THÈSE DE DOCTORAT DE SORBONNE UNIVERSITÉ

Spécialité

Informatique

École Doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Augustin Bariant

Pour obtenir le grade de

DOCTEUR de SORBONNE UNIVERSITÉ

Analysis of AES-based and arithmetization-oriented symmetric cryptography primitives

soutenue publiquement le 27 juin 2024

devant le jury composé de :

Gaëtan Leurent	Inria, Paris, France	Directeur
Anne Canteaut	Inria, Paris, France	Co-directice
Pierre-Alain Fouque	Université de Rennes, Rennes, France	Rapporteur
Willi Meier	FHNW, Windisch, Suisse	Rapporteur
Martin Albrecht	King's College, Londres, Angleterre	Examinateur
Magali Bardet	Université de Rouen, Rouen, France	Examinatrice
Carlos Cid	Simula UiB, Bergen, Norvège	Examinateur
Patrick Derbez	Université de Rennes, Rennes, France	Examinateur
Reynald Lercier	DGA, Rennes, France	Invité

Remerciements

Ainsi s'achèvent trois superbes années de thèse. Superbes, car elles ont été truffées de rencontres enrichissantes, sur le plan scientifique comme humain. Superbes aussi car elles m'ont donné des amis qui le resteront après cette thèse, que nos chemins professionnels se recroisent ou non. Superbes surtout car elles m'ont donné le temps de me plonger dans le monde de la recherche et de pleinement l'apprécier - elles ont même réussi à me convaincre de ne pas le quitter totalement ! Je n'aurais cependant pas aussi bien vécu ces années sans toutes les personnes qui m'ont soutenu tout au long de cette épopée. Merci à tous d'avoir égayé cette thèse.

Merci à ceux qui ont rendu cette thèse scientifiquement possible. Merci à Willi Meier et Pierre-Alain Fouque d'avoir accepté de rapporter cette thèse. Merci aux autres membres du jury, Anne Canteaut, Carlos Cid, Gaëtan Leurent, Magali Bardet, Martin Albretch, Patrick Derbez et Reynald Lercier. Merci aux membres de mon comité de suivi de thèse, Charles Bouillaguet et Yann Rotella.

Merci plus particulièrement à Gaëtan de m'avoir introduit à la cryptographie symétrique en 2019, puis d'avoir accepté de m'accueillir en thèse. J'ai énormément apprécié ces après-midi passés dans ton bureau, devant ton tableau, à investiguer de nouvelles pistes - sans aucune garantie de résultat. J'ai toujours trouvé tes idées extrêmement claires et bien pensées. Merci également pour les mille et un[1] conseils concernant linux, latex, la programmation ou l'art de la présentation, je les garde précieusement en tête. Merci Anne d'avoir accepté d'être ma directrice de thèse, avant que Gaëtan ne reprenne le flambeau depuis quelques mois. Merci également d'avoir avancé mes frais d'inscription à Sorbonne Université, ce geste plein d'altruisme ne sera jamais oublié !

Plus généralement, c'est toute l'équipe COSMIQ qui a rendu les moments passés à l'Inria si spéciaux. Merci à tous ses membres ainsi qu'à ses satellites pendant ces trois années de thèse : Agathe, André, Anne, Anthony, Antonio, Aurélie, Aurélien, Axel, Charles, Christelle, Christina, Clara, Clémence, Daniel, Dounia, Gaëtan, Johanna, Jean-Pierre, Jules, Justine, Kévin, Léo, Loïc, Magali, Marìa, Matthieu, Margot, Maxime, Nicholas, Nicolas le grand, Nicolas l'autre grand, Pascale, Paul, Pierre, Quentin, Rachelle, Rocco, Simon, Simona, Thomas, Valentin, Virgile, et Yann. Merci Christelle, je n'étais pas un cas spécialement facile à gérer[2], mais tu as toujours été d'une précieuse aide, et ce même après mon départ de l'Inria.

[1] Je pense que je ne suis pas très loin du vrai compte.

[2] En particulier quand je partais à l'autre bout du monde.

Merci à la team mots croisés pour tous ces moments passés après le repas autour de grilles toujours plus difficiles. Grâce à vous, j'ai appris qu'une esse était à la fois un crochet et un trou[3]. Merci aussi à la team baby-foot, qui a rendu le temps passé en salle AGOS si drôle et pleine de compétition. J'en profite pour m'excuser, Jean-Pierre, pour toutes ces fois où tu es reparti de mauvaise humeur après avoir perdu contre un adversaire trop fort pour toi. Merci aussi à tous ceux qui connaissent le chemin éclairé vers l'AGRAF, et qui alimentaient la pause midi de discussions intéressantes, de débats parfois enflammés et aussi (surtout !) de drama. Merci à l'ALJCC de m'avoir initié à la course. Merci à Clara, Nicolas, Paul et Dounia d'avoir fait de notre bureau C233 un lieu de convivialité et de partage. On a même réussi l'exploit de ne lui faire perdre ni chaise ni table pendant ces trois ans[4].

Merci Clara, l'ancienne queen du bureau, aide irremplacable pour obtenir des bons plans nourriture, cinéma, opéra, parc Astérix, prime d'activité, et j'en passe. Sans toi, je pense que je n'aurais toujours pas validé mon PIF. Merci Nicolas pour tes partages de pépites à tout moment de la journée, et pour ton sourire si facile et communicatif. Il n'y a pas de mauvaises occasions pour de bonnes boutades. Clémence, merci pour toutes les initiatives que tu prenais : l'ALJCC, l'Ekiden, les différents groupes sociaux et sportifs de l'Inria. J'ai toujours été impressionné par ton sérieux et ta détermination, et j'ai hâte de te recroiser pour de futures recherches.

Merci Jules, mon fidèle partenaire de voyage, compagnon de course, camarade de travail, sans qui la thèse aurait été bien différente. Tu as toujours bien fait les choses, et tu as été un exemple à suivre sur bien des points. Et Charles, il y aurait plein de façons de te montrer à quel point je te suis reconnaissant d'avoir été toi-même, mais aucune n'égale la confiance que je t'accorde en te confiant mes clés d'appartement. Tu m'as souvent fait me sentir moins seul en prenant toute l'attention pour toi, là où en réalité tu n'étais pas toujours le seul à blâmer.

Merci Thomas de m'avoir si bien accueilli à Singapour, avec ouverture et enthousiasme, et pour l'ami que tu es rapidement devenu. Merci à tous les membres de l'équipe SyLLab que j'ai pu côtoyer : Adrien, Kai, Minghui, Mustapha, Quan Quan, Trevor et Tristan. En particulier, merci Tristan de m'avoir fait découvrir Singapour, les secrets de NTU et bien sûr les bières à la cerise devant une Tania sauvage.

Merci à tous mes coauteurs, Aurélien, Axel, Clara, Clémence, Gaëtan, Håvard, Irati, Jules, Léo, Morten, Nathan, Nicolas, Orr, Thomas et Victor. Vous m'avez tous offert une vision différente et intéressante du travail de recherche, et je vous en suis très reconnaissant.

Merci à toute ma nouvelle équipe à l'ANSSI : Ange, Henri, Hugues, Guirec, Louiza, Jean-René, Jérémy, Jérome, Julien, Maxime, Mélissa et Rachelle de m'avoir accompagné pendant toute cette fin de thèse. Je me réjouis de passer mes prochaines années avec vous.

[3] Ça sert parfois pour briller en société.

[4] En contre-partie il a perdu une serrure et un double des clés.

Merci infiniment Papa et Maman de m'avoir soutenu depuis ma plus tendre enfance et de m'avoir offert la possibilité d'arriver là où j'en suis. Je vous dois tout. Merci aussi Mamie, Cyprout, Magui, Thibaut, Loulou, Thibault, et toute la famille de m'avoir laissé parler de cryptographie pendant les repas. Je remercie tout particulièrement Stan pour la très jolie illustration qui orne la couverture de ce manuscrit.

Je ne veux pas me lancer dans le remerciement des amis hors thèse, car il y aurait trop à dire et ça ne logerait pas ici ; je vous remercie donc tous un grand coup. Je vais cependant faire une exception pour ceux avec qui j'ai partagé un toit et des souris pendant 2 ans et demi, Louise et Théo, merci du fond du cœur, vous êtes les meilleurs. Et il y a maintenant officiellement votre nom dans la thèse.

Et comment écrire ces mots sans te remercier Gilles, toi qui es finalement à l'origine du commencement de cette thèse, en m'ayant mis en contact avec Anne en 2018 pour amorcer un stage de recherche, après nous avoir vus tous les deux à quelques semaines d'intervalle. Maintenant que j'y repense, je me dis que le hasard fait plutôt bien les choses.

Résumé

La cryptographie joue un rôle clé dans la communication numérique, en assurant que des utilisateurs malveillants ne soient pas en mesure d'obtenir des informations sensibles qui ne leur appartiennent pas. En cryptographie symétrique, deux utilisateurs s'accordent sur une clé secrète partagée, et sur un algorithme de chiffrement pour chiffrer leurs communications, le plus utilisé étant l'AES. Cependant, la sécurité de tels chiffrements symétriques n'est pas prouvable mathématiquement, ce qui implique qu'un grand effort doit être alloué à la cryptanalyse, i.e. la recherche des meilleures attaques.

Dans ce contexte, cette thèse améliore des techniques de cryptanalyse contre des chiffrements basés sur l'AES. Premièrement, nous présentons une attaque sur une version complète de ForkAES et une attaque différentielle impossible améliorée sur ForkSkinny. Deuxièmement, nous montrons de nouvelles attaques boomerang sur 6 tours d'AES et sur plusieurs chiffrements basés sur l'AES. En particulier, nous introduisons un nouveau framework d'attaque boomerang, l'attaque boomerang tronquée, qui produit les meilleures attaques connues contre Kiasu-BC, Deoxys-BC et TNT-AES.

Nous présentons ensuite un framework de fonctions de hachage universelles basées sur l'AES, duquel nous dérivons deux MACs basés sur l'AES, LeMac et PetitMac. Les performances de LeMac sont les meilleures de la littérature sur les processeurs récents.

Enfin, nous étudions les attaques algébriques contre une nouvelle génération de primitives symétriques, dites Orientées-Arithmétisation (OA). Nous montrons que ces attaques peuvent être améliorées avec des techniques de cryptanalyse symétrique, et soulignons que les attaques univariées sont bien moins coûteuses que les attaques multivariées. Dans ce cadre, nous présentons l'attaque FreeLunch, un nouveau type d'attaque algébrique qui remet en question la sécurité de plusieurs primitives OA récentes.

Abstract

Cryptography plays a critical role in digital communication, by ensuring that malicious users cannot obtain sensitive information that do not belong to them. In symmetric cryptography, two users agree on a secret key, and use a cipher to encrypt their communication, the most used of which being AES. However, the security of symmetric ciphers is not mathematically provable, therefore a lot of effort needs to be dedicated to cryptanalysis, i.e. the search for the best attacks.

In this context, this thesis improves on some cryptanalysis techniques against AES-based ciphers. First, we present an attack on full ForkAES, together with an improved impossible differential attack on ForkSkinny. Second, we show some new boomerang attacks on 6-round AES and on several AES-based ciphers. In particular, we introduce a new boomerang attack framework, the truncated boomerang attack, that yields the best known attacks against Kiasu-BC, Deoxys-BC and TNT-AES.

Then, we present an AES-based universal hash function framework, from which we design two AES-based MACs, LeMac and PetitMac. LeMac has the best performance among existing MAC algorithms on recent desktop CPUs.

We finally study algebraic attacks against a new generation of symmetric primitives, called Arithmetization-Oriented (AO). We show that these attacks can be improved with symmetric techniques, and highlight that univariate attacks are much cheaper than multivariate attacks. We also present the FreeLunch attack, a new type of algebraic attack that challenges the security of several recent AO primitives.

Résumé des travaux

La cryptographie est l'étude des techniques de chiffrement des communications, afin que des utilisateurs malicieux soient dans l'incapacité d'obtenir une information qui ne leur est pas destinée. Les deux entités souhaitant communiquer sont communément dénommées Alice et Bob, et l'attaquant, qui a accès aux communications chiffrées entre Alice et Bob, s'appelle Eve (voir Figure 1).

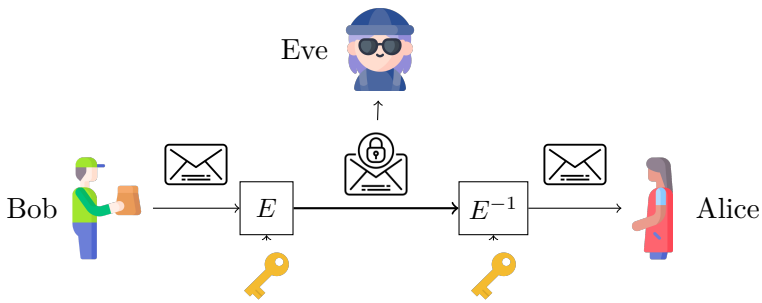


Figure 1: Illustration du modèle de sécurité en cryptographie symétrique.

Chapitre 1 : Introduction à la cryptographie

Ce premier chapitre est destiné à introduire le lecteur à l'histoire de la cryptographie et aux enjeux actuels portés par ce domaine. La cryptographie doit être différenciée de la stéganographie, cette dernière étant l'étude de la dissimulation d'un message caché sous différentes formes, dans un message intelligible par exemple. La cryptographie, de son côté, assure un des principes fondamentaux d'Auguste Kerckhoffs énoncé dans son ouvrage *La Cryptographie Militaire* [Ker83]: « Il faut que [le moyen de chiffrement] n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi ». L'algorithme cryptographique doit être considéré comme public et connu de tous, et la sécurité du chiffrement doit reposer sur un secret court, que l'on appelle une *clé cryptographique*.

Historiquement, la cryptographie s'est avérée d'une importance cruciale pendant la deuxième guerre mondiale, où elle a été une des principales raisons de la victoire des Alliés, grâce au cassage par les Polonais et les Britanniques de la machine Enigma, utilisée par les nazis.

La deuxième partie du XIX-ème siècle s'illustre par l'introduction de cryptosystèmes asymétriques : le protocole d'échange de clé de Diffie-Hellman [DH76] et le système de chiffrement à clé publique RSA [RSA83]. Dans RSA, chaque utilisateur génère à une clé publique, connue de tous, dont la clé privée n'est connue que de l'utilisateur en question. La clé privée doit être dure à calculer à partir de la clé publique. Cette difficulté repose sur des problèmes mathématiques difficiles à résoudre, telle que la factorisation de grands entiers, ou le calcul du logarithme discret.

Chapitre 2 : La cryptographie symétrique

En cryptographie symétrique, les deux utilisateurs, Alice et Bob, possèdent une information secrète commune, une clé de chiffrement. La sécurité d'un chiffrement symétrique repose sur le fait que Eve, ne connaissant pas la clé, ne peut déduire aucune information sur les messages transmis entre Alice et Bob. En particulier, Eve ne doit pas être en capacité de recouvrer la clé à partir des communications écoutées entre Alice et Bob. Nous représenterons Eve par un attaquant qui fait des requêtes à un oracle de chiffrement, lui donnant des chiffrés à partir d'un texte clair qu'elle choisit. C'est le modèle à texte clair choisi, qui existe sous différentes variantes. Une attaque par recouvrement de clé est ainsi un algorithme pouvant utiliser cet oracle, qui recouvre la clé plus rapidement que la recherche exhaustive sur l'ensemble de clés. D'autres types d'attaques existent, tels que les distingueurs. L'étude de l'ensemble des attaques contre un chiffrement s'appelle la cryptanalyse, et certaines techniques générales s'appliquent à différents algorithmes de chiffrement. Une des techniques les plus connues est la cryptanalyse différentielle, qui traque une différence injectée dans les textes clairs et estime la probabilité que les chiffrés correspondants possèdent une différence fixée.

Les algorithmes de chiffrement symétriques sont généralement construits à partir de briques élémentaires, les primitives cryptographiques, qui sont plus petites et plus faciles à analyser. Une des primitives cryptographiques les plus étudiées est le chiffrement par bloc. Au lieu de chiffrer un message de taille arbitraire, le chiffrement par bloc chiffre un bloc de message, de taille typique $n \in \{64, 128\}$ bits. Le chiffrement par bloc a la signature suivante:

$$\begin{array}{c} E : \{0,1\}^k \times \{0,1\}^n \mapsto \{0,1\}^n \\ K \quad , \quad P \quad \mapsto \quad C. \end{array}$$

La première entrée est une clé K, la deuxième entrée un texte clair P, et la sortie le chiffré C. La fonction à clé fixée qui envoie le texte clair sur le chiffré est une permutation, ce qui permet de déchiffrer C en utilisant la permutation inverse, sous connaissance de la clé. Le chiffrement par bloc le plus utilisé de nos jours est l'AES [DR02], standardisé par l'Institut National (Américain) des Standards et de la Technologie (NIST) en 2001. Une variante du chiffrement par bloc est le chiffrement par bloc avec tweak, qui prend une entrée additionnelle, le tweak T.

Chapitre 3 : Attaques contre les Forkciphers

Les Forkciphers [ALP+19b] sont des primitives avec la signature:

$$\begin{array}{ccccccc} E : \{0,1\}^k & \times & \{0,1\}^n & \mapsto & \{0,1\}^n & \times & \{0,1\}^n \\ K & , & P & \mapsto & C_1 & , & C_2. \end{array}$$

Ces primitives sont telles qu'à clé K fixé, $E_K^1 : P \mapsto C_1$ et $E_K^2 : P \mapsto C_2$ sont toutes deux des permutations, et dans les instances proposées de Forkciphers, E_K^1 correspond exactement à un chiffrement par bloc de la littérature. Elles peuvent servir à construire un chiffrement authentifié avec données additionnelles (AEAD), destiné pour des messages extrêmement courts (typiquement moins de 128 bits), et introduits pour la compétition des chiffrements à bas coût du NIST. Dans ce chapitre, nous présentons des attaques améliorées présentées dans [BDL20] sur deux instances des Forkciphers: ForkAES [ARV+18] basé sur l'AES (plus précisément dans sa variante tweakée Kiasu-BC [JNP14a]), et ForkSkinny, basé sur Skinny [BJK+16].

L'attaque sur ForkAES exploite le manque de diffusion dans les tours du milieu des requêtes de reconstruction ($C_1 \mapsto C_2$), propres aux Forkciphers. Cela mène à une attaque sur les 10 tours de ForkAES, comme illustré par la Table 1.

Algorithme	Attaque	Trs.	D.	Tps.	Mem.	ϵ	Réference
AES-128	MitM	7	2^{97}	2^{99}	2^{98}	1	[DFJ13]
Kiasu-BC	Boomerang	8	2^{103}	2^{103}	2^{60}	1	[DL17]
Kiasu-BC	Boomerang	8	2^{83}	2^{83}	2^{80}	1	[BL23b]
ForkAES-$*$-4-4	Diff. Imp.	8	$2^{39.5}$	2^{47}	2^{35}	1	[BBJ+19]
ForkAES-$*$-4-4	Diff. Refl.	8	2^{35}	2^{35}	2^{33}	1	[BBJ+19]
ForkAES-$*$-5-5	Diff. Tronq.	10	$2^{74.5}$	$2^{74.5}$	$2^{59.5}$	2^{-32}	Cette thèse
ForkAES-$*$-5-5	Diff. Tronq.	10	$2^{100.5}$	2^{114}	$2^{80.5}$	2^{-4}	Cette thèse
ForkAES-$*$-5-5	Diff. Tronq.	10	2^{119}	2^{125}	2^{83}	1	Cette thèse

Table 1: Résultats de cryptanalyse contre AES, Kiasu-BC, et ForkAES. ϵ est la fraction des clés faibles attaquée.

Nos attaques sur ForkSkinny exploitent un cycle dans les Linear Feedback Shift Registers (LFSR) utilisés dans le cadencement de clé et de tweak, et nous permettent d'augmenter la longueur des caractéristiques de différentielles impossibles. Les résultats de nos attaques sont présentés dans la Table 2.

Nos attaques montrent que les Forkciphers n'héritent pas directement de la sécurité du chiffrement par bloc sous-jacent, contrairement à l'idée des concepteurs, et qu'une analyse dédiée est nécessaire, pour chaque instance de Forkciphers.

Algorithme	Mod.	κ	Attaque	Trs.	D.	Tps.	Mem.	Ref.
ForkSkinny-128-256	RTK2	128	DI	24	$2^{126.5}$	$2^{126.5}$	$2^{101.5}$	Cette thèse
ForkSkinny-128-256	RTK2	128	Rect.	25	$2^{118.9}$	$2^{118.9}$	$2^{119.2}$	[QDW+21]
Skinny-128-∗	RTK2	256	DI	23	$2^{124.5}$	$2^{243.5}$	$2^{163.4}$	[SMB18]
Skinny-128-∗	RTK2	256	Rect.	26	$2^{126.5}$	$2^{241.4}$	2^{136}	[DQS+22]
ForkSkinny-128-256	RTK2	256	DI	26	2^{125}	$2^{254.6}$	2^{160}	Cette thèse
ForkSkinny-128-256	RTK2	256	DI	26	2^{127}	$2^{250.3}$	2^{160}	Cette thèse
ForkSkinny-128-256	RTK2	256	Rect.	28	$2^{118.9}$	$2^{224.8}$	$2^{118.9}$	[DQS+22]

Table 2: Résultats de cryptanalyse sur Skinny et ForkSkinny. La colonne κ indique la taille de clé.

	Trs.	Type	Donnée		Tps.	Ref
Kiasu-BC	8	Boomerang (RC)	2^{103}	ACC	$2^{103.1}$	[DL17]
	8	Boomerang Tronqué (RC)	2^{83}	ACC	2^{83}	Cette thèse
TNT-AES	∗-5-∗	Boomerang (dist.)	2^{126}	ACC	2^{126}	[BGG+20]
	5-∗-∗	Diff Imp. (RC)	$2^{113.6}$	CP	$2^{113.6}$	[GGL+20]
	∗-∗-∗	Generic (dist.)	$2^{99.5}$	CP	$2^{99.5}$	[GGL+20]
	∗-5-∗	Boomerang Tronqué (dist.)	2^{76}	ACC	2^{76}	Cette thèse
	5-5-∗	Boomerang Tronqué (RC)	2^{87}	ACC	2^{87}	Cette thèse
	∗-6-∗	Boomerang Tronqué (dist.)	$2^{127.8}$	ACC	$2^{127.8}$	Cette thèse

Table 3: Attaques en recouvrement de clé (RC) et distingueurs (dist.) contre Kiasu-BC and TNT-AES.

Chapitre 4 : Attaques boomerang sur AES et ses variantes

Dans ce chapitre, nous présentons de nouvelles attaques boomerangs sur l'AES et sur des chiffrements basés sur l'AES, dont une partie a été présentée à EUROCRYPT 2023 [BL23b]. L'attaque boomerang est une attaque introduite par Wagner en 1999 [Wag99], qui utilise des messages clairs et chiffrés choisis adaptativement (en fonction des réponses précédentes de l'oracle).

Dans un premier temps, nous proposons une nouvelle variante de l'attaque boomerang, l'attaque boomerang tronquée, en prenant en compte les structures de textes clairs et de chiffrés et en analysant l'attaque boomerang comme une attaque complète sur tous les tours du chiffrement, au lieu de l'analyse traditionnelle partant d'une caractéristique sur un nombre de tours réduits et ajoutant des tours de recouvrement de clé. L'attaque boomerang tronquée permet d'améliorer les attaques boomerang sur 6 tours d'AES, et d'obtenir les meilleures attaques contre Kiasu-BC [JNP14a], TNT-AES [BGG+20] et Deoxys-BC [JNP+21].

	Attaque		Donnée	Tps.	Mem.	Ref.
Distingueur	Attaque Échange	ACC	2^{84}	2^{83}	2^{32}	[Bar19]
	Boomerang Tronqué	ACC	2^{87}	2^{87}	2^{66}	Cette thèse
Recouvr. de clé	Square	CP	2^{33}	2^{40}	2^{32}	[DGK+24]
	Boomerang Retraçant	ACC	2^{55}	2^{80}	2^{31}	[DKR+20]
	Boomerang Tronqué	ACC	2^{59}	2^{61}	2^{59}	Cette thèse
	Boomerang	ACC	2^{51}	2^{68}	2^{32}	Cette thèse
	Boomerang	ACC	2^{51}	2^{66}	2^{42}	Cette thèse
	Boomerang	ACC	2^{57}	2^{61}	2^{33}	Cette thèse
S-boîte secrète	Square	CP	2^{64}	2^{90}	2^{69}	[TKK+15]
	Boomerang Tronqué	ACC	2^{94}	2^{94}	2^{56}	Cette thèse

Table 4: Attaques contre 6 tours d'AES dans différents scénarios.
CP: Clairs choisis / ACC: Clairs et chiffrés choisis adaptativement.

Dans un deuxième temps, nous présentons deux variantes de l'attaque boomerang sur 6 tours d'AES, améliorant les précédentes attaques boomerangs. Ce ne sont pas les attaques les plus compétitives sur 6 tours d'AES car elles sont moins performantes que l'attaque Square [DKR97] et ses améliorations [FKL+01; TA14], mais elles sont plus avantageuses que les autres classes d'attaque sur 6 tours d'AES. Les résultats de ce chapitre sont présentés dans les Tables 3, 4 et 5.

		Ancien					Cette thèse		
Mod.	Trs.		Donnée	Tps.	Mem.	Ref.		Donnée	Tps.	Mem.
RTK1	8						B	2^{88}	2^{88}	2^{73}
	9						B	2^{135}	2^{174}	2^{129}
RTK2	8	B	2^{28}	2^{28}	2^{27}	[Sas18a]	B	2^{27}	2^{27}	2^{27}
	9	B	2^{98}	2^{112}	2^{17}	[Sas18a]	B	$2^{55.2}$	$2^{55.2}$	$2^{55.2}$
	10	B	$2^{98.4}$	$2^{109.1}$	2^{88}	[ZDJ19]	B	$2^{94.2}$	$2^{95.2}$	$2^{94.2}$
	11	R	$2^{122.1}$	$2^{249.9}$	$2^{128.2}$	[ZDJ19]	B	2^{129}	$2^{223.9}$	2^{129}
RTK3	10	B	2^{22}	2^{22}	2^{17}	[Sas18a]	B	$2^{19.4}$	$2^{19.4}$	2^{18}
	11	B	2^{100}	2^{100}	2^{17}	[Sas18a]	B	$2^{32.7}$	$2^{32.7}$	$2^{32.7}$
	12	B	2^{98}	2^{98}	2^{64}	[ZDJ19]	B	$2^{67.4}$	$2^{67.4}$	2^{65}
	13	R	$2^{125.2}$	$2^{186.7}$	2^{136}	[ZDJ+19]	B	$2^{126.7}$	$2^{170.2}$	$2^{126.7}$
	14	R	$2^{125.2}$	$2^{282.7}$	2^{136}	[ZDJ+19]	B	2^{129}	$2^{278.8}$	2^{129}

Table 5: Attaques boomerang (B) et rectangle (R) contre des variantes de Deoxys-BC. La plupart des attaques ont une probabilité de succès de 1/2. Dans le modèle RTKi, l'attaquant contrôle i blocs de tweakey de 128 bits.

Chapitre 5 : Conception de fonctions de hachage universelles et de MACs basés sur AES

Dans ce chapitre, nous concevons un framework de fonctions de hachage universelles basées sur l'AES, vouées à atteindre une vitesse ultrarapide en implémentation logicielle. Nous proposons ensuite deux constructions de codes d'authentification de messages (MAC)s à partir de fonctions de hachage universelles resepectant certaines propriétés de sécurité. Pour cela, nous exploitons le jeu d'instructions AES-NI [Gue08], introduit par Intel en 2008, qui offre une accélération matérielle à la fonction de tour de l'AES et qui augmente sa vitesse d'un ordre de grandeur. La fonction de tour de l'AES possède donc des propriétés cryptographiques remarquables comparées à celles des instructions processeur ayant une vitesse d'exécution similaire. Pour cette raison, la conception d'algorithmes cryptographiques basés sur l'AES est très populaire, comme l'illustrent AEGIS-128 [WP14] et Deoxys-BC [JNP+21], sélectionnés dans le portefeuille CAESAR.

De façon surprenante, la conception de fonctions de hachage universelles et de MACs basés sur l'AES n'a pas été directement considérée dans la littérature. Certains chiffrements authentifiés avec données additionnelles peuvent être convertis en MACs, mais ils ne sont pas optimisés pour cet usage. C'est dans ce contexte que nous avons conçu les premiers MACs basés sur l'AES, LeMac et PetitMac. Les performances de LeMac sont comparées à celles de la littérature dans la Table 6.

Pour la conception de ces deux MACs, nous nous sommes basés sur la construction EWCDM [CS16], pour obtenir 128 bits de sécurité contre les attaques par forge. Cette construction requiert une primitive nommée fonction de hachage universelle, qui possède des propriétés statistiques très simplement formulables. Ces propriétés sont facilement dérivables à partir de la connaissance de la meilleure différentielle existant sur la primitive. En partant d'un candidat de notre framework, nous avons mis au point un outil automatique basé sur de la pogrammation linéaire en nombres entiers (MILP) pour borner la probabilité du meilleur chemin différentiel, qui permet donc d'estimer la probabilité de la meilleure différentielle. En outre, nous avons implémenté un outil automatique qui génère une implémentation de candidats de notre framework en C, la compile et mesure sa performance. Cela nous permet donc, automatiquement, de générer des candidats et de vérifier leur sécurité et leur rapidité, afin de conserver les candidats sécurisés les plus rapides. Cette stratégie nous a permis d'avoir les meilleures performances de la littérature pour un MAC sécurisé (LeMac).

Chapitre 6 : Cryptanalyse algébrique de primitives orientées arithmétisation

Dans ce chapitre, nous introduisons le lecteur aux primitives orientées arithmétisation (OA) et à la cryptanalyse algébrique, et présentons des attaques algébriques

CPU	Cipher	Speed (c/B) 1kB	16kB	256kB
Intel Ice Lake (Xeon Gold 5320)	GCM (AD only)	0.699	0.311	0.286
	Rocca (AD only)	0.528	0.171	0.149
	Rocca-S (AD only)	0.478	0.172	0.151
	AEGIS128L (AD only)	0.416	0.208	0.195
	Tiaoxin-346 v2 (AD only)	0.328	0.131	0.121
	Jean-Nikolić	0.307	0.126	0.113
	LeMac	0.289	0.082	0.068
AMD Zen3 (EPYC 7513)	GCM (AD only)	0.794	0.470	0.451
	Rocca (AD only)	0.393	0.139	0.124
	Rocca-S (AD only)	0.417	0.157	0.141
	AEGIS128L (AD only)	0.358	0.183	0.173
	Tiaoxin-346 v2 (AD only)	0.311	0.121	0.109
	Jean-Nikolić	0.312	0.111	0.098
	LeMac	0.309	0.085	0.072

Table 6: Résultats de benchmark de MACs pour différentes tailles de message.

sur des primitives OA, qui ont été publiées à ToSC 2022 [BBL+22] et soumises pour publication à SAC 2024 [Bar23].

Depuis plus d'une dizaine d'années, une attention particulière en cryptographie a été portée sur les protocoles avancés pour le chiffrement entièrement homomorphe (FHE), le calcul multipartite (MPC), ou les preuves à divulgation nulle de connaissance (ZK). Ces protocoles ont un point commun majeur: ils opèrent sur des grands corps finis[5] $\mathbb{F}_q$, avec q typiquement plus grand que 2^{63}, et le coût est très fortement lié au nombre de multiplications dans le corps en question. Dans ce cadre, l'utilisation d'algorithmes de chiffrement symétrique nécessite leur conversion en séquences d'opérations dans $\mathbb{F}_q$, et cela engendre souvent un surcoût important.

LowMC [ARS+15] a été une première tentative de minimiser le nombre de multiplications binaires dans un chiffrement classique, afin de réduire le nombre de multiplications dans $\mathbb{F}_p$. Mais une nouvelle approche, dont MiMC [AGR+16] a été le précurseur, offre une accélération particulièrement importante comparé à LowMC : au lieu d'opérer sur des bits, un chiffrement peut directement opérer sur des éléments de $\mathbb{F}_q$, de sorte à entièrement rentabiliser chaque multiplication dans $\mathbb{F}_q$. Un tel chiffrement est dit orienté arithmétisation (OA). La liste des chiffrements orientés arithmétisation est longue, mais ils ont tous moins de 10 ans d'ancienneté : nous pouvons citer Jarvis [AD18], Vision, Rescue [AAB+20], Poseidon [GKR+21], Ciminion [DGG+21b], `Anemoi` [BBC+23], Griffin [GHR+23], Hydra [GØS+23], XHash8, XHash12 [AKM23] et Arion [RST23].

[5]Certains protocoles FHE sont construits sur des anneaux, mais nous ne considérons que les corps dans cette thèse.

Algorithme	Paramètres	Securité estimée par la FE ($\log_2$)	Résolu	Cxté ($\log_2$) de notre attaque
Rescue–Prime [AAB+20]	$N = 4, m = 3$	38	✓	43
	$N = 6, m = 2$	38		53
	$N = 7, m = 2$	44		62
	$N = 5, m = 3$	45		57
	$N = 8, m = 2$	50		72
Feistel–MiMC [AGR+16] (challenges originaux)	$r = 6$	18	✓	19
	$r = 10$	30	✓	26
	$r = 14$	44	✓	33
	$r = 18$	56	✓	40
	$r = 22$	68		47
Poseidon [GKR+21]	$R_P = 3$	45	✓	26
	$R_P = 8$	53	✓	35
	$R_P = 13$	61	✓	44
	$R_P = 19$	69		54
	$R_P = 24$	77		62

Table 7: Challenges proposés par la Fondation Ethereum (FE).

L'objectif de primitives OA étant de minimiser le nombre de multiplications dans $\mathbb{F}_q$ afin d'offrir les meilleures performances, les attaques algébriques deviennent particulièrement avantageuses comparées aux attaques statistiques classiques telles que les attaques linéaires et différentielles. L'impact de ces attaques algébriques doit être bien compris afin de correctement calibrer les nouveaux algorithmes OA. C'est dans cette perspective que la Fondation Ethereum (FE) a lancé une série de challenges, en 2021, rémunérant les attaques à entrée et sorties contraintes (CICO) contre des permutations OA. Nous listons certains challenges dans la Table 7, en indiquant ceux que nous avons résolus.

Il existe un certain nombre d'attaques algébriques : les attaques par interpolation [JK97], les attaques Cube [DS09], l'attaque PGCD [AGR+16], et les attaques par résolution de système (univarié ou multivarié). Ce sont ces dernières qui nous intéressent particulièrement, et que nous exploitons dans ce chapitre et dans le suivant. Dans cette attaque, l'attaquant modélise la primitive avec des équations, i.e. il écrit les équations liant l'entrée et la sortie (et potentiellement la clé), en ajoutant des variables intermédiaires si nécessaires. Si le système polynomial obtenu est univarié, il existe un algorithme relativement efficace pour trouver ses solutions. S'il est multivarié, l'attaquant le résout en calculant une base de Gröbner [Buc76] de ce système.

Nous combinons ces techniques algébriques avec des techniques de cryptanalyse symétrique pour gagner plusieurs tours de plusieurs permutations OA et obtenir des attaques améliorées sur Rescue–Prime [AAB+20], Feistel–MiMC [AGR+16] et Poseidon [GKR+21]. Ces résultats montrent que les estimations de sécurité faites par la fondation Ethereum étaient erronées. De plus, nous montrons deux modélisations astucieuses de Ciminion qui permettent d'obtenir des attaques contredisant les revendications de sécurité de ses concepteurs.

Chapitre 7 : L'attaque FreeLunch sur des primitives orientées arithmétisation

Ce dernier chapitre est dédié à la présentation d'une nouvelle classe d'attaques algébriques sur des primitives OA, accepté à CRYPTO 2024 et disponible sur e-print [BBL+24b]. Dans cette attaque, nous choisissons un ordre monomial pondéré, de telle sorte qu'une base de Gröbner est automatiquement obtenue dès la modélisation de la primitive (étape nommée `sysGen`). En outre, nous améliorons une partie (nommée `polyDet`.) de la deuxième phase de l'attaque par base de Gröbner, traditionnellement appelée FGLM [FGL+93]. Néanmoins, cette deuxième phase comporte toujours une étape dont l'estimation de la complexité semble non-triviale, que nous appelons `matGen`. Nous avons seulement une borne lâche de la complexité de cette étape.

Nous appliquons l'attaque FreeLunch au problème CICO sur trois permutations OA, `Griffin` [GHR+23], `Arion` [RST23] and `Anemoi` [BBC+23], en utilisant des techniques de cryptanalyse symétrique pour réduire la complexité de l'attaque, et montrons que la sécurité de `Griffin` et de α-`Arion` est compromise malgré la borne lâche pour `matGen`. Cela semble aussi montrer que dans le cas où la borne pour la complexité de `matGen` venait à s'améliorer, `Anemoi` pourrait aussi voir sa sécurité compromise, car la complexité de `polyDet` est trop faible pour la sécurité revendiquée.

Nous présentons nos estimations de complexité de l'étape `polyDet` pour ces trois permutations OA dans les Tables 9, 8, 10 (nombre de tours entre parenthèses) : la complexité de cette étape ne suffit pas pour garantir la sécurité de la plupart des variantes des trois permutations.

De plus, nous avons implémenté cette attaque sur des versions réduites de ces trois permutations, et avons obtenu les temps d'attaques décrits dans les Tables 11 et 12. En particulier, nous résolvons en pratique le problème CICO sur 7 des 10 tours de `Griffin`.

	`Arion`-π & `ArionHash`		α-`Arion`& α-`ArionHash`	
Branches	Complexité ($\log_2$)		Complexité ($\log_2$)	
	$e = 3$	$e = 5$	$e = 3$	$e = 5$
3	128 (6)	132 (6)	104 (5)	83 (4)
4	134 (6)	113 (5)	84 (4)	87 (4)
5	114 (5)	118 (5)	88 (4)	91 (4)
6	119 (5)	122 (5)	92 (4)	94 (4)
8	98 (4)	101 (4)	98 (4)	101 (4)

Table 8: Complexité estimée en temps de `polyDet` pour les différentes instances complètes d'`ArionHash`, où $\alpha = 121$ et $\omega = 2.81$.

Branches	Complexité ($\log_2$)	
	$\alpha = 3$	$\alpha = 5$
3	120 (16)	141 (14)
4	112 (15)	110 (11)
8	76 (11)	81 (9)
12,16,20,24	64 (10)	74 (9)

Table 9: Complexité estimée en temps de `polyDet` pour les différentes instances complètes de `Griffin`, où l'exposant de multiplication matricielle $\omega = 2.81$.

Sécurité attendue	$\alpha = 3$	$\alpha = 5$	$\alpha = 7$	$\alpha = 11$
128	118 (21)	156 (21)	174 (20)	198 (19)
256	203 (37)	270 (37)	307 (36)	358 (35)

Table 10: Complexité estimée en temps de `polyDet` pour les différentes instances complètes d'`Anemoi` sur $\mathbb{F}_p$ avec $\ell = 1$.

Nombre de tours	Complexité de `polyDet`	Temps (s)			Mémoire (MB)
		`sysGen`	`matGen`	`polyDet`	
5	26	0.17	0.02	0.53	14
6	34	4.0	6.67	50.78	471
7	41	2,558	3,361	5,727	27,600

Table 11: Résultats experimentaux sur `Griffin` avec $(t, \alpha) = (12, 3)$.

Nombre de tours	Complexité de `polyDet`	Temps (s)			Mémoire (MB)
		`sysGen`	`matGen`	`polyDet`	
3	20	< 0.01	< 0.01	0.02	< 400
4	26	< 0.01	0.34	0.24	< 400
5	32	0.07	23.3	7.6	< 400
6	37	2.52	2,127	292	2,863
7	43	128	156,348	10,725	42,337

Table 12: Résultats experimentaux sur `Anemoi` avec $(\ell, \alpha) = (1, 3)$.

Nombre de branches	Complexité de `polyDet`	Temps (s)			Mémoire (MB)
		`sysGen`	`matGen`	`polyDet`	
3	32	1.31	< 0.01	6.8	3,387
4	33	1.46	0.07	18.7	7,551
5	35	9.54	0.08	64.5	15,903
6	36	247	0.31	215	32,626
8	39	24,872	4.86	2,545	134,165

Table 13: Résultats experimentaux sur 2 tours d'`Arion`, avec $(e, \alpha) = (3, 121)$.

Contents

List of Figures

List of Tables

Publications And Preprints

[Bar23] Augustin Bariant. *Algebraic Cryptanalysis of Full Ciminion*. Cryptology ePrint Archive, Paper 2023/1283. https://eprint.iacr.org/2023/1283. 2023 (cit. on pp. xv, 199, 226).

[BBL+22] Augustin Bariant, Clémence Bouvier, Gaëtan Leurent, and Léo Perrin. "Algebraic Attacks against Some Arithmetization-Oriented Primitives". In: *IACR Transactions on Symmetric Cryptology* 2022.3 (2022), pp. 73–101. DOI: 10.46586/tosc.v2022.i3.73-101 (cit. on pp. xv, 199, 213, 214, 226, 235).

[BBL+24a] Augustin Bariant, Jules Baudrin, Gaëtan Leurent, Clara Pernot, Léo Perrin, and Thomas Peyrin. *Fast AES-based Universal Hash Functions and MACs*. Accepted at ToSC 2024.2. 2024.

[BBL+24b] Augustin Bariant, Aurélien Boeuf, Axel Lemoine, Irati Manterola Ayala, Morten Øygarden, Léo Perrin, and Håvard Raddum. *The Algebraic Freelunch Efficient Gröbner Basis Attacks Against Arithmetization-Oriented Primitives*. Cryptology ePrint Archive, Paper 2024/347. https://eprint.iacr.org/2024/347. 2024 (cit. on pp. xvii, 239).

[BDK+24] Augustin Bariant, Orr Dunkelman, Nathan Keller, Gaëtan Leurent, and Victor Mollimard. *Improved Boomerang Attacks on 6-Round AES*. In submission. 2024.

[BDL20] Augustin Bariant, Nicolas David, and Gaëtan Leurent. "Cryptanalysis of Forkciphers". In: *IACR Transactions on Symmetric Cryptology* 2020.1 (2020), pp. 233–265. ISSN: 2519-173X. DOI: 10.13154/tosc.v2020.i1.233-265 (cit. on pp. xi, 47, 66).

[BL23b] Augustin Bariant and Gaëtan Leurent. "Truncated Boomerang Attacks and Application to AES-Based Ciphers". In: *Advances in Cryptology – EUROCRYPT 2023, Part IV*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14007. Lecture Notes in Computer Science. Lyon, France: Springer, Heidelberg, Germany, 2023, pp. 3–35. DOI: 10.1007/978-3-031-30634-1_1 (cit. on pp. xi, xii, 49, 51, 83, 89, 109, 110, 116).

Notation

$\mathbb{Z}$	The set of integers
$\mathbb{N}$	The set of positive integers
$\mathbb{F}_q$	The finite field with q elements
$\lvert S \rvert$	The cardinality of a set S
$\mathrm{Supp}(V)$	The set of indices where the vector V is non-zero
$[\![i, j]\!]$	The set $\{i, i+1, \ldots, j-1, j\}$ if $i \leqslant j$ are integers
$a + b$	Bitwise addition (XOR) if a and b are bit vectors
$a \parallel b$	The concatenation of a and b
$\langle a, b \rangle$	Given $a, b \in \mathbb{F}_2^n$, the scalar product of a and b
$\mathbb{1}_F$	Equals 1 if the predicate F is true, and 0 otherwise
$\mathbb{F}[x]$	The set of univariate polynomials defined over the field $\mathbb{F}$
$\mathbb{F}[x_0 \ldots x_{n-1}]$	The set of multivariate polynomials defined over the field $\mathbb{F}$
$P \mid Q$	Polynomial P divides polynomial Q
S_n	The symmetric group of size n
$\mathcal{M}_{n \times m}(\mathbb{F})$	The set of matrices of size $n \times m$ over the field $\mathbb{F}$
$\mathbf{GL}_n(\mathbb{F})$	The set of invertible matrices among $\mathcal{M}_{n \times n}(\mathbb{F})$
$\mathbf{Diag}(A_0 \cdots A_\ell)$	A diagonal block matrix composed of matrices $A_0, \cdots A_\ell$
$\mathbf{Poisson}(\lambda)$	The Poisson law with parameter λ

Contents

Part I
Preliminaries

Chapter 1
Introduction to Cryptography

In this chapter, we introduce the reader to the world of cryptography. This chapter remains high-level, and explains some challenges related to cryptography without technical details.

