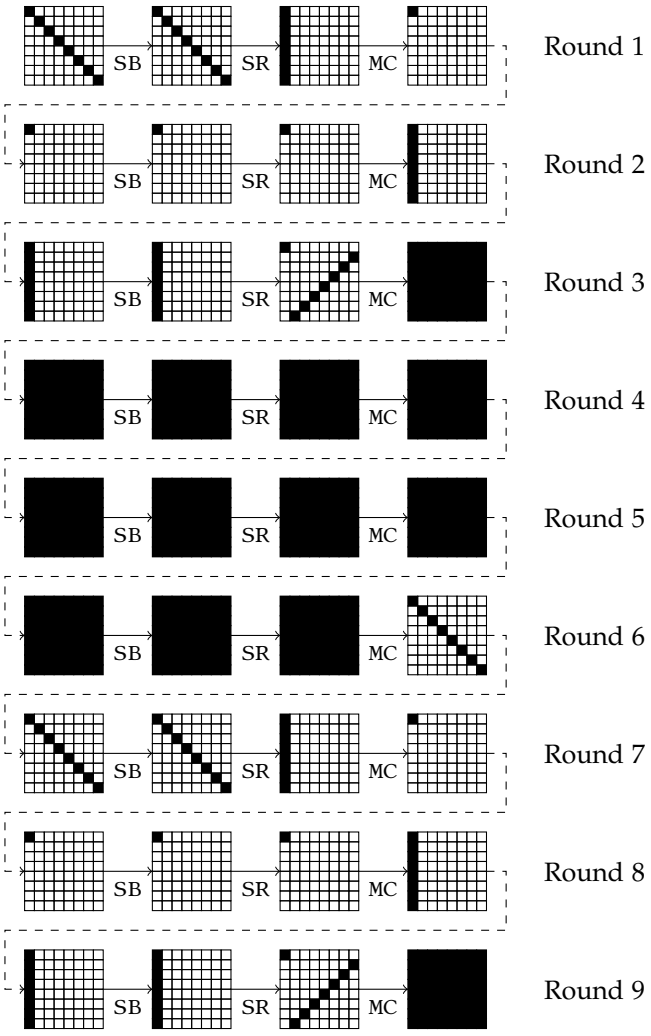


Figure 8.6: The 9-round truncated differential characteristic used to distinguish an AES-like permutation from an ideal permutation.

Note that there are three fully active internal states in the middle of the differential characteristic, and this kind of path is impossible to solve with previous rebound or **Super-SBox** techniques. The number of controlled rounds would indeed be too small and the cost for the uncontrolled part would be extremely high.

8.2.1.2 Finding a conforming pair

The method to find a pair of inputs conforming to this truncated differential characteristic is similar to the rebound technique: we first find many solutions for the middle rounds (from the beginning of round 3 to the end of round 6) and then we filter them out during the outwards probabilistic transitions through the **MixColumns** layers (round 2 and round 7). Since in our case we have two **MixColumns** transitions $t \to 1$ (see Figure 8.6), the outbound phase has a success probability of $2^{-2c(t-1)}$ and is straightforward to handle once we found enough solutions for the inbound phase.

In order to find solutions for the middle rounds (see Figure 8.7), we propose an algorithm inspired by the ones by María Naya-Plasencia in [NP10b, NP11]. As in [LMR+09, GP10], instead of dealing with the classical t^2 parallel c-bit **SubBytes** S-Box applications, one can consider the t parallel tc-bit S-Boxes (named **Super-SBoxes**) each composed of two S-Box layers surrounding one **MixColumns** and one **AddRoundConstant** function. The part of the internal state modified by one **Super-SBox** is called a **Super-SBox** set. The total state is formed by t such sets, and their particularity is that their transformation through the **Super-SBox** can be computed independently.

We start by choosing the input difference δ_{IN} after the first **SubBytes** layer at round 3 in state S1 and the output difference δ_{OUT} after the last **MixColumns** layer at round 6 in state S12. Both δ_{IN} and δ_{OUT} are exact differences, not truncated ones, but they are chosen so that they are compliant with the truncated characteristic in S0 and S12. Since we have t active cells in S1 and S12, there are as many as $(2^c-1)^{2t} \approx 2^{2ct}$ different ways of choosing $(\delta_{IN}, \delta_{OUT})$. Note that differences in S1 can be directly propagated to δ'_{IN} in S3 since **MixColumns** is linear. We continue by computing the t forward **Super-SBox** sets independently by considering the 2^{ct} possible input values for each of them in state S3. This generates t independent lists, each of size 2^{ct} and composed by paired values in S3 that can be used to compute the corresponding paired values in S8. Doing the same for the t backwards **Super-SBox** sets from state S12, we again get t independent lists of 2^{ct} elements each, and we can compute for each element of each list the pair of values of the **Super-SBox** set in state S8, where the t forward and the t backward lists overlap. In the sequel, we denote L_i the ith forward **Super-SBox** list and L'_i the ith backward one, for $1 \leq i \leq t$.

In terms of freedom degrees in state S8, we want to merge $2t$ lists of 2^{ct} elements each for a merging condition on $2 \times ct^2$ bits, where we use the definition of list merging from [NP11], on ct^2 for values and ct^2 for differences since the merging state is fully active. We then expect $2^{2t \times ct}\, 2^{-2ct^2} = 1$ solution on average as a result of the merging process.

In the following, we describe a method to find this solution and we compute its complexity afterwards. In comparison to the algorithm suggested in [JNPP12] where the case $t = 8$ is treated, we generalize the concept to any t, especially odd ones where the direct extension of [JNPP12] is not applicable. To detail this algorithm, we use an auxiliary parameter $t' \in [1, t]$ such that the time complexity will be written in terms of t'. In the end, we give the best choice for t' such that the time complexity is minimal for any t.

Step 1. We start by considering every possible combination of elements in each of the first t' lists $L'_1, \ldots, L'_{t'}$. There are $2^{c \cdot t \cdot t'}$ possibilities (see Figure 8.8).

Step 2. Each choice in Step 1 fixes the first t' columns of the internal state (both values and differences) and thus forces $2c$ constraints on t' cells in each of the t lists L_i, $1 \leq i \leq t$. For each list L_i, we then expect on average $2^{ct}\, 2^{-2ct'} = 2^{c(t-2t')}$ elements to match this constraint for each choice in Step 1, and these elements can be found with one operation by sorting the lists L_i beforehand[1].

Step 3. We continue by considering every possible combination of elements in each of the $t - t'$ last lists $L_{t-t'+1}, \ldots, L_t$. Depending on the value of t', we may have different scenarios at this point: if $t - 2t' \geq 0$, then the time complexity is multiplied by $2^{c(t-2t')(t-t')}$, which

[1] We consider lists for the sake of clarity, but we can reach the constant-time access of elements using hash tables. Otherwise, it would introduce a logarithmic factor.

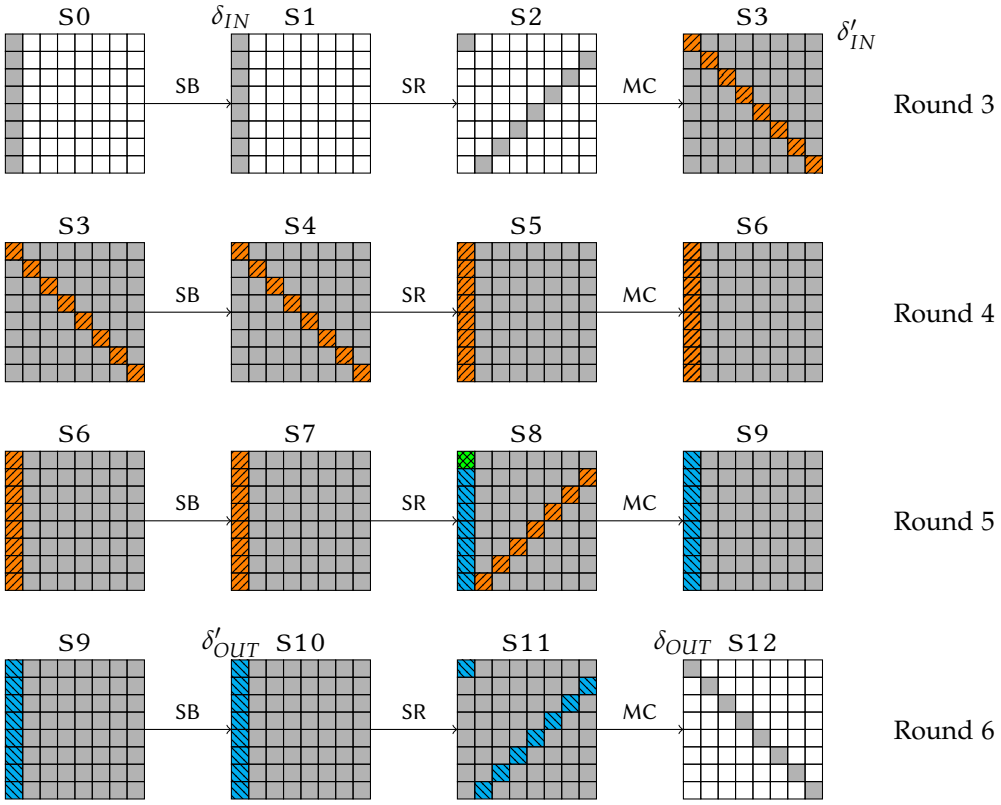


Figure 8.7: Inbound phase for the 9-round distinguishing attack on an AES-like permutation instantiated with $t = 8$. The four rounds represented are the rounds 3 to 6 from the whole truncated differential characteristic. A gray cell indicates an active cell; hatched (or colored) cells emphasize one **Super-SBox** set: there are seven similar others for each one of the two hatched senses.

is the number of expected elements in the lists. Otherwise, the probability of success decreases from 1 to $2^{c(t-2t')(t-t')}$, as the constraints imposed by the previous step are too strong and elements in those lists would exist only with probability smaller than 1.

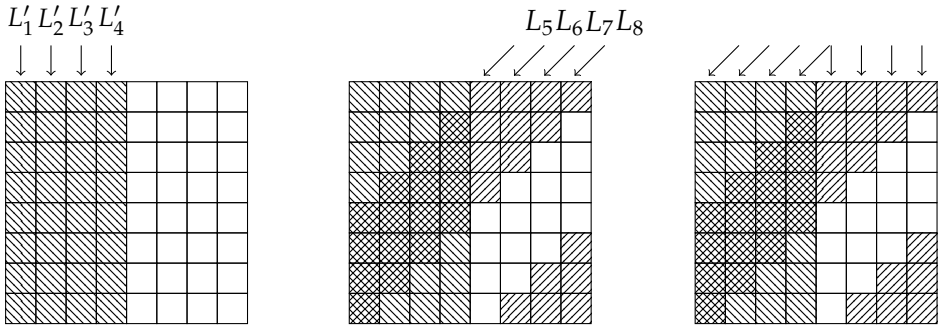
Step 4. We now need to ensure that the first t' lists $L_1, \ldots, L_{t'}$ and the $t - t'$ last lists $L'_{t-t'+1}, \ldots, L'_t$ contain a candidate fulfilling the constraints deduced in the previous steps. In the L'_i lists, we already determined $2c(t - t')$ bits so that there are $2^{ct-2c(t-t')}$ elements remaining in each of those. Again, we can check if these elements exist with one operation by sorting the lists beforehand. Finally, the value and difference of all the cells have been determined, which leads to a probability $2^{ct-2ct} = 2^{-ct}$ of finding a valid element in each of the first t' lists L_i.

All in all, trying all the $2^{c \cdot t \cdot t'}$ elements in Step 1, we find

$$2^{c \cdot t \cdot t' + c(t-2t')(t-t') + (ct-2c(t-t'))(t-t') - ct \cdot t'} = 1$$

solution during the merge process. We find this solution in time T_m operations, with two cases to consider. Either $t - 2t' \geq 0$, in which case we enumerate $2^{c \cdot t \cdot t'}$ elements in Step 1 followed by the enumeration of $2^{c(t-2t')}$ elements in Step 2. In that case, we have

$$\log_2(T_m) = ctt' + c(t - 2t')(t - t') = 2t'^2 - 2tt' + t^2. \tag{8.8}$$



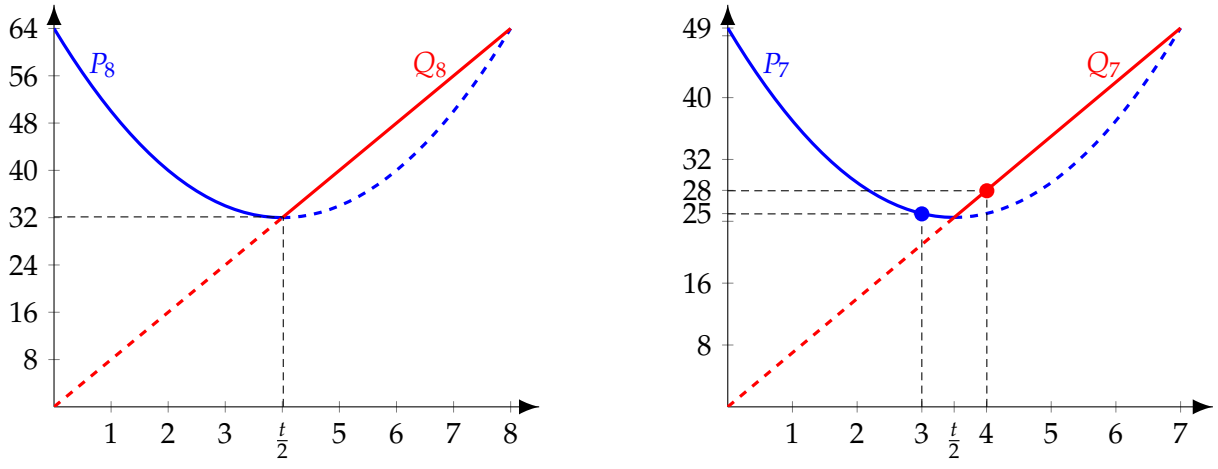
(a) Step 1. **(b)** Steps 2 and 3. **(c)** Steps 4.

Figure 8.8: In the case where $t = 8$, the figure shows the steps to merge the $2 \times t$ lists. Grey cells denote cells fully constrained by a choice of elements in $L'_1, \ldots, L'_{t/2}$ during the first step.

Otherwise, if $t - 2t' \leq 0$, the conditions imposed by the elements enumerated in the first steps make the lists from Step 2 to be nonempty with probability smaller than 1. Hence, we simply have $\log_2(T_m) = ctt'$. This can be summarized by:

$$\log_2(T_m) = \begin{cases} c \cdot P_t(t') & \text{if } t - 2t' \geq 0 \text{ with } P_t(X) = 2X^2 - 2tX + t^2, \\ c \cdot Q_t(t') & \text{if } t - 2t' \leq 0 \text{ with } Q_t(X) = tX. \end{cases} \tag{8.9}$$

To find the value t' that minimizes the time complexity, we need to determine for which value the minimum of both polynomials P_t and Q_t is reached. For P_t, we get $\frac{t}{2}$ and the nearest integer value satisfying $t - 2t' \geq 0$ is $\lceil \frac{t}{2} \rceil$. For Q_t, we get $\lfloor \frac{t}{2} \rfloor$. For example, see Figure 8.9a and Figure 8.9b, when t equals 8 and 7 respectively.



(a) $t = 8$. **(b)** $t = 7$.

Figure 8.9: Plot of the two polynomials P_t and Q_t in two cases: $t = 8$ and $t = 7$.

Consequently, if t is even we set $t' = \frac{t}{2}$, which leads to an algorithm running in $2^{ct^2/2}$ computations and $t' \cdot 2^{ct}$ memory. If t is odd, then we need to decide whether t' should be $\lfloor \frac{t}{2} \rfloor$ or $\lceil \frac{t}{2} \rceil$. If we write $t = 2k + 1$, this is equivalent to find the smallest value between $P_t(k)$ and

$Q_t(k+1)$. We find $P_t(k) = 2k^2 + 2k + 1$ and $Q_t(k+1) = 2k^2 + 3k + 1$ so that $P_t(k) < Q_t(k+1)$ (see for example Figure 8.9b when $t = 7$). Hence, when t is odd, we fix $t' = \lceil \frac{t}{2} \rceil$. Note that $\frac{t}{2} = \lceil \frac{t}{2} \rceil$ if t is even.

Summing up, for any t, our algorithm performing the merge requires T_m computations, with:

$$\log_2(T_m) = c \cdot P_t\left(\left\lceil \frac{t}{2} \right\rceil\right) = ct^2 - 2c \left\lfloor \frac{t}{2} \right\rfloor \left\lceil \frac{t}{2} \right\rceil \tag{8.10}$$

and a memory requirement of $t \cdot 2^{ct}$.

Hence, from a pair of random fixed differences $(\delta_{IN}, \delta_{OUT})$, we have shown how to find a pair of internal states of the permutation that conforms to the middle rounds. To pass the probabilistic transitions of the outbound phase, we need to repeat this merging procedure $2^{2c(t-1)}$ times by picking another couple of differences $(\delta_{IN}, \delta_{OUT})$. In total, we find a pair of inputs to the permutation that conforms to the truncated differential characteristic in time $T_9 = 2^{2c(t-1)} \cdot T_m$ computations, that is:

$$\log_2(T_9) = ct(t+2) - 2c\left(\left\lfloor \frac{t}{2} \right\rfloor \left\lceil \frac{t}{2} \right\rceil + 1\right) \tag{8.11}$$

with a memory requirement of $t \cdot 2^{ct}$.

8.2.1.3 Comparison with the ideal case

In the ideal case (Section 7.1.3), obtaining a pair whose input and output differences lie in a subset of size $IN = OUT = 2^{ct}$ for a ct^2-bit permutation requires

$$LB(t,t) = 2^{\max\left\{\frac{ct(t-1)}{2}, ct^2 - ct - ct\right\}} = 2^{ct(t-2)}, \tag{8.12}$$

computations (assuming $t \geq 3$). Therefore, our algorithm distinguishes an AES-like permutation from a random one if and only if its time complexity is smaller than the generic one. This occurs when

$$\log_2(T_9) \leq ct(t-2),$$

which happens as soon as $t \geq 8$. Note that for the AES in the known-key model, we have $t = 4$ and thus our attack does not apply.

One can also derive slightly cheaper distinguishers by aiming at fewer rounds: instead of using the 9-round truncated characteristic from Figure 8.6, it is possible to remove either round 2 or 8 and spare one $t \to 1$ truncated differential transition. Overall, the generic complexity remains the same and this gives a distinguishing attack on the 8-round reduced version of the AES-like permutation with T_8 computations, with:

$$\log_2(T_8) = \log_2(T_m) + c(t-1) = ct(t+1) - c\left(2\left\lfloor \frac{t}{2} \right\rfloor \left\lceil \frac{t}{2} \right\rceil + 1\right) \tag{8.13}$$

and still 2^{ct} memory provided that $t \geq 6$. If we spare both $t \to 1$ transitions, we end up with a 7-round distinguishing attack with time complexity $T_7 = T_m$ and $t \cdot 2^{ct}$ memory for any $t \geq 4$. Note that those reduced versions of this attack can have a greater time complexity than other techniques: we provide them only for the sake of completeness.

8.2.2 Non-fully-active truncated differential characteristic

8.2.2.1 The generic truncated characteristic

In [SLW+10], Sasaki et al. present new truncated differential characteristics that are not totally active in the middle. Their analysis allows to derive distinguishers for 8 rounds of AES-like permutations with no totally-active state in the middle, provided that the state-size verifies $t \geq 5$. In this section, we reuse their idea by introducing an additional round in the middle (see also Section 7.2.2.2), which is the unique fully active state of the characteristic. With a similar algorithm as in the previous section, we show how to find a pair conforming to that case.

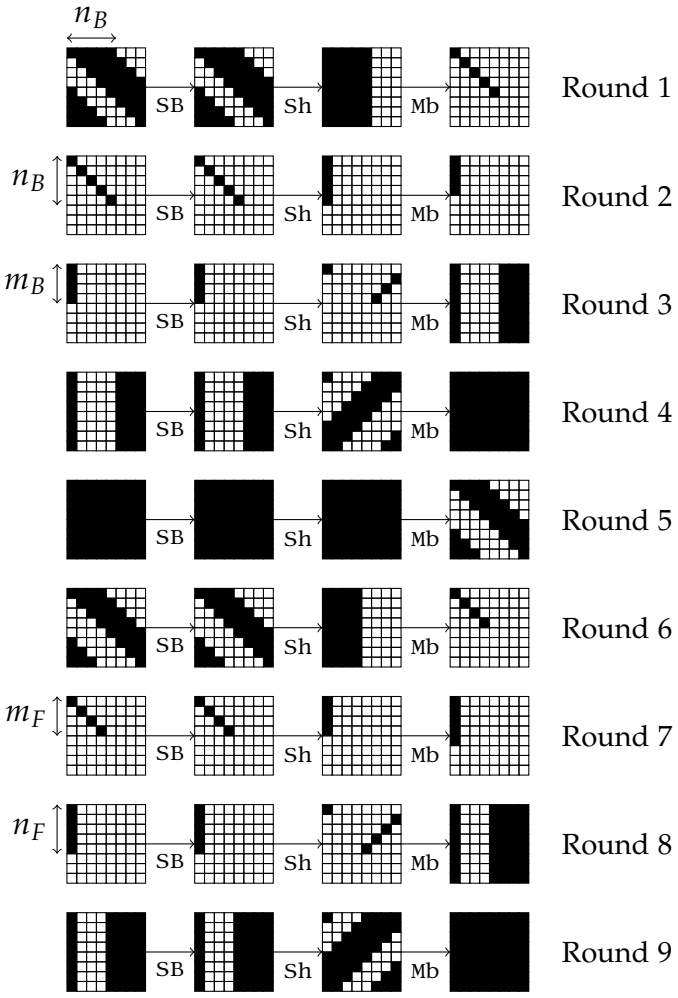


Figure 8.10: Non-fully-active truncated differential characteristic on 9 rounds of an AES-like permutation instantiated with $t = 8$.

To keep our reasoning as general as possible, we parameterize the truncated differential characteristic by four variables (see Figure 8.10) such that trade-offs will be possible by finding the right values for each one of them. Namely, we denote n_B the number of active diagonals in the plaintext (alternatively, the number of active cells in the second round), n_F the number of active diagonals in the ciphertext (alternatively, the number of active cells in the eighth round), m_B the number of active cells in the third round and m_F the number of active cells in

the seventh round.

Hence, the sequence of active cells in the truncated differential characteristic becomes:

$$t\,n_B \xrightarrow{R_1} n_B \xrightarrow{R_2} m_B \xrightarrow{R_3} t\,m_B \xrightarrow{R_4} t^2 \xrightarrow{R_5} t\,m_F \xrightarrow{R_6} m_F \xrightarrow{R_7} n_F \xrightarrow{R_8} t\,n_F \xrightarrow{R_9} t^2, \tag{8.14}$$

with the constraints $n_F + m_F \geq t+1$ and $n_B + m_B \geq t+1$ that come from the MDS property. The amount of solutions that can be generated for the differential path equals to ($\log_2$):

$$ct^2 + ctn_B - c(t-1)n_B - c(t-m_B) - ct(t-m_F) - c(t-1)m_F - c(t-n_F) \tag{8.15}$$

$$= c(n_B + n_F + m_B + m_F - 2t). \tag{8.16}$$

From the MDS constraints $m_B + n_B \geq t+1$ and $m_F + n_F \geq t+1$, we can bound the amount of expected solutions by $2^{c(t+1+t+1-2t)} = 2^{2c}$. This means that, there will always be at least 2^{2c} freedom degrees, independently of t.

8.2.2.2 Finding a conforming pair

As in the previous case, the algorithm that finds a pair of inputs conforming to this characteristic first produces many pairs for the middle rounds and then exhausts them outwards until one passes the probabilistic filter. The cost of those uncontrolled rounds is given by:

$$2^{c(t-n_B)}\,2^{c(t-n_F)} = 2^{c(2t-n_B-n_F)}, \tag{8.17}$$

since we need to pass one $n_B \leftarrow m_B$ transition in the backward direction and one $m_F \rightarrow n_F$ in the forward direction.

We now detail a way to find a solution for the middle rounds (Figure 8.11) when the input difference δ_{IN} after the first **SubBytes** layer in state S1 and the output difference δ_{OUT} after the last **MixColumns** layer in state S12 are fixed in such a way that the truncated characteristic holds in S0 and S12. The beginning of the attack is exactly the same as before in the sense that once the input and output differences of the middle rounds have been fixed, we generate the $2t$ lists that contains the paired values of the t forward **Super-SBox** sets and the t backward **Super-SBox** sets. Again, the same $2t$ lists overlap and we show how to find the solution of the merging problem in $2^{ct\cdot\min(m_F, m_B, \lceil \frac{t}{2} \rceil)}$ computations and $m_B \cdot 2^{ct}$ memory. We recall that L_i is the ith forward **Super-SBox** list (orange) and L'_i is the ith backward one (blue), for $1 \leq i \leq t$.

We proceed in three steps: the first one guesses the elements from some lists, which determines the remaining cells and we finish by checking probabilistic events. Without loss of generality, we assume in the sequel that $m_B \leq m_F$; if this is not the case, then we start Step 1 by guessing elements of lists L_i in S8. We split the analysis into two cases, depending on whether $m_B \leq \lceil \frac{t}{2} \rceil$ or $m_B > \lceil \frac{t}{2} \rceil$.

First case: $m_B \leq \lceil \frac{t}{2} \rceil$ — In this case, we use the strong constraints on the vector spaces spanned by the m_B differences on each column to find a solution to the merge problem.

Step 1. We start by guessing the elements of the m_B lists $L'_1, \ldots, L'_{m_B}$ in state S6. There is a total of 2^{ctm_B} possible combinations.

Step 2. In particular, the previous step sets the differences of the m_B first diagonals of S6 such that there are exactly m_B known differences on each of the t columns of

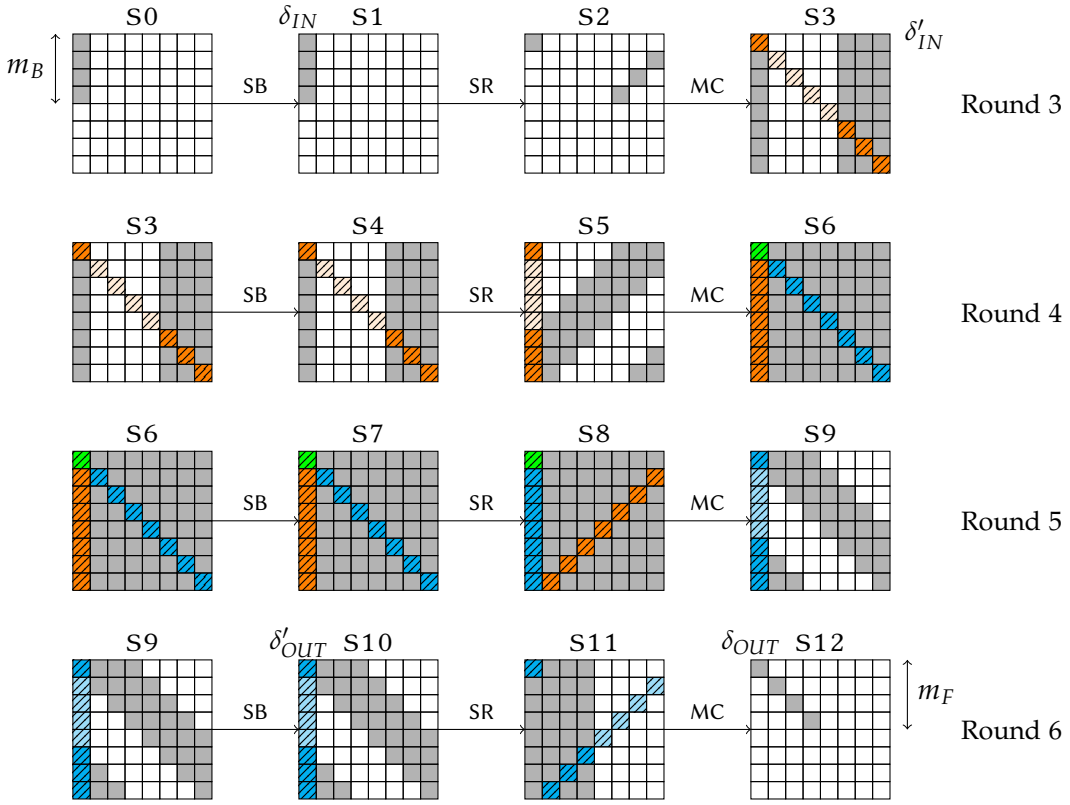


Figure 8.11: Inbound phase for the 9-round distinguisher attack on an AES-like permutation instantiated with $t = 8$ with a single fully-active state in the middle. A gray cell indicates an active cell; hatched (or colored) cells emphasize one **Super-SBox** set: there are seven similar others.

the state. This allows to determine all the differences in S5 since there are exactly m_B independent differences in each column of that state. Consequently, we linearly learn all the differences of S6.

Step 3. Since all differences are known in S6, we determine 1 element in each of the $t - m_B$ remaining L'_i lists: they are of size 2^{ct} and we count ct bits of constraints coming from t differences. From the known differences, we also get a suggestion of 2^{ct-cm_B} values for the cells of each column. Indeed, the elements of the t lists L_i in S5 can be represented as disjointed sets regarding the values of the differences, since the differences can only take 2^{cm_B} values per column. Assuming that they are uniformly distributed[2], we get $2^{ct}/2^{cm_B} = 2^{ct-cm_B}$ elements per disjointed set for each list: they all share the same value of the differences, but have different values. Additionally, the ct-bit constraints on each list L_i allows to find one element in each, and therefore a solution to the merge problem, with probability

$$2^{((ct-cm_B)-ct)t} = 2^{-ctm_B}.$$

Step 4. Finally, trying all the 2^{ctm_B} elements in $(L'_1, \ldots, L'_{m_B})$, we expect to find

$$2^{ctm_B}\, 2^{-ctm_B} = 1$$

[2]This is a classical assumption, and here it is due to the non-linear S-Box.

solution that gives a pair of internal states conforming to the four middle rounds with a few operations.

Second case: $m_B > \lceil \frac{t}{2} \rceil$ — The columns of differences are less constrained, and it is enough to guess $\lceil \frac{t}{2} \rceil$ lists in the first step to find a solution to the merge problem.

Step 1. We start by guessing the elements of the $\lceil \frac{t}{2} \rceil$ lists $L'_1, \ldots, L'_{m_B}$ in state S6. There is a total of $2^{ct\lceil \frac{t}{2} \rceil}$ possible combinations.

Step 2. The previous step allows to filter $2^{c(t-2\lceil \frac{t}{2} \rceil)}$ elements in each of the t lists L_i. Depending on the parity of t, we get 1 element per list for even t, and 2^{-c} for odd ones. Indeed, $t - 2\lceil \frac{t}{2} \rceil = \lfloor \frac{t}{2} \rfloor - \lceil \frac{t}{2} \rceil$ equals 0 when t is even, and -1 when t is odd. In the latter odd case, there are then a probability 2^{-ct} that the t elements are found in the t lists L_i.

Step 3. In the event that elements have been found in the previous step, we determine completely the remaining $2ct(t - \lceil \frac{t}{2} \rceil)$ values and differences of the remaining $t - \lceil \frac{t}{2} \rceil = \lfloor \frac{t}{2} \rfloor$ lists L'_i. We find a match in those lists with probability

$$2^{-ct} \times 2^{(ct-2ct)(t-\lceil \frac{t}{2} \rceil)} = 2^{-ct(1+\lfloor \frac{t}{2} \rfloor)}.$$

Step 4. Finally, trying all the $2^{ct\lceil \frac{t}{2} \rceil}$ elements in $(L'_1, \ldots, L'_{\lceil \frac{t}{2} \rceil})$, we expect to find

$$2^{ct\lceil \frac{t}{2} \rceil}\, 2^{-ct(1+\lfloor \frac{t}{2} \rfloor)} = 1$$

solution that gives a pair of internal states that conforms to the four middle rounds with a few operations.

Hence, in both cases, from random differences $(\delta_{IN}, \delta_{OUT})$, we find a pair of internal states of the permutation that conforms to the middle rounds in time $2^{ct \min\left(m_B, \lceil \frac{t}{2} \rceil\right)}$ and memory $m_B 2^{ct}$. To pass the probabilistic transitions of the outbound phase, we need to repeat the merging $2^{c(2t-n_B-n_F)}$ times by picking another couple of differences $(\delta_{IN}, \delta_{OUT})$. In total, we find a pair of inputs to the permutation conforming to the truncated differential characteristic in time complexity

$$2^{ct \min(m_B, \lceil \frac{t}{2} \rceil)}\, 2^{c(2t-n_B-n_F)} = 2^{c(t(\min(m_B, \lceil \frac{t}{2} \rceil)+2)-n_B-n_F)}$$

and memory complexity $m_B \cdot 2^{ct}$.

Finally, without assuming $m_B \leq m_F$, the time complexity T of the algorithm generalizes to:

$$\log_2(T) = c\left(t \cdot \min\left\{m_B, m_F, \left\lceil \frac{t}{2} \right\rceil\right\} + 2t - n_B - n_F\right), \tag{8.18}$$

with $n_F + m_F \geq t + 1$ and $n_B + m_B \geq t + 1$, and memory requirements of $m_B \cdot 2^{ct}$.

8.2.2.3 Comparison with ideal case

In the ideal case, the generic complexity $LB(a, b)$ is given by the limited birthday distinguisher that we recall here:

$$\log_{2^c}\Big(LB(a,b)\Big) = \max\left\{\min\left\{\frac{t^2-a}{2}, \frac{t^2-b}{2}\right\}, t^2 - a - b\right\}, \tag{8.19}$$

since we get an input space of size $IN = 2^{c \cdot a}$ and output space of size $OUT = 2^{c \cdot b}$. Without loss of generality, assume that $a \leq b$: this only selects whether we attack the permutation or its inverse. In that case, we recall that we can rewrite the complexity as:

$$\log_{2^c}\Big(LB(a,b)\Big) = \begin{cases} C_1(a,b) := (t^2-b)/2, & \text{if: } t^2 < 2a+b, \\ C_2(a,b) := a, & \text{if: } t^2 = 2a+b, \\ C_3(a,b) := t^2-a-b, & \text{if: } t^2 > 2a+b. \end{cases} \tag{8.20}$$

In the case of the 9-round distinguisher, the generic complexity equals $LB(t \cdot n_B, t \cdot n_F)$ since there are n_B active diagonals at the input, and n_F active diagonals at the output. Let us compare T and the case of $C_3(t \cdot n_B, t \cdot n_F)$ where $t > 2n_B + n_F$ corresponding to the limited birthday distinguisher. We want to find set of values for the parameters (t, n_F, n_B, m_F, m_B) such that our algorithm is faster that the generic one, that is T is smaller than $C_3(t \cdot n_B, t \cdot n_F)$. In the event that $\min\left(m_F, m_B, \lceil \frac{t}{2} \rceil\right)$ is either m_F or m_B, we can show that T is always greater than $C_3\left(t \cdot n_B, t \cdot n_F\right)$, and so are the cases involving $C_2\left(t \cdot n_B, t \cdot n_F\right)$ and $C_1\left(t \cdot n_B, t \cdot n_F\right)$.

We consider the case $\min\left(m_F, m_B, \lceil \frac{t}{2} \rceil\right) = \lceil \frac{t}{2} \rceil$:

$$\log_{2^c}\Big(C_3\left(t \cdot n_B, t \cdot n_F\right)\Big) - \log_{2^c}\Big(T\Big) = t\left(t - n_F - n_B\right) - t\left\lceil \frac{t}{2} \right\rceil - 2t + n_B + n_F. \tag{8.21}$$

With t as a parameter and $n_F, n_B \in \{1, \ldots, t\}$, our algorithm turns out to be a distinguisher when the quantity from (8.21) is positive, which is true as soon as:

$$(n_B + n_F)(1-t) + t\left(t - 2 - \left\lceil \frac{t}{2} \right\rceil\right) \geq 0. \tag{8.22}$$

Since $t - \lceil \frac{t}{2} \rceil = \lfloor \frac{t}{2} \rfloor$, we can show that if $n_F \in \{1, \ldots, t\}$ and $n_B \in \{1, \ldots, t\}$ are chosen such that

$$2 \leq n_F + n_B \leq \frac{t}{t-1}\left(\left\lfloor \frac{t}{2} \right\rfloor - 2\right), \tag{8.23}$$

then our algorithm is more efficient than the generic one. Note that this may happen only when $t \geq 8$ and that m_F and m_B are still constrained by the MDS bound: $n_F + m_F \geq t+1$ and $n_B + m_B \geq t+1$.

We can also consider an 8-round case by considering the characteristic from Figure 8.10 where the last round is removed[3]: the generic complexity becomes $LB(t \cdot n_B, n_F)$. Note that the complexity of our algorithm remains unchanged: there are still two probabilistic transitions to pass. For $t \geq 4$, we can show that there are many ways to set the parameters (n_F, n_B, m_F, m_B) so that $T \geq C(t \cdot n_B, n_F)$, and the best choice providing the most efficient distinguisher happens when the MDS bounds are tight, i.e.: $n_F + m_F = t+1$ and $n_B + m_B = t+1$.

For the sake of completeness, we can also derive distinguishers for 7-round of the permutation by considering the characteristic from Figure 8.10 where the first and last rounds

[3]We still assume that $n_B \leq n_F$. If not, then the generic complexity becomes $LB(n_B, t \cdot n_F)$ by removing the first round.

are removed, as soon as $t \geq 4$. The generic complexity in that scenario is $LB(n_B, n_F)$. Again, there are several ways to set the parameters, but the one that minimizes the runtime T of our algorithm also verifies the MDS bounds: $n_B = 1$, $m_B = t$, $m_F = 1$ and $n_F = t$.