Contents

1.1 A Short History

The term cryptography originates from the Greek words *kryptós* (meaning "secret") and *graphein* (meaning "to write"), and refers to the study of encryption techniques that provide secure communication in the presence of adversarial behavior. It is to be differentiated with the word steganography, from the Greek *steganos* ("conceal"). Steganography, instead of encrypting the communication, involves techniques to conceal a hidden message, physically or within a dummy message. Whereas an adversarial party knowing the employed steganography technique can recover the hidden message, cryptography techniques ensure that an adversarial party can not recover the secret information, even if he knows the encryption technique.

In its simplest form, an encryption technique transforms a message, called plaintext, into an encrypted version of it, called ciphertext, which is then sent to the correspondent. Though a malicious eavesdropper can perceive that the ciphertext is unintelligible, and therefore that an encryption mechanism is used, he is not capable to decrypt the ciphertext. The ciphertext can however be decrypted by the legitimate correspondant using a secret information shared with the sender: a secret key.

The earliest records of cryptographic applications point to Mesopotamia around 1500 BC, when a craftsman wrote an encoded recipe on a clay tablet to protect the secrecy of the recipe. The cipher used was a monoalphabetic substitution cipher: each letter is replaced deterministically with another letter. Other use cases of cryptography were documented in Ancient Greece, in India or in the Roman Empire during the Antiquity. In the latter, Julius Caesar gave its name to one of

the most famous encryption process: the Caesar cipher. The generalized Caesar cipher consists in shifting each letter by a fixed position in the alphabet. Caesar used a shift of 3 letters (A is encrypted to D, B is encrypted to E...) in his private correspondences.

In the 8th century, the Arabs made drastic advancements in cryptography. An Arab mathematician, Al-Kindi, invented frequency analysis to break all monoalphabetic substitution ciphers. He started with the observation that certain letters in the Arabic alphabet occurred more often than others. For a sufficiently long ciphertext, the most frequent letter of the ciphertext is likely to be decrypted to the most frequent letter of the alphabet. This is the first known cryptanalysis technique.

Most ciphers could not resist frequency analysis, until the polyalphabetic ciphers introduced in the 15th century by Alberti, an Italian polymath. The work of Alberti was followed by a new design of one of his compatriots, Bellaso, which remained unbroken until the second half of the 19th century. This cipher acquired the name of Vigenere cipher, giving the credit to the French Blaise de Vigenere, although he only invented a variant of the cipher decades after Bellaso. The Vigenere cipher is an extension of the Caesar cipher, where a sequence of letters is used as a secret key, and the secret key is repeated enough times to match the length of the plaintext; each letter of the plaintext is shifted by a position depending on the key letter at the same index. This cipher had the particularity of being easy to use and remaining strong at the time, although it is now broken.

The theory behind cryptography strengthened consequently in the 19th century. Auguste Kerckhoffs, a Dutch cryptographer, proposed a set of rules for cipher design in his paper *La Cryptographie Militaire* [Ker83]. Although some rules are outdated with the huge technological advances of this last century, cryptographers still agree with some seminal principles of his work: the security of a cipher should not depend on the secrecy of the encryption method, but rather on the secrecy of the key, a piece of information known only by the sender and the receiver.

In essence, this is the foundation of the cryptography we know today, where encryption methods rely on the use of a secret key to transform a message into ciphertext. From there, encryption algorithms are commonly designed to take two inputs: the plaintext and the secret key, and to output the ciphertext. The corresponding decryption algorithm uses the same secret key to recover the plaintext from the ciphertext. The use of a secret key offers some important advantages: it can be easily changed if it has fallen into the hands of the wrong person, and the cipher can be disclosed publicly so that it can be rightfully cryptanalysed.

The first and the second world war played crucial roles in the development of a robust cryptography. For fast long distance coordination, most communications were transmitted through telegramms or radio; these are channels that the enemy could easily read. In the first World War, the German navy used codes that substituted words and punctuation with large numbers, where the correspondence between the coded numbers and their signification was shared beforehand, physically. With an access to a sufficient number of coded messages, the cryptanalysis team of the British Admiralty, Room 40, successfully cracked the code. The decryption

of the Zimmermann Telegram, sent by the Germans to Mexico to propose an alliance if the United States were to enter the war, was one of the reason which lead the United States to take part in the war. This highlighted the need for strong cryptographic algorithms for secure communication, in particular for military purposes. The first cryptographic machines emerged after the end of the first world war. Among them, the Enigma machine would later be used by the German army throughout the Second World War. Before the war, the Polish Intelligence cryptographic team found some advanced techniques to decrypt some messages encrypted with Enigma. During the war, the British Intelligence continued the work of the Polish to successfully break the Enigma machine. According to some estimations, this would have shortened the war by approximately 2 years.

1.2 Modern Cryptography

In 1945, the American cryptographer Claude Shannon wrote a seminal paper entitled "A mathematical theory of cryptography" [Sha45], further declassified in 1949. In his work, Shannon defined a desirable property for a secure cipher: the perfect secrecy. A cipher satisfying the perfect secrecy property does not allow an attacker to learn any information on the message or the key from the ciphertext. Shannon proved that for an attacker to learn absolutely zero information on the key or the message, the key size must be at least the size of the message. However, very long secret keys are unpractical in most scenarios, and Shannon's perfect secrecy property can rarely be satisfied.

In the next decades, the development of computers highly influenced the design of cryptographic primitives. Instead of being built on character-based machines, the encryption algorithms were designed to be efficient on software, as well as integrated circuits, which are both bit-oriented. The older character-based encryption techniques (Enigma, Vigenere, Caesar...), in addition of being broken in the majority, were also unsuited for the new use cases. IBM designed in 1975 the Data Encryption Standard (DES) [Des], a cryptographic primitive which became in 1978 the first cryptography standard of the American governement. DES is a block cipher, that takes a 64-bit plaintext and a 56-bit key as input and outputs a 64-bit ciphertext. In the big picture, messages longer than 64 bits need to be split into chunks of 64 bits (after being padded if necessary), the chunks are assembled and encrypted using DES in a special manner, defined with a mode of operation. During this process, the same 56-bit key is used to encrypt multiple blocks, and this directly contradicts the perfect secrecy property defined by Claude Shannon. Therefore, the security of a block cipher relies on the non feasability for an attacker to retrieve the key: even though he theoretically possesses enough information to confirm that a given key is used for the encryption, recovering the right key is expected to require a brute force search. In the case of DES, the bruteforce search for the right key takes 2^{56} operations. It should be noted that 2^{56} operations is a significant amount of computation, but it is computable in practice; it is widely believed the National Security Agency (NSA) purposely reduced the key size of

DES from 2^{64} to 2^{56} to be able to recover secret keys by brute force.

In 1976, Diffie and Hellman proposed the first key exchange protocol [DH76]. This ground-breaking protocol allows two parties to communicate via an unsecure channel and to guarantee at the end of the communication that they know a shared secret key that any eavesdropper on the unsecure channel is unable to recover. The security of the Diffie-Hellman protocol relies on the hardness of the discrete logarithm problem. This makes it possible to fully communicate on the internet without the need to share a secret key with a secure physical channel. But the question of authenticity remains: how to make sure that you are talking to the right person?

To answer this question, Diffie and Hellman introduced the concept of public key cryptography [DH76]. Each user generates a secret key and shares to the world a public key derived from the secret key, which is associated to its identity. Messages encrypted under a public key of can be decrypted by the intended recipient, using the associated private key. The security of a public key cryptosystem relies on the difficulty to compute the secret key from the public key. One year later, Rivest, Shamir and Adleman presented the first *asymmetric* (a.k.a. public key) algorithm: RSA [RSA83], whose security relies on the hardness of large integer factorization.

In the 1980s, multiple other public key algorithms were designed, such as the Elgamal cryptosystem [Gam84] or elliptic curve cryptosystems [Mil85; Kob87]. This new genre of algorithms, the *asymmetric* algorithms, lead to the rise of public key infrastructures: when navigating on the internet, we can be ensured to communicate with the holder of the private key associated to the public key of a given website. The link between the public keys and the identities is performed and guaranteed by trusted Certificate Authorities.

The study of symmetric ciphers (where the same key is shared between the sender and the recipient) made a lot of progress in the 1990s. New cryptanalysis techniques were discovered, such as differential cryptanalysis [BS92b] and linear cryptanalysis [Mat94]. Raising concerns about the low key size of DES pushed the National Institute of Standard and Technology (NIST) to organize a standardization competition in 1997: the Advanced Encryption Standard (AES). AES was won by the Rjindael algorithm, designed by Daemen and Rijmen [DR02], and the competition gave the name to the cipher.

In the meantime, Shor made in 1994 a major breakthrough in cryptography: he presented very efficient quantum algorithms to factor large numbers or to solve the discrete logarithm problem [Sho94]. Although the quantum computer technology was (and still is) not advanced enough to make these attacks practicable, most asymmetric key algorithms were nonetheless potentially vulnerable to the "Store now, decrypt later" strategy. A party can indeed store current encrypted communications, and decrypt them in the future, when it has access to a quantum computer. Several years after Shor, Grover proposed a quantum algorithm to reduce the complexity of the brute-force search on N elements to $\sqrt{N}$ [Gro96].

1.3 Cryptography Nowadays

Undoubtedly, the quantum algorithms proposed by Shor pose a big threat to asymmetric cryptography. Some core problems in asymmetric cryptography have been broken by quantum computers, but some hard asymmetric problems, such as the Learning With Error (LWE) problem in lattices, or code-based cryptography, are still safe from quantum attacks. Schemes based on these quantum-resistant problems are said to be *post-quantum*. As RSA is still one of the most widely used asymmetric algorithms despite being broken by quantum computers, the NIST launched a competition to standardize post-quantum asymmetric cryptosystems, won by CRYSTALS-Kyber [SAB+22], CRYSTALS-Dilithium [LDK+22], Falcon [PFH+22] and SPHINCS+ [HBD+22]. The migration from quantum-broken to post-quantum cryptography is currently a major challenge in applied cryptography.

In the scope of symmetric cryptography, Grover's algorithm offers a speed-up from 2^{128} classical operations to 2^{64} quantum operations on symmetric ciphers with 128-bit key. Although it will remain out of reach by quantum computers for decades (or more probably centuries), it is recommanded to use ciphers with 256-bit key for quantum-resistance. Moreover, the AES still has a confortable security margin after more than 25 years of cryptanalysis. If AES is indeed trusted to be secure, it is not the best suited algorithm for all computing environnements. The growing usage of the Internet of Things (IoT), connected micro-chips, tiny sensors, or implanted medical devices has raised a demand for very efficient encryption methods in micro-architectures. The performance metrics of these new types of algorithms, called *lightweight* algorithms, vary depending on the use case. The NIST launched in 2019 a lightweight cryptography competition, won by ASCON [DEM+21] in 2023. This competition led to the design of many structurally different ciphers, whose cryptanalysis is still making progress, several years after their publications.

Many other cryptographic primitives fall under the umbrella of symmetric cryptography, such as hash functions, universal hash functions, or Message Authentification Codes (MACs); their design and analysis is still the subject of active research.

Chapter 2
Symmetric Cryptography

Symmetric cryptography is a field of cryptography relative to the study of secure communication through an unsecure channel using a shared secret key, a so-called symmetric key.

Contents

In this section, we denote the two users Alice and Bob and the malicious eavesdropper Eve, as highlighted by Figure 2.1. Eve can intercept all the trafic between Bob and Alice, encrypted with an encryption algorithm E using a key K, denoted E_K, and we further suppose that she is aware of some of the clear messages that Bob sends to Alice: typically, each day, the first message that Bob sends to Alice is likely to be "Hello". Thus, it is commonly assessed that the attacker Eve has access some pairs of plaintext/ciphertext encrypted with E_K. A secure symmetric encryption algorithm guarantees that Eve is unable to recover

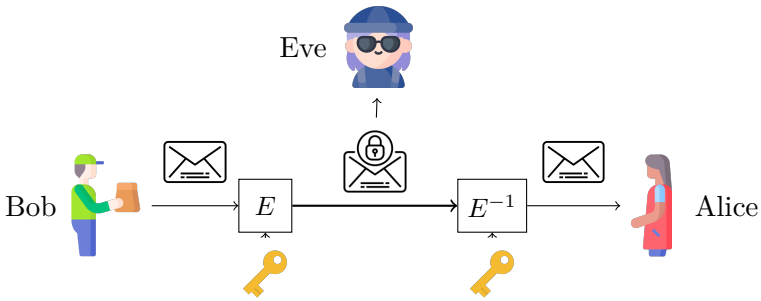


Figure 2.1: Secure communication via an unsecure channel.

any information on K or on the unkown messages. Moreover, we expect that Eve should not be able to recover any information on the key even if she chooses the messages that Bob sends to Alice. In this thesis, we consider the *black-box model*, where Eve only has access to inputs and outputs of E_K. Other models exist where Eve is able to intercept additional pieces of information on the execution of the algorithm, such as its running time, power consumption, electromagnetical radiations, or even variables used during the execution of the algorithm. These models encompass the *grey-box* and the *white-box* models.

Section 2.1 presents different cryptographic primitives studied in this thesis. Section 2.2.1 defines the notion of cryptanalysis and presents the main cryptanalysis techniques. The AES is described in Section 2.3, along with some AES-based constructions studied in this thesis. In Section 2.4, we describe how Mixed Integer Linear Programming (MILP) can be used in cryptography.

2.1 Symmetric Primitives and Constructions

Cryptographic schemes are often built from small building blocks, called *cryptographic primitives*, which are easier to analyse. Each primitive is designed to have specific and precise properties, and they can be combined to create a larger and more sophisticated construction. The security of the scheme can be proven, under the hypothesis that the primitives satisfy the required properties. In this section, we describe the main symmetric cryptography primitives.

2.1.1 (Tweakable) Block Ciphers

Block ciphers are one of the most popular types of primitives, and can be extended to encryption schemes using a mode of operation. The two most famous symmetric primitives, DES and AES, are both block ciphers.

Definition 2.1 (Block Cipher)**.** A block cipher is a family of n-bit permutations indexed with a k-bit key (choosing a key corresponds to choosing a member in the family), of signature:

$$E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$$
$$K, P \mapsto C.$$

For any fixed key K, $E_K : P \mapsto C$ is a permutation of n bits (typically, $n \in \{64, 128\}$). In the corresponding encryption algorithm, long messages are padded to a length multiple of n, split into chunks of n bits, and assembled and encrypted with E and a mode of operation.

Modes of operation. Block ciphers are used jointly with modes of operation. Among them, we can list the cipher block chaining (CBC), the cipher feedback (CFB), the output feedback (OFB) and the counter mode (CTR). Most modes of operation have proven security up to $2^{n/2}$ blocks [Rog11, p. 36], due to the birthday bound. In order to increase the proven security to 2^n, tweakable block ciphers were introduced.

Definition 2.2 (Tweakable Block Cipher [LRW02])**.** A tweakable block cipher is a family of n-bit permutations indexed with a k-bit key and a t-bit tweak, with signature:

$$E : \{0,1\}^t \times \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$$
$$T, K, P \mapsto C.$$

Notation. In a block cipher, E_K denotes the n-bit permutation under the key K. When the choice of K is unambiguous, we may use the notation E instead of E_K to denote the n-bit permutation. In a tweakable block cipher, the pair composed of the tweak and the key (T, K) is commonly called tweakey and denoted TK. Similarly to the block cipher, we denote E_{TK} the n-bit permutation under a tweakey (T, K) and denote it E if the tweakey choice is unambiguous.

The advantage of tweakable block ciphers is that the modes of operation can easily be proven up to 2^n operations. The tweak, as opposed to the key, is public and supposed to be controlled by the the attacker. This offers more freedom for an attacker to mount an attack, therefore tweakable block ciphers are usually slightly heavier than block ciphers with similar security.

2.1.2 Cryptographic Hash Functions and Sponges

Definition 2.3 (Cryptographic Hash Functions)**.** A hash function is a function that maps arbitrary messages to a fixed size (n bits) digest, of signature:

$$H : \{0,1\}^* \to \{0,1\}^n$$
$$M \mapsto D.$$

A cryptographic hash function should have the following three properties:

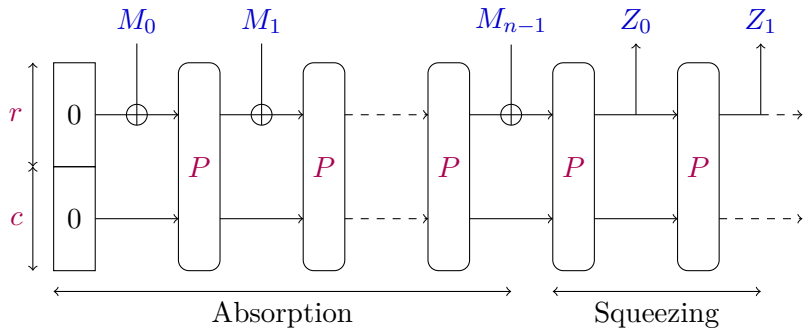


Figure 2.2: Scheme of the sponge construction.

- *Preimage Resistance*: given a digest D, it should be hard to find a message $M \in \{0,1\}^*$ such that $H(M) = D$.

- *Second Preimage Resistance*: given a message $M_0 \in \{0,1\}^*$, it should be hard to find a message $M_1 \in \{0,1\}^*$ such that $M_1 \neq M_0$ and $H(M_0) = H(M_1)$.

- *Collision Resistance*: it should be hard to find a pair of messages $(M_0, M_1) \in (\{0,1\}^*)^2$ such that $M_0 \neq M_1$ and $H(M_0) = H(M_1)$.

In particular, a cryptographic hash function should be hard to inverse. Cryptographic hash functions find numerous applications in cryptography, from password storage to digital signatures, and are used in advanced protocols, such as Fiat-Shamir for zero-knowledge proofs. In the following, we refer to them as hash functions, omitting the "cryptographic" adjective.

Hash functions are commonly built with the sponge construction, introduced in 2007 by Daemen *et al.* [BDP+07]. The sponge construction relies on a public permutation P and splits the state in two parts: the r-bit outer part and c-bit inner part, where r and c are respectively the rate and the capacity. The hash function consists in two phases: first, in the absorbtion phase, the message is added to the outer part between each iteration of P. Second, when the whole message has been ingested, the squeezing phase iterates P and outputs the outer part between each iteration. The sponge construction is depicted in Figure 2.2. The security of the sponge construction is proven up to $2^{\frac{c}{2}}$. To guarantee 128-bit security, the state size $r + c$ should be greater than 256, which makes hash functions heavier than block ciphers.

2.1.3 Constructions Based on Iterated Round Functions

Building a secure cryptographic primitives from scratch is a difficult task. Instead, designers often start from a round function, and iterate it a sufficent number of time so that the primitives possesses the required security properties.

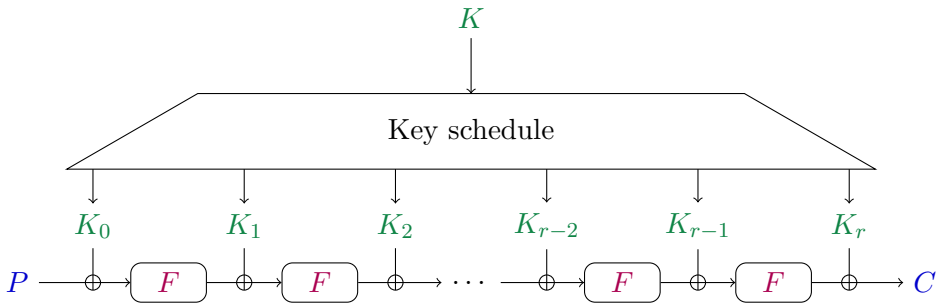


Figure 2.3: Block cipher built with the iterated round function strategy.

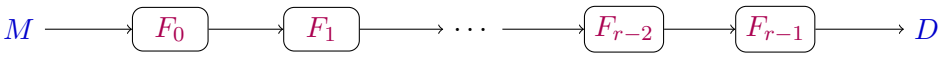


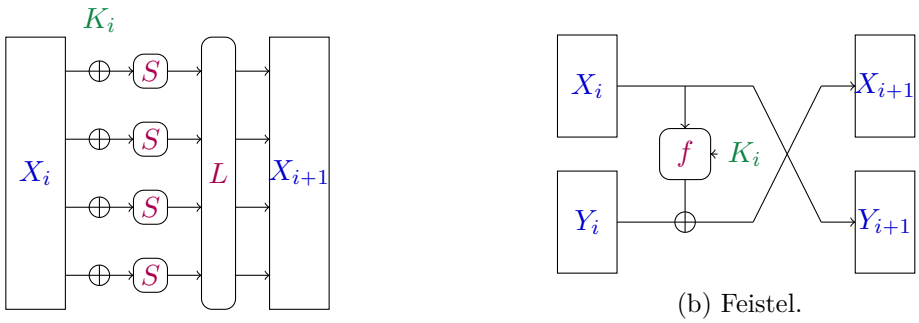
Figure 2.4: Public permutation built with the iterated round function strategy.

Block ciphers. A common choice to design a secure block cipher is to iterate a round function over a certain number of rounds r, and to add a *round key* K_i before each round and after the last round. The round keys are derived from the master key K, using a *key schedule*, and are also called *subkeys*. Such an approach is depicted in Figure 2.3. In the case of a tweakable block cipher, the key schedule is replaced by a tweakey schedule, and a *round tweakey* (or *subtweakey*) TK_i is added before each round and after the last round, derived from the master tweakey (T, K).

In order to obtain a permutation from P to C when the (twea)key is fixed, the round function F needs to be a permutation. The round i is defined as the key addition K_i followed by the round function F, and an additionnal round (twea)key addition $(T)K_r$ is performed after the last round.

Public permutations. Likewise, public permutations can be designed with the iterated round strategy. The round function is a permutation iterated with no key addition, and to avoid some attacks, the round function F_i of round i slightly differs from F_0 (typically with a constant addition). Figure 2.4 depicts this construction for public permutations.

Security benefits. An advantage of the iterated round function strategy is its convenience for cryptanalysis: cryptanalysts can try to mount attacks on reduced versions of the primitives with a lower number of rounds. The gap between the number of attacked rounds and the total number of rounds defines the security margin. If the security margin remains high enough after significant cryptanalysis, the primitive is thought to be secure.



(a) SPN.

(b) Feistel.

Figure 2.5: Round function constructions.

2.1.4 Round Function Constructions

Multiple design strategies exist to build a round function, among which three stand out by the amount of analysis they have received.

Substitution-Permutation Networks (SPN). In SPNs, the n-bit state is divided into t words of ℓ bits (with typically $\ell = 4$ or $\ell = 8$). The round key K_i is first added to the state or to part of the state. Each word then goes through a fixed permutation of ℓ bits, called S-box. This layer is called the *S-box layer* or the *non-linear layer*. Then, the state is mixed by multiplying it by a binary matrix (defined over $\mathbb{F}_2$); the corresponding step in the round function is called the *linear layer*. SPN are very popular among cryptographic designs, as highlighted by the numerous candidates using this strategy: AES [DR02], Serpent [BAK98], Present [BKL+07], Rectangle [ZBL+15], Skinny [BJK+16], Gift [BPP+17], among others.

Feistel Networks. Feistel networks are an older but still popular design strategy for block ciphers, first applied to Lucifer [Fei73] in 1973. In Feistel networks, the state is split in two parts, commonly named the left and right part. At round i, the left part goes through a function f, keyed with the round key K_i, and is XORed to the right part. Then, the left and right parts are swapped. Figure 2.5b depicts one Feistel round. Among ciphers built on Feistel Networks, we can cite the Data Encryption Standard (DES) [Des], GOST [Gos], CAST-128 [Ada97a], Simon [BSS+13], and Simeck [YZS+15], among others.

Addition-Rotations-XORs (ARX). ARX constructions developped from a more practical perspective. These operations can be implemented with a low cost in software and in hardware and the addition provides the construction with non-linearity. Multiple hash functions use this strategy, such as two SHA-3 finalists Blake [AMP+14] and Skein [FLS+10], as well as stream ciphers, such as Salsa20 [Ber08b], ChaCha [Ber08a] (supported by TLS 1.3), and block ciphers TEA [WN95] and XTEA [WN98].

2.1.5 MACs and Related Constructions

Definition 2.4 (Message Authentification Code (MAC))**.** A MAC is a key-dependant tag associated to a message, with the following signature:

$$\begin{aligned} \mathrm{MAC} : \{0,1\}^k \times \{0,1\}^* &\to \{0,1\}^n \\ K, M &\mapsto T. \end{aligned}$$

In order to be secure, a MAC should be hard to forge by a party that does not know the key K. In order to guarantee the authenticity and integrity of a message, Bob can send a message with its corresponding MAC under the shared secret key K. Upon receipt, Alice checks if the MAC of the message under the key K corresponds to the MAC sent by Bob. If so, Alice is ensured that Bob sent this message and that the message was not modified.

MACs can be combined with encryptions algorithms, with three main approaches:

- *Encrypt-then-MAC*: the plaintext is encrypted to a ciphertext, and the MAC is performed on the ciphertext.
- *MAC-and-Encrypt*: the plaintext is encrypted to a ciphertext, and the MAC is performed on the plaintext in parallel.
- *MAC-then-Encrypt*: the MAC is performed on the plaintext, and the MAC and the plaintext are encrypted into a ciphertext.

More generally, encryption schemes that provide authenticity are called *Authenticated Encryptions* (AE). Multiple other constructions exist for Authenticated Encryption, such as the Duplex construction [BDP+12], the Galois Counter Mode (GCM), or the Counter with Cipher Block Chaining-Message Authentication Code (CCM) [Dwo07b] modes of operation.

MACs can be constructed from mathematical constructions, known as *Universal Hash Functions* (UHFs). UHFs take as input a secret key and a plaintext, and map them to a fixed-length digest. Formally, we consider them as a family of functions indexed by a key, with two different security notions: almost-universal hash functions (ε-AU), and almost-XOR-universal hash functions (ε-AXU), defined as follows:

Definition 2.5 (ε-AU)**.** A family of functions $H_K : A \to B$ for $K \in \mathcal{K}$ is ε-almost-universal if:

$$\forall m \neq m' \in A,\ |\{K \in \mathcal{K} : H_K(m) = H_K(m')\}| \leqslant \varepsilon|\mathcal{K}|.$$

Definition 2.6 (ε-AXU)**.** A family of functions $H_K : A \to \mathbb{F}_2^b$ for $K \in \mathcal{K}$ is ε-almost-XOR-universal if:

$$\forall m \neq m' \in A,\ \forall d \in \mathbb{F}_2^b,\ |\{K \in \mathcal{K} : H_K(m) \oplus H_K(m') = d\}| \leqslant \varepsilon|\mathcal{K}|.$$

The ε-AU notion only requires collision resistance on average over a random key. The ε-AXU notion is a stronger variant to cover an arbitrary output difference, rather than just collisions. In particular, if H is an ε-AXU family, it is also an ε-AU family.

UHF security notions are relatively weak, so that they can be fulfilled by purely combinatorial constructions. For instance, GHASH (used in GCM [MV04]), Poly1305 [Ber05], and NH (used in UMAC [BHK+99]), rely on finite-field multiplications; they are quite fast and have provable security. UHF are on the other hand quite versatile: a UHF can be turned into a MAC with a few extra components. For instance, GMAC [Dwo07a] and Poly1305 [Ber05] are two popular MACs using the Wegman-Carter-Shoup construction [CW79; Sho96] to turn the UHF into a MAC.

In Chapter 5, we will construct a ε-AU Universal Hash Function framework, and will propose two MACs based on UHFs of the framework.

2.1.6 Stream Ciphers

Stream ciphers are, together with block ciphers, one of the two most popular families of symmetric ciphers. Instead of processing the plaintext, stream ciphers generate keysteam bits to be XORed to the plaintext. The keysteam bits should differ for each encryption, therefore an *Initialisation Vector* (IV) is generated at each encryption and used to derive keystream bits independant from one message encryption to another. In short, a stream cipher has the following signature:

$$\begin{aligned} \mathrm{S} : \{0,1\}^n \times \{0,1\}^k &\to \{0,1\}^* \\ IV, K &\mapsto (Z_i)_{i\in\mathbb{Z}}. \end{aligned}$$

Ciphertext bits are obtained by XORing plaintext bits $(P_i)_{i\in\mathbb{Z}}$ with keystream bits:

$$\forall i \in \mathbb{Z}, \qquad C_i = P_i \oplus Z_i.$$

Examples of stream ciphers include Snow [EJ00], Trivium [De 06], Salsa20 [Ber08b] and ChaCha [Ber08a].

2.2 Cryptanalysis

Unfortunately, the security properties of most constructions presented in Section 2.1 can not be proven. Instead, these security properties are studied by cryptographers through cryptanalysis: cryptographers look for the best cryptographic attacks against cryptographic algorithms under different *attacker models*. Cryptographic attacks are algorithms that break a security property, by for instance recovering the secret key used in a block cipher.

The confidence in a particular cryptographic primitive is gained after a sufficient amount of cryptanalysis is performed against this primitive, if the best attack does not threaten its security. A cryptographic attack is valid if its time complexity is

less than 2^s, where s is the claimed security of the underlying algorithm. Common security parameters are $s = 128$, as it is unlikely that at any time in the future, a party will be capable of running 2^{128} classical operations. For sensitive encryption, $s = 256$ can be used to further mitigate the risks.

The goals and types of cryptographic attacks heavily depend on the cryptographic constructions under consideration and on the attacker models. These attacker models are chosen to represent the capabilities of an attacker in the real world. Naturally, different use cases of cryptography lead to different attacker models. In Section 2.2.1, we describe different types of attacks and attacker models against symmetric cryptographic constructions. We then detail differential cryptanalysis in Section 2.2.2, a concept that will be extensively studied in this thesis, and finally introduce other cryptanalysis techniques in Section 2.2.3.

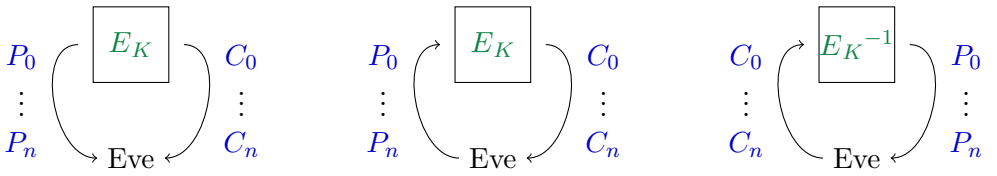
2.2.1 Attacker Models

2.2.1.1 Cryptanalysis of block ciphers

Let us denote E a block cipher with a k-bit key K, claiming a security of k. In real world scenarios, an attacker may have access to pairs of plaintext/ciphertext values of E. For example, the attacker may know that the plaintext sent is a TLS packet, and by another way capture the corresponding ciphertext. Using pieces of information on the structure of a TLS packet, the attacker may recover pairs of corresponding plaintext/ciphertext through E.

Attacker model. A common attacker model assumption is to consider that the attacker has access to an oracle which gives the attacker pairs of plaintext/ciphertext values. From there, three attacker models are described in Figure 2.6. In the known plaintext model, the attacker gets access to a series of plaintext/ciphertext pairs under the key K, but does not choose either of them. In the chosen plaintext (resp. ciphertext) models, the attacker chooses a set of plaintexts (resp. ciphertexts) to encrypt (resp. decrypt) through E_K (resp. E_K^{-1}) and gets the results. In the CP and CC models, the attacker is supposed to query the data once and for all: the plaintexts $P_0 \dots P_n$ (resp. ciphertext $C_0 \dots C_n$) are queried at the beginning of the attack as a single big query. A variant of this attacker model is the Adaptatively Chosen Plaintext (ACP) or Ciphertext (ACC) models, in which the attacker may ask for queries which depend on previous output queries. Combinations of previously mentionned models are also possible. Although these attacker models may relate differently to real-world scenarios, they are often all considered in the cryptanalysis of block ciphers, since they all give important insights on the security of a block cipher. Another special yet important attacker model is the Related Key (RK) model. The attacker has access to encryption and decryption oracle under the secret key, and may also ask to add a chosen difference to the key.

We will now present the different goals of an attack against block ciphers. Two main types of attack exist against block ciphers: distinguishing and key-recovery attacks.



(a) Known plaintext (KP). (b) Chosen plaintext (CP). (c) Chosen ciphertext (CC).

Figure 2.6: Attacker models for block ciphers.

Distinguishing attacks. In distinguishing attacks, the attacker is given one of two possible oracles, each with probability 1/2:

- The first oracle corresponds to encryptions/decryptions of the block cipher under a secret key K.
- The second oracle is an oracle simulating a random permutation.

The goal of the attacker is to distinguish between both cases, with success probability $p > 1/2$.

Key-recovery attacks. In key-recovery attacks, the attacker has access to an oracle depending on its attacker model, and its goal is to recover the key K. A variant of the key-recovery attack is the equivalent-key-recovery attack, where a key-dependant piece of information is recovered by the attacker, which is enough to compute part of the plaintext from the ciphertext or vice-versa.

Complexity of an attack. The complexity of an attack against a block cipher has 3 components: the data complexity, corresponding to the number of queries made to the oracle, the time complexity, corresponding to the number of operations performed by the algorithm (without taking into account parallelization), and the memory complexity, corresponding to the maximum amount of storage space needed to mount the attack. In cryptography, it is standard to measure the time complexity as the number of equivalent encryptions, as this allows to compare the time complexity of the attack to the bruteforce search, which requires 2^k encryptions for a success probability of 1. In this thesis, these three complexity components are referred as (D, T, M).

Attacking weak keys. An attacker can mount an attack by hoping that a key belongs to a set of weak keys of size 2^w. In this case, the attack has a low success probability $p = 2^{w-k}$, and is acceptable if the time complexity T is such that $T < 2^w$, since 2^w corresponds to the complexity of the brute-force over the weak key space.

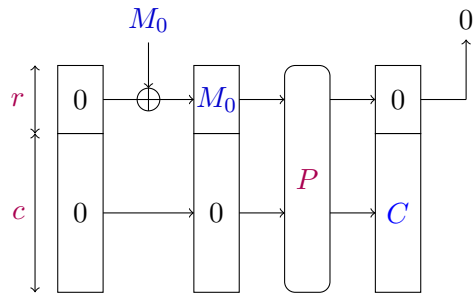


Figure 2.7: Preimage attack on a Sponge construction from a CICO solution.

2.2.1.2 Cryptanalysis of cryptographic permutations

As mentionned in Section 2.1.2, some constructions, such as the Sponge construction [BDP+07], rely on strong cryptographic permutations. The authors of Keccak propose to analyse the security of such permutations with the *Constraint Input Constraint Output* (CICO) problem [BDP+09]. The CICO problem has become a standard for the analysis of cryptographic permutations. A natural instance of the problem can be stated in the following form:

Problem 2.1 (CICO problem)**.** *Let* $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ *be a permutation and* $1 \leqslant \ell < n$ *an integer. The goal is to find* $x \in \{0\}^\ell \times \mathbb{F}_2^{n-\ell}$ *such that* $F(x) \in \{0\}^\ell \times \mathbb{F}^{n-\ell}$.

The generic approach to solve the CICO problem is to generate many inputs $x \in \{0\}^\ell \times \mathbb{F}_2^{n-\ell}$, then apply F on them and check whether $F(x) \in \{0\}^\ell \times \mathbb{F}^{n-\ell}$ holds. After 2^ℓ applications of F on average, a CICO solution is found. Therefore, we often consider the CICO problem with a large ℓ so that the generic attack is infeasible (typically $\ell \geqslant 128$). Note that the generic CICO attack can also be considered the other way around: generate random $y \in \{0\}^\ell \times \mathbb{F}_2^{n-\ell}$, apply F^{-1} and check whether $F^{-1}(y) \in \{0\}^\ell \times \mathbb{F}_2^{n-\ell}$, which should happen on average after 2^ℓ iterations. This is the reason that lead us to fix the same number of bits in input and output in our formulation of the CICO problem. Note that the l-bit prefix of x and $F(x)$ are arbitrarily fixed to 0 in the original CICO problem, but we expect the same difficulty with any other input/output values.

In practice, a variant to the CICO problem (with fixed bits in the c-bit input suffix and fixed bits in the r-bit output prefix) can be solved to find a preimage of 0 under a sponge function, as described in Figure 2.7.

2.2.2 Differential Cryptanalysis

Differential cryptanalysis is one of the main attack families against cryptographic algorithms. It was introduced by Biham et Shamir in 1991 [BS92a] to break 15 out of 16 rounds of DES. The idea is to introduce a difference in a message or another public input controlled by the attacker, and to track the propagation of this difference through the cryptographic algorithm. By carefully analysing the probability of the difference propagation, it is possible to estimate the probability

that a certain difference occurs in the output of the cryptographic algorithm, and use this knowledge to mount an attack.

2.2.2.1 Formalisation of differential cryptanalysis

The following definition is a formal description of a differential over a function.

Definition 2.7 (Differential probability)**.** Let $F : \mathbb{F}_2^i \to \mathbb{F}_2^o$ be a function, $\Delta_{\text{in}} \in \mathbb{F}_2^i$ and $\Delta_{\text{out}} \in \mathbb{F}_2^o$ be differences. A differential $\Delta_{\text{in}} \xrightarrow[F]{} \Delta_{\text{out}}$ through a function F has a probability defined as:

$$\begin{aligned} p = \Pr[\Delta_{\text{in}} \xrightarrow[F]{} \Delta_{\text{out}}] &= \Pr_{X \in \mathbb{F}_2^i}[F(X) + F(X + \Delta_{\text{in}}) = \Delta_{\text{out}}] \\ &= \frac{1}{2^i} \left| \left\{ X \in \mathbb{F}_2^i \mid F(X) + F(X + \Delta_{\text{in}}) = \Delta_{\text{out}} \right\} \right|. \end{aligned}$$

In this case, we denote the differential $\Delta_{\text{in}} \xrightarrow[F]{p} \Delta_{\text{out}}$.

The probability of a differential can theoretically be computed by counting the number of inputs X that satisfy the differential; in practice, the number of possible inputs X is too large and counting the number of such inputs naïvely is infeasible.

When considering block ciphers, the function F is a permutation ($i = o$) and depends on the key K. We define the expected differential probability as an average on all keys:

Definition 2.8 (Expected differential probability)**.** Let E be a block cipher $E_K : \mathbb{F}_2^n \to \mathbb{F}_2^n$ with $K \in \mathcal{K}$, and let $\Delta_{\text{in}} \in \mathbb{F}_2^n, \Delta_{\text{out}} \in \mathbb{F}_2^n$ be differences. A differential $\Delta_{\text{in}} \xrightarrow[E]{} \Delta_{\text{out}}$ through E has an expected differential probability p defined as:

$$\begin{aligned} \Pr[\Delta_{\text{in}} \xrightarrow[E]{} \Delta_{\text{out}}] &= \operatorname*{Avg}_{K \in \mathcal{K}} \left[\Pr_{X \in \mathbb{F}_2^n}[E_K(X) + E_K(X + \Delta_{\text{in}}) = \Delta_{\text{out}}] \right] \\ &= \frac{1}{2^n |\mathcal{K}|} \left| \{ (K, X) \in \mathcal{K} \times \mathbb{F}_2^n \mid E_K(X) + E_K(X + \Delta_{\text{in}}) = \Delta_{\text{out}} \} \right|. \end{aligned}$$

In this case, we denote the differential $\Delta_{\text{in}} \xrightarrow[E]{p} \Delta_{\text{out}}$.

For permutations, the probability of the backward differential is the same as the probability as the forward differential. This is highlighted by the following lemma, easily provable by counting the elements satisfying the differential transition:

Lemma 2.1. *Let $\Delta_{\text{in}} \xrightarrow[E]{p} \Delta_{\text{out}}$ be a differential over a family of permutation E. The differential $\Delta_{\text{out}} \xrightarrow[E^{-1}]{} \Delta_{\text{in}}$ has a probability p.*

Computing the exact probability of a given differential over an arbitrary family or permutations is challenging. Therefore, to estimate the probability of a differential of a SPN cipher, a common method is to assume that the cipher is a *Markov cipher*.

Definition 2.9 (Markov cipher [LMM91])**.** A Markov cipher is an iterated cipher $E_K = E_K^{r-1} \circ \cdots \circ E_K^0$ operating on $\mathbb{F}_2^n$, such that:

$$\forall i \in [\![0, r-1]\!], \Delta_{\text{in}}, \Delta_{\text{out}} \in \mathbb{F}_2^n, \qquad \Pr_{K \in \mathcal{K}}[E_K^i(X) + E_K^i(X + \Delta_{\text{in}}) = \Delta_{\text{out}} \mid X = \alpha]$$

is independant of α. This typically happens when E_K^i starts with a subkey addition and if the subkey is uniformely distributed over $\mathbb{F}_2^n$ when K is uniformely distributed of $\mathcal{K}$.

If the round subkeys of a Markov ciphers are independant, the probability of a multiple SPN rounds can be computed using *differential trails*.

Definition 2.10 (Differential trails)**.** Let $E_K = E_K^{r-1} \circ \cdots \circ E_K^0$ be an iterated cipher on $\mathbb{F}_2^n$. Let us denote $\tilde{E}_K^i = E_K^i \circ \cdots \circ E_K^0$ for $i \in [\![0, r-1]\!]$. A differential trail $\Delta_0 \xrightarrow[E_K^0]{} \Delta_1 \xrightarrow[E_K^1]{} \ldots \Delta_{r-1} \xrightarrow[E_K^{r-1}]{} \Delta_r$ over E_K has a probability defined as:

$$\Pr[\Delta_0 \xrightarrow[E_K^0]{} \ldots \xrightarrow[E_K^{r-1}]{} \Delta_r] = \Pr_{K,X}[\forall i \in [\![1, r]\!], \tilde{E}_K^{i-1}(X) + \tilde{E}_K^{i-1}(X + \Delta_0) = \Delta_i].$$

In particular, it follows from the definition that:

$$\begin{aligned}\Pr[\Delta_0 \xrightarrow[E_K^0]{} \Delta_1 \xrightarrow[E_K^1]{} \ldots \Delta_{r-1} \xrightarrow[E_K^{r-1}]{} \Delta_r] &\leqslant \Pr_{K,X}[E_K(X) + E_K(X + \Delta_0) = \Delta_r] \\ &= \Pr[\Delta_0 \xrightarrow[E_K]{} \Delta_r],\end{aligned}$$

and it can easily be proven that:

$$\sum_{\Delta_1 \ldots \Delta_{r-1} \in \mathbb{F}_2^n} \Pr[\Delta_0 \xrightarrow[E_K^0]{} \Delta_1 \xrightarrow[E_K^1]{} \ldots \Delta_{r-1} \xrightarrow[E_K^{r-1}]{} \Delta_r] = \Pr[\Delta_0 \xrightarrow[E_K]{} \Delta_r]. \tag{2.1}$$

If E_K is a Markov cipher with independant round keys, the round transitions are independant, so the probability of the differential trail can be computed as the product of each round transition probability:

$$\Pr[\Delta_0 \xrightarrow[E_K^0]{} \Delta_1 \xrightarrow[E_K^1]{} \ldots \Delta_{r-1} \xrightarrow[E_K^{r-1}]{} \Delta_r] = \prod_{i=0}^{r-1} \Pr\left[\Delta_i \xrightarrow[E_K^i]{} \Delta_{i+1}\right].$$

The Markov cipher assumption is very convenient as it allows to multiply the differential probabilities of each round and obtain a lower bound on the probability, but its accuracy needs to be carefully analysed. For unkeyed functions, computing the probability of differential trails as the product of round transition probabilities is questionnable, as no subkey addition occurs, and the notion of Markov cipher lacks sense. This is discussed by Beyne and Rijmen for ciphers in the fixed key model [BR22b]. Even for keyed ciphers, the Markov cipher assumption has been challenged, for instance on ARX ciphers when multiple operations are performed with no key addition [XLJ+22], or when the subkey addition is partial and when the subkeys are not independant [PT22]. Despite these inaccuracies, differential trails are still the most popular way of finding high probability differentials over iterated ciphers.