We give examples of more different cases in Table 8.3, which for instance match the AES and `Grøstl` instantiations. We note that the complexities of our algorithm may be worse that other published results.

Rounds	Cipher		Parameters				Complexities	
	t	**c**	$\mathbf{n}_B$	$\mathbf{m}_B$	$\mathbf{m}_F$	$\mathbf{n}_F$	$\log_2(\mathbf{T})$	$\log_2(\mathbf{C})$
9	8	8	1	8	8	1	368	$\log_2 LB(t \cdot n_B, t \cdot n_F) = 384$
8	8	8	8	1	4	5	88	$\log_2 LB(n_B, t \cdot n_F) = 128$
8	8	8	5	4	1	8	88	$\log_2 LB(t \cdot n_B, n_F) = 128$
7	8	8	8	1	1	8	64	$\log_2 LB(n_B, n_F) = 384$
8	7	8	7	1	4	4	80	$\log_2 LB(n_B, t \cdot n_F) = 112$
8	7	8	4	4	1	7	80	$\log_2 LB(t \cdot n_B, n_F) = 112$
7	7	8	7	1	1	7	56	$\log_2 LB(n_B, n_F) = 280$
8	6	8	6	1	4	3	72	$\log_2 LB(n_B, t \cdot n_F) = 96$
8	6	8	3	4	1	6	72	$\log_2 LB(t \cdot n_B, n_F) = 96$
7	6	8	6	1	1	6	56	$\log_2 LB(n_B, n_F) = 192$
8	5	4	5	1	4	2	32	$\log_2 LB(n_B, t \cdot n_F) = 40$
8	5	4	2	4	1	5	32	$\log_2 LB(t \cdot n_B, n_F) = 40$
7	5	4	5	1	1	5	20	$\log_2 LB(n_B, t \cdot n_F) = 60$
8	4	8	4	1	4	1	56	$\log_2 LB(n_B, t \cdot n_F) = 64$
8	4	8	1	4	1	4	56	$\log_2 LB(t \cdot n_B, n_F) = 64$
7	4	8	4	1	1	4	32	$\log_2 LB(n_B, t \cdot n_F) = 64$

Table 8.3: Examples of reached time complexities of our algorithm for several numbers of rounds and different (t, c) scenarios.

8.2.3 Application to `Grøstl-256` permutations

The permutations of the `Grøstl-256` hash function implement the previous generic algorithms will the following parameters: $t = 8$, $c = 8$ and $N_r = 10$.

8.2.3.1 Three fully-active states

From the analysis of Section 8.2.1, we can directly conclude that this leads to a distinguishing attack on the 9-round reduced version of the `Grøstl-256` permutation with

$$2^{c(t^2/2+2(t-1))} = 2^{368}$$

computations and $2^{ct} = 2^{64}$ memory, when the ideal complexity requires $2^{ct(t-2)} = 2^{384}$ computations.

As detailed previously, we could derive distinguishers for 8-round Grøstl-256 with

$$2^{c(t^2/2+t-1)} = 2^{312}$$

computations and for 7-round Grøstl-256 with $2^{ct^2/2} = 2^{256}$, but those results are more costly than previous known results.

Similarly, as explained in Section 8.1.1, this result also induces a nontrivial observation on the 9-round reduced version of the Grøstl-256 compression function with identical complexities.

8.2.3.2 Non-fully-active characteristic

With the generic analysis of Section 8.2.2 that uses a single fully-active middle state, $t = 8$ only allows to instantiate the parameterized truncated differential characteristic with $n_F = n_B = 1$, which determines $m_F = m_B = 8$. Indeed, (8.23) imposes $2 \leq n_B + n_F \leq \frac{16}{7}$, which gives integer values $n_F = n_B = 1$. Note that it is exactly the case of the three fully-active states in the middle treated in Section 8.2.1, with the same complexities.

For 8-round distinguishers, the case $t = 8$ where $n_B \leq n_F$ may give the parameters $n_B = 5$, $m_B = 4$, $m_F = 1$ and $n_F = 8$ with the last round of the characteristic of Figure 8.10 is removed. If $n_B > n_F$, we instantiate the characteristic with the first round removed with the values $n_B = 8$, $m_B = 1$, $m_F = 4$ and $n_F = 5$. In both cases, the time complexity of the distinguishers are 2^{88} computations with 2^{64} of memory requirements, whereas the generic algorithm terminates in about 2^{128} operations.

As for 7-round distinguishers, removing both first and last rounds of the characteristic of Figure 8.10 leads to an efficient distinguishers for Grøstl-256 when $n_B = 8$, $m_B = 1$, $m_F = 1$ and $n_F = 8$. The corresponding algorithm runs in 2^{64} computations with 2^{64} of memory requirements, when the corresponding generic algorithm needs 2^{384} operations to terminate. We note that those 8- and 7-round distinguishers are not as efficient as other available techniques.

8.2.4 Distinguisher for 10-round Grøstl-512

The 512-bit version of the Grøstl hash function uses a non-square 8×16 matrix as 1024-bit internal state, which therefore presents a lack of optimal diffusion: a single difference generates a fully active state after three rounds where a square state only needs two. This allows to add an extra round to the generalization of the regular 9-round characteristic of AES-like permutation (Section 8.2.1) to reach 10 rounds.

8.2.4.1 The truncated differential characteristic

To distinguish the permutation P_{512} reduced to 10 rounds, we use the truncated differential characteristic with the sequence of active bytes

$$64 \xrightarrow{R_1} 8 \xrightarrow{R_2} 1 \xrightarrow{R_3} 8 \xrightarrow{R_4} 64 \xrightarrow{R_5} 128 \xrightarrow{R_6} 64 \xrightarrow{R_7} 8 \xrightarrow{R_8} 1 \xrightarrow{R_9} 8 \xrightarrow{R_{10}} 64, \tag{8.24}$$

where the size of the input differences subset is $IN = 2^{512}$ and the size of the output differences subset is $OUT = 2^{64}$. We note that it would work exactly the same way for the other permutation Q_{512}.

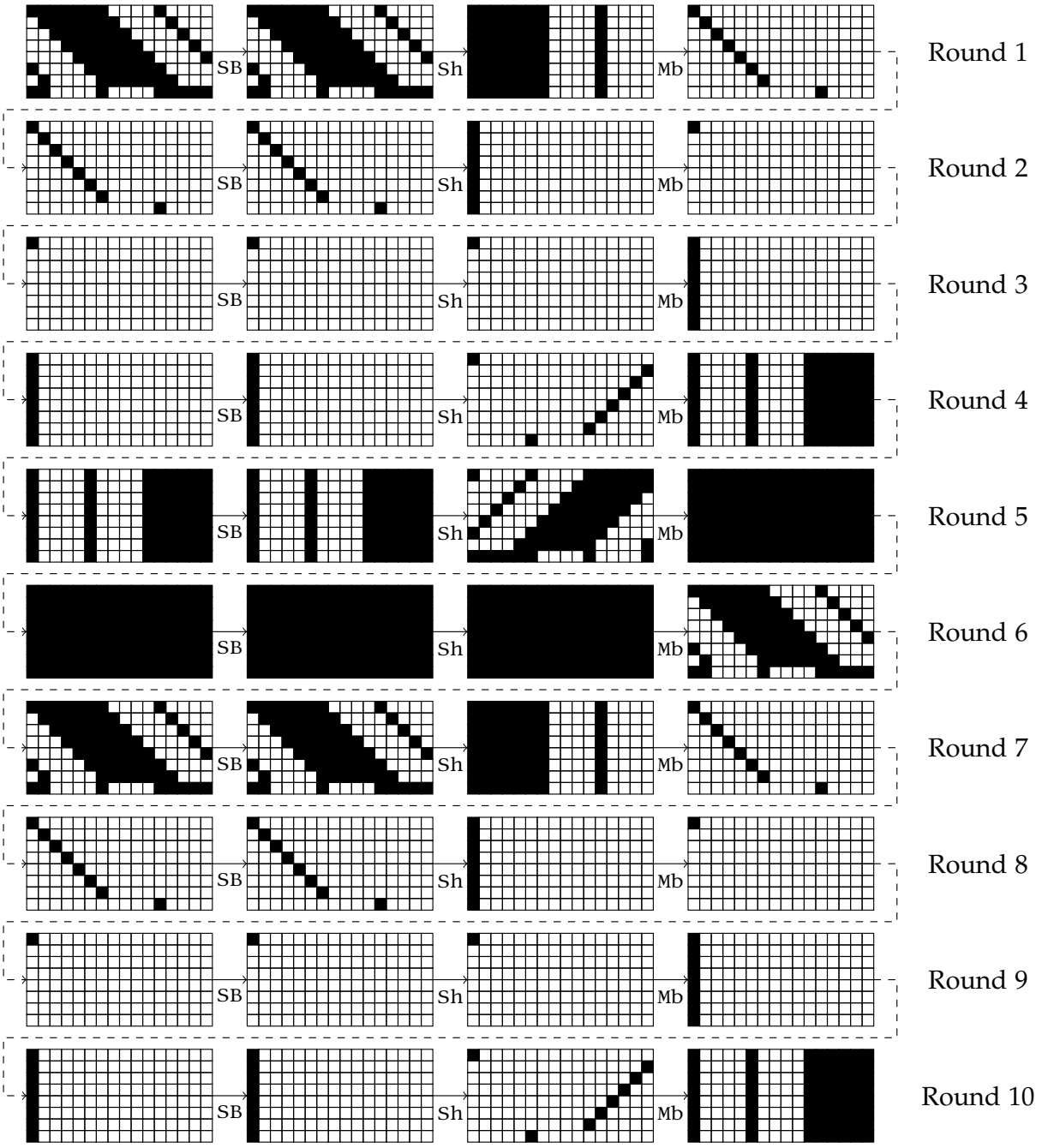


Figure 8.12: The 10-round truncated differential characteristic used to distinguish the permutation P of Grøstl-512 from an ideal permutation.

The actual truncated characteristic is represented on Figure 8.12. Again, we split the characteristic into two parts: the inbound phase involving a merging of lists in the four middle rounds (round 4 to round 7), and an outbound phase that behaves as a probabilistic filter ensuring both $8 \longrightarrow 1$ transitions in the outward directions. Again, passing those two

transitions with random values occurs with probability 2^{-112}.

8.2.4.2 Finding a conforming pair

In the following, we present an algorithm to solve the middle rounds in time 2^{280} and memory 2^{64}. In total, we need to repeat this process 2^{112} times to get a pair of internal states that conforms to the whole truncated differential characteristic, and then cost $2^{280+112} = 2^{392}$ computations and 2^{64} in memory. The strategy of this algorithm (see Figure 8.13) is similar to the one presented in [NP11, NP10b] and the one from the previous section: we start by fixing the difference to a random value δ_{IN} in S1 and δ_{OUT} in S12 and linearly deduce the difference δ'_{IN} in S3 and δ'_{OUT} in S10. Then, we construct the 32 lists corresponding to the 32 **Super-SBoxes**: the 16 forward **Super-SBoxes** have an input difference fixed to δ'_{IN} and cover states S3 to S8, whereas the 16 backward **Super-SBoxes** spread over states S10 to S6 with an output difference fixed to δ'_{OUT}. In the sequel, we denote L_i the 16 forward **Super-SBoxes** and L'_i the backward ones, $1 \leq i \leq 16$.

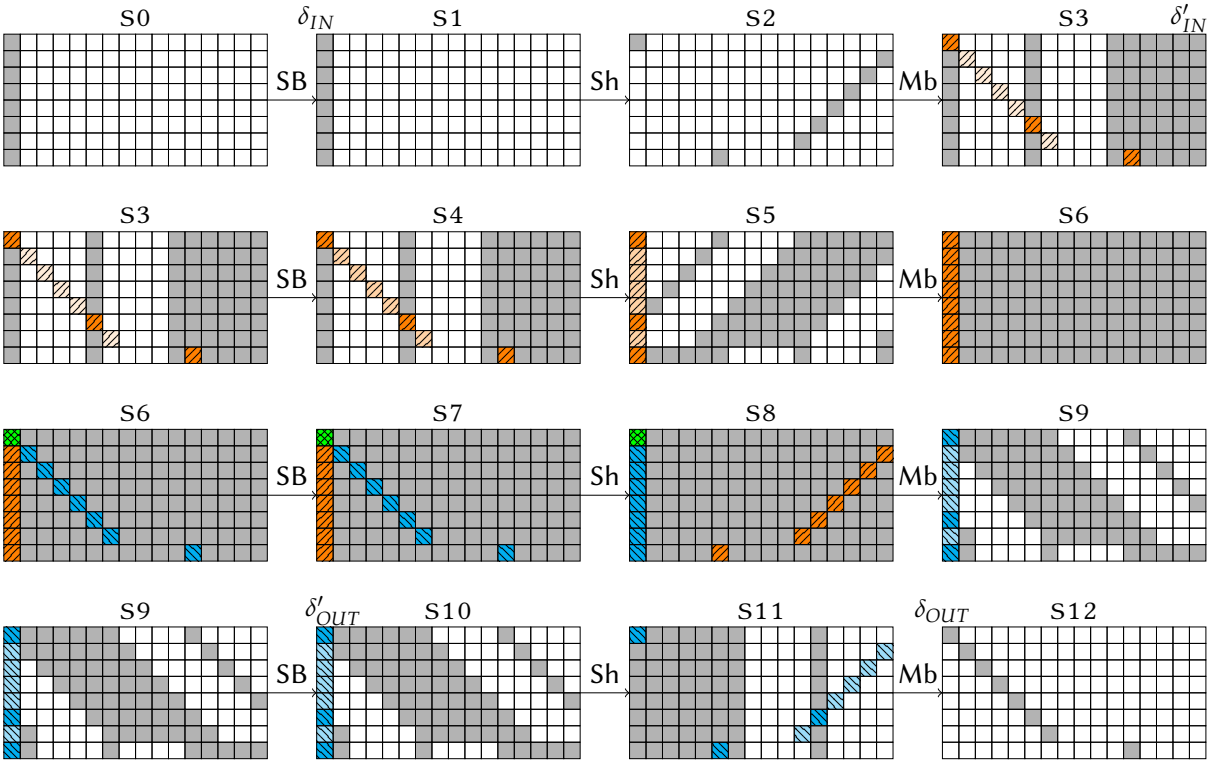


Figure 8.13: Inbound phase for the 10-round distinguisher attack on the Grøstl-512 permutation P_{512}. The four rounds represented are the rounds 4 to 7 from the whole truncated differential characteristic (Figure 8.12). A gray byte indicates an active byte; hatched (or colored) bytes emphasize the **Super-SBoxes**.

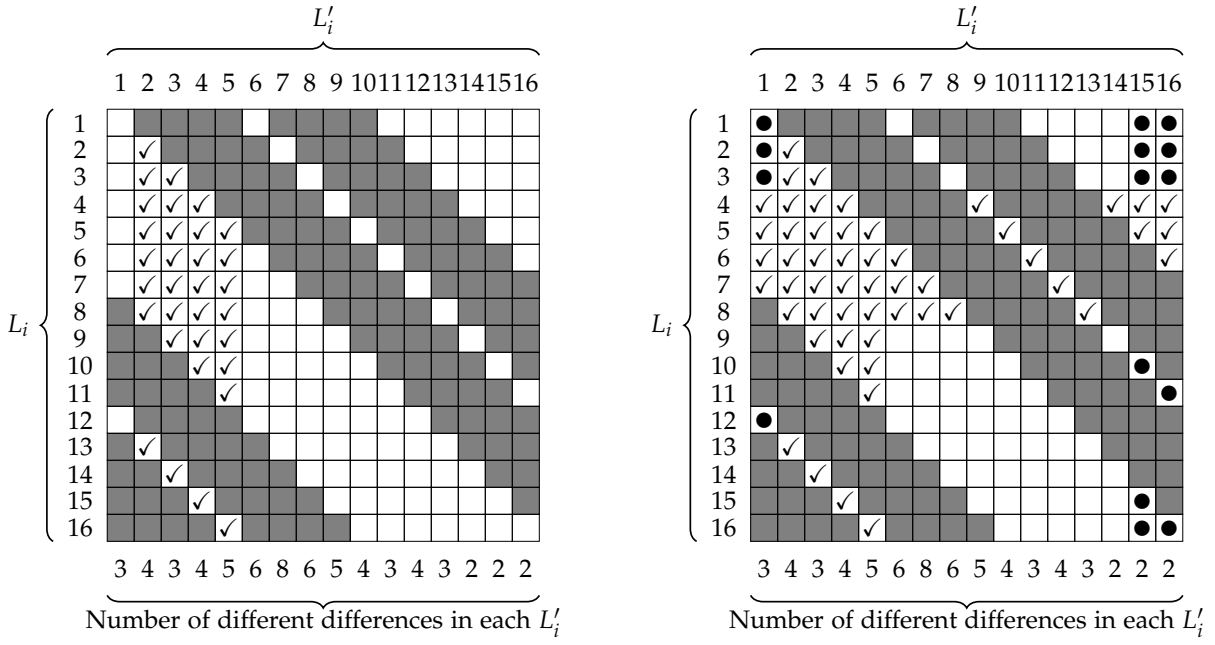
The 32 lists overlap in S8, where we merge them on 2048 bits to find $2^{64\times 32}\,2^{-2048} = 1$ solution, since each list is of size 2^{64}. The 2048 bits come from 1024 bits of values and 1024 bits of differences. The naive way to find the solution would cost 2^{1024} in time by considering each element of the Cartesian product of the 16 lists L_i to check whether it satisfies the output 1024 bit difference condition. We describe now the algorithm that achieves the same goal in 2^{280}

computations.

First, we observe that due to the geometry of the non-square state, any list L_i intersects with only half of the L'_i. For instance, the first list L_1 associated to the first column of state S7 intersects with lists L'_1, L'_6, L'_{11}, L'_{12}, L'_{13}, L'_{14}, L'_{15} and L'_{16}. We represent this property with a 16×16 array on Figure 8.14: the 16 columns correspond to the 16 lists L'_i and the lines to the L_i, $1 \leq i \leq 16$. The cell (i, j) is white if and only if L_i has a non-null intersection with the list L'_j, otherwise it is gray.

Then, we note that the **MixColumns** transition between the states S8 and S9 constraints the differences in the lists L'_i: in the first column of S9 for example, only three bytes are active, so that the same column in S8 can only have $(2^8 - 1)^3 \approx 2^{3\times 8}$ different differences, which means that knowing three out of the eight differences in an element of L'_1 is enough to deduce the other five. For a column-vector of differences lying in a n-dimensional subspace, we can divide the 2^{64} elements of the associated lists in 2^{8n} disjointed sets of 2^{64-8n} values each. So, whenever we know the n independent differences, the only freedom that remains lies in the values. The bottom line of Figure 8.14 reports the subspace dimensions for each L'_i.

Using a guess-and-determine approach, we derive a way to use the previous facts to find the solution to the merge problem in 2^{280} computations. As stated before, we expect only one solution; that is, we want to find a single element in each of the 32 lists. In the sequel, we describe a sequence of four guess-and-determine steps illustrated by pictures before and after each *determine* phase.



(a) Initial guesses.

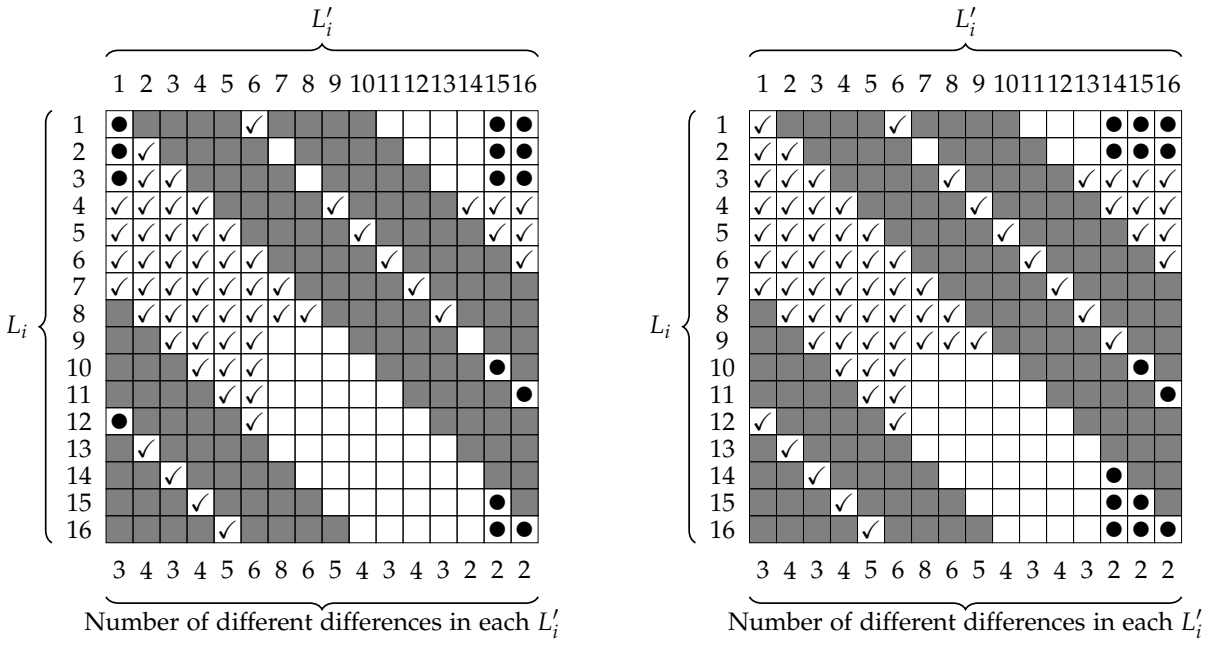
(b) End of first determine step.

Figure 8.14: First guess on the algorithm. A ✓ means we know both value and difference for that byte, a ● means that we only determined the difference for that byte and white bytes are not constrained yet.

Step 1. We start by guessing the values and the differences of the elements associated to the

lists L'_2, L'_3, L'_4 and L'_5. For this, we try all the possible combinations of their elements, there are $2^{4\times 64} = 2^{256}$ in total. For each one of the 2^{256} trials, all the checked cells ✓ from Figure 8.14a now have known value and difference. From here, 8 bytes are known in each of the four lists L_5, L_6, L_7 and L_8: this imposes a 64-bit constraint on those lists, which filter out a single element in each. Thereby, we determine the value and difference in the other 16 bytes marked by ✓ in Figure 8.14b. In lists L'_1 and L'_{16}, we have reached the maximum number of independent differences (three and two, respectively), so we can determine the differences for the other bytes of those two columns: we mark them by ●. In L_4, the 8 constraints (three ✓ and two ●) filter out one element; then, we deduce the correct element in L_4 and mark it by ✓. We can now determine the differences in L'_{15} since the corresponding subspace has a dimension equal to two. See Figure 8.14b for the current situation of the guess-and-determine algorithm.

Step 2. At this point, no more byte can be determined based on the information propagated so far. We continue by guessing the elements remaining in L'_6 (see Figure 8.15a). Since there are already six byte-constraints on that list (three ✓), only 2^{16} elements conform to the conditions. The time complexity until now is thus $2^{256+16} = 2^{272}$ computations.



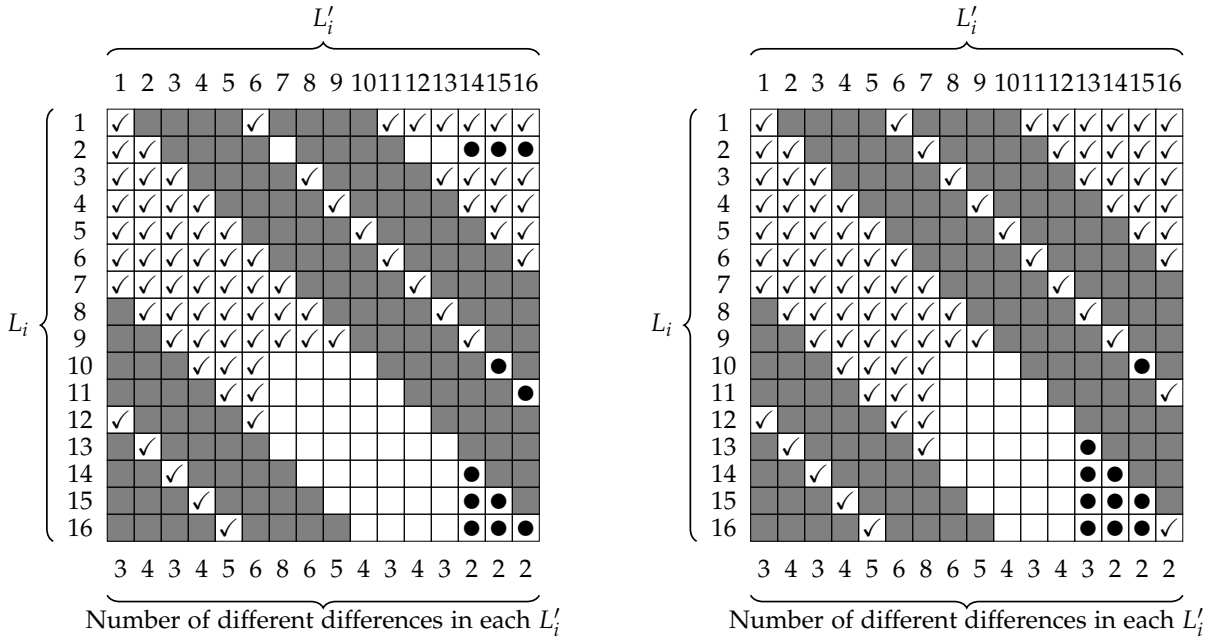
(a) Second guess: L'_6.

(b) End of the second determine step.

Figure 8.15: Second guess on the algorithm. A ✓ means we know both value and difference for that byte, a ● means that we only determined the difference for that byte and white bytes are not constrained yet.

Guessing the list L'_6 implies a 64-bit constraint of the list L_9 so that we get a single element out of it and determine four yet-unknown other bytes. This enables to learn the independent differences in L'_{14} and therefore, we filter an element from L_3 (two ✓ and four ●). At this stage, the list L'_1 is already fully constrained on its differences, so that we are left with a set of $2^{64-3\times 8} = 2^{40}$ values constrained on five bytes (five ✓). Hence, we are able to determine all the unset values in L'_1: see Figure 8.15b for the current situation.

Step 3. Again, the lack of constraints prevents us to determine more bytes. We continue by guessing the 2^8 elements left in L_1 (two ✓ and three ●), which makes the time complexity increase to 2^{280} (see Figure 8.16a).



(a) Third guess: L_1.

(b) End of the third determine step.

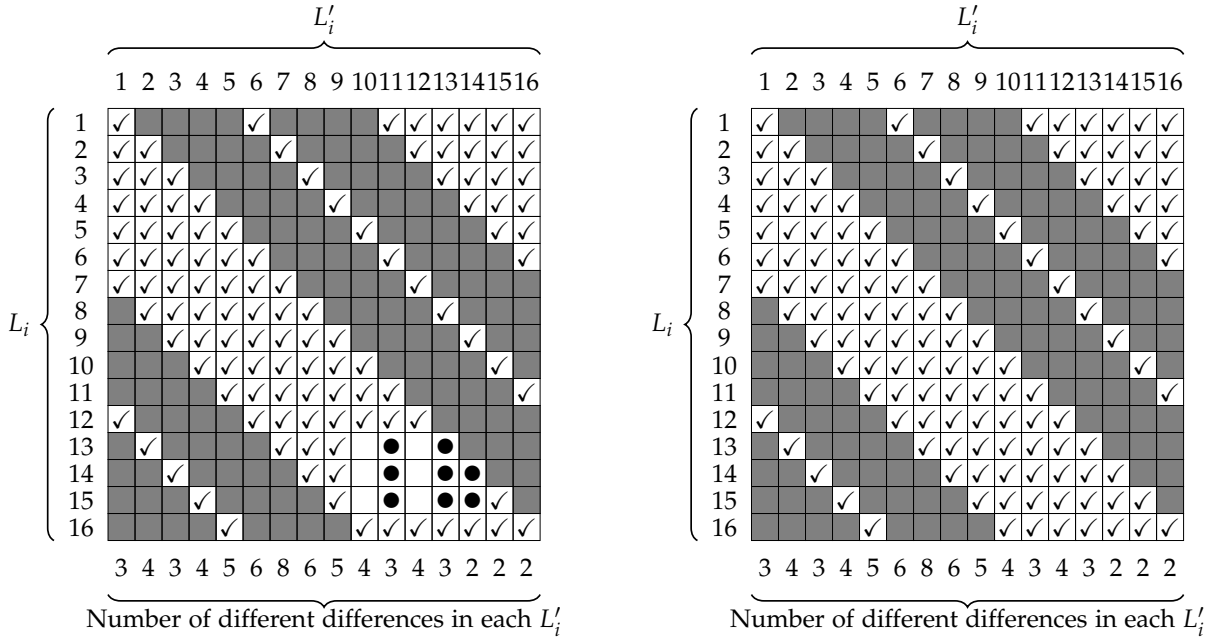
Figure 8.16: Third guess on the algorithm. A ✓ means we know both value and difference for that byte, a ● means that we only determined the difference for that byte and white bytes are not constrained yet.

The list L_1 being totally known, we derive the vector of differences in L'_{13}, which adds an extra byte-constraint on L_2 where only one element was left, and so fully determines it. From here, L'_7 becomes fully determined as well (four ✓) and so is L'_{16}. In the latter, the differences being known, we were left with a set of $2^{64-2\times 8} = 2^{48}$ values, which are now constrained on six bytes (six ✓).

Step 4. We describe in Figure 8.16b the knowledge propagated so far, with time complexity 2^{280} and probability 1. In this step, no new guess is needed, and we show how to end the algorithm by probabilistic filtering on the remaining unset lists.

First, we observe that L_{10} is over-determined (four ✓ and one ●) by one byte. This means that we get the correct value with probability 2^{-8}, whereas L_{11} is filtered with probability 1 (four ✓). We assume the correct values are found, such that the element of L'_8 happens to be correctly defined with probability 2^{-16} (five ✓), L'_9 with probability 1 (four ✓) and L'_{15} also with probability 1 since we get 6 ✓ that complete the knowledge of the 2-dimensional subspace of differences (six ✓ and two •). We continue in L'_{11} by learning the full vector of differences (three independent ✓ for a subspace of dimension 3), which constraints L_{12} on 11 bytes (five ✓ and one ●) so that we get a valid element with probability 2^{-24}.

At this point, L_{16} is reduced to a single element with probability 2^{-8} (three ✓ and three •), which adds constraints on the three lists L'_{11}, L'_{13} and L'_{14}, where we already know all the



(a) Almost finished.

(b) Finished.

Figure 8.17: End of the guess-and-determine algorithm. (a): After list L_{16} has been fully determined, we filter $L'_{10}, \ldots, L'_{14}$ with probability 1 and then $L_{13}, \ldots, L_{15}$ with probability 2^{-64}. (b): Each of the 32 lists has been reduced to one element in 2^{280} operations.

differences (Figure 8.17a). Consequently, we get respectively 5, 5 and 6 independent values (✓) on subspaces of respective dimensions 3, 3 and 2, which filter those three lists to a single element with probability 1. Finishing the guess and determine technique is done by filtering L'_{10} and L'_{12} with probability 1 (four ✓ in a subspace of dimension 4 for both lists), and then the three remaining lists L_{13}, L_{14} and L_{15} are all reduced to a single element which are the valid one with probability 2^{-64} for each (eight ✓). After this, if a solution is found, everything has been determined (Figure 8.17b).

In total, for each guess, we successfully merge the 32 lists with probability

$$2^{-8-16-24-40-64-64-64} = 2^{-280},$$

but the whole procedure is repeated $2^{64\times 4+16+8} = 2^{280}$ times, so we expect to find the one existing solution. All in all, we described a way to do the merge with time complexity 2^{280} and memory complexity 2^{64}. The final complexity to find a valid candidate for the whole characteristic is then 2^{392} computations and 2^{64} memory.

8.2.4.3 Comparison with ideal case

In the ideal case, obtaining a pair whose input difference lies in a subset of size $IN = 2^{512}$ and whose output difference lies in a subset of size $OUT = 2^{64}$ for a 1024-bit permutation requires $LB(512, 64) = 2^{448}$ computations. We can directly conclude that this leads to a distinguishing attack on the 10-round reduced version of the `Grøstl-512` permutation with 2^{392} computations and 2^{64} memory. Similarly, as explained in Section 8.1.1, this results

also induces a nontrivial observation on the 10-round reduced version of the Grøstl-512 compression function with identical complexity.

One can also derive slightly cheaper distinguishers by aiming fewer rounds while keeping the same generic complexity: instead of using the 10-round truncated characteristic from Figure 8.12, it is possible to remove either round 3 or 9 and spare one $8 \rightarrow 1$ truncated differential transition. Overall, this gives a distinguishing attack on the 9-round reduced version of the Grøstl-512 permutation with 2^{336} computations and 2^{64} memory. By removing both rounds 3 and 9, we achieve 8 rounds with 2^{280} computations.

One can further gain another small factor for the 9-round case by using a $8 \rightarrow 2$ truncated differential transition instead of $8 \rightarrow 1$, for a final complexity of 2^{328} computations and 2^{64} memory. Indeed, the generic complexity drops to $LB(512, 128) = 2^{384}$ since $OUT = 2^{128}$.

8.2.5 Distinguishers for reduced PHOTON permutations

Using the same cryptanalysis technique, it is possible to study the recent lightweight hash function family PHOTON [GPP11], which is based on five different versions of AES-like permutations. Using the notations previously described in Section 8.1.2, the five versions (c, t) for PHOTON are $(4, 5)$, $(4, 6)$, $(4, 7)$, $(4, 8)$ and $(8, 6)$ for increasing versions. All versions are defined to apply $N_r = 12$ rounds of an AES-like process.

Since the internal state is always square, by trivially adapting the method from Section 8.2.1 to the specific parameters of PHOTON, one can hope to obtain distinguishers for 9 rounds of the PHOTON internal permutations. However, we are able to do so only for the parameters $(4, 8)$ used in PHOTON-224/32/32 (see Table 8.2 with the comparison to previously known results). Indeed, the size t of the matrix plays an important role in the gap between the complexity of our algorithm and the generic one. The bigger is the matrix, the better is the gap between the algorithm complexity and the generic one.

8.3 Improved Outbound Part

In this section, we propose another new improvement of the previous rebound techniques, reducing the complexity of known differential distinguishers and by a lesser extend, reducing some collision attack complexities. We observe that the gap between the distinguisher complexity and the generic case is often large and some conditions might be relaxed in order to minimize as much as possible the overall complexity. The main idea is to generalize the various rebound techniques and to relax some of the input and output conditions of the differential distinguishers. That is, instead of considering pre-specified active cells in the input and output like full columns or diagonals, we consider several possible position combinations of these cells. In some way, this idea is related to the outbound difference randomization that was proposed in [DGPW12] for a rebound attack on Keccak, a non-AES-like function. Yet, in [DGPW12], the randomization was not used to reduce the attack complexity, but to provide enough freedom degrees to perform the attack.

As this improvement affects directly the properties of the inputs and outputs, we now

have to deal with a new differential property observed and we name this new problem the *multiple limited-birthday problem*, which is more general than the limited-birthday one. A very important question arising next is: what is the complexity of the best generic algorithm for obtaining such set of inputs/outputs? For previous distinguishers, where the active input and output columns were fixed, the limited-birthday algorithm (Section 7.1.3) is yet the best one for solving the problem in the generic case. Now, the multiple limited-birthday is more complex, and in Section 8.3.1 we discuss how to bound the complexity of the best generic distinguisher. Moreover, we also propose an efficient, generic and non-trivial algorithm in order to solve the multiple limited-birthday problem, providing the best known complexity for solving this problem.

Finally, we generalize the various rebound-like techniques in Section 8.3.2 and we apply our findings in Section 8.3.3 on various AES-like primitives such as AES [AES01], ECHO [BBG+09], Grøstl [GKM+11], LED [GPPR11], PHOTON [GPP11] and Whirlpool [BR11]. Our results are summarized and compared to previous works in Table 8.4.

8.3.1 Multiple limited-birthday and generic complexity

In this section, we present a new type of distinguisher: the multiple limited-birthday. It is inspired from the limited-birthday one (see Section 7.1.3), where some of the input and output conditions are relaxed. We discuss how to bound the complexity of the best generic algorithm for this problem, and we provide an efficient algorithm that solves the problem with the best known complexity. In more detail, we fix the number of active diagonals (resp. anti-diagonals) in the input (resp. output), but not their positions. Therefore, we have $\binom{t}{n_B}$ possible different configurations in the input and $\binom{t}{n_F}$ in the output. We state the following problem.