Differential transitions over simple components. One can compute probabilities of differentials over simple components of the iterated cipher, and multiply the probabilities to give a lower bound of the probability of the full-round differential. For some special components, computing the exact probability of differential trails is feasible:

- When the function F is linear (and independant of the key), the differential $\Delta_{\text{in}} \xrightarrow[F]{} \Delta_{\text{out}}$ occurs with probability $\mathbb{1}_{F(\Delta_{\text{in}})=\Delta_{\text{out}}}$.
- When the function F is a constant addition or a key addition, the differential $\Delta_{\text{in}} \xrightarrow[F]{} \Delta_{\text{out}}$ occurs with probability $\mathbb{1}_{\Delta_{\text{in}}=\Delta_{\text{out}}}$.
- When the function F is an S-box layer (independant of the key), i.e. there exists an integer k such that $\ell = \frac{n}{k}$ is a small integer (typically $\ell \in \{4, 8\}$) and:

  $$X = X_0 \| \dots \| X_{k-1} \qquad F(X) = S_0(X_0) \| \dots \| S_{k-1}(X_{k-1}),$$

  we denote:

  $$\Delta_{\text{in}} = \Delta_{\text{in}}^0 \| \dots \| \Delta_{\text{in}}^{k-1} \qquad \Delta_{\text{out}} = \Delta_{\text{out}}^0 \| \dots \| \Delta_{\text{out}}^{k-1}.$$

  For $i \in [\![0, k-1]\!]$, we may compute s_i the number of partial solutions for X_i by enumerating all 2^ℓ possible X_i:

  $$s_i = \left| \{X_i \in \mathbb{F}_2^\ell \mid S_i(X_i + \Delta_{\text{in}}^i) = \Delta_{\text{out}}^i\} \right|.$$

  Then, it follows that:

  $$\begin{aligned} \Pr[\Delta_{\text{in}} \xrightarrow[F]{} \Delta_{\text{out}}] &= \frac{1}{2^n} \left| \{X \in \mathbb{F}_2^n \mid F(X) + F(X + \Delta_{\text{in}}) = \Delta_{\text{out}}\} \right| \\ &= \frac{\prod_{i=0}^{k-1} s_i}{2^n}. \end{aligned} \tag{2.2}$$

In a differential trail, S-boxes which have a non-zero difference in input and output are called *active S-boxes*. We now study in more detail the differential transitions over an S-box S.

2.2.2.2 Differential properties of S-boxes

Let us first define S-boxes.

Definition 2.11 (S-box)**.** An ℓ-to-k-bit S-box is a function of signature $\mathbb{F}_2^\ell \to \mathbb{F}_2^k$.

In this thesis, we only consider bijective S-boxes with $\ell = k$; we call them ℓ-bit S-boxes. In the following, S denotes the S-box of a cipher, and ℓ the bit size of the S-box.

Definition 2.12 (Difference Distribution Table)**.** The *Difference Distribution Table* (DDT) of an S-box S is a $2^\ell \times 2^\ell$ table DDT_S such that for all $\Delta_{\text{in}}, \Delta_{\text{out}} \in \mathbb{F}_2^\ell$,

$$\begin{aligned}\mathrm{DDT}_S[\Delta_{\text{in}}, \Delta_{\text{out}}] &= |\{X \in \mathbb{F}_2^\ell | S(X) + S(X + \Delta_{\text{in}}) = \Delta_{\text{out}}\}| \\ &= 2^\ell \Pr[\Delta_{\text{in}} \underset{S}{\rightarrow} \Delta_{\text{out}}].\end{aligned}$$

The DDT is an important tool for cryptographers since it keeps the information of all the differential transitions over an S-box. It can be precomputed in $2^{2\ell}$ lookups, and is a great tool for designers to find S-boxes with good properties, or for cryptanalysts to efficiently mount attacks based on differential trails. A first property of the DDT of an S-box over $\mathbb{F}_2^\ell$ is that each entry of the DDT is even. Indeed, if X is a solution to $S(X) + S(X + \Delta_{\text{in}}) = \Delta_{\text{out}}$, then $X + \Delta_{\text{in}}$ is also solution: all solutions go by pair if $\Delta_{\text{in}} \neq 0$ (the case $\Delta_{\text{in}} = 0$ is trivial).

Definition 2.13 (Differential Uniformity)**.** The *Differential Uniformity* of an S-box S is defined as:

$$\mathrm{DU}_S = \max_{\substack{\Delta_{\text{in}} \in \mathbb{F}_2^\ell \setminus \{0\} \\ \Delta_{\text{out}} \in \mathbb{F}_2^\ell \setminus \{0\}}} \left[\mathrm{DDT}_S[\Delta_{\text{in}}, \Delta_{\text{out}}]\right].$$

The differential uniformity relates to the strength of an S-box. Over $\mathbb{F}_2^\ell$, S-boxes have differential uniformities of at least 2; an S-box with differential uniformity exactly 2 is said to be Almost Perfectly Non-linear (APN).

Property 2.1. *Let S be a ℓ-bit S-box. We have:*

$$\operatorname*{Avg}_{\substack{\Delta_{\text{in}} \in \mathbb{F}_2^\ell \setminus \{0\} \\ \Delta_{\text{out}} \in \mathbb{F}_2^\ell \setminus \{0\}}} DDT_S[\Delta_{\text{in}}, \Delta_{\text{out}}] = \frac{2^\ell}{2^\ell - 1}.$$

Proof.

$$\operatorname*{Avg}_{\substack{\Delta_{\text{in}} \in \mathbb{F}_2^\ell \setminus \{0\} \\ \Delta_{\text{out}} \in \mathbb{F}_2^\ell \setminus \{0\}}} \mathrm{DDT}_S[\Delta_{\text{in}}, \Delta_{\text{out}}] = \frac{\left|\{(X, \Delta_{\text{in}}, \Delta_{\text{out}}) \in \mathbb{F}_2^\ell \times (\mathbb{F}_2^\ell \setminus \{0\})^2 \,|\, S(X + \Delta_{\text{in}}) + S(X) = \Delta_{\text{out}}\}\right|}{(2^\ell - 1)^2},$$

Given $X \in \mathbb{F}_2^\ell$ and $\Delta_{\text{in}} \in \mathbb{F}_2^\ell \setminus \{0\}$, it is easy to see that there is exactly one $\Delta_{\text{out}} \neq 0$ verifying $S(X + \Delta_{\text{in}}) + S(X) = \Delta_{\text{out}}$, therefore:

$$\begin{aligned}\operatorname*{Avg}_{\substack{\Delta_{\text{in}} \in \mathbb{F}_2^\ell \setminus \{0\} \\ \Delta_{\text{out}} \in \mathbb{F}_2^\ell \setminus \{0\}}} \mathrm{DDT}_S[\Delta_{\text{in}}, \Delta_{\text{out}}] &= \frac{1}{(2^\ell - 1)^2} |\{(X, \Delta_{\text{in}}) \in \mathbb{F}_2^\ell \times \mathbb{F}_2^\ell \setminus \{0\}\}| \\ &= \frac{2^\ell (2^\ell - 1)}{(2^\ell - 1)^2} \\ &= \frac{2^\ell}{2^\ell - 1}.\end{aligned}$$

□

This implies the following lemma, that we extensively use in Chapter 3.

Lemma 2.2. *Let S be a 8-bit S-box. For two random difference $\Delta_{\text{in}} \neq 0, \Delta_{\text{out}} \neq 0$, the equation $S(X + \Delta_{\text{in}}) + S(X) = \Delta_{\text{out}}$ has approximately one solution X on average.*

Precisely, the number of solution the equation is $\frac{256}{255}$ on average, but it is reasonable to approximate it to 1, as it greatly simplifies the writing of the attacks.

Equation 2.2 can now be revisited with the DDT of an S-box S, when F is an S-box layer using a single S-box S. Let

$$\Delta_{\text{in}} = \Delta_{\text{in}}^0 \| \dots \| \Delta_{\text{in}}^{k-1} \qquad \Delta_{\text{out}} = \Delta_{\text{out}}^0 \| \dots \| \Delta_{\text{out}}^{k-1}.$$

We have:

$$\Pr[\Delta_{\text{in}} \xrightarrow[F]{} \Delta_{\text{out}}] = \frac{1}{2^n} \prod_{i=0}^{k-1} \text{DDT}_S[\Delta_{\text{in}}^i, \Delta_{\text{out}}^i]. \tag{2.3}$$

Computing the probability of a differential through an S-box layer being done efficiently, we can now compute the probability of a differential through a SPN round function $F = L \circ SB \circ AK$, where AK is the key addition, SB is the S-box layer, and L is the linear layer. First, the key addition does not change the difference of the input. Then, we remark that:

$$\{X \in \mathbb{F}_2^n \mid L \circ SB(X + \Delta_{\text{in}}) = \Delta_{\text{out}}\} = \{X \in \mathbb{F}_2^n \mid SB(X + \Delta_{\text{in}}) = L^{-1}(\Delta_{\text{out}})\}.$$

The size of the second set is related to the differential probability through a S-box layer from a difference Δ_{in} to a difference $L^{-1}(\Delta_{\text{out}})$ and can be computed efficiently with Equation 2.3.

2.2.2.3 Truncated dfferential cryptanalysis

In some cases, the probability of a differential is well estimated by the highest probability differential trail with the given input/output differences. However, for some ciphers, there is not a single dominant differential trail, but instead many small probability ones, whose probabilities can be summed up to better estimate the probability of the full differential, using Equation 2.1; this is known as the clustering effect. In addition, there are some cases where many high probability differentials exist over a cipher with different input and output differences. These observations lead to the introduction of *truncated differential cryptanalysis* by Knudsen in 1994 [Knu95]. Instead of considering pairs with fixed input/output differences, a truncated differential tracks pairs whose input and output differences belong to predefined sets of differences. Truncated differentials are constructed to be of probability higher than standard differentials, since they include differentials with multiple input/output differences. The same holds for truncated differential trails, which include many differential trails.

Definition 2.14 (Truncated differential)**.** Let E be a block cipher $E_K : \mathbb{F}_2^n \to \mathbb{F}_2^n$, with $K \in \mathcal{K}$. Let $\mathcal{D}_{\text{in}}$ and $\mathcal{D}_{\text{out}}$ be sets of elements of $\mathbb{F}_2^n$. The probability of the truncated differential $\mathcal{D}_{\text{in}} \xrightarrow[E_K]{} \mathcal{D}_{\text{out}}$ is defined as:

$$\Pr\left[\mathcal{D}_{\text{in}} \xrightarrow[E_K]{} \mathcal{D}_{\text{out}}\right] = \operatorname*{Avg}_{\substack{K\in\mathcal{K}\\ \Delta_{\text{in}}\in\mathcal{D}_{\text{in}}}} \Pr_X[E_K(X) + E_K(X+\Delta_{\text{in}}) \in \mathcal{D}_{\text{out}}].$$

The probability of a truncated differential trail can also be written as:

$$\begin{aligned}\Pr\left[\mathcal{D}_{\text{in}} \xrightarrow[E_K]{} \mathcal{D}_{\text{out}}\right] &= \sum_{\Delta_{\text{out}}\in\mathcal{D}_{\text{out}}} \left(\operatorname*{Avg}_{\substack{K\in\mathcal{K}\\ \Delta_{\text{in}}\in\mathcal{D}_{\text{in}}}} \Pr_X[E_K(X) + E_K(X+\Delta_{\text{in}}) = \Delta_{\text{out}}] \right) \\ &= |\mathcal{D}_{\text{out}}| \operatorname*{Avg}_{\substack{K\in\mathcal{K}\\ \Delta_{\text{in}}\in\mathcal{D}_{\text{in}}\\ \Delta_{\text{out}}\in\mathcal{D}_{\text{out}}}} \Pr_X[E_K(X) + E_K(X+\Delta_{\text{in}}) = \Delta_{\text{out}}].\end{aligned}$$

It follows straightfowardly that the probabilities of forward and backward truncated differentials are linked:

Lemma 2.3. *Let E be a block cipher $E_K : \mathbb{F}_2^n \to \mathbb{F}_2^n$, with $K \in \mathcal{K}$. Let $\mathcal{D}_{\text{in}}$ and $\mathcal{D}_{\text{out}}$ be sets of elements of $\mathbb{F}_2^n$. We have:*

$$\Pr\left[\mathcal{D}_{\text{in}} \xrightarrow[E_K]{} \mathcal{D}_{\text{out}}\right] |\mathcal{D}_{\text{out}}|^{-1} = \Pr\left[\mathcal{D}_{\text{out}} \xrightarrow[E_K^{-1}]{} \mathcal{D}_{\text{in}}\right] |\mathcal{D}_{\text{in}}|^{-1}.$$

The probability of truncated differentials is easily computable on a single round, similarly to standard differentials. Therefore, we introduce *truncated differential trails* whose probability give a lower bound to the full truncated differential.

Definition 2.15 (Truncated differential trails)**.** Let $E_K = E_K^{r-1} \circ \cdots \circ E_K^0$ be an iterated block cipher $E_K : \mathbb{F}_2^n \to \mathbb{F}_2^n$, with $K \in \mathcal{K}$. We denote $\tilde{E}_K^i = E_K^i \circ \cdots \circ E_K^0$ for $i \in [\![0, r-1]\!]$. Let $\mathcal{D}_0 \dots \mathcal{D}_r$ be sets of elements of $\mathbb{F}_2^n$. The truncated differential $\mathcal{D}_0 \xrightarrow[E_K^0]{} \mathcal{D}_1 \xrightarrow[E_K^1]{} \dots \mathcal{D}_{r-1} \xrightarrow[E_K^{r-1}]{} \mathcal{D}_r$ is of probability:

$$\Pr[\mathcal{D}_0 \xrightarrow[E_K^0]{} \dots \xrightarrow[E_K^{r-1}]{} \mathcal{D}_r] = \operatorname*{Avg}_{\substack{K\in\mathcal{K}\\ \Delta_0\in\mathcal{D}_0}} \left[\Pr_X[\forall i \in [\![1,r]\!], \tilde{E}_K^{i-1}(X) + \tilde{E}_K^{i-1}(X+\Delta_0) \in \mathcal{D}_i]\right].$$

And straightfowardly, we have:

$$\Pr[\mathcal{D}_0 \xrightarrow[E_K^0]{} \mathcal{D}_1 \xrightarrow[E_K^1]{} \dots \mathcal{D}_{r-1} \xrightarrow[E_K^{r-1}]{} \mathcal{D}_r] \leqslant \Pr[\mathcal{D}_0 \xrightarrow[E_K]{} \mathcal{D}_r].$$

By abuse of notation, we denote $\mathcal{D}_{\text{in}} \xrightarrow[E_K]{} \Delta_{\text{out}}$ (resp. $\Delta_{\text{in}} \xrightarrow[E_K]{} \mathcal{D}_{\text{out}}$) the truncated differential from a set of differences $\mathcal{D}_{\text{in}}$ (resp. a single difference Δ_{in}) to the set of the single element Δ_{out} (resp. a set of differences $\mathcal{D}_{\text{in}}$).

Unlike for differential trails, the Markov cipher and subkey independance assumptions do not suffice to state that the probability of a truncated differential trail is the product of the transition probabilities of the truncated differentials composing it. In addition to those two hypothesis, we need the condition that each internal truncated differential are *well-distributed*, a notion that is not defined in the literature but that we briefly introduce here.

Definition 2.16 (Well-distributed truncated differentials)**.** Let E be a block cipher $E_K : \mathbb{F}_2^n \to \mathbb{F}_2^n$, with $K \in \mathcal{K}$. A truncated differential $\mathcal{D}_{\text{in}} \xrightarrow[E_K]{} \mathcal{D}_{\text{out}}$ is said to be well-distributed if the output differences are all equally reached, i.e.:

$$\forall \Delta_{\text{out}} \in \mathcal{D}_{\text{out}}, \Pr\left[\mathcal{D}_{\text{in}} \xrightarrow[E_K]{} \Delta_{\text{out}}\right] = \frac{1}{|\mathcal{D}_{\text{out}}|} \Pr\left[\mathcal{D}_{\text{in}} \xrightarrow[E_K]{} \mathcal{D}_{\text{out}}\right].$$

This allows to define the following lemma:

Lemma 2.4. *Let E be an iterated Markov cipher with independant subkeys $E_K = E_K^{r-1} \circ \cdots \circ E_K^0$ of $\mathbb{F}_2^n \to \mathbb{F}_2^n$ indexed with $K \in \mathcal{K}$, and $\mathcal{D}_0 \xrightarrow[E_K^0]{} \mathcal{D}_1 \xrightarrow[E_K^1]{} \ldots \mathcal{D}_{r-1} \xrightarrow[E_K^{r-1}]{} \mathcal{D}_r$ be a truncated differential trail. If $\mathcal{D}_i \xrightarrow[E_K^i]{} \mathcal{D}_{i+1}$ is well-distributed for all i, then the truncated differential trail probability can be computed:*

$$\Pr\left[\mathcal{D}_0 \xrightarrow[E_K^0]{} \mathcal{D}_1 \xrightarrow[E_K^1]{} \ldots \mathcal{D}_{r-1} \xrightarrow[E_K^{r-1}]{} \mathcal{D}_r\right] = \prod_{i=0}^{r-1} \Pr\left[\mathcal{D}_i \xrightarrow[E_K^i]{} \mathcal{D}_{i+1}\right].$$

Most of the truncated differentials are not well-distributed in practice, but it is still common in the literature to use the above formula to approximate the probability of truncated differential trails.

2.2.3 Other Cryptanalysis Techniques

Other cryptanalysis families exist and need to be considered by the designers of cryptographic algorithms. We here present an overview of the most popular cryptanalysis techniques.

2.2.3.1 Linear cryptanalysis

Matsui introduced linear cryptanalysis in 1993 [Mat94]; it is the analysis of biased linear relations between the input, the output and the key of a cipher. Matsui proposed two algorithms in his original paper, and applied them on full-round DES.

Definition 2.17 (Linear approximation)**.** Let $E_K : \mathbb{F}_2^n \mapsto \mathbb{F}_2^n$ be a block cipher with a k-bit key K. The masks $\gamma_P \in \mathbb{F}_2^n$, $\gamma_C \in \mathbb{F}_2^n$ and $\gamma_K \in \mathbb{F}_2^k$ constitute a biased linear approximation of the cipher E_K if there exists a non-negligible ϵ such that:

$$\Pr_{K,X}\left[\langle X, \gamma_P\rangle + \langle E_K(X), \gamma_C\rangle + \langle K, \gamma_K\rangle = 0\right] = \frac{1}{2} + \epsilon.$$

If $\epsilon \gg 2^{-n/2}$, one can use this approximation to recover a linear relation on the key bits ($\langle K, \gamma_K\rangle$), by generating enough data $X, E_K(X)$ and checking if $\langle X, \gamma_P\rangle + \langle E_K(X), \gamma_C\rangle$ takes the value 0 or 1 more frequently.

Linear approximations are commonly found with linear trails by combining the linear approximations of each round. A variant of linear cryptanalysis is the differential-linear cryptanalysis introduced in 1994 by Langford and Hellman [LH94], which combines a differential with a linear approximation to yield longer attacks.

2.2.3.2 Algebraic cryptanalysis

Algebraic cryptanalysis is the study of the algebraic representation of a cipher E_K and its derivated attacks. The bits of the output of a cipher $E_K(X)$ can always be represented with polynomial Boolean equations from the input bits of X and the bits of the key K, using the *Algebraic Normal Form* (ANF) of E_K. However, the polynomials at play are often too large to be computable in practice. Some properties of the algebraic representation of the cipher can still be used to derive attacks. If the ANF if of low degree, interpolation attacks [JK97], integral attacks [KW02] or cube attacks [DS09] can be mounted. A deeper look into algebraic attacks is given in Chapter 6.

2.2.3.3 Impossible differential cryptanalysis

In contrast to differential cryptanalysis, impossible differential attacks exploit differentials that happen with probability 0 on E_K, i.e. $\Delta_{\text{in}} \xrightarrow[E_K]{} \Delta_{\text{out}}$ such that $\Pr\left[\Delta_{\text{in}} \xrightarrow[E_K]{} \Delta_{\text{out}}\right] = 0$. Usually, such a property is used as a distinguisher inside the cipher, and key-recovery rounds are added before and after the distinguisher. Keys that lead to differences Δ_{in} in input and Δ_{out} in output of the distinguisher are impossible thus are discarded, and the attack proceeds until most keys are discarded, and only a few key candidates remain. We point the reader to the paper of Boura *et al.* [BNS14] for detailed explanations.

2.2.3.4 Boomerang attacks

Boomerang attacks, introduced by Wagner in 1999 [Wag99], split a cipher E in two sub-ciphers $E = E_1 \circ E_0$, and exploits two short differential trails on E_0 and E_1 to yield an adaptatively chosen ciphertext (ACC) attack. The boomerang attack is decribed in details in Chapter 4.

2.2.3.5 Meet-in-the-Middle attacks

In Meet-in-the-Middle (MitM) attacks, the attacker divides the cipher E_K in two parts, $E_K = E_K^1 \circ E_K^0$, and recovers a known plaintext/ciphertext pair $P, C = E_K(P)$. The equality $(E_K^1)^{-1}(C) = E_K^0(P)$ holds. In its simplest form, the MitM applies when E_K^0 does not depend on the full k-bit key K but rather on $t < k$ key bits, and similarly for E_K^1. In this case, the attacker can iterate over all t key bits allowing to compute $E_K^0(P)$, and store the different values of $E_K^0(P)$ encountered. Then he exhaustively tries all t key bits allowing to compute $(E_K^1)^{-1}(C)$ and looks for collisions between $E_K^0(P)$ and $(E_K^1)^{-1}(C)$, among which the right key will be found. This basic attack requires 2^t key guesses, instead of 2^k, to recover the key. It was first applied by Diffie and Hellman as an attack in roughly 2^{56} operations against double DES [DH77] with 112-bit key, and this led to the spread of triple DES with 168-bit key (and 112-bit security) to replace single DES.

2.3 The Block Cipher AES

AES (previously Rjindael) [RD01] is a SPN cipher who won in 2001 the Advanced Encryption Standard competition launched by the National Institute of Standards and Technology (NIST) in 1997, and therefore became the AES. The AES block cipher has remained the most widely used block cipher for 20 years, and it is the cipher that received the most cryptanalysis in the literature. Three instances of the cipher exist, for key sizes of 128, 196 and 256 bits. They all share the same round function, with different key schedules and different numbers of rounds. AES-128 (resp. AES-192, AES-256) has $r = 10$ rounds (resp. $r = 12$, $r = 14$ rounds).

2.3.1 Description

The AES block cipher operates on a 128-bit state, represented as a 4×4-byte matrix. The bytes are ordered as follows:

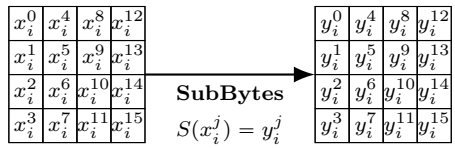
0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

If s is an AES state, the j-th byte of s is denoted s^j or $s[j]$.

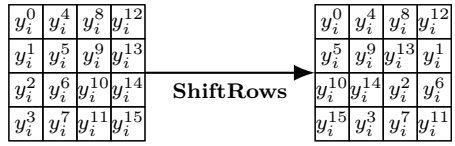
Key schedule of AES. As standard SPNs, the AES has a key schedule to produce $r + 1$ round keys of 128 bits. Since we do not exploit the key schedule of the AES throughout this thesis, we will not describe it here and consider that it generates independant subkeys in our attacks. The key schedule is non-linear, and its properties are mostly relevant in related-key attacks, which are in fact very powerful against AES [BK09; BKN09]. Some single-key attacks also take advantage of the key schedule properties, such as the key-bridging technique [DKS15]. More recently, an alternative representation of the AES key schedule has been described to explain some of its properties [LP21].

Round function of AES. The round function of the AES is composed of the following operations, for round $0 \leqslant i \leqslant r - 1$:

- **SubBytes (SB)**: The AES S-box is applied to each byte of the state. The properties of the AES S-box are discussed in Section 2.3.2.1. We denote x_i the state before SubBytes, and y_i the state after the SubBytes.



- **ShiftRows (SR)**: The second row is shifted by 1 byte to the left, the third row by 2 bytes, and the fourth row by 3 bytes. We denote z_i the state after SubBytes.



- **MixColumns (MC)**: Each column is multiplied by a Matrix MC (the multiplications are in $\mathbb{F}_{2^8}$), whose properties are discussed in Section 2.3.2.2. We denote w_i the state after MixColumns.

$$MC = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}, \qquad \begin{bmatrix} w_i^{4j} \\ w_i^{4j+1} \\ w_i^{4j+2} \\ w_i^{4j+3} \end{bmatrix} = MC \times \begin{bmatrix} z_i^{4j} \\ z_i^{4j+1} \\ z_i^{4j+2} \\ z_i^{4j+3} \end{bmatrix}.$$

- **AddRoundKey (AK)**: Each byte is XORed with a byte of the round key k_i, i.e. for all $j \in [\![0 \dots 15]\!]$, $x_{i+1}^j = w_i^j + k_i^j$.

There is one extra AddRoundKey operation before the first round, and the last round omits the MixColumns operation.

Decryption. The decryption of the AES is performed by inverting all the operations of AES, step by step. The inverse of the SubBytes step is a S-box layer with the inverse of the AES S-box. The ShiftRows is trivially invertible, and the MixColumns is invertible by multiplying each column by the inverse of MC:

$$MC^{-1} = \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix}.$$

Finally, the AddRoundKey is its own inverse.

2.3.2 Properties of the Components

2.3.2.1 AES S-box

The AES S-box S is a 8-bit S-box defined from the inverse mapping $x \mapsto x^{-1}$ in $\mathbb{F}_{2^8} \backslash \{0\}$, and from an affine permutation A in $\mathbb{F}_2^8$ (which is not affine in $\mathbb{F}_{2^8}$):

$$S(x) = \begin{cases} A(x^{-1}) & \text{if } x \neq 0, \\ A(0) & \text{else.} \end{cases}$$

This construction offers advantageous properties for a cryptographic S-box. First, its differential uniformity is 4:

$$\mathrm{DU}_S = 4.$$

Indeed, let us denote L the bijective linear part of A ($L(x) = A(x) - A(0)$). For non-zero differences $\Delta_{\text{in}}, \Delta_{\text{out}}$, the equation $S(x) + S(x + \Delta_{\text{in}}) = \Delta_{\text{out}}$ can be re-written, for $x \notin \{0, \Delta_{\text{in}}\}$ as:

$$\begin{aligned} &x^{-1} + (x + \Delta_{\text{in}})^{-1} = L^{-1}(\Delta_{\text{out}}), && \text{or equivalently,} \\ &(x + \Delta_{\text{in}}) + x = L^{-1}(\Delta_{\text{out}}) \times x(x + \Delta_{\text{in}}), \end{aligned}$$

where the multiplications are performed in $\mathbb{F}_{2^8}$. This yields an equation of degree exactly 2 in x (since $L(\Delta_{\text{out}}) \neq 0$), which has at most 2 solutions. In the worst case, $x = 0$ and $x = \Delta_{\text{in}}$ are also solution to $S(x) + S(x + \Delta_{\text{in}}) = \Delta_{\text{out}}$, which makes 4 solutions at most. This furthermore shows that for any non-zero difference Δ_{in} (resp. Δ_{out}), there exists at most single Δ_{out} (resp. Δ_{in}) such that $\mathrm{DDT}_S[\Delta_{\text{in}}, \Delta_{\text{out}}] = 4$ (it is verified experimentally that such a Δ_{out} (resp. Δ_{in}) always exists).

Therefore, the optimal differential transition through the AES S-box happens with probability $4 \times 2^{-8} = 2^{-6}$, and the possible differential transition probabilities are $\{0, 2^{-7}, 2^{-6}\}$. The inverse of the AES S-box has the same properties.

In addition, the AES S-box has good linear properties, as the linear correlation between the inputs and outputs of S does not exceed 2^{-3}. This mitigates linear attacks presented in Section 2.2.3.1.

2.3.2.2 AES MixColumns

The AES MixColumns matrix MC was chosen such that it is Maximal Distance Separable (MDS): for all $\alpha \in \mathbb{F}_{2^8}^4$, either $\alpha = (0, 0, 0, 0)$ or the number of non-zero bytes among $\alpha \| MC(\alpha)$ is at least 5. The MDS property is a good security property against differential cryptanalysis, as it guarantees that a single-byte difference in input is propagated to a difference in the 4 output bytes. Similarly, this is a good security property to prevent linear attacks. This property was chosen following the wide-trail strategy, a design strategy adopted by Daemen and Rijmen [DR01] to guarantee that differential or linear trail with a low number of active S-boxes do not exist after iterating a few rounds of AES.

2.3.2.3 Super S-box

Daemen and Rijmen analysed the differentials on 2-round AES in 2006 [DR06], and introduce Super boxes. They presented a new way to represent the AES, by concatenating two rounds of AES, and considering 32-bit key-dependant S-boxes. Later, Gilbert and Peyrin used Super boxes to improve the different cryptanalysis of AES-like permutations [GP10], which they call Super S-boxes. We use the notation Super S-box in this thesis.

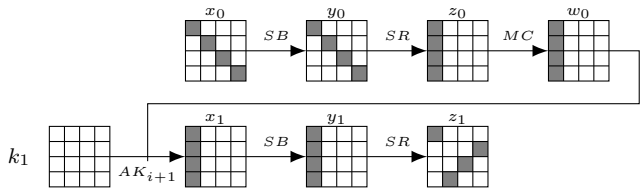


Figure 2.8: A Super S-box on 2-round AES.

Two rounds of AES are composed of the following operations:

$$MC \circ SR \circ SB \circ AK_{i+1} \circ MC \circ SR \circ SB \circ AK_i.$$

Because the ShiftRows and SubBytes operations commute, Daemen and Rijmen rewrote the 2-round AES as:

$$MC \circ SR \circ \underbrace{SB \circ AK_{i+1} \circ MC \circ SB}_{\text{Super S-box layer}} \circ SR \circ AK_i.$$

The middle part $SB \circ AK_{i+1} \circ MC \circ SB$ operates independantly on each 32-bit column, and is considered as 4 parallel applications of a 32-bit Super S-box. The ShiftRows operations before and after are sometimes included in the Super S-box layer, in which case a Super S-box maps a diagonal to an anti-diagonal, as depicted in Figure 2.8.

Unlike traditional S-boxes, the AES Super S-box is dependant on the key, and the notion of difference distribution table needs to be averaged on the keys. Furthermore, the inputs and outputs (32 bits) are too large to compute the fixed-key DDT in practice: this would require 2^{64} operations. On the other hand, it is easy to estimate the probability of Super S-box transitions for truncated differentials.

The Super S-box modelization of AES is useful when mounting generic attacks against several rounds of SPN ciphers, such as the yoyo attack [RBH17].

2.3.3 Security

AES has been the subject to a considerable amount of cryptanalysis. We describe some basic blocks that will be used in this thesis, and we present the complexity of the best existing attacks against AES in the literature.

2.3.3.1 Security against differential cryptanalysis

Because of the low differential uniformity of the AES S-box and the MDS property of the AES MixColumns, the attacker does not have a lot of freedom when choosing a differential trail on AES. Daemen and Rijmen proved minimal bounds on the number of active S-boxes in any active differential trail for less than 4 rounds of AES [DR02]. Later, Mouha *et al.* used automatic tools to give a tight lower bound on the minimal number of active S-boxes for a greater number of rounds [MWG+11]. The proven bounds are:

Number of AES rounds	1	2	3	4	5	6	7	8
Minimal number of active S-boxes	1	5	9	25	26	30	34	50

The probability of a single differential trail can be estimated by multiplying the probabilities of each round differential transition assuming independant subkeys. Moreover, this probability is bounded by the highest probability of a differential transition through a single S-box, 2^{-6} in the case of the AES, raised to the power of the number of active S-boxes. Therefore, it is proven that the best differential trail has a probability at most $2^{-6\times 25} = 2^{-150}$ on 4 rounds, with independant subkeys. However, due to the strong alignment of the AES, truncated trails gather many good differential trails. The truncated trail of Figure 2.9 (working both for forward or backward 3-round AES) reaches a probability of 2^{-24} (instead of the bound of 2^{-54} for the best differential trail). The forward probability is indeed the probability that given any input active in the main diagonal of x_0, the difference cancels on the second, third and fourth bytes of state w_0 (probability 2^{-8} for each).

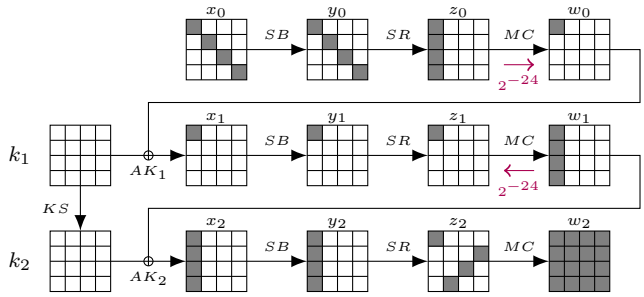


Figure 2.9: Example of a truncated differential trail on 3-round AES.

2.3.3.2 State-of-the-art attacks against AES

Table 2.1 sums up the best cryptanalytic results on AES-128 in the single-key model. In particular, it should be noted that the best attacks in the single key model reach 7 out of 10 rounds of AES-128.

Some attacks against AES have been defined in attacker models that differ from standard attacker models against block ciphers. Key-recovery attacks against AES with secret S-boxes were defined by Tiessen *et al.* [TKK+15]; the goal is to recover the secret key and the secret S-box used in an AES where the AES S-box is unkown by the attacker. Structural distinguishers were defined by Grassi *et al.* [GRR17] as distinguishing attacks that are "independant of the secret key". In particular, those distinguishers are typically applicable with unkown S-boxes The best attacks in these two settings are given in Section 4.1.

Rounds	Attack type	Model	Data	Time	Mem	Ref
5	Retracing Boomerang	ACC	2^{9}	2^{23}	2^{9}	[DKR+20]
6	Square	CP	2^{35}	2^{45}	2^{32}	[FKL+01]
	Square	CP	2^{33}	2^{40}	2^{32}	[DGK+24]
	Truncated Boomerang	ACC	2^{59}	2^{61}	2^{59}	This thesis
	Boomerang	ACC	2^{51}	2^{68}	2^{32}	This thesis
	Boomerang	ACC	2^{51}	2^{66}	2^{42}	This thesis
	Boomerang	ACC	2^{57}	2^{61}	2^{33}	This thesis
7	Meet-in-the-middle	CP	2^{97}	2^{99}	2^{98}	[DFJ13]
	Impossible Differential	CP	2^{105}	2^{113}	2^{74}	[BLN+18]
	Impossible Differential	CP	2^{105}	2^{111}	2^{72}	[LP21]
	Related Differential	CP	2^{110}	2^{110}	2^{110}	[BR22a]

Table 2.1: Best key-recovery attacks on reduced-round AES.
CP: chosen plaintexts / ACC: chosen plaintexts and adaptively-chosen ciphertexts.

2.3.4 AES-based Constructions

Since its standardization, the AES has deeply influenced the design of symmetric-key cryptographic primitives. This trend even accelerated after the introduction in modern CPUs of AES-NI [Gue08], a set of dedicated hardware accelerated instructions implementing the AES encryption and decryption. This directly improves modes using AES [MV04; RBB03; KR21b]. Yet, since AES-NI granularity lies at the round level, many new cryptographic designs actually use the AES round function as a building block, either for hash functions [BBG+08; IAC+08; BD08; GK08], for authenticated encryption schemes [WP14; Nik14; JNP+21; SLN+21; NFI24], for permutations [IIL+23; GM16; KLM+16; BLL+22], or for collision resistant building blocks [JN16; Nik17], among other applications.

We now present some AES tweakable variants that we study in this thesis: Kiasu-BC [JNP14b] and Deoxys-BC [JNP+21]. These constructions are part of the TWEAKEY framework.

2.3.4.1 The TWEAKEY framework

The TWEAKEY framework was introduced by Jean *et al.* [JNP14b] to give a general construction for tweakable block cipher designs. In the TWEAKEY framework, the key and the tweak form a master tweakey TK of the tweakeable block cipher. The block cipher is then built with the iterated round function strategy; the tweakey goes through a tweakey schedule, and the subkey addition of each round is replaced with a subtweakey addition. The TWEAKEY construction is depicted in Figure 2.10. The framework specifies a tweakey schedule construction: the master tweakey TK gooes iteratively through a tweakey state update function h, to produce the round tweakey states TK_j. The round tweakeys tk_j are extracted from the round tweakey state TK_j with a function g.

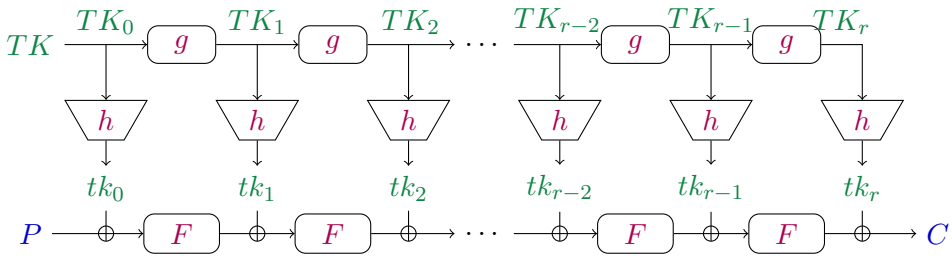


Figure 2.10: The TWEAKEY construction for tweakable block ciphers.

The STK construction. Along with this generic constructions, the designers proposed STK, a subclass of the TWEAKEY framework where "the tweak and the key are treated almost the same way". The tweakey TK is composed of a concatenation of the key and the tweak of $k+t$ bits (where $k+t$ is supposed to be a multiple of n), and is represented as p states of n bits $TK^1 \dots TK^p$. The STK construction assumes that the cipher has a ℓ-bit S-box operating in parallel on $m = n/\ell$ nibbles. The round tweakey states TK_i are iteratively updated from the master tweakey as follows:

Let r denote the number of rounds of the cipher and let h' denote a nibble permutation. The round tweakey states are initialized with $TK_0^i = TK^i$ for $i \in [\![1, p]\!]$ and for all $j \in [\![0, r-1]\!]$:

$$\begin{cases} TK_{j+1}^1 = h'(\mathsf{LFSR}_1(TK_j^1)) \\ TK_{j+1}^2 = h'(\mathsf{LFSR}_2(TK_j^2)) \\ \vdots \\ TK_{j+1}^p = h'(\mathsf{LFSR}_p(TK_j^p)), \end{cases}$$

where the functions LFSR_i are defined from elements α_i of $\mathbb{F}_{2^\ell} \backslash \{0\}$ and operate independantly on the m nibbles of the state as:

$$\forall i \in [\![1, p]\!], \mathsf{LFSR}_i(X_0 \| \dots \| X_{m-1}) = (\alpha_i \times X_0 \| \dots \alpha_i \times X_{m-1}),$$

the multiplications being performed in $\mathbb{F}_{2^\ell}$. LFSR_i are based on Linear Feedback Shift Registers (LFSR) and are linear. The function g is the sum of all round tweakey states:

$$tk_j = \sum_{i=1}^{p} TK_j^i.$$

Two widely studied ciphers, Deoxys-BC [JNP+21] and Skinny [BJK+16], follow the STK construction. Ciphers designed with the STK construction are often studied in the related tweakey model (RTK). In this model, the attacker can inject chosen differences in the master tweakey, with no special consideration of whether it corresponds to the tweak or to the key part. In particular, the RTK model is a related key (RK) model. Variants of the RTK model exist, named RTK_i, where

the attacker can inject chosen differences only in i out of the p master tweakey states. If $i \leqslant t/n$, this is a single key model, with chosen tweaks. The RTK_i model can be analyzed with mathematical properties, and the construction ensures that a difference in the tweakey is reinjected regularly into the cipher, thus limiting the differential trails with a large number of inactive rounds.

2.3.4.2 Deoxys-BC

Deoxys-BC [JNP+21] is a tweakable block cipher proposed by Jean *et al.* following the STK construction, which was selected in the CAESAR portfolio. Deoxys-BC re-uses the round function of the AES, and it has two variants: Deoxys-BC-256 with a 256-bit tweakey and 14 rounds, and Deoxys-BC-384 with a 384-bit tweakey and 16 rounds. The tweakey material is composed of a variable length key and tweak summing to 256 or 384 bits; for simplicity, we assume that the key length is a multiple of 128. The tweakey material is divided in words of 128 bits, denoted TK^i. LFSR_1 is defined as the identity. The others LFSRs are chosen so that in the RTK_i model, for each nibble u, the tweakey nibbles $\{h'^{-j}(tk_j)[u] \text{ for } j \in [\![0, r]\!]\}$ are either all inactive, or inactive at most $i-1$ times every 15 rounds.

2.3.4.3 Kiasu-BC

Kiasu-BC is a cipher presented by Jean *et al.* [JNP14a], as part of the TWEAKEY framework, with a 128-bit key and 64-bit tweak. The round function of Kiasu-BC is the round function of the AES, and the key-schedule of Kiasu-BC is very similar to that of the AES, with a tweak addition: the round tweakeys are computed as $tk_i = k_i + t$ where k_i is the round key following the AES key schedule, and t is the tweak T encoded in the first two rows. In particular, Kiasu-BC with $T = 0$ is the AES-128.

2.3.4.4 TNT-AES

TNT-AES is a tweakable block cipher reusing the AES round function published at EUROCRYPT 2020 [BGG+20]. It is part of the Tweak-aNd-Tweak framework, building a tweakable block cipher $\tilde{E}$ from a block cipher E:

$$\tilde{E}_{K_0,K_1,K_2} : P, T \mapsto C = E_{K_2}\Big(T + E_{K_1}\big(T + E_{K_0}(P)\big)\Big).$$

In order to improve its efficiency, TNT-AES uses a 6-round AES as building block E. The designers of TNT proved its security up to $2^{2n/3}$ queries, and conjectured a higher security bound. Later work [GGL+20] proved the bound to be at least $\Omega(2^{3n/4})$ queries, and exhibited a distinguisher with $\mathcal{O}(\sqrt{n} \cdot 2^{3n/4})$ queries. However, a recent work from Jha *et al.* [JKN+24] showed that these proofs were flawed and exhibited an attack in $\mathcal{O}(2^{n/2})$. In particular, TNT-AES is vulnerable to an attack with complexity 2^{69}.

2.4 MILP: an Automatic Tool for Cryptography

One of the most groundbreaking changes in cryptanalysis in the past decade is the systematisation of the usage of automated tools. Automated tools thrive because they can find cryptographic characteristics (linear, differential, boomerang, differential-linear,impossible differential ...) that would be tedious to compute by hand. They are great companions to the cryptanalyst since they can be used to improve existing attacks. They are at the same time very helpful for designers, since they can often be used to derive bounds of the complexity of attacks whithin a framework. Bouillaguet *et al.* [BFL11] first successfully used an Integer Linear Programming (ILP) tool to find bounds on the probability of differential trails on SIMD. One year later, Mouha *et al.* [MWG+11] generalised the technique to search for good differential or linear trails on multiple ciphers, using Mixed Integer Linear Programming (MILP). Since then, other automated tools have been experimented, such as SAT solvers [MP13], or Constraint Programming (CP) [MSR14]. Multiple recent works have been conducted to compare the efficiencies of the different tools for the search of differential trails on several primitives [DDH+20; BPF+23]. Although these works tend to show that CP and SAT modeling can offer good performances on bit-oriented ciphers, we focus on AES-based primitives at the byte-level, and choose MILP as a first-choice automated tool in this thesis.

Overall, MILP has been used for the search of many types of characteristics: differentials characteristics [FWG+16; AST+17; ZDY19; ZZD+19], linear characteristics [FWG+16; ZZD+19], differential-linear trails [BGG+23; HDE24], impossible differentials [LKH+16; ZD19; LXC+23], meet-in-the-middle attacks [Sas18b; SSD+18] or boomerang characteristics [ZDJ19; DDV20; QDW+21]. These works are by no mean bound to be exhaustive, but they give an overview on the importance of MILP modeling in cryptanalysis.

2.4.1 Description of a MILP model

A MILP model is composed of three types of objects:

- *variables*, representing either real numbers or integers[1].
- *constraints*, defined as inequalities between affine combinations of variables.
- *An objective function* which is a linear combination of variables that needs to be maximized (or minimized) when subjected to the given constraints[2].

A MILP solver takes as input a MILP model and returns, if it exists, *values* for the variables that both satisfy the constraints and maximizes (or minimizes) the objective function. Many MILP solvers exist, and Bellini *et al.* [BPF+23]

[1] "Mixed" in MILP actually highlights the different natures of variables. In Integer Linear Programming (ILP), the variables are only integers.

[2] In the Gurobi MILP solver [Gur23], this objective function can be quadratic.

compared the efficiency of GLPK [Oki12] and Gurobi [Gur23], out of which Gurobi had better performance. In this thesis, we use the MILP solver Gurobi.

In the literature, several strategies exist to find good differential trails on SPN with MILP. The most generic method is to set a binary MILP variable for each bit of each state. The MILP variable representing a bit equals 1 if and only if this bit is active in the differential trail. Each S-box is modelized with a set of inequalities on the input/output bits, encoding the DDT of the Sbox. Multiple works have been conducted to find efficient MILP modelings of S-boxes [AST+17; BC20]. Then, the linear layer is modelized with a set of linear equalities, and some MILP variables can even be removed because of redundancy, though Gurobi does it automatically in its pre-solving step. The objective function to maximize is the sum of all DDT values (taken in logarithm scale) of each S-box transition. In addition, an inequality is set to force at least one variable to be active, in order to avoid the trivial trail fully inactive. The set of variables maximizing the objective function of the model represents a differential trail that has an maximal probability among all existing trails. This approach is generic, but involves a lot of variables and of constraints, because of the heavy S-box modeling.