Problem 8.1 (Multiple limited-birthday). *Let $n_F, n_B \in \{1, \ldots, t\}$, P a permutation from the symmetric group $\mathfrak{S}_{\mathcal{S}}$ of the set of states $\mathcal{S}$, and Δ_{IN} be the set of truncated patterns containing all the $\binom{t}{n_B}$ possible ways to choose n_B active diagonals among the t ones. Let Δ_{OUT} defined similarly with n_F active anti-diagonals. The problem asks to find a pair $(m, m') \in \mathcal{S}^2$ of inputs to P such that $m \oplus m' \in \Delta_{IN}$ and $P(m) \oplus P(m') \in \Delta_{OUT}$.*

We conjecture that the best generic algorithm for finding one solution to Problem 8.1 has a time complexity that is lower bounded by the limited-birthday algorithm (Section 7.1.3) when considering

$$\underline{IN} = \binom{t}{n_B} 2^{t \cdot c \cdot n_B} \qquad \text{and} \qquad \underline{OUT} = \binom{t}{n_F} 2^{t \cdot c \cdot n_F}.$$

This can be reasonably argued as we can transform the multiple limited-birthday algorithm into a similar (but not equivalent) limited-birthday one, with a size of all the possible truncated input and output differences of $\underline{IN}$ and $\underline{OUT}$ respectively. Solving the similar limited-birthday problem requires a complexity of $LB(\underline{IN}, \underline{OUT})$, but solving the original multiple limited-birthday problem would require an equal or higher complexity, as though having the same possible input and output difference sizes, for the same number of inputs (or outputs), the number of valid input pairs that can be built might be lower. This is directly reflected on the

Target	Subtarget	Rounds	Type	Time	Memory	Ideal	Reference
AES-128	Cipher	8	KK dist.	2^{48}	2^{32}	2^{65}	[GP10]
		8	KK dist.	2^{44}	2^{32}	2^{61}	Section 8.3.3.1
		8	CK dist.	2^{24}	2^{16}	2^{65}	Chapter 7
		8	CK dist.	$2^{13.4}$	2^{16}	$2^{31.7}$	Section 8.3.3.1
AES-128	DM-mode	6	collision	2^{56}	2^{32}	2^{65}	[MPRS09]
		6	collision	2^{32}	2^{16}	2^{65}	Section 8.3.3.1
ECHO	Permutation	7	dist.	2^{118}	2^{38}	2^{1025}	[SLW+10]
		7	dist.	2^{102}	2^{38}	2^{256}	Section 8.3.3.2
		8	dist.	2^{151}	2^{67}	2^{257}	[NP11]
		8	dist.	2^{147}	2^{67}	2^{256}	Section 8.3.3.2
Grøstl-256	Permutation	8	dist.	2^{16}	2^{8}	2^{33}	[SLW+10]
		8	dist.	2^{10}	2^{8}	$2^{31.5}$	Section 8.3.3.3
		9	dist.	2^{368}	2^{64}	2^{384}	Section 8.2
		9	dist.	2^{362}	2^{64}	2^{379}	Section 8.3.3.3
Grøstl-256	Comp. func.	6	collision	2^{120}	2^{64}	2^{257}	[Sch11]
		6	collision	2^{119}	2^{64}	2^{257}	Section 8.3.3.3
Grøstl-256	Hash func.	3	collision	2^{64}	2^{64}	2^{129}	[Sch11]
		3	collision	2^{63}	2^{64}	2^{129}	Section 8.3.3.3
LED-64	Cipher	15	CK dist.	2^{16}	2^{16}	2^{33}	[GPPR11]
		16	CK dist.	$2^{33.5}$	2^{32}	$2^{41.4}$	[NWW13]
		20	CK dist.	$2^{60.2}$	$2^{61.5}$	$2^{66.1}$	[NWW13]
		19	CK dist.	2^{18}	2^{16}	2^{33}	Section 8.3.3.4
PHOTON-80/20/16	Permutation	8	dist.	2^{8}	2^{4}	2^{11}	[GPP11]
		8	dist.	$2^{3.4}$	2^{4}	$2^{9.8}$	Section 8.3.3.5
PHOTON-128/16/16	Permutation	8	dist.	2^{8}	2^{4}	2^{13}	[GPP11]
		8	dist.	$2^{2.8}$	2^{4}	$2^{11.7}$	Section 8.3.3.5
PHOTON-160/36/36	Permutation	8	dist.	2^{8}	2^{4}	2^{15}	[GPP11]
		8	dist.	$2^{2.4}$	2^{4}	$2^{13.6}$	Section 8.3.3.5
PHOTON-224/32/32	Permutation	8	dist.	2^{8}	2^{4}	2^{17}	[GPP11]
		8	dist.	2^{2}	2^{4}	$2^{15.5}$	Section 8.3.3.5
		9	dist.	2^{184}	2^{32}	2^{192}	Section 8.2
		9	dist.	2^{178}	2^{32}	2^{187}	Section 8.3.3.5
PHOTON-256/32/32	Permutation	8	dist.	2^{16}	2^{8}	2^{25}	[GPP11]
		8	dist.	$2^{10.8}$	2^{8}	$2^{23.7}$	Section 8.3.3.5
Whirlpool	Permutation	10	dist.	2^{176}	2^{8}	2^{384}	[LMR+10]
		10	dist.	$2^{115.7}$	2^{8}	2^{125}	Section 8.3.3.6
Whirlpool	Comp. func.	7.5	collision	2^{184}	2^{8}	2^{256}	[LMR+10]
		7.5	collision	2^{176}	2^{8}	2^{256}	Section 8.3.3.6
Whirlpool	Hash func.	5.5	collision	2^{184}	2^{8}	2^{256}	[LMR+10]
		5.5	collision	2^{176}	2^{8}	2^{256}	Section 8.3.3.6

Table 8.4: Known and improved results for various rebound-based attacks on AES-based primitives.

complexity of solving the problem, as in the limited-birthday algorithm, it is considered that for 2^n inputs queried, we can build 2^{2n-1} valid input pairs. The optimal algorithm solving Problem 8.1 would have a time complexity T such that: $LB(\underline{IN}, \underline{OUT}) \leq T$.

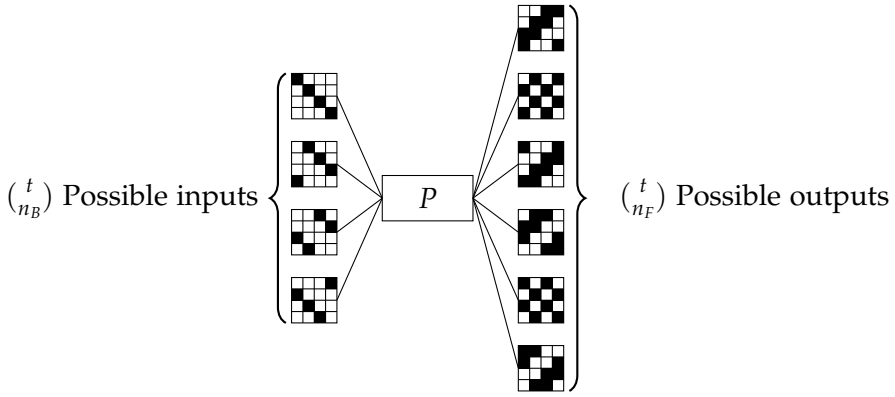


Figure 8.18: Possible inputs and outputs of the relaxed generic distinguisher. The blackbox P implements a random permutation uniformly drawn from $\mathfrak{S}_{\mathcal{S}}$. The figure shows the case $t = 4$, $n_B = 1$ and $n_F = 2$.

We have just provided a lower bound for the complexity of solving Problem 8.1 in the ideal case, but an efficient generic algorithm is not known. For finding a solution, we could repeat the algorithm for solving the limited-birthday while considering sets of input or output differences that do not overlap, with a complexity of $\min\{LB(\overline{IN}, \underline{OUT}), LB(\underline{IN}, \overline{OUT})\}$, where:

$$\overline{IN} = 2^{t \cdot c \cdot n_B} \qquad \underline{IN} = \binom{t}{n_B} 2^{t \cdot c \cdot n_B}$$
$$\overline{OUT} = 2^{t \cdot c \cdot n_F} \qquad \underline{OUT} = \binom{t}{n_F} 2^{t \cdot c \cdot n_F}.$$

We propose in the sequel a new generic algorithm to solve Problem 8.1 whose time complexity verifies the claimed bound and improves the complexity of the algorithm previously sketched. It allows then to find solutions faster than previous algorithms, as detailed in Table 8.5. Without loss of generality, because the problem is completely symmetrical, we explain the procedure in the forward direction. The same reasoning applies for the backward direction, when changing the roles between input and output of the permutation, and the complexity would then be the lowest one.

From Problem 8.1, we see that a random pair of inputs has a probability $P_{out} = \binom{t}{n_F} 2^{-t(t-n_F)c}$ to verify the output condition. We therefore need at least P_{out}^{-1} input pairs so that one verifying the input and output conditions can be found. The first goal of the procedure consists in constructing a structure containing enough input pairs.

8.3.1.1 Structures of input data

We want to generate the amount of valid input pairs previously determined, and we want do this while minimizing the numbers of queries performed to the encryption oracle, as the complexity directly depends on them. A natural way to obtain pairs of inputs consists in packing the data into structured sets. These structures contain all 2^{ct} possible values on n'_B different diagonals at the input, and make the data complexity equivalent to $2^{n'_B ct}$ encryptions.

If there exists $n'_B \leq n_B$ such that the number N of possible pairs $\binom{2^{n'_B ct}}{2}$ we can construct within the structure verifies $N \geq P_{out}^{-1}$, then Problem 8.1 can be solved easily by using the birthday algorithm. If this does not hold, we need to consider a structure with $n'_B > n_B$. In this case we can construct as many as

$$\binom{n'_B}{n_B} 2^{(n'_B - n_B)tc} \binom{2^{n_B tc}}{2}$$

pairs (m, m') of inputs such that $m \oplus m'$ already belongs to Δ_{IN}. We now propose an algorithm that handles this case.

We show how to build a fixed number of pairs with the smallest structure we could find, and we conjecture that the construction is optimal in the sense that this structure is the smallest possible. The structure of input data considers n'_B diagonals $D_1, \ldots, D_{n'_B}$ assuming all the 2^{ct} possible values, and an extra diagonal D_0 assuming $2^y < 2^{ct}$ values. In total, the number of queries equals $2^{y+n'_B tc}$. Within this structure, we can construct a number of pairs parameterized by n'_B and $y > 0$ that is equal to

$$N_{pairs}(n'_B, y) \stackrel{\text{def}}{:=} \binom{n'_B}{n_B} \binom{2^{n_B ct}}{2} 2^y \, 2^{(n'_B - n_B)tc} + \binom{n'_B}{n_B - 1} \binom{2^{y+(n_B-1)ct}}{2} 2^{(n'_B - (n_B-1))ct}.$$

The first term of the sum considers the pairs generated from n_B diagonals among the $D_1, \ldots, D_{n'_B}$ diagonals, while the second term considers D_0 and $n_B - 1$ of the other diagonals. The problem of finding an algorithm with the smallest time complexity is therefore reduced to finding the smallest n'_B and the associated y so that $N_{pairs}(n'_B, y) = P_{out}^{-1}$. Depending on the considered scenarios, P_{out}^{-1} would have different values, but finding (n'_B, y) such that $N_{pairs}(n'_B, y) = P_{out}^{-1}$ can easily be done by an intelligent search in $\log(t) + \log(ct)$ simple computations by trying different parameters until the ones that generate the wanted amount of pairs P_{out}^{-1} is found.

When $y = 0$, we compute the number of terms as $N_{pairs}(n'_B, 0) := \binom{n'_B}{n_B} \binom{2^{n_B ct}}{2} 2^{(n'_B - n_B)tc}$.

8.3.1.2 Generic algorithm

Once we have found the good parameters n'_B and y, we generate the $2^{y+n'_B ct}$ inputs as previously described, and query their corresponding outputs to the permutation P. We store the input/output pairs in a table ordered by the output values. Assuming they are uniformly distributed, there exists a pair in this table satisfying the input and output properties from Problem 8.1 with probability close to 1.

To find it, we first check for each output if a matching output exists in the list. When this is the case, we next check if the found pair also verifies the input conditions. The time complexity of this algorithm therefore is about

$$2^{y+n'_B ct} + 2^{2y+2n'_B tc} P_{out}$$

computations. The first term in the sum is the number of outputs in the table: we check for each one of them if a match exists at cost about one computation. The second term is the

number of output matches that we expect to find, for which we also test if the input patterns conform to the wanted ones.

Finally, from the expression of P_{out}, we approximate the time complexity $2^{y+n'_B ct} + 2^{2y+2n'_B tc} P_{out}$ to $2^{y+n'_B ct}$ operations, as the second term is always smaller than the first one. The memory complexity if we store the table would be $2^{y+n'_B ct}$ as well, but we can actually perform this research without memory, as in fact what we are doing is a collision search. In Table 8.5, we show some examples of different complexities achieved by the bounds proposed and by our algorithm.

Parameters (t, c, n_B, n_F)	**bound** $LB(\underline{IN}, \underline{OUT})$	**Our algorithm**	**bound** $LB(\overline{IN}, \underline{OUT})$
$(8, 8, 1, 1)$	2^{379}	$2^{379.7}$	2^{382}
$(8, 8, 1, 2)$	$2^{313.2}$	$2^{314.2}$	$2^{316.2}$
$(8, 8, 2, 2)$	$2^{248.4}$	$2^{250.6}$	$2^{253.2}$
$(8, 8, 1, 3)$	$2^{248.19}$	$2^{249.65}$	$2^{251.19}$
$(4, 8, 1, 1)$	2^{61}	$2^{62.6}$	2^{63}
$(4, 4, 1, 1)$	2^{29}	$2^{30.6}$	2^{31}

Table 8.5: Examples of time complexities for several algorithms solving the multiple limited-birthday problem.

8.3.2 Truncated characteristic with relaxed conditions

In this section, we present a representative 9-round example of our new distinguisher.

8.3.2.1 Relaxed 9-round distinguisher for AES-like permutation

We show how to build a 9-round distinguisher when including the idea of relaxing the input and output conditions. In fact, this new improvement allows to reduce the complexity of the distinguisher, as the probability of verifying the outbound is higher. We point out that we have chosen to provide an example for 9 rounds to extend the improved results for the inbound phase of the previous Section 8.2.

We also recall that for a smaller number of rounds, the only difference with the presented distinguisher is the complexity $C_{inbound}$ for the inbound part, that can be solved using already described and well-known methods such as rebound attacks, **Super-SBox** or start-from-the-middle, depending on the particular situation that we have. For the sake of simplicity, in the end of this section, we provide the complexity of the distinguisher depending on the inbound complexity $C_{inbound}$.

In the end of the section, we compare our distinguisher with the previously explained best known generic algorithm to find pairs conforming to those cases. We show how the complexities of our distinguisher are still lower than the lowest bound for such a generic case.

Following the notation from the previous section, we parameterize the truncated differential characteristic by four variables (see Figure 8.19) such that trade-offs are possible by finding the

right values for each one of them. Namely, we denote c the size of the cells, $t \times t$ the size of the state matrix, n_B the number of active diagonals in the input (alternatively, the number of active cells in the second round), n_F the number of active independent diagonals in the output (alternatively, the number of active cells in the eighth round), m_B the number of active cells in the third round and m_F the number of active cells in the seventh round.

Hence, the sequence of active cells in the truncated differential characteristic becomes:

$$t\,n_B \xrightarrow{R_1} n_B \xrightarrow{R_2} m_B \xrightarrow{R_3} t\,m_B \xrightarrow{R_4} t^2 \xrightarrow{R_5} t\,m_F \xrightarrow{R_6} m_F \xrightarrow{R_7} n_F \xrightarrow{R_8} t\,n_F \xrightarrow{R_9} t^2, \tag{8.25}$$

with the constraints $n_F + m_F \geq t+1$ and $n_B + m_B \geq t+1$ that come from the MDS property, and relaxation conditions on the input and output, meaning that the positions of the n_B input active diagonals, and of the n_F active anti-diagonals generating the output can take any possible configuration, and not a fixed one. This allows to increase the probability of the outbound part and the number of solutions conforming to the characteristic. The binary logarithm of the amount of solutions that we can now generate equals to:

$$\log_2\left(\binom{t}{n_B}\binom{t}{n_F}\right) + ct^2 + ctn_B \\ - c(t-1)n_B - c(t-m_B) - ct(t-m_F) - c(t-1)m_F - c(t-n_F),$$

which simplifies to:

$$c(n_B + n_F + m_B + m_F - 2t) + \log_2\left(\binom{t}{n_B}\binom{t}{n_F}\right).$$

If follows from the MDS constraints that there are always at least $\binom{t}{n_B}\binom{t}{n_F}2^{2c}$ freedom degrees, independently of t.

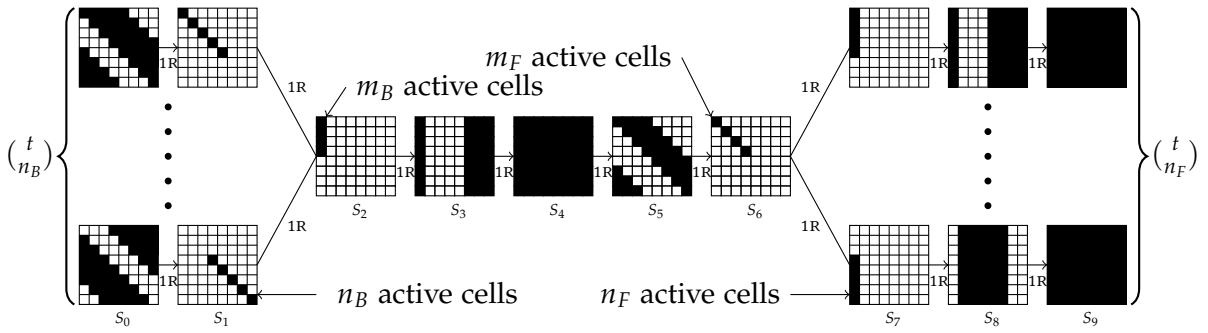


Figure 8.19: The 9-round truncated differential characteristic used to distinguish an AES-like permutation from an ideal permutation. The figure shows some particular values: $t = 8$, $n_B = 5$, $m_B = 4$, $m_F = 4$ and $n_F = 5$.

To find a conforming pair we use the algorithm proposed in Section 8.2 for solving the inbound part and finding a solution for the middle rounds. The cost of those uncontrolled rounds is given by:

$$C_{outbound} := \frac{2^{c(t-n_B)}}{\binom{t}{n_B}} \cdot \frac{2^{c(t-n_F)}}{\binom{t}{n_F}} = \frac{2^{c(2t-n_B-n_F)}}{\binom{t}{n_B}\binom{t}{n_F}},$$

since we need to pass one $n_B \leftarrow m_B$ transition in the backward direction with $\binom{t}{n_B}$ possibilities and one $m_F \rightarrow n_F$ transition in the forward direction with $\binom{t}{n_F}$ possibilities.

8.3.2.2 Comparison with ideal case

As we discussed in Section 8.3.1, in the ideal case, the generic complexity T is bounded by

$$LB(\underline{IN}, \underline{OUT}) \leq T \leq \min\left\{LB(\underline{IN}, \overline{OUT}), LB(\overline{IN}, \underline{OUT})\right\},$$

where we have

$$\underline{IN} = \binom{t}{n_B} 2^{t \cdot c \cdot n_B} \qquad \overline{IN} = 2^{t \cdot c \cdot n_B}$$
$$\underline{OUT} = \binom{t}{n_F} 2^{t \cdot c \cdot n_F} \qquad \overline{OUT} = 2^{t \cdot c \cdot n_F}.$$

We have proposed the algorithm with the best known complexity for solving the problem in the ideal case in Section 8.3.1, for being sure that our distinguishers have smaller complexity than the best generic algorithm, we compare our complexities with the inferior bound given: $LB(\underline{IN}, \underline{OUT})$, so that we are sure that our distinguishers are valid.

We recall here that the complexity of the distinguishers that we build varies depending on the number of rounds solved in the middle, or the parameters chosen, and we provide some examples of improvements of previous distinguishers and their comparisons with the general bounds and algorithms in the next section.

8.3.3 Applications

In this section, we apply our new techniques to improve the best known results on various primitives using AES-like permutations. When we randomize the input/output differences positions, the generic complexities that we compare with are the ones coming from the classical limited-birthday problem $LB(\underline{IN}, \underline{OUT})$ (updated with the right amount of differences), since they lower bound the corresponding multiple limited-birthday problem.

8.3.3.1 AES

We recall that the full specifications of the AES are given in Section 4.2.

Distinguisher

Except for the biclique technique [BKR11] which allows to do a speed-up search for the key by a factor of 0.27, the current best distinguishers can reach 8 rounds with 2^{48} computations in the known-key model (see [GP10]) and with 2^{24} computations in the chosen-key model (see Chapter 7). By relaxing some input/output conditions, we are able to obtain an 8-round distinguisher with 2^{44} computations in the known-key model and with $2^{13.4}$ computations in the chosen-key model.

In the case of the known-key distinguisher, we start with the 8-round differential characteristic depicted in Figure 8.20. One can see that it is possible to randomize the position of the unique active byte in both states S_1 and S_6, resulting in 4 possible positions for both the input and output differences. We reuse the **Super-SBox** technique that can find solutions from state S_2 to state S_5 with a single operation on average. Then, one has to pay $2^{24}/4 = 2^{22}$ for both transitions from state S_2 to S_1 backward and from state S_5 to S_6 forward, for a total complexity of 2^{44} computations. In the ideal case, our multiple limited-birthday problem gives us a generic complexity bounded by 2^7 computations.

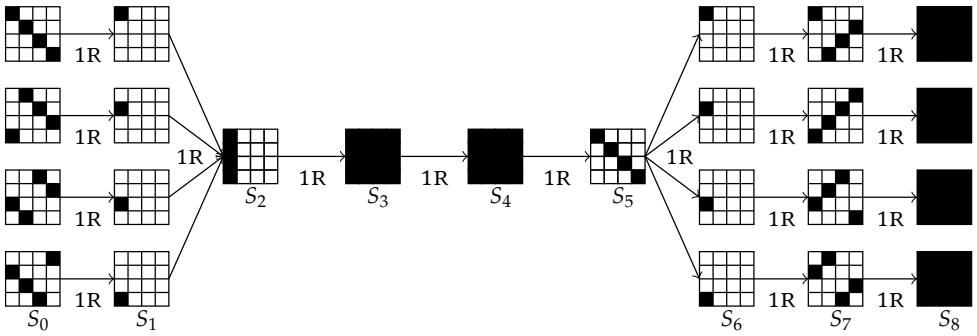


Figure 8.20: Differential characteristic for the 8-round known-key distinguisher for AES-128

Concerning the chosen-key distinguisher, we start with the 8-round differential characteristic depicted in Figure 8.21. Here, we use the technique introduced in Section 7.3 that can find solutions from state S_2 to state S_6 with a single operation on average. It is therefore not possible to randomize the position of the unique active byte in state S_6 since it is already specified. However, for the transition from state S_2 to S_1, we let two active bytes to be present in S_2, with random positions (6 possible choices). This happens with a probability $6 \cdot 2^{-16}$ and the total complexity to find a solution for the entire characteristic is $2^{13.4}$ computations. In the ideal case, our multiple limited-birthday problem gives us a generic complexity bounded by $2^{31.7}$ computations.

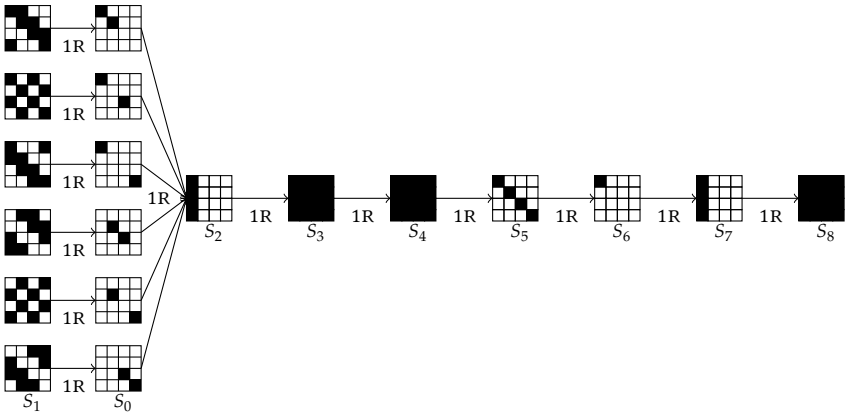


Figure 8.21: Differential characteristic for the 8-round chosen-key distinguisher for AES-128

Collision

It is also interesting to check what happens if the AES cipher is plugged into a classical Davies-Meyer mode in order to get a compression function. A collision attack for this scenario was proposed in [MPRS09] for 6 rounds of AES with 2^{56} computations. By considering the characteristic from state S_1 to state S_7 state in Figure 8.20 (the **MixColumns** in the last round is omitted for AES, thus S_7 contains only a single active byte), and by using the technique introduced in Section 7.3 (only for chosen-key model, but in the Davies-Meyer mode the key input of the cipher is fully controlled by the attacker since it represents the message block input), we can find solutions from state S_2 to state S_6 with a single operation on average. Then, one has to pay a probability 2^{-24} for the differential transition from state S_2 to state S_1 when computing backward. One cannot randomize the single active cells positions here because the collision forces us to place them at the very same position. Getting the single input and output active bytes to collide requires 2^8 tries and the total complexity of the 6-round collision search is therefore 2^{32} computations.

8.3.3.2 ECHO

The ECHO hash function reuses the AES for its compression function: we present dedicated attacks for this hash functions in the next Chapter 9, so refer to that chapter or to the submission document [BBG+09] for its complete description.

Distinguisher for 8 rounds

The current best distinguisher for the full ECHO permutation has been published by María Naya-Plasencia in [NP11]. The algorithm improves a rebound technique with a non-fully-active characteristic from Sasaki et al. in [SLW+10] and runs in 2^{151} computations and 2^{67} memory.

Using the same strategy as [NP11], we relax the outbound phase by allowing more truncated patterns in both the plaintext and the ciphertext. While the inbound phase remains unchanged with the same average time complexity of 2^{65} computations to get one pair for the middle rounds, we increase the probability $P_{out} = 2^{-32} \times 2^{-54}$ of the outbound phase. Namely, the backward part of probability $p_1 = 2^{-32}$ which makes a precise AES state to become inactive can be randomized to any of the four possibles AES states, so that we get a probability of success of $4 \times p_1$. Similarly, in the forward part of probability $p_2 = 2^{-54}$ can be increased to $4 \times p_2$ for the same reasons. In return, we now have four possibles truncated patterns at both sides, which makes the generic complexity cheaper and bounded by 2^{252}. Overall, we gain a factor $4^2 = 2^4$ over the previous algorithms, so that we can distinguish 8 rounds of the ECHO permutation in $2^{65+86}/2^4 = 2^{147}$ computations and memory 2^{67}.

Distinguisher for 7 rounds

In the original paper [SLW+10], Sasaki et et al. also introduce a distinguisher on 7 rounds of the ECHO permutation which requires 2^{118} computations and 2^{38} memory.

Using a similar strategy, we derive a distinguisher requiring 2^{102} computations and the same amount of memory. The idea consists in relaxing the outbound phase of probability 2^{-118}

by allowing 3 out of 4 AES states with one column active after the probabilistic filter. Due to the **SuperMixColumns** linear transformation (see Section 9.1.2, [Sch10, JF11]), the probability of the whole outbound phase increases to $4 \cdot 2^{-96} \cdot 2^{-8} = 2^{-102}$. Indeed, while a $4 \rightarrow 4$ transition in the **SuperMixColumns** transformations occurs with probability 2^{-24}, a $4 \rightarrow 12$ transition occurs with probability 2^{-8}.

8.3.3.3 Grøstl

Distinguisher

The current best known-key distinguishers on Grøstl-256 internal permutations can reach 8 rounds with 2^{16} computations (see [SLW+10]), 9 rounds with 2^{368} computations (see Section 8.2) and 10 rounds with a zero-sum distinguisher requiring 2^{509} computations (see [BCD11]).

For the 8-round distinguisher case, we use the same attack as in [SLW+10] with only a single inactive diagonal on the input and a single inactive column on the output, but their positions can be randomized through the 8 possible choices. Overall, we gain a factor 8^2 over the previous complexity, leading to 2^{10} operations, while the multiple limited-birthday problem gives a generic complexity bounded by $2^{31.5}$.

The 9-round case is already described in Section 8.3.2, and we minimize the distinguisher complexity by using parameters $n_B = 1$, $m_B = 8$, $m_F = 8$, $n_F = 1$. Again, we can randomize the forward and backward single active cell position (for an improvement factor of t^2) and this gives a total complexity of 2^{362} computations, while the multiple limited-birthday problem has a generic complexity bounded by 2^{379} computations.

Collision

We recall that Grøstl-256 uses two AES-like 512-bit permutations P and Q in a special mode in order to build its compression function:

$$h(H, M) = Q(M) \oplus P(M \oplus H) \oplus H.$$

In [Sch11], a semi-free-start collision on 6 rounds of the compression function is given, with a differential characteristic equivalent to the one from state S_1 to state S_7 in Figure 8.20 (but with $t = 8$). We can randomize the position of the single active byte forward in S_1 and backward in S_6, however not all 8 positions are possible. Indeed, in order to get a collision, in [Sch11] solutions are found for the 6-round characteristic in Q and for the 6-round characteristic in P, and a birthday between the two sets is performed to match the differences in both the input and the output. The issue is that the **ShiftRows** constants defined in P and Q are different and some positions randomization always fail to provide a collision. When analyzing the distinct **ShiftRows** constants from P and Q, one can check that the position of the active columns in S_2 and diagonals in S_5 can be chosen such that there are two single active byte position randomizations possible for both the input and the output. This improves the total collision complexity by a factor 2 and not 4 because we are performing a birthday.

Concerning the hash function, we can improve the 3-round collision attack given in [Sch11], but by only randomizing the single active byte position in the output (the input is fully active

in the differential characteristic). Again, two positions are possible for randomization and since no birthday is applied between the two permutations, we get an improvement by a factor 2.

Note that for the first submitted version of Grøstl (renamed Grøstl-0), the active cells position randomization gain factor is much higher (8 positions are possible instead of 2 for both forward and backward) because the issue with the distinct **ShiftRows** constants in P and Q is avoided. However, nothing can be improved concerning the internal differential attacks [Pey10] because they require the single active bits position to be placed exactly where the constants between P and Q are different.

8.3.3.4 LED

The best attacks on LED so far reach 15, 16 and 20 rounds with 2^{16}, $2^{33.5}$ and $2^{60.2}$ computations respectively (chosen key distinguishers from [GPPR11, NWW13]).

We describe a chosen-key distinguisher that can reach 19 rounds (over 32) with 2^{18} computations only. We use the differential characteristic from Figure 8.22 with a fixed single active nibble difference in the key input K. We solve independently the 4-round subparts from state

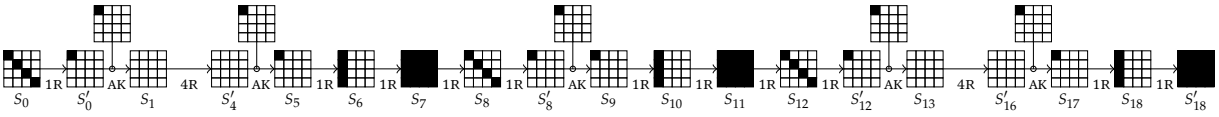


Figure 8.22: 19-round truncated differential characteristic for LED-64.

S_5 to S'_8 and from state S_9 to S'_{12}. Each subpart can be handled with the Super-SBox technique with 2^{12} computations on average (for example the Super-SBox technique finds one solution on average from state S_6 to S'_8, and the differential transitions from state S_6 to S_5 cost 2^{12} tries). For each subpart, the single nibble difference has to be erased by the fixed key difference (in S_5 and in S'_{12}) and this happens with probability 2^{-4}. Therefore, 2^{16} computations are required to generate one solution per subpart. We now have to connect solutions of these two subparts and we first handle the connection of the single active nibble difference (from S'_8 to S_9) by producing 2^2 solutions for each subpart, and merging these single nibble differences using the birthday paradox. Once a solution is found for the difference merge, we simply connect the values of the two subparts by choosing the appropriate value of the key K. The rest of the differential characteristic is verified with probability 1: generating a solution for the entire characteristic therefore costs 2^{18} computations, while in the ideal case the limited-birthday problem gives us 2^{33} computations.

8.3.3.5 PHOTON

The current best distinguishers on the internal permutations of PHOTON can reach 8 rounds with very low complexity, 2^8 for the four first functions and 2^{16} for PHOTON-256/32/32. For 9 rounds, only PHOTON-224/32/32 was attacked (with complexity 2^{184}) because it is the only one that uses a large enough internal state.

For all the 8-round attacks, the differential characteristics considered are equivalent to the one depicted in Figure 8.20 for AES-128, but with a matrix size adapted to the parameter t. As such, we can randomize the forward and backward single active cell position, which provides

in total a complexity improvement factor of t^2 (t forwards and t backwards) and of course, the ideal complexity decreases as well according to the multiple limited-birthday problem. It is to be noted that, as in [GPP11], the complexities for 8 rounds of PHOTON are average complexities per solution, but finding a single solution might cost more because the inbound solving outputs $2^{t \cdot c}$ solutions with $2^{t \cdot c}$ computations.

8.3.3.6 Whirlpool

The current best distinguishing attack for Whirlpool can reach the full 10 rounds of the internal permutation and compression function with 2^{176} computations, while the best collision attack can reach 5.5 rounds of the hash function and 7.5 rounds of the compression function [LMR+10] with 2^{184} computations. We show how to improve the complexities of all these attacks.

Distinguisher

We reuse the same differential characteristic from [LMR+10] for the distinguishing attack on the full 10-round Whirlpool compression function (which contains no difference on the key schedule of E), but we let three more active bytes in both states S_1 and S_8 of the outbound part and this is depicted in Figure 8.23. The effect is that the outbound cost of the differential characteristic is reduced to 2^{64} computations: 2^{32} for the differential transition from state S_2 to S_1 and 2^{32} from state S_7 to S_8. Moreover, we can leverage the difference position randomization in states S_1 and S_8, which both provide an improvement factor of $\binom{8}{4} = 70$. The inbound part in [LMR+10] (from states S_2 to S_7) requires 2^{64} computations to generate a single solution on average, and we obtain a final complexity of $2^{64} \cdot 2^{64} \cdot (70)^{-2} = 2^{115.7}$ Whirlpool evaluations, while the multiple limited-birthday problem has a generic complexity bounded by 2^{125} computations.

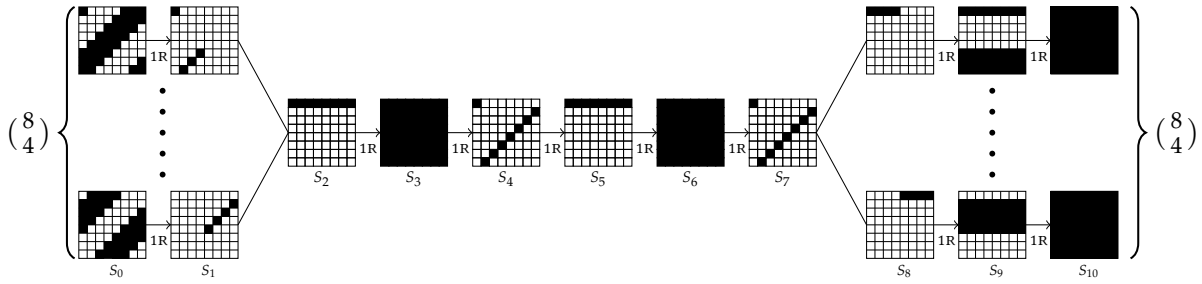


Figure 8.23: 10-round truncated differential characteristic for the full Whirlpool compression function distinguisher.

Collision

We reuse the same differential characteristic from [LMR+10] for the 7.5-round collision attack on the Whirlpool compression function that contains no difference on the key schedule of E, but we let one more active byte in both states S_0 and S_7 of the outbound part (see Figure 8.24). From this, we gain an improvement factor of 2^8 in both forward and backward directions of the outbound (from state S_1 to S_0 and from state S_6 to S_7), but we have two byte positions to collide

on with the feed-forward instead of one. After incorporating this 2^8 extra cost, we obtain a final improvement factor of 2^8 over the original attack (it is to be noted that this improvement does not work for 7-round reduced `Whirlpool` since the active byte position randomization would not be possible anymore). The very same method applies to the 5.5-round collision attack on the `Whirlpool` hash function.

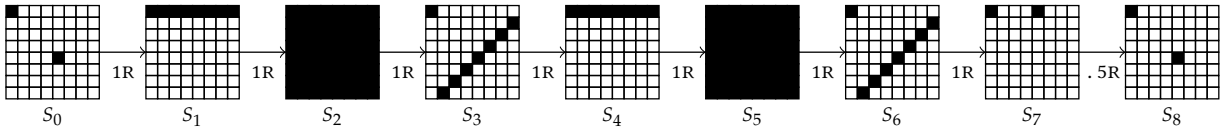


Figure 8.24: 7.5-round truncated differential characteristic for the `Whirlpool` compression function collision.

CHAPTER 9

Rebound Attacks on ECHO

Contents

9.1 Description of ECHO **219**
9.1.1 Original description 220
9.1.2 Alternative description 222
9.1.3 Current cryptanalysis 226
9.2 Attacks on ECHO-256 . . . . . **227**
9.2.1 Collision attack on the 4-round compression function 227
9.2.2 Collision attack on the 4-round hash function 237
9.2.3 Collision attack on the 5-round hash function 239
9.2.4 Distinguisher for the 7-round compression function 247
9.2.5 Collision attack on the 6-round compression function 252
9.2.6 Chosen-salt attacks on the compression function 252

In this chapter, we give an analysis of the ECHO-256 variant of the ECHO hash function [BBG+09]. It has been developed by Ryad Benadjila, Olivier Billet, Henri Gilbert, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw and Yannick Seurin, and has been submitted as a candidate for the SHA-3 competition. It has been selected to enter the second round of the selection but was not accepted to enter the final.

The content of this chapter is largely inspired by the article [JF11] co-authored with Pierre-Alain Fouque and published at FSE 2011, and by the article [JNPS11a] written together with María Naya-Plasencia and Martin Schläffer and published at SAC 2011. Some results have been reported in an extended version of the SAC 2011 paper in [JNPS11b].