Another strategy for word-aligned ciphers, such as AES, is to create a variable for each word of each state, instead of a variable for each bit. This reduces significantly the number of variables and constraints of the model, thus reducing the solver time by a big factor. The MILP variable corresponding to a state word equals 1 if the word is active in the differential trail, but the exact value of the difference is not fixed. The S-box transition is simple: the input word is active if and only if the output word is active. The linear layer is then modelized with a set of constraints depending on its structure. Similarly to the first method, an inequality is set to force at least one variable to be active. The objective function to minimize is the number of active words in input of the S-box; it is a sum of all binary variables representating the state words before the S-box layer. Unlike the bit-oriented approach, this approach minimizes the number of active S-boxes in the trail, but does not guarantee that the differential trail can be instantiated with concrete differences satisfying the DDT transition of the S-box. However, if S is the ℓ-bit S-box of the SPN, and DU_S is its differential uniformity, then the maximal transition probability around an active S-box is $\mathrm{DU}_S \times 2^{-\ell}$. Therefore, if t is a lower bound on the number of active S-boxes in all differential trails, it ensures that the best differential trail has a probability of at most $(\mathrm{DU}_S \times 2^{-\ell})^t$ under the Markov cipher assumption; this allows designers to give convincing arguments relating to the protection of a cipher against differential attacks.

2.4.2 Example on AES

We now describe a MILP model, aiming at finding the minimal number of active S-boxes in a differential trail on r-round AES. As we only want to minimize the number of active S-boxes, we use the the byte-oriented approach. In the following model, we set a variable for each byte, and do not care for the redundancy of the variables; the variables can be removed subsequently if redundant.

Variables. We create a binary variable for each of the 16 bytes of each internal state of r rounds. In the following, $i \in [\![1, r]\!]$ and $j \in [\![0, 15]\!]$:

- x_j^i represents the j-th byte of the state before SubBytes of the i-th round.
- y_j^i represents the j-th byte of the state before ShiftRows of the i-th round.
- z_j^i represents the j-th byte of the state before MixColumns of the i-th round.
- w_j^i represents the j-th byte of the state after MixColumns of the i-th round.

Constraints. Now we modelize the operations with MILP constraints:

- SubBytes: the input of a S-box is active if and only if the output is inactive, therefore:
$$\forall i \in [\![1, r]\!], j \in [\![0, 15]\!], \qquad y_j^i = x_j^i.$$
- ShiftRows: the bytes in row k are shifted by k columns to the left:
$$\forall i \in [\![1, r]\!], j \in [\![0, 3]\!], k \in [\![0, 3]\!], \qquad z_{4j+k}^i = y_{4((j+k) \bmod 4)+k}^i.$$
- MixColumns: because of the MDS property of the AES MC matrix, we are ensured that either a column is inactive, or at least 5 bytes among the 8 input/output bytes are active. To encode this relation, we add a binary variable α_c^i for $c \in [\![0, 3]\!]$ and $i \in [\![1, r]\!]$ representing the activity of the column c of the MixColumns of round i: $\alpha_c^i = 1$ if and only if the column c has at least one active difference in the input or output of the MixColumns of round i. This is encoded by:
$$\forall i \in [\![1, r]\!], c \in [\![0, 3]\!], \qquad 8\alpha_c^i \geqslant \sum_{j=0}^{3} (z_{4c+j} + w_{4c+j}).$$
In addition, if $\alpha_c^i = 1$, we want to ensure that at least 5 input/output bytes are active. This is encoded by:
$$\forall i \in [\![1, r]\!], c \in [\![0, 3]\!], \qquad 5\alpha_c^i \leqslant \sum_{j=0}^{3} (z_{4c+j} + w_{4c+j}).$$
- AddRoundKey: the key addition does not change the activity pattern, therefore we add the constraints:
$$\forall i \in [\![1, r-1]\!], j \in [\![0, 15]\!], \qquad w_j^i = x_j^{i+1}.$$

In addition, to remove the trivial fully inactive differential trail, we add the constraint that at least one byte is active in the differential trail. For AES in the single-key setting, this is equivalent to asking that the initial state is not fully-inactive, therefore we add the following constraint:

$$\sum_{j=0}^{15} x_j^0 \geqslant 1.$$

Objective function. The objective function to minimize is the number of active S-boxes. The activity of the S-box inputs are represented with the variables x_j^i, therefore the objective function simply is:

$$\text{Obj} = \sum_{i=1}^{r} \sum_{j=0}^{15} x_j^i.$$

If this MILP model is solved, the optimal value of the objective corresponds to the minimal number of active S-boxes in a differential trail on r-round AES.

Part II
Contributions

Chapter 3
Improved Attacks against the Forkcipher Framework

The forkcipher framework was introduced by Andreeva *et al.* [ALP+19b] to provide efficient authenticated encryption for short messages. Two dedicated ciphers were proposed in this framework: ForkAES [ARV+18] based on the AES (more precisely on its tweakable variant Kiasu-BC [JNP14a]), and ForkSkinny based on Skinny [BJK+16]. Multiple instances of ForkSkinny were part of ForkAE [ALP+19a], a second round candidate of the NIST lightweight competition[1].

In this chapter, we present a joint work with Nicolas David and Gaëtan Leurent, "Cryptanalysis of Forkciphers", published in *IACR Transactions on Symmetric Cryptoglogy* (ToSC) [BDL20]. Notably, we show an attack on full ForkAES and we show how the best attacks on Skinny can be extended by several rounds on ForkSkinny by exploiting tweakey schedule modifications. The attacks on ForkSkinny have been further improved recently [HGS+24], and the same idea has been used to improve rectangle attacks [QDW+21; DQS+22].

Contents

[1] https://csrc.nist.gov/projects/lightweight-cryptography

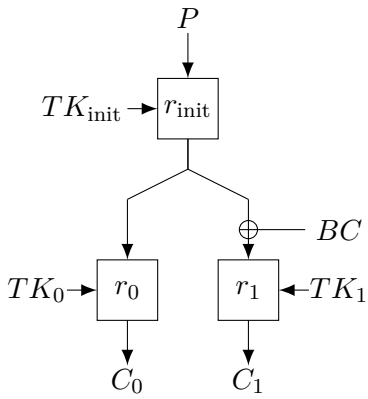


Figure 3.1: Illustration of an encryption by a forkcipher. A box represents several rounds of a block cipher round function.

3.1 Description of Forkciphers

3.1.1 The Forkcipher Framework

Definition 3.1. A **forkcipher** is a variant of a tweakable block cipher that takes one n-bit plaintext as input and outputs two n-bit ciphertexts:

$$\widetilde{F} : \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^n$$
$$K, T, P \mapsto C_0, C_1.$$

Each forkcipher instance is based on an underlying tweakable block cipher. To this day, the two proposed instances are ForkAES [ARV+18], based on an AES tweakable variant Kiasu-BC [JNP14a], and ForkSkinny, based on Skinny [BJK+16]. The forkcipher framework was designed on the *Iterate-Fork-Iterate* (IFI) paradigm: the underlying tweakable block cipher round-function is iterated r_{init} times in an initialisation phase, then forked, then iterated r_0 and r_1 respectively in each branch to produce two ciphertexts, as described in Figure 3.1. Right after the fork, a branch constant BC is added to the second branch. The tweakey material used in the initialisation phase and in both branches differs. The underlying block cipher tweakey schedule is iterated to produce a sequence of subtweakeys, interpreted as $TK_{\text{init}} \| TK_0 \| TK_1$.

The forkcipher can be used directly for authenticated encryption of short messages of less than one block, without needing a mode of operation (using the nonce as tweak). The cost to process the message will be roughly 1.5 block cipher calls, while block cipher-based modes require at least 2 block cipher calls.

Security Analysis. For each key and tweak, the functions from the input P to each half of the output ($P \mapsto C_0$ and $P \mapsto C_1$) should be permutations, and the corresponding families should be secure tweakable block ciphers (See Figure 3.1). In addition to encryption queries ($P \mapsto C_0 \| C_1$) and decryption queries ($C_i \mapsto P$), the security model of forkciphers also considers so-called *reconstruction queries*, which take C_0 as input and output C_1 or vice-versa.

In ForkAES and ForkSkinny, the numbers of rounds are chosen such that $r_0 = r_1 \geqslant \frac{r_{\text{tot}}}{2}$ and $r_{\text{init}} + r_0 = r_{\text{tot}}$, where r_{tot} is the number of rounds of the underlying block cipher. This ensures that all encryption and reconstruction queries go through at least r_{tot} rounds. In particular, encryption queries $P \mapsto C_0$ correspond exactly to an encryption query of the underlying block cipher.

Notation. For cryptanalysis, there are multiple ways to derive reduced versions of a Forkcipher. In the general case, we say that ForkAES-*a*-*b*-*c* (resp. ForkSkinny-*a*-*b*-*c*) corresponds to the version of ForkAES (resp. ForkSkinny) with a rounds before the forking point, b rounds in the first branch and c rounds in the second branch.

3.1.2 ForkAES

ForkAES is an instance of the forkcipher framework based on Kiasu-BC, an AES-based tweakable block cipher introduced in [JNP14a] and presented in Section 2.3.4.3. In the version of ForkAES presented by Andreeva *et al.* in 2018 [ARV+18], the numbers of rounds are the following:

$$r_{\text{init}} = r_0 = r_1 = 5.$$

It should be noted that Kiasu-BC is composed of 10 rounds, and that the best attack on Kiasu-BC reaches 8 rounds [BL23b].

The initialisation phase does not perform the AddRoundTweakey operation before the forking point, therefore uses r_{init} subtweakeys. The branch constant BC is set to 0. Each branch $i \in \{0, 1\}$ performs a subtweakey addition at the beginning (after the forking point) and at the end; TK_i is therefore composed of $r_i + 1$ subtweakeys. In total, $r_{\text{init}} + r_0 + r_1 + 2$ subtweakeys are computed using the Kiasu-BC tweakey schedule. We refer to [ARV+18] for implementation details.

3.1.3 ForkSkinny

ForkSkinny is an instance of the forkcipher framework based on Skinny [BJK+16].

Skinny. Skinny is an SPN cipher following the TWEAKEY framework [JNP14b], whose operations are optimized to reduce the hardware requirement. In particular, the S-box does not have optimal properties, the round keys are added to only half of the sate (without whitening keys in the first and last rounds), and the MixColumns operation does not use an MDS matrix (it only has branch number 2).

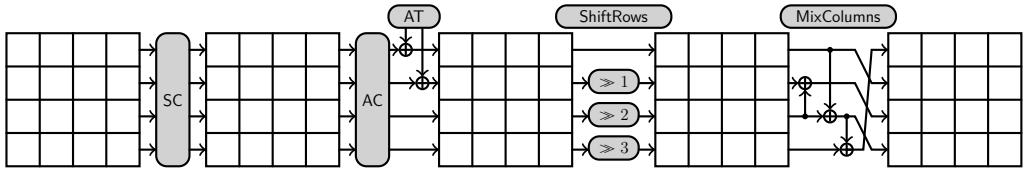


Figure 3.2: Skinny round function.

The are several variants of Skinny, with a state size of $n = 64$ or $n = 128$ bits, and a tweakey size of n bits, $2n$ bits, or $3n$ bits. The state is considered as a 4×4 matrix of cells (bytes or nibbles depending on the state size), and the round function (shown in Figure 3.2) follows roughly the AES structure:

- SubCells applies an S-box on each cell of the state;
- AddConstants adds round constants to the state;
- AddRoundTweakey adds tweakey material to the first two rows of the state;
- ShiftRows shifts the second row of the state by 1 cell, the third row by 2 cells, and the last row by 3 cells;
- MixColumns multiplies each column of the state by an invertible matrix.

Tweakey schedule. The tweakey schedule of Skinny is inspired by the STK construction of [JNP14b], but slightly differs from it by only adding the round tweakey to half of the state each round. The tweakey input (concatenation of the key and tweak) is divided into tweakey words of n bits, each word follows an independent schedule, and the subkeys are created by xoring elements from each word. For instance, with a tweakey size of $2n$ bits, the tweakey state has two words TK^1 and TK^2, and the subkeys are constructed from the values $TK^1 \oplus \alpha^i(TK^1)$, where α is a linear transformation implemented with an LFSR[2]. This limits the number of steps where tweakey differences can cancel out if the LFSR has a large order.

Since Skinny uses only $n/2$ bits of key material in each round, the value $\mathrm{TK1} \oplus \alpha^i(\mathrm{TK2})$ is used for the subtweakeys of rounds $2i$ and $2i+1$. Step $2i$ uses the first two rows of $\mathrm{TK1} \oplus \alpha^i(\mathrm{TK2})$ and permutes the cells, while step $2i+1$ uses the last two rows and permutes the cells.

Formally, the tweakey schedule for each chunk is defined as shown is Figure 3.3, where P_T is the following permutation, mapping the first two rows on the last two rows and vice-versa:

$$P_T = [9, 15, 8, 13, 10, 14, 12, 11, 0, 1, 2, 3, 4, 5, 6, 7].$$

The key schedule uses different LFSRs for each word (with the identity for TK1), but the same permutation P_T.

A version of Skinny with a n-bit state and a t-bit tweakey is denoted Skinny-n-t.

[2]With a tweakey of $3n$ bits, subkeys are constructed from the values $TK^1 \oplus \alpha^i(TK^2) \oplus \beta^i(TK^3)$.

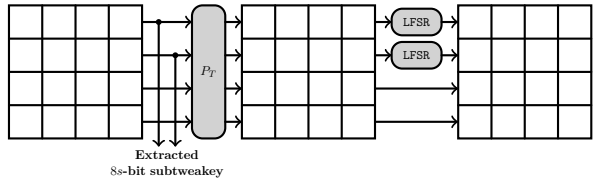


Figure 3.3: Skinny tweakey schedule.

Primitive	block	tweak	tweakey	r_{init}	r_0	r_1
ForkSkinny-64-192	64	64	192	17	23	23
ForkSkinny-128-192	128	64	192	21	27	27
ForkSkinny-128-256	128	128	256	21	27	27
ForkSkinny-128-288	128	128	288	25	31	31

Table 3.1: ForkSkinny parameters.

ForkSkinny. ForkSkinny was designed after the first attacks were published on ForkAES [BBJ+19], and uses $r_0 = r_1 > r_{\text{tot}}/2$ to limit the impact of the reduced diffusion in reconstruction queries. For instance, the original Skinny-128-256 possesses 48 rounds, but reconstruction queries against ForkSkinny-128-256 go through a total of 54 rounds: 27 decryption rounds and 27 encryption rounds. The parameters suggested by the designers are given in Table 3.1.

3.2 Cryptanalysis of full ForkAES

3.2.1 Results

Banik *et al.* [BBJ+19] showed an almost-practical truncated differential attack against ForkAES-$*$-4-4 with a time complexity of 2^{35} encryptions, using reconstruction queries following 8 rounds of Kiasu-BC (4 decryption rounds followed by 4 encryption rounds). In comparison, the best attack on 8-round Kiasu-BC, from [BL23b] and presented in Section 4.4.4, is a boomerang attack of complexity 2^{83}. This already shows that ForkAES is not as robust as the underlying cipher Kiasu-BC.

In this section, we extend the attack of Banik *et al.* from ForkAES-$*$-4-4 to the full ForkAES-$*$-5-5 rounds by introducing two new ideas. First, we consider weak keys which diffuse very poorly in the middle rounds. Second, instead of considering a single pair satisfying the truncated differential characteristic, we consider two related pairs simultaneously, such that if one pair follows the characteristic, then the second pair also follows it with high probability.

We give several attacks with varying weak key spaces, as shown in Table 3.2; for each attack the expected complexity is lower than $2^{127} \times \epsilon$, where ϵ corresponds to the fraction of weak keys among the full key space. In particular, the last attack

Algorithm	Attack Type	Rds.	Data	Time	Mem.	ϵ	Reference
AES-128	Square	6	2^{32}	2^{71}	2^{32}	1	[DKR97]
AES-128	Imp. Diff.	7	$2^{106.2}$	$2^{110.2}$	$2^{90.2}$	1	[MDR+10]
AES-128	MitM	7	2^{97}	2^{99}	2^{98}	1	[DFJ13]
Kiasu-BC	Square	7	$2^{48.5}$	$2^{43.6}$	$2^{41.7}$	1	[DEM16]
Kiasu-BC	MitM	8	2^{116}	2^{116}	2^{86}	1	[TAY16]
Kiasu-BC	Imp. Diff.	8	2^{118}	$2^{120.2}$	2^{102}	1	[DL17]
Kiasu-BC	Boomerang	8	2^{103}	2^{103}	2^{60}	1	[DL17]
ForkAES-$*$-4-4	Imp. Diff.	8	$2^{39.5}$	2^{47}	2^{35}	1	[BBJ+19]
ForkAES-$*$-4-4	Refl. Diff.	8	2^{35}	2^{35}	2^{33}	1	[BBJ+19]
ForkAES-$*$-5-5	Trunc. Diff.	10	$2^{74.5}$	$2^{74.5}$	$2^{59.5}$	2^{-32}	Section 3.2.3
ForkAES-$*$-5-5	Trunc. Diff.	10	$2^{100.5}$	2^{114}	$2^{80.5}$	2^{-4}	Section 3.2.4.2
ForkAES-$*$-5-5	Trunc. Diff.	10	2^{119}	2^{125}	2^{83}	1	Section 3.2.4.3

Table 3.2: Cryptanalysis results against AES, Kiasu-BC, and ForkAES. ϵ is the fraction of weak keys targeted by the attack.

targets the full keyspace with an expected time complexity of 2^{125} encryption.

Even though the attacks are not practical, these are the first that break the security claim of the full ForkAES. In particular, it shows that ForkAES offers a significantly lower security that the underlying block cipher Kiasu-BC, where the best attacks reach only 8 rounds.

Notation. The first branch takes round keys k_5 to k_{10} (resp. k_5 to k_9) and the second branch takes round keys k_{11} to k_{16} (resp. k_{10} to k_{14}) for ForkAES-5-5-5 (resp. ForkAES-5-4-4), derived from the master key k with an extended key schedule. We denote the ciphertext obtained from the first branch as C_0 and the ciphertext obtained from the second branch as C_1.

Equivalent last rounds. For the following attacks, we denote $\widehat{C}_0$ (resp. $\widehat{C}_1$) the state after partial inversion of the last round of the first (resp. second) branch (we invert AddRoundTweak, MixColumns and ShiftRows), and $\widehat{k}_i$ the equivalent round key corresponding to k_i, where k_i is typically a whitening key in the output of a branch. We point out that ForkAES does not omit the MixColumns operation in the last round. The equivalent operation is denoted by AK_i^{eq}. In other words,

$$\widehat{C}_0(T) = \mathsf{SR}^{-1}(\mathsf{MC}^{-1}(\mathsf{AT}(C_0))), \qquad \widehat{k}_i = \mathsf{SR}^{-1}(\mathsf{MC}^{-1}(k_i)),$$
$$\widehat{C}_1(T) = \mathsf{SR}^{-1}(\mathsf{MC}^{-1}(\mathsf{AT}(C_1)).$$

Although $\widehat{C}_i(T)$ depends on the tweak used for reconstruction, if there is no ambiguity, we allow ourselves to omit the tweak in the notation and denote the equivalent ciphertext $\widehat{C}_i$.

3.2.2 Previous Attack Against ForkAES-∗-4-4

We start by explaining the details of the previous reflection differential attack of Banik et al. [BBJ+19] on ForkAES-5-4-4, although it can trivially be extended to ForkAES-*-4-4 with a reindexing of subkeys. This attack is a truncated differential attack exploiting reconstruction query.

3.2.2.1 Truncated differential characteristic and probability

The attack uses the truncated differential characteristic of Figure 3.4. For each byte of the input, output, and internal state, the characteristic specifies whether there should be a difference or not. We approximate the probability of the characteristic as the product of the probability of each rounds, using the Markov Cipher Assumption. The attack has two phases:

1. We build a structure $\mathcal{S}$ on the first column of C_0 and the first byte of the tweak T: for each tweak T and ciphertext C_0 in the structure, the different $\widehat{C}_0(T)$ have the same values on the last three columns and differ only on the first one.

2. Among $\mathcal{S}$, look for pairs with the correct difference pattern in $\widehat{C}_1(T)$. If we identify a pair, we assume that the internal states follow the characteristic, and we use this assumption to recover part of the key.

We can compute the probability of the characteristic as follows:

- $\Pr(w_8 \to z_8) = 2^{-24}$ since three bytes become inactive after inverse MixColumns.
- $\Pr(x_8 \to w_7) = 2^{-8}$ since the difference in byte 0 cancels out the tweak difference.
- $\Pr(z_{10} \to w_{10}) = 2^{-24}$ since three bytes become inactive after MixColumns.
- $\Pr(w_{10} \to x_{11}) = 2^{-8}$ since the difference in byte 0 cancels out the tweak difference.

In total, the probability of the characteristic is 2^{-64}.

3.2.2.2 Attack procedure

While the description given in [BBJ+19] uses a fixed tweak difference, we will not fix it in advance, but we use structures on the tweak to further reduce the complexity of the attack. Moreover, we will show in Section 3.2.4.1 that fixing the tweak does not ensure the above probability of going through the characteristic, as the middle rounds can only be satisfied if the tweak difference and the key are compatible.

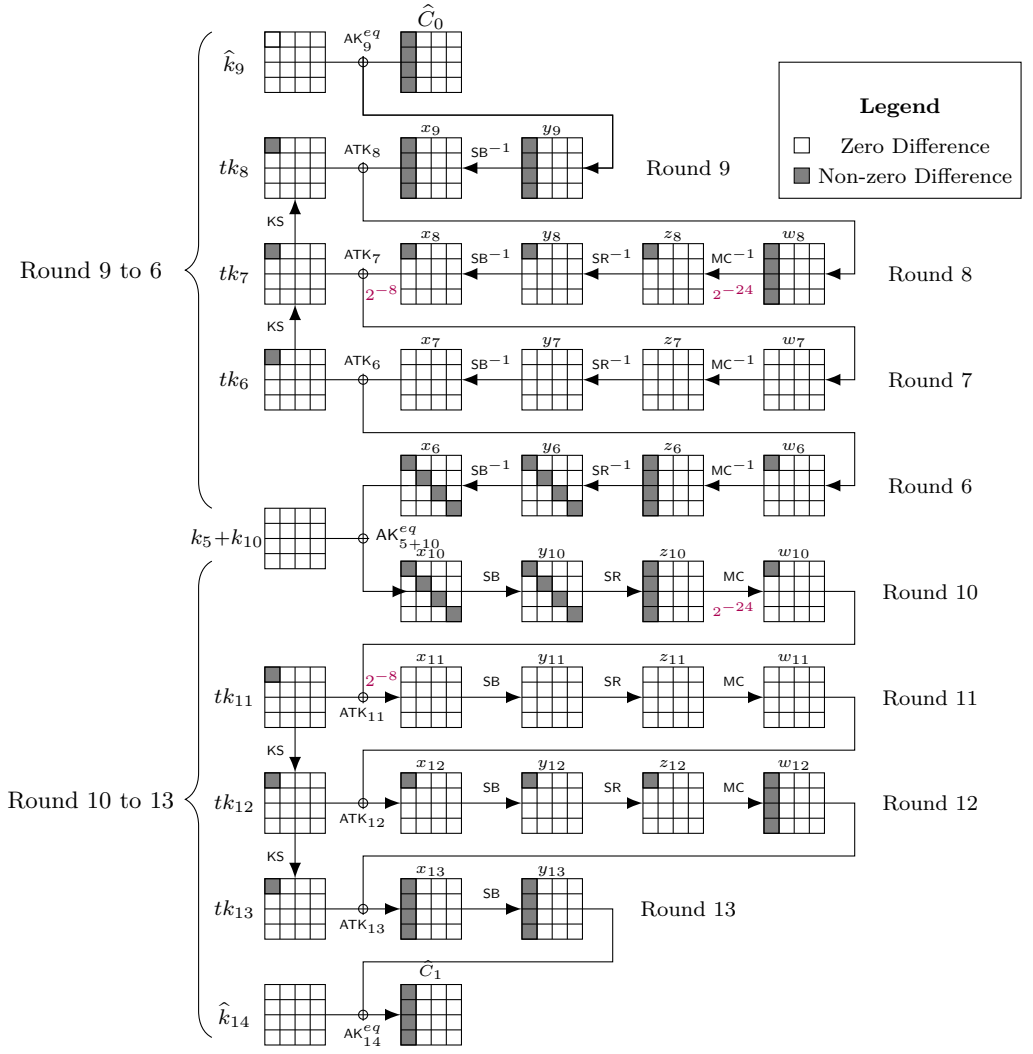


Figure 3.4: Truncated differential characteristic for ForkAES-5-4-4.

Construction of the structure. First, we choose arbitrary values for the last three columns of C_0, along with arbritrary values for all bytes of the tweak except the first one. Iterating over the first byte of the tweak and the values of the first column of C_0 gives us a set of 2^{40} (T, C_0) values. Among them, we query $2^{32.5}$ randomly chosen elements. This gives us $2^{32.5} \times (2^{32.5} - 1)/2 \approx 2^{64}$ pairs of inputs verifying the input difference pattern of the characteristic. On average, one pair satisfies the characteristic.

Filtering the pairs. We can distinguish the right pair from the wrong ones by storing the corresponding $\widehat{C}_1(T)$ ciphertexts into a hash table, indexed by the values of the three last columns. If a pair of inputs satisfies the characteristic, we detect it as a collision in the hash table. For random pairs of outputs, there is a

collision with probability 2^{-96}. As we have 2^{64} pairs stored in the hash table, a random collision occurs with probability 2^{-32}, which can be neglected. The only collision is therefore the right pair.

Reducing the key space. Supposing that the pair follows the characteristic, the difference in state x_{12} is exactly the tweak difference. As mentioned in Section 2.2.2.2, there are only 2^7 possible output differences through the S-box if the input difference is fixed. We can compute these differences from the known tweak difference, and propagate the differences through SR, MC and ATK_{13} since they are linear operations, in order to get 2^7 candidates for the x_{13} difference. We also know the difference in $\widehat{C}_1$, therefore the difference in y_{13}. Using Lemma 2.2, we recover the value of the first column of y_{13} for each of the 2^7 potential differences. Then we can deduce 2^7 candidates for the first column of the key $\widehat{k}_{14}$, from the value of y_{13} and $\widehat{C}_1$.

Full key-recovery. No operation depends specifically on the column position in AES, Kiasu-BC and ForkAES. Consequently, we can iterate this attack by shifting the entire characteristic to another column. We can therefore recover 2^7 candidates for each column of $\widehat{k}_{14}$. We can invert the key schedule to recover the master key from $\widehat{k}_{14}$. The 2^{28} candidates are ultimately tested exhaustively.

3.2.2.3 Complexity evaluation

We require $2^{32.5}$ queries per column, which induces a data complexity of $2^{34.5}$. The memory complexity is equivalent to $2^{32.5}$ AES states to store $2^{32.5}$ values of $\widehat{C}_1$ in the hash table. The time complexity is 2^{28} encryptions for the final exhaustive search and $2^{34.5}$ memory accesses for the data processing. Eventually the complexity is:

$$(D, T, M) = (2^{34.5}, 2^{34.5}, 2^{32.5}).$$

3.2.3 Attack Against Full ForkAES for 2^{96} Weak Keys

We now describe new attacks against ForkAES-$*$-5-5, by improving the previous techniques. Again, the key indexing corresponds to ForkAES-5-5-5 but the attack can trivially be adapted to ForkAES-$*$-5-5. A reconstruction query from C_0 to C_1 now starts with the whitening key k_{10}, goes through 5 inverse rounds with round keys k_9, k_8, k_7, k_6, then key $k_5 + k_{11}$ is xored at the forking point followed by 5 forward rounds with keys k_{12}, k_{13}, k_{14}, k_{15} and k_{16} (final whitening key).

Our first attack targets weak keys such that $k_5 + k_{11}$ only has zero values on the diagonal; this happens with probability 2^{-32}. Our attack uses a differential characteristic with high probability, shown in Figure 3.5. An important particularity of this characteristic is that if one pair satisfies this characteristic, then it is easy to construct another pair that has a very high chance of satisfying the characteristic as well.

Notation. To describe our attacks, we use θ to denote a non-zero byte difference such that $\Pr(\theta \xrightarrow[SB]{} \theta/2) = 2^{-7}$, and λ to denote the unique value such that $\Pr(\theta \xrightarrow[SB]{} \lambda) = 2^{-6}$. There are 123 values of θ satisfying this condition. We use Θ to denote the state difference active only on the first byte with $\Theta[0] = \theta$. Typically, Θ corresponds to the tweak difference in our attacks. For instance, we can use $\theta = \texttt{10}$, $\theta/2 = \texttt{08}$, $\lambda = \texttt{a9}$.

3.2.3.1 Differential characteristic for the first pair

First, we guess the first byte of the equivalent key $\widehat{k}_{10}$ and denote it K, and we consider input pairs $p = ((C_0, T), (C'_0, T'))$ of the form:

$$\widehat{C}_0 = \left(\begin{array}{c|c} S(0)+K & \\ 0 & \\ 0 & u \\ 0 & \end{array}\right), \qquad T = 0, \qquad \widehat{C}'_0 = \left(\begin{array}{c|c} S(\theta)+K & \\ 0 & \\ 0 & v \\ 0 & \end{array}\right), \qquad T' = \Theta.$$

for any 96-bit vectors u and v (seen as 4×3 byte matrices). By construction, the first byte difference at state x_{10} is exactly the tweak difference, so that the state w_9 has an inactive first column. Therefore, the probability of the characteristic can be computed as

- $\Pr(w_9 \to z_9) = 2^{-72}$ since 9 bytes become inactive after the MixColumns operation.
- $\Pr(w_8 \to z_8) = 2^{-24}$ since 3 bytes become inactive after the MixColumns operation. Moreover, since the difference in $w_8[0]$ is θ, cancelling three byte implies that the difference in the first column of w_8 must be $[\theta, \theta/2, \theta/2, \theta + \theta/2]$. Alternatively, we can consider that $\Pr(y_9 \to x_9) = 2^{-24}$ if the difference in the first column of x_9 is fixed in advance to $[0, \theta/2, \theta/2, \theta + \theta/2]$.
- $\Pr(y_8 \to x_8) = \Pr(\theta \xrightarrow[SB]{} \theta/2)$ because of the choice of θ.
- Round 7 is then inactive.
- Values in the diagonals of both elements of the pair in state x_6 do not change with the AK^{eq}_{5+11} operation, because of the hypothesis we made on the key. Consequently, both elements of the pair have the same diagonal values in state y_6 and in state y_{11}. The difference in state w_{11} is therefore the tweak difference with probability 1.
- Round 12 is then inactive.
- $\Pr(x_{13} \to y_{13}) = 2^{-7}$ because of the choice of θ.
- $\Pr(x_{15} \to y_{15}) = 2^{-6}$ because we are looking for pairs with a first byte difference of λ.

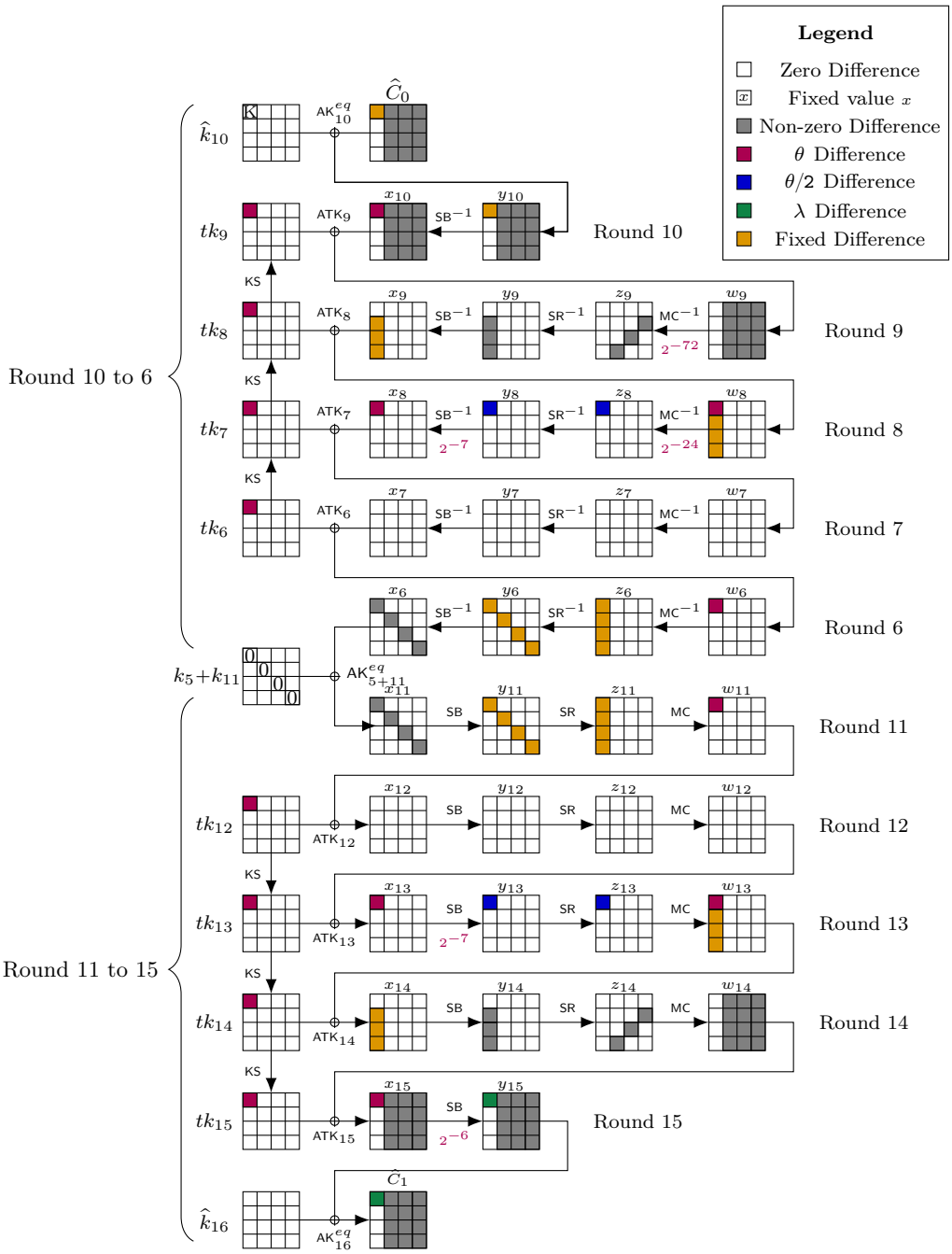


Figure 3.5: Differential characteristic for very weak keys.

In summary, for the first pair, $\Pr(\widehat{C}_0 \to \widehat{C}_1) = 2^{-72-24-7-7-6} = 2^{-116}$. Since a random pair has the prescribed output difference with probability 2^{-32}, this is too low for a direct attack, but we use an amplification technique using twin pairs.

3.2.3.2 Construction of twin pairs

We build a twin pair $((\overline{C_0}, \overline{T}), (\overline{C'_0}, \overline{T'}))$ from the pair $((C_0, T), (C'_0, T'))$ so that the twin pair follows the first three rounds of the characteristic with probability 1 assuming that the original pair follows it. The probability of the total characteristic for $((\overline{C_0}, \overline{T}), (\overline{C'_0}, \overline{T'}))$ will therefore be 2^{-13}:

$$\widehat{\overline{C_0}} = \left(\begin{array}{c|c} S(\tau[0]) + K & \\ 0 & \\ 0 & u \\ 0 & \end{array}\right), \quad \overline{T} = \tau, \quad \widehat{\overline{C'_0}} = \left(\begin{array}{c|c} S(\tau[0] + \theta) + K & \\ 0 & \\ 0 & v \\ 0 & \end{array}\right), \quad \overline{T'} = \tau + \Theta.$$

where τ is the tweak active only on the first byte, such that $\tau[0] = \theta/\mathtt{1c}$. By construction, the twin pair satisfies:

$$\begin{aligned} \overline{w}_9[0] &= \mathsf{ATK}_9(\mathsf{SB}^{-1}(\mathsf{AK}^{eq}_{10}(\mathsf{SB}(\tau[0] \quad\;\;) + K))) = 0 = w_9[0], \\ \overline{w}'_9[0] &= \mathsf{ATK}_9(\mathsf{SB}^{-1}(\mathsf{AK}^{eq}_{10}(\mathsf{SB}(\tau[0] + \theta) + K))) = 0 = w'_9[0]. \end{aligned}$$

Up to this point, neither MixColumns nor ShiftRows have been applied on the states, therefore each byte of the state has a specific value depending only on the corresponding byte of the input. This implies $\overline{w}_9 = w_9$ and $\overline{w}'_9 = w'_9$. This equality of states is satisfied for the whole round 9, up to x_9. After the next tweak addition, we have the following property:

$$\overline{w}_8 = w_8 + \tau, \qquad\qquad \overline{w}'_8 = w'_8 + \tau.$$

In particular, the differences of both pairs in state w_8 are equal. Because $w_8 \to y_8$ consists only in linear operations, the differences in state y_8 of both pairs are equal, in particular the difference $\overline{y}_8 + \overline{y}'_8$ is only active on the first byte, with difference $\theta/2$.

More precisely, we have:

$$\overline{y}_8 = y_8 + \mathsf{SR}^{-1}(\mathsf{MC}^{-1}(\tau)), \qquad\qquad \overline{y}'_8 = y'_8 + \mathsf{SR}^{-1}(\mathsf{MC}^{-1}(\tau)).$$

Note that τ was chosen so that the first byte of $\mathsf{SR}^{-1}(\mathsf{MC}^{-1}(\tau))$ is $\theta/2$. Therefore the pair $(\overline{y}_8[0], \overline{y}'_8[0])$ is the same as the pair $(y_8[0], y'_8[0])$ and follows the S-box transition with probability 1:

$$\overline{y}_8[0] = y_8[0] + \theta/2 = y'_8[0], \qquad\qquad \overline{y}'_8[0] = y'_8[0] + \theta/2 = y_8[0].$$

3.2.3.3 Filtering twin pairs

The main step of the attack is to build quartets of oracle queries corresponding to a pair and its twin, and to filter the candidates that follow the truncated characteristic. The differential characteristic for both pairs has a total probability of 2^{-129}, and the filter on the output is satisfied with probability 2^{-64} (32 bits of fixed difference in each pair).

We strengthen the filter using the fact that both pairs follow the characteristic with the same key: for each pair, we can deduce four candidates for $\widehat{k}_{16}[0]$ so that the transition $\theta \to \lambda$ is followed in the last round, and the two sets should have a common value. More precisely, we know that there are exactly four possible values for $y_{15}[0]$, corresponding to the four values following the S-box transition $\theta \to \lambda$; we denote them as a_0, a_1, a_2 and a_3. If both pairs go through the characteristic, the four values $\widehat{C}_1[0]$, $\widehat{C}'_1[0]$, $\widehat{\overline{C_1}}[0]$ and $\widehat{\overline{C'_1}}[0]$ must be in the same coset $\widehat{k}_{16} + \{a_0, a_1, a_2, a_3\}$. We can build a linear function φ that identifies the coset uniquely,[3] and we use an additional 6-bit filter $\varphi(\widehat{C}_1[0]) = \varphi(\widehat{\overline{C_1}}[0])$. Note that we already have $\varphi(\widehat{C}_1[0]) = \varphi(\widehat{C}'_1[0])$ and $\varphi(\widehat{\overline{C_1}}[0]) = \varphi(\widehat{\overline{C'_1}}[0])$ by definition if the pairs satisfy the output difference.

Random twin pairs pass the complete filter with probability 2^{-70}.

However, a good quartet satisfies much more restrictive conditions: the difference at the end of round 14 is in a space of 2^{24} output differences. In the following, we try to find efficiently good quartet.

3.2.3.4 Using structures

We now explain how to build structures on C_0 and identify candidates pairs efficiently. Following the previous notations, for 96-bit vectors u and v, we make four reconstruction queries:

$$\widehat{C}_0(u), 0 \mapsto \widehat{C}_1(u), \qquad \widehat{\overline{C_0}}(u), \tau \mapsto \widehat{\overline{C_1}}(u),$$
$$\widehat{C}'_0(v), \Theta \mapsto \widehat{C}'_1(v), \qquad \widehat{\overline{C'_0}}(v), \Theta + \tau \mapsto \widehat{\overline{C'_1}}(v).$$

and we want to identify u and v such that the output satisfy a 70-bit condition:

$$\widehat{C}_1(u)[0,1,2,3] = \widehat{C}'_1(v)[0,1,2,3] + [\lambda, 0, 0, 0], \tag{3.1}$$
$$\widehat{\overline{C_1}}(u)[0,1,2,3] = \widehat{\overline{C'_1}}(v)[0,1,2,3] + [\lambda, 0, 0, 0], \tag{3.2}$$
$$\varphi(\widehat{C}_1(u)[0]) = \varphi(\widehat{\overline{C_1}}(u)[0]). \tag{3.3}$$

The last condition depends only on u and we can filter out values that do not satisfy it. The three conditions imply another 6-bit condition that can be used to filter v values:

$$\varphi(\widehat{C}'_1(v)[0]) = \varphi(\widehat{\overline{C'_1}}(v)[0]). \tag{3.4}$$

[3] Such as the orthogonal projection with the dimension 2 kernel $\langle a_0 + a_1, a_0 + a_2, a_0 + a_3 \rangle$.

We store the remaining values in two hash tables:

- H indexed by $\widehat{C}_1(u)[0,1,2,3] \| \widehat{\overline{C_1}}(u)[0,1,2,3]$;
- H' indexed by $\widehat{C}'_1(v)[0,1,2,3] \| \widehat{\overline{C'_1}}(v)[0,1,2,3] + [\lambda,0,0,0,\lambda,0,0,0]$.

Let us assume that we have a match between an element of the first table and an element of the second one, corresponding to vectors u and v. Our twin pairs are $((\widehat{C}_0(u),0),(\widehat{C}'_0(v),\Theta))$ and $((\widehat{\overline{C_0}}(u),\tau),(\widehat{\overline{C'_0}}(v)),\Theta+\tau))$.

A match in the table ensures that conditions (3.1) and (3.2) are satisfied, and condition (3.3) was checked before filling the table.

Both hash tables have 64-bit hash keys. However, a pair (u,v) matches with probability 2^{-58} due to redundancy: conditions (3.1), (3.2) and (3.3) imply condition (3.4).

Therefore, we have a 70-bit filter implemented with a 6-bit pre-filter and two 64-bit hash tables. The filter has a memory complexity of only $2 \times D \times 2^{-6}$ 96-bit blocks if D is the number of candidate vectors.

3.2.3.5 Extended filtering

The probability that twin pairs satisfy the characteristic is 2^{-129}. Therefore, we need two sets of $2^{64.5}$ 96-bit vectors to form 2^{129} twin pairs in order to expect one right pair. After the 70-bit filter, we will still have $2^{129-70} = 2^{59}$ pairs, and we need to distinguish the right pair from the wrong ones.

The difference in z_{13} is $\theta/2$, and $z_{13} \to x_{14}$ is composed of linear operations, so we can compute the fixed difference in state x_{14}. Then, we precompute the 2^7 possible difference values of a byte difference in state y_{14}, for instance $y_{14}[1]$. We then compute the possible differences of the last column of the state x_{15}. Knowing the differences in $\widehat{C}_1$, we can deduce 2^7 possible values for the last column of $\widehat{k}_{16}$. We can perform this deduction for both twin pairs and check whether the intersection of the possible column values is empty or not. If it is, the pairs are incompatible. This costs 2^8 operations at most, and the probability of having a non-empty intersection is approximately $2^7 \times 2^7 \times 2^{-32} = 2^{-18}$. When this happens, there is on average one element in the intersection. We repeat the process with the two remaining columns, and obtain another filter, which a random pair passes with probability $2^{3\times(-18)} = 2^{-54}$.

Using precomputed tables. In the following sections, this filtering is the most expensive part of the attack, with a complexity of 2^8 elementary operations per candidates. Instead of computing the sets of $\widehat{k}_{16}$ candidates on the fly, we can build a table indexed by bytes of $\widehat{C}_1$, $\widehat{C}'_1$, $\widehat{\overline{C_1}}$ and $\widehat{\overline{C'_1}}$ that contains a boolean value indicating whether the pairs are compatible or not. This implementation of the filter requires a single table access instead of 2^8 operations.