9.1 Description of ECHO

The hash function ECHO is a wide-pipe AES-based design that *echoes* the well-known operations of the AES. Indeed, one goal of the designers is to reuse the AES design principles to reach an effective security against differential cryptanalysis. This has clearly been the strongest point in the design of the AES, so that a larger variant as a hash function should provide very good resistance as well.

The NIST has chosen ECHO to enter the second round on the 24 July 2009, for both its *unique hash algorithm design* and for its *performance on current high-end platforms*, even if AES-NI instructions are required to achieve impressive performance. Unfortunately, the NIST has not selected ECHO for the final round, mostly due to the very good performances of its competitors, despite that the security margin for collision has not been threatened.

9.1.1 Original description

ECHO is an iterated hash function and its compression function updates an internal state described by a 16×16 matrix of elements from $\mathrm{GF}(2^8)$, and can be viewed as a 4×4 matrix of 16 AES states (Figure 9.1).

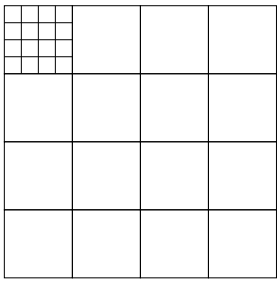


Figure 9.1: The internal state of ECHO is seen as a square matrix 4×4 of AES states of 128 bits.

Transformations on this large 2048-bit state are very similar to the one of the AES (see Figure 9.2), the main difference being the equivalent S-Box called **BigSubWords** (BigSB), which consists of two AES rounds. There are two constants added in those rounds that act as round keys: the first one is a counter depending on the current AES state being processed, and the second one is a salt. This salt parameter is constant for the compression function, and behaves as an additional input value to the function. These constants are mainly introduced to break the existing symmetries of the AES unkeyed permutation [LSWD04]. Since we are not using any property relying on symmetry and that adding constants does not change differences, we omit these steps in the following.

The diffusion of the AES states in ECHO follows the wide trail strategy [DR02] and is ensured by two *big* transformations: **BigShiftRows** (BigSR) and **BigMixColumns** (BigMC). As in the AES, the **BigShiftRows** only moves the cells of the internal states in a row, while the **BigMixColumns** linearly mixes the four states within a column.

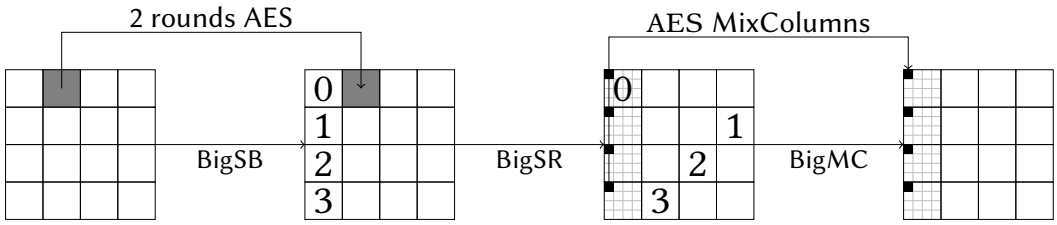


Figure 9.2: One round of the ECHO permutation. Each of the 16 cells is an AES state.

The permutation P used in the compression function f applies 8 rounds of the composition of the three presented layers: BigMC $\circ$ BigS $\circ$ BigSB, and in the end, the **BigFinal** operation

adds the current state to the initial one (feed-forward) and compresses the output by XORing columns together to produce the new chaining value (see Figure 9.3).

The mode of operation chosen for ECHO is the HAIFA framework [BD06], which fixes some issues of the famous Merkle-Damgård construction. In more detail, the salt parameter of ECHO is used directly in the HAIFA mode of operations to differentiate each new message block and prevent fixed points. It is to be noted that even if the salt is a known parameter, then the previously known attacks on the Merkle-Darmgård construction are also prevented in the HAIFA framework thanks to an additionnal counter parameter incorporated in each compression function calls

Two versions of the hash function ECHO have been submitted to the SHA-3 competition: ECHO-256 and ECHO-512, which share the same state size (2048 bits) and inner permutation P, but inject messages of size 1536 or 1024 bits respectively in the compression function. Note that the message is padded by adding a single 1 followed by zeros to fill up the last message block. The last 18 bytes of the last message block always contain the 2-byte hash output size, followed by the 16-byte message length.

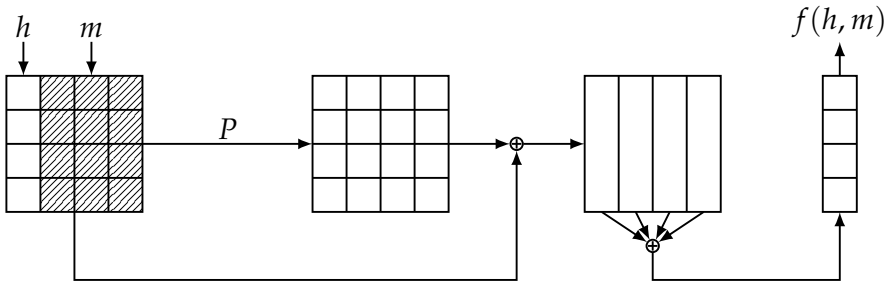


Figure 9.3: Compression function of ECHO-256.

By using the notation $s = [C_0, C_1, C_2, C_3]$ to denote a state s where the four columns are named C_i, the final compression step **BigFinal** can be described by (see also Figure 9.3 and Figure 9.4):

$$f(h,m) = Compress\Big(P(h||m) \oplus (h||m)\Big)$$

$$\text{where: } \begin{cases} Compress\Big([C_0, C_1, C_2, C_3]\Big) & = C_0 \oplus C_1 \oplus C_2 \oplus C_3 \text{ for ECHO-256,} \\ Compress\Big([C_0, C_1, C_2, C_3]\Big) & = \Big[C_0 \oplus C_2,\ C_1 \oplus C_3\Big] \text{ for ECHO-512.} \end{cases}$$

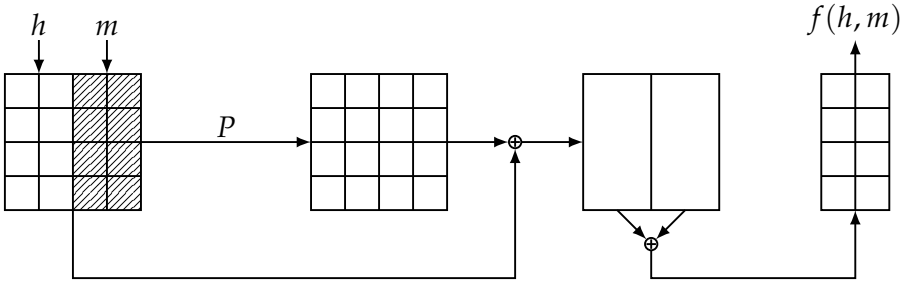


Figure 9.4: Compression function of ECHO-512.

Notation

We name each state during the application of the ECHO permutation: we start with S0, where the current chaining value (or IV[1]) and the message are combined and end the first round after eight transformations in S8. To refer to the AES-state at row i and column j of a particular ECHO-state Sn, we use the notation Sn$[i, j]$. Additionally, we introduce the term *column-slice* to refer to a thin column of size 16×1 of the ECHO state. Finally, in an AES-state, we consider four diagonals (from 0 to 3): diagonal $j \in [0, 3]$ consists in the four elements $(i, i + j \pmod 4)$, with $i \in 0, \ldots, 3$.

9.1.2 Alternative description

For an easier description of some of the following attacks, we use an equivalent description of one round of the ECHO permutation (see Figure 9.5).

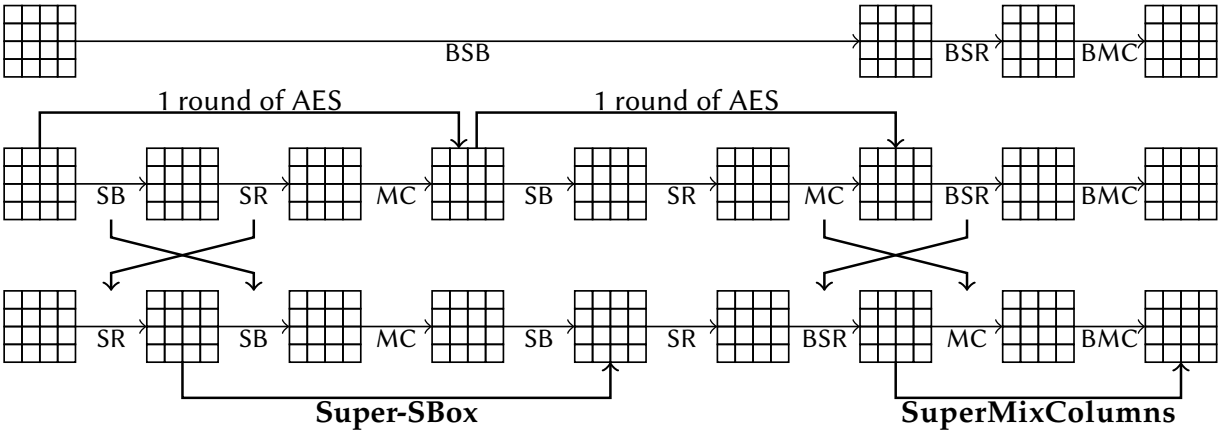


Figure 9.5: Alternative view of one round of the inner permutation of ECHO.

First, we swap the **BigShiftRows** transformation with the **MixColumns** transformation of the second AES round. Second, we swap **SubBytes** with **ShiftRows** of the first AES round. Swapping these operations does not change the computational result of ECHO and similar alternative descriptions have already been used in the analysis of AES. Hence, one round of ECHO results in the two transformations **Super-SBox** (**SubBytes** $\circ$ **MixColumns** $\circ$ **SubBytes**) and **SuperMixColumns** (**BigMixColumns** $\circ$ **MixColumns**), which are only separated by byte-shuffling operations, that we may call **SuperShiftRows** (**ShiftRows** $\circ$ **BigShiftRows**).

9.1.2.1 Super-SBox

The **Super-SBox** has first been analyzed by Daemen and Rijmen in [DR06b] to study two rounds of AES and has been independently used by Lamberger et al. in [LMR+09] and Gilbert and Peyrin in [GP10] to analyze AES-based hash functions. We have already described this non-linear construction in Section 7.2.

[1] Initial Vector.

9.1.2.2 SuperMixColumns

In a similar way, by permuting the **BigShiftRows** transformation with the parallel **MixColumns** transformations of the second AES round, a new "super" linear operation has been introduced by Martin Schläffer in [Sch10], which works on column-slices of size 16×1.

The matrix of the **SuperMixColumns** transformation is defined as the Kronecker product (or tensor product) of $\mathbf{M}$ with itself, $\mathbf{M}$ being the matrix of the **MixColumns** operation in the AES:

$$
\begin{aligned}
\mathbf{M}_{SMC} &= \mathbf{M} \otimes \mathbf{M} \\
&= \begin{bmatrix} 2\mathbf{M} & 3\mathbf{M} & \mathbf{M} & \mathbf{M} \\ \mathbf{M} & 2\mathbf{M} & 3\mathbf{M} & \mathbf{M} \\ \mathbf{M} & \mathbf{M} & 2\mathbf{M} & 3\mathbf{M} \\ 3\mathbf{M} & \mathbf{M} & \mathbf{M} & 2\mathbf{M} \end{bmatrix} \\
&= \begin{bmatrix}
4 & 6 & 2 & 2 & 6 & 5 & 3 & 3 & 2 & 3 & 1 & 1 & 2 & 3 & 1 & 1 \\
2 & 4 & 6 & 2 & 3 & 6 & 5 & 3 & 1 & 2 & 3 & 1 & 1 & 2 & 3 & 1 \\
2 & 2 & 4 & 6 & 3 & 3 & 6 & 5 & 1 & 1 & 2 & 3 & 1 & 1 & 2 & 3 \\
6 & 2 & 2 & 4 & 5 & 3 & 3 & 6 & 3 & 1 & 1 & 2 & 3 & 1 & 1 & 2 \\
2 & 3 & 1 & 1 & 4 & 6 & 2 & 2 & 6 & 5 & 3 & 3 & 2 & 3 & 1 & 1 \\
1 & 2 & 3 & 1 & 2 & 4 & 6 & 2 & 3 & 6 & 5 & 3 & 1 & 2 & 3 & 1 \\
1 & 1 & 2 & 3 & 2 & 2 & 4 & 6 & 3 & 3 & 6 & 5 & 1 & 1 & 2 & 3 \\
3 & 1 & 1 & 2 & 6 & 2 & 2 & 4 & 5 & 3 & 3 & 6 & 3 & 1 & 1 & 2 \\
2 & 3 & 1 & 1 & 2 & 3 & 1 & 1 & 4 & 6 & 2 & 2 & 6 & 5 & 3 & 3 \\
1 & 2 & 3 & 1 & 1 & 2 & 3 & 1 & 2 & 4 & 6 & 2 & 3 & 6 & 5 & 3 \\
1 & 1 & 2 & 3 & 1 & 1 & 2 & 3 & 2 & 2 & 4 & 6 & 3 & 3 & 6 & 5 \\
3 & 1 & 1 & 2 & 3 & 1 & 1 & 2 & 6 & 2 & 2 & 4 & 5 & 3 & 3 & 6 \\
6 & 5 & 3 & 3 & 2 & 3 & 1 & 1 & 2 & 3 & 1 & 1 & 4 & 6 & 2 & 2 \\
3 & 6 & 5 & 3 & 1 & 2 & 3 & 1 & 1 & 2 & 3 & 1 & 2 & 4 & 6 & 2 \\
3 & 3 & 6 & 5 & 1 & 1 & 2 & 3 & 1 & 1 & 2 & 3 & 2 & 2 & 4 & 6 \\
5 & 3 & 3 & 6 & 3 & 1 & 1 & 2 & 3 & 1 & 1 & 2 & 6 & 2 & 2 & 4
\end{bmatrix}.
\end{aligned}
$$

As described previously in Section 4.2, the matrix $\mathbf{M}$ represents a maximum distance separable (MDS) code: it has good diffusion properties since its branch number, i.e. the sum of input and output active bytes, will always be 0 or greater than 5. However, Martin Schläffer noted in [Sch10] that $\mathbf{M}_{SMC}$ is not a MDS matrix and its branch number is only 8, and not 17. This surprising linear behavior affects the security of the primitive as the diffusion is not as perfect as it could be.

From this observation, it is possible to build sparse truncated differentials (Figure 9.6) where there are only 4 active bytes in both the input and the output slices of the transformation. The differential characteristic $4 \rightarrow 16 \rightarrow 4$ holds with probability 2^{-24}, which reduces to 2^8 the number of valid differentials, among the 2^{32} existing ones.

For a given position of output active bytes, valid slices of differences actually lie in a subspace of dimension one. In particular, in order for slice s, $s \in \{0, 4, 8, 12\}$, to follow the

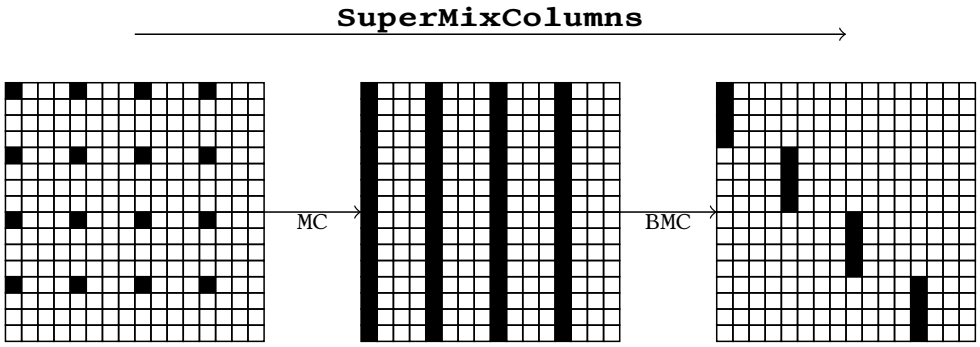


Figure 9.6: The **SMC** layer in the particular case of the truncated differential $4 \to 16 \to 4$ for a single slice. This figure shows four parallel slices of the type $4 \to 16 \to 4$. Black bytes are active where white ones are inactive.

truncated differential $4 \to 16 \to 4$ of Figure 9.6, we need to pick each slice of differences in the one-dimensional subspace spanned by the vector v_s, where:

$$\begin{aligned} v_0 &= [\mathrm{E000}\ \ 9000\ \ \mathrm{D000}\ \ \mathrm{B000}]^{\mathrm{T}}, \\ v_4 &= [\mathrm{B000}\ \ \mathrm{E000}\ \ 9000\ \ \mathrm{D000}]^{\mathrm{T}}, \\ v_8 &= [\mathrm{D000}\ \ \mathrm{B000}\ \ \mathrm{E000}\ \ 9000]^{\mathrm{T}}, \\ v_{12} &= [9000\ \ \mathrm{D000}\ \ \mathrm{B000}\ \ \mathrm{E000}]^{\mathrm{T}}. \end{aligned}$$

While this new approach of the combined linear layers allows to build sparser truncated differentials, it also caused erroneous conclusions when it has been used in the original article [Sch10]. Namely, at the end of the attack, where two partial solutions need to be merged to get a solution for the whole differential characteristic, everything relies on this critical transformation: we need to solve 16 systems of linear equations that do not always have solution. This is due to the fact that several sub-matrices of $\mathbf{M}_{SMC}$ used in the system resolutions do not have full rank.

In more detail, the error in [Sch10] boils down to the linear equation

$$\begin{aligned} \mathbf{M_{SMC}} \left[a_0\ x_0\ x_1\ x_2\ a_1\ x_3\ x_4\ x_5\ a_2\ x_6\ x_7\ x_8\ a_3\ x_9\ x_{10}\ x_{11}\right]^{\mathrm{T}} = \\ \left[b_0\ b_1\ b_2\ b_3\ *\ *\ *\ *\ *\ *\ *\ *\ *\ *\ *\ *\right]^{\mathrm{T}}, \end{aligned} \tag{9.1}$$

where a_i and b_i are known values, x_i are unknowns and $*$ is any value in $\mathrm{GF}(2^8)$. It has been assumed in [Sch10] this equation has solutions for any values of the constants a_i, b_i. We show in the following that the structure of the matrix $\mathbf{M}_{SMC}$ makes the analysis more complex, and some constants in particular are not independent but are expressed as linear functions of the others.

By discarding the lines from $\mathbf{M}_{SMC}$ associated to the unspecified values $*$ of the Equation 9.1,

we can rewrite it as:

$$\begin{bmatrix} 6 & 2 & 2 & 5 & 3 & 3 & 3 & 1 & 1 & 3 & 1 & 1 \\ 4 & 6 & 2 & 6 & 5 & 3 & 2 & 3 & 1 & 2 & 3 & 1 \\ 2 & 4 & 6 & 3 & 6 & 5 & 1 & 2 & 3 & 1 & 2 & 3 \\ 2 & 2 & 4 & 3 & 3 & 6 & 1 & 1 & 2 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} x_0 \\ \vdots \\ x_{11} \end{bmatrix} = \begin{bmatrix} 4 & 6 & 2 & 2 \\ 2 & 3 & 1 & 1 \\ 2 & 3 & 1 & 1 \\ 6 & 5 & 3 & 3 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \oplus \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

Then, by transforming this system of linear equations into reduced row echelon form, we get:

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} x_0 \\ \vdots \\ x_{11} \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix},$$

where $c_0, \ldots, c_3$ are linear functions of $a_0, \ldots, a_3, b_0, \ldots, b_3$, and the fourth equation gives a linear relation between them such that the system is consistent. In this particular case, the relation is:

$$2a_0 + 3a_1 + a_2 + a_3 = 14b_0 + 11b_1 + 13b_2 + 9b_3.$$

As long as this relation is verified, the system is consistent, but for random values of the constants, there is no solution with probability 255/256.

In total, there are 16 possibilities to generalize Equation 9.1, by combining the positions of the constants. Namely, we denote by $\alpha \in \{0, \ldots, 3\}$ the four scenarios $(\alpha, 4+\alpha, 8+\alpha, 12+\alpha)$ for the positions of the a_i, and by $\beta \in \{0, \ldots, 3\}$ the four scenarios $(4\beta, 4\beta+1, 4\beta+2, 4\beta+3)$ for the positions of the b_i. We denote $\mathcal{S}_{\alpha,\beta}$ the resulting system, such that Equation 9.1 would refer to $\mathcal{S}_{0,0}$. The following Table 9.1 lists all the linear constraints for all the 16 scenarios.

α	β	Condition for $\mathcal{S}_{\alpha,\beta}$
0	0	$14b_0 + 11b_1 + 13b_2 + 9b_3 = 2a_0 + 3a_1 + a_2 + a_3$
1	0	$11b_0 + 13b_1 + 9b_2 + 14b_3 = 2a_0 + 3a_1 + a_2 + a_3$
2	0	$13b_0 + 9b_1 + 14b_2 + 11b_3 = 2a_0 + 3a_1 + a_2 + a_3$
3	0	$9b_0 + 14b_1 + 11b_2 + 13b_3 = 2a_0 + 3a_1 + a_2 + a_3$
0	1	$14b_0 + 11b_1 + 13b_2 + 9b_3 = a_0 + 2a_1 + 3a_2 + a_3$
1	1	$11b_0 + 13b_1 + 9b_2 + 14b_3 = a_0 + 2a_1 + 3a_2 + a_3$
2	1	$13b_0 + 9b_1 + 14b_2 + 11b_3 = a_0 + 2a_1 + 3a_2 + a_3$
3	1	$9b_0 + 14b_1 + 11b_2 + 13b_3 = a_0 + 2a_1 + 3a_2 + a_3$
0	2	$14b_0 + 11b_1 + 13b_2 + 9b_3 = a_0 + a_1 + 2a_2 + 3a_3$
1	2	$11b_0 + 13b_1 + 9b_2 + 14b_3 = a_0 + a_1 + 2a_2 + 3a_3$
2	2	$13b_0 + 9b_1 + 14b_2 + 11b_3 = a_0 + a_1 + 2a_2 + 3a_3$
3	2	$9b_0 + 14b_1 + 11b_2 + 13b_3 = a_0 + a_1 + 2a_2 + 3a_3$
0	3	$14b_0 + 11b_1 + 13b_2 + 9b_3 = 3a_0 + a_1 + a_2 + 2a_3$
1	3	$11b_0 + 13b_1 + 9b_2 + 14b_3 = 3a_0 + a_1 + a_2 + 2a_3$
2	3	$13b_0 + 9b_1 + 14b_2 + 11b_3 = 3a_0 + a_1 + a_2 + 2a_3$
3	3	$9b_0 + 14b_1 + 11b_2 + 13b_3 = 3a_0 + a_1 + a_2 + 2a_3$

Table 9.1: Linear constraint required to ensure the consistency of the system of linear equations $\mathcal{S}_{\alpha,\beta}$.

9.1.3 Current cryptanalysis

We give in this section the most notable cryptanalytic results published on the building blocks of the ECHO-256 hash function. The larger ECHO-512 hash function has received significantly less amount of cryptanalysis, but we note that all the attacks listed here can probably be adapted in some ways to this larger case.

Table 9.2: Best known cryptanalytic results on ECHO-256 to date.

Target	Rounds	Time	Memory	Generic	Type	Reference
IP	7	2^{384}	2^{64}	2^{1024}	Dist.	[MPRS09]
	8	2^{768}	2^{512}	2^{1024}	Dist.	[GP10]
	8	2^{182}	2^{37}	2^{256}	Dist.	[SLW+10]
	8	2^{151}	2^{67}	2^{256}	Dist.	[NP11]
CF	4	2^{52}	2^{16}	2^{256}	Collision	Section 9.2.1, [JF11]
	6	2^{193}	2^{128}	2^{256}	Collision	Section 9.2.5, [JNPS11a]
	6	2^{160}	2^{128}	2^{256}	Collision *	Section 9.2.6, [JNPS11b]
	7	2^{193}	2^{128}	2^{240}	Dist.	Section 9.2.4, [JNPS11a]
	7	2^{160}	2^{128}	2^{240}	Dist. *	Section 9.2.6, [JNPS11b]
HF	4	2^{64}	2^{64}	2^{128}	Collision	Section 9.2.2, [JNPS11b]
	5	2^{112}	$2^{85.3}$	2^{128}	Collision	Section 9.2.3, [JNPS11a]

IP: Internal Permutation.
CF: Compression Function.
HF: Hash Function.
Dist.: Distinguisher.
* : Chosen salt

9.2 Attacks on ECHO-256

In this section, we present the most efficient cryptanalytic results that has been published on ECHO. The earliest results working at the AES-state level like done in [GP10, MPRS09] are not considered here. Indeed, the latest results have shown that an analysis conducted at the byte level yields much lower time complexities as we can control more precisely the propagation of differences.

We describe several attacks on both the compression function and the hash function.

1. In Section 9.2.1, we give a collision attack on the 4-round compression function that has been published in a paper co-authored by Pierre-Alain Fouque at FSE 2011 in [JF11], which corrects the previous attack from [Sch10].
2. In Section 9.2.2, we extend the previous attack from the compression function reduced to 4 rounds to the hash function reduced to 4 rounds [JNPS11b].
3. In Section 9.2.3, we show a collision attack on the hash function reduced to 5 rounds that has been published in an article written with María Naya-Plasencia and Martin Schläffer at SAC 2011 in [JNPS11a].
4. In Section 9.2.4, we present a distinguishing attack on the compression function reduced to 7 rounds, and has been published in the same article as the previous one.
5. In Section 9.2.5, we extend the previous attack with a free-start collision attack on the 6-round compression function.
6. In Section 9.2.6, we show how the salt can be used to decrease the time complexity of the 7-round distinguisher (near-collisions) on the compression function and to reach a collision attack on the compression function reduced to 6 rounds [JNPS11b].

9.2.1 Collision attack on the 4-round compression function

This attack has been published at FSE 2011, mostly to emphasize an error in a previous paper and corrects it by mounting an collision attack on the ECHO-256 compression function reduced to four rounds.

The new attack exploits ideas from the rebound strategy, but uses them a bit differently in almost all the steps of the attack. The intuition is simple and is justified by the two levels of ECHO: one state of ECHO is built on top of 16 AES states, which are independent at some point during the inner permutation. In the original rebound technique, the randomization of differences around the non-linear layer is performed on the *same* state, whereas in ECHO, we can randomize one state without affecting the others thanks to this independency.

In more detail, rather than randomizing the differences in *both* sides of the non-linear layer, we can use some of the independent states to randomize both the values *and* n differences on one side, propagate them towards the other side, and therefore solve linear equations to deduce the n independent differences on this side. Now, we can apply the original rebound technique on a problem reduced by n.

9.2.1.1 Truncated differential characteristic

The truncated differential characteristic we use (see Figure 9.7) is mostly borrowed from [Sch10] and counts 418 active S-Boxes for the ECHO-permutation reduced to 4 rounds. In comparison to the characteristic from [Sch10], we increase significantly the number of active S-Boxes in the first round to decrease the time complexity of the attack. The attack being quite technical, colors have been used in order to improve the reader's understanding of the attack.

We begin by describing an elementary algorithm used to find paired inputs to sparse differentials in the **Super-SBox**, and we continue by the explanation of the complete attack.

9.2.1.2 Super-SBox sparse differentials

In the differential characteristic described, there are many differential transitions through the **Super-SBox** of the third round where input differences are reduced to *one* active byte. We are then interested in differential transitions such as the one described in Figure 9.8. For this kind of transition, we could find a paired input following a given differential by using the DDT of the **Super-SBox** but this would require about 2^{64} operations and 2^{64} memory units to store the information. Even if in this particular case, one could compute and store smaller tables of 2^{40} elements for the four possible positions of active bytes, we show in the following how to construct a pair of columns satisfying this differential in about 2^7 computations.

We denote the input difference of Figure 9.8 by $\Delta_i = [\delta_i, 0, 0, 0]^{\mathrm{T}}$, which is reduced to a single active byte δ_i arbitrarily set at position 0 and let the output difference be $\Delta_o = [\delta_o^1, \delta_o^2, \delta_o^3, \delta_o^4]^{\mathrm{T}}$. We want to find a pair of AES-columns (c_1, c_2) following the differential $\Delta_i \rightarrow \Delta_o$; that is:

$$\begin{aligned} & c_1 \oplus c_2 = \Delta_i, \\ \text{and:} \quad & \mathbf{Super\text{-}SBox}(c_1) \oplus \mathbf{Super\text{-}SBox}(c_2) = \Delta_o. \end{aligned}$$

We can already estimate the number of solutions of this problem. With fixed input and output differences, there are exactly 2^{32} possible input pairs, which are transformed into 2^{32} output pairs with an assumed uniform distribution of differences. This hypothesis is classical and justified by the good differential properties of the AES S-Box. Consequently, the particular output difference given as parameter is reached with probability 2^{-32}, which makes on average $2^{32-32} = 1$ solution to the problem.

To find this solution efficiently, we start by precomputing the differential distribution table of the AES S-Box in 2^{16} computations. The differential properties of the AES S-Box (Theorem 4.2) restrict the number of output differences of the first **SubBytes** layer to $2^7 - 1$. Denoting δ_i' one of the output differences of this layer and λ the associated value such that

$$S^{-1}(\lambda) + S^{-1}(\lambda + \delta_i') = \delta_i, \tag{9.2}$$

we can propagate this difference $\Delta_i' = [\delta_i', 0, 0, 0]^{\mathrm{T}}$ linearly to learn the four differences at the input of the second **SubBytes** layer. We note $\Delta_o' = \mathbf{MC}(\Delta_i') = [\delta_o^{1\prime}, \delta_o^{2\prime}, \delta_o^{3\prime}, \delta_o^{4\prime}]^{\mathrm{T}}$ those differences. Here, both the input and the output differences are known and the four differentials $\delta_o^{i\prime} \rightarrow \delta_o^i$ in the S-Box have non-zero differential probability with probability approximately

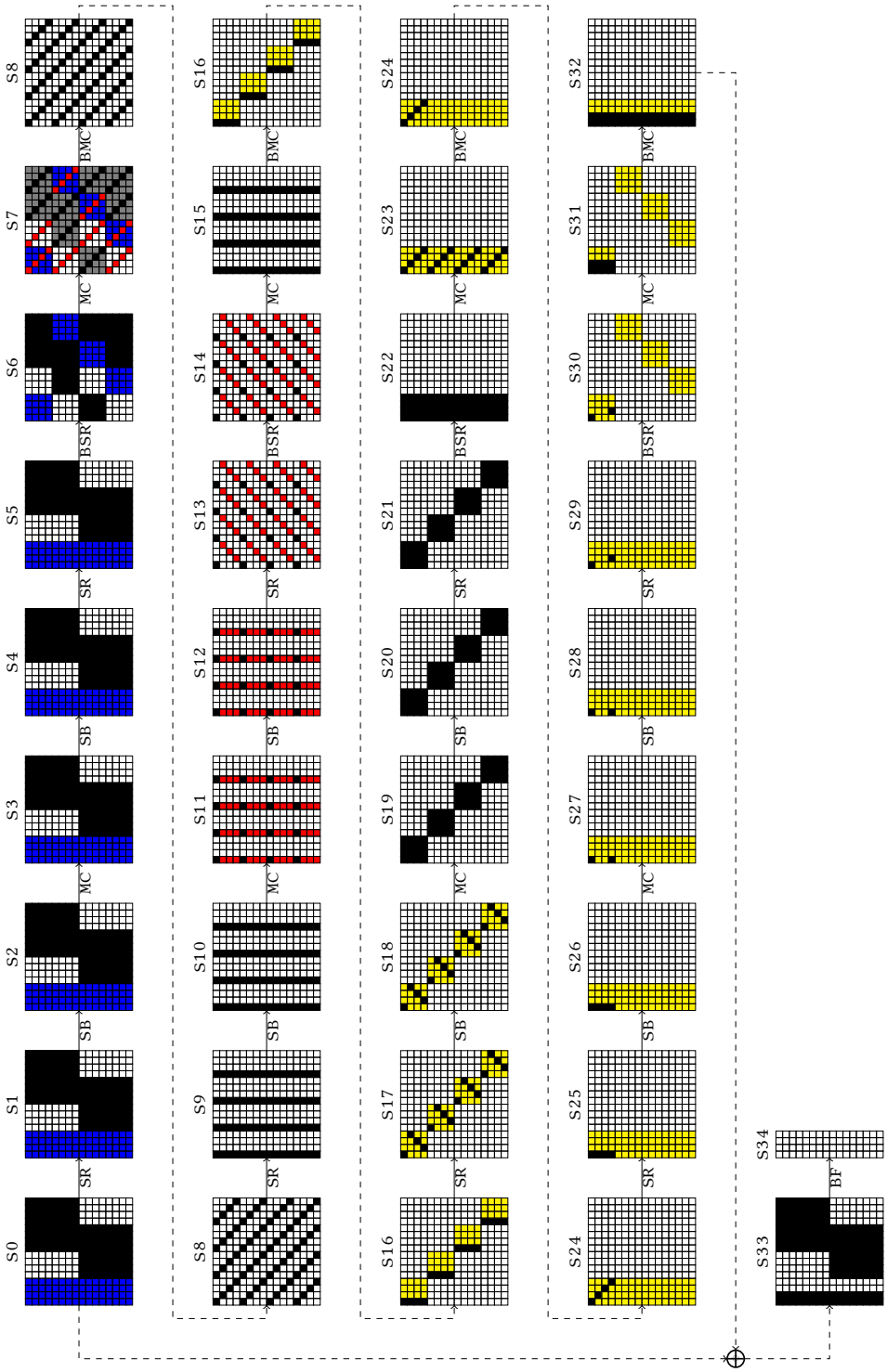


Figure 9.7: The truncated differential characteristic used in this attack on the ECHO-256 compression function reduced to four rounds. To find a valid pair of messages, we split the characteristic into two parts: the first inbound between S7 and S14 (red bytes) and the second inbound between S16 and S31 (yellow bytes). Black bytes are the only active bytes, blue bytes come from the chaining value and gray bytes in the first round are set to get a collision (or a near-collision) in the compression step.

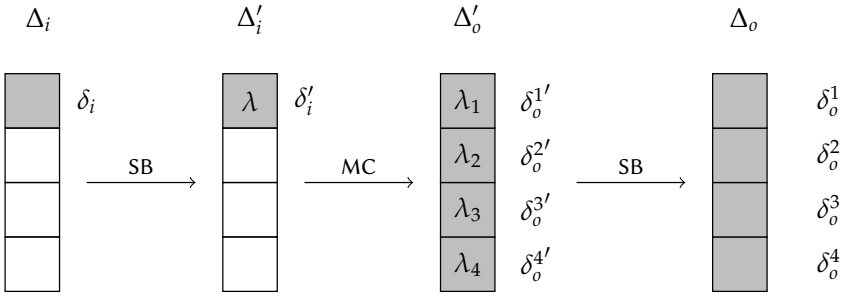


Figure 9.8: A sparse differential in the **Super-SBox** with only one active input byte.

2^{-4}. Since we can restart with $2^7 - 1$ different δ'_i, we get approximately 2^3 valid differential transitions. Each of these transitions fixes the underlying values, noted $\lambda_1, \lambda_2, \lambda_3, \lambda_4$.

At this point, all intermediate differences conform to the characteristic, but in terms of values, we need to ensure that λ is consistent with $\lambda_1, \lambda_2, \lambda_3, \lambda_4$. To check this, we exhaust the $2^4 - 1$ valid vectors of values we can build by interchanging λ_i and $\lambda_i + \delta_o^{i\prime}$. All in all, among the 2^{3+4} vectors of values we can build, only a fraction 2^{-8} satisfies the 8-bit condition on λ. This means that the considered differential transition $\Delta_i \rightarrow \Delta_o$ through the **Super-SBox** occurs with probability 2^{-1} and if the transition exists, we can recover an actual AES-column pair in about 2^7 computations. We note that the cost of generating the 2^4 vectors when the differentials exist is absorbed by all the cases where the differentials are impossible.

We note that our strategy to adapt the general case of the **Super-SBox** when the input (or output) differences verify additional properties has also been independently considered by Sasaki et al. in [SLW+10] that we have described in Section 7.2.2.2.

9.2.1.3 Finding a message pair

To find a pair of messages that follows the differential characteristic of Figure 9.7, our attack splits the whole characteristic into two distinct inbound phases and merges them at the end. In the sequel, we refer to these two parts as *first inbound* and *second inbound*. The attack of Schläffer in [Sch10] proceeds similarly but uses the basic rebound attack technique in the two inbounds. We reuse this idea of finding pairs of messages conforming to partial truncated parts but most of our new techniques avoid the rebound attack on the **Super-SBox**.

Both inbounds are represented in Figure 9.7: the first one starts in S7 and ends in S14 and fixes the red bytes of the two messages, whereas the second one starts at S16 until the end of the four rounds in S31 and fixes the yellow bytes. We also represent the chaining value in the first round of the characteristic by the blue bytes. We now give an overview of the attack, and then describe to all the steps in a more detailed way.

Step 1. We begin by finding a pair of **BigColumns** satisfying the truncated characteristic reduced to the first **BigColumns** between S7 and S12. This is done with a randomized AES-state on the column, used to solve linear equations giving all differences between S7 and S9. Indeed, differences between S7 and S9 for the first column only depend on the four differences

in S7[2,0][2]. Then, we search for valid differential transitions through the AES S-Box between S9 and S10 to finally deduce a good pair of **BigColumns**.

Step 2. Once we have solved the first **BigColumn**, we can deduce all differences between S12 and S16. Indeed, the sparse **SuperMixColumns** transition imposes strong constraints on the differences as exposed in Section 9.1.2.2, and we thus have a starting point to find a message pair for the second inbound of the whole truncated characteristic: namely, states between S16 and S31 (yellow bytes). To do so, the idea is similar as in Step 1: since all differences between S20 and S24 only depend on the four differences of S24[3], we can use a randomized AES-column c in S18 to get four independent linear equations in S20 and thus deduce all differences between S20 and S24. Then, we search for input values for the 15 remaining sparse differentials in the **Super-SBox** with the dedicated efficient algorithm.