More precisely, we can deduce 2^7 candidates for $\widehat{k}_{16}[12,13]$ from $\widehat{C}_1[12,13]$ and $\widehat{C}'_1[12,13]$. Therefore, we can check if the candidates for $\widehat{k}_{16}[12,13]$ have a non-empty intersection using only the values of $\widehat{C}_1[12,13]$, $\widehat{C}'_1[12,13]$, $\widehat{\widehat{C_1}}[12,13]$ and $\widehat{\widehat{C'_1}}[12,13]$. After pre-computing a table indexed by these 8 bytes (of size 2^{64}), we filter candidates with complexity 1, keeping only a fraction $2^{7+7-16} = 2^{-2}$. We can actually reduce the table size to 2^{48} because the valid values of $\widehat{C}_1[12,13]$, $\widehat{C}'_1[12,13]$, $\widehat{\widehat{C_1}}[12,13]$ and $\widehat{\widehat{C'_1}}[12,13]$ are invariant by translation. Therefore, we create a table indexed by $\widehat{\widehat{C_1}}[12,13] + \widehat{C}_1[12,13]$, $\widehat{\widehat{C'_1}}[12,13] + \widehat{C}'_1[12,13]$, $\widehat{C}_1[12,13] + \widehat{C}'_1[12,13]$.

We generate 4 such tables checking for intersection of different key bytes, in order to keep only a fraction 2^{-8} of the candidates, and then apply the full filter to the remaining candidates. Therefore, in the description of our attacks, we consider that this filter costs only one operation per candidate.

After this extended filter, we only have $2^{59-54} = 2^5$ twin pairs left. For each of them, there are on average one key guess for the three last columns and four key guesses for the first byte. We then iterate over the bytes $\widehat{k}_{16}[1,2,3]$ to proceed in an exhaustive search for the key. This has a complexity of $2^{5+2+24} = 2^{31}$ which is significantly lower than the complexities of other steps of this attack.

3.2.3.6 Complexity of the attack

First, we guess K of the first byte of the equivalent round key $\widehat{k}_{10}$. For each guess, we query two structures of size $2^{64.5}$, corresponding to $2^{66.5}$ encryptions, which makes a total data complexity of $2^{74.5}$. The memory complexity is $2^{59.5}$, because the average number of vectors written in each hash table is $2^{58.5}$. The time complexity of the pair processing of Section 3.2.3.5 corresponds 2^{59} simple operations for each guess of K, hence a total of $2^8 \times 2^{59} = 2^{67}$. Therefore, the time complexity is bounded by the data complexity.

Therefore, the complexity is:

$$(D, T, M) = (2^{74.5}, 2^{74.5}, 2^{59.5}).$$

3.2.4 Larger Classes of Weak Keys

We now describe attacks with larger classes of weak keys, by relaxing the constraint that $k_5 + k_{11}$ should have a diagonal of zeroes. However, without this constraint the path becomes impossible to satisfy for some tweak differences.

3.2.4.1 Incompatibility between the tweak difference and the key

We focus on the middle rounds of the previous characteristic, with a tweak difference Θ active only in the first cell. As explained earlier, we want to have rounds 7 and 12 inactive, so that the difference in y_6 and y_{11} is completely determined:

$$(y_{11} + y'_{11})[0,5,10,15] = \mathsf{MC}^{-1}([\theta,0,0,0]) = (y_6 + y'_6)[0,5,10,15]. \tag{3.5}$$

Moreover, we have the following relation between y_6 and y_{11}:

$$x_6[i] = x_{11}[i] + k_5[i] + k_{11}[i],$$
$$S^{-1}(y_6[i]) = S^{-1}(y_{11}[i]) + k_5[i] + k_{11}[i].$$

Therefore, for each $i \in \{0, 5, 10, 15\}$, the value of $y_{11}[i]$ must satisfy the following equation:

$$S(S^{-1}(t) + \kappa) = S(S^{-1}(t + \alpha) + \kappa) + \alpha, \tag{3.6}$$

with unknown t, where κ and α are parameters (corresponding respectively to $k_5[i] + k_{11}[i]$ and $\mathsf{MC}^{-1}([\theta, 0, 0, 0])[i/5]$). This equation admits solutions if and only if the coefficient (α, κ) of the Boomerang Connectivity Table of S^{-1} is non-zero [CHP+18]. Following the analysis of [BC18], there are exactly 128 such values of κ for each $\alpha \neq 0$ when S is the AES S-box.

This implies that for each choice of θ there is a probability of 2^{-4} that the key is compatible with the tweak difference (2^{-1} for each diagonal byte). When the key is compatible we have $\Pr(x_7 \to x_{12}) \geqslant 2^{-28}$, because Equation 3.6 has at least 2 solution. In other cases, the probability of passing the total characteristic is zero.

3.2.4.2 A class of 2^{119} weak keys

We now assume that the key $k_5 + k_{11}$ has at least one diagonal byte equal to zero. This happens with probability 2^{-6}. Without loss of generality, we consider that the first byte of the key is zero. We also assume that the tweak difference θ is compatible with the key, which happens with probability 2^{-3} (2^{-1} for each non-zero diagonal key byte).

This variant of the attack is very similar to the previous one. The main difference is that the characteristic (Figure 3.6) now has a probability 2^{-21} of following the middle rounds (2^{-7} for each of the three rightmost columns).

As in the previous attack, we start by guessing the first byte of $\widehat{k}_{10}$, denoted by K. The probability of following the characteristic is $2^{-116} \cdot 2^{-21} = 2^{-137}$ for the first pair and $2^{-13} \cdot 2^{-21} = 2^{-34}$ for its twin pair, which induces a total probability of 2^{-171}. We therefore build structures of size $2^{85.5}$. After the 70-bit filter, 2^{101} twin pairs remain. The extended filtering costs a single operation per pair (using tables) and keeps a fraction 2^{-54} of the pairs, which gives 2^{47} remaining twin pairs. On average, each remaining twin pair has one key guess for the three last columns, four key guesses for the first byte, and 2^{24} key guesses for bytes 1,2,3. Exhaustively testing these candidates has a complexity of $2^{47+2+24} = 2^{73}$.

This attack covers 2^{119} keys and has complexity $(D, T, M) = (2^{95.5}, 2^{109}, 2^{80.5})$ after iterating over K, with success probability $1 - 1/e \approx 0.63$.

In order to cover more keys, we can repeat the attack with different choices of θ, to cover other keys with a zero byte. We can also modify the characteristic by using a tweak active in a different column and rotating the whole characteristic. For instance, let us assume that we repeat the attack with 32 characteristics, 8 active in each column. If the key has at least one byte of $k_5 + k_{11}$ equal to zero, then 8 of these characteristics correspond to the correct column and

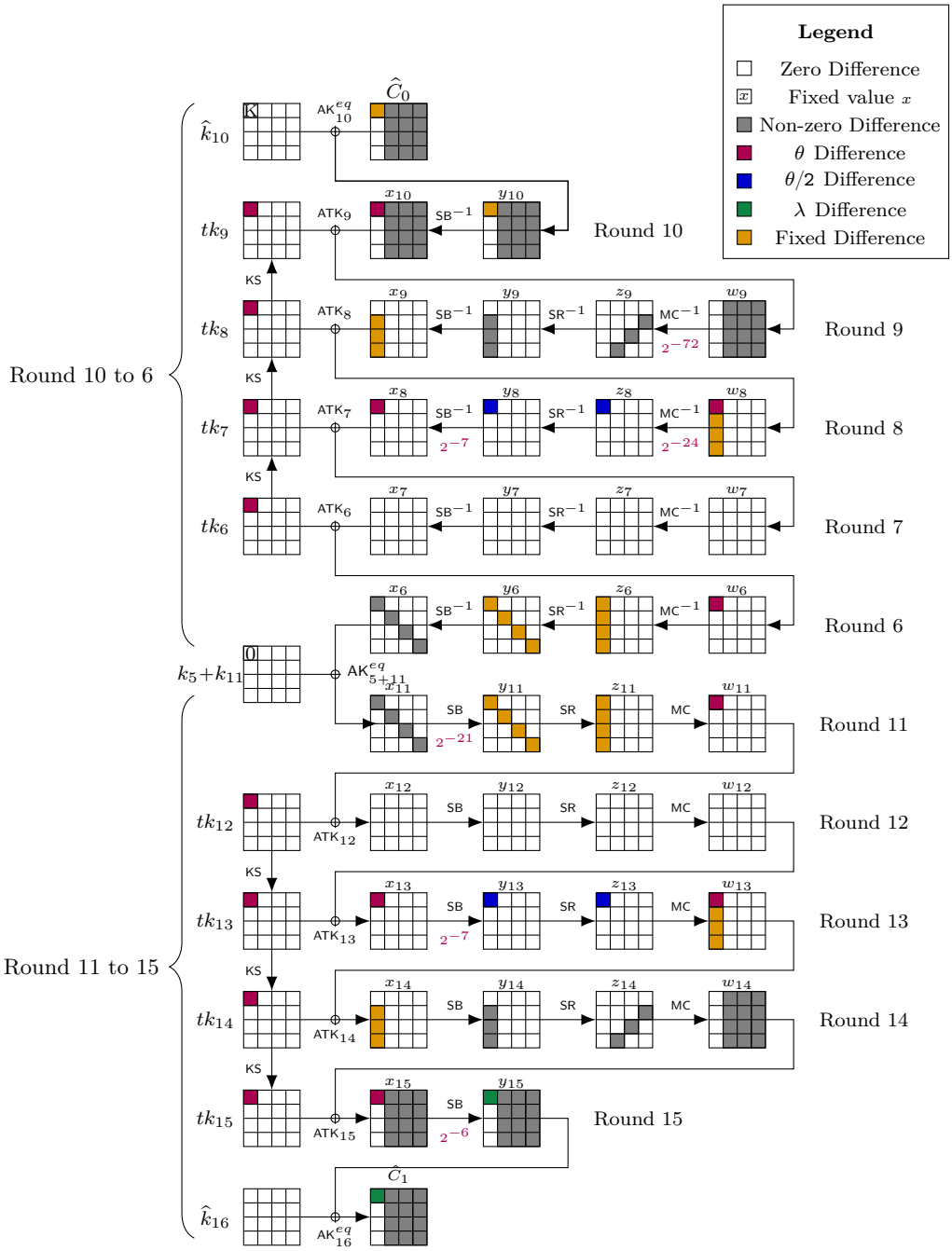


Figure 3.6: Truncated differential characteristic for 2^{119} weak keys against ForkAES-$*$-5-5.

succeed with probability $2^{-3} \cdot (1 - 1/e)$. Therefore the success probability is $1 - \left(1 - 2^{-3} \cdot (1 - 1/e)\right)^8 \approx 0.48$ for this class of $2^{123.95}$ keys, with an attack complexity $(D, T, M) = (2^{100.5}, 2^{114}, 2^{80.5})$.

3.2.4.3 Attacking all keys

We can extend the weak key class to the set of keys compatible with the tweak difference. A random key is compatible with probability 2^{-1} on each diagonal byte, which makes a total probability of 2^{-4}. In order to apply the attack to all keys, we repeat the attack with different choices of θ and different structures, until it succeeds.

The characteristic is still essentially the same, but the middle rounds now have a probability of 2^{-28} (Figure 3.7). Again, we start by guessing the first byte of $\widehat{k}_{10}$, denoted by K. The first pair satisfies the total characteristic with probability $2^{-116} \cdot 2^{-28} = 2^{-144}$, and the twin pair with probability $2^{-13} \cdot 2^{-28} = 2^{-41}$, so that the quartet satisfies the characteristic with probability 2^{-185}. With structures of size $2^{92.5}$, we are left with 2^{115} candidates after the 70-bit filter. Using the table-based variant of the extended filtering of Section 3.2.3.5, we can filter the 2^{115} pairs down to 2^{61} with a complexity of 2^{115}, but since this attack only covers 2^{116} keys with a given guess of K, it is not faster than exhaustive search.

More efficient filtering. In order to reduce the time complexity of the attack, we use a time-data trade-off, increasing the data complexity in order to have a stronger filtering. More precisely, we start with two structures of size 2^{96} (the maximum size possible within this framework) and we only keep the 2^{88} values with $\widehat{C}_1(u)[12] = 0$ and $\widehat{C}'_1(v)[12] = 0$, respectively. As explained in Section 3.2.3.4, we keep 2^{82} values after the 6-bit pre-filter of conditions (3.3) and (3.4). We store the first structure in a hash table indexed by

$$i = \widehat{C}_1[0,1,2,3] \| \widehat{\overline{C_1}}[0,1,2,3], \qquad j = \widehat{C}_1[13] \| \widehat{\overline{C_1}}[12,13],$$

and the second structure in a hash table indexed by

$$i' = \widehat{C}'_1[0,1,2,3] \| \widehat{\overline{C'_1}}[0,1,2,3] + [\lambda,0,0,0,\lambda,0,0,0], \quad j' = \widehat{C}'_1[13] \| \widehat{\overline{C'_1}}[12,13].$$

Because of the 6-bit pre-filter, there are only 2^{58} values of i and i' with non-empty buckets, and each bucket with a fixed (i, j) (respectively (i', j')) contains on average 1 element.

We can now generate efficiently the pairs that pass both the 70-bit filter, and the first 2-bit filter used in the extended filtering of Section 3.2.3.5. We first iterate over all 2^{58} choices of $i = i'$ with non-empty buckets, corresponding to the 70-bit filter. For each $i = i'$, we iterate over the 2^{46} choices of j and j' such that the 2-bit filter is satisfied, and generate the corresponding pairs. Since we have $\widehat{C}_1[12] = \widehat{C}'_1[12] = 0$, the 2-bit filter only depends on j and j', and we can pre-compute the 2^{46} values that satisfy it. This generates 2^{104} pairs for a cost of $2^{58} \cdot 2^{46} = 2^{104}$.

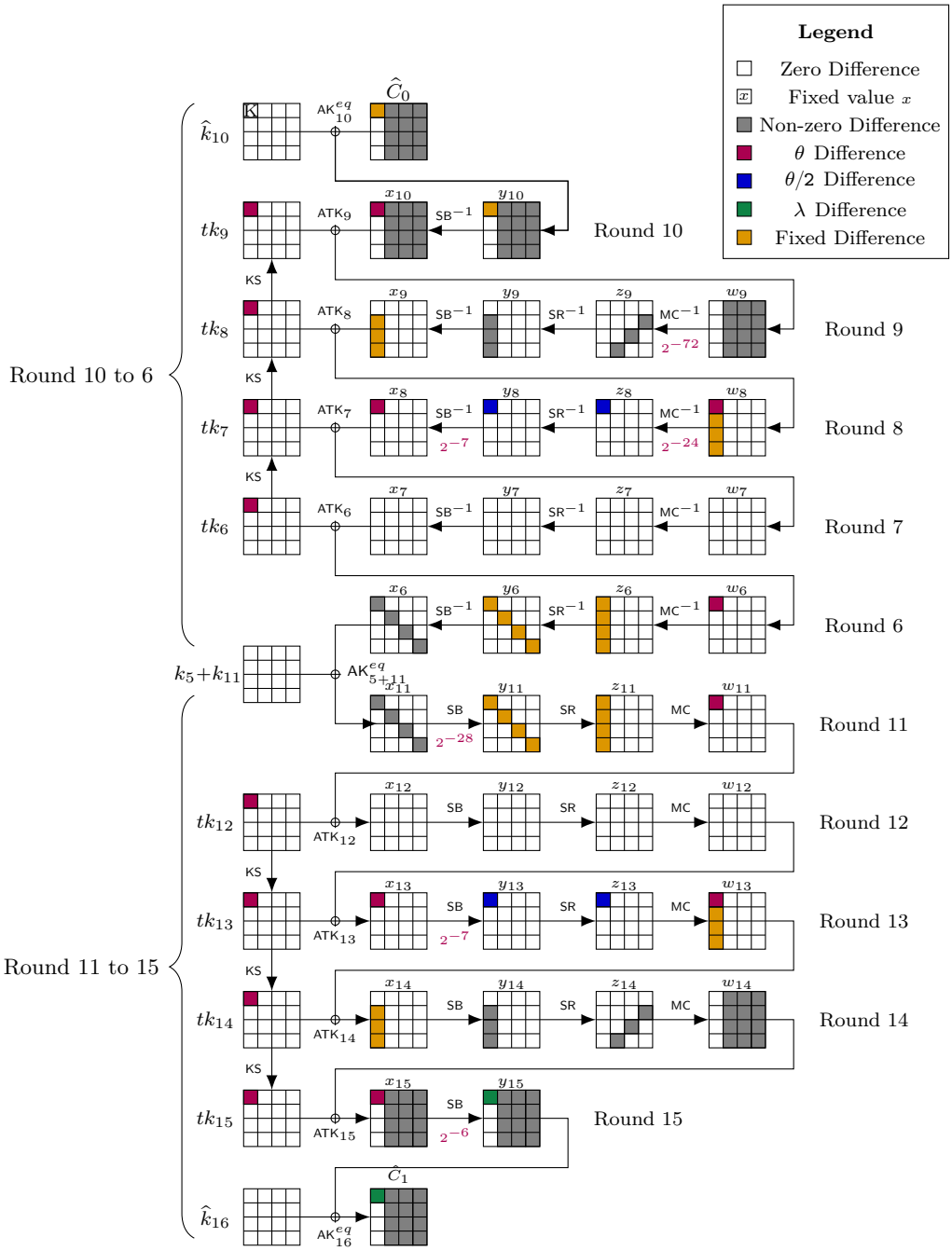


Figure 3.7: Truncated differential characteristic for 2^{124} weak keys against ForkAES-$*$-5-5. The weak key space can be further increased by changing the value of θ.

Then we apply the rest of the extended filter to reduce to 2^{52} pairs, and exhaustively test the suggested keys (for a cost of $2^{52+2+24}$). This attack covers 2^{124} keys and has complexity $(D, T, M) = (2^{106}, 2^{112}, 2^{83})$, after iterating over K. However, since the structures are smaller than required, the success probability is only $1 - e^{-1/512}$ (we consider 2^{176} quadruples, but the probability of following the characteristic is 2^{-185}).

Expected complexity. For a random key, the previous attack succeeds with probability $2^{-4} \cdot (1 - e^{-1/512}) \approx 1/8200$. Therefore, the full attack succeeds after 8200 repetitions on average,[4] with a total complexity of $(D, T, M) = (2^{119}, 2^{125}, 2^{83})$.

We can also slightly reduce the data complexity (at the expense of memory complexity) by reusing the structures of 2^{96} queries with different constraints on $\widehat{C}_1(u)[12]$ and $\widehat{C}'_1(v)[12]$.

3.3 Cryptanalysis of ForkSkinny

ForkSkinny has more rounds than Skinny in reconstruction queries to limit the impact of the weaker diffusion, but the parameters used by the designers have a bad interaction with the tweakey schedule. In Section 3.3.3 and Section 3.3.4, we show attacks against several variants of ForkSkinny exploiting this weakness. We rely on two well-known properties of the Skinny tweakey schedule. First, each round uses only key material from one half of the key (depending on the parity of the round number). Second, some tweakey differences lead to inactive round keys every 30 rounds because differences cancel out (the tweakey schedule is linear).

In particular, when r_0 is odd, encryption queries for C_1 have two consecutive rounds that use key material from the same half of the master key. Moreover, when $r_0 = 27$, encryption queries for C_1 have 27 rounds of "blank" key schedule; this allows to have two consecutive cancellation events (cancellation at round i is followed by a round using the inactive half of the key, then 27 blank rounds, another round with the inactive half, and a second cancellation event). Since ForkSkinny-128-256 uses $r_0 = 27$, this allows to extend the best attacks by three rounds: we have a related-key attack against 24-round ForkSkinny-128-256 with a 128-bit key (corresponding to the parameters used in the NIST submission) and a related-key attack against 26-round ForkSkinny-128-256 with a 256-bit key. These results do not affect the security of the full ForkSkinny because it was designed with a large security margin (the full version of ForkSkinny-128-256 has 48 rounds), but they show that bad parameter choices make it significantly weaker than Skinny.

At the time of the publication of our work [BDL20], no previous third-party cryptanalysis of ForkSkinny existed. However, the different analysis of Skinny can directly be applied to ForkSkinny, since the ForkSkinny encryption $P \mapsto C_0$ corresponds exactly to a Skinny encryption. Since our publication

[4] Note that we can use 492 different characteristics (with 123 choices of θ and using the four rotations), so that the probability of success of each attempt is mostly independent.

Algorithm	Model	κ	Attack	Rds.	Data	Time	Mem.	Reference
ForkSkinny-128-256	RTK_2	128	Imp. Diff.	24	$2^{126.5}$	$2^{126.5}$	$2^{101.5}$	Section 3.3.3[1]
ForkSkinny-128-256	RTK_2	128	Imp. Diff.	24	2^{118}	$2^{126.3}$	2^{121}	[HGS+24][1,2]
ForkSkinny-128-256	RTK_2	128	Rectangle	25	$2^{118.9}$	$2^{118.9}$	$2^{119.2}$	[QDW+21][2]
Skinny-128-$*$	RTK_2	256	Imp. Diff.	23	$2^{124.5}$	$2^{251.5}$	2^{248}	[LGS17]
Skinny-128-$*$	RTK_2	256	Imp. Diff.	23	$2^{124.4}$	$2^{243.6}$	$2^{163.4}$	[SMB18][1]
Skinny-128-$*$	RTK_2	256	Rectangle	26	$2^{126.5}$	$2^{241.4}$	2^{136}	[DQS+22][2]
ForkSkinny-128-256	RTK_2	256	Imp. Diff.	26	2^{125}	$2^{254.6}$	2^{160}	Section 3.3.4
ForkSkinny-128-256	RTK_2	256	Imp. Diff.	26	2^{127}	$2^{250.3}$	2^{160}	Section 3.3.4
ForkSkinny-128-256	RTK_2	256	Imp. Diff.	26	$2^{128.6}$	$2^{238.5}$	$2^{179.6}$	[HGS+24][1,2]
ForkSkinny-128-256	RTK_2	256	Rectangle	28	$2^{118.9}$	2^{247}	2^{136}	[QDW+21][2]
ForkSkinny-128-256	RTK_2	256	Rectangle	28	$2^{118.9}$	$2^{224.8}$	$2^{118.9}$	[DQS+22][2]
ForkSkinny-128-256	RTK_2	256	Rectangle	28	$2^{123.9}$	$2^{212.9}$	$2^{123.9}$	[SYC+24][2]

Table 3.3: Cryptanalysis results against Skinny and ForkSkinny. The column κ denotes the key size. Since our work, the best attacks on the different versions of Skinny and ForkSkinny have improved.

[1] As pointed out by [BL24], the complexity of the original attack is flawed, following an inaccuracy in [SMB18].

[2] These results were found after our publication.

however, the cryptanalysis of ForkSkinny was improved with other impossible differential [HGS+24] and rectangle attacks [QDW+21; DQS+22; SYC+24]. Results on Skinny and ForkSkinny are listed in Table 3.3.

Notation We denote the state at the input of round i as x_i, the state after SubCells as y_i, after AddRoundTweakey as z_i and after ShiftRows as w_i. The output of MixColumns is x_{i+1}. In our description of the attack we ignore the AddConstants operation and the branch constant for simplicity, since they do not impact differential attacks. The round subkey of round i (generated from the tweakey by the tweakey schedule) is denoted as tk_i, while the input tweakey words are denoted as $TK1$ and $TK2$.

For a 128-bit state s, we denote by $s[j]$ the j-th byte of the state, with the following byte-order (following the Skinny specification):

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

Equivalent first and last round. Since there are no whitening keys in Skinny, we can ignore the SubCells operation of the first round, and the ShiftRows and MixColumns operations in the last round, because they can be evaluated without knowing the key.

We also define an equivalent first round key that is applied after ShiftRows and MixColumns, so that we can also ignore those operations:

$$\widehat{tk}_1 = \mathsf{MC}(\mathsf{SR}(tk_1)).$$

3.3.1 Related-tweakey Attacks on Skinny

As described in Section 2.3.4.1, multiple attacker models exist in the TWEAKEY framework. In related-tweakey models RTK_i (for $i \in \{1, 2, 3\}$), the attacker may introduce differences in i tweakey states. He can use the tweakey difference to cancel the state difference and obtain inactive rounds in the middle of a differential characteristic. Furthermore, some attacks, such as the impossible differential attack, use two independent trails, and each trail can take advantage of inactive rounds.

On Skinny, half of the tweakey state is XORed to the state each round. In this subsection, we briefly analyse the possible state cancelations in differential trails under the RTK_1 and RTK_2 attacker models.

Differential trails with two inactive rounds in the RTK_1 attacker model. When there is a single tweakey word with a difference, the analysis is rather simple. If there is an active byte in the master tweakey, it will alternatively move to the top and bottom half of the tweakey state, and every second round key has a non-zero difference. Therefore, we can have two consecutive inactive rounds. This is illustrated on Figure 3.8.

Differential trails with four inactive rounds in the RTK_2 attacker model. When the attacker can introduce a difference in two tweakey states, the analysis is more complex. The attacker can choose differences in each tweakey word that will cancel at some intermediate round. This gives three consecutive inactive round keys, and four consecutive inactive rounds. Since all tweakey words have the same cell permutation in the tweakey schedule, the difference stays in a single cell of the round keys. This is illustrated on Figure 3.9.

The LFSR construction used in the tweakey schedule makes it possible to prove that such a cancellation can only happen every 30 rounds. More precisely, the smallest i such that there exist values $\delta \neq 0$ with $\alpha^i(\delta) = \delta$ is $i = 15$.

Alternatively, Table 3.4 represents the tweakey differences of a differential trail construction in the RKT2 attacker model.

3.3.2 Related-tweakey Attacks on ForkSkinny

The analysis of Skinny differential trail in related tweakey attacker models is also applicable to ForkSkinny. Encryption queries from P to C_0 correspond exactly to Skinny encryption queries, but encryption queries from P to C_1 use a slightly different tweakey schedule: the subkeys used are taken from indices $0, 1, \ldots, r_{\text{init}} - 1, r_{\text{init}} + r_0, r_{\text{init}} + r_0 + 1, \ldots r_{\text{init}} + r_0 + r_1 - 1$. To put it another way, there are r_0 "blank" rounds of tweakey schedule at the forking point. This is

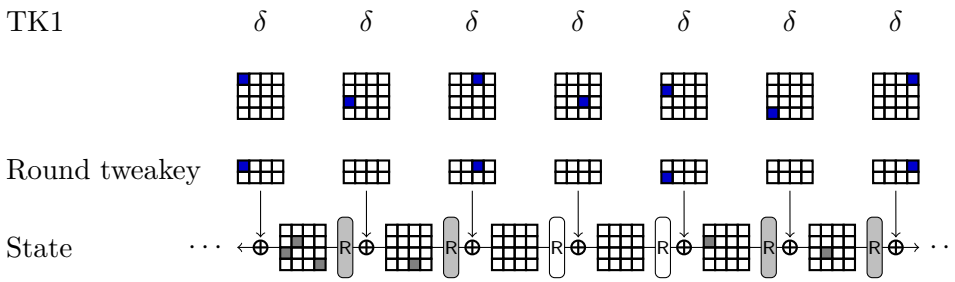


Figure 3.8: Differential trail construction on Skinny in the RTK_1 model.

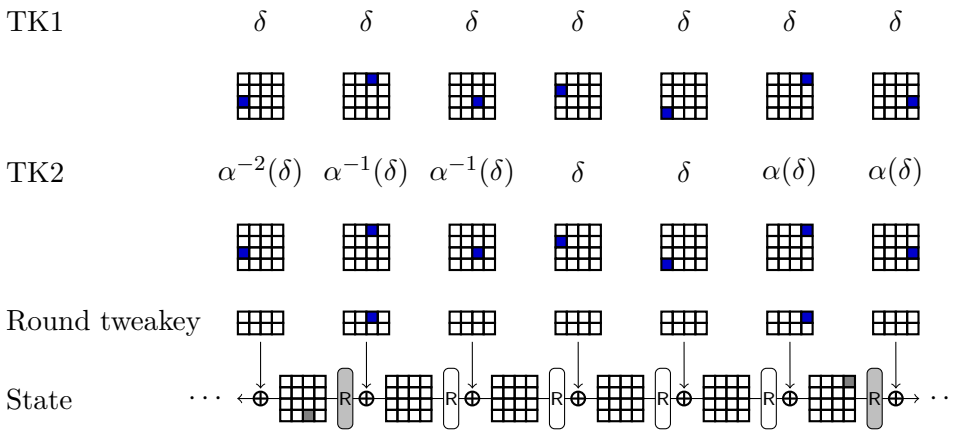


Figure 3.9: Differential trail construction on Skinny in the RTK_2 model.

Round	2	3	4	5	6
TK^1	δ	$[\delta]$	δ	$[\delta]$	δ
TK^2	$\alpha^{-1}(\delta)$	$[\alpha^{-1}(\delta)]$	δ	$[\delta]$	$\alpha(\delta)$
Subtweakey	$\alpha^{-1}(\delta)+\delta$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\alpha(\delta)+\delta$

Table 3.4: Tweakey difference in a differential trail of Skinny in the RTK_2 attacker model. Values in square brackets indicate that the difference is in the lower part of the state.

a weakness if the value of r_0 is poorly chosen, because cancellation in the round keys can occur more frequently than in the original Skinny tweakey schedule.

3.3.2.1 Extending attacks by one round when r_0 is Odd

When r_0 is odd, the round keys before and after the forking point are taken from the same half of the master tweakey. In particular, if there is no difference in this half, there are two consecutive inactive round keys. This allows to extend most differential trails (and therefore differential attacks) by one round.

For instance, attacks in the RTK_2 setting can use the differential trail pattern represented in Table 3.5.

Round	1	2	3	4	$(2t+1)+5$	$(2t+1)+6$
TK^1	δ	$[\delta]$	δ	$[\delta]$	$[\delta]$	δ
TK^2	$\alpha^{-1}(\delta)$	$[\alpha^{-1}(\delta)]$	δ	$[\delta]$	$[\alpha^{t+1}(\delta)]$	$\alpha^{t+2}(\delta)$
Subtweakey	$\alpha^{-1}(\delta)+\delta$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\alpha^{t+2}(\delta)+\delta$

Table 3.5: Tweakey difference in a differential trail on ForkSkinny in the RTK_2 attacker model. Values in square brackets indicate that the difference is in the lower part of the state and the vertical line represents the forking point.

In particular, all parameters proposed for ForkSkinny use odd values of r_0 which make this type of attack possible. This is a deliberate choice of the designers, because an even value of r_0 would give a similar property for reconstruction queries.

In general, taking advantage of this property requires to design a dedicated attack on the primitive because the blank rounds move the position of the active tweakey. However, the use of $r_1 = 31$ in ForkSkinny-128-288 makes it easier to reuse existing attacks because the cell permutation P_T has a period of 16. More precisely, the first active step after the blank rounds is step $2t + 7 = 37$ in the previous figure. Therefore the active tweakey is in the same position as at round 5, which would be the first active step in an attack on Skinny. This allows to reuse the same trails when extending an existing attack by one round (the same type of reuse will be shown in detail in Section 3.3.3).

3.3.2.2 Extending attacks by three rounds when $r_0 = 27$

For some specific values of r_0, there might exist differential trails with more than one additional consecutive inactive round. In particular, the pattern of Table 3.5 leads to 6 inactive round keys if $\alpha^{t+2}(\delta) = \delta$. For the α corresponding to the LFSR used in Skinny, there are some choices of δ such that $\alpha^{15}(\delta) = \delta$. We denote by $\mathcal{S}$ the set of those values[5] (15 non-zero values, and the zero value). When $r_0 = 27$, and using such a δ, we have differential characteristics with 6 inactive round keys rather than 3 in the RTK_2 setting, as highlighted by Table 3.6.

As it turns out, ForkSkinny-128-256 uses precisely $r_0 = 27$, therefore it is possible to extend attacks against Skinny-128-256 by three rounds. However, this generally requires to adapt the Skinny attacks and to repeat the analysis, because

[5]This corresponds to all values for ForkSkinny variants with 4-bit cells.

Round	1	2	3	4	27+5	27+6	27+7	27+8
TK^1	δ	$[\delta]$	δ	$[\delta]$	$[\delta]$	δ	$[\delta]$	δ
TK^2	$\alpha^{-1}(\delta)$	$[\alpha^{-1}(\delta)]$	δ	$[\delta]$	$[\alpha^{14}(\delta)]$	$\alpha^{15}(\delta)$	$[\alpha^{15}(\delta)]$	$\alpha^{16}(\delta)$
Subtweakey	$\alpha^{-1}(\delta)+\delta$	**0**	**0**	**0**	**0**	**0**	**0**	$\alpha^{16}(\delta)+\delta$

Table 3.6: Tweakey difference in a differential trail when $r_0 = 27$. Values in square brackets indicate that the difference is in the lower part of the state and the vertical line represents the forking point.

the difference at round $27 + 8$ against ForkSkinny is at a different position than the round 5 difference in the original Skinny attack.

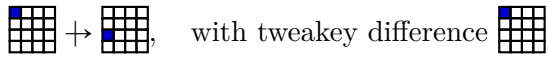
We now present two explicit related-key attacks taking advantage of this property: a 24-round attack against ForkSkinny-128-256 when used in the RTK_2 model with a 128-bit key, and a 26-round attack against ForkSkinny-128-256 in the RTK_2 model with a 256-bit key. Both attacks reach about 3 more rounds than the best existing attack at publication time against Skinny with the same parameters.

3.3.3 A 24-round Attack on ForkSkinny-128-256 with 128-bit Key in the RTK_2 Model

We first focus on ForkSkinny-128-256 with a 128-bit key under the RTK_2 model.

In this setting, we have a very interesting result: we can exactly reproduce the impossible differential attack against Skinny-128-128 in the RTK_1 model presented in [SMB18, Appendix A] to attack ForkSkinny-128-256 in the RTK_2 model, gaining 5 rounds. However, as noted by Bonnetain & Lallemand [BL24], the complexity of the attack of [SMB18, Appendix A] possesses a slight inaccuracy, which we take into account in this thesis. A description of impossible differential attacks is given in Section 2.2.3.3. The initial attack is based on trails with two inactive rounds (following the general idea of Figure 3.8). We turn it into an impossible differential attack with trails with seven inactive rounds as given in Section 3.3.2.2, by relaxing the attacker model from RTK_1 to RTK_2.

More precisely, the existing attack on Skinny-128-128 in the RTK_1 model uses a 13-round impossible differential distinguisher built with the miss-in-the-middle technique, where the upper characteristic starts with 2 inactive rounds [SMB18, Figure 3]:



There are 255 specific values of input/output differences for which the impossible differential holds (see [SMB18] for details). The full impossible differential is shown in Figure 3.10. The 13-round distinguisher is used to attack 19 rounds, by adding three rounds at the top, and three at the bottom.

In our attack, we reuse this distinguisher, but there are 7 inactive rounds at the beginning, using two cancellations in the tweakey schedule (taking advantage of $r_0 = 27$). The new distinguisher has 5 extra rounds, plus 27 blank rounds in the

key schedule. Since we add 32 key schedule rounds compared to the initial attack, the position of the active tweakey differences after the cancellations are the same as in the original attack, because the order of P_T is 16. Therefore, the following 18-round impossible differential distinguisher holds for ForkSkinny-128-256 versions with $r_0 = 27$, with the forking point after 4 rounds:

The tweakey differences must be chosen in $\mathcal{S}$, therefore in total, there are 15 impossible differentials: when the difference in TK^1 is $\delta \neq 0 \in \mathcal{S}$, the difference in TK^2 is $\alpha^{-1}(\delta)$, the difference at the input is $\alpha^{-1}(\delta) + \delta$, and the difference at the output is $\alpha^6(\delta) + \delta$.

Since the impossible differential has the same shape as in the original Skinny attack, we can almost reuse entirely the key-recovery part of the attack of [BL24], adding 3 rounds in the backward and forward directions. The only difference lies in the additionnal requirement of our impossible differential distinguisher: the tweak difference should belong to $\mathcal{S}$. This implies that we can not use 8-bit tweak structures like in the corrected analysis of Bonnetain and Lallemand [BL24], but we can only build 4-bit structures on the tweak, using differences belonging to the vectorial subspace $\mathcal{S}$. We therefore lose 4 bits in data complexity and gain 4 bits in memory complexity compared to the analysis of [BL24].

This leads to an attack on a 24-round reduced version of ForkSkinny-128-256 with 128-bit key where $r_{\text{init}} = 7$, $r_0 = 27$ and $r_1 = 17$, with complexity:

$$(D, T, M) = (2^{126.5}, 2^{126.5}, 2^{101.5}).$$

Unfortunately, we did not find an impossible differential attack against Skinny-128-$*$ in the RTK_2 model with 128-bit key in the literature to directly compare the security of Skinny and ForkSkinny, but it seems reasonable to expect that this type of attack would reach 21 rounds.[6]

More generally, we can reuse most of the differential attacks on Skinny-128-128 in the RTK_1 model based on trails with two inactive rounds, and extend them by 5 rounds when attacking ForkSkinny-128-256 with a 128-bit key and $r_0 = 27$.

3.3.4 A 26-round Attack on ForkSkinny-128-256 with 256-bit Key in the RTK_2 Model

In order to show an attack against more rounds, we assume the use of a 256-bit key (without tweak) in the RTK_2 model. In this settings, the attacks which cost up to 2^{256} in time complexity are valid. This setting has been studied by several previous works in the case of Skinny-128-256 [SMB18; LGS17], and the best known attack reaches 23 rounds. We extend and adapt the key-recovery step of the attack of

[6] Using a cancellation in the key schedule allows trails with four inactive rounds, so this type of impossible differential attack should reach 21 rounds. However, building the concrete attack requires some tedious work to verify the impossible differential and the key-recovery.

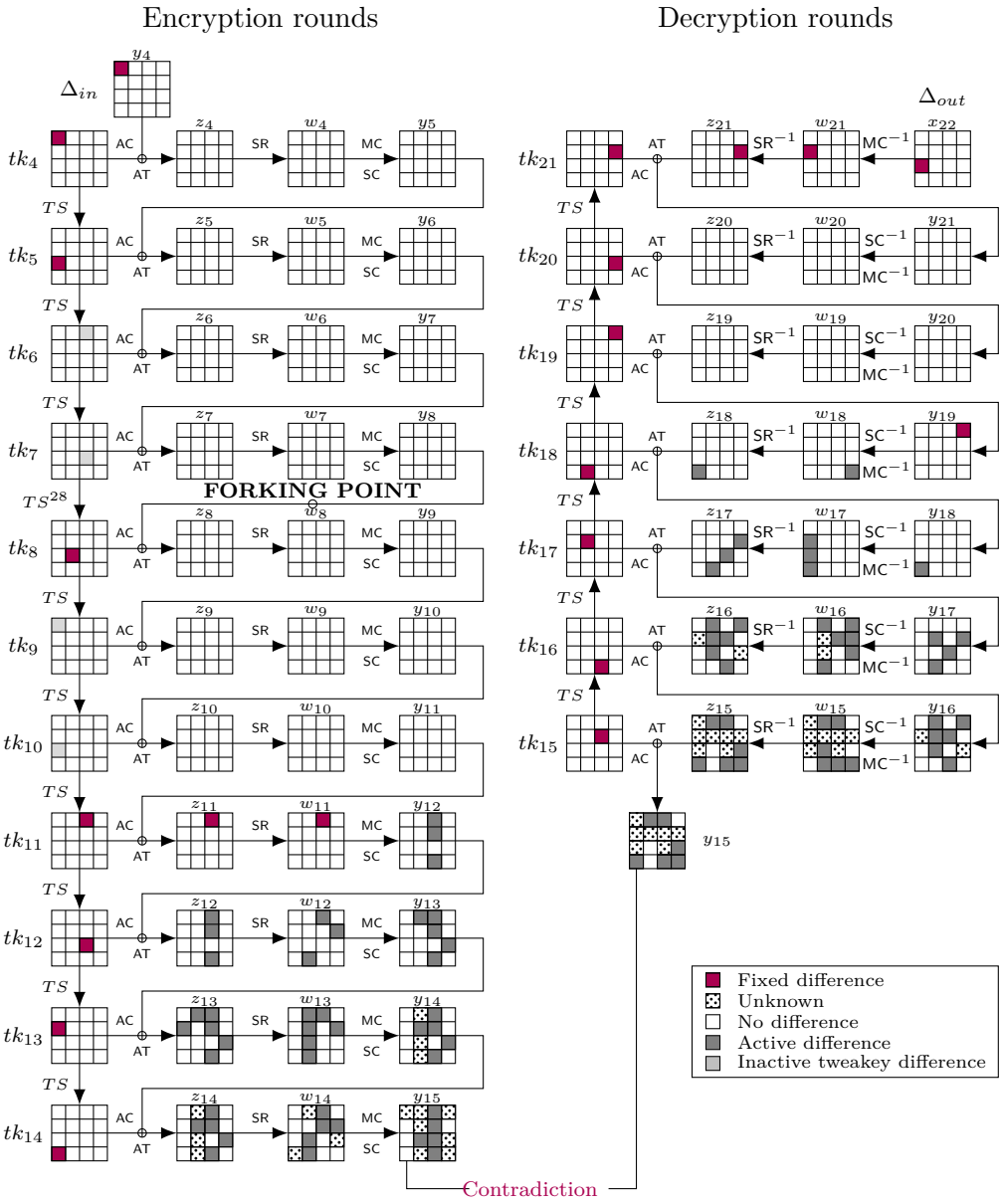


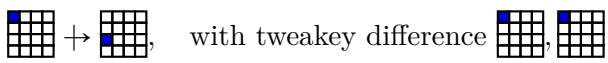
Figure 3.10: Impossible differential characteristic on 18-round ForkSkinny-128-256 in the RTK_2 model, using the second branch (4 rounds before forking point, 14 rounds after forking point).

Section 3.3.3 to propose a 26-round attack on ForkSkinny-128-256. The impossible differential attack is based on the impossible differential chararacteristic shown in Figure 3.10. The chararacteristic spans 18 round, but the first round does not include the SubCells operation (the S-Box layer). This distinguisher gains 3 rounds compared to the attack on Skinny-128-256 in the RTK_2 model of [SMB18].

Next, we use the distinguisher in a key-recovery attack. As explained above, there are 3 rounds before the distinguisher, and 5 rounds after. The key-recovery part of our attack is slightly different to the key-recovery part of the attack on Skinny-128-256 of [SMB18], since the input difference of our characteristic is in cell 0 instead of cell 1. The new key-recovery is illustrated by Figure 3.11 and is detailed in the next subsections.

3.3.4.1 Description of the attack

An impossible differential attack starts from an impossible differential distinguisher. Following [SMB18], there are 15 related-key impossible differences for 18 rounds, of the form:



when the difference in TK^1 is $\delta \neq 0 \in \mathcal{S}$, the difference in TK^2 is $\alpha^{-1}(\delta)$, the difference at the input is $\alpha^{-1}(\delta) + \delta$, and the difference at the output is $\alpha^6(\delta) + \delta$.

The distinguisher is placed at rounds 4 to 21. We collect a number of plaintext/ciphertext pairs under related tweakeys with the input/output differences described in the tweakey-recovery part (Figure 3.11). Then, we process each pair to find whether some keys would lead to the input and output differences of the impossible differential characteristic at rounds 4 and 21; such keys can not be the encryption key. After processing enough data, we expect that only a small fraction of the keyspace remains. More precisely, the attack will process several structures of plaintexts, such that each structure will rule out a number of key candidates.

A structure is generated by fixing all bytes of the plaintext except bytes 1,4,11,14 to a random value, and taking the 2^{32} possible values of bytes 1,4,11, and 14. Since the attack uses related keys, we encrypt each structure under several keys. More precisely, we use 16 different keys, and we encrypt each plaintext under key $k_1^* + [0, \ldots, 0, \delta], k_2^* + [0, \ldots, 0, \alpha^{-3}(\delta)]$ (where $k_1^* \| k_2^*$ is the secret key), for all $\delta \in \mathcal{S}$, because byte 15 of the master key moves to byte 0 at round 4 for the impossible differential distinguisher.

By linearity of the LFSR α, any pair of tweakey in this set satisfies the conditions for the impossible differential distinguisher, with a difference δ in both words of the tweakey state tk_6 (the TK^1 word is updated three times through the LFSR).

Each structure contains 2^{36} plaintext and values, and there are roughly $2^{36+35} = 2^{71}$ useful pairs in each set. In total, we denote N the number of pair produced, with $N/2^{71}$ different structures.