Step 3. First, we find a pair of **BigColumns** between S7 and S12 similarly as before, independently for the second and the third **BigColumns**. We enforce the linear constraints imposed by the sparse differential in the **SuperMixColumns** and merge the solutions from the first and second inbounds by deterministically deducing the last pair of **BigColumns**. Then, we verify probabilistically that the truncated characteristic for the **BigColumn** is verified, which holds with probability 2^{-32}.

Step 4. We reach the collision in the feed-forward of the compression function by consuming the remaining freedom degrees in the gray bytes of state S7 to force a zero sum between S0 and S32. The attack time complexity can be made practical at this point by discarding the zero-difference requirement of the collision on some bytes to produce near-collisions.

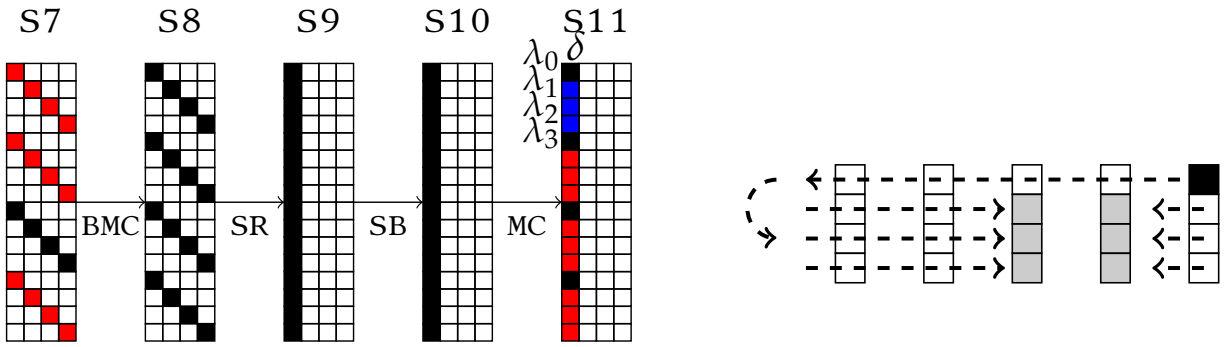
9.2.1.4 Step 1 - Partial first inbound

This step finds a pair of **BigColumns** satisfying the truncated differential of Figure 9.9a. The reason why we consider the first column separately is twofold: first, this allows to enforce the extra linear relations imposed by the sparse differential in the **SuperMixColumns** between S14 and S16, and also to avoid the rebound technique on the **Super-SBox** which would be less efficient than our method.

As described on Figure 9.9b, we begin by choosing random values $(\lambda_0, \lambda_1, \lambda_2, \lambda_3)$ for the first AES-column of S11[0,0] (blue bytes), and a random value δ for the difference in that column. We note that we have about $2^{8\times 4} \times (2^8 - 1) \approx 2^{40}$ ways of selecting this pair of AES-columns. Since both values and differences are known, we can propagate them backwards until S8, and therefore compute the differences in S8[0,0]. As there is only one active byte per slice in the considered **BigColumn** of S7, each of the associated four slices of S8 lies in a subspace of dimension one. Therefore, solving four simple linear systems leads to the determination of the 12 other differences of S8.

Therefore, in the active slice of S9 of Figure 9.9 at the input of the **SubBytes** layer, the four first paired bytes have known values and differences, whereas in the 12 other positions, only differences are set. We now randomize the three remaining differences in S11 and linearly compute the differences in the corresponding states in S10. With probability 2^{-12}, we find good

[2]Linear relations can be deduced by linearly propagating differences in S7[2,0] forwards until S9.
[3]Linear relations can be deduced by linearly propagating the four differences of S24[0,0] backwards until S20.



(a) Truncated differential. **(b)** Flow of AES states.

Figure 9.9: Truncated differential characteristic (a) used for the first inbound of the attack for one **BigColumn**. We represent on (b) the order in which AES states are randomized (black) or deduced by a small rebound attack (gray).

values for these 12 paired byte pairs, so we need to repeat this process about 2^{12} times with new differences in S11.

All in all, we can restart about $2^{40} \times (2^8 - 1)^3 \approx 2^{64}$ times, but we only need 2^{12} operations to find a paired **BigColumn** that conforms to the truncated differential. We note that as soon as we get one solution, we can construct 2^{12} of them by swapping elements. Consequently, we can get up to 2^{64} solutions, but the cost to get on is 2^{12} computations.

9.2.1.5 Step 2 - Second inbound

We now get a partial message pair conforming to the first inbound of the truncated characteristic reduced to a single **BigColumn**. Rather than completing this partial message pair for the three other active slices in S12, we now find a message pair conforming to the second inbound of the truncated characteristic, located in the third round from S16 to S24 (yellow bytes).

Indeed, the knowledge of a single active slice pair of S12 is sufficient to get a starting point to find a message pair for the second inbound, i.e. yellow bytes. This is due to the sparse transition in the **SuperMixColumns**: as explained in Section 9.1.2.2, differences in S14 lie in subspaces of dimension 1. In particular, once a slice pair for the first slice of S12 is known and computed forwards to S14 (black and red bytes on Figure 9.10), there is no more choice for the other differences in S14.

Consequently, all differences between S12 and S17 have been determined by linearity of involved transformations, and in particular, we know the differences of states S16 and S17.

At this point, only the input differences of the 16 **Super-SBoxes** of the third round are known in state S17. We note that all operations between S20 and S24 are linear, so that any differences in those states can be expressed as a linear function of any four differences, for example the four differences in the first AES-column of S20[0,0] (black bytes on Figure 9.11).

We start by randomizing the difference δ in the state S20, at the output of the **SubBytes** layer where the corresponding input difference has already been fixed previously. As a consequence, we determine in average one value λ_0 for that byte. Then, we randomize the values of the

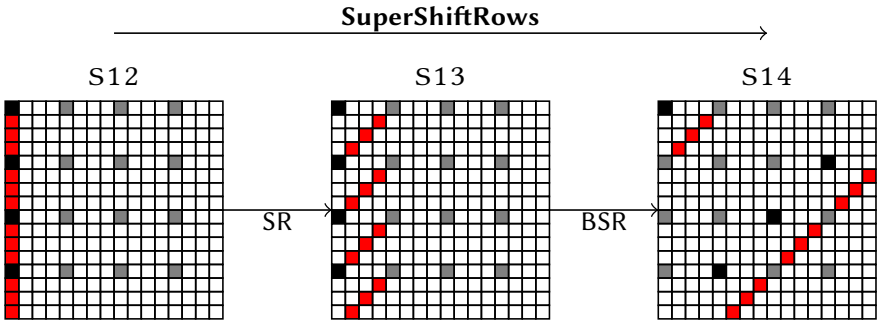


Figure 9.10: The **SuperShiftRows** layer where only the values and differences of the first slice of S12 are known (black and red bytes).

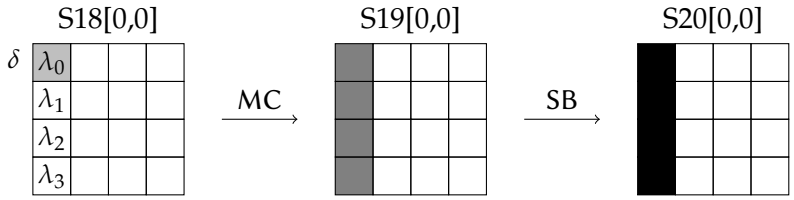


Figure 9.11: The **MixColumns** and **SubBytes** transitions on the first AES-column between S18[0,0] and S20[0,0].

inactive bytes of the same column, that we denote λ_1, λ_2 and λ_3. We compute the image of this paired AES-column forwards until S20, where we compute the four differences, and as stated before, this allows to compute all the differences between S20 and S24.

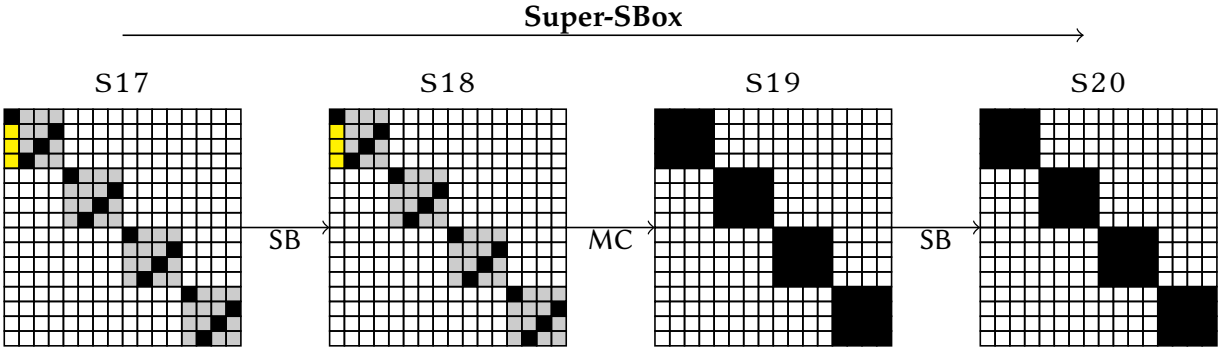


Figure 9.12: Last step to get a partial message pair conforming to the second inbound of the characteristic: finding the 15 remaining AES-columns using the sparse **Super-SBox** properties. Black bytes are active and yellow bytes have already been defined in the previous step, as well as differences of the first AES-column of the first AES-state. Gray bytes are inactive and the target of this step.

We now have 15 sparse differentials in the **Super-SBox** between S17 and S20 where all differences have been previously set. As described in Section 9.2.1.2, the 15 transitions have a non-zero differential probability simultaneously with probability 2^{-15} and by the efficient algorithm we have proposed, we can recover the 15 AES-column pairs in parallel in about 2^7 simple operations. We expect to find a solution after 2^{15} retries, for an overall cost of $2^{15+7} = 2^{22}$ simple operations. We note that for a single starting point in S17, we can get up to $(2^7 - 1) \times 2^{8\times 3} \times 2^{-15} \approx 2^{16}$ solutions for that part (the yellow bytes).

In the collision attack on the compression function, we further extend this step by probabilistically filtering the active bytes in the **MixColumns** transition between S26 and S27. Among the 2^{16} partial pairs of messages we can build that follow the truncated characteristic between S16 and S26, we expect only one to verify the $4 \rightarrow 2$ transition in the **MixColumns**. If found, this pair conforms to the truncated characteristic until the end of the four rounds. We note that we reduce to two active bytes and not one or three as this is the best tradeoff to lower the overall time complexity of the collision attack.

9.2.1.6 Step 3 - Merging the two inbounds

Here, we merge the solutions from the first and second inbounds of the characteristic. We begin by repeating step 1 for the second and the third **BigColumns** between S7 and S11. Thus, what we have found so far is a partial pair of messages for the first inbound and a partial pair of messages that fixed the yellow bytes in the second inbound. To merge them, we need to consider the extra linear conditions imposed at the sparse **SuperMixColumns** differential. In detail, we write independently 16 systems of linear equations like Equation 9.1 and we derive for each of the 16 slices the formal condition reported Table 9.1. In each of the 16 relations, all but one variables have been fixed by the previous steps. Indeed, the fourth paired **BigColumn** in the first inbound has been left unset to enforce the linear relations.

Consequently, we merge the solutions by deducing the missing parts of the partial pair of messages in the first inbound. As we have not controlled the truncated behavior of the last **BigColumn**, we then need to ensure that it conforms to the truncated characteristic of Figure 9.7. Namely, we want four transitions $3 \leftarrow 4$ between S8 and S7 in the **BigMixColumns**, which holds with probability $(2^{-8})^4 = 2^{-32}$. So we need to repeat all the previous steps about 2^{32} times, at a cost of about $2^{32} \times 2^4 = 2^{36}$ computations by generating a new paired **BigColumn** for the third column.

9.2.1.7 Step 4 - Reaching the collision

After four rounds, the reduced compression function applies the feed forward and XORs the four **BigColumns** together (**BigFinal**):

$$\begin{aligned} S33 &\leftarrow S0 \oplus S32, \\ h' &\leftarrow S33[*,0] \oplus S33[*,1] \oplus S33[*,2] \oplus S33[*,3]. \end{aligned}$$

This compression operation allows to build the differential characteristic such that differences would cancel out each other. As shown on Figure 9.13, states S0 and S32 XORed together lead to state S33 where there are three active AES-states in each row. In terms of differences, if each row sums up to zero, then we get a collision for the compression function in S34 after the **BigFinal**.

As we constructed the characteristic until now, in both S0 and S32, we still have freedom on the values (bytes in white): only differences in S32 located in the two first slices are known from the message pair conforming to the second inbound of the truncated characteristic. Since we want zero difference in h', these differences impose constraints on the two other active pair states per row in S0. Namely, for each row r of S0 where active AES states are located in

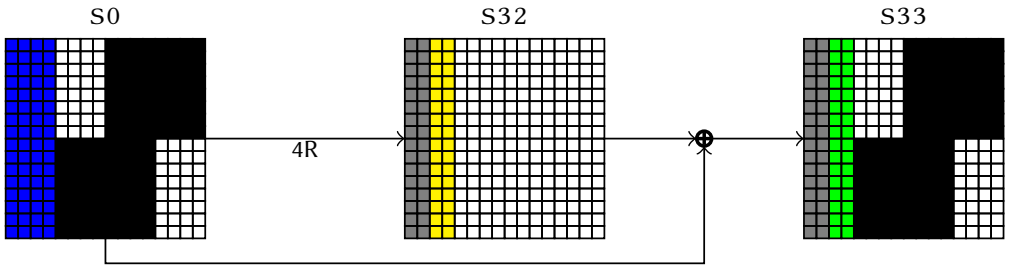


Figure 9.13: Feed-forward after the inner permutation reduced to 4 rounds. Gray bytes are active with known values and differences; black bytes are active with unknown values and differences; blue, yellow and green bytes are inactive with known values.

columns c_r and c'_r, we have

$$\Delta S0[r, c_r] \oplus \Delta S0[r, c'_r] = \Delta S32[r, 0].$$

Additionally, differences in S4 are known by linearly propagating the known differences from S7.

After the feed-forward, we reach a collision by canceling differences of each row independently: we describe the reasoning for an arbitrary row. We want to find paired values in the two active states of the considered row of S0, say (A, A') and (B, B'), such that they propagate with correct differences in S4, which are known, and with correct diagonal values (red bytes) in S7 after the **MixColumns**. In the sequel (Figure 9.14), we subscript the AES-state A by j to indicate that A_j is the AES-state A propagated until ECHO-state Sj with relevant transformations according to Figure 9.7.

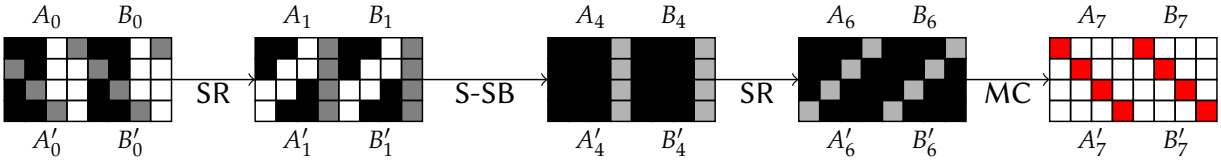


Figure 9.14: Propagation of the pairs of AES-states (A_i, A'_i) and (B_i, B'_i) in a single ECHO-row in the first round. Non-white bytes represent active bytes; those in S7 (in red) are the known values and differences from the message pair conforming to the first inbound of the truncated characteristic.

The known differences of S4 actually sets the output differences of the **Super-SBox** layer: namely,

$$A_4 \oplus A'_4 = \Delta_4$$
$$\text{and} \quad B_4 \oplus B'_4 = \Delta'_4,$$

where Δ_4 and Δ'_4 are the known differences in the considered row of S4, which are known. The constraint on the known diagonal values in A_7 and B_7 restricts the available freedom in the choice of the AES-columns of A_6 and B_6 (and linearly, to their equivalent A'_6 and B'_6 with diagonal values in A'_7 and B'_7) to reach the already-known diagonal values in S7 (red bytes), due to the first inbound as well.

An alternative way of stating this is: we can construct freely the three first columns of (A_4, A'_4) and (B_4, B'_4) and deduce deterministically the fourth ones with the next **MixColumns**

transition, since 4 out of 8 input or output bytes of **MixColumns** fix the 4 others. Furthermore, this means that if the three first columns of A_1, A'_1, B_1 and B'_1 are known, then we can learn the values of the remaining columns of S1 (bytes in gray).

We thus search for valid input values for the three first **Super-SBoxes** of S1: to do so, we randomize the two differences per AES-column in this state and get valid paired values with probability 2^{-1} in 2^{14} computations with respect to the output differences Δ_4, by guessing the two differences inside the **Super-SBox**. Consequently, we can deduce the differences of the same AES-columns in $B_1 \oplus B'_1$ to get a zero sum with S32 after the **BigFinal**. This holds with the same 2^{-1} probability, with respect to Δ'_4. Once we have the three differential transitions for the three first AES-columns of both AES-states, all the corresponding values are then known and we propagate them in A_6, A'_6, B_6 and B'_6 (black bytes). Since in S7, diagonal values are known, we deduce the remaining byte of each column in A_6, A'_6, B_6 and B'_6 (gray) and propagate them backwards until S1.

The final step defines the nature of the attack: to get a collision, we check if those constrained values cancel out in the feed-forward, which holds with probability 2^{-32}. Restarting with new random values in S1 and in parallel on the four rows, we find a collision in $2^{14}\,2^2\,2^{32} = 2^{48}$ computations. Indeed, we need to repeat 2^{32} times the search for valid paired input values for the **Super-SBox**, which is done in time 2^{14} and succeeds with probability 2^{-2}.

After we have found message pairs following both inbounds of the truncated differential characteristic so that the merge is possible, we need to finalize the attack by merging the two partial solutions. This means finding values for each white bytes in the truncated differential characteristic, and in particular, at the second **SuperMixColumns** transition between S14 and S16. For each of the 16 slices, we get a system of linear equations like Equation 9.1. In each solution set, each variable only depends on 3 others, and not on *all* the 11 others. This stems from the strong structure of the matrix $\mathbf{M}_{SMC}$. For example, in the first slice, we have:

$$\begin{aligned} L_0(x_0, x_3, x_6, x_9) &= c_0 \\ L_1(x_1, x_4, x_7, x_{10}) &= c_1 \\ L_2(x_2, x_5, x_8, x_{11}) &= c_2 \end{aligned}$$

where L_0, L_1, L_2 are linear functions and c_0, c_1, c_2 constants linearly deduced from the 8 known-values a_i and b_i, $0 \leq i \leq 3$, of the considered system. Consequently, by indirectly choosing values for gray bytes in S14, we set the values of half of the unknowns per slice. For example, the system for the first slice becomes:

$$\begin{aligned} L'_0(x_0, x_3) &= c'_0 \\ L'_1(x_1, x_4) &= c'_1 \\ L'_2(x_2, x_5) &= c'_2 \end{aligned}$$

where L'_0, L'_1, L'_2 are linear functions and c'_0, c'_1, c'_2 some constants. Those three equations are independent, which allows to do the merge in three steps: one on each pair of slices $(1, 5)$, $(2, 6)$ and $(3, 7)$ of S12. The Figure 9.15 represents in color only the first step, on the slice pair $(1, 5)$ of S12. We show in following that each of the three steps can be done in 2^{32} computations, by focusing on the first one.

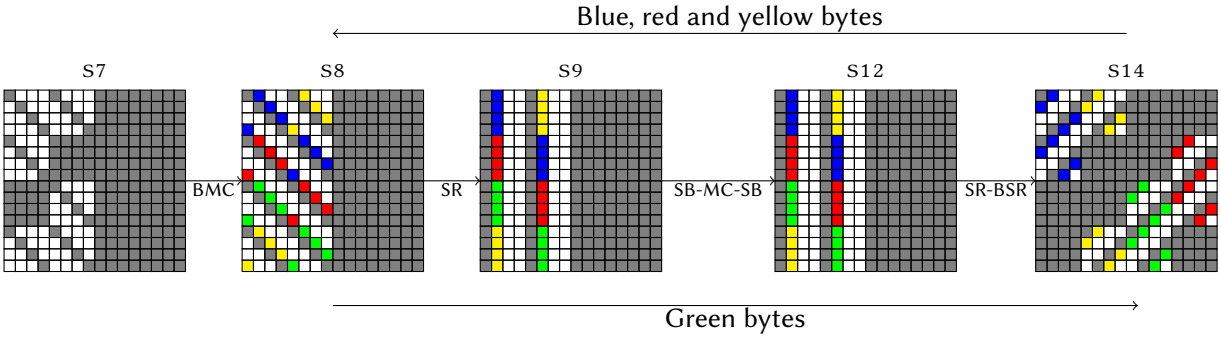


Figure 9.15: After randomization of states S7[1,3] and S7[2,2], all values of gray bytes are known. Colors show the flow of values in one step of the merging process.

Because of the dependencies between bytes within a slice in S14, any choice of blue bytes in S12[0,0] determines blue bytes on S12[1,1] (and the same for yellow and red bytes, Figure 9.15). In total, we can choose $\left(2^{8\times 4}\right)^3 = 2^{96}$ different values for the blue, yellow and red AES-columns of state S12. Since we are dealing with values, we propagate them backwards until S8. The **BigMixColumns** transition from S7 to S8 for these two slices imposes the 8 green values in S8[2,0] and S8[3,1]. Going forwards through the **Super-SBox**, we deduce green values in S14 and check whether the four pairs of green bytes satisfy the linear constraints in S14, which occur with probability $\left(2^{-8}\right)^4 = 2^{-32}$. We then have to restart with approximately 2^{32} new blue bytes and random yellow and red ones before satisfying the four constraints simultaneously.

After repeating this step for slices $(2,6)$ and $(3,7)$, we get a valid message pair that follows all the truncated differential characteristic of Figure 9.7.

9.2.1.8 Experimental verification

We have implemented every part of this attack to check the validity of our claims and avoid a new introduction of mistakes. As detailed in the last part, we have decreased the requirements of the attack to get a practical time complexity. As a result, we can produce near-collisions in about 2^{36} operations where the bottleneck is the very last step, with low memory requirements of about 2^{16} stored elements.

We give the following Table 9.3 as an example of a near-collision on $512 - 4 \times 4 \times 8 = 384$ bytes.

9.2.2 Collision attack on the 4-round hash function

In this section, we describe a way to extend the compression function collision attack presented in the previous section into a collision attack on the 4-round hash function ECHO-256 with time and memory complexity of 2^{64}. We have just shown how to find a message pair (M_1, M_1') and a chaining value h such that $f(h, M_1) = f(h, M_1')$, where f is the ECHO-256 compression function. To get a collision in the hash function, the difference then consists in finding a message block M_0 which verifies $f(IV, M_0) = h$. This way, the collision in the hash function is the result of an internal collision in the compression function. Consequently, we do not need to

Table 9.3: Example of a near-collision on 384 bits out of 512 bits of the output of the ECHO-256 compression function reduced to 4 rounds. The constants added during the inner permutation have been ignored. A byte difference . . means zero difference.

$S[i,j]$	h_i	h'_i	$h_i \oplus h'_i$
$S0[0,0]$	DEDF73AC E834ABF3 1DA654E7 8B80E057	DEDF73AC E834ABF3 1DA654E7 8B80E057	....
$S0[1,0]$	8C82AF64 E938032D EA498F65 4F3FA168	8C82AF64 E938032D EA498F65 4F3FA168	....
$S0[2,0]$	A3DEC6EE BDD97F9C 69425DE7 B88FAE55	A3DEC6EE BDD97F9C 69425DE7 B88FAE55	....
$S0[3,0]$	E0276510 531114BA 8EA8ADD3 9037426B	E0276510 531114BA 8EA8ADD3 9037426B	....
$S[i,j]$	m	m'	$m \oplus m'$
$S0[0,1]$	B1B7D769 8B7AD57A 7B57FF05 472BECEF	B1B7D769 8B7AD57A 7B57FF05 472BECEF	....
$S0[1,1]$	D9E41EF0 FB869029 29B437B2 CC398919	D9E41EF0 FB869029 29B437B2 CC398919	....
$S0[2,1]$	CAAAC63A E8B4F522 DCA83BB4 52227A82	B6477E77 581C4385 A0035D3E 8C061217	7CEDB84D B0A8B6A7 7CAB668A DE246895
$S0[3,1]$	9142CAB0 D8421346 E35702E9 477A5AAB	6104E89C 8E995FCC 2AF9D466 B2C3D16C	F046222C 56DB4C8A C9AED68F F5B98BC7
$S0[0,2]$	F097871F B8733C73 3BD02C4C F7004240	A1E83191 315E7268 04D6F3D6 BF87220C	517FB68E 892D4E1B 3F06DF9A 4887604C
$S0[1,2]$	A765E039 EB6C558F B444631F DD4BC1AB	6993F70F 5F87B6BF 6402FB87 CA7859C6	CEF61736 B4EBE330 D0469898 1733986D
$S0[2,2]$	BCEAEFAA 8304B57E F2C6732D D396D8F8	2507A8FD 67F83C71 9B523FBF 3534F32E	99ED4757 E4FC890F 69944C92 E6A22BD6
$S0[3,2]$	C406CB83 EA157529 E008A7CB 11675D1A	005DF381 40322440 16E70F34 454F1318	C45B3802 AA275169 F6EFA8FF 54284E02
$S0[0,3]$	84258159 7A87E98E B750B21D 31D0F510	0429D2E3 5B02D7DE A22839AA 174013DA	800C53BA 21853E50 15788BB7 2690E6CA
$S0[1,3]$	A5808F25 DBDE4281 ECAFEF87 3607ACBB	8EEC6709 3B61D819 29D65D83 09B27795	2B6CE82C E0BF9A98 C579B204 3FB5DB2E
$S0[2,3]$	E9B4133F F7C776FC E9F2C741 754EBC6B	E9B4133F F7C776FC E9F2C741 754EBC6B	....
$S0[3,3]$	8C219844 7E17C475 7AED625F 3B685665	8C219844 7E17C475 7AED625F 3B685665	....
$S[i,j]$	h_{i+1}	h'_{i+1}	$h_{i+1} \oplus h'_{i+1}$
$S34[0,0]$	0EC3168C C7F787CA 4006FA09 3E29BA5E	0E55168C C7F714CA 4006FA0E C129BA5E	..96.... ....93.. ......07 FF......
$S34[1,0]$	FF729D65 2B555D10 AD0CF15C 9A9AFF87	FF179D65 2B55D810 AD0CF1D5 779AFF87	..65.... ....85.. ......89 ED......
$S34[2,0]$	7E2C1C9D 542E3BE0 AF880377 8887502A	7ED31C9D 542EF8E0 AF88037A 7587502A	..FF.... ....C3.. ......0D FD......
$S34[3,0]$	A776FCAF 96C2F792 FF051583 FF6482C6	A771FCAF 96C2F592 FF0515CC 0A6482C6	..07.... ....02.. ......4F F5......

take care of the padding in that scenario. For any message M, we would have:

$$\text{ECHO}_{4R}(IV, M_0||M_1||M) = \text{ECHO}_{4R}(IV, M_0||M'_1||M),$$

where $||$ denotes the concatenation of messages and ECHO_{4R} the ECHO-256 hash function reduced to 4 rounds.

As in most of the attacks on ECHO, we use the rebound technique with multiple inbound phases to find a valid message pair. The truncated differential characteristic used here is the same as in the previous section (Figure 9.7). The first inbound is located in the second round between states S7 and S14 and can be done in parallel on the four **BigColumns** and the second one is in the third round between states S16 and S24. We extend the latter by a probabilistic outbound phase to filter out some of its valid pairs in order to reduce the number of active bytes in the final fourth round. In the sequel, we consider the alternative description of the permutation (see Section 9.1.2).

At first, we precompute and store the differential distribution table $\mathcal{T}$ of the **Super-SBox** in time and memory 2^{64}. Then, we randomize the differences around the merging point: the **SuperMixColumns** at the end of the second round. Using $\mathcal{T}$, we find a pair of internal states conforming to the second inbound and the outbound phase. This can be done in the same way as in the previous attack in less than 2^{64} computations. We then find a valid pair of **BigColumns** for the first **BigColumn** in the first inbound: this fixes the value of $\text{diag}(\text{S7}[0,0])$ overlapping with an AES state directly coming from the previous chaining value yet to be determined.

By generating 2^{64} message M_0, we obtain a list L of 2^{32} chaining values sharing the same value on $\text{diag}(\text{S7}[0,0])$. We now generate all the 2^{32} possible **BigColumns** pairs for the second **BigColumn** in the first inbound. For each pair, we compute the value of $\text{diag}(\text{S7}[3,1])$: we expect one element of L to share this value because L contains 2^{32} elements. Consequently, we can update L by completing each of its entries by the associated pair for the second **BigColumn**

and get 2^{32} pairs for L again. For the third **BigColumn**, we compute 2^{64} pairs conforming to the path among the 2^{96} possible ones. With the same argument, for any element of L, we expect to find 2^{32} pairs where the value of $\mathrm{diag}(\mathrm{S7}[2,2])$ matches the one dictated by that particular chaining value. We link each of the 2^{32} elements of L with a set E of 2^{32} valid pairs for the third **BigColumn**. In other words, we get 2^{64} pairs where the three first **BigColumns** match the respective chaining value.

At this point, each of the 2^{32} entries of L consists of a chaining value $h = f(IV, M_0)$, a pair of **BigColumns** for the two first **BigColumns** and a set E of 2^{32} valid pairs for the third **BigColumn**, all conforming to the first inbound. As demonstrated in the previous section, the 128-bit condition to merge the two inbound phases can be deported to the fourth **BigColumn** in the first inbound. More precisely, once we know a pair for each of the three first **BigColumns**, we can deduce the fourth one so that the merge around the **SuperMixColumns** layer is possible. For each entry in L and for each associated set E, we are in that case so that we can deduce the pair for the fourth **BigColumn**. Finally, we check if the truncated path is verified for that **BigColumn** and if the value in $\mathrm{diag}(\mathrm{S7}[3,1])$ matches the expected one. The two events occur with probability 2^{-64} but since we can repeat the check $2^{32} \times 2^{32} = 2^{64}$ times, we should find one M_0 and one pair for each of the four **BigColumns**.

Finally, we can finish the attack as before by ensuring that the differences cancel out in the feed-forward and by merging the two partial solutions. We note that the merging is slightly different since we need to take care of the known AES states in S7 but we can still perform it in less than 2^{64} computations.

9.2.3 Collision attack on the 5-round hash function

In this section, we give a collision attack on the hash function ECHO-256 reduced to 5 rounds. The strategy first finds a sparse truncated differential characteristic and then, we observe an unwanted linear behavior in the output space of differences. This allows to efficiently find collision in the hash output with the birthday algorithm. We give all the details of this attack in the following sections.

9.2.3.1 The truncated differential characteristic

In this attack on the ECHO-256 hash function, we use two message blocks where the first one does not contain differences. For the second one, we use the truncated differential characteristic given in Figure 9.16. We use colors (red, yellow, green, blue, cyan) to describe different phases of the attack and to denote their resulting solutions. Active bytes are denoted by black color, and active AES states contain at least one active byte. Hence, the sequence of active AES states for each round of ECHO is as follows:

$$5 \xrightarrow{r_1} 16 \xrightarrow{r_2} 4 \xrightarrow{r_3} 1 \xrightarrow{r_4} 4 \xrightarrow{r_5} 16.$$

Note that in this truncated characteristic, we keep the number of active bytes low, except for the beginning and end. Therefore, we have enough freedom to find many solutions. We do not allow differences in the chaining input (blue) and in the padding (cyan). The last 16 bytes of the padding contain the message length and the two bytes above contain the size of the hash

function output. Note that the AES states containing the chaining values (blue) and padding (cyan) do not get mixed with other AES states until the first **BigMixColumns** transformation. Since the lower half of the state (row 2 and 3) is truncated to compute the final hash value, we force all differences to be in the lower half of the message: the feed-forward then preserves that property.

9.2.3.2 Colliding subspace differences

In the following, we show that the resulting output differences after 5 rounds lie in a vector space of reduced dimension. This can be used to construct a distinguisher for 5 rounds of the ECHO-256 hash function. However, due to the low dimension of the output vector space, we can even extend this distinguisher to get a collision attack on 5 rounds of the ECHO-256 hash function.

First, we need to determine the dimension of the vector space at the output of the hash function. This dimension is upper-bounded by the number of active bytes prior to the linear transformations in the last round (16 active bytes after the last **SubBytes**), combined with the number of active bytes at the input due to the feed-forward (0 active bytes in our case). Here, with the characteristic from Figure 9.16, this would result in a vector space dimension of $(16+0)\times 8 = 128$. However, a weakness in the combined transformations **SuperMixColumns**, **BigFinal** and the output truncation reduces the vector space to a dimension of 64 at the output of the hash function for this characteristic.

We can indeed move the **BigFinal** function prior to **SuperMixColumns**, since **BigFinal** is a linear transformation and the same linear transformation $\mathbf{M_{SMC}}$ is applied to all columns in the **SuperMixColumns**. Then, we get 4 active bytes at the same position in each AES state of the 4 resulting column-slices. To each active column-slice C_{16}, we first apply the **SuperMixColumns** multiplication with $\mathbf{M_{SMC}}$ and then, a matrix multiplication using

$$\mathbf{M_{trunc}} \stackrel{\text{def}}{:=} \left[I_8 \,\middle|\, 0_8 \right],$$

which truncates the lower 8 rows. Since only 4 bytes are active in C_{16}, these transformations can be combined into a transformation using a reduced 4×8 matrix $\mathbf{M_{comb}}$ applied to the reduced input C_4, which contains only the 4 active bytes of C_{16}:

$$\mathbf{M_{trunc}} \cdot \mathbf{M_{SMC}} \cdot C_{16} = \mathbf{M_{comb}} \cdot C_4,$$

The multiplication with zero differences of C_{16} removes 12 columns of $\mathbf{M_{SMC}}$ while the truncation removes 8 rows of $\mathbf{M_{SMC}}$. For example, considering the first active column-slice leads to:

$$\mathbf{M_{trunc}} \cdot \mathbf{M_{SMC}} \cdot \begin{bmatrix} a & 0 & 0 & 0 & b & 0 & 0 & 0 & c & 0 & 0 & 0 & d & 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}} = \underbrace{\begin{bmatrix} 4 & 6 & 2 & 2 & 6 & 5 & 3 & 3 \\ 2 & 3 & 1 & 1 & 4 & 6 & 2 & 2 \\ 2 & 3 & 1 & 1 & 2 & 3 & 1 & 1 \\ 6 & 5 & 3 & 3 & 2 & 3 & 1 & 1 \end{bmatrix}^{\mathrm{T}}}_{\mathbf{M_{comb}}} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}.$$

Analyzing the resulting matrix $\mathbf{M_{comb}}$ for all four active column-slices shows that in each case, the rank of $\mathbf{M_{comb}}$ is two, and not four. This reduces the dimension of the vector space in

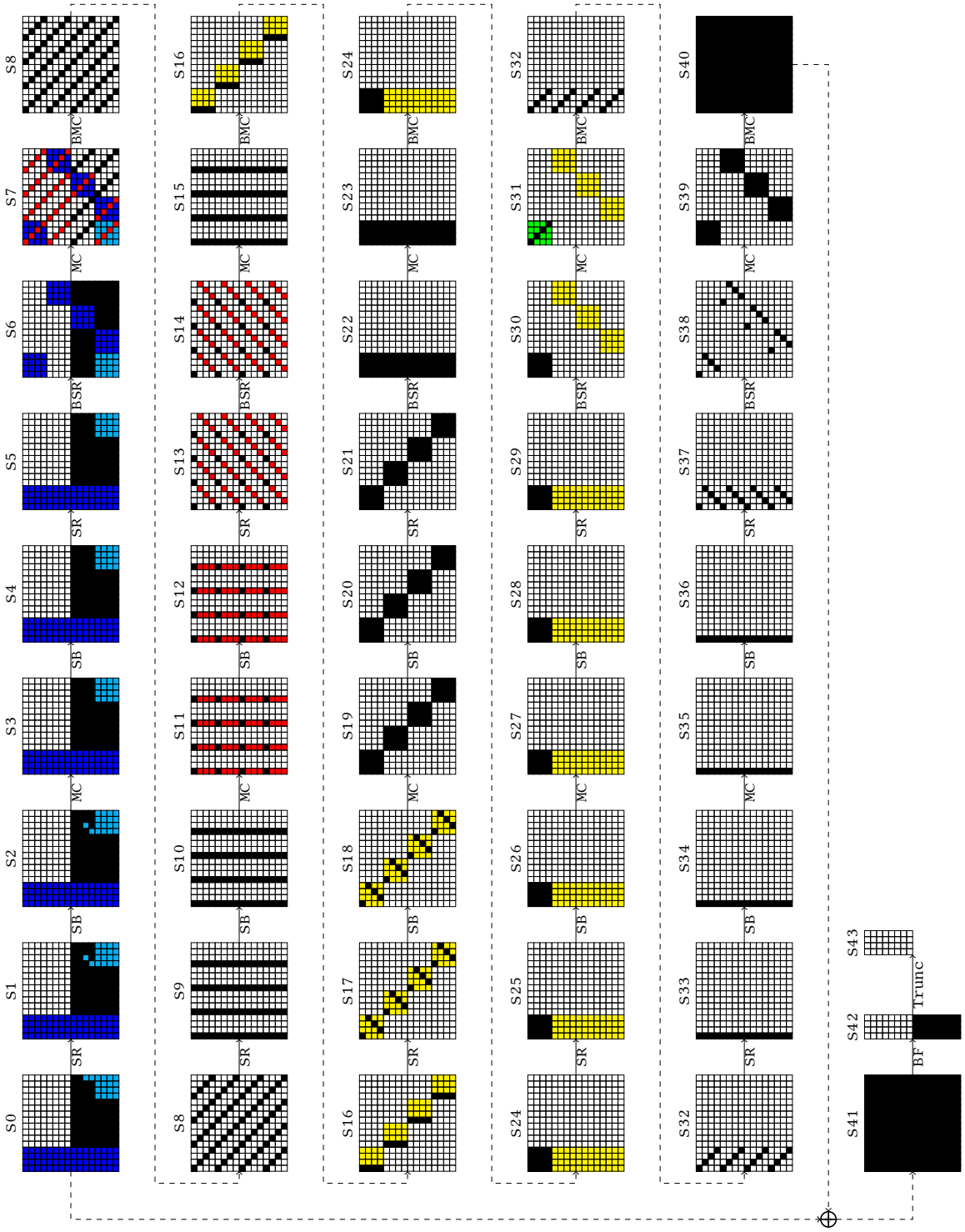


Figure 9.16: [

Rebound attack on 5 rounds of the ECHO hash function.] The truncated differential characteristic to get a collision for 5 rounds of ECHO-256. Black bytes are active, blue and cyan bytes are determined by the chaining input and padding, red bytes are values computed in the red inbound phase, yellow bytes in the yellow inbound phase and green bytes in the outbound phase.

each active column-slice from 32 to 16. Since we have four active columns, the total dimension of the vector space at the output of the hash function is 64. Furthermore, column $i \in \{0,1,2,3\}$ of the output hash value depends only on columns $4i$ of state S_{38}. It follows that the output difference in the first column $i = 0$ of the output hash value depends only on the four active differences in columns 0, 4, 8 and 12 of state S_{38}, which we denote by a, b, c and d. To get a collision in the first column of the hash function output, we get the following linear system of equations:

$$\mathbf{M_{comb}} \cdot \left[\begin{array}{cccc} a & b & c & d \end{array} \right]^{\mathrm{T}} = \left[\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right]^{\mathrm{T}}.$$

Since we cannot control the differences a, b, c and d in the following attack, we need to find a solution for this system of equations by brute-force. However, the brute-force complexity is less than expected due to the reduced rank of the given matrix. Since the rank is two, 2^{16} solutions exist and a random difference results in a collision with a probability of 2^{-16} instead of 2^{-32} for the first output column. Since the rank of all four output column matrices is two, we get a collision at the output of the hash function with a probability of $2^{-16\times 4} = 2^{-64}$ for the given truncated differential characteristic.