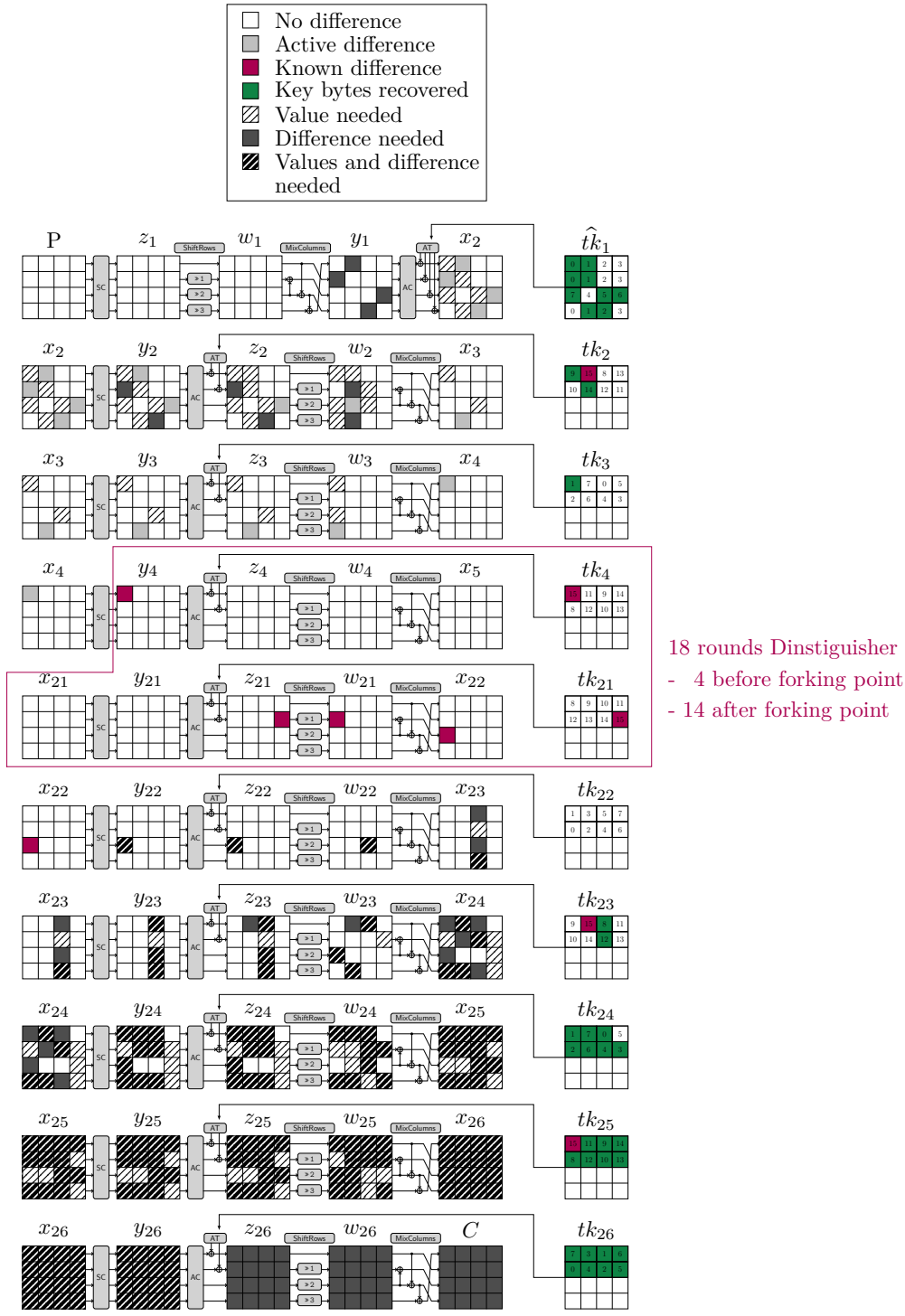


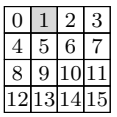
Figure 3.11: Key-recovery of the impossible differential attack of ForkSkinny-128-256 on 26 rounds.

3.3.4.2 Processing the pairs

We now explain how to process each pair to identify the keys that lead to the impossible differential. The following key-recovery procedure is inspired by [SMB18]. We attach partial key information to each of the N pairs collected, initially empty, and we will incrementally fill up the key information.

1. **Round 1**. From the fixed difference $\Delta y_2[1] = \Delta tk_2[1]$ and the difference $\Delta x_2[1]$ derived from the plaintext, use Lemma 2.2 to deduce $\widehat{tk}_1[1]$ $(TK[1])$.

   We can represent the knowledge about the key graphically:

   Cells colored in light gray have been recovered once by the attacker (he does not know the values of both tweakey states TK^1 and TK^2). Cells colored in dark gray have been recovered twice by the attacker, so he can freely recover this tweakey byte at any round (he knows the corresponding cells in TK^1 and TK^2 by linearity). We will use this representation after each step of the process.

2. **Round 26**. Because $\Delta w_{25}[4, 10, 11, 14] = 0$, the MixColumns operation gives us four equations on Δx_{26}:

   (i) $$\Delta w_{25}[4] = \Delta x_{26}[4] + \Delta x_{26}[12] + \Delta x_{26}[8] = 0.$$

   From the knowledge of the ciphertext, we can compute $\Delta x_{26}[8, 12]$ since no key material is added on bytes 8 and 12. Then, we compute the quantity $\Delta x_{26}[4]$. Therefore we know both $\Delta x_{26}[4]$ and $\Delta y_{26}[4]$. Using Lemma 2.2, we can determine $tk_{26}[4]$ $(TK[0])$.

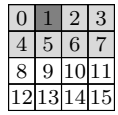
   (ii) $$\Delta w_{25}[14] = \Delta x_{26}[2] + \Delta x_{26}[14] = 0,$$
   (iii) $$\Delta w_{25}[10] = \Delta x_{26}[6] + \Delta x_{26}[14] = 0.$$

   $\Delta x_{26}[14]$ can be computed from the ciphertext to derive $\Delta x_{26}[2]$ and $\Delta x_{26}[6]$. Then we can apply Lemma 2.2 and determine $tk_{26}[2, 6]$ $(TK[1, 2])$.

   (iv) $$\Delta w_{25}[11] = \Delta x_{26}[7] + \Delta x_{26}[15] = 0.$$

   $\Delta x_{26}[15]$ can be computed from the ciphertext to derive $\Delta x_{26}[7]$. We use Lemma 2.2 to determine $tk_{26}[7]$ $(TK[5])$.

   We then guess $tk_{26}[0, 1, 3, 5]$ $(TK[3, 4, 6, 7])$ to compute the full state x_{26}. This step has a complexity of $N \times 2^{4\times 8}$ and we are left with $N \times 2^{4\times 8}$ candidates.



3. **Round 25**. We have $\Delta w_{24}[8,9,14] = 0$, the following equations can be derived:

(i) $$\Delta w_{24}[14] = \Delta x_{25}[2] + \Delta x_{25}[14] = 0,$$
(ii) $$\Delta w_{24}[8] = \Delta x_{25}[4] + \Delta x_{25}[12] = 0,$$
(iii) $$\Delta w_{24}[9] = \Delta x_{25}[5] + \Delta x_{25}[13] = 0.$$

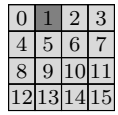
Since we can compute $\Delta x_{25}[12,13,14]$ from x_{26}, we can deduce $\Delta x_{25}[2,4,5]$ and apply Lemma 2.2 to recover $tk_{25}[2,4,5]$ ($TK[8,9,12]$).

We then guess $tk_{25}[3,6,7]$ ($TK[10,13,14]$) and compute the rightmost two columns of w_{24}. We can then compute $\Delta x_{24}[8,12]$ from the values and differences of $w_{24}[10,15]$. Since $w_{23}[0,4] = 0$, we have an 8-bit filter:

$$\Delta w_{23}[0] + \Delta w_{23}[4] = \Delta x_{24}[8] + \Delta x_{24}[12] = 0.$$

Next, we guess $tk_{25}[0,1]$ ($TK[11,15]$), which allows to compute the full state x_{25}.

The complexity of this step is $2^{4\times 8}$ per candidate left after the previous step, therefore $N \times 2^{8\times 8}$, and there are $N \times 2^{8\times 8}$ remaining candidates.



4. **Round 24**. Having $\Delta w_{23}[9,10,12,14] = 0$, the following equations can be derived:

(i) $$\Delta w_{23}[14] = \Delta x_{24}[14] + \Delta x_{24}[2] = 0,$$
(ii) $$\Delta w_{23}[10] = \Delta x_{24}[14] + \Delta x_{24}[6] = 0.$$

We can compute $\Delta x_{24}[14]$ from x_{25}, deduce $\Delta x_{24}[2,6]$ and recover $tk_{24}[2,6]$ ($TK[0,4]$) using Lemma 2.2.

(iii) $$\Delta w_{23}[9] = \Delta x_{24}[5] + \Delta x_{24}[13] = 0.$$

We can compute $\Delta x_{24}[13]$ from x_{25}, deduce $\Delta x_{24}[5]$ and recover $tk_{24}[5]$ ($TK[6]$) using Lemma 2.2. Since the difference $\Delta w_{23}[1]$ cancels out with the tweakey, we have an 8-bit filter:

$$\Delta tk_{23}[1] = \Delta w_{23}[1] = \Delta x_{24}[5].$$

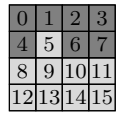
Since $tk_{24}[0]$ ($TK[1]$) has already been recovered twice, the attacker can compute $\Delta x_{24}[0]$ and the last equation becomes an 8-bit condition:

(iv) $$\Delta w_{23}[12] = \Delta x_{24}[0] + \Delta x_{24}[12] = 0.$$

We then guess $tk_{24}[1, 4, 7]$ ($TK[2, 3, 7]$), and compute the values and difference of $x_{23}[10, 14]$ from $w_{24}[8, 13]$. Because $\Delta w_{22}[2, 6] = 0$, we have an 8-bit filter:

$$\Delta w_{22}[2] + \Delta w_{22}[6] = \Delta x_{23}[10] + \Delta x_{23}[14] = 0.$$

This step has a complexity of $2^{5\times 8}$ per candidate from the previous step, therefore a total complexity of $N \times 2^{9\times 8}$. At the end of this step, there are $N \times 2^{8\times 8}$ candidates.



5. **Round 1**. Since we know $\widehat{tk}_1[4, 11, 14]$ ($TK[0, 2, 6]$) from the previous steps, we can compute $\Delta w_2[5, 9, 13]$. Because $\Delta x_3[1, 5, 9] = 0$, we have a 16-bit filter:

(i) $$\Delta x_3[9] = \Delta w_2[5] + \Delta w_2[9] = 0,$$

(ii) $$\Delta x_3[1] + \Delta x_3[5] = \Delta w_2[9] + \Delta w_2[13] = 0.$$

This step has a complexity of 1 per candidate, therefore a total complexity of $N \times 2^{8\times 8}$. We are left with $N \times 2^{6\times 8}$ candidates.

6. **Round 23**.Because $\Delta w_{22}[14] = 0$, the following equation can be derived:

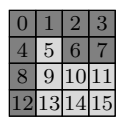
(i) $$\Delta w_{22}[14] = \Delta x_{23}[2] + \Delta x_{23}[14] = 0.$$

We can compute $\Delta x_{23}[14]$ from x_{24} and deduce $\Delta x_{23}[2]$. Using Lemma 2.2, we recover $tk_{23}[2]$ ($TK[8]$).

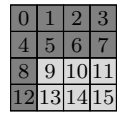
Next, we guess $tk_{23}[6]$ ($TK[12]$). $\Delta z_{21}[7]$ cancels out with the tweakey, therefore we have an 8-bit filter:

$$\Delta tk_{21}[7] = \Delta z_{21}[7] = \Delta x_{22}[8].$$

This step has a complexity of 2^8 per candidate, therefore a total complexity of $N \times 2^{7\times 8}$, and we are left with $N \times 2^{6\times 8}$ candidates.



7. **Round 1**. Guess $\widehat{tk}_1[10]$ ($TK[5]$). Since $\widehat{tk}_1$ is fully known, we can fully compute y_2. This step has a complexity of 2^8 per candidate, therefore a total complexity of $N \times 2^{7\times 8}$, and we are left with $N \times 2^{7\times 8}$ candidates.



8. **Round 2 and 3**. Guess $tk_2[1,5]$ ($TK[14,15]$). We know $w_3[8,12]$ and $\Delta w_3[12]$ from $x_3[8,13]$ and $\Delta x_3[13]$. From $\Delta w_3[12]$ we can compute $\Delta x_4[0]$. Additionally, we know $\Delta y_4[0] = \Delta tk_4[0]$, so we can apply Lemma 2.2, giving us one possible value on average for $x_4[0]$. We derive the following equation from the round 3 MixColumns operation:

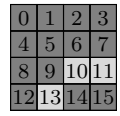
$$x_4[0] = w_3[0] + w_3[8] + w_3[12].$$

Knowing $w_3[8,12]$ and $x_4[0]$, we can deduce $w_3[0]$. We also know $tk_3[0]$ ($TK[1]$) from the previous steps, so we can compute $y_3[0]$ then $x_3[0]$. We derive the following equation from the round 2 MixColumns operation:

$$x_3[0] = w_2[0] + w_2[8] + w_2[12].$$

Knowing $w_2[8,12]$ we can recover $w_2[0] = z_2[0]$. We get $tk_2[0]$ ($TK[9]$).

This step has a complexity of $2^{2\times 8}$ per candidate, therefore a total complexity of $N \times 2^{9\times 8}$, and we are left with $N \times 2^{9\times 8}$ candidates.



3.3.4.3 Complexity analysis

The total number of key bytes recovered is 29, and we deduce $2^{9\times 8}$ impossible keys from each pair. In order to reduce the key space by at least a factor $1/e$, the number of pairs N should be such that:

$$N \times 2^{9\times 8} \geqslant 2^{29\times 8}.$$

This implies $N \geqslant 2^{160}$, therefore we need to construct at least $2^{160-71} = 2^{89}$ sets, using $D = 2^{89+36} = 2^{125}$ plaintexts. The analysis phase has a complexity of $N \times 2^{9\times 8} = 2^{232}$ so the time complexity will be dominated by the exhaustive search over the remaining key space, with $T = 2^{256}/e \approx 2^{254.6}$. A naive implementation would require a memory of $2^{29\times 8}$ to store all the potential keys and remove impossible ones. However an implementation using the early abort technique of [LKK+08] only requires to store the plaintext/ciphertext pairs. Indeed, Algorithm 3.1 gives an

implementation of the attack were we store the pairs and iterate over the possible key bytes values to perform the key-recovery.

Therefore, we end up with a complexity of

$$(D, T, M) = (2^{125}, 2^{254.6}, 2^{160}).$$

Complexity parameters from [BNS14]. We can verify our complexity analysis using the generic formula of [BNS14]. The parameters corresponding to our attack are:

$$\begin{aligned} |\Delta_{\text{in}}| &= 4.5 \times 8 & |\Delta_{\text{out}}| &= 16 \times 8 \\ c_{\text{in}} &= 4 \times 8 & c_{\text{out}} &= 16 \times 8 \\ |k_{\text{in}} \cup k_{\text{out}}| &= 29 \times 8. \end{aligned}$$

The formula for the minimum data complexity D_{min} given in [BNS14] confirms our analysis:

$$D_{\text{min}} = N_{\text{min}} \times 2^{n+1-|\Delta_{\text{in}}|-|\Delta_{\text{out}}|} = 2^{c_{\text{in}}+c_{\text{out}}} \times 2^{n+1-|\Delta_{\text{in}}|-|\Delta_{\text{out}}|} = 2^{125}.$$

We can also reduce the time complexity, at the expanse of an increase in the data complexity. For instance with a data complexity of $D = 2^{127}$, we reduce the fraction of remaining keys to $P \approx e^{-4} \approx 2^{-5.7}$, so that the time complexity is reduced to $2^{250.3}$.

3.4 Conclusion

In this chapter, we showed that the security of Forkciphers can not automatically be derived from the security of the underlying ciphers. We presented an attack on full ForkAES exploiting the weak diffusion in the middle rounds of reconstruction queries. We therefore approve the designers choice to propose a higher number of rounds r_0 and r_1 compared to r_{init} in ForkSkinny [ALP+19a] .

However, the chosen number of rounds (27) for some variants of ForkSkinny leads to improved characteristics on the second branch, by exploiting properties of the tweakey schedule. In particular, we extend an existing impossible differential characteristic on Skinny-128-256 by 3 rounds to attack ForkSkinny-128-256, thus reaching 26 rounds. Our attack is however outperformed by some recent rectangle attacks: the state-of-the-art rectangle attacks of Song *et al.* [SYC+24] reach 26 rounds of Skinny-128-256 and 28 rounds of ForkSkinny-128-256 on the second branch. This nonetheless highlights that the weakness due to the number of rounds $r_0 = 27$ can be exploited in other types of attack.

Algorithm 3.1: Implementation of this attack with early abort and low memory. tk represents the accumulated knowledge about the key.

Input: L_0 with $|L_0| = N$
  for all $tk_{26}[0, 1, 3, 5], tk_{25}[3, 6, 7]$ **do**
    $L_1 \leftarrow \{\}$
    for all $(p, c) \in L_0$ **do** ▷ $2^{7s} \times N$ iterations
      Deduce $\widehat{tk}_1[1], tk_{26}[2, 4, 6, 7], tk_{25}[2, 4, 5]$
      if $\Delta x_{24}[8] = \Delta x_{24}[12]$ **then**
        Append$(L_1, (p, c, tk))$
        ▷ $|L_1| = 2^{-s} \times N$
    for all $tk_{25}[0, 1]$ **do**
      $L_2 \leftarrow \{\}$
      for all $(p, c, tk) \in L_1$ **do** ▷ $2^{8s} \times N$ iterations
        Deduce $tk_{24}[2, 5, 6]$
        if $\Delta x_{24}[13] = \Delta tk_{23}[1]$ **and** $\Delta x_{24}[0] = \Delta x_{24}[8]$ **then**
          Append$(L_2, (p, c, tk))$
          ▷ $|L_2| = 2^{-3s} \times N$
      for all $tk_{24}[1, 4, 7]$ **do**
        $L_3 \leftarrow \{\}$
        for all $(p, c, tk) \in L_2$ **do** ▷ $2^{9s} \times N$ iterations
          Deduce $tk_{23}[2]$
          if $\Delta x_{23}[10] = \Delta x_{23}[14]$ **and** $\Delta w_2[5] = \Delta w_2[9] = \Delta w_2[13]$ **then**
            Append$(L_3, (p, c, tk))$
            ▷ $|L_3| = 2^{-6s} \times N$
        for all $tk_{23}[6]$ **do**
          $L_4 \leftarrow \{\}$
          for all $(p, c, tk) \in L_3$ **do** ▷ $2^{7s} \times N$ iterations
            if $\Delta x_{22}[8] = \Delta tk_{21}[7]$ **then**
              Append$(L_4, (p, c, tk))$
              ▷ $|L_4| = 2^{-7s} \times N$
          for all $\widehat{tk}_1[10], tk_2[1, 5]$ **do**
            Create hash table H indexed by 13 deduced keys words
            for all $(p, c, tk) \in L_4$ **do** ▷ $2^{9s} \times N$ iterations
              Deduce $tk_2[0]$
              $H[tk] \leftarrow 1$
            for all tk **do** ▷ 2^{29s} iterations
              if $H[tk] = 0$ **then**
                Run exhaustive search over 2^{3s} keys

Chapter 4
Boomerang Attacks on AES and AES-based Ciphers

In this chapter, we propose a thorough description of the boomerang attack, an overview of boomerang attacks against the AES in the literature, and present a new boomerang attack framework, with applications to AES-based ciphers. The new framework, the truncated boomerang attack, is described in a paper coauthored with Gaëtan Leurent: "Truncated Boomerang Attacks and Application to AES-based Ciphers" [BL23b], published at *EUROCRYPT* 2023. This framework leads to new distinguishing, key-recovery or secret S-box key-recovery attacks against round-reduced AES [DR02], Kiasu-BC [JNP14a], Deoxys-BC [JNP+21], and a marginal distinguisher against full TNT-AES [BGG+20]. We also present a dedicated boomerang attack on 6-round AES based on a joint work with Orr Dunkelman, Gaëtan Leurent, Nathan Keller and Victor Mollimard, submitted at ToSC 2024, leading to improved boomerang key-recovery attacks on 6-round AES.

Contents

4.1 Summary

We start with an overview of the results of our boomerang attacks on AES and AES-based ciphers, presented in the following sections of the chapter. The results are compared to state-of-the art attacks on reduced-round AES in Table 4.1, Table 4.2, and Table 4.3.

	Type		Data	Time	Mem.	Ref
Distinguisher	Yoyo	ACC	2^{123}	2^{123}	2^{61}	[RBH17]
	Exchange attack	CP	2^{88}	2^{88}	2^{88}	[BR19]
	Exchange attack	ACC	2^{84}	2^{83}	2^{32}	[Bar19]
	Truncated differential	CP	$2^{89.4}$	$2^{96.5}$	2^{33}	[BGL20]
	Truncated boomerang	ACC	2^{87}	2^{87}	2^{66}	Section 4.4.3.1
Key-Recovery	Square	CP	2^{32}	2^{72}	2^{32}	[DKR97]
	Square	CP	2^{35}	2^{44}	2^{32}	[FKL+01]
	Square	CP	2^{33}	2^{40}	2^{32}	[DGK+24]
	Mixture	CP	2^{31}	2^{73}	2^{31}	[BDK+18]
	Mixture	CP	2^{44}	2^{63}	2^{44}	[YTX+24]
	Boomerang	ACC	2^{71}	2^{71}	2^{33}	[Bir04]
	Retracing boomerang	ACC	2^{55}	2^{80}	2^{31}	[DKR+20]
	Boomeyong	ACC	2^{80}	2^{80}	2^{28}	[RSP21]
	Truncated boomerang	ACC	2^{59}	2^{61}	2^{59}	Section 4.4.3.2
	Boomerang	ACC	2^{51}	2^{68}	2^{32}	Section 4.5.1
	Boomerang	ACC	2^{51}	2^{66}	2^{42}	Section 4.5.1
	Boomerang	ACC	2^{57}	2^{61}	2^{33}	Section 4.5.2
Secret S-box KR	Square	CP	2^{64}	2^{90}	2^{69}	[TKK+15]
	Truncated boomerang	ACC	2^{94}	2^{94}	2^{56}	Section 4.4.3.3

Table 4.1: Attacks against 6-round AES in different settings.
CP: chosen plaintexts / ACC: chosen plaintexts and adaptively-chosen ciphertexts.

	Rounds	Type	Data		Time	Ref
Kiasu-BC	7	Square (KR)	$2^{43.6}$	CP	$2^{48.5}$	[DEM16]
	8	Meet-in-the-Middle (KR)	2^{116}	CP	2^{116}	[TAY16]
	8	Imposs. Diff (KR)	2^{118}	CP	2^{118}	[DL17]
	8	Boomerang (KR)	2^{103}	ACC	$2^{103.1}$	[DL17]
	8	Truncated boomerang (KR)	2^{83}	ACC	2^{83}	Section 4.4.4
TNT-AES	∗-5-∗	Boomerang (dist.)	2^{126}	ACC	2^{126}	[BGG+20]
	5-∗-∗	Impossible differential (KR)	$2^{113.6}$	CP	$2^{113.6}$	[GGL+20]
	∗-∗-∗	Generic (dist.)	$2^{99.5}$	CP	$2^{99.5}$	[GGL+20]
	∗-∗-∗	Generic (dist.)	2^{69}	ACC	2^{69}	[JKN+24]
	∗-5-∗	Truncated boomerang (dist.)	2^{76}	ACC	2^{76}	Section 4.4.5
	5-5-∗	Truncated boomerang (KR)	2^{87}	ACC	2^{87}	Section 4.4.5
	∗-6-∗	Truncated boomerang (dist.)	$2^{127.8}$	ACC	$2^{127.8}$	Section 4.4.5

Table 4.2: Attacks against Kiasu-BC and TNT-AES.

		Previous				New			
Model	Rnd	Data	Time	Mem	Ref	Data	Time	Mem	Ref
RTK1	8					B 2^{88}	2^{88}	2^{73}	Figure 4.12
	9					B 2^{135}	2^{174}	2^{129}	Figure 4.13
RTK2	8	B 2^{28}	2^{28}	2^{27}	[Sas18a][1]	B 2^{27}	2^{27}	2^{27}	Figure 4.14
	9	B 2^{98}	2^{112}	2^{17}	[Sas18a]	B $2^{55.2}$	$2^{55.2}$	$2^{55.2}$	Figure 4.15
	10	B $2^{98.4}$	$2^{109.1}$	2^{88}	[ZDJ19]	B $2^{94.2}$	$2^{95.2}$	$2^{94.2}$	Figure 4.16
	11	R $2^{122.1}$	$2^{249.9}$	$2^{128.2}$	[ZDJ19]	B 2^{129}	$2^{223.9}$	2^{129}	Figure 4.16
RTK3	10	B 2^{22}	2^{22}	2^{17}	[Sas18a]	B $2^{19.4}$	$2^{19.4}$	2^{18}	Figure 4.17
	11	B 2^{100}	2^{100}	2^{17}	[Sas18a]	B $2^{32.7}$	$2^{32.7}$	$2^{32.7}$	Figure 4.18
	12	B 2^{98}	2^{98}	2^{64}	[ZDJ19]	B $2^{67.4}$	$2^{67.4}$	2^{65}	Figure 4.19
	13	R $2^{125.2}$	$2^{186.7}$	2^{136}	[ZDJ+19]	B $2^{126.7}$	$2^{170.2}$	$2^{126.7}$	Figure 4.20
	14	R $2^{125.2}$	$2^{282.7}$	2^{136}	[ZDJ+19]	B 2^{129}	$2^{278.8}$	2^{129}	Figure 4.21

[1]The probability of Sasaki's trail is 2^{-56} with structures, thus we believe that the complexity of the attack is actually 2^{30} in data and time and 2^{29} in memory.

Table 4.3: Boomerang (B) and rectangle (R) attacks against variants of Deoxys-BC. Most attacks succeed with probability 1/2. In the RTKi model, the attacker controls i blocks of 128-bit tweakey.

Truncated Boomerang Attacks. In Section 4.4, we present a generic framework to describe boomerang attacks based on truncated differentials. Our framework allows to easily evaluate the complexity of an attack based on properties of the truncated differentials, and to compare different settings. We first apply our framework to reduced AES in Section 4.4.3. On 6-round AES, we obtain a distinguisher with complexity 2^{87}, and a key-recovery with complexity 2^{61}, improving the previous best boomerang attack with complexity 2^{71} [Bir04].

We adapt the key-recovery attack to 8-round Kiasu-BC in Section 4.4.4 by revisiting a previous boomerang attack with complexity 2^{103} [DL17]. Using structures of ciphertexts, we obtain the best attack against Kiasu-BC, with complexity 2^{83}.

We also apply a variant of the 6-round attack to the full TNT-AES [BGG+20], and obtain a marginal distinguisher with complexity slightly below 2^{128} in Section 4.4.5. The attack is not competitive with the recent generic attack against TNT with complexity $\mathcal{O}(2^{n/2})$ (or $\tilde{\mathcal{O}}(2^{n/2})$ in its memory-efficient variant) [JKN+24], but it uses a lower memory (2^{32} instead of 2^{64}), and it can distinguish TNT with 6-round AES from TNT with a PRP. Moreover, this is the first property of 6-round AES that can be used to target a generic construction using 6-round AES as a building block, to the best of our knowledge. We also provide an attack on reduced TNT-AES, using a 5-round boomerang trail.

Finally, we apply the framework to Deoxys-BC using MILP, and obtain improved attacks against most variants. We present the detailed attacks in Section 4.4.7.

Improved Boomerang Attacks on AES. In Section 4.5, we improve on the retracing boomerang attack [DKR+20] and present the second best key-recovery attack on 6-round AES (after the Square attack [DKR97; FKL+01; DGK+24]) with complexity 2^{61}. We use insights from the truncated boomerang attack to decrease the time complexity of the retracing boomerang attack, without the memory overhead of the truncated boomerang attack. We also show two other attacks against 6-round AES with different data/time/memory trade-offs.

4.2 The Boomerang Attack

4.2.1 Description of the Boomerang Attack

The boomerang attack was introduced by Wagner in 1999 [Wag99] to attack the ciphers COCONUT98 [Vau03], KHUFU [Mer91], 6-round FEAL [Miy91] and CAST [Ada97b]. The attack uses chosen plaintext and adaptative ciphertext queries to generate quartets with specific differences at an intermediate state of the cipher. The attacker decomposes the full cipher E into two subciphers E_0 (the upper part) and E_1 (the lower part), with $E = E_1 \circ E_0$, with high probability differentials on E_0 and E_1 (of probabilities p and q), denoted respectively $\Delta_{\text{in}} \xrightarrow[E_0]{p} \Delta_{\text{out}}$ and

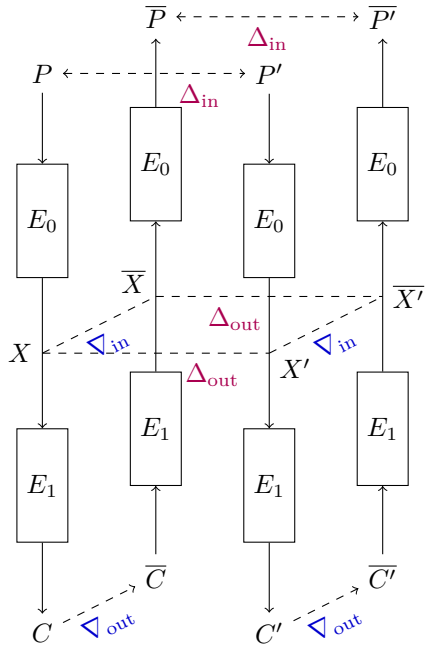


Figure 4.1: Construction of a boomerang quartet.

$\nabla_{\text{in}} \xrightarrow[E_1]{q} \nabla_{\text{out}}$, using the notations introduced in Section 2.2.2. The attack proceeds as follows, as highlighted in Figure 4.1:

1. Generate pairs of plaintext (P, P') such that $P + P' = \Delta_{\text{in}}$, and query the corresponding ciphertexts $(C, C') = (E(P), E(P'))$.
2. Shift the ciphertexts pairs into new pairs $(\bar{C}, \bar{C}') = (C + \nabla_{\text{out}}, C' + \nabla_{\text{out}})$ and query their decryptions $(\bar{P}, \bar{P}') = (E^{-1}(\bar{C}), E^{-1}(\bar{C}'))$.
3. Look for pairs with $\bar{P} + \bar{P}' = \Delta_{\text{in}}$.

4.2.2 Analysis

We here present the analysis of this attack, as performed by Wagner [Wag99]. We denote $(X, X', \bar{X}, \bar{X}')$ the middle states of the four plaintext/ciphertext pairs, as described in Figure 4.1. We have the following relations:

$$\begin{aligned} X &= E_0(P), & X' &= E_0(P'), & \bar{X} &= E_0(\bar{P}), & \bar{X}' &= E_0(\bar{P}'), \\ C &= E_1(X), & C' &= E_1(X'), & \bar{C} &= E_1(\bar{X}), & \bar{C}' &= E_1(\bar{X}'). \end{aligned}$$

The differentials in E_0 and E_1 imply that:

$$\begin{aligned} \Pr[X + X' = \Delta_{\text{out}} | P + P' = \Delta_{\text{in}}] &= p \\ \Pr[X + \bar{X} = \nabla_{\text{in}} | C + \bar{C} = \nabla_{\text{out}}] &= q \\ \Pr[X' + \bar{X}' = \nabla_{\text{in}} | C' + \bar{C}' = \nabla_{\text{out}}] &= q, \end{aligned} \tag{4.1}$$

where the probabilities are averaged on the keys and on respectively P, C, and C'.

When $X + X' = \Delta_{\text{out}}$ and $X + \bar{X} = X' + \bar{X}' = \nabla_{\text{in}}$, we obtain:

$$\begin{aligned}\bar{X} + \bar{X}' &= \bar{X} + X + X + X' + X' + \bar{X}' \\ &= \nabla_{\text{in}} + \Delta_{\text{out}} + \nabla_{\text{in}} \\ &= \Delta_{\text{out}}.\end{aligned}$$

Eventually, the differential on E_0 gives:

$$\Pr[\bar{P} + \bar{P}' = \Delta_{\text{in}} | \bar{X} + \bar{X}' = \Delta_{\text{out}}] = p. \tag{4.2}$$

Wagner pointed out that if the four events in Equation 4.1 and Equation 4.2 are independent, the boomerang trail probability p_b can be computed as follows:

$$\begin{aligned}p_b = \Pr[&E_0(P) + E_0(P + \Delta_{\text{in}}) = \Delta_{\text{out}}, \\ &E_0(P) + E_1^{-1}(E(P) + \nabla_{\text{out}}) = \nabla_{\text{in}}, \\ &E_0(P + \Delta_{\text{in}}) + E_1^{-1}(E(P + \Delta_{\text{in}}) + \nabla_{\text{out}}) = \nabla_{\text{in}}, \\ &E^{-1}(E(P) + \nabla_{\text{out}}) + E^{-1}(E(P + \Delta_{\text{in}}) + \nabla_{\text{out}}) = \Delta_{\text{in}}] \quad = p^2 q^2.\end{aligned}$$

In particular, p_b is a lower bound of the probability that $\bar{P} + \bar{P}' = \Delta_{\text{in}}$:

$$\Pr\left[E^{-1}(E(P) + \nabla_{\text{out}}) + E^{-1}(E(P + \Delta_{\text{in}}) + \nabla_{\text{out}}) = \Delta_{\text{in}}\right] \geqslant p_b. \tag{4.3}$$

When $p^2 q^2 \gg 2^{-n}$, this gives a distinguisher for the cipher using $\mathcal{O}(p^{-2} q^{-2})$ quartets. Indeed, the probability that $\bar{P} + \bar{P}' = \Delta_{\text{in}}$ is 2^{-n} for a random permutation, and a simple distinguisher consists in counting the number of thrown boomerang quartets before the event $\bar{P} + \bar{P}' = \Delta_{\text{in}}$ occurs; it should be roughly 2^n for a random permutation and roughly p_b^{-1} for the target cipher. In most cases, the distinguisher can be converted into a key-recovery by exploiting key dependencies in the distinguisher.

However, Murphy presented in 2011 [Mur11] some incompatible boomerang trails on AES and DES. In Murphy's boomerang, the independance assumption between the four events (Equation 4.1 and Equation 4.2) does not hold, and the four events are actually incompatible. Since then, a special interest has been dedicated to the analysis of the middle round transition. This middle round transition is also called the *boomerang switch*.

In the following, we say that a quartet of plaintexts $(P, P', \bar{P}, \bar{P}')$ is a boomerang quartet if $P + P' = \bar{P} + \bar{P}' = \Delta_{\text{in}}$ and $E(P) + E(\bar{P}) = E(P') + E(\bar{P}') = \nabla_{\text{out}}$; in this case the thrown boomerang *returns*. From a high level point a view, a boomerang attack exploits the fact that there are more boomerang quartets in the cipher E than for a random cipher.

4.2.3 Improvements of the Boomerang Attack

Several variants and improvements of the boomerang attack have been proposed since Wagner's original work, both from a conceptual point of view and in the analysis of the attack. In the past years, there has been a regain of interest for this type of attack, as highlighted by numerous works [CHP+17; Sas18a; ZDJ+19; ZDM+20; DDV20; QDW+21; HBS21; RSP21; ZCJ21; HNE22; SZY+22; LWL22; DQS+22; BL23b; LTX23a; LTX23b], and this can be explained with two main reasons. First, lightweight tweakable block ciphers with linear tweakey schedules, such as Skinny [BJK+16], Craft [BLM+19] or Deoxys-BC [JNP+21] usually allow the introduction of inactive rounds in differential trails in related tweakey models. This leads to very high probability differentials on multiple rounds, while the differential probability highly decreases for subsequent rounds. This type of property is favourable to attacks using boomerang characteristics: to the best of our knowledge, the current best attacks in most related-tweakey models on Skinny, ForkSkinny, Craft and Deoxys-BC are boomerang-based attacks [SZY+22; BL23b; ZZY+23; LTX23b; SYC+24]. Second, computer-aided tools, such as MILP, SAT or CP, whose applicability in cryptography was demonstrated by Bouillaguet *et al.* [BFL11] in 2010 and later by Mouha *et al.* [MWG+11], offer new approachs to improve attacks. These tools allow to find optimal differentail trails, and can be adapted to find optimal boomerang trails. In addition, mounting key-recovery attacks using MILP programming is not trivial, and has seen a series of improvements recently.

We will now describe some variants and improvements of the boomerang attack, commonly used in recent works.

4.2.3.1 Multiple differentials

Since the differences Δ_{out} and ∇_{in} are not used by the attacker, boomerang quartets can be detected with any internal difference, as long as the same difference is obtained with both pairs. This was spotted by Wagner in his original work [Wag99]. The probability of a boomerang trail can be approximated to the sum of the probabilities of each trail:

$$p_b = \sum_{\Delta_{\text{out}}, \nabla_{\text{in}}} \Pr[\Delta_{\text{in}} \xrightarrow[E_0]{} \Delta_{\text{out}}]^2 \Pr[\nabla_{\text{in}} \xrightarrow[E_1]{} \nabla_{\text{out}}]^2.$$

We can then define equivalent probabilities $\hat{p}$ and $\hat{q}$:

$$p_b = \hat{p}^2 \hat{q}^2, \quad \hat{p} = \sqrt{\sum_{\Delta_{\text{out}}} \Pr[\Delta_{\text{in}} \xrightarrow[E_0]{} \Delta_{\text{out}}]^2}, \quad \hat{q} = \sqrt{\sum_{\nabla_{\text{in}}} \Pr[\nabla_{\text{in}} \xrightarrow[E_1]{} \nabla_{\text{out}}]^2}.$$

In addition, as performed in the boomerang attack of Biryukov against the AES [Bir04], it is common to allow several differences for the returning pair $(\bar{P}, \bar{P}')$, in order to increase the boomerang probability. We accept the boomerang quartet if $\bar{P} + \bar{P}' \in \mathcal{D}^0_{\text{in}}$, where $\mathcal{D}^0_{\text{in}}$ is a set of differences. The boomerang probability becomes:

$$p_b = \sum_{\substack{\bar{\Delta}_{\text{in}} \in \mathcal{D}^0_{\text{in}} \\ \Delta_{\text{out}}, \nabla_{\text{in}}}} \Pr[\Delta_{\text{in}} \xrightarrow[E_0]{} \Delta_{\text{out}}] \Pr[\nabla_{\text{in}} \xrightarrow[E_1]{} \nabla_{\text{out}}]^2 \Pr[\bar{\Delta}_{\text{in}} \xrightarrow[E_0]{} \Delta_{\text{out}}]$$
$$= \sum_{\Delta_{\text{out}}, \nabla_{\text{in}}} \Pr[\Delta_{\text{in}} \xrightarrow[E_0]{} \Delta_{\text{out}}] \Pr[\nabla_{\text{in}} \xrightarrow[E_1]{} \nabla_{\text{out}}]^2 \Pr[\Delta_{\text{out}} \xrightarrow[E_0^{-1}]{} \mathcal{D}^0_{\text{in}}].$$

This sum is a bit heavy to compute in practice. Let us denote $\mathcal{D}^0_{\text{out}}$ the set of differences Δ_{out} such that a high probability differential $\Delta_{\text{in}} \xrightarrow[E_0]{} \Delta_{\text{out}}$ exists. Instead of computing the aforementioned formula, if differentials $\Delta_{\text{in}} \xrightarrow[E_0]{} \Delta_{\text{out}}$ have similar probabilities for $\Delta_{\text{in}} \in \mathcal{D}^0_{\text{in}}$ and $\Delta_{\text{out}} \in \mathcal{D}^0_{\text{out}}$, the following approximation can be performed:

$$\sum_{\Delta_{\text{out}}} \Pr[\Delta_{\text{in}} \xrightarrow[E_0]{} \Delta_{\text{out}}] \Pr[\Delta_{\text{out}} \xrightarrow[E_0^{-1}]{} \mathcal{D}^0_{\text{in}}]$$
$$\approx \Pr[\Delta_{\text{in}} \xrightarrow[E_0]{} \mathcal{D}^0_{\text{out}}] \times \Pr[\mathcal{D}^0_{\text{out}} \xrightarrow[E_0^{-1}]{} \mathcal{D}^0_{\text{in}}].$$

Similarly, if $\mathcal{D}^1_{\text{in}}$ denotes the set of differences ∇_{in} such that a high probability differential $\nabla_{\text{in}} \xrightarrow[E_1]{} \nabla_{\text{out}}$ exists, and if the corresponding probabilities are similar, the following truncated approximation can be performed, using truncated differentials:

$$\sqrt{\sum_{\nabla_{\text{in}} \in \mathcal{D}^1_{\text{in}}} \Pr[\nabla_{\text{in}} \xrightarrow[E_1]{} \nabla_{\text{out}}]^2} \approx \sum_{\nabla_{\text{in}} \in \mathcal{D}^1_{\text{in}}} \Pr[\nabla_{\text{in}} \xrightarrow[E_1]{} \nabla_{\text{out}}]$$
$$= \Pr[\nabla_{\text{out}} \xrightarrow[E_1^{-1}]{} \mathcal{D}^1_{\text{in}}].$$

These approximations are only acceptable if all differential probabilities involved are close to the average differential probability; this approximation is extensively used in truncated differential cryptanalysis. With well chosen sets of differences, this is the case on AES, which will be our main point of focus in this sections. The boomerang probability can then be computed under these approximations:

$$p_b = \vec{p}\,\overleftarrow{q}^2\overleftarrow{p}, \qquad \vec{p} = \Pr[\Delta_{\text{in}} \xrightarrow[E_0]{} \mathcal{D}^0_{\text{out}}],$$
$$\overleftarrow{q} = \Pr[\nabla_{\text{out}} \xrightarrow[E_1^{-1}]{} \mathcal{D}^1_{\text{in}}], \qquad \overleftarrow{p} = \Pr[\mathcal{D}^0_{\text{out}} \xrightarrow[E_0^{-1}]{} \mathcal{D}^0_{\text{in}}].$$

In counterpart, since we accept returning quartets with $\bar{P} + \bar{P}' \in \mathcal{D}^0_{\text{in}}$, there is less filter and it is more likely that random quartets are accepted, i.e. quartets that do not follow the boomerang characteristic but randomly possess a right returning difference. Therefore, if we want to get more boomerang quartets than random quartets, we need the following relation:

$$p_b \geqslant \frac{|\mathcal{D}^0_{\text{in}}|}{2^n}.$$

4.2.3.2 Structures

Biham, Dunkelman and Shamir introduced a variant of the boomerang attack using structures for the key-recovery [BDK02]. They start from a boomerang distinguisher with fixed differences Δ_{in} and ∇_{out}, and add extra rounds at the beginning and at the end. By propagating the differences Δ_{in} and ∇_{out}, they obtain a set of possible input differences $\mathcal{D}_{\text{in}}$ and output differences $\mathcal{D}_{\text{out}}$. In a typical SPN cipher, these sets are vector spaces.

The attacker builds a structure $P + \mathcal{D}_{\text{in}} = \{P + \delta : \delta \in \mathcal{D}_{\text{in}}\}$, and uses it as starting point for the attack. A structure of $|\mathcal{D}_{\text{in}}|$ elements defines $|\mathcal{D}_{\text{in}}|^2/2$ pairs, and $|\mathcal{D}_{\text{in}}|/2$ of them lead to the fixed difference Δ_{in}. Therefore, a structure requires $|\mathcal{D}_{\text{in}}|$ queries and produces $|\mathcal{D}_{\text{in}}|/2$ pairs; the use of structures covers additional rounds without increasing the data complexity, under the assumption that the additional filter to discard wrong quartets is not costly. When combined with multiple differentials, the structure vector space $\mathcal{D}_{\text{in}}$ might be different from the set of returning differences $\mathcal{D}^0_{\text{in}}$ defined in the previous subsubsection. This is typically the case on most boomerang attacks against AES.

Structures can also be used on the ciphertext side, by shifting each ciphertext with all differences in $\mathcal{D}_{\text{out}}$. However, many later works do not use structures on the ciphertext side. We provide an analysis of boomerang attacks with truncated trails in Section 4.4, where we improve the standard boomerang analysis, using plaintext and ciphertext structures.

4.2.3.3 Plaintext-only attacks

The amplified boomerang attack [KKS01], later improved to the rectangle attack [BDK01], are variants of the boomerang attack using only encryption queries (without adaptively chosen decryption queries). The complexity of these attacks increase from $(pq)^{-2}$ to roughly $2^{n/2}(pq)^{-1}$ (with the same condition that $pq \gg 2^{-n/2}$). These attacks benefit from the improvements of the computation of boomerang trail probabilities (with fixed Δ_{in}) discussed earlier in this subsection, since they use the exact same boomerang characteristic.

In short, the standard boomerang attack builds candidate quartets that satisfy three out of the four the boomerang conditions ($P + P' = \Delta_{\text{in}}$ and $E(P) + E(\bar{P}) = E(P') + E(\bar{P}') = \nabla_{\text{out}}$), and looks for quartets that follow the last equality ($\bar{P} + \bar{P}' = \Delta_{\text{in}}$). On the other hand, the rectangle and amplified boomerang attacks build candidate quartets that satisfy two of those conditions ($P + P' = \bar{P} + \bar{P}' = \Delta_{\text{in}}$) and look for quartets satisfying the two last equalities ($E(P) + E(\bar{P}) = E(P') + E(\bar{P}') = \nabla_{\text{out}}$). The same boomerang characteristic is used, but the way to generate the data changes. When the boomerang probability is high, the boomerang attack achieves much better complexities than the rectangle attack. However, the rectangle attack becomes interesting when the boomerang probability is low: a large amount of data is needed, and very large structures can be built on the plaintext side in rectangle attacks, decreasing drastically the complexity of the attacks. An in-depth analysis of rectangle attacks including key-recovery was performed recently by

Song *et al.* [SZY+22].

In this thesis however, we mainly focus on standard boomerang attacks.

4.2.3.4 Towards a better understanding of the boomerang switch

The boomerang incompatibility. Murphy showed in 2011 that the boomerang switch might never happen [Mur11], and highlighted this fact with incompatible boomerang characterstics on AES and DES: for any key, there exists no boomerang quartet satisfying the boomerang characteristic. To understand why an incompatibility can occur, we need to dive into differentials. Let us consider a cipher $\tilde{E}_K$ and a differential $\Delta_{\text{in}} \xrightarrow[\tilde{E}_K]{p} \Delta_{\text{out}}$. Let us define the sets of input (resp. output) values that satisfy the differential for a fixed key K:

$$\mathcal{S}_{K,\text{in}} = \{P | \tilde{E}_K(P) + \tilde{E}_K(P + \Delta_{\text{in}}) = \Delta_{\text{out}}\},$$
$$\mathcal{S}_{K,\text{out}} = \{C | \tilde{E}_K^{-1}(C) + \tilde{E}_K^{-1}(C + \Delta_{\text{out}}) = \Delta_{\text{in}}\}.$$

The probability p is an average probability on the keys and on the plaintexts:

$$p = \underset{K}{\text{Avg}} \, |\mathcal{S}_{K,\text{in}}| \cdot 2^{-n} = \underset{K}{\text{Avg}} \, |\mathcal{S}_{K,\text{out}}| \cdot 2^{-n}.$$

By definition, $(P, P+\Delta_{\text{in}})$ is a correct input pair of the differential $\Delta_{\text{in}} \xrightarrow[\tilde{E}_K]{p} \Delta_{\text{out}}$ if and only if $P \in \mathcal{S}_{K,\text{in}}$. Similarly, $(C, C + \Delta_{\text{out}})$ is a correct output pair of the differential if $C \in \mathcal{S}_{K,\text{out}}$. This highlights that satisfying a differential adds constraints on the input P and output C.

In standard differential cryptanalysis of iterated block ciphers, it is a common practice to ignore the constraints induced by the differentials and to concatenate the differential transitions of each round by multiplying the transition probabilities. This is the Markov cipher assumption defined in Section 2.2.2. It is justified if the subkeys added each round are unrelated, but some subkey dependances can lead to incompatible differential trails [PT22].

However, this constraints on the input/output pairs satisfying a differential trail can lead to incompatible boomerangs, even if we consider independant subkeys. To demonstrate the concept of boomerang incompatibility, we will consider an SPN cipher, where the upper differential (resp. lower) on E_0 (resp. E_1) is based on a single differential trail. We assume that we can divide the lower part:

$$E_1 = E_1^1 \circ E_1^0.$$

such that E_1^0 does not depend on the key. This happens for instance if E_1^0 consists of a round function without the key addition (which is encompassed in E_0).

The boomerang characteristic is illustrated in Figure 4.2.

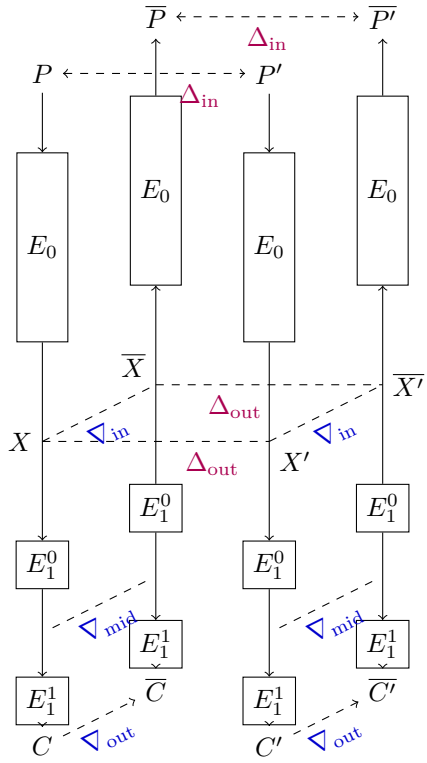


Figure 4.2: Illustration of a boomerang incompatibility.

Let us denote $\mathcal{S}^1_{K,\text{in}}$ the sets of input states satisfying the differential $\nabla_{\text{in}} \xrightarrow[E_1^0]{} \nabla_{\text{mid}}$. Since E_1^0 is independant on the key, we omit the K and denote the set $\mathcal{S}^1_{\text{in}}$ in the rest of the analysis.

Let us consider a boomerang quartet $(P, P', \bar{P}, \bar{P}')$ following the boomerang characteristic and denote $(X, X', \bar{X}, \bar{X}')$ its internal states between E_0 and E_1, as illustrated by Figure 4.2. From the boomerang characteristic, we have the following relations:

$$X + X' = \Delta_{\text{out}}, \quad \bar{X} + \bar{X}' = \Delta_{\text{out}}, \quad X + \bar{X} = \nabla_{\text{in}}, \quad X' + \bar{X}' = \nabla_{\text{in}},$$
$$X \in \mathcal{S}^1_{\text{in}}, \quad X' \in \mathcal{S}^1_{\text{in}}.$$

In particular:

$$X \in \mathcal{S}^1_{\text{in}}, \quad X + \Delta_{\text{out}} \in \mathcal{S}^1_{\text{in}} \quad \Longrightarrow \; X \in \mathcal{S}^1_{\text{in}} \cap (\mathcal{S}^1_{\text{in}} + \Delta_{\text{out}}).$$

The boomerang incompatibility appears when the set $\mathcal{S} = \mathcal{S}^1_{\text{in}} \cap (\mathcal{S}^1_{\text{in}} + \Delta_{\text{out}})$ is empty: no solution exists for X. After Murphy's work, multiple works were conducted to better analyse the boomerang switch and avoid incompatibilities.

The Sandwich attack. Instead of splitting the cipher E into two parts $E = E_1 \circ E_0$, Dunkelman, Keller and Shamir [DKS10] proposed in 2010 to split it in

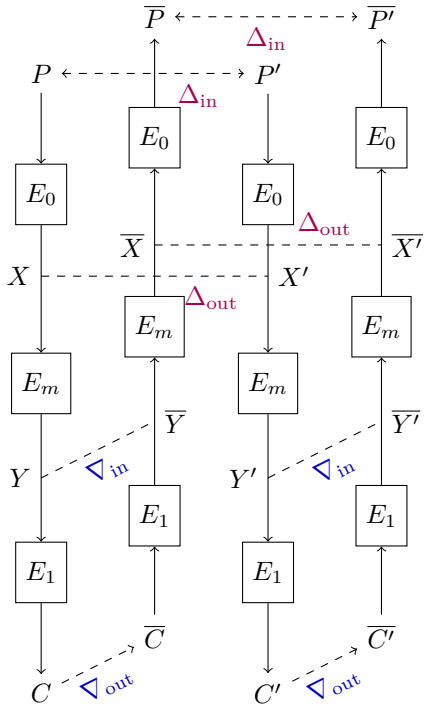


Figure 4.3: The Sandwich attack [DKS10].

three parts $E = E_1 \circ E_m \circ E_0$ with a small E_m in the middle, as illustrated in Figure 4.3. For the analysis, they evaluate the probability of the boomerang trail using the connection probability r of E_m:

$$\Pr\left[\bar{P} + \bar{P}' = \Delta_{\text{in}}\right] \geqslant p_b = p^2 q^2 r,$$
$$r = \Pr\left[E_m^{-1}(E_m(X) + \nabla_{\text{in}}) + E_m^{-1}(E_m(X + \Delta_{\text{out}}) + \nabla_{\text{in}}) = \Delta_{\text{out}}\right].$$

The connection probability r can be evaluated experimentally, and some specific choices of E_m result in $r = 1$ (in particular, when E_m is the identity, we fall back to the standard analysis of boomerangs). Although this is an attempt to improve the boomerang analysis, the sandwich analysis does not ensure by itself that there is no boomerang incompatibility.

The Boomerang Connectivity Table. The Boomerang Connectivity Table (BCT) was introduced by Cid *et al.* [CHP+18] in 2018 to analyze the case of the sandwich attack where E_m is an S-box layer.

Definition 4.1. The *Boomerang Connectivity Table* (BCT) of a ℓ-bit S-box S is defined as a bidimensional table of size $\ell \times \ell$ such that:

$$\text{BCT}[a, b] = |\{x \in \mathbb{F}_2^k | S^{-1}(S(x) + b) + S^{-1}(S(x + a) + b) = a\}|.$$

Let us assume that the n-bit state is composed of $\frac{n}{\ell}$ words of ℓ bits, and the S-box layer consists of the application of a ℓ-bit S-box S on each word independantly. Let us restrict ourselves to the first word. For $i \in [\![0, \dots \frac{n}{\ell} - 1]\!]$, we define ${\Delta_{\text{out}}}^i$ (resp. ${\nabla_{\text{in}}}^i$) as Δ_{out} (resp. ∇_{in}) restricted to the i-th word of the state. In this case, we can count the number of n-bit states satisfying the middle constraints:

$$\begin{aligned}&|\{x \in \mathbb{F}_2^n | E_m^{-1}(E_m(x) + \nabla_{\text{in}}) + E_m^{-1}(E_m(x + \Delta_{\text{out}}) + \nabla_{\text{in}}) = \Delta_{\text{out}}\}| \\ &= \prod_{i=0}^{\frac{n}{k}-1} \text{BCT}[{\Delta_{\text{out}}}^i, {\nabla_{\text{in}}}^i].\end{aligned}$$

The probability r is exactly:

$$r = 2^{-n} \cdot \prod_{i=0}^{\frac{n}{k}-1} \text{BCT}[{\Delta_{\text{out}}}^i, {\nabla_{\text{in}}}^i].$$

As part of the sandwich attack, this analysis improves the standard boomerang probability estimation but does not ensure than no middle round incompatibility exists. Recent works show for instance that setting E_m to a single S-box layer might lead to unaccurate, or even incompatible connection probabilities [YSS+22].

Analysis of multiple rounds. The case where E_m is composed of several rounds has been analyzed in recent works [SQH19; WP19; DDV20]. Several similar cryptographic tools were introduced: a new framework in [SQH19], the Boomerang Difference Table (BDT) in [WP19] and the Extended (resp. Upper and Lower) Boomerang Connectivity Table EBCT (resp. UBCT and LBCT) in [DDV20]. There are different names for the same concept, so we will chose to use the notations of Delaune *et al.* in [DDV20], i.e. EBCT, UBCT and LBCT, as we believe that their names accurately represent the concept.

Definition 4.2. The *Extended Boomerang Connectivity Table* (EBCT) of a ℓ-bit S-box S is defined as a quadridimensional table of size $\ell \times \ell \times \ell \times \ell$ such that:

$$\text{EBCT}[a, b, c, d] = \left| \left\{ x \in \mathbb{F}_2^\ell \middle| \begin{array}{l} S(x) + S(x + a) = b \\ S(x) + S(x + c) = d \\ S^{-1}(S(x) + d) + S^{-1}(S(x + a) + d) = a \end{array} \right\} \right|.$$

Definition 4.3. The *Upper Boomerang Connectivity Table* (UBCT) of a ℓ-bit S-box S is defined as a tridimensional table of size $\ell \times \ell \times \ell$ such that:

$$\text{UBCT}[a, b, d] = \left| \left\{ x \in \mathbb{F}_2^\ell \middle| \begin{array}{l} S(x) + S(x + a) = b \\ S^{-1}(S(x) + d) + S^{-1}(S(x + a) + d) = a \end{array} \right\} \right|.$$

Definition 4.4. The *Lower Boomerang Connectivity Table* (LBCT) of a ℓ-bit S-box S is defined as a tridimensional table of size $\ell \times \ell \times \ell$ such that:

$$\text{LBCT}[a, c, d] = \left| \left\{ x \in \mathbb{F}_2^\ell \middle| \begin{array}{l} S(x) + S(x + c) = d \\ S^{-1}(S(x) + d) + S^{-1}(S(x + a) + d) = a \end{array} \right\} \right|.$$

The properties of these tables are described in [DDV20] but will not be explained in details in this thesis. We will use these tables in Section 4.4.6.

When E_m is composed of several rounds, the analysis of three works [SQH19; WP19; DDV20] is to fix a *2-differential trail* in E_m: throughout E_m, we keep track of the (potentially truncated) differences of $X + X'$ and $X + \bar{X}$, assuming that $X + X' = \bar{X} + \bar{X}'$ throughout E_m. Similarly to standard differential trails or truncated differential trails, the transition probabilities of the 2-differential trail in E_m can be computed around each Sbox layer, under the Markov Cipher assumption, using the three tables aforementioned. We refer to [DDV20, Section 3.2] for a deeper analysis of the boomerang probability in this framework.

It is good to note that a generalization of the 2-differential trail exists, where $X + X' = \bar{X} + \bar{X}'$ is not assumed throughout E_m. These trails are called 3-differential trails, and have been analyzed by very recent works [KT22; WSW+23].

4.3 Boomerang Attacks on AES in the Literature

In this section, we present an overview of notable boomerang attacks on AES. The first boomerang attack on 5-round and 6-round AES, presented by Biryukov in 2004 [Bir04], is detailed in Section 4.3.1. The yoyo attack, a chosen plaintext and adaptatively chosen ciphertext attack against AES presented by Rønjom *et al.* in 2017 [RBH17], is detailed in Section 4.3.2. The yoyo attack offers a surprising attack on 4-round AES with a data complexity of 4, and we stress out that it is related to the boomerang attack. In particular, the retracing boomerang attack, presented in 2020 by Dunkelman *et al.* [DKR+20], revisits and extends the yoyo attack to yield boomerang attacks against 5-round and 6-round AES; it is presented in Section 4.3.3. Finally, we shortly describe other similar attacks against AES, such as the Exchange attack [BR19; Bar19], the Boomeyong attack [RSP21], and a 7-round AES attack using related differences [BR22a].

Unified analysis. In this section, we will analyse all attacks with the same notations, for readability concerns. In all attacks, we divide E in two part $E = E_1 \circ E_0$, and consider the sets of differences $\mathcal{D}^0_{\text{in}}$, $\bar{\mathcal{D}}^0_{\text{in}}$, $\mathcal{D}^0_{\text{out}}$, $\mathcal{D}^1_{\text{in}}$, $\mathcal{D}^1_{\text{out}}$, and the following following truncated differentials:

$$\mathcal{D}^0_{\text{in}} \xrightarrow[E_0]{\vec{p}} \mathcal{D}^0_{\text{out}}, \qquad \mathcal{D}^0_{\text{out}} \xrightarrow[E_0^{-1}]{\vec{\bar{p}}} \bar{\mathcal{D}}^0_{\text{in}},$$
$$\mathcal{D}^1_{\text{out}} \xrightarrow[E_1^{-1}]{\vec{\bar{q}}} \mathcal{D}^1_{\text{in}}.$$

Also, we consider the truncated boomerang switch probability r defined as:

$$r = \Pr\left[E_0(\overline{P}) + E_0(\overline{P'}) \in \mathcal{D}^0_{\text{out}} \;\middle|\; \begin{array}{r} E_0(P) + E_0(P') \in \mathcal{D}^0_{\text{out}} \\ E_1^{-1}(C) + E_1^{-1}(\overline{C}) \in \mathcal{D}^1_{\text{in}} \\ E_1^{-1}(C') + E_1^{-1}(\overline{C'}) \in \mathcal{D}^1_{\text{in}} \end{array} \right].$$

Under independance assumptions, the event $E_1^{-1}(C) + E_1^{-1}(\overline{C}) = E_1^{-1}(C') + E_1^{-1}(\overline{C'})$ happens with probability $|\mathcal{D}^1_{\text{in}}|^{-1}$, and implies that $E_0(\overline{P}) + E_0(\overline{P'}) = E_0(P) + E_0(P') \in \mathcal{D}^0_{\text{out}}$, therefore we consider that $r \geqslant |\mathcal{D}^1_{\text{in}}|^{-1}$. In some cases, when independance assumption do not hold, the probability r can be equal to 1. We discuss this further in the yoyo and retracing attacks of Section 4.3.2 and Section 4.3.3.

Using these notations, the boomerang probability can be estimated:

$$p_b = \vec{p}^2 \bar{p}^2 r \bar{q}^2.$$

Note that this analysis is sufficient for the boomerang attacks on AES that we present in this section, but we improve upon it using plaintext and ciphertext structures in the truncated boomerang framework of Section 4.4.

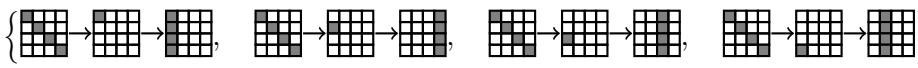
4.3.1 Biryukov's Original Boomerang Attack

In 2004, Biryukov presented the first boomerang attack on AES. He started with a distinguisher on 5-round AES, and extended it to an attack on 6-round AES.

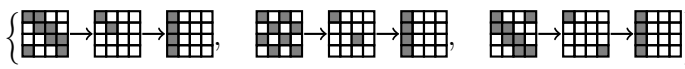
4.3.1.1 A 5-round boomerang key-recovery attack against AES

Biryuokov's boomerang attack on 5-round AES decomposes the cipher E into $E = E_1 \circ E_0$; E_0 is composed of 3 AES rounds, and E_1 of 2 AES rounds. The trail is depicted in Figure 4.4. Note that in standard boomerang characteristics, the top part represents the differences $P + P'$ and $\bar{P} + \bar{P}'$, while the bottom part represents the differences $P + \bar{P}$ and $P' + \bar{P}'$; this will be used in all subsequent boomerang characteristics.

In his work, Biryukov used multiple trails in the upper forward trail. Instead of requiring the first byte to be active on state $w_0 + w_0'$, any byte on the first column of $w_0 + w_0'$ can be active, as long as a single one is active. In short, four different patterns are accounted for $x_0 + x_0', x_1 + x_1'$, and $x_2 + x_2'$ (recall that $x_i + x_i'$ is the difference $P + P'$ on state x_i):



Similarly, the returning pair states $\bar{x_2} + \bar{x_2}', \bar{x_1} + \bar{x_1}'$, and $\bar{x_0} + \bar{x_0}'$ might take 3 different patterns (we ask for $\bar{P} + \bar{P}'$ to be active on the first diagonal). If we suppose that $x_2 + x_2'$ is active on the first column:



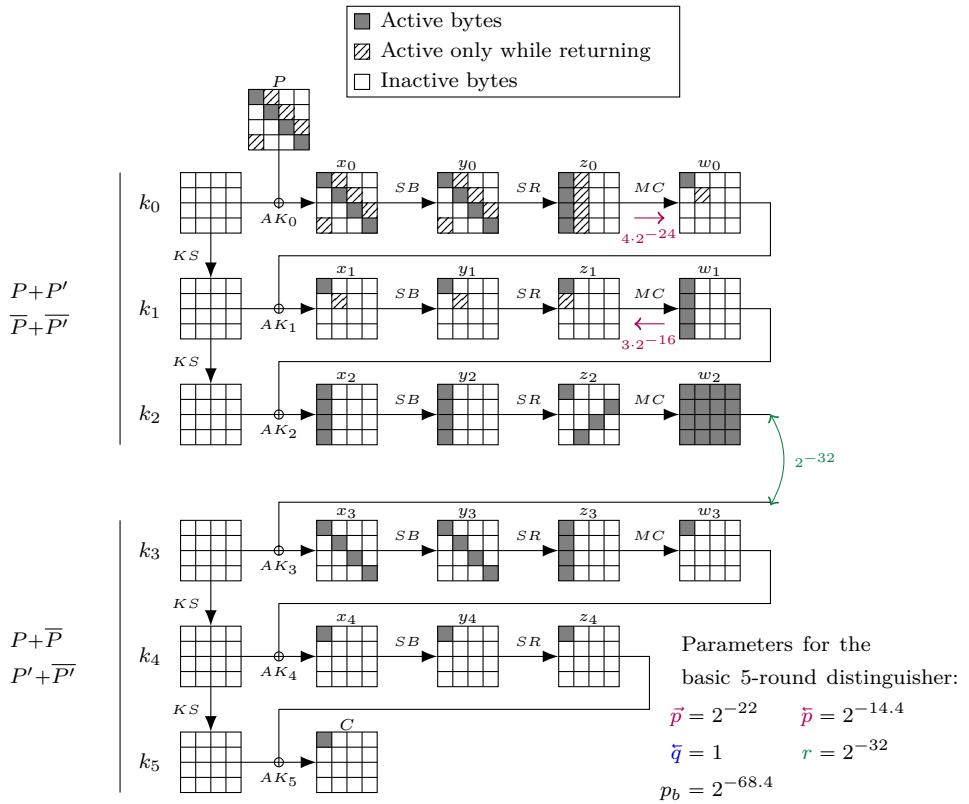


Figure 4.4: Biryukov's boomerang characteristic on 5-round AES.

If $x_2 + x_2'$ is active on a different column, 3 different patterns with 2 active S-boxes in $\bar{x_1} + \bar{x_1}'$ would be used. Note that we require $\bar{P} + \bar{P}'$ and $P + P'$ to have a common active diagonal. This assumption was not performed by Biryukov. Although it reduces the number of different patterns for the returning pair from 6 to 3, it greatly simplifies the analysis and only slightly decreases the success probability. For additional details on the original attack, we refer to [Bir04].

The boomerang probability is:

$$\vec{p} = 2^{-22}, \qquad \overleftarrow{p} = 2^{-14.4}, \qquad p_b = \vec{p}\,\overleftarrow{p}\,r\overleftarrow{q}^2 = 2^{-68.4}.$$

Biryukov's attack on 5-round AES can be performed by repeating the following process 2^6 times:

1. Select a random plaintext P and encrypt a structure $P + \Delta_{\text{in}}$ for $\Delta_{\text{in}} \in \mathcal{D}_{\text{in}}$, where $\mathcal{D}_{\text{in}}$ is the set of differences over the main diagonal. We denote the plaintexts P_i for $0 \leqslant i \leqslant 2^{32} - 1$ and $C_i = E(P_i)$.
2. Shift C_i by a fixed difference ∇_{out} over the first byte: $\bar{C}_i = C_i + \nabla_{\text{out}}$.
3. For $0 \leqslant i \leqslant 2^{32} - 1$, decrypt $\bar{C}_i$: $\bar{P}_i = E^{-1}(\bar{C}_i)$.

4. Look for i, j such that $\bar{P}_i + \bar{P}_j$ is inactive on two diagonals (excluding the main diagonal). The collisions can be found efficiently using a hash table. If no collision is found, go back to step 1.

5. For each collision found, verify that it is not a wrong quartet. To do so, guess the 32-bit main diagonal of k_0, and partially encrypt one round of $P_i, P_j, \bar{P}_i, \bar{P}_j$. Then check wether states w_0 of pairs (P_i,P_j) and $(\bar{P}_i,\bar{P}_j)$ each have a single active byte in the same position. This gives a 46-bit filter (Biryukov wrongly claimed a 44-bit filter). If so, return the 32-bit diagonal value of k_0.

Complexity analysis Generate about 2^6 structures of 2^{32} elements. In each structure, there are approximately 2^{63} quartets $(P_i, P_j, \bar{P}_i, \bar{P}_j)$ for $0 \leqslant i < j \leqslant 2^{32} - 1$. Therefore, there is a total of $2^{6+63} = 2^{69}$ quartets. The probability that a quartet randomly possesses 2 inactive input diagonals, including the main diagonal, is $2^{-128} \times 3 \times 2^{64} \approx 2^{-62.4}$. Therefore, there are approximately $2^{69-62.4} = 2^{6.6}$ wrong quartets, and $2^{69} p_b = 1.5$ right quartets. Each wrong quartet has a probability $2^{32-46} = 2^{-14}$ of detecting a key candidate which gives the right patterns in w_0. Therefore, the probability of false positives is $2^{6.6-14} = 2^{-7.4}$. The number of right quartets follows a Poisson distribution with parameter $\lambda = 1.5$. The probability that no right quartet is found is given by $\Pr[\text{Poisson}(1.5) = 0] \approx 0.22$ Therefore, with probability 0.78, we recover the first diagonal of k_0 and can proceed to the attack on another diagonal, using the knowledge on the first diagonal of k_0 to improve the complexity of subsequent attacks. The time complexity of step 5. is negligible: instead of looping through 2^{32} key candidates, we can loop through the set of 2^{10} possible differences for the pair (P_i, P_j) in state y_0, and determine the 2^{10} keys by fetching values corresponding to non-zero entries in the DDT table of the AES Sbox from the differences in input and ouptut of the Sbox layer from x_0 to y_0.

All in all, this attack requires 2^{38} encryption queries and 2^{38} adaptative decryption queries. The time complexity is bounded by the data complexity: 2^{39}. The memory complexity is 2^{32} to store each structure. This gives:

$$(D, T, M) = (2^{39}, 2^{39}, 2^{32}).$$

In this variant, we require that $\bar{P} + \bar{P}'$ is active on the main diagonal to make the key-recovery simpler; this was not assumed in Biryukov's original attack. This lowers the success probability from $1 - \Pr[\text{Poisson}(3) = 0] \approx 0.95$ to 0.78.

4.3.1.2 Extension to a 6-round key-recovery attack on AES

Biryukov proposes to guess a full 32-bit anti-diagonal of k_6 in order to build data corresponding to the 5-round attack. In this case, Biryukov only considers quartets such that $P + P'$ and $\bar{P} + \bar{P}'$ have a common active diagonal, as considered in our analysis. In short, the 6-round attack can be described by iterating the following:

1. Guess the first anti-diagonal of k_6 (2^{32} guesses).
2. For each guess, run the 5-round attack with 2^7 structures. Step. 2 of the 5-round attack is performed by partially decrypting C_i using the guessed key material.
3. Stop when the first diagonal of k_0 has been suggested twice for the same key k_6.

Biryukov increases the number of structures from 2^6 to 2^7, so that we expect on average 3 good quartets (and therefore 3 correct suggestions for the first diagonal of k_0) for the right key guess of k_6. On the other hand, we expect $2^{7.6}$ wrong quartets for each of the 2^{32} key guesses of k_6, each suggesting on average 2^{-14} wrong key candidate for the first diagonal of k_0 (see the 5-round attack). In total, there are $2^{32+7.6-14} = 2^{25.5}$ wrong candidates 64 key bits (the main diagonal of k_0 and the first anti-diagonal of k_6). Since $2^{25.5} \ll 2^{32}$, we do not expect a 64-bit candidate to be wrongfuly suggested twice. We expect at least 2 good quartets with probability $\Pr[\text{Poisson}(3) \geqslant 2] = 0.8$, which both point to the correct 64-bit key value. The only 64-bit key candidate suggested at least twice is the correct 64-bit key material.

This attack requires 2^{39} encryption queries and 2^{71} decryption queries, and the time complexity is bounded by the data complexity. The memory complexity is 2^{33}: 2^{32} to store each structure and 2^{32} to keep in memory the queried ciphertexts between each guess of k_6. This gives:

$$(D, T, M) = (2^{71}, 2^{71}, 2^{33}).$$

4.3.2 The Yoyo Attack

The yoyo attack was introduced by Rønjom *et al.* [RBH17] at ASIACRYPT 2017. This attack has direct application to AES, and leads to the fastest distinguishers against 3-round, 4-round and 5-round AES. In 2017, the yoyo attack was also the fastest key-recovery attack against 5-round AES, which was then improved by the retracing boomerang framework [DKR+20]. The aim of this subsection is to give an overview of the yoyo attack without diving into the heavy notations of the original paper [RBH17].

4.3.2.1 Core idea

The yoyo analysis is straightforward on a permutation F on $\mathbb{F}_2^{\ell \times t}$, which has the following form:

$$F = S \circ A \circ S,$$

where S is an S-box layer (an ℓ-bit S-box is applied in parallel to all words of the state) and A any affine layer.

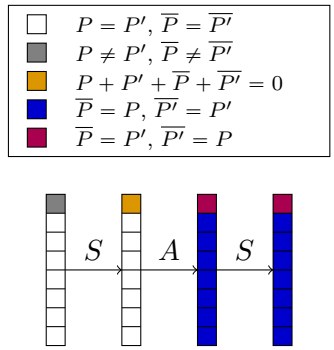


Figure 4.5: Yoyo distinguisher on a $S \circ A \circ S$ permutation.

Let us consider a difference Δ active only on the first word (corresponding to the input of the first S-box). Let $C = F(P)$ and $C' = F(P + \Delta)$. Let ∇ be the difference active only on the first word with $\nabla[0] = C[0] + C'[0]$. The yoyo distinguisher states that $F^{-1}(C + \nabla) + F^{-1}(C' + \nabla)$ is active only on the first word with probability 1. This can be extended to all non-trivial input/output activity patterns. For instance, Δ may be active on any non-trivial subset of words, and ∇ can be the difference of C and C' restricted to any non-trivial subset of words, potentially different from the subset of Δ.

In short, we exchange certain words of C and C' and decrypt to get $\bar{P}$ and $\bar{P}'$; the inactive words of $P + P'$ are also inactive words of $\bar{P} + \bar{P}'$. The yoyo distinguisher is depicted in Figure 4.5.

Let us define $\bar{C} = C + \nabla$ and $\bar{C}' = C' + \nabla$. For each word of index $i \in [\![0, t-1]\!]$, the following holds:

$$\{C[i], C'[i]\} = \{\bar{C}[i], \bar{C}'[i]\}.$$

Since S is applied independantly on all words of the state, this implies that for $i \in [\![0, t-1]\!]$:

$$\{S^{-1}(C)[i], S^{-1}(C')[i]\} = \{S^{-1}(\bar{C})[i], S^{-1}(\bar{C}')[i]\}.$$

This implies that:

$$S^{-1}(C) + S^{-1}(C') + S^{-1}(\bar{C}) + S^{-1}(\bar{C}') = 0.$$

By applying the linear layer L^{-1} and remarking that $L^{-1} \circ S^{-1}(C) = S(P)$, we get:

$$S(P) + S(P') + S(\bar{P}) + S(\bar{P}') = 0.$$

Therefore, on words where $P = P'$, $S(P)+S(P')$ is also inactive, so $S(\bar{P})+S(\bar{P}')$ is inactive, and finally $P = P'$.

Alternatively, the yoyo attack on a $S \circ A \circ S$ permutation can be interpreted as a boomerang attack where $E_0 = A \circ S$, $E_1 = S$ and $E_m = \mathrm{Id}$, with $\mathcal{D}^0_{\mathrm{in}} = \{\Delta\}$,

$\mathcal{D}^0_{\text{out}} = A \times \mathcal{D}$, $\bar{\mathcal{D}}^0_{\text{in}} = \mathcal{D}$ where $\mathcal{D}$ is the set of all differences on $\text{supp}(\Delta)$, $\mathcal{D}^1_{\text{in}}$ the vector space of all differences on $\text{supp}(\nabla)$, and $\mathcal{D}^1_{\text{out}} = \{\nabla\}$. The parameters are $\vec{p} = \bar{p} = \bar{q} = r = 1$. The fact that $r = 1$ comes from the fact that the two pairs $(C, \bar{C})$ and $(C', \bar{C}')$ are not independant.

4.3.2.2 Direct application: a distinguisher on 4-round AES

4-round AES can be distinguished from a random permutation by fixing $S = \text{SubBytes} \circ \text{AddRoundKey} \circ \text{MixColumns} \circ \text{SubBytes}$ to a AES Super S-box [DR06]. Recall that the super S-box representation of the AES was discussed in Section 2.3.2.3. The Super S-box layer maps a column to a column (it can also be seen as a map from diagonals to anti-diagonals but we prefer to include the two ShiftRows operation inside the affine layer). In this case, $A = \text{ShiftRows} \circ \text{AddRoundKey} \circ \text{MixColumns} \circ \text{ShiftRows}$.

The algorithm is the following:

1. Generate a random P, and ask for the encryption $C = E(P)$ and $C' = E(P + \Delta)$ with Δ only active on the first diagonal. Let us denote $C = (C[0], C[1], C[2], C[3])$, where $C[i]$ is the i-th anti-diagonal of C, and let use have similar notations for C'.

2. Define $\bar{C} = (C'[0], C[1], C[2], C[3])$, $\bar{C}' = (C[0], C'[1], C'[2], C'[3])$, and ask for the decryption of $\bar{P} = E^{-1}(\bar{C})$ and $\bar{P}' = E^{-1}(\bar{C}')$.

3. If $\bar{P} + \bar{P}'$ is inactive on the three last diagonals, the encryption is 4-round AES, else it is a random permutation.

This gives:

$$(D, T, M) = (4, 4, 2).$$

4.3.2.3 Extension to a 5-round distnguishing attack on AES

The basic 4-round AES distinguisher can be extended to a 5-round AES distinguisher by adding one round at the beginning, starting with plaintexts P, P' with $P + P'$ active on a single diagonal. With probability $2^{-13.4}$, the difference retracts to only 2 active bytes after the first MixColumns operation. In this case, when the yoyo returns, the state is active on only 2 anti-diagonals after the first MixColumns operation. In particular, because of the MDS property of the AES MixColumns matrix, it is impossible for the plaintext difference to be inactive on 2 bytes of the same column. A naive 5-round yoyo distinguisher can be obtained by generating a lot of plaintext pairs (P, P') with such differences, and counting the number of returning yoyo pairs $\bar{P}, \bar{P}'$ whose difference is inactive on at least 2 bytes the same column. While this happens with probability $2^{-11.4}$ for a random permutation, it happens with probability $2^{-11.4} \times (1 - 2^{-13.4})$ for 5-round AES. Following [MS02], the two cases can be distinguished with roughly $2^{-11.4} \times 2^{2\times -13.4} = 2^{-38.2}$ samples.

[RBH17] describes a more advanced attack using sets of friend plaintext pairs, which have the following property: if one of the pairs has a state difference retracting to only 2 active anti-diagonals after one round, then all the friend plaintext pairs also do. It is sufficient to generate $2^{13.4}$ sets of friend plaintext pairs, and check in each one wether no yoyo returns with 2 inactive bytes in the same column (which should happen with probability $2^{-11.4}$ for a random permutation). This leads to a distinguisher with complexity roughly $2^{1+11.4+13.4} = 2^{25.8}$.

The complexity of the distinguisher becomes:

$$(D, T, M) = (2^{25.8}, 2^{25.8}, 4).$$

More details can be found on the original yoyo paper [RBH17].

4.3.3 The Retracing Boomerang Attack

The retracing boomerang attack is a framework improving on the yoyo attack and presented by Dunkelman *et al.* at EUROCRYPT 2020 [DKR+20]. To this day, the framework results in the fastest key-recovery attack against 5-round AES, with or without secret S-boxes, and produces interesting attacks on 6-round AES. Although the retracing boomerang key-recovery attacks on 5-round AES are very interesting, they involve advanced technique to lower the key-recovery complexity. In this subsection, we will therefore only present the retracing boomerang distinguisher on 5-round AES (as defined by [GRR17]), and its extension to a key-recovery attack on 6-round AES, presented in [DKR+19, Appendix C].

4.3.3.1 Core idea

The retracing boomerang makes use of correlated values in the ciphertext side such that the independance assumption between equations of Equation 4.1 does not hold. Instead, the two pairs $(C, \bar{C})$ and $(C', \bar{C}')$ simultaneously follow the lower differential $\mathcal{D}^1_{\text{out}} \xrightarrow[E_1^{-1}]{\overleftarrow{q}} \mathcal{D}^1_{\text{in}}$ with probability $\overleftarrow{q}$. The events are therefore correlated positively and this increases the boomerang probability to $p_b = \vec{p}\,\overleftarrow{p}\,r\overleftarrow{q}$.

In the case of AES, interesting 2-round or 3-round differential trails are only active on one anti-diagonal of the output. For AES, the main idea of the retracing boomerang is to define ∇_{out} active on a single anti-diagonal, such that the returning pairs $(C, C + \nabla_{\text{out}})$ and $(C', C' + \nabla_{\text{out}})$ have the same pair of values on the active anti-diagonal: i.e. either $C = C'$ or $C = C' + \nabla_{\text{out}}$ on the active anti-diagonal.

In their framework, Dunkelman *et al.* propose two types of retracing boomerangs, which we describe for AES:

- The mixing retracing boomerang: chose ∇_{out} active on an anti-diagonal, such that $\nabla_{\text{out}} = C + C'$ on the active anti-diagonal. The mixing retracing boomerang is very close to the yoyo attack. In that case, $C = \bar{C}'$ and $C' = \bar{C}$ on the anti-diagonal.

- The shifting retracing boomerang: only keep pairs (C, C') such that $C = C'$ on an anti-diagonal, and take any ∇_{out} active only on the corresponding anti-diagonal. In that case, $C = C'$ and $\bar{C} = \bar{C}'$ on the anti-diagonal.

4.3.3.2 Application: a distinguisher attack on 5-round AES.

The retracing boomerang allows to improve on Biryukov's key-recovery boomerang attack on 5-round AES [Bir04] presented in Section 4.3.1. We present a distinguishing attack resulting from this improvement. In this attack, the authors of [DKR+20] proposed to use both the mixing and shifting variants. However, for subtles reasons, this characteristic is incompatible with the shifting variant (see Section 4.5.3)[1], and we consider the mixing variant instead. The 5-round retracing boomerang is represented on Figure 4.6.

Compared to Biryukov's attack, this attack carefully selects the difference ∇_{out} active on the first anti-diagonal such that $\nabla_{\text{out}} = C' + C$ on the first anti-diagonal, and thus $C = \bar{C}'$ and $C' = \bar{C}$ on the first diagonal. This implies that $w_2 + w_2' + \bar{w}_2 + \bar{w}_2{}' = 0$, and the same property holds for state z_2. Thus, the choice of difference ∇_{out} increases the boomerang switch probability from $r = 2^{-32}$ to $r = 1$, which results in a boomerang probability increase from $p_b = 2^{-68.5}$ to $p_b = 2^{-36.5}$. Therefore, the attacker can use a larger set $\bar{\mathcal{D}}^0_{\text{in}}$, and Dunkelman *et al.* relax the constraint on z_1 for the returning pair to 3 active bytes. After decryption, $\bar{P} + \bar{P}'$ should be inactive on 1 diagonal (instead of 2 in Biryukov's attack). This further increases the boomerang probability from $p_b = 2^{-36.5}$ to $p_b = 2^{-28}$. The corresponding parameters are:

$$\vec{p} = 2^{-22}, \qquad \overleftarrow{p} = 2^{-6}, \qquad r = 1,$$
$$\overleftarrow{q} = 1, \qquad p_b = \vec{p}\,\overleftarrow{p}\,r\,\overleftarrow{q} = 2^{-28}.$$

The distinguisher consists in encrypting a structure of $2^{15.5}$ plaintexts, yielding 2^{30} pairs of ciphertexts (C, C')[2]. For each of the 2^{30} pairs (C, C'), we check whether $E^{-1}(C + \nabla_{\text{out}}) + E^{-1}(C' + \nabla_{\text{out}})$ has an inactive diagonal, where ∇_{out} is equal to the first anti-diagonal of $C + C'$. On average, we expect $2^{30} \times 4 \times 2^{-32} = 1$ wrong boomerang quartets for a random permutation, and $2^{30} \times 2^{-28} = 4$ for 5-round AES. We distinguish AES when the number of quartets is 2 or more.

The time complexity is bounded by the 2^{30} decryptions of pairs of ciphertexts, and the memory is bounded by the storage of the $2^{15.5}$ ciphertexts. This gives:

$$(D, T, M) = (2^{31}, 2^{31}, 2^{15.5}).$$

[1] In short, the shifting variant characteristic would have white cells instead of green cells in the forward characteristic of Figure 4.6, leading to an impossible transition from z_2 to w_2.

[2] Note that in the retracing boomerang attack from [DKR+19, Appendix C.2], the authors fix a 32-bit difference δ_L and discard ciphertexts (C, C') such that $C + C' \neq \delta_L$; we do not add this condition in our analysis since it is not useful. It does not change the time complexity, since the bottleneck comes from the decryptions, but our version decreases the memory complexity compared to their attack.

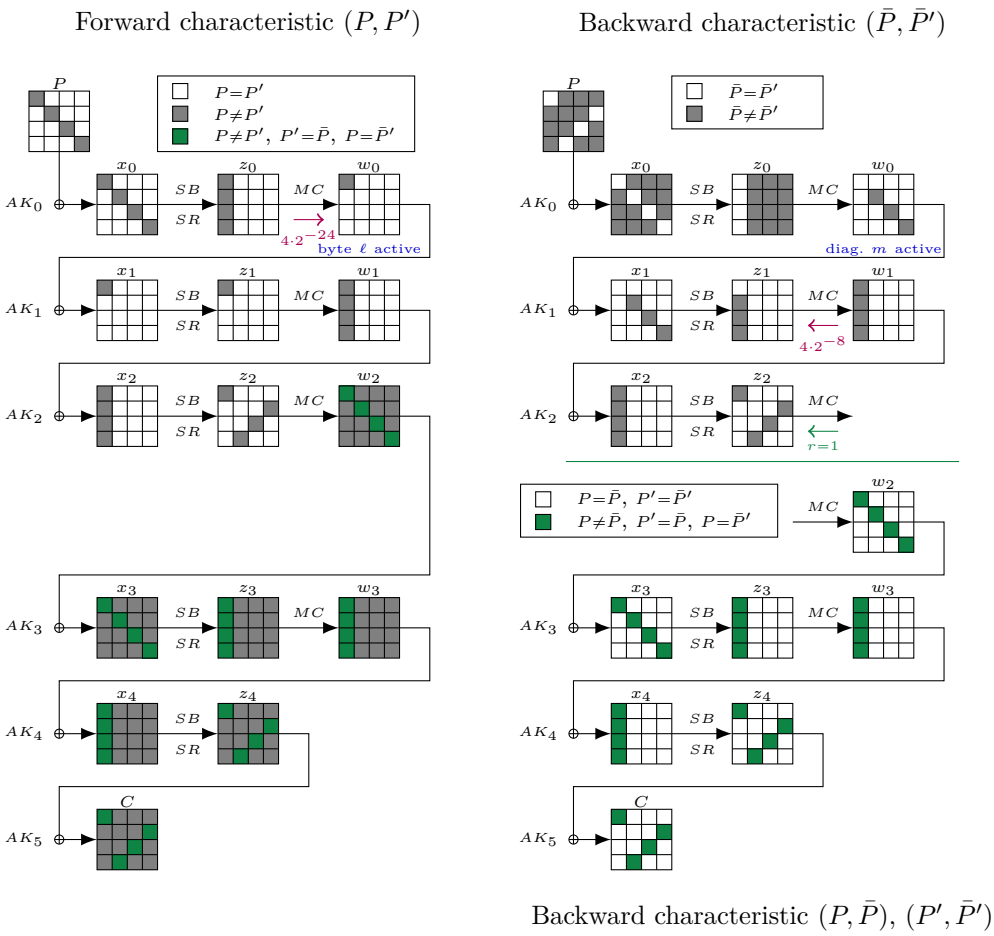


Figure 4.6: Retracing boomerang on 5-round AES (distinguisher). Instead of the standard boomerang representation, this characteristic depicts a full trail for the forward pair (P, P') and a different trail for the upper part of the returning pair $(\bar{P}, \bar{P}')$.

4.3.3.3 Application: key-recovery attack on 6-round AES

A direct extension of the distinguishing attack from 5-round AES is hard to perform, since the 16 subkey bytes of the last round key are required to compute the anti-diagonal "exchange" one round before the ciphertexts. Instead, Dunkelman *et al.* use the shifting variant of the retracing boomerang. In the key-recovery scenario, it is easier to filter boomerang quartets on the plaintext side; even if a returning pair is fully active, it is possible to deduce key candidates that satisfy the correct pattern after one round. Because of this, Dunkelman *et al.* chose to relax the constraints on the upper pair differential trail: $P + P'$ can have 3 active bytes in state w_0 (instead of 1 for the distinguishing attack on 5-round AES), which increases $\vec{p}$ from 2^{-24} to 2^{-6}. With this relaxation, the shifting variant incompatibility of Section 4.5.3 is not a problem.

Since our results of Section 4.5 are heavily inspired by this attack, we explain this attack in details, using the characteristics from Figure 4.7. We describe the algorithm in Algorithm 4.1.