9.2.3.3 High-level outline of the attack

To find input pairs according to the truncated differential path given in Figure 9.16, we use a rebound attack with multiple inbound phases. Furthermore, we also use multiple outbound phases and separate the merging process into three different parts which can be solved mostly independently:

1. **First Inbound between S16 and S24:** find 2^{96} partial pairs (yellow and black bytes) with a complexity of 2^{96} computations and 2^{64} memory.
2. **First Outbound between S24 and S31:** filter the previous solutions to get 1 partial pair (green, yellow and black bytes) with a complexity of 2^{96} computations and 2^{64} memory.
3. **Second Inbound between S7 and S14:** find 2^{32} partial pairs (red and black) for each of the first three **BigColumns** and 2^{64} partial pairs for the last **BigColumn** of state S7 with a total complexity of 2^{64} computations and memory.
4. **First Part in Merging the Inbound Phases:** combine the 2^{160} solutions of the previous phases according to the 128-bit **SuperMixColumns** condition given in Section 9.1.2.2. We get 2^{32} partial pairs (black, red, yellow and green bytes between state S7 and S31) with 2^{96} computations and 2^{64} memory.
5. **Merge Chaining Input:** repeat from Step 1 for 2^{16} times to get 2^{48} solutions for the previous phases. Compute 2^{112} chaining values (blue) using 2^{112} random first message blocks. Merge these solutions according to the overlapping 20 bytes (red with blue/cyan) in state S7 to get $2^{48} \times 2^{112} \times 2^{-160} = 1$ partial pair in 2^{112} computations in time and 2^{48} memory.
6. **Second Part in Merging the Inbound Phases:** find one partial solution for the first two columns of state S7 according to the 128-bit condition imposed at the **SuperMixColumns** transition between S14 and S16 with complexity 2^{64} in time and memory.
7. **Third Part in Merging the Inbound Phases:** find one solution for all remaining bytes (last two columns of state S7) by fulfilling the resulting 192-bit condition using a generalized birthday attack with 4 lists [Wag02]. The complexity is 2^{64} in time and memory to find

one solution, and $2^{85.3}$ computations and memory to find 2^{64} solutions.

8. **Second Outbound Phase to get Collisions:** in a final outbound phase, the resulting differences at the output of the hash function collide with a probability of 2^{-64} and we get one collision among the 2^{64} solutions of the previous step.

The total time complexity of the attack is 2^{112} computations and determined by Step 5; the memory complexity is $2^{85.3}$ and determined by Step 7.

9.2.3.4 Details of the attack

In this section, we describe in detail each step of the collision attack on 5 rounds of ECHO-256. Note that some phases are also reused in the following attacks on the compression function of Section 9.2.4.

First Inbound between S16 and S24

We first search for internal state pairs conforming to the truncated differential characteristic in round 3 (yellow and black bytes). We start the attack by choosing differences for the active bytes in state S16 such that the truncated differential characteristic of **SuperMixColumns** between state S14 and S16 is fulfilled (Section 9.1.2). We compute this difference forward to state S17 through the linear layers.

We continue by randomly choosing differences for state S24 and compute them backwards to state S20, the output of the **Super-SBoxes**. Since we have 64 active S-Boxes in this state, the probability of a differential is about $2^{-1\times 64}$. Hence, we need 2^{64} starting differences but as soon as we get one, we can construct 2^{64} of them, so we get 2^{64} solutions for the inbound phase in round 3 in about 2^{64} operations. We determine the right pairs for each of the 16 **Super-SBox** between state S17 and S20 independently: using the differential distribution table of the **Super-SBoxes**, we can find one right pair with average complexity one.

In total, we compute 2^{96} solutions for this inbound phase in 2^{96} computations and memory complexity of at most 2^{64} to store the DDT of the **Super-SBoxes**. For each of these pairs, differences and values of all yellow and black bytes in the third round of Figure 9.16 are determined.

Second outbound between S24 and S31

In the outbound phase, we ensure the propagation in round 4 of the truncated differential characteristic by propagating the right pairs of the previous inbound phase forwards to state S31. With a probability of $(2^{-3\times 8})^4 = 2^{-96}$, we get four active bytes after **MixColumns** in state S31 (green) conforming to the truncated characteristic. Hence, among the 2^{96} right pairs of the inbound phase between S16 and S24, we expect to find one such right pair.

The total complexity to find this partial pair between S16 and S31 is then 2^{96} computations by exhausting the set of solutions obtained in previous step. Note that for this pair, the values and differences of the yellow, green and black bytes between states S16 and S31 can be determined. Furthermore, note that for any choice of the remaining bytes, the truncated

differential characteristic between state S31 and state S40 is fulfilled.

Second inbound between S7 and S14

Here, we search for many pairs of internal states conforming to the truncated differential characteristic between states S7 and S14. Note that as we have done before in Figure 9.16 for the 4-round attack, we can independently search for pairs for each of the four **BigColumn** in state S7, since they stay independent until they are mixed by the following **BigMixColumns** transformation between states S15 and S16. For each **BigColumn**, four **Super-SBoxes** are active, and we need at least 2^{16} starting differentials for each one to find the first right pair.

The difference in S14 is already fixed due to the yellow inbound phase but we can still choose at least $(2^8 - 1)^4 \approx 2^{32}$ differences for each active AES state in S7. Using the rebound technique, we can find one pair on average for each starting difference in the inbound phase. Then, we independently iterate through all 2^{32} starting differences for the first, second and third column and through all 2^{64} starting differences for the fourth column of state S7. We get 2^{32} right pairs for each of the first three columns and 2^{64} pairs for the fourth column. The complexity to find all these pairs is 2^{64} in time and memory.

We note that while we could apply similar techniques as the one presented in the previous attack (Section 9.2.1) that decrease the minimal time complexity of the rebound strategy, here the gain would be negligible in comparison to the much higher complexities of all the other steps.

For each resulting right pair, the values and differences of the red and black bytes between states S7 and S14 can be computed. Furthermore, the truncated differential characteristic in backward direction, except for two cyan bytes in the first states, is fulfilled. In the next phase, we partially merge the right pairs of the yellow and red inbound phase together, while also considering the extra linear conditions imposed by the sparse differential in the **SuperMixColumns**.

First part in merging the inbound phases

Until now, we have constructed one pair for the yellow inbound phase and in total, $2^{32} \times 2^{32} \times 2^{32} \times 2^{64} = 2^{160}$ pairs for the red inbound phase. Among these 2^{160} pairs, we expect to find 2^{32} right pairs which also satisfy the 128-bit condition of the **SuperMixColumns** between states S14 and S16 (see Section 9.1.2.2). In the following, we show how to find all these 2^{32} pairs using a meet-in-the-middle strategy with a complexity of 2^{96} operations.

First, we combine the $2^{32} \times 2^{32} = 2^{64}$ pairs determined by the two first **BigColumns** of state S7 in a list L_1 and the $2^{32} \times 2^{64} = 2^{96}$ pairs determined by the last two **BigColumns** of state S7 in a list L_2. Note that the pairs in these two lists are independent. Then, we separate the linear relations from Table 9.1 into terms determined by L_1 and terms determined by L_2 to perform the meet-in-the-middle.

Then, we can simply merge these lists to find those pairs which satisfy the 128-bit condition imposed by the **SuperMixColumns** and store these results in list L_{12}. This way, we get $2^{64} \times 2^{96} \times 2^{-128} = 2^{32}$ right pairs with a total complexity of 2^{96} computations. We note that the

memory requirements can be reduced to 2^{64} if we do not store the elements of L_2 but compute them online. The resulting 2^{32} solutions are partial right pairs for the black, red, yellow and green bytes between state S7 and S31.

Merge chaining input

Next, we need to merge the 2^{32} results of the previous phases with the chaining input (blue) and the bytes fixed by the padding (cyan). The chaining input and padding overlap with the red inbound phase in state S7 on $5 \times 4 = 20$ bytes. This results in a 160-bit condition on the overlapping blue/cyan/red bytes. To find a pair verifying this condition, we first generate 2^{112} random first message blocks, compute the blue bytes of state S7 and store the results in a list L_3.

Additionally, we repeat 2^{16} times from the yellow inbound phase but with other starting points[4] in state S24. This way, we get $2^{16} \times 2^{32} = 2^{48}$ right pairs for the combined yellow and red inbound phases, which also satisfy the 128-bit condition of **SuperMixColumns** between states S14 and S16. This requires about $2^{16} \times 2^{96} = 2^{112}$ operations. We store the resulting 2^{48} pairs in list L_{12}.

Next, we merge the lists according to the overlapping 160 bits and get $2^{48} \times 2^{112} \times 2^{-160} = 1$ right pair. If we compute the 2^{112} message blocks of list L_3 online, the time complexity of this merging step is 2^{112} operations, with memory requirements of 2^{48} states to store the list L_{12}. For the resulting pair, all differences between states S4 and S33 and all colored byte values (blue, cyan, red, yellow, green and black) between states S0 and S31 can be determined.

Second part in merging inbound phases

To completely merge the two inbound phases, we need to find values for the white bytes. We use Figure 9.17 to illustrate the second and third part of the merge inbound phase. In this figure, we only consider values and therefore, do not show active bytes (black). Furthermore, all brown and cyan bytes have already been chosen in one of the previous steps. In the second part of the merge inbound phase, we only choose values for the gray and light-gray bytes. All other colored bytes show steps of the third and last merging phase (next section).

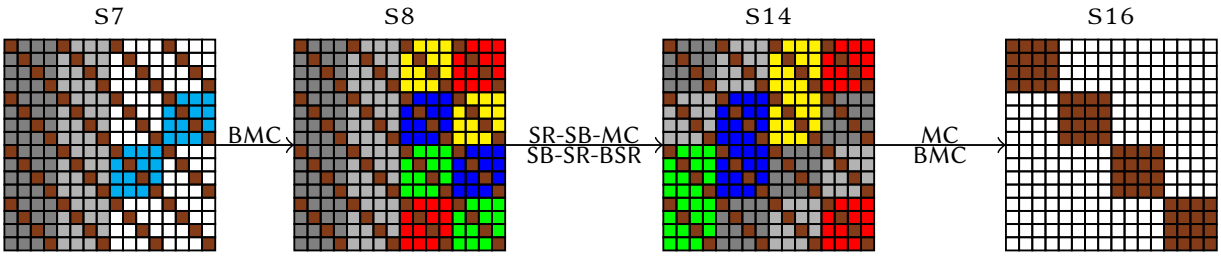


Figure 9.17: States used to merge the two inbound phases with the chaining values. The merge inbound phase consists of three parts. Brown bytes show values already determined (first part) and gray values are chosen at random (second part). Green, blue, yellow and red bytes show independent values used in the generalized birthday attack (third part) and cyan bytes represent values with the target conditions.

[4] Until now, we have chosen only 2^{96} out of 2^{128} differences for this state.

We first choose random values for all remaining bytes of the first two columns in state S7 (gray and light-gray) and independently compute the **BigColumns** forward to state S14. Note that we need to try $2^{2\times 8+1}$ values for AES state S7$[2,1]$ to also match the 2-byte (cyan) and 1-bit padding at the input in AES state S0$[2,3]$. Then, all gray, light-gray, cyan and brown bytes have already been determined either by an inbound phase, chaining value, padding or just by choosing random values for the remaining free bytes of the first two columns of S7. However, all white, red, green, yellow and blue bytes are still free to choose.

By considering the linear **SuperMixColumns** transformation between state S14 and S16, we observe that in each column-slice, 14 out of 32 input/output bytes are already fixed and 2 bytes are still free to choose. Hence, we expect to get 2^{16} solutions for this linear system of equations. Unfortunately, also for the given position of already determined 14 bytes, the linear system of equations does not have a full rank. Again, we can determine the resulting system using the matrix $\mathbf{M_{SMC}}$ of **SuperMixColumns**. As an example, for the first column-slice, the system is given as follows:

$$\begin{aligned}\mathbf{M_{SMC}} \cdot [&A_0\ L_0\ L_1\ L_2\ A_1\ L'_0\ L'_1\ L'_2\ A_2\ x_6\ x_7\ x_8\ A_3\ x_9\ x_{10}\ x_{11}]^{\mathrm{T}}\\ =&[B_0\ B_1\ B_2\ B_3\ y_0\ y_1\ y_2\ y_3\ y_4\ y_5\ y_6\ y_7\ y_8\ y_9\ y_{10}\ y_{11}]^{\mathrm{T}}.\end{aligned}$$

The free variables in this system are $x_6, \ldots, x_{11}$ (green). The values A_0, A_1, A_2, A_3, B_0, B_1, B_2, B_3 (brown) have been determined by the first or second inbound phase and the values L_0, L_1, L_2 (light-gray) and L'_0, L'_1, L'_2 (gray) are determined by the choice of arbitrary values in state S7. We proceed as before and determine the linear system of equations which needs to have a solution:

$$\begin{bmatrix} 3 & 1 & 1 & 3 & 1 & 1 \\ 2 & 3 & 1 & 2 & 3 & 1 \\ 1 & 2 & 3 & 1 & 2 & 3 \\ 1 & 1 & 2 & 1 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}.$$

The resulting linear 8-bit equation to get a solution for this system can be separated into terms depending on values of L_i and on L'_i, and we get

$$f_1(L_i) + f_2(L'_i) + f_3(a_i, b_i) = 0,$$

where f_1, f_2 and f_3 are linear functions. For all other 16 column-slices and fixed positions of gray bytes, we get matrices of rank three as well. In total, we get 16 8-bit conditions and the probability to find a solution for a given choice of gray and light-gray values in states S14 and S16 is 2^{-128}. However, we can find a solution to these linear equations using the birthday effect and a meet-in-the-middle attack with a time and memory complexity 2^{64}.

We start by choosing 2^{64} different values for each of the first (gray) and second (light-gray) **BigColumns** in state S7. We compute these values independently forward to state S14 and store them in two lists L and L'. We also separate all equations of the 128-bit condition into parts depending only on values of L and L'. We apply the resulting functions f_1, f_2 and f_3 to the elements of lists L_i and L'_i, and merge two lists L and L' using the birthday effect.

Third part in merging inbound phases

We continue with a generalized birthday match to find values for all remaining bytes of the state (blue, red, green, yellow, cyan and white of Figure 9.17). For each **BigColumn** in state S14, we independently choose 2^{64} values for the green, blue, yellow and red columns, and compute them independently backward to S8. We need to match the values of the cyan bytes of state S7, which results in a condition on 24 bytes or 192 bits. Since we have four independent lists with 2^{64} values in state S8, we can use the generalized birthday attack [Wag02] to find one solution with a complexity of $2^{192/3} = 2^{64}$ in both time and memory.

In more detail, we need to match values after the **BigMixColumns** transformation in the backward direction. Hence, we first multiply each byte of the four independent lists by the four multipliers of the **InvMixColumns** transformation. Then, we get 24 equations containing only XOR conditions on bytes between the target value and elements of the four independent lists, which can be solved using a generalized birthday attack.

To improve the average complexity of this generalized birthday attack, we can start with larger lists for the green, blue, yellow and red columns in state S14. Since we need to match a 192-bit condition, we can get $2^{3 \cdot x} \times 2^{-192} = 2^x$ solutions with a time and memory complexity of $\max\{2^{64}, 2^x\}$ (see [Wag02] for more details). Note that we can even find solutions with an average complexity of 1 using lists of size 2^{96}. Each solution of the generalized birthday match results in a valid pair conforming to the whole 5-round truncated differential characteristic.

Second outbound phase to get collisions

For the collision attack on 5 rounds, we start the generalized birthday attack of the previous phase with lists of size $2^{85.3}$. This results in $2^{3 \cdot 85.3} \times 2^{-192} = 2^{64}$ solutions with a time and memory complexity of $2^{85.3}$, or with an average complexity of $2^{21.3}$ per solution. These solutions are propagated outwards in a second, independent outbound phase. Since the differences at the output collide with a probability of 2^{-64}, we expect to find one pair which collides at the output of the hash function. The time complexity is determined by merging the chaining input and the memory requirements by the generalized birthday attack. To summarize, the complexity to find a collision for 5 rounds of the ECHO-256 hash function is given by about 2^{112} compression function evaluations with memory requirements of $2^{85.3}$.

9.2.4 Distinguisher for the 7-round compression function

In this section, we detail the distinguisher on the compression of ECHO-256 reduced to 7 rounds in the known-salt model. The truncated differential characteristic that we use now is depicted on Figure 9.18.

First, we show how to obtain partial solutions that verify the characteristic from the state S6 to S23 with an average time complexity of 2^{64} computations, as we obtain 2^{64} solutions with a cost of 2^{128} computations. These partial solutions determine also the values of the blue bytes of Figure 9.18. Next, we show how to do the same for the yellow part of the characteristic from S30 to S47. Finally, we explain how to merge these partial solutions to find one that verifies the complete characteristic.

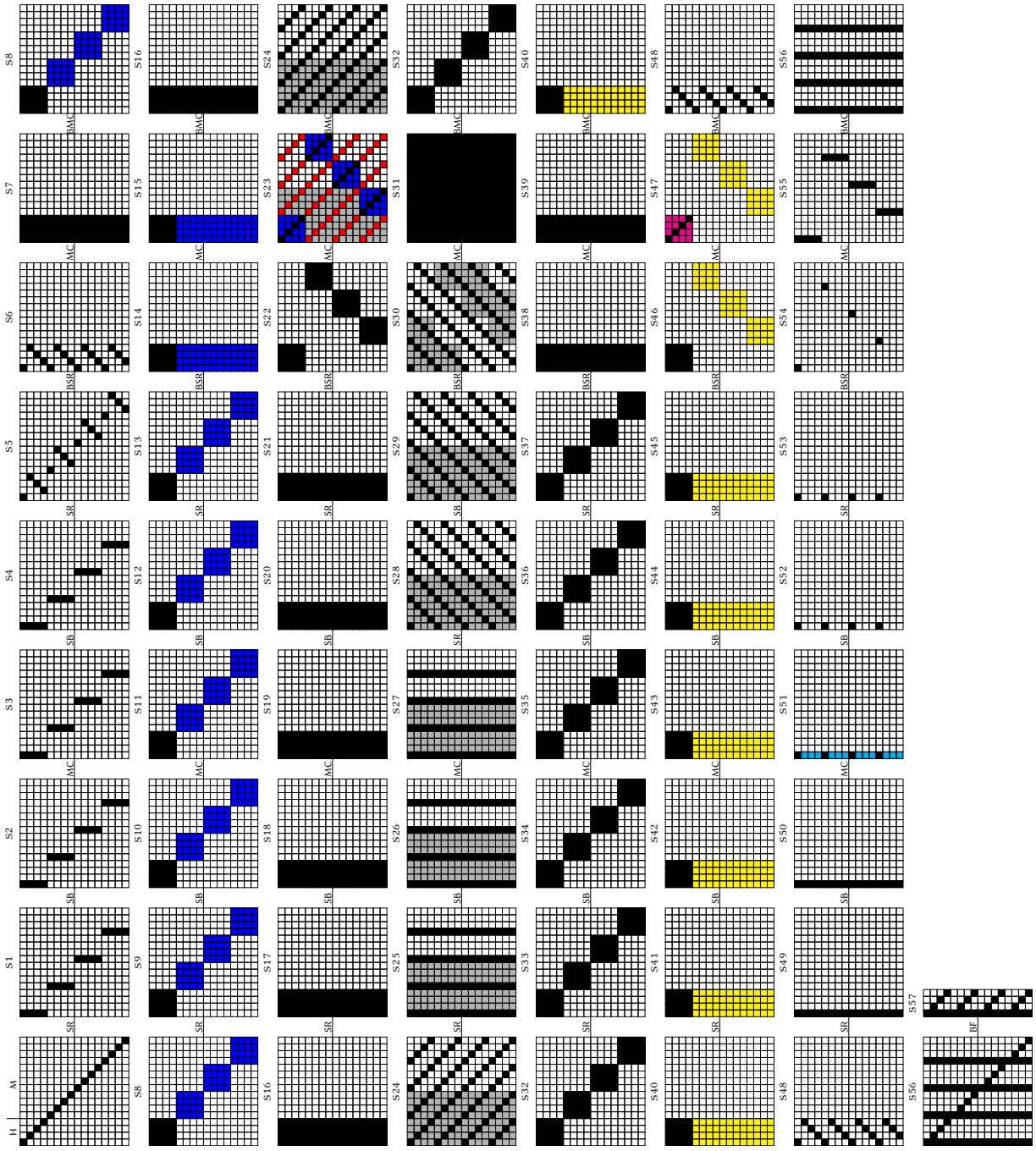


Figure 9.18: Truncated differential characteristic for the distinguisher on the 7-round compression function of ECHO-256.

9.2.4.1 Finding pairs between S6 and S23

We explain here how to find 2^{64} solutions for the blue part with a time complexity of 2^{128} operations and 2^{64} in memory. This is done with a stop-in-the-middle algorithm similar to the one presented by María Naya-Plasencia in [NP11] for improving the time complexity of the ECHO-256 distinguisher. This algorithm has to be adapted to this particular situation, where all the active states belong to the same **BigColumn**.

We start by fixing the difference in S8 to a chosen value, so that the transition between S6 and S8 is verified. We fix the difference in the active diagonals of the two AES-states S23$[0,0]$ and S23$[3,1]$ to a chosen value.

From state S8 to S13, we have four different **Super-SBox** groups involved in the active part. From states S16 to S22, we have 4×4 **Super-SBox** groups involved (4 per active AES state). Those 16 groups, as well as the 4 previous ones, are completely independent from S16 to S22 (respectively from S8 to S13). From the known difference in S8, we build four lists of values and differences in S13: each list corresponds to one of the four **Super-SBox** groups. Each list is of size 2^{32} because once we know the input difference, we try all the possible 2^{32} possible values and then we can compute the values and differences in S13 (as we said, the four groups are independent in this part of the characteristic). In the sequel, those lists are denoted L_A^i, $i = 0, \ldots, 3$.

There are 64 bits of differences not yet fixed in S23. Each active diagonal only affects the AES state where it is in, so we can independently consider 2^{32} possible differences for one diagonal and 2^{32} differences for the other. We can now build the 16 lists corresponding to the 16 **Super-SBox** groups as we did before, but considering that the 8 lists corresponding to 8 groups of the two AES states S16$[0,0]$ and S16$[3,0]$, as they have their differences in S22 already fixed, have a size of 2^{32} (corresponding to the possible values for each group). These are the lists $L_{0,0}^i$ and $L_{3,0}^i$, with $i \in [0,3]$ that represent the ith diagonal of the state. But the lists $L_{1,0}^i$, $L_{2,0}^i$, with $i \in [0,3]$, as they do not have yet the difference fixed, have a size of 2^{32+32} each, as we can consider the 2^{32} possible differences for each not fixed diagonal independently.

Next, we go through the 2^{64} possible differences of the first two diagonals (diagonals 0 and 1) of the active AES state in S15. For each one of these 2^{64} possible differences:

- The associated differences in the two same diagonals in the four active AES states of S16 can be computed. Consequently, we can check in the previously computed ordered lists $L_{j,0}^i$ with $j \in [0,3]$ and $i \in [0,1]$ where we find this difference. Here, i is either 0 or 1 because we are just considering the first two diagonals. For $j \in \{0,3\}$, on average, we obtain one match on each one of the lists $L_{0,0}^0$, $L_{0,0}^1$, $L_{3,0}^0$ and $L_{3,0}^1$. For $j \in \{1,2\}$, we obtain 2^{32} matches, one for each of the 2^{32} possible differences in the associated diagonals in S23. That is, 2^{32} matches for $L_{1,0}^0$ and $L_{1,0}^1$, where a pair of values formed by one element of each list is only valid if they were generated from the same difference in S23. Consequently, we can construct the list $L_{1,0}^{0,1}$ of size 2^{32} where we store the values and differences of those two diagonals in the AES state S16$[1,0]$ as well as the difference in S23 from which they were generated. Repeating the process for $L_{2,0}^0$ and $L_{2,0}^1$, we construct the list $L_{2,0}^{0,1}$ of size 2^{32}. We can merge the lists $L_{1,0}^{0,1}$, $L_{2,0}^{0,1}$ and the four fixed values for differences and values obtained from the matches in the lists $L_{0,0}^0$, $L_{0,0}^1$, $L_{3,0}^0$ and $L_{3,0}^1$, corresponding to the AES

states S16$[0,0]$ and S16$[3,0]$. This generates the list $L^{0,1}$ of size 2^{64}. Each element of this list contains the values and differences of the two diagonals 0 and 1 of the four active AES states in S16. As we have all the values for the two first diagonals in the four AES states, for each one of these elements, we compute the values in the first two diagonals of the active state in S15 by applying the inverse of **BigMixColumns**. We order them according to these values.

- Next, we go through the 2^{64} possible differences of the two next diagonals (diagonals 2 and 3) of the active AES state in S15. For each one of these 2^{64} possible differences:
  - All the differences in the AES state S13$[0,0]$ are determined. We check in the lists L^0_A, L^1_A, L^2_A and L^3_A if we find a match for the differences. We expect to find one in each list and this determines the values for the whole state S15$[0,0]$ (as the elements in these lists are formed by differences and values). This means that the value of the active AES state in S15 is also completely determined. This way, we can check in the previously generated list $L^{0,1}$ if the correct value for the two diagonals 0 and 1 appears. We expect to find it once.
  - As we have just found a valid element from $L^{0,1}$, it determines the differences in the AES states S23$[1,0]$ and S23$[2,0]$ that were not fixed yet. Now, we need to check if, for those differences in S23, the corresponding elements in the four lists $L^i_{1,0}$, $L^i_{2,0}$ for $i \in [2,3]$ that match with the differences fixed in the diagonals 2 and 3 of S15 [5], satisfy the values in S15 that were also determined by the lists L^i_A. This occurs with probability 2^{-64}.

All in all, the time complexity of this algorithm is $2^{64} \cdot (2^{64} + 2^{64}) = 2^{129}$ computations with a memory requirement of 2^{64}. The resulting expected number of valid pairs is $2^{64} \cdot 2^{64} \cdot 2^{64} \cdot 2^{-64} \cdot 2^{-64} = 2^{64}$.

9.2.4.2 Finding pairs between S30 and S47

In quite the same way as the previous section, we can find solutions for the yellow part with an average cost of 2^{64} computations. To do so, we take into account the fact that the **MixColumns** and **BigMixColumns** transformations commute. So, if we swap their positions between states S39 and S40, we only have one active AES state in S39. We fix the differences in S47 and in two AES states, say S32$[0,0]$ and S32$[1,1]$, and we still have 2^{32} possible differences for each of the two remaining active AES states in S32. Then, the lists L^i_A are generated from the end and contain values and differences from S40. Similarly, the lists $L^i_{j,j}$ contain values and differences from S38. We can apply the same algorithm as before and obtain 2^{64} solutions with a time complexity of 2^{128} computations and 2^{64} in memory.

9.2.4.3 Merging solutions

In this section, we explain how to get a solution for the whole characteristic. As explained in Section 9.2.4.1, we can find 2^{64} solutions for the blue part, that have the same difference for the active AES states of columns 0 and 1 in S23. We obtain 2^{64} solutions from a fixed value for the differences in S8 and the AES states S23$[0,0]$ and S23$[3,1]$. Repeating this process for the 2^{32} possible differences in S8, we obtain in total 2^{96} solutions for the blue part with the

[5] We expect one match per list.

same differences in the columns 0 and 1 in S23. The cost of this step is 2^{160} in time and 2^{96} in memory.

The same way, using the algorithm explained Section 9.2.4.2, we can also find 2^{96} solutions for the yellow part, that have the same difference value for the AES active states of columns 0 and 1 in S32 (we fix the difference value of these two columns in S32, and we try all the 2^{32} possible values for the difference in S47). The cost of this step is also 2^{160} in time and 2^{96} in memory.

Now, from the partial solutions obtained in the previous steps, we want to find a solution that verifies the whole differential characteristic. For this, we want to merge the solutions from S23 with the solutions from S32. We know that the differences of the columns 0 and 1 of S24 and S31 are fixed. Hence, from S24 to S31, there are four AES states for which we know the input difference and the output difference, as they are fixed[6]. We can then apply a variant of the **Super-SBox** technique in these four AES states: it fixes the possible values for the active diagonals of those states.

The differences in the other four AES states in S24 that are fixed are associated to other differences that are not fixed[7]. There are 2^{64} possible differences, each one associated to 2^{32} solutions for S32-S47 given by the solutions that we found in the second step. For each one of these 2^{64} possible differences, one possible value is associated by the **Super-SBox**. When computing backwards these values to state S24, as we have also the values for the other four AES states of the columns 0 and 1 that are also fixed (in the third step), we can compute the values for these two columns in S23, and we need 32×2 bit conditions to be verified on the values. So for each one of the 2^{64} possible differences in S31, we obtain $2^{96-64} = 2^{32}$ that verify the conditions on S23. In total, we have $2^{64+32} = 2^{96}$ possible partial matches.

For each of the 2^{64} possible differences in S31, its associated 2^{32} possible partial matches also need to verify the 128-bit condition in S30-S32 at the **SuperMixColumns** layer (Section 9.1.2.2) and the remaining 2×32 bit conditions on the values of S23. Since for each of the 2^{64} differences we have 2^{32} possible associated values in S32, the probability of finding a good pair is $2^{96-128-64+32} = 2^{-64}$.

If we repeat this merging procedure 2^{64} times, namely for 2^{32} differences in the columns 0 and 1 of S23 and for 2^{32} differences in the columns 0 and 1 of S32, we should find a solution. We then repeat the procedure for the cross product of the 2^{32} solutions for each side. As we do not want to compute them each time that we use them, as it would increase the time complexity, we can just store the $2^{64+32+32} = 2^{128}$ solutions for the first part and use the corresponding ones when needed, while the second part is computed in sequence. The complexity would be: $2^{192} + 2^{192} + 2^{96+64}$ in time and 2^{128} in memory. So far, we have found a partial solution for the differential part for rounds from S6 to S48. We still have the passive bytes to determine and the condition to pass from S50 to S51 to verify. This can be done exactly as in the second and third parts of the merge inbound phase of the previous attack (in Section 9.2.3.4) with no additional cost.

[6] S24[0, 0], S24[0, 1], S24[1, 1], S24[3, 0] correspond to S31[0, 0], S31[0, 1], S31[1, 0], S31[3, 1], respectively.
[7] S24[1, 0], S24[2, 0], S24[2, 1], S24[3, 1] correspond to S31[1, 3], S31[2, 2], S31[2, 3], S31[3, 2].

9.2.5 Collision attack on the 6-round compression function

Moreover, since we can find x solutions with complexity $\max\{x, 2^{96}\}$ in time and 2^{96} memory for the (independent) merge inbound phase, we can get $x < 2^{193}$ solutions with time complexity $2^{193} + \max\{x, 2^{96}\} \approx 2^{193}$ and 2^{128} memory. We need only 2^{96} of these solutions to pass the probabilistic propagation in the last round from S50 to S51. Hence, we can find a complete solution for the whole characteristic with a cost of about 2^{193} computations and 2^{128} in memory.

Furthermore, with a probability of 2^{-128}, the input and output differences in S0 and S48 collide in the feed-forward and **BigFinal** transformation. Therefore, we can also generate free-start collisions for 6 rounds of the compression function with a time complexity of $2^{193} + 2^{128} \approx 2^{193}$ computations and 2^{128} memory.

9.2.6 Chosen-salt attacks on the compression function

In this section, we show how to get a collision attack for 6 rounds and a distinguisher for 7 rounds of the ECHO-256 compression function in the chosen-salt model. For both attacks, we get a complexity of 2^{160} compression function evaluations with memory requirements of 2^{128}.

The attacks on the hash functions of ECHO can be extended to the compression function in a straightforward way. In this case, instead of the chaining value, a 512-bit value of another inbound phase is merged with the first inbound phase. In fact, we can continue with a similar 3-round path in backward direction as we have in the hash function case in forward direction. Then, the full active ECHO state is located in the middle round and we can construct attacks for up to 7 rounds for the compression functions of ECHO-256 (see Figure 9.19).

9.2.6.1 The truncated differential characteristic

We use the 7-round truncated differential characteristic given in Figure 9.19. Black bytes are active and colored bytes show the different inbound and outbound phases. Since the characteristic is sparse, we are able to find many right pairs that conform to it. We can already compute the expected number of right pairs by considering the **MixColumns** and **SuperMixColumns** transformations. At the input, we can freely choose the 256-byte values, the 16-byte difference between the values and the 16-byte salt. We get a reduction of pairs at the first **MixColumns** and **SMC** of round 1, the second **MixColumns** of round 3, the first **MixColumns** and **SMC** of round 4, the **BMC** of round 5 and the second **MixColumns** of round 6. The differential probability (in base-2 logarithm) for the path is given as follows:

$$8 \times (-12 - 3 - 48 - 48 - 12 - 48 - 12) = -8 \times 183.$$

To summarize, the expected number of pairs conforming to this 7-round truncated differential path is

$$2^{8\times(256+16+16)} \times 2^{-8\times 183} = 2^{800},$$

which corresponds to 800 degrees of freedom. Note that this is much more than for the characteristics given in [MPRS09] and [GP10].

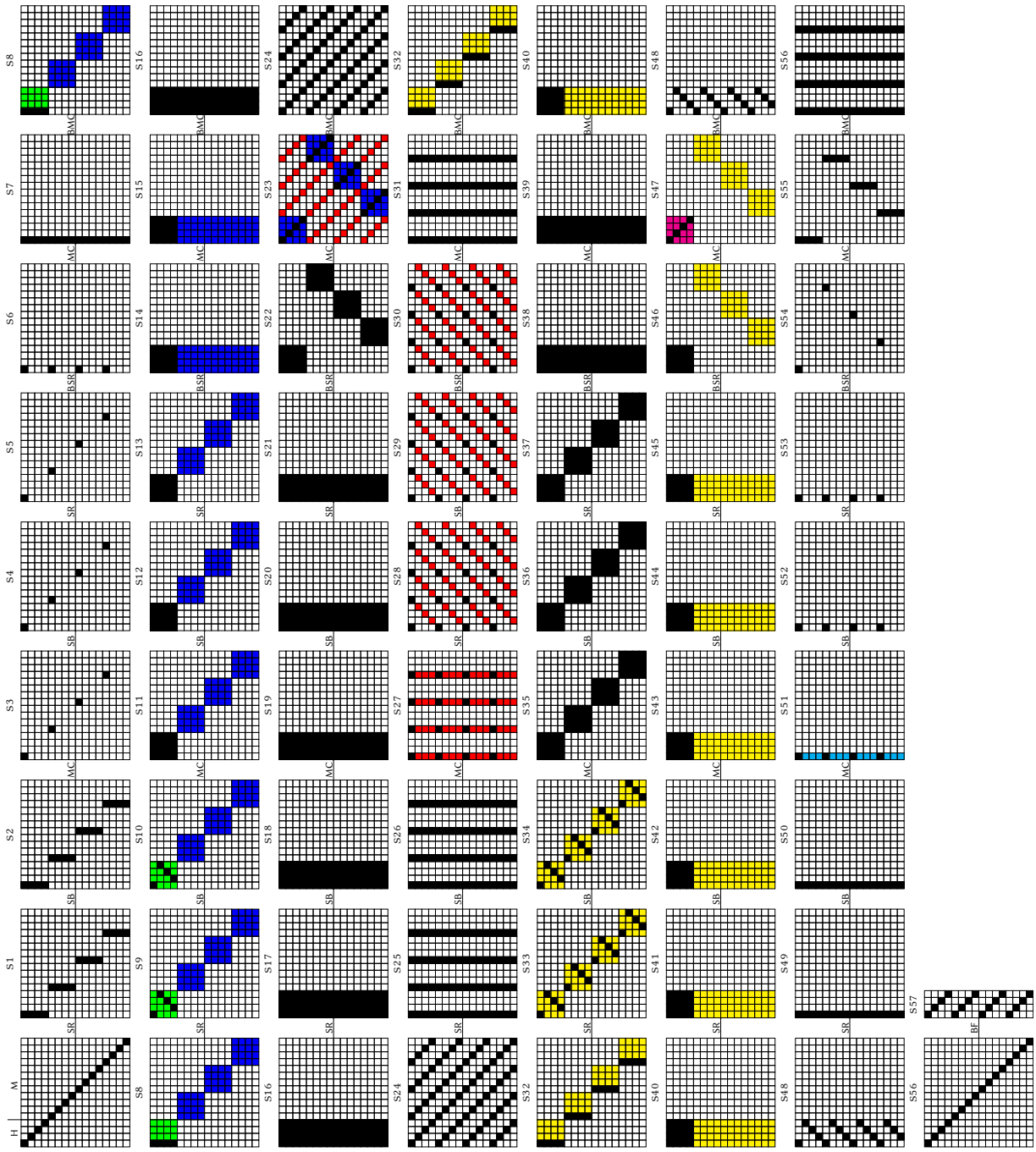


Figure 9.19: The truncated differential path to get collisions for 6 rounds and near-collisions for 7 rounds of the ECHO-256 compression function. Black bytes are active, red bytes are values computed in the first inbound phase, yellow bytes in the second, blue bytes in the third and green bytes in the fourth inbound or second outbound phase, and cyan bytes in the third outbound phase. Purple bytes are determined in the first outbound phase and gray bytes are chosen in the merge inbound phase.

9.2.6.2 Outline of the attack

The main idea of the attack is to find solutions for the forward and backward parts independently for fixed differences at the same layer between states S30 and S32. For the yellow/purple part, we can find 2^{128} pairs with a complexity of 2^{128} computations by choosing the salt value. For the green/blue/red part, we detail how to find 2^{128} pairs as well, but with a complexity of 2^{160} computations by constructing the salt.

Then, we just need to match the 128-bit salt values of the forward and backward parts and fulfill the 128-bit condition on the input (red) and output (yellow) values of **SuperMixColumns** for the merge to be possible. Since we get 2^{128} independent pairs for both the forward and backward parts, we can fulfill the resulting 256-bit condition by merging the two resulting lists.

9.2.6.3 Finding right pairs

In this section, we show how to find a pair for the first 6 rounds of the 7-round truncated differential characteristic of Figure 9.19. We detail how to find such a right pair in 2^{160} computations with 2^{128} memory. We use this pair in the chosen-salt model to get a collision for 6 rounds and a distinguisher for 7 rounds of the ECHO-256 compression function.

Inbound between S30 and S40

We choose a difference for state S32 such that the differential in the **SuperMixColumns** transformation between state S30 and S32 is fulfilled. Then, for each of the 2^{128} differences in state S40, we perform an inbound phase between states S32 and S40. In average, we get one solution with average complexity one: we can then compute 2^{128} pairs for the yellow inbound phase with complexity 2^{128} computations. We store these pairs sorted by their difference in state S40 in list L_1.

Outbound between S40 and S47

We continue to find pairs which also satisfy the truncated differential characteristic until state S47. We choose 2^{128} random pairs for the AES state in S47 (according to the given truncated differential characteristic) and compute backwards to state S40. For each resulting difference in S40, we lookup the matching difference in list L_1. To match also the values, we can construct the 128-bit salt value accordingly. Thus, we get 2^{128} pairs with 2^{128} computations according to the truncated differential characteristic from state S32 to state S48.

Inbound between S23 and S30

The red inbound phase is the same as in the hash function attack (Section 9.2.3): we start with the difference between states S30 and S32, which has been chosen in the yellow inbound phase. Then, we do four independent inbound phases for each **BigColumn** in state S23. Since we can start with at least 2^{32} differences for each column in state S23, we also get 2^{32} pairs for each column with a time complexity of 2^{32} computations.

Inbound between S15 and S23

Independently from the previous step, in the blue inbound phase, we start with a fixed difference in state S15 and compute this difference forward to state S17. Again, we can choose all 2^{32} differences for each **BigColumn** of state S23 and perform the blue inbound phases independently for each active AES state in the backward direction. For each column, we get 2^{32} pairs with a complexity of 2^{32} operations.

Merging inbounds

When merging the solutions of the blue and red inbound phases, we want to get one pair with average complexity one. Note that for each inbound phase and each column of state S23, we have 2^{32} right pairs. Moreover, we are allowed to set the salt value. We then start by matching the differences in the overlapping four bytes of each **BigColumn**. Since we have 2^{32} solutions for each of the blue and the red part, we get $2^{32} \times 2^{32} \times 2^{-32} = 2^{32}$ pairs with matching differences but non-matching values.

To match also these 4-byte values, we only set the four diagonal bytes of the salt value. For each of the 2^{32} pairs with matching differences, we compute the diagonal bytes of the salt such that the values also match. We sort the resulting list according to the 4-byte salt value and repeat the same for all four **BigColumns** of state S23. Then, we just need to iterate through all four lists and search for matching salt values. Note that for some salt values, we get no solution, but for some we will get more than one solution. On average, we expect to get 2^{32} matching pairs with a complexity of 2^{32} computations with chosen diagonal bytes of the salt.

Inbound between S6 and S15

To find a pair of states conforming to the green part, we first choose a difference verifying the truncated differential characteristic between state S6 and state S8. The second starting point for the green inbound phase is the difference in state S15, which has been chosen in the blue inbound phase. Again, we get one pair on average for each starting differential. This pair needs to be connected with the solutions of the blue inbound phase. To do so, we first match the values in the diagonal bytes of state S15. Remember that in the previous phases, we have constructed 2^{32} pairs for a single difference in state S15. Among these pairs, we expect to find one such that the diagonal 4-byte values between the green and blue inbound phase match. To connect the other 12 bytes, we can simply set the remaining 12 bytes of the salt value. Hence, we get one solution for the combined green, blue and red part with an average complexity of 2^{32} computations.

First Part in Merging Inbound Phases

To merge the inbound phases, we first compute 2^{128} pairs for the yellow/purple part with a time and memory complexity of 2^{128} and store these pairs in a list L_2. Similarly, we compute 2^{128} pairs for the green/blue/red part. Since the complexity to compute one solution for this part is 2^{32} computations, the time complexity to compute all 2^{128} pairs is 2^{160} operations. To connect the resulting pairs between states S30 and S32, we need to satisfy two 128-bit conditions.

First, we need to verify the 128-bit linear condition due to the sparse **SuperMixColumns** differential recalled in Section 9.1.2.2. Second, since each solution of the yellow/purple and green/blue/red part has also a different salt value, we need to match the 128-bit salt as well. In the end, this leads to a 256-bit condition, which we can be satisfied by merging the two lists L_1 and L_2 under that condition to produce $2^{128} \times 2^{128} \times 2^{-256} = 1$ right pair, which satisfies the whole 6-round truncated differential path. The time complexity of this step is 2^{160} computations with memory requirements of 2^{128}.

Second Part in Merging Inbound Phases

In the second part of the merge inbound phase, we need to find values for the first two columns of Figure 9.17. This part of the attack is the same as in the hash function attack on ECHO-256 (see Section 9.2.3).

Third Part in Merging Inbound Phases

The only difference in the third part of the merge inbound phase is that we change the time-memory trade-off slightly to get an average complexity of 1 for each solution. Again, we do a generalized birthday attack [Wag02] but this time, we start with 2^{96} independent values for each column of state S30 (Figure 9.17). Since we have a 192-bit condition in state S23, we get $2^{3\times 96} \times 2^{-192} = 2^{96}$ solutions with a complexity of 2^{96} in time and memory, or with an average complexity of 1 per solution. It follows that we can find up to 2^{160} right pairs for the 6-round truncated differential path with a total complexity of 2^{160} operations and memory requirements of 2^{128}.

9.2.6.4 Chosen-salt collision attack for 6 rounds

To get a collision for 6 rounds of the 512-bit compression function of ECHO-256 in the chosen-salt model, we need to ensure that the differences in the feed-forward cancel the output differences of the permutation: this happens with a probability of 2^{-128}. Since we can find 2^{160} pairs for the truncated differential path with a complexity of 2^{160} computations, we expect to find 2^{32} collisions at the output of the 6-round compression function with a time complexity of 2^{160} computations and memory requirements of 2^{128}.

9.2.6.5 Chosen-salt distinguisher for 7 rounds

To get a chosen-salt distinguisher for the compression function of ECHO-256 reduced to 7 rounds, we use the complete truncated differential characteristic given in Figure 9.19. Note that the last round of this truncated differential characteristic holds with probability $(2^{-3\times 8})^4 = 2^{-96}$. Furthermore, with an additional 32-bit condition on the active bytes in state S52, we can fix the difference at the output of the permutation, prior to the feed-forward. In this case, only the 16-byte differences in the diagonal bytes of the output of the compression function change for each additional found pair.

In other words, the vector space of differences at the output of the compression function is reduced to a dimension of 128. We use a third outbound phase to satisfy these conditions

in the last round. Since we can find one solution for the white bytes of the 6-round path with an average complexity one, we can find one pair which also satisfies the conditions in the last round with a time and memory complexity 2^{128} operations. Note that we can find up to 2^{32} such pairs with a total complexity of 2^{160} computations in time and 2^{128} memory.

The generic complexity for a random function with 2048 input bits and 512 output bits is given by the limited-birthday algorithm (see Section 7.1.3). The 128 active inputs allow to reach a collision on 256 output bits in 2^{128} operations, so that we reach the collision on the 384 wanted bits by repeating this procedure 2^{128} times. All in all, the generic complexity requires about 2^{256} evaluations of the compression functions. Therefore, the algorithm we have described is a chosen-salt distinguisher for 7 rounds of the ECHO-256 compression function with a complexity of 2^{160} computations in time and uses a memory of 2^{128} elements.

Conclusions

In the first chapters of the document, we have recalled the basics of the differential cryptanalysis techniques and its most notable applications to the cryptanalysis of the AES. Then, we have studied this block cipher in several security models in three different chapters. First, in the most classical standard model where the adversary can only make encryptions/decryptions queries to a blackbox embedding the secret key k. We have shown how to improve the previous best impossible differential cryptanalysis on all variants of the AES. For instance, we have detailed a key-recovery attack on 7 rounds of AES-128 with complexities below 2^{100}, and a new attack on 9 rounds of AES-256 with smaller complexities than previous work. In the next model, we have analyzed the AES in the related-key model where the adversary observes the encryption of messages under an unknown key k and a second key $k \oplus \delta$ where δ is a known difference. Thanks to an algorithm searching automatically for differential characteristics, we have proved impossibility results linked to the provable security of the structure of the AES in this model. Finally, we have scrutinized the possible improvements of the previously known algorithms in the open-key model. In that setting, the adversary knows or can choose the key bits to help him satisfy some nontrivial properties on the AES faster than for an ideal permutation. We have shown how to significantly improve the previous time complexities by efficiently using the freedom degrees in a rebound attack on the AES permutation.

Then, we have been interested in the cryptanalysis of hash functions, and especially by the rebound strategy. The results of the penultimate chapter are twofold. Indeed, we have studied the limitations of the two parts of the rebound technique: the inbound and the outbound phases. First, we have shown that for larger variants of AES-like permutation, it is possible to control one more round in the inbound phase, in the middle of the differential characteristic. This has allowed to improve the best results on the Grøstl hash function, which was still in the final of the SHA-3 selection process. Second, we have shown that relaxing some constraints introduced in the outbound phase, it is possible to increase the probability of this filter and decrease the overall time complexity of all the rebound attacks. This has slightly improved almost all the previous rebound attacks published so far.

Finally, we have detailed new attacks on the ECHO hash function in the last chapter. This hash function has also been a candidate in the SHA-3 competition, but has not been selected for the final. In that chapter, we have described several rebound-based attacks with multiple inbound phases. We have for instance detailed a distinguishing attack on 7 rounds of the inner permutation of the compression function and a collision attack on its variant reduced to 6 rounds.

The different subjects that I have addressed in the document probably leave many open problems that could be interesting research directions for anyone interested. For instance, it would be interesting to study again the differential cryptanalytic framework of Chapter 5 to

check whether we could use a different property for the meet-in-the-middle attack on the AES. So far, we have used a particular differential characteristic to reduce the entropy of the middle rounds, maybe we could imagine using another one, or combine it with impossible differential techniques.

As we have solved an open problem in the Chapter 6 of this document by constructing a distinguisher for 9 rounds of AES-128, the new natural open problem would be to extend it by one more round to reach the full 10 rounds of the AES-128. First, the current technique for 9 rounds should be more investigated to check whether it could be directly extended, or if a completely new approach could be successful. This result would complete the already known distinguishers for the full variants of AES-192 and AES-256 published a few years ago. The main difficulties in the case of the smaller variant are the reduced amount of freedom degrees available to the cryptanalyst and the low bounds of the generic algorithms.

Finally, there may still exist some possible improvements of the rebound attack on AES-like permutations. The open question of whether we could add a third round in the inbound phase in some cases has been answered positively in the Chapter 8, but it would be interesting to check whether we could add another one with similar or different techniques.

Bibliography

[ABNP+11] Mohamed Ahmed Abdelraheem, Céline Blondeau, María Naya-Plasencia, Marion Videau, and Erik Zenner. Cryptanalysis of ARMADILLO2. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 308–326. Springer, December 2011. (Cited on page 124)

[AES97] Announcing Request for Candidate Algorithm Nominations for the Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), September 1997. (Cited on page 58)

[AES01] Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce, November 2001. (Cited on pages 7, 36, 58, and 206)

[AHMNP10] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and María Naya-Plasencia. Quark: A Lightweight Hash. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 1–15. Springer, August 2010. (Cited on page 16)

[AKK+10] Jean-Philippe Aumasson, Emilia Käsper, Lars R. Knudsen, Krystian Matusiewicz, Rune Steinsmo Ødegård, Thomas Peyrin, and Martin Schläffer. Distinguishers for the Compression Function and Output Transformation of Hamsi-256. In Ron Steinfeld and Philip Hawkes, editors, *ACISP 10: 15th Australasian Conference on Information Security and Privacy*, volume 6168 of *Lecture Notes in Computer Science*, pages 87–103. Springer, July 2010. (Cited on page 150)

[BA08] Behnam Bahrak and Mohammad Reza Aref. Impossible differential attack on seven-round AES-128. *IET Information Security*, 2(2):28–32, 2008. (Cited on pages 47, 67, and 80)

[BAK98] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A New Block Cipher Proposal. In Serge Vaudenay, editor, *Fast Software Encryption – FSE'98*, volume 1372 of *Lecture Notes in Computer Science*, pages 222–238. Springer, March 1998. (Cited on page 36)

[BBG+09] Ryad Benadjila, Olivier Billet, Henri Gilbert, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw, and Yannick Seurin. SHA-3 Proposal: ECHO. Submission to NIST (updated), 2009. (Cited on pages 113, 206, 214, and 219)

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer, May 1999. (Cited on page 47)

[BBS05] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. *Journal of Cryptology*, 18(4):291–311, September 2005. (Cited on page 47)

[BCC10] Céline Blondeau, Anne Canteaut, and Pascale Charpin. Differential properties of power functions. *IJICoT*, 1(2):149–170, 2010. (Cited on page 62)

[BCCLC06] Thierry P. Berger, Anne Canteaut, Pascale Charpin, and Yann Laigle-Chapuy. On Almost Perfect Nonlinear Functions Over $F_2{}^n$. *IEEE Transactions on Information Theory*, 52(9):4160–4170, 2006. (Cited on page 62)

[BCD11] Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-Order Differential Properties of Keccak and Luffa. In Antoine Joux, editor, *Fast Software Encryption – FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 252–269. Springer, February 2011. (Cited on pages 186 and 215)

[BD06] Eli Biham and Orr Dunkelman. A Framework for Iterative Hash Functions: HAIFA. In *In Proceedings of Second NIST Cryptographic Hash Workshop*, 2006. (Cited on page 221)

[BD09] Eli Biham and Orr Dunkelman. The SHAvite-3 Hash Function. Submission to NIST (Round 2), 2009. (Cited on page 113)

[BDAP06] Guido Bertoni, Joan Daemen, Gilles Van Assche, and Micha"el Peeters. RadioGat 'un, a Belt-and-Mill Hash Function. NIST - Second Cryptographic Hash Workshop, August 24-25, 2006. (Cited on page 15)

[BDF11] Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque. Automatic Search of Attacks on Round-Reduced AES and Applications. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 169–187. Springer, August 2011. (Cited on page 113)

[BDK01] Eli Biham, Orr Dunkelman, and Nathan Keller. The Rectangle Attack - Rectangling the Serpent. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 340–357. Springer, May 2001. (Cited on pages 50 and 51)

[BDK02] Eli Biham, Orr Dunkelman, and Nathan Keller. New Results on Boomerang and Rectangle Attacks. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption – FSE 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 1–16. Springer, February 2002. (Cited on page 51)

[BDK05a] Eli Biham, Orr Dunkelman, and Nathan Keller. Related-Key Boomerang and Rectangle Attacks. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 507–525. Springer, May 2005. (Cited on pages 53, 84, and 88)

[BDK05b] Eli Biham, Orr Dunkelman, and Nathan Keller. A Related-Key Rectangle Attack on the Full KASUMI. In Bimal K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 443–461. Springer, December 2005. (Cited on pages 51 and 53)

[BDK08] Eli Biham, Orr Dunkelman, and Nathan Keller. A Unified Approach to Related-Key Attacks. In Kaisa Nyberg, editor, *Fast Software Encryption – FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 73–96. Springer, February 2008. (Cited on page 112)

[BDK+10] Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, and Adi Shamir. Key Recovery Attacks of Practical Complexity on AES-256 Variants with up to 10 Rounds. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 299–319. Springer, May 2010. (Cited on page 52)

[BDMW10] K.A. Browning, J.F. Dillon, M.T. McQuistan, and A.J. Wolfe. An APN permutation in dimension six. McGuire, Gary (ed.) et al., Finite fields. Theory and applications. Proceedings of the 9th international conference on finite fields and applications, American Mathematical Society (AMS). Contemporary Mathematics 518., 2010. (Cited on page 62)

[BDPA10] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Note on zero-sum distinguishers of Keccak-f. NIST mailing list, 2010. (Cited on page 150)

[BDPV08] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the Indifferentiability of the Sponge Construction. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, April 2008. (Cited on page 16)

[BGT11] Céline Blondeau, Benoît Gérard, and Jean-Pierre Tillich. Accurate estimates of the data complexity and success probability for various cryptanalyses. *Des. Codes Cryptography*, 59(1-3):3–34, 2011. (Cited on page 41)

[Bih93] Eli Biham. New Types of Cryptoanalytic Attacks Using related Keys (Extended Abstract). In Tor Helleseth, editor, *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 398–409. Springer, May 1993. (Cited on pages 51, 84, and 112)

[Bih94] Eli Biham. New Types of Cryptanalytic Attacks Using Related Keys. *Journal of Cryptology*, 7(4):229–246, 1994. (Cited on pages 51 and 84)

[Bih00] Eli Biham. Cryptanalysis of Patarin's 2-Round Public Key System with S Boxes (2R). In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 408–416. Springer, May 2000. (Cited on page 117)

[Bir04] Alex Biryukov. The Boomerang Attack on 5 and 6-Round Reduced AES. In Dobbertin et al. [DRS05], pages 11–15. (Cited on page 88)

[BK03] Mihir Bellare and Tadayoshi Kohno. A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 491–506. Springer, May 2003. (Cited on page 52)

[BK09] Alex Biryukov and Dmitry Khovratovich. Related-Key Cryptanalysis of the Full AES-192 and AES-256. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 1–18. Springer, December 2009. (Cited on pages 27, 52, 53, 67, 84, 88, and 113)

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, September 2007. (Cited on pages 36 and 113)

[BKL+11] Andrey Bogdanov, Miroslav Knezevic, Gregor Leander, Deniz Toz, Kerem Varici, and Ingrid Verbauwhede. spongent: A Lightweight Hash Function. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 312–325. Springer, September / October 2011. (Cited on page 16)

[BKN09] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolić. Distinguisher and Related-Key Attack on the Full AES-256. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 231–249. Springer, August 2009. (Cited on pages 27, 52, 53, 67, 84, 88, 113, 145, and 161)

[BKR11] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique Cryptanalysis of the Full AES. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, December 2011. (Cited on page 212)

[BM97] Mihir Bellare and Daniele Micciancio. A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 163–192. Springer, May 1997. (Cited on page 150)

[BN10] Alex Biryukov and Ivica Nikolić. Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 322–344. Springer, May 2010. (Cited on pages 84, 85, 88, 113, 114, 115, 135, 136, and 173)

[BN11] Alex Biryukov and Ivica Nikolić. Search for Related-Key Differential Characteristics in DES-Like Ciphers. In Antoine Joux, editor, *Fast Software Encryption – FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 18–34. Springer, February 2011. (Cited on page 113)

[BR00] Paulo S. L. M. Barreto and Vincent Rijmen. The `Whirlpool` Hashing Function. Submitted to NESSIE, September 2000, 2000. (Cited on pages 113 and 184)

[BR11] Paulo S. L. M. Barreto and Vincent Rijmen. `Whirlpool`. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 1384–1385. Springer, 2011. (Cited on pages 184 and 206)

[Bre80] Richard P. Brent. An Improved Monte Carlo Factorization Algorithm. *BIT*, 20:176–184, 1980. (Cited on page 12)

[BRS02] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer, August 2002. (Cited on page 145)

[BRSS10] John Black, Phillip Rogaway, Thomas Shrimpton, and Martijn Stam. An Analysis of the Blockcipher-Based Hash Functions from PGV. *Journal of Cryptology*, 23(4):519–545, October 2010. (Cited on page 145)

[BS91a] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In Alfred J. Menezes and Scott A. Vanstone, editors, *Advances in Cryptology – CRYPTO'90*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, August 1991. (Cited on pages 33 and 119)

[BS91b] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991. (Cited on pages 33 and 119)

[BS93] Eli Biham and Adi Shamir. Differential Cryptanalysis of the Full 16-Round DES. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 487–496. Springer, August 1993. (Cited on pages 33, 34, 42, 45, and 57)

[BS10] Alex Biryukov and Adi Shamir. Structural Cryptanalysis of SASAS. *Journal of Cryptology*, 23(4):505–518, October 2010. (Cited on pages 116 and 117)

[BW00] Alex Biryukov and David Wagner. Advanced Slide Attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 589–606. Springer, May 2000. (Cited on page 37)

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The Random Oracle Methodology, Revisited (Preliminary Version). In *30th Annual ACM Symposium on Theory of Computing*, pages 209–218. ACM Press, May 1998. (Cited on page 144)

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004. (Cited on page 144)

[Cop94] Don Coppersmith. The Data Encryption Standard (DES) and its strength against attacks. *IBM J. Res. Dev.*, 38(3):243–250, May 1994. (Cited on page 34)

[Dae91] Joan Daemen. Limitations of the Even-Mansour Construction (Rump Session). In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology – ASIACRYPT'91*, volume 739 of *Lecture Notes in Computer Science*, pages 495–498. Springer, November 1991. (Cited on page 36)

[Dam90] Ivan Damgård. A Design Principle for Hash Functions. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, August 1990. (Cited on page 13)

[DC98] Joan Daemen and Craig S. K. Clapp. Fast Hashing and Stream Encryption with PANAMA. In Serge Vaudenay, editor, *Fast Software Encryption – FSE'98*, volume 1372 of *Lecture Notes in Computer Science*, pages 60–74. Springer, March 1998. (Cited on page 15)

[DES77] Data Encryption Standard. National Bureau of Standards, NBS FIPS PUB 46, U.S. Department of Commerce, January 1977. (Cited on page 6)

[DFJ12a] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Faster Chosen-Key Distinguishers on Reduced-Round AES. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT*, volume 7668 of *Lecture Notes in Computer Science*, pages 225–243. Springer, 2012. (Cited on pages 28, 31, 96, 143, 145, and 161)

[DFJ12b] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting (extended version). Cryptology ePrint Archive, Report 2012/477, 2012. (Cited on pages 31 and 67)

[DFJ13] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In *EUROCRYPT*, Lecture Notes in Computer Science, 2013. *to appear*. (Cited on pages 26, 31, 47, and 89)

[DGPW12] Alexandre Duc, Jian Guo, Thomas Peyrin, and Lei Wei. Unaligned Rebound Attack: Application to Keccak. In Anne Canteaut, editor, *Fast Software Encryption – FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 402–421. Springer, March 2012. (Cited on page 205)

[DH76] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. (Cited on page 6)

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The Block Cipher Square. In Eli Biham, editor, *Fast Software Encryption – FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, January 1997. (Cited on pages 36, 58, 67, 70, 87, 90, and 149)

[DKS10] Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved Single-Key Attacks on 8-Round AES-192 and AES-256. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 158–176. Springer, December 2010. (Cited on pages 26, 67, 87, 89, 90, 91, 92, 93, 101, and 107)

[DKS12] Orr Dunkelman, Nathan Keller, and Adi Shamir. Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 336–354. Springer, April 2012. (Cited on page 37)

[DLP+09] Joan Daemen, Mario Lamberger, Norbert Pramstaller, Vincent Rijmen, and Frederik Vercauteren. Computational aspects of the expected differential probability of 4-round AES and AES-like ciphers. *Computing*, 85(1-2):85–104, 2009. (Cited on page 43)

[Dob96] Hans Dobbertin. Cryptanalysis of MD4. In Dieter Gollmann, editor, *Fast Software Encryption – FSE'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 53–69. Springer, February 1996. (Cited on page 34)

[DR01] Joan Daemen and Vincent Rijmen. The Wide Trail Design Strategy. In Bahram Honary, editor, *8th IMA International Conference on Cryptography and Coding*, volume 2260 of *Lecture Notes in Computer Science*, pages 222–238. Springer, December 2001. (Cited on page 155)

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002. (Cited on pages 7, 43, 116, 117, 133, 134, 155, 173, and 220)

[DR06a] Joan Daemen and Vincent Rijmen. Two-Round AES Differentials. Cryptology ePrint Archive, Report 2006/039, 2006. (Cited on pages 43 and 45)

[DR06b] Joan Daemen and Vincent Rijmen. Understanding Two-Round Differentials in AES. In Roberto De Prisco and Moti Yung, editors, *SCN 06: 5th International Conference on Security in Communication Networks*, volume 4116 of *Lecture Notes in Computer Science*, pages 78–94. Springer, September 2006. (Cited on pages 43, 45, 155, and 222)

[DR06c] Christophe De Cannière and Christian Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer, December 2006. (Cited on page 113)

[DR07] Joan Daemen and Vincent Rijmen. Probability distributions of correlation and differentials in block ciphers. *J. Mathematical Cryptology*, 1(3):221–242, 2007. (Cited on page 45)

[DR09] Joan Daemen and Vincent Rijmen. New criteria for linear maps in AES-like ciphers. *Cryptography and Communications*, 1(1):47–69, 2009. (Cited on page 45)

[DRS05] Hans Dobbertin, Vincent Rijmen, and Aleksandra Sowa, editors. *Advanced Encryption Standard - AES, 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers*, volume 3373 of *LNCS*. Springer, 2005. (Cited on pages 263 and 270)

[DS08] Hüseyin Demirci and Ali Aydin Selçuk. A Meet-in-the-Middle Attack on 8-Round AES. In Kaisa Nyberg, editor, *Fast Software Encryption – FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 116–126. Springer, February 2008. (Cited on pages 26, 89, 90, 92, and 93)

[DTCB09] Hüseyin Demirci, Ihsan Taskin, Mustafa Coban, and Adnan Baysal. Improved Meet-in-the-Middle Attacks on AES. In Bimal K. Roy and Nicolas Sendrier, editors, *Progress in Cryptology - INDOCRYPT 2009: 10th International Conference in Cryptology in India*, volume 5922 of *Lecture Notes in Computer Science*, pages 144–156. Springer, December 2009. (Cited on page 90)

[Dwo01a] M. Dworkin. Recommandations for Block Cipher Modes of Operation. SP 800-38a, National Institute of Standards and Technology (NIST), 2001. (Cited on page 59)

[Dwo01b] M. Dworkin. Recommandations for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. SP 800-38a, National Institute of Standards and Technology (NIST), 2001. (Cited on page 59)

[Dwo01c] M. Dworkin. Recommandations for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. SP 800-38a, National Institute of Standards and Technology (NIST), 2001. (Cited on page 59)

[Dwo01d] M. Dworkin. Recommandations for Block Cipher Modes of Operation: The CMAC Mode for Authentication. SP 800-38a, National Institute of Standards and Technology (NIST), 2001. (Cited on page 59)

[ElG85] Taher ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985. (Cited on page 5)

[EM91] Shimon Even and Yishay Mansour. A Construction of a Cipher From a Single Pseudorandom Permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology – ASIACRYPT'91*, volume 739 of *Lecture Notes in Computer Science*, pages 210–224. Springer, November 1991. (Cited on page 36)

[EM97] Shimon Even and Yishay Mansour. A Construction of a Cipher from a Single Pseudorandom Permutation. *Journal of Cryptology*, 10(3):151–162, 1997. (Cited on page 36)

[FJP13a] Pierre-Alain Fouque, Jérémy Jean, and Thomas Peyrin. Structural Evaluation of AES and Chosen-Key Distinguisher of 9-round AES-128. In *CRYPTO*, Lecture Notes in Computer Science, 2013. *to appear*. (Cited on pages 27, 28, 31, 112, 143, and 162)

[FJP13b] Pierre-Alain Fouque, Jérémy Jean, and Thomas Peyrin. Structural Evaluation of AES and Chosen-Key Distinguisher of 9-round AES-128 (extended version). Cryptology ePrint Archive, Report 2013/366, 2013. (Cited on pages 27 and 31)

[FKL+00] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. Improved Cryptanalysis of Rijndael. In Bruce Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 213–230. Springer, April 2000. (Cited on pages 67, 71, 72, 74, 87, and 90)

[Flo67] Robert W. Floyd. Nondeterministic Algorithms. *J. ACM*, 14(4):636–644, October 1967. (Cited on page 12)

[GKM+11] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl – a SHA-3 candidate. Submission to NIST (Round 3), 2011. (Cited on pages 113 and 206)

[GL08] Michael Gorski and Stefan Lucks. New Related-Key Boomerang Attacks on AES. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *Progress in Cryptology - INDOCRYPT 2008: 9th International Conference in Cryptology in India*, volume 5365 of *Lecture Notes in Computer Science*, pages 266–278. Springer, December 2008. (Cited on pages 51, 53, and 88)

[GM00] Henri Gilbert and Marine Minier. A Collision Attack on 7 Rounds of Rijndael. In *AES Candidate Conference*, pages 230–241, 2000. (Cited on pages 67, 75, 77, 78, 87, 90, and 92)

[GP10] Henri Gilbert and Thomas Peyrin. Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In Seokhie Hong and Tetsu Iwata, editors, *Fast Software Encryption – FSE 2010*, volume 6147 of *Lecture Notes in Computer Science*, pages 365–383. Springer, February 2010. (Cited on pages 144, 145, 146, 155, 157, 168, 173, 176, 179, 186, 188, 207, 212, 222, 226, 227, and 252)

[GPP11] Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON Family of Lightweight Hash Functions. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 222–239. Springer, August 2011. (Cited on pages 16, 182, 186, 205, 206, 207, and 217)

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, September / October 2011. (Cited on pages 113, 183, 206, 207, and 216)

[GPPR12] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The LED Block Cipher. Cryptology ePrint Archive, Report 2012/600, 2012. (Cited on page 183)

[HNR68] Peter Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis For The Heuristic Determination Of Minimum Cost Paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107, 1968. (Cited on pages 114, 115, and 126)

[HPS98] Jeffrey Hoffstein, Jill Pipher, and JosephH. Silverman. NTRU: A ring-based public key cryptosystem. 1423:267–288, 1998. (Cited on page 5)

[ISO04] ISO. *ISO/IEC 10118-3:2004: Information technology — Security techniques — Hash-functions — Part 3: Dedicated hash-functions*. feb 2004. (Cited on page 112)

[JF11] Jérémy Jean and Pierre-Alain Fouque. Practical Near-Collisions and Collisions on Round-Reduced ECHO-256 Compression Function. In Antoine Joux, editor, *Fast Software Encryption – FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 107–127. Springer, February 2011. (Cited on pages 30, 31, 215, 219, 226, and 227)

[JNP+13] Jérémy Jean, Ivica Nikolic, Thomas Peyrin, Lei Wang, and Shuang Wu. Security Analysis of PRINCE. In *FSE*, Lecture Notes in Computer Science, 2013. *to appear*. (Cited on pages 31 and 51)

[JNPP12] Jérémy Jean, María Naya-Plasencia, and Thomas Peyrin. Improved Rebound Attack on the Finalist Grøstl. In Anne Canteaut, editor, *Fast Software Encryption – FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 110–126. Springer, March 2012. (Cited on pages 29, 31, 145, 173, 179, and 188)

[JNPP13a] Jérémy Jean, María Naya-Plasencia, and Thomas Peyrin. Improved Cryptanalysis of AES-like Permutations. *J. Cryptology*, 2013. *to appear*. (Cited on pages 31, 180, and 186)

[JNPP13b] Jérémy Jean, María Naya-Plasencia, and Thomas Peyrin. Multiple Limited-Birthday Distinguishers and Applications. In *SAC*, Lecture Notes in Computer Science, 2013. *to appear*. (Cited on page 31)

[JNPP13c] Jérémy Jean, María Naya-Plasencia, and Thomas Peyrin. Multiple Limited-Birthday Distinguishers and Applications. Cryptology ePrint Archive, Report 2013/521, 2013. (Cited on page 31)

[JNPS11a] Jérémy Jean, María Naya-Plasencia, and Martin Schläffer. Improved Analysis of ECHO-256. In Ali Miri and Serge Vaudenay, editors, *SAC 2011: 18th Annual International Workshop on Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 19–36. Springer, August 2011. (Cited on pages 30, 31, 56, 219, 226, and 227)

[JNPS11b] Jérémy Jean, María Naya-Plasencia, and Martin Schläffer. Improved Analysis of ECHO-256 (extended version). Cryptology ePrint Archive, Report 2011/422, 2011. (Cited on pages 30, 31, 219, 226, and 227)

[Jou04] Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, August 2004. (Cited on page 14)

[KBN09] Dmitry Khovratovich, Alex Biryukov, and Ivica Nikolić. Speeding up Collision Search for Byte-Oriented Hash Functions. In Marc Fischlin, editor, *Topics in Cryptology – CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 164–181. Springer, April 2009. (Cited on page 113)

[Ker83] Auguste Kerckhoffs. La Cryptographie Militaire. *Journal des Sciences Militaires*, IX:5–38, 1883. (Cited on page 4)

[KHP07] Jongsung Kim, Seokhie Hong, and Bart Preneel. Related-Key Rectangle Attacks on Reduced AES-192 and AES-256. In Alex Biryukov, editor, *Fast Software Encryption – FSE 2007*, volume 4593 of *Lecture Notes in Computer Science*, pages 225–241. Springer, March 2007. (Cited on pages 84 and 88)

[KKS00] John Kelsey, Tadayoshi Kohno, and Bruce Schneier. Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent. In Bruce Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 75–93. Springer, April 2000. (Cited on pages 50 and 51)

[KNPRS10] Dmitry Khovratovich, María Naya-Plasencia, Andrea Röck, and Martin Schläffer. Cryptanalysis of Luffa v2 Components. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *SAC 2010: 17th Annual International Workshop on Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 388–409. Springer, August 2010. (Cited on page 56)

[KNR10] Dmitry Khovratovich, Ivica Nikolić, and Christian Rechberger. Rotational Rebound Attacks on Reduced Skein. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 1–19. Springer, December 2010. (Cited on page 56)

[Knu94] Lars R. Knudsen. Truncated and Higher Order Differentials. In Bart Preneel, editor, *Fast Software Encryption – FSE'94*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, December 1994. (Cited on pages 34, 46, and 119)

[Knu97] Donald E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997. (Cited on page 12)

[Knu98] Lars Knudsen. DEAL - A 128-bit Block Cipher. In *NIST AES Proposal*, 1998. (Cited on page 47)

[KR96] Joe Kilian and Phillip Rogaway. How to Protect DES Against Exhaustive Key Search. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 252–267. Springer, August 1996. (Cited on page 37)

[KR01] Joe Kilian and Phillip Rogaway. How to Protect DES Against Exhaustive Key Search (an Analysis of DESX). *Journal of Cryptology*, 14(1):17–35, 2001. (Cited on page 37)

[KR07] Lars R. Knudsen and Vincent Rijmen. Known-Key Distinguishers for Some Block Ciphers. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 315–324. Springer, December 2007. (Cited on pages 28, 143, 144, and 149)

[KRW99] Lars R. Knudsen, Matthew J. B. Robshaw, and David Wagner. Truncated Differentials and Skipjack. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 165–180. Springer, August 1999. (Cited on page 47)

[KS07] Liam Keliher and Jiayuan Sui. Exact maximum expected differential and linear probability for two-round Advanced Encryption Standard. *IET Information Security*, 1(2):53–57, 2007. (Cited on page 43)

[KW02] Lars R. Knudsen and David Wagner. Integral Cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption – FSE 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 112–127. Springer, February 2002. (Cited on page 67)

[LDKK08] Jiqiang Lu, Orr Dunkelman, Nathan Keller, and Jongsung Kim. New Impossible Differential Attacks on AES. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *Progress in Cryptology - INDOCRYPT 2008: 9th International Conference in Cryptology in India*, volume 5365 of *Lecture Notes in Computer Science*, pages 279–293. Springer, December 2008. (Cited on pages 47, 67, 82, 87, 91, and 94)

[Leu12] Gaëtan Leurent. Analysis of Differential Attacks in ARX Constructions. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 226–243. Springer, 2012. (Cited on page 113)

[LM90] Xuejia Lai and James L. Massey. A Proposal for a New Block Encryption Standard. In Ivan Damgård, editor, *Advances in Cryptology – EUROCRYPT'90*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer, May 1990. (Cited on page 47)

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. Markov Ciphers and Differential Cryptoanalysis. In Donald W. Davies, editor, *Advances in Cryptology – EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer, April 1991. (Cited on pages 34, 41, 42, 43, 44, and 46)

[LMR+09] Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 126–143. Springer, December 2009. (Cited on pages 56, 188, and 222)

[LMR+10] Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. The Rebound Attack and Subspace Distinguishers: Application to Whirlpool. Cryptology ePrint Archive, Report 2010/198, 2010. (Cited on pages 207 and 217)

[LSWD04] Tri Van Le, Rüdiger Sparr, Ralph Wernsdorf, and Yvo Desmedt. Complementation-Like and Cyclic Properties of AES Round Functions. In Dobbertin et al. [DRS05], pages 128–141. (Cited on page 220)

[Luc04] Stefan Lucks. Design Principles for Iterated Hash Functions. Cryptology ePrint Archive, Report 2004/253, 2004. (Cited on page 14)

[Mat94a] Mitsuru Matsui. The First Experimental Cryptanalysis of the Data Encryption Standard. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 1–11. Springer, August 1994. (Cited on pages 33 and 57)

[Mat94b] Mitsuru Matsui. On Correlation Between the Order of S-boxes and the Strength of DES. In Alfredo De Santis, editor, *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 366–375. Springer, May 1994. (Cited on pages 113 and 126)

[MDRMH10] Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen, and Mahmoud Modarres-Hashemi. Improved Impossible Differential Cryptanalysis of 7-Round AES-128. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010: 11th International Conference in Cryptology in India*, volume 6498 of *Lecture Notes in Computer Science*, pages 282–291. Springer, December 2010. (Cited on pages 47, 67, 82, 84, and 87)

[Mer90] Ralph C. Merkle. A Certified Digital Signature. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, August 1990. (Cited on pages 12 and 13)

[Min02] M. Minier. *Preuves d'Analyses et de Sécurité en Cryptologie à Clé Secrète*. PhD thesis, Université de Limoges, France, 2002. (Cited on page 78)

[MMO85] S.M. Matyas, C.H. Meyer, and J. Oseas. Generating Strong One-Way Functions With Cryptographic Algorithm - IBM Technical Disclosure Bulletin, Vol. 27, No. 10A, 1985. (Cited on page 112)

[MNPN+09] Krystian Matusiewicz, María Naya-Plasencia, Ivica Nikolić, Yu Sasaki, and Martin Schläffer. Rebound Attack on the Full Lane Compression Function. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 106–125. Springer, December 2009. (Cited on page 56)

[MP08] Stéphane Manuel and Thomas Peyrin. Collisions on SHA-0 in One Hour. In Kaisa Nyberg, editor, *Fast Software Encryption – FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 16–35. Springer, February 2008. (Cited on page 113)

[MPRS09] Florian Mendel, Thomas Peyrin, Christian Rechberger, and Martin Schläffer. Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *SAC 2009: 16th Annual International Workshop on Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 16–35. Springer, August 2009. (Cited on pages 145, 149, 153, 207, 214, 226, 227, and 252)

[MRST09] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Orr Dunkelman, editor, *Fast Software Encryption – FSE 2009*, volume 5665 of *Lecture Notes in Computer Science*, pages 260–276. Springer, February 2009. (Cited on pages 28, 55, 96, 145, 149, 151, 155, 157, and 179)

[MS90] Willi Meier and Othmar Staffelbach. Nonlinearity Criteria for Cryptographic Functions. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 549–562. Springer, April 1990. (Cited on page 62)

[NP10a] Mridul Nandi and Souradyuti Paul. Speeding Up the Wide-Pipe: Secure and Fast Hashing. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010: 11th International Conference in Cryptology in India*, volume 6498 of *Lecture Notes in Computer Science*, pages 144–162. Springer, December 2010. (Cited on page 15)

[NP10b] María Naya-Plasencia. How to Improve Rebound Attacks. Cryptology ePrint Archive, Report 2010/607, 2010. (extended version). (Cited on pages 188 and 200)

[NP11] María Naya-Plasencia. How to Improve Rebound Attacks. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 188–205. Springer, August 2011. (Cited on pages 188, 200, 207, 214, 226, and 249)

[NWW13] Ivica Nikolic, Lei Wang, and Shuang Wu. Cryptanalysis of Round-Reduced LED. In *FSE*, Lecture Notes in Computer Science, 2013. *to appear*. (Cited on pages 207 and 216)

[Nyb91] Kaisa Nyberg. Perfect Nonlinear S-Boxes. In Donald W. Davies, editor, *Advances in Cryptology – EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 378–386. Springer, April 1991. (Cited on page 62)

[Nyb93] Kaisa Nyberg. Differentially Uniform Mappings for Cryptography. In Tor Helleseth, editor, *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 55–64. Springer, May 1993. (Cited on page 62)

[Pey10] Thomas Peyrin. Improved Differential Attacks for ECHO and Grøstl. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 370–392. Springer, August 2010. (Cited on pages 181 and 216)

[PG97] Jacques Patarin and Louis Goubin. Asymmetric cryptography with S-Boxes. In Yongfei Han, Tatsuaki Okamoto, and Sihan Qing, editors, *ICICS 97: 1st International Conference on Information and Communication Security*, volume 1334 of *Lecture Notes in Computer Science*, pages 369–380. Springer, November 1997. (Cited on page 117)

[PGV94] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer, August 1994. (Cited on pages 20, 112, and 145)

[Pie90] Josef Pieprzyk. Non-linearity of Exponent Permutations. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 80–92. Springer, April 1990. (Cited on page 62)

[Pol75] John Pollard. A Monte Carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334, 1975. (Cited on page 12)

[Rab78] Michael O. Rabin. Digitalized Signatures. In Richard J. Lipton and Richard A. DeMillo, editors, *Foundations of Secure Computation*, pages 155–166. Acedemic Press, New York, 1978. (Cited on page 13)

[RDP+96] Vincent Rijmen, Joan Daemen, Bart Preneel, Anton Bossalaers, and Erik De Win. The Cipher SHARK. In Dieter Gollmann, editor, *Fast Software Encryption – FSE'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 99–111. Springer, February 1996. (Cited on page 58)

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signature and Public-Key Cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978. (Cited on page 5)

[RTV13] Vincent Rijmen, Deniz Toz, and Kerem Varici. On the Four-Round AES Characteristics. In *WCC 2013*, Bergen, Norway, April 2013. (Cited on page 43)

[RW03] Ben Reichardt and David Wagner. Markov Truncated Differential Cryptanalysis of Skipjack. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 110–128. Springer, August 2003. (Cited on page 47)

[Sch89] Claus-Peter Schnorr. Efficient Identification and Signatures for Smart Cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, August 1989. (Cited on page 5)

[Sch10] Martin Schläffer. Subspace Distinguisher for 5/8 Rounds of the ECHO-256 Hash Function. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *SAC 2010: 17th Annual International Workshop on Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 369–387. Springer, August 2010. (Cited on pages 215, 223, 224, 227, 228, and 230)

[Sch11] Martin Schläffer. Updated Differential Analysis of `Grøstl`. `Grøstl` website, January 2011. (Cited on pages 207 and 215)

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949. (Cited on pages 3 and 17)

[SIH+11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An Ultra-Lightweight Blockcipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 342–357. Springer, September / October 2011. (Cited on page 113)

[Ski98] SKIPJACK and KEA Algorithm Specifications Version 2.0. National Institute of Standards and Technology (NIST), May 1998. (Cited on page 47)

[SLW+10] Yu Sasaki, Yang Li, Lei Wang, Kazuo Sakiyama, and Kazuo Ohta. Non-full-active Super-Sbox Analysis: Applications to ECHO and Grøstl. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 38–55. Springer, December 2010. (Cited on pages 145, 158, 179, 186, 192, 207, 214, 215, 226, and 230)

[SSNO12] Siamak Fayyaz Shahandashti, Reihaneh Safavi-Naini, and Philip Ogunbona. Private Fingerprint Matching (Short Paper). In Willy Susilo, Yi Mu, and Jennifer Seberry, editors, *ACISP 12: 17th Australasian Conference on Information Security and Privacy*, volume 7372 of *Lecture Notes in Computer Science*, pages 426–433. Springer, July 2012. (Cited on page 5)

[TWP07] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 Bit WEP in Less Than 60 Seconds. In Sehun Kim, Moti Yung, and Hyung-Woo Lee, editors, *WISA 07: 8th International Workshop on Information Security Applications*, volume 4867 of *Lecture Notes in Computer Science*, pages 188–202. Springer, August 2007. (Cited on page 52)

[Vau98] Serge Vaudenay. Provable Security for Block Ciphers by Decorrelation. In Michel Morvan, Christoph Meinel, and Daniel Krob, editors, *STACS*, volume 1373 of *Lecture Notes in Computer Science*, pages 249–275. Springer, 1998. (Cited on page 51)

[Vau03] Serge Vaudenay. Decorrelation: A Theory for Block Cipher Security. *Journal of Cryptology*, 16(4):249–286, September 2003. (Cited on page 51)

[vOW99] Paul C. van Oorschot and Michael J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *Journal of Cryptology*, 12(1):1–28, 1999. (Cited on page 12)

[Wag99] David Wagner. The Boomerang Attack. In Lars R. Knudsen, editor, *Fast Software Encryption – FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer, March 1999. (Cited on pages 48, 50, and 51)

[Wag02] David Wagner. A Generalized Birthday Problem. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer, August 2002. (Cited on pages 150, 242, 247, and 256)

[WLF+05] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer, May 2005. (Cited on page 34)

[WLH11] Yongzhuang Wei, Jiqiang Lu, and Yupu Hu. Meet-in-the-Middle Attack on 8 Rounds of the AES Block Cipher under 192 Key Bits. In Feng Bao and Jian Weng, editors, *ISPEC*, volume 6672 of *Lecture Notes in Computer Science*, pages 222–232. Springer, 2011. (Cited on page 93)

[WY05] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, May 2005. (Cited on pages 34 and 113)

[WYY05a] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, August 2005. (Cited on pages 34 and 113)

[WYY05b] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient Collision Search Attacks on SHA-0. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer, August 2005. (Cited on page 34)

[Yuv79] Gideon Yuval. How to Swindle Rabin. *Cryptologia*, 3:187–189, 1979. (Cited on page 12)

Appendices

APPENDIX A

Advanced Encryption Standard

A.1 AES S-Box lookup table

Table A.1: The AES S-Box (see Section 4.2.3).

63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

A.2 PRESENT S-Box lookup table

Table A.2: The 4-bit PRESENT S-Box.

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

A.3 Whirlpool S-Box lookup table

Table A.3: The `Whirlpool` S-Box.

18	23	c6	E8	87	B8	01	4F	36	A6	d2	F5	79	6F	91	52
60	Bc	9B	8E	A3	0c	7B	35	1d	E0	d7	c2	2E	4B	FE	57
15	77	37	E5	9F	F0	4A	dA	58	c9	29	0A	B1	A0	6B	85
Bd	5d	10	F4	cB	3E	05	67	E4	27	41	8B	A7	7d	95	d8
FB	EE	7c	66	dd	17	47	9E	cA	2d	BF	07	Ad	5A	83	33
63	02	AA	71	c8	19	49	d9	F2	E3	5B	88	9A	26	32	B0
E9	0F	d5	80	BE	cd	34	48	FF	7A	90	5F	20	68	1A	AE
B4	54	93	22	64	F1	73	12	40	08	c3	Ec	dB	A1	8d	3d
97	00	cF	2B	76	82	d6	1B	B5	AF	6A	50	45	F3	30	EF
3F	55	A2	EA	65	BA	2F	c0	dE	1c	Fd	4d	92	75	06	8A
B2	E6	0E	1F	62	d4	A8	96	F9	c5	25	59	84	72	39	4c
5E	78	38	8c	d1	A5	E2	61	B3	21	9c	1E	43	c7	Fc	04
51	99	6d	0d	FA	dF	7E	24	3B	AB	cE	11	8F	4E	B7	EB
3c	81	94	F7	B9	13	2c	d3	E7	6E	c4	03	56	44	7F	A9
2A	BB	c1	53	dc	0B	9d	6c	31	74	F6	46	Ac	89	14	E1
16	3A	69	09	70	B6	d0	Ed	cc	42	98	A4	28	5c	F8	86

List of Figures

1.1 Chiffre de César . . . 2
1.2 Canal de communication . . . 4
1.3 Attaque par le milieu . . . 6
1.4 Fonction de hachage . . . 7
1.5 Table de hachage . . . 10
1.6 Vue schématique d'une fonction de hachag . . . 12
1.7 Construction de Merkle-Damgd̊ . . . 13
1.8 Construction de multicollisions . . . 14
1.9 La construction sponge . . . 15
1.10 Algorithme de chiffrement par bloc itéré à clefs alternantes . . . 17
1.11 Le mode ECB . . . 18
1.12 Démonstration des modes ECB et CBC sur des images présentants une forte structure . . . 18
1.13 Le mode CBC . . . 19
1.14 Modes opératoires pour fonctions de compression à base de chiffrement par bloc . . . 20
1.15 Attaque par le milieu . . . 21
1.16 Modèles d'attaques en clef secrète . . . 22
1.17 Modèles d'attaques en clef ouverte . . . 23

3.1 Key-alternating block cipher . . . 36
3.2 Even-Mansour construction . . . 37
3.3 Even-Mansour simplified construction . . . 37
3.4 Differential and differential characteristic . . . 38
3.5 Key recovery and differential cryptanalysis . . . 40
3.6 The boomerang attack . . . 49
3.7 The related-key boomerang attack. . . . 53
3.8 Overview of the rebound attack . . . 55

4.1 Ordering of the bytes in one AES state. . . . 59
4.2 Encryption with AES . . . 59
4.3 Key schedules of the three variants of the AES . . . 60
4.4 One round of AES: description of the round function f . . . 61
4.5 Visual representations of two difference distribution tables . . . 64
4.6 Truncated difference representation in AES . . . 66
4.7 Transition $1 \rightarrow 4$ in the MixColumns . . . 66
4.8 Generalized round function of an AES-like permutation with $t = 8$. . . . 67
4.9 Integral distinguisher on 3-round AES . . . 68
4.10 Integral attack on 4-round AES . . . 69
4.11 Integral attack on 5-round AES . . . 70
4.12 Integral attack on 6-round AES . . . 71
4.13 Basic summation for the integral attack . . . 73
4.14 Improved summation for the integral attack . . . 73
4.15 Functional distinguisher from Gilbert and Minier for 3 rounds of AES. . . . 75
4.16 Gilbert and Minier attack on 7-round AES . . . 78
4.17 Impossible differential characteristic on 4-round AES . . . 79
4.18 Impossible differential on 4-round AES . . . 80
4.19 Differential characteristic used in the Bahrak and Aref attack . . . 81
4.20 Impossible differential attack on 7-round AES by Mala et al. . . . 83
4.21 Relatad-key boomerang attack on 7-round AES-128 . . . 85

5.1 Functional relation in 4-round AES . . . 90
5.2 General scheme of the meet-in-the-middle attack on AES . . . 92
5.3 Middle rounds from the 7-round attack on AES . . . 93
5.4 7-round characteristic used in the attacks on AES . . . 97
5.5 Complete 7-round characteristic for the simple attack . . . 99
5.6 Complete 7-round truncated differential characteristic used in the efficient attack. . . . 101
5.7 8-round characteristic fir the 8-round attacks . . . 105
5.8 Five steps of the key scheduling algorithm of AES-192 . . . 107
5.9 Four steps of the key scheduling algorithm of AES-256. . . . 108
5.10 9-round characteristic for the 9-round attack on AES-256. . . . 110

6.1 Tree representation of a set of differential characteristics . . . 114
6.2 Graph representation of a set of differential characteristics . . . 115
6.3 SASAS construction . . . 116
6.4 ASASA construction . . . 117
6.5 One round of the generic SPN and AES-like ciphers. . . . 119
6.6 Examples of simplified versions of the two graphs G and G_5 . . . 122
6.7 Example of graph product to build G . . . 123
6.8 Example of shortest paths in G_5 . . . 126
6.9 Example of linear incompatibility in the case of AES-128 . . . 127
6.10 Compressions of an AES state . . . 131

6.11 Best 5-round truncated differential characteristic for AES-128 with 11 active S-Boxes. . . . 133
6.12 Best 8-round truncated differential characteristic for AES-128 with 21 active S-Boxes. . . . 134
6.13 Best 10-round truncated differential characteristic for AES-128 with 25 active S-Boxes. . . 134
6.14 Variables used as a basis in the system resolution 136
6.15 Best differential characteristic on 2-round AES-128 137
6.16 Best differential characteristic on 3-round AES-128 138
6.17 First best differential characteristic on 4-round AES-128 139
6.18 Another best differential characteristic on 4-round AES-128 140
6.19 Best differential characteristic on 5-round AES-128 141

7.1 General framework to apply the rebound technique 146
7.2 Generalized collision problem for the limited birthday problem 147
7.3 Chart of the $\log_2$ time complexities for the limited-birthday algorithm 148
7.4 The full 7-round integral distinguisher for an AES-like permutation 149
7.5 Integral distinguisher for 7 rounds of an AES-like permutation 150
7.6 Characteristic for the 7-round known-key distinguisher on AES-like permutations 151
7.7 Generic constraints required for a random permutation 152
7.8 Second round of the 7-round characteristic used in the known-key distinguisher 154
7.9 8-round characteristic used in the known-key distinguisher on AES-like permutations . . . 156
7.10 The Super-SBox construction 156
7.11 Non-fully-active truncated differential characteristic on 8 rounds 158
7.12 Sparse Super-SBox differential 159
7.13 7-round characteristic used in the chosen-key distinguisher on AES 162
7.14 First step of the distinguishing algorithm 164
7.15 Second step of the distinguishing algorithm 165
7.16 Generating a compatible key for AES-128 165
7.17 Generating a compatible key for AES-256 168
7.18 8-round characteristic used in the chosen-key distinguisher on AES-128 169
7.19 Chosen-key Super-SBox differential 170
7.20 Middle round of the 7-round characteristic of the chosen-key distinguisher of AES-128 . . 171
7.21 Differential characteristic of 9-round AES-128 used in the distinguisher 174

8.1 The compression function of Grøstl using the permutations P and Q 181
8.2 The sponge construction. 183
8.3 The LED-64 block cipher 184
8.4 The LED-128 block cipher 184
8.5 One round of the Whirlpool hash function inner permutation. 185
8.6 Characteristic used in the 9-round attack on AES-like permutations 187
8.7 Inbound phase for the 9-round distinguishing attack 189
8.8 Steps of the merging process for the 9-round distinguisher 190
8.9 Plot of the two polynomials P_t and Q_t 190
8.10 Non-fully-active truncated differential characteristic on 9 rounds 192
8.11 Non-fully-active inbound phase for the 9-round distinguisher attack 194
8.12 Characteristic used in the 10-round attack on the permutation of Grøstl-256 199
8.13 Inbound phase for the 10-round distinguishing attack on Grøstl 200
8.14 First step of the guess and determine algorithm 201
8.15 Second step of the guess and determine algorithm 202
8.16 Third step of the guess and determine algorithm 203
8.17 Fourth step of the guess and determine algorithm 204
8.18 Possible inputs and outputs of the relaxed generic distinguisher 208
8.19 Generalized 9-round truncated differential characteristic for the relatex outbound 211
8.20 Differential characteristic for the 8-round known-key distinguisher for AES-128 213

8.21 Differential characteristic for the 8-round chosen-key distinguisher for AES-128 213
8.22 19-round truncated differential characteristic for LED-64. 216
8.23 Characteristic used in the 10-round attack on Whirlpool 217
8.24 Characteristic used in the 7.5-round attack on Whirlpool 218

9.1 The internal state of ECHO is seen as a square matrix 4×4 of AES states of 128 bits. 220
9.2 One round of the ECHO permutation 220
9.3 Compression function of ECHO-256. 221
9.4 Compression function of ECHO-512. 221
9.5 Alternative view of one round of the inner permutation of ECHO. 222
9.6 The SuperMixColumns layer 224
9.7 Characteristic used in the 4-round attack on ECHO-256 229
9.8 A sparse differential in the **Super-SBox** with only one active input byte. 230
9.9 Characteristic for a single BigColumn 232
9.10 The SuperShiftRows layer 233
9.11 Part of the second inbound in the 4-round attack on ECHO-256 233
9.12 Second inbound in the 4-round attack on ECHO-256 233
9.13 Feed-forward after the inner permutation reduced to 4 rounds 235
9.14 Collision in the feed-forward of 4-round ECHO-256 235
9.15 Final step of the collision attack of ECHO-256 compression function 237
9.16 Characteristic for the 5-round collision attack on ECHO-256 hash function 241
9.17 Merging the inbound phases. 245
9.18 Characteristic used in the 7-round attack on the ECHO-256 compression function 248
9.19 Rebound attack on 7 rounds of the ECHO compression function. 253

List of Tables

1.1 Complexité des attaques génériques sur une fonction de hachage . . . 11

3.1 Complexity of boomerang and amplified boomerang attacks . . . 51

4.1 Best cryptanalytic results on reduced AES variants in the secret-key model. . . . 87

4.2 Best cryptanalytic results on reduced AES variants in the related-key model. . . . 88

6.1 Structural evaluation of the AES-128 in the related-key model . . . 133
6.2 Best related-key differential attacks on AES-128 . . . 137
6.3 Best differential characteristic on 2-round AES-128 . . . 138
6.4 Best differential characteristic on 3-round AES-128 . . . 138
6.5 Pair of messages for the 3-round characteristic . . . 139
6.6 Best differential characteristic on 5-round AES-128 . . . 140
6.7 Pair of message for the 5-round characteristic . . . 141

7.1 Summary of rebound-based algorithms for AES-like permutations . . . 145
7.2 Pair of messages for the 7-round chosen-key distinguisher of AES-128 . . . 166
7.3 Pair of messages for the 8-round chosen-key distinguisher of AES-128 . . . 172
7.4 Differential characteristic of 9-round AES-128 used in the distinguisher . . . 175
7.5 Pair of keys used in the 9-round chosen-key distinguisher on AES-128 . . . 176

8.1 Variants of the PHOTON hash function family . . . 183
8.2 Rebound attacks with an improved algorithm for the inbound phase . . . 186
8.3 Examples of parameters for our improved inbound algorithm . . . 197
8.4 Known and improved results for various rebound-based attacks on AES-based primitives. . . 207
8.5 Complexities of some algorithms solving the multiple limited-birthday problem . . . 210

9.1 Linear constraint required to ensure the consistency of the system of linear equations $\mathcal{S}_{\alpha,\beta}$. 226
9.2 Best known cryptanalytic results on ECHO-256 to date. . . . 226
9.3 Example of a near-collision on 384 bits of ECHO-256 compression function . . . 238

A.1 The AES S-Box . . . 277
A.2 The 4-bit PRESENT S-Box. . . . 277
A.3 The Whirlpool S-Box. . . . 278

List of Algorithms

5.1 Construction of the tables 97
5.2 A simple attack on 7-round AES 100
5.3 An efficient attack on 7-round AES 103
6.1 Search all the shortest paths in G_r 125
7.1 Distinguishing algorithm for 9 rounds of AES-128 175

Author Index

Abdelraheem, Mohamed Ahmed 124

Adleman, Leonard M. 5

Akishita, Toru 113

Anderson, Ross J. 36

Aref, Mohammad Reza 47, 67, 80

Assche, G. Van 150

Assche, Gilles Van 15

Aumasson, Jean-Philippe 16, 150

Bahrak, Behnam 47, 67, 80

Barreto, Paulo S. L. M. 113, 184, 206

Baysal, Adnan 90

Bellare, Mihir 52, 150

Benadjila, Ryad 113, 206, 214, 219

Berger, Thierry P. 62

Bertoni, G. 150

Bertoni, Guido 15, 16

Biham, Eli 33, 34, 36, 42, 45, 47, 50, 51, 53, 57, 84, 88, 112, 113, 117, 119, 221

Billet, Olivier 113, 206, 214, 219

Biryukov, Alex 27, 37, 47, 52, 53, 67, 84, 85, 88, 113–117, 135, 136, 145, 161, 173

Black, John 145

Blondeau, Céline 41, 62, 124

Bogdanov, Andrey 16, 36, 113, 212

Bossalaers, Anton 58

Bouillaguet, Charles 113

Boura, Christina 186, 215

Brent, Richard P. 12

Browning, K.A. 62

Canetti, Ran 144

Canteaut, Anne 62, 186, 215

Charpin, Pascale 62

Chen, Hui 34

Clapp, Craig S. K. 15

Coban, Mustafa 90

Coppersmith, Don 34

Daemen, J. 150

Daemen, Joan 7, 15, 16, 36, 43, 45, 58, 67, 70, 87, 90, 116, 117, 133, 134, 149, 155, 173, 220, 222

Dakhilalian, Mohammad 47, 67, 82, 84, 87

Damgård, Ivan 13

De Cannière, Christophe 113, 186, 215

De Win, Erik 58

Demirci, Hüseyin 26, 89, 90, 92, 93

Derbez, Patrick 26, 28, 31, 47, 67, 89, 96, 113, 143, 145, 161

Desmedt, Yvo 220

Diffie, Whitfield 6

Dillon, J.F. 62

Dobbertin, Hans 34

Duc, Alexandre 205

Dunkelman, Orr 26, 37, 47, 50–53, 67, 82, 84, 87–94, 101, 107, 112, 113, 221

Dworkin, M. 59

ElGamal, Taher 5

Even, Shimon 36

Feng, Dengguo 34

Ferguson, Niels 67, 71, 72, 74, 87, 90

Floyd, Robert W. 12

Fouque, Pierre-Alain 26–28, 30, 31, 47, 67, 89, 96, 112, 113, 143, 145, 161, 162, 215, 219, 226, 227

Gauravaram, Praveen 113, 206

Gérard, Benoît 41

Gilbert, Henri 67, 75, 77, 78, 87, 90, 92, 113, 144–146, 155, 157, 168, 173, 176, 179, 186, 188, 206, 207, 212, 214, 219, 222, 226, 227, 252

Goldreich, Oded 144

Gorski, Michael 51, 53, 88

Goubin, Louis 117

Govaerts, René 20, 112, 145

Guo, Jian 16, 113, 182, 183, 186, 205–207, 216, 217

Halevi, Shai 144

Hart, Peter 114, 115, 126

Hellman, Martin E. 6

Henzen, Luca 16

Hiwatari, Harunaga 113

Hoffstein, Jeffrey 5

Hong, Seokhie 84, 88

Hu, Yupu 93

ISO 112

Isobe, Takanori 113

Jean, Jérémy 26–28, 30, 31, 47, 51, 67, 89, 112, 143, 162, 180, 186, 219, 226, 227

Joux, Antoine 14

Käsper, Emilia 150

Keliher, Liam 43

Keller, Nathan 26, 37, 47, 50–53, 67, 82, 84, 87–94, 101, 107, 112

Kelsey, John 50, 51, 67, 71, 72, 74, 87, 90

Kerckhoffs, Auguste 4

Khovratovich, Dmitry 27, 52, 53, 56, 67, 84, 88, 113, 145, 161, 212

Kilian, Joe 37

Kim, Jongsung 47, 67, 82, 84, 87, 88, 91, 94

Knezevic, Miroslav 16

Knudsen, Lars 47

Knudsen, Lars R. 28, 34, 36, 46, 47, 58, 67, 70, 87, 90, 113, 119, 143, 144, 149, 150, 206

Knuth, Donald E. 12

Kohno, Tadayoshi 50–52

Lai, Xuejia 34, 41–44, 46, 47

Laigle-Chapuy, Yann 62

Lamberger, Mario 43, 56, 188, 207, 217, 222

Le, Tri Van 220

Leander, Gregor 16, 36, 113

Leurent, Gaëtan 113

Li, Yang 145, 158, 179, 186, 192, 207, 214, 215, 226, 230

Lu, Jiqiang 47, 67, 82, 87, 91, 93, 94

Lucks, Stefan 14, 51, 53, 67, 71, 72, 74, 87, 88, 90

Macario-Rat, Gilles 113, 206, 214, 219

Mala, Hamid 47, 67, 82, 84, 87

Mansour, Yishay 36

Manuel, Stéphane 113

Massey, James L. 34, 41–44, 46, 47

Matsui, Mitsuru 33, 57, 113, 126

Matusiewicz, Krystian 56, 113, 150, 206

Matyas, S.M. 112

McQuistan, M.T. 62

Meier, Willi 16, 62

Mendel, Florian 28, 55, 56, 96, 113, 145, 149, 151, 153, 155, 157, 179, 188, 206, 207, 214, 217, 222, 226, 227, 252

Merkle, Ralph C. 12, 13

Meyer, C.H. 112

Micciancio, Daniele 150

Minier, M. 78

Minier, Marine 67, 75, 77, 78, 87, 90, 92

Mitsuda, Atsushi 113

Modarres-Hashemi, Mahmoud 47, 67, 82, 84, 87

Murphy, Sean 34, 41–44, 46

Nandi, Mridul 15

Naya-Plasencia, María 16, 29–31, 56, 124, 145, 173, 179, 188, 200, 207, 214, 219, 226, 227, 249

Nikolic, Ivica 31, 51, 207, 216

Nilsson, Nils 114, 115, 126

Nyberg, Kaisa 62

Ødegård, Rune Steinsmo 150

Ogunbona, Philip 5

Ohta, Kazuo 145, 158, 179, 186, 192, 207, 214, 215, 226, 230

Oseas, J. 112

Paar, Christof 36, 113

Patarin, Jacques 117

Paul, Souradyuti 15

Peeters, M. 150

Peeters, Michael 16

Peeters, Micha"el 15

Peyrin, Thomas 16, 27–29, 31, 51, 112, 113, 143–146, 149, 150, 153, 155, 157, 162, 168, 173, 176, 179–183, 186, 188, 205–207, 212, 214, 216, 217, 219, 222, 226, 227, 252

Pieprzyk, Josef 62

Pipher, Jill 5

Pollard, John 12

Poschmann, Axel 16, 36, 113, 182, 183, 186, 205–207, 216, 217

Pramstaller, Norbert 43

Preneel, Bart 20, 58, 84, 88, 112, 145

Pyshkin, Andrei 52

Rabin, Michael O. 13

Raphael, Bertram 114, 115, 126

Rechberger, Christian 28, 55, 56, 96, 113, 145, 149, 151, 153, 155, 157, 179, 188, 206, 207, 212, 214, 217, 222, 226, 227, 252

Reichardt, Ben 47

Rijmen, Vincent 7, 28, 36, 43, 45, 47, 56, 58, 67, 70, 82, 84, 87, 90, 113, 116, 117, 133, 134, 143, 144, 149, 155, 173, 184, 188, 206, 207, 217, 220, 222

Rivest, Ronald L. 5

Robshaw, Matt 113, 183, 206, 214, 219

Robshaw, Matthew J. B. 36, 47, 113, 183, 206, 207, 216

Röck, Andrea 56

Rogaway, Phillip 37, 145

Safavi-Naini, Reihaneh 5

Sakiyama, Kazuo 145, 158, 179, 186, 192, 207, 214, 215, 226, 230

Sasaki, Yu 56, 145, 158, 179, 186, 192, 207, 214, 215, 226, 230

Schläffer, Martin 28, 30, 31, 55, 56, 96, 145, 149–151, 153, 155, 157, 179, 188, 207, 214, 215, 219, 222–224, 226–228, 230, 252

Schneier, Bruce 50, 51, 67, 71, 72, 74, 87, 90

Schnorr, Claus-Peter 5

Selçuk, Ali Aydin 26, 89, 90, 92, 93

Seurin, Yannick 36, 113, 206, 214, 219

Shahandashti, Siamak Fayyaz 5

Shamir, Adi 5, 26, 33, 34, 37, 42, 45, 47, 52, 57, 67, 87, 89–93, 101, 107, 116, 117, 119

Shannon, Claude E. 3, 17

Shibutani, Kyoji 113

Shirai, Taizo 113

Shrimpton, Thomas 145

Silverman, JosephH. 5

Sparr, Rüdiger 220

Staffelbach, Othmar 62

Stam, Martijn 145

Stay, Michael 67, 71, 72, 74, 87, 90

Sui, Jiayuan 43

Taskin, Ihsan 90

Tews, Erik 52

Thomsen, Søren S. 28, 55, 96, 113, 145, 149, 151, 155, 157, 179, 206

Tillich, Jean-Pierre 41

Toz, Deniz 16, 43

Van Assche, Gilles 16

van Oorschot, Paul C. 12

Vandewalle, Joos 20, 112, 145

Varici, Kerem 16, 43

Vaudenay, Serge 51

Verbauwhede, Ingrid 16

Vercauteren, Frederik 43

Videau, Marion 124

Vikkelsoe, C. 36, 113

Wagner, David 37, 47, 48, 50, 51, 67, 71, 72, 74, 87, 90, 150, 242, 247, 256

Wang, Lei 31, 51, 145, 158, 179, 186, 192, 207, 214–216, 226, 230

Wang, Xiaoyun 34, 113

Wei, Lei 205

Wei, Yongzhuang 93

Weinmann, Ralf-Philipp 52

Wernsdorf, Ralph 220

Whiting, Doug 67, 71, 72, 74, 87, 90

Wiener, Michael J. 12

Wolfe, A.J. 62

Wu, Shuang 31, 51, 207, 216

Yin, Yiqun Lisa 34, 113

Yu, Hongbo 34, 113

Yu, Xiuyuan 34

Yuval, Gideon 12

Zenner, Erik 124

Résumé

Dans cette thèse, nous nous intéressons à la cryptanalyse de certaines primitives de cryptographie symétrique qui utilisent les concepts de construction du schéma de chiffrement AES. Nous commençons par une analyse de l'AES lui-même dans trois modèles de sécurité différents : le modèle standard, le modèle à clefs reliées et le modèle ouvert. Dans le modèle standard, où l'adversaire cherche à récupérer la clef secrète, nous décrivons les meilleures attaques différentielles existantes sur cet algorithme de chiffrement, en améliorant les attaques différentielles précédemment publiées. Ensuite, nous procédons à une analyse structurelle de l'AES dans le modèle à clefs reliées. Nous montrons des résultats d'impossibilité, indiquant que l'on ne peut pas prouver la sécurité de la structure de l'AES contre les attaques différentielles dans ce modèle. Enfin, dans le modèle ouvert, nous proposons le premier distingueur pour neuf tours d'AES-128, ce qui résout un problème ouvert depuis plusieurs années dans la communauté symétrique.

Dans une deuxième partie, nous analysons en détail l'application de l'attaque par rebond sur les primitives basées sur l'AES. Nous montrons qu'il est possible de considérer un tour de plus dans la première des deux phases de cette stratégie, ce qui améliore les meilleurs résultats connus sur les permutations à base d'AES. Ceci résout le problème ouvert consistant à augmenter le nombre total de tours attaqués grâce à cette technique. Nous montrons également qu'il est possible de relâcher certaines contraintes pour augmenter la probabilité de succès de la deuxième étape. Ceci conduit à une diminution des complexités de toutes les attaques publiées. Nous appliquons ces améliorations à la fonction de hachage Grøstl, obtenant les meilleures attaques sur la permutation interne. Finalement, nous nous intéressons à la fonction de hachage ECHO pour montrer qu'il est possible d'appliquer plusieurs fois l'attaque par rebond et ainsi attaquer plus de tours de la permutation interne.

Abstract

In this thesis, we are interested in the cryptanalysis of some symmetric primitives using the structural concepts of the current encryption standard AES. We begin by an analysis of the AES itself in three different security models: the standard model, the related-key model and the open-key model. In the standard model, where the adversary tries to recover the secret key, we describe the best differential attacks, improving on the results previously published on this block cipher. Then, we conduct a structural analysis of the AES in the related-key model. We show impossibility results claiming that one cannot prove the security of the structure of the AES against differential cryptanalysis in that model. Finally, in the open-key model, we propose the first distinguisher for 9-round AES-128, which solves a long-lasting open problem in the symmetric community.

In a second part, we scrutinize the application of the rebound technique to AES-based permutations. We show that it is possible to control one more round in the first of the two parts of this strategy, which improves the best known results on this type of permutation. This result solves the open problem consisting in increasing the total number of rounds that can be attacked thanks to this technique. We also discuss the possibility to relax some constraints in the second phase to increase its probability of success. This reduces all the time complexities of the results previously published using the rebound technique. We apply these improvements to the Grøstl hash function and reach the best cryptanalysis to date on the internal permutation. Finally, we study the ECHO hash function and show how we can apply the rebound technique multiple times to attack more rounds of the internal permutation.