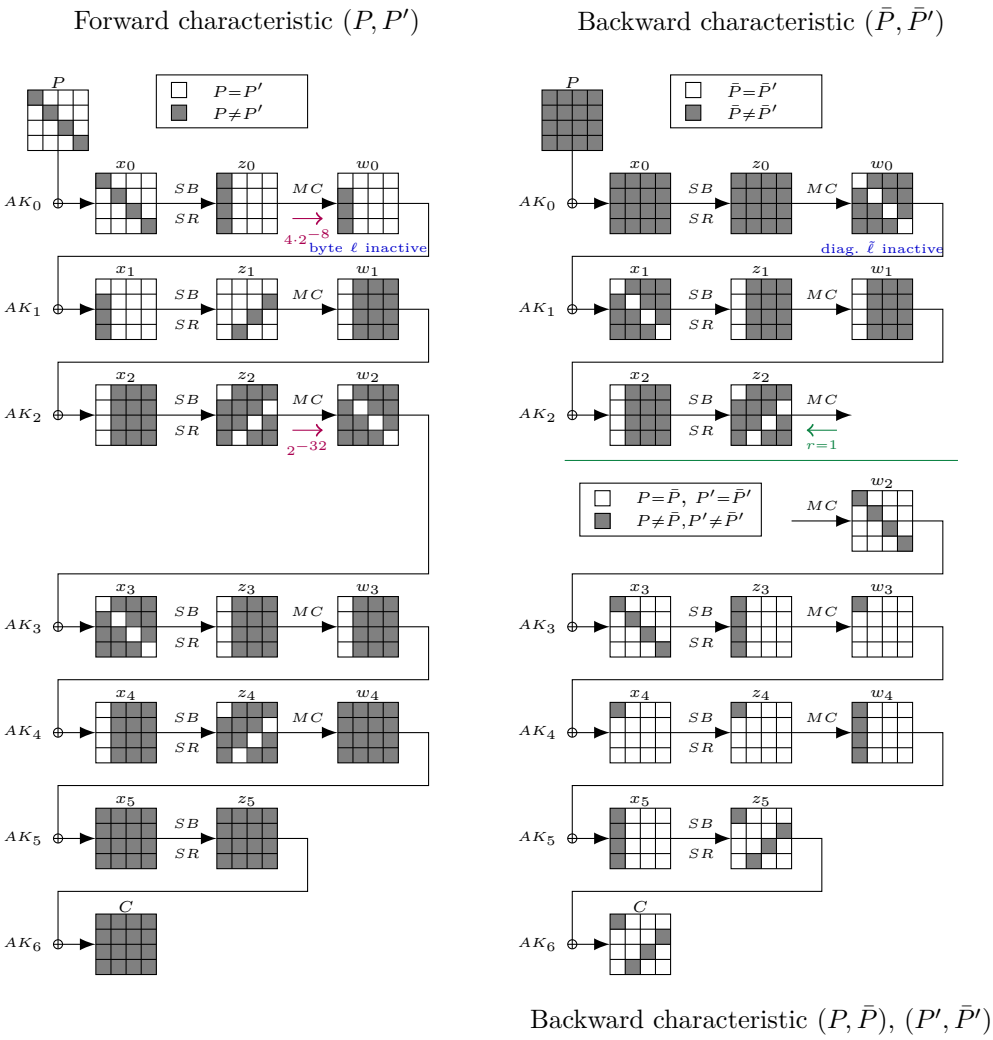


Figure 4.7: Retracing boomerang on 6-round AES (key-recovery).

Characteristics analysis. Starting from a pair (P, P') with an active diagonal, we consider that it is a *right forward pair* if it follows the forward characteristic of Figure 4.7. We require that one byte is inactive at the end of the first round (probability 2^{-6}), and that the main diagonal is inactive at the end of the third round (probability 2^{-32}). Therefore a forward pair (P, P') is right with probability $\vec{p} = 2^{-38}$.

In the backwards direction, $\bar{C}$ and $\bar{C}'$ are obtained by shifting C and C' with a difference in the main antidiagonal. We assume that the difference collapses to

only the first byte of the state after one round (state z_4), and that the difference is the same for the pairs $(C, \bar{C})$ and $(C', \bar{C}')$. This happens with probability $2^{-2\times 24-8} = 2^{-56}$ with random ciphertexts, but the attacker will guess the value of key bytes $k_6[0, 7, 10, 13]$ in order to directly construct values $\bar{C}$ and $\bar{C}'$ with this property. Under the right guess of $k_6[0, 7, 10, 13]$, this gives $\overleftarrow{q} = 1$.

Starting from z_4, we have a shifting retracing boomerang: C and C' have the same value on the main antidiagonal, and $\bar{C}$ and $\bar{C}'$ differ from C and C' by the same difference in a single byte of the main antidiagonal. Thefore the active S-box transition from z_4 to x_4 is the same for the pairs $(C, \bar{C})$ and $(C', \bar{C}')$: $x_4 + x'_4$ and $\bar{x}_4 + \bar{x}'_4$ are both inactive in the first column.

This implies that $z_3[j] = z'_3[j]$ and $\bar{z}_3[j] = \bar{z}'_3[j]$ for $j = 0, 1, 2, 3$, and the transitions from z_3 to x_3 are also the same for both returning pairs. In particular, $S^{-1}(z_3[j]) + S^{-1}(\bar{z}_3[j]) = S^{-1}(z'_3[j]) + S^{-1}(\bar{z}'_3[j])$ for $j = 0, 1, 2, 3$. The difference $z_3 + \bar{z}_3$ (resp. $z'_3 + \bar{z}'_3$) being active only in the first column, we obtain $x_3 + \bar{x}_3 = x'_3 + \bar{x}'_3$ and $z_2 + \bar{z}_2 = z'_2 + \bar{z}'_2$ with probability one. This shows that $r = 1$ (under the right guess of $k_6[0, 7, 10, 13]$).

When considering the pair $(\bar{C}, \bar{C}')$, we deduce that the difference in z_2 is the same as in the forward pair: $\bar{z}_2 + \bar{z}'_2 = z_2 + z'_2$. In particular, one antidiagonal is inactive in z_2; this implies that one diagonal of w_0 is inactive, with $\overleftarrow{p} = 1$.

To summarize: assuming that (P, P') is a right forward pair, and that $\bar{C}, \bar{C}'$ are constructed such that $z_4 + \bar{z}_4 = z'_4 + \bar{z}'_4$ with this difference only active in byte 0, then with probability 1 a diagonal of w_0 is inactive for the pair $\bar{C}, \bar{C}'$ (the same diagonal that is inactive for (P, P')).

Seen as a simple boomerang characteristic, this would give a boomerang probability of $p_b = \vec{p}\,\overleftarrow{p}\, r\,\overleftarrow{q} = 2^{-38}$. However, some filtering is strongly needed; as is, returning pairs can not be detected since all the input state is active. Instead, the retracing boomerang attack uses multiple returning pairs which are all inactive on a diagonal of w_0 for a right forward pair and under the right key guess of $k_6[0, 7, 10, 13]$, and performs an efficient key-recovery on top of it.

Attack description. The attack proceeds as follows, with 8 fixed values $\nabla_0, \ldots \nabla_7$ active only on byte 0:

1. Ask for the encryption of a structure of 2^{20} plaintexts with different values on the main diagonal.

2. For each candidate K for $k_6[0, 7, 10, 13]$, partially decrypt the ciphertexts to compute $Y = MC^{-1}(x_5)$, define new ciphertexts $\bar{C}_i$ such that $\bar{Y}_i = Y + \nabla_i$ and query and store their corresponding plaintexts $\bar{P}_i$.

3. For each candidate K for $k_6[0, 7, 10, 13]$, filter all pairs (P, P') such that $Y[0] = Y'[0]$.

   For each such pair, consider the 8 quartets $(Y, Y', \bar{Y}_i, \bar{Y}'_i)$ with $\bar{Y}_i = Y + \nabla_i$ and $\bar{Y}'_i = Y' + \nabla_i$, whose corresponding plaintexts $\bar{P}_i, \bar{P}'_i$ were queried during step 2.

For each $\ell \in \{0, 1, 2, 3\}$, assume that the ℓ-th diagonal of w_0 is inactive for all quartets simultaneously and deduce the key k_0. If a candidate k_0 is compatible with all quartets, return it as the correct key.

Algorithm 4.1: Retracing boomerang attack on 6-round AES.

Query the encryption of 2^{20} plaintexts with different values on the main diagonal
for all $k_6[0, 7, 10, 13]$ **do**
  for all ciphertext C **do**
    Partially decrypt C to obtain $Y[0, 1, 2, 3]$
    for $0 \leqslant i < 8$ **do**
      Define $\bar{Y}_i = Y + \nabla_i, \bar{Y}'_i = Y' + \nabla_i$, compute corresponding $\bar{C}_i, \bar{C}'_i$
      Query $\bar{P}_i = E^{-1}(\bar{C}_i), \bar{P}'_i = E^{-1}(\bar{C}'_i)$
  for all pairs (C, C') with $Y[0] = Y'[0]$ **do**
    for $0 \leqslant \ell < 4$ **do**
      Assume $w_0[\ell]$ is inactive for all quartets $(Y, Y', \bar{Y}_i, \bar{Y}'_i)$
      Deduce key candidates for k_0

Complexity. With 2^{20} plaintexts, the expected number of right forward pairs is $2^{20} \cdot 2^{19} \cdot 2^{-38} = 2$. If there is a right forward pair (P, P'), then it satisfies $Y[0] = Y'[0]$. At step 3, with the correct key guess, all the quartets $(Y, Y', \bar{Y}_i, \bar{Y}'_i)$ follow the boomerang, hence w_0 has an inactive diagonal with probability 1. Therefore the correct key candidate will be recovered and the attack succeeds with high probability.

Reciprocally, a wrong guess of $k_0[0, 5, 10, 15] \| k_6[0, 7, 10, 13]$ and ℓ passes the test with probability $2^{-8\times 8}$; we expect on average $2^{39-8+32+32+2-64} = 2^{33}$ false positives. False positives are detected and discarded by recovering key candidates for another diagonal of k_0.

Step 3 iterates over 2^{32} keys, $2^{39} \cdot 2^{-8} = 2^{31}$ pairs for each key, and 4 values of ℓ. The complexity to recover the key candidates is estimated as equivalent to 2^{15} encryptions in [DKR+19]. Therefore the total time complexity is $2^{32+31+2+15} = 2^{80}$.

The data complexity is $2^{32} \cdot 2^{20} \cdot 8 = 2^{55}$ at step 2, and the memory complexity is $2^{20} \cdot 8$ to store the $\bar{P}_i$ and $\bar{P}'_i$ (step 2), resulting in:

$$(D, T, M) = (2^{55}, 2^{80}, 2^{23}).$$

We observe that the 2^{55} decryption query actually correspond to only $2^{20} \cdot 2^{32} = 2^{52}$ distinct ciphertexts. Therefore we can reduce the data complexity by storing all the queries:

$$(D, T, M) = (2^{52}, 2^{80}, 2^{52}).$$

4.3.4 Other Boomerang-like Attacks on AES

Other attacks on 6-round AES exist in the same spirit. We list them and give an overview of their contribution in this subsection, but do not dive into the details.

The exchange attack. The exchange attack [BR19; Bar19] uses quartet properties similar to that of the yoyo attack to yield efficient "key independant" distinguishers on AES in the Chosen Plaintext and Adaptatively Chosen Ciphertext models, with complexities of respectively $(2^{88}, 2^{88}, 2^{88})$ and $(2^{83}, 2^{83}, 2^{32})$. The ACC attack principle can be seen as a mixing retracing boomerang: a pair of plaintexts (P, P') differing in the main diagonal is encrypted to (C, C'), then some columns of C and C' are exchanged, and the resulting ciphertexts are decrypted. Their corresponding plaintexts are more likely than random to be active in only one diagonal.

The boomeyong attack. The boomeyong attack [RSP21] uses a 4-round yoyo distinguisher and appends one round the plaintext and on the ciphertext sides. The attack achieves a complexity of $(D, T, M) = (2^{80}, 2^{80}, 2^{28})$. The attack is similar to the ACC exchange attack, but is a key-recovery attack instead of a distinguishing attack. The attack can also be seen as a mixing retracing boomerang: a pair of plaintexts (P, P') differing in the main diagonal is encrypted to (C, C'), then the last columns of C and C' are exchanged, and the resulting ciphertexts are decrypted. If the returning pair has an inactive diagonal which is not the main diagonal, the corresponding quartet is used for key-recovery.

A 7-round attack using related differences. In 2022, Bardeh and Rijmen published an attack on 7-round AES based on related differences [BR22a]. Related differences are differences on a quartet such that in each byte, only two values are taken among the quartet. This implies that differentials between each pair of the quartets are related and not independant, and this dependance is exploited to increase the transition probabilities. Their attack may be interpreted as a rectangle attack using similar techniques to the retracing boomerang attack to increase the characteristic probability.

Existing boomerang-like attacks on 6-round AES were overviewed in this section. In the two next sections, we present our contributions: truncated boomerang attacks, and improved retracing boomerang attacks.

4.4 Truncated Boomerang Attacks

In this section, we present the framework of truncated boomerangs, which is a joint work with Gaëtan Leurent, presented in *EUROCRYPT* 2023 [BL23b].

Instead of first building a boomerang distinguisher and then appending extra key-recovery rounds, we consider the truncated boomerang attack as a whole, including the key-recovery exploiting the first and last round transitions. The framework integrates and improves on previous analyses, including structures of plaintexts and ciphertexts, and truncated differentials. Some boomerang attacks using truncated trails have been proposed on AES [Bir04] and Kiasu-BC [DL17], but they do not exploit the full potential of truncated trails: in particular, they only use structures on the plaintext side. The improvements of our framework

partly come from the systematic use of structures of ciphertexts. We also consider boomerang trails with smaller probability than the random case, and mount attacks by gathering enough samples to detect the bias.

In Section 4.4.1, we present the basic framework to mount distinguishing attacks, while in Section 4.4.2, we present a more in depth analysis of key-recovery attacks. In both sections, we give a concrete example of an application of the framework on 6-round AES, though these attacks are not optimized. In Section 4.4.3, we present improved distinguishing, key-recovery and secret-Sbox recovery attacks on 5-round and 6-round AES. The complexities are detailed and compared to previous works in Section 4.1. We then present our new attack on 8-round Kiasu-BC in Section 4.4.4, improving the state-of-the-art attack of [DL17], and we present the first marginal distinguisher of TNT-AES in Section 4.4.5.

In Section 4.4.6, we build a MILP model implementing our framework, to find good parameters for the full attack automatically. The model allows both fixed differences and truncated differences, and takes into account the key-recovery step, instead of just optimizing a boomerang distinguisher. Although the boomerang trails on AES and Deoxys-BC are quite different, the underlying analysis is the same. The code related to this work is available on github [BL23a].

In this section, the analysis slightly differs from our work in EUROCRYPT 2022 [BL23b]. In [BL23b], the analysis was performed in the case were the set of input differences (between P and P') is the same as the set of accepted differences for returning pairs (between $\bar{P}$ and $\bar{P}'$). Instead, we consider the case where these sets of differences might be different. This allows to apply some of our attacks more straightforwardly.

4.4.1 Truncated Boomerang Distinguisher

Let us split the cipher $E = E_1 \circ E_0$ and consider the following truncated differentials:

$$\mathcal{D}^0_{\text{in}} \xrightarrow[E_0]{\vec{p}} \mathcal{D}^0_{\text{out}}, \qquad \mathcal{D}^0_{\text{out}} \xrightarrow[E_0^{-1}]{\vec{p}} \bar{\mathcal{D}}^0_{\text{in}}, \qquad \mathcal{D}^1_{\text{out}} \xrightarrow[E_1^{-1}]{\vec{q}} \mathcal{D}^1_{\text{in}}.$$

We assume that $\mathcal{D}^0_{\text{in}}$ is a vector subspace of $\{0,1\}^n$ and $0 \notin \mathcal{D}^1_{\text{out}}$. The truncated boomerang attack proceeds as follows:

1. Choose a random plaintext P_0, and query the encryption oracle over the structure $P_0 + \mathcal{D}^0_{\text{in}}$; for each $i \in \mathcal{D}^0_{\text{in}}$, we define $P_i = P_0 + i$ and $C_i = E(P_i)$.

2. For each ciphertext C_i, query the decryption oracle over the set $C_i + \mathcal{D}^1_{\text{out}}$: for each $j \in \mathcal{D}^1_{\text{out}}$, we define $\bar{C}_i^{\,j} = C_i + j$ and $\bar{P}_i^{\,j} = E^{-1}(\bar{C}_i^{\,j})$.

3. Count the number of pairs $(\bar{P}_i^{\,j}, \bar{P}_{i'}^{\,j'})$ with $\bar{P}_i^{\,j} + \bar{P}_{i'}^{\,j'} \in \bar{\mathcal{D}}^0_{\text{in}}$ (and $i \neq i'$). When $\bar{\mathcal{D}}^0_{\text{in}}$ is a vector space, this can be done efficiently by projecting the plaintext values on the orthogonal complement of $\bar{\mathcal{D}}^0_{\text{in}}$ in $\{0,1\}^n$, and looking for collisions.

4. If needed, repeat steps 1 to 3 with different plaintext structures.

4.4.1.1 Analysis

We consider a potential quartet $(P, P', \bar{P}, \bar{P}')$ corresponding to $(C, C', \bar{C}, \bar{C}')$, with $P + P' \in \mathcal{D}^0_{\text{in}}$ and $C + \bar{C}, C' + \bar{C}' \in \mathcal{D}^1_{\text{out}}$. We have:

$$\begin{aligned}\Pr[E_0(P) + E_0(P') \in \mathcal{D}^0_{\text{out}}] &= \vec{p},\\ \Pr[E_1^{-1}(C) + E_1^{-1}(\bar{C}) \in \mathcal{D}^1_{\text{in}}] &= \overleftarrow{q},\\ \Pr[E_1^{-1}(C') + E_1^{-1}(\bar{C}') \in \mathcal{D}^1_{\text{in}}] &= \overleftarrow{q}.\end{aligned}$$

Following the sandwich attack analysis (with $E_m = \text{id}$), we define the connection probability:

$$r = \Pr\left[E_0(\bar{P}) + E_0(\bar{P}') \in \mathcal{D}^0_{\text{out}} \;\middle|\; \begin{array}{c} E_0(P) + E_0(P') \in \mathcal{D}^0_{\text{out}} \\ E_1^{-1}(C) + E_1^{-1}(\bar{C}) \in \mathcal{D}^1_{\text{in}} \\ E_1^{-1}(C') + E_1^{-1}(\bar{C}') \in \mathcal{D}^1_{\text{in}} \end{array}\right].$$

If the four events hold, we have $\bar{P} + \bar{P}' \in \bar{\mathcal{D}}^0_{\text{in}}$ with an additional probability $\overleftarrow{p}$. This analysis of the truncated boomerang distinguisher is the same as proposed by Wagner [Wag99], but our attack is more general with structures on both sides.

In general, we have $E_1^{-1}(C) + E_1^{-1}(\bar{C})$ and $E_1^{-1}(C') + E_1^{-1}(\bar{C}')$ in $\mathcal{D}^1_{\text{in}}$, therefore we expect that they are equal with probability $|\mathcal{D}^1_{\text{in}}|^{-1}$ (under independance assumptions), and this implies $E_0(\bar{P}) + E_0(\bar{P}') \in \mathcal{D}^0_{\text{out}}$; hence $r \geqslant |\mathcal{D}^1_{\text{in}}|^{-1}$. Moreover, if $\mathcal{D}^1_{\text{in}}$ and $\mathcal{D}^0_{\text{out}}$ are vector subspaces, then $\Sigma = E_0(P) + E_0(P') + E_0(\bar{P}) + E_0(\bar{P}') \in \mathcal{D}^1_{\text{in}}$; in particular, $\Sigma \in \mathcal{D}^0_{\text{out}}$ with probability $|\mathcal{D}^0_{\text{out}} \cap \mathcal{D}^1_{\text{in}}|/|\mathcal{D}^1_{\text{in}}|$; this implies $E_0(\bar{P}) + E_0(\bar{P}') \in \mathcal{D}^0_{\text{out}}$ hence $r \geqslant |\mathcal{D}^0_{\text{out}} \cap \mathcal{D}^1_{\text{in}}|/|\mathcal{D}^1_{\text{in}}|$.

Assuming that all the events are independent, each quartet $(P_i, P_{i'}, \bar{P}_i^{\,j}, \bar{P}_{i'}^{\,j'})$, defined by a pair $(i, j), (i', j')$, follows the truncated boomerang trail with probability p_b, and randomly satisfies $\bar{P}_i^{\,j} + \bar{P}_{i'}^{\,j'} \in \bar{\mathcal{D}}^0_{\text{in}}$ with probability $p_\$$:

$$\begin{aligned} p_b &= \vec{p} \cdot \overleftarrow{p} \cdot \overleftarrow{q}^2 \cdot r, & r &\geqslant |\mathcal{D}^1_{\text{in}}|^{-1},\\ p_\$ &= |\bar{\mathcal{D}}^0_{\text{in}}|/2^n. \end{aligned}$$

We assume that the boomerang probability can be well approximated as $p_b + p_\$$. We distinguish the cipher E from a random permutation when the expected number of remaining quartets (quartets with $\bar{P}_i^{\,j} + \bar{P}_{i'}^{\,j'} \in \bar{\mathcal{D}}^0_{\text{in}}$) is significantly higher for E than for a random permutation. We define the signal-to-noise ratio:

$$\sigma = p_b/p_\$.$$

When $\sigma \gg 1$, we obtain a distinguisher using $Q = \mathcal{O}(p_b^{-1})$ quartets. More precisely, with $Q = \mu \cdot p_b^{-1}$ (μ a small constant) we expect μ remaining quartets with the cipher E, versus $\mu \cdot \sigma^{-1} \ll 1$ for a random permutation. Therefore, a good approximation is that the number of detected quartets with the cipher E follows a poisson distribution of parameter μ, while the number of detected quartets with a random permutation is approximated to 0. A distinguisher that detects the presence of at least one quartet has a success rate of $\Pr(\text{Poisson}(\mu) \geqslant 1) = 1 - e^{-\mu}$.

When σ is smaller, we need to collect a large number of quartets, and compare the expected number of remaining quartets q_E for E and $q_\$$ in the random case:

$$q_E = Q \times (p_\$ + p_b) = Q \times p_\$(1+\sigma), \qquad q_\$ = Q \times p_\$.$$

We detect the bias with $Q = \mathcal{O}(p_\$^{-1}\sigma^{-2}) = \mathcal{O}(p_b^{-1}\sigma^{-1})$ samples, following [MS02, Theorem 2]. Using $Q = c \times p_b^{-1}\sigma^{-1}$ with a small constant c and setting a threshold at $Q \times p_\$(1+\sigma/2)$, the distinguisher has a success rate of $\Phi(\sqrt{c}/2)$, with Φ the cumulative distribution function of the standard normal distribution.

If Q is smaller than the number of quartets in a full structure ($|\mathcal{D}^0_{\text{in}}|^2|\mathcal{D}^1_{\text{out}}|^2/2$), we use a partial structure with only $\sqrt{2Q}$ elements. Otherwise, we need $N = 2Q \times |\mathcal{D}^0_{\text{in}}|^{-2}|\mathcal{D}^1_{\text{out}}|^{-2}$ structures of $S = |\mathcal{D}^0_{\text{in}}||\mathcal{D}^1_{\text{out}}|$ elements. Finally, we obtain a distinguisher with a constant probability of success with the following complexity in number of quartets, time, data, and memory:

$$Q = \mathcal{O}\left(\max(p_b^{-1}, \sigma^{-1} \cdot p_b^{-1})\right), \tag{4.4}$$

$$T = D = \max(\sqrt{2Q}, 2Q \times |\mathcal{D}^0_{\text{in}}|^{-1}|\mathcal{D}^1_{\text{out}}|^{-1}), \tag{4.5}$$

$$M = \min(D, |\mathcal{D}^0_{\text{in}}||\mathcal{D}^1_{\text{out}}|). \tag{4.6}$$

4.4.1.2 Application to 6-round AES

To explain the truncated boomerang distinguisher in practice, we give a truncated boomerang trail on 6-round AES in Figure 4.8, using a simple 3-round truncated trail on AES twice, as described in Figure 2.9 (page 36). $\mathcal{D}^0_{\text{in}}$, $\bar{\mathcal{D}}^0_{\text{in}}$ and $\mathcal{D}^1_{\text{in}}$ are the sets of states that have zeros on all diagonals except the main one. $\mathcal{D}^0_{\text{out}}$ is the set of differences MixColumns(Δ), for all Δ active only on the main anti-diagonal, and $\mathcal{D}^1_{\text{out}}$ is active on the main anti-diagonal (they differ because we omit the last MixColumns operation). We have

$$|\mathcal{D}^0_{\text{in}}| = |\bar{\mathcal{D}}^0_{\text{in}}| = |\mathcal{D}^0_{\text{out}}| = 2^{32}, \qquad \vec{p} = 2^{-24}, \qquad \overleftarrow{p} = 2^{-24},$$
$$|\mathcal{D}^1_{\text{in}}| = |\mathcal{D}^1_{\text{out}}| = 2^{32}, \qquad \vec{q} = 2^{-24}, \qquad \overleftarrow{q} = 2^{-24}.$$

Since $\mathcal{D}^0_{\text{out}} \cap \mathcal{D}^1_{\text{in}} = \{0\}$, we have $r = |\mathcal{D}^1_{\text{in}}|^{-1}$, and the analysis above gives the following parameters:

$$p_b = \vec{p} \cdot \overleftarrow{p} \cdot \overleftarrow{q}^2 \times |\mathcal{D}^1_{\text{in}}|^{-1} = 2^{-128}, \qquad \sigma = 2^{-32},$$
$$p_\$ = |\bar{\mathcal{D}}^0_{\text{in}}|/2^n = 2^{-96}, \qquad Q = c \cdot 2^{160}.$$

Using $c = 4$ and the formulas of Equations (4.4), (4.5), and (4.6), we obtain a distinguisher with complexity:

$$T = D = 2^{99}, \qquad M = 2^{64}.$$

The distinguisher is detailed in Algorithm 4.2. It makes 2^{67} encryption queries and 2^{99} decryption queries, for a total data complexity of $D = 2^{67} + 2^{99} \approx 2^{99}$. In

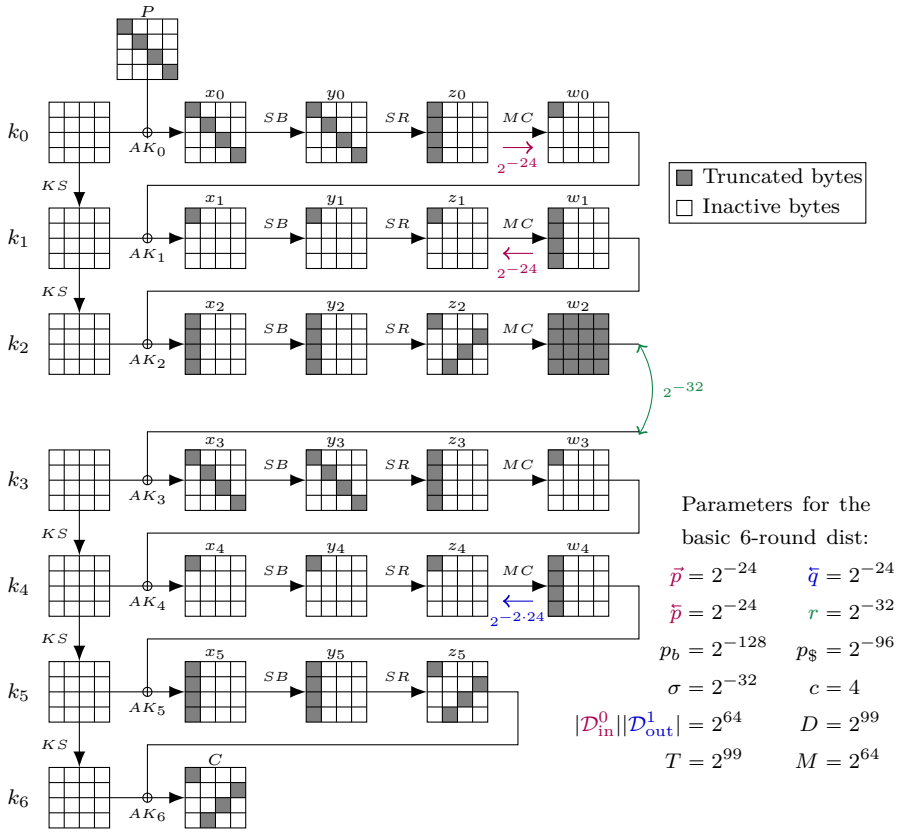


Figure 4.8: A truncated boomerang trail on 6-round AES (non-optimized).

total, we have $Q = 2^{35} \times 2^{64} \times 2^{64}/2 = 2^{162}$ quartets $(P_i, P_{i'}, \bar{P}_i^j, \bar{P}_{i'}^{j'})$, so that the expected number of remaining quartets is:

$$q_\$ = Q \times 2^{-96} = 2^{66}, \qquad q_E = Q \times (2^{-96} + 2^{-128}) = 2^{66} + 2^{34}.$$

The distinguisher returns the correct answer with probability $\Phi(\sqrt{c}/2) \approx 0.84$.

This distinguisher is interesting because it is very generic: it does not require knowledge of the S-box or the MDS matrix, and it can be considered as "key-independent" in the sense of [GRR17]. The complexity is slightly higher than previous distinguishers with similar properties. However, the simplicity of this distinguisher makes it more likely to be applicable when 6-round AES is used as a building block in a more complex structure, as shown in Section 4.4.5 against TNT-AES.

4.4.2 Truncated Boomerang Key-recovery Attack

We now consider key-recovery attacks. As opposed to typical differential or linear attacks, we do not add rounds on top of the distinguisher. Instead, we assume

Algorithm 4.2: Truncated boomerang distinguisher on 6-round AES

$q \leftarrow 0$
for $0 \leqslant s < 2^{35}$ **do** ▷ 2^{35} structures
  $P_0 \leftarrow \$$
  for $i \in \mathcal{D}^0_{\text{in}}$ **do** ▷ Iterate over the main diagonal
    $C_i \leftarrow E(P_0 + i)$
    for $j \in \mathcal{D}^1_{\text{out}}$ **do** ▷ Iterate over the main anti-diagonal
      $\bar{P}_i^{\,j} \leftarrow E^{-1}(C_i + j)$
      Store $\bar{P}_i^{\,j}$ in a hash table indexed by three diagonals
  Count collisions in the hash table, and increment q
if $q > 2^{66} + 2^{33}$ **then**
  return AES
else
  return $

that the truncated boomerang covers the full targeted cipher, and we design a key-recovery attack with smaller complexity than the corresponding distinguisher.

When $\sigma \geqslant 1$, the truncated boomerang distinguisher is easy to turn into a key-recovery attack, but we cannot reduce the complexity. Indeed, the bottleneck of the distinguisher is to have enough data so that a boomerang quartet exists. When a quartet with $\bar{P} + \bar{P}' \in \bar{\mathcal{D}}^0_{\text{in}}$ is found, it has a high probability of following the boomerang, and standard methods can be used to recover key candidates. Therefore, we focus on the case $\sigma \ll 1$, where the distinguisher requires multiple quartets following the boomerang.

Given a candidate quartet with $\bar{P} + \bar{P}' \in \bar{\mathcal{D}}^0_{\text{in}}$, we can extract some key information assuming that it follows the boomerang. If this is the case, we have two pairs of known plaintexts (P, P') and $(\bar{P}, \bar{P}')$ respectively following the truncated differentials $\mathcal{D}^0_{\text{in}} \xrightarrow[E_0]{\vec{p}} \mathcal{D}^0_{\text{out}}$ and $\mathcal{D}^0_{\text{out}} \xrightarrow[E_0^{-1}]{\vec{p}} \bar{\mathcal{D}}^0_{\text{in}}$, and two pairs of known ciphertexts $(C, \bar{C})$ and $(C', \bar{C}')$ following the truncated differential $\mathcal{D}^1_{\text{out}} \xrightarrow[E_1^{-1}]{\vec{q}} \mathcal{D}^1_{\text{in}}$. Using standard techniques from differential cryptanalysis, we can usually extract partial information about the first and last subkeys. We denote by κ the number of key bits that can be extracted, and by ℓ the average number of κ-bit key candidates suggested by a quartet. Note that the key information suggested by a quartet might be incompatible between both pairs of plaintexts following the upper differentials (or between both pairs of ciphertexts following the lower differential), in this case the quartet is discarded.

We follow the standard approach to identify the most likely candidates for the κ bits of key: we build a table of 2^κ counters corresponding to key candidates, and we increment the counter of each key suggested by each quartet. With enough data, the right key is expected to be among the top $2^{\kappa-a}$ counters (a denotes the advantage of the attack).

4.4.2.1 Analysis

Following the previous analysis, we expect $Q \times (p_\$ + p_b)$ quartets with $\bar{P} + \bar{P}' \in \bar{\mathcal{D}}^0_{\text{in}}$: $Q \times p_b$ quartets following the boomerang trail (right quartets), and $Q \times p_\$$ false positives. For a right quartet, the correct key is among the deduced key candidates, and for a wrong quartet, we expect that ℓ random key candidates are deduced. Assuming that all the quartets behave independently, the wrong counters follow the binomial distribution $\mathcal{B}(Q, (p_\$ + p_b) \times \ell \times 2^{-\kappa})$ and the right counter follows the distribution $\mathcal{B}(Q, p_\$ \times \ell \times 2^{-\kappa} + p_b)$. We denote the probabilities of suggesting a wrong key and the right key as:

$$p_w = (p_\$ + p_b) \times \ell \times 2^{-\kappa} \approx p_\$ \times \ell \times 2^{-\kappa},$$
$$p_0 = p_\$ \times \ell \times 2^{-\kappa} + p_b \approx p_w + p_b.$$

We obtain a higher signal-to-noise ratio $\tilde{\sigma}$ than previously:

$$\tilde{\sigma} = p_b/p_w = \sigma \times 2^\kappa/\ell.$$

When $\tilde{\sigma} \gg 1$, only a handful of right quartets are necessary to have the right key ranked first, so that $Q = \mathcal{O}(p_b^{-1})$.

When $\tilde{\sigma} \ll 1$, the counters can be approximated by normal distributions, and we use the work of Selçuk [Sel08, Theorem 3] to evaluate the number of samples needed to have the right key among the top $2^{\kappa-a}$ key candidates (depending on the success rate). For a fixed value of a, we need Q proportional to $p_b^{-1}\tilde{\sigma}^{-1}$, and the complexity increases linearly in a. Finally, the increased signal-to-noise ratio $\tilde{\sigma} \gg \sigma$ reduces the data complexity to:

$$Q = \mathcal{O}\left(\max(p_b^{-1}, \tilde{\sigma}^{-1} \times p_b^{-1})\right),$$
$$D = \max(\sqrt{2Q}, 2Q \times |\mathcal{D}^0_{\text{in}}|^{-1}|\mathcal{D}^1_{\text{out}}|^{-1}).$$

The full attack is described in Algorithm 4.3. The time complexity is harder to evaluate; it can be bounded with T_E the cost of an oracle call (by convention, $T_E = 1$), and T_C the cost of deducing key candidates from a quartet:

$$T = D \times T_E + Q \times p_\$ \times T_C.$$

When $T_C \ll 2^n \times |\mathcal{D}^0_{\text{in}}|^{-1}|\bar{\mathcal{D}}^0_{\text{in}}|^{-1}|\mathcal{D}^1_{\text{out}}|^{-1}$, we have $Q \times p_\$ \times T_C \ll D$ and the second term is negligible; the cost of the attack is thus dominated by the oracle queries. Otherwise, it is often possible to reduce the second term with more advanced filtering, but this requires a dedicated analysis for each attack.

After recovering $2^{\kappa-a}$ candidates for the κ-bit partial key, the full key can be recovered by exhaustive search of the remaining bits with complexity 2^{n-a}, or by launching a variant of the attack on a different set of key bits.

4.4.2.2 Success Probability

When $\tilde{\sigma} \ll 1$, the average values of right and wrong counters are high enough to approximate them with normal distributions. In that case, the success rate can be

Algorithm 4.3: Truncated boomerang key-recovery attack

Require: $\mathcal{D}^0_{\text{in}}$,$\bar{\mathcal{D}}^0_{\text{in}}$, $\mathcal{D}^1_{\text{out}}$
  $\mathcal{K} \leftarrow$ InitKeyCounters()
  for $i \leftarrow 1$ to N **do**
    $P_0 \leftarrow$ Rand()
    $\mathcal{P} \leftarrow [P_0 + \Delta_{\text{in}}, \text{ for } \Delta_{\text{in}} \in \mathcal{D}^0_{\text{in}}]$
    $\mathcal{C} \leftarrow [E(P), \text{ for } P \in \mathcal{P}]$ ▷ $N \times |\mathcal{D}^0_{\text{in}}|$ encryptions
    $\bar{\mathcal{P}} \leftarrow [E^{-1}(C + \Delta_{\text{out}}), \text{ for } C \in \mathcal{C}, \Delta_{\text{out}} \in \mathcal{D}^1_{\text{out}}]$ ▷ $N \times |\mathcal{D}^0_{\text{in}}||\mathcal{D}^1_{\text{out}}|$ decryptions
    $\mathcal{H} \leftarrow$ InitHashMap()
    for $\bar{P} \in \bar{\mathcal{P}}$ **do**
      Insert the projection of $\bar{P}$ on $\{0,1\}^n/\bar{\mathcal{D}}^0_{\text{in}}$ in $\mathcal{H}$ ▷ $|\mathcal{D}^0_{\text{in}}||\mathcal{D}^1_{\text{out}}|$ in memory
    if a collision occurs in $\mathcal{H}$ between $\bar{P}$ and $\bar{P}'$ **then**
      Track back to the corresponding P and P'
      if $P \neq P'$ **then** ▷ $N(p_\$ + p_b)|\mathcal{D}^0_{\text{in}}|^2|\mathcal{D}^1_{\text{out}}|^2/2$ such quartets
        for K in the ℓ key candidates induced by the quartet **do**
          Increment the counter K in $\mathcal{K}$
  Recover the key material K of the maximal counter of $\mathcal{K}$
  Recover the rest of the key by performing the attack with a different $\mathcal{D}^0_{\text{in}}$,$\bar{\mathcal{D}}^0_{\text{in}}$

evaluated using the formula given by [Sel08], under additional assumptions about the independence of key counters, and order statistics:

$$P_S = \Phi\left(\frac{\sqrt{\mu\tilde{\sigma}} - \Phi^{-1}(1-2^{-a})}{\sqrt{\tilde{\sigma}+1}}\right). \tag{4.7}$$

with $\mu = Q \times p_b$ the expected number of right quartets.

When $\tilde{\sigma}$ is high, the binomial distributions of right and wrong counters have average values of respectively $Q \times p_b \approx 1$ and $Q \times p_w \ll 1$. As discussed in [Sel08, section 3.2.1], the normal approximation is inaccurate in this case; we obtain a more accurate estimate of the success probability using a Poisson approximation.

4.4.2.3 Extracting Key Candidates

When the truncated differentials are described by truncated trails (with a set of intermediate differences at each step), the parameters ℓ and κ can often be deduced directly from the trail. For simplicity, as in the truncated boomerang paper [BL23b], we consider the case where $\bar{\mathcal{D}}^0_{\text{in}} = \mathcal{D}^0_{\text{in}}$, i.e. the input and returning differences are in the same set. We assume that E_0 starts with the addition of a subkey K_0, followed by an S-box layer SB, and we denote the set of differences after the S-box layer by $\mathcal{D}^0_{\text{mid}}$:

$$E_0 = \tilde{E}_0 \circ \mathsf{SB} \circ \mathsf{AK}_{K_0}, \qquad \mathcal{D}^0_{\text{in}} \xrightarrow[\mathsf{SB}]{\vec{p}_0} \mathcal{D}^0_{\text{mid}}, \qquad \mathcal{D}^0_{\text{mid}} \xrightarrow[\tilde{E}_0]{\vec{p}_1} \mathcal{D}^0_{\text{out}}.$$

We also assume that $\mathcal{D}^0_{\text{in}}$ is a vector subspace aligned with the S-box layer (each S-box is either inactive, or active with all possible differences). $\mathcal{D}^0_{\text{mid}}$ is a subset of

$\mathcal{D}^0_{\text{in}}$; typically it is constructed so that some parts of the state have fixed differences after the linear layer. For instance, in the boomerang trail on 6-round AES of Figure 4.8, $\mathcal{D}^0_{\text{mid}}$ corresponds to differences δ such that $\mathsf{MixColumns}(\mathsf{ShiftRows}(\delta))$ is active only on the first cell, with $|\mathcal{D}^0_{\text{mid}}| = 2^8$ and $\vec{p}_0 = 2^{-24}$. In general, we have:

$$\vec{p}_0 = |\mathcal{D}^0_{\text{mid}}|/|\mathcal{D}^0_{\text{in}}|, \qquad \vec{p} = \vec{p}_0 \times \vec{p}_1. \tag{4.8}$$

We consider a pair (P, P'), and assume that it follows the truncated trail, *i.e.* $\mathsf{SB}(P+K_0)+\mathsf{SB}(P'+K_0) \in \mathcal{D}^0_{\text{mid}}$. This constrains the subkey $K_{0|\mathcal{D}^0_{\text{in}}}$ corresponding to the active S-boxes in SB. More precisely, given (P, P'), for each difference Δ_{mid} in $\mathcal{D}^0_{\text{mid}}$, we expect on average 0.5 unordered pairs of values $\{X, X'\}$ such that $X + X' = P + P'$ and $\mathsf{SB}(X) + \mathsf{SB}(X') = \Delta_{\text{mid}}$ (restricted to the active bytes $\mathcal{D}^0_{\text{mid}}$). This pair can be recovered efficiently after pre-computing the DDT of the S-box, and we deduce two possible keys $X + P$ and $X + P'$. Therefore, for each pair (P, P') and difference $\Delta_{\text{mid}} \in \mathcal{D}^0_{\text{mid}}$, we can deduce on average one key candidate, which implies that given a pair (P, P'), we can deduce on average $|\mathcal{D}^0_{\text{mid}}|$ key candidates for $K_{0|\mathcal{D}^0_{\text{in}}}$. This gives the following parameters when extracting key candidates from a pair (P, P'):

$$\ell^0 = |\mathcal{D}^0_{\text{mid}}|, \qquad \kappa^0 = \log_2(|\mathcal{D}^0_{\text{in}}|), \qquad T^0_C = \ell^0 = |\mathcal{D}^0_{\text{mid}}|.$$

Starting from a candidate quartet, we have two different pairs (P, P') and $(\bar{P}, \bar{P}')$ assumed to follow the upper differential $\mathcal{D}^0_{\text{in}} \xrightarrow[\vec{p}_0]{\mathsf{SB}} \mathcal{D}^0_{\text{mid}}$. Therefore, we expect only $|\mathcal{D}^0_{\text{mid}}|^2/|\mathcal{D}^0_{\text{in}}|$ key candidates compatible with both pairs. We apply the same reasoning to the lower trail (using ciphertext pairs), and deduce the parameters ℓ and κ for a quartet in the general case:

$$\ell = |\mathcal{D}^0_{\text{mid}}|^2 \cdot |\mathcal{D}^1_{\text{mid}}|^2 \cdot |\mathcal{D}^0_{\text{in}}|^{-1} \cdot |\mathcal{D}^1_{\text{out}}|^{-1}, \qquad \kappa = \log_2(|\mathcal{D}^0_{\text{in}}| \cdot |\mathcal{D}^1_{\text{out}}|).$$

Using the probability $\vec{p}_0$ for the first round and $\overleftarrow{q}_0$ for the last round, we have:

$$\ell \cdot 2^{-\kappa} = {\vec{p}_0}^2 \cdot {\overleftarrow{q}_0}^2.$$

This strategy needs to be applied to each quartet with the correct returning difference. If we start by the recovery $K_{0|\mathcal{D}^0_{\text{in}}}$ (in the upper part), then for the lower trail, we only have to process a fraction $|\mathcal{D}^0_{\text{mid}}|^2/|\mathcal{D}^0_{\text{in}}|$ of the candidate quartets (with a key compatible with both pairs). In particular, when $|\mathcal{D}^0_{\text{mid}}|^2 \ll |\mathcal{D}^0_{\text{in}}|$, the time complexity is dominated by the first extraction step: $T_C = |\mathcal{D}^0_{\text{mid}}|$. We might also want to start from the bottom side if the time complexity of extracting key candidates in the bottom side ($|\mathcal{D}^1_{\text{mid}}|$) is less that the complexity of extraction in the top side ($|\mathcal{D}^0_{\text{mid}}|$).

In some cases, better results are obtained with a dedicated analysis, e.g. using more than one round of the trail. We provide such examples in our attacks against Deoxys-BC of Section 4.4.7.

4.4.2.4 Application to 6-round AES

This attack can directly be applied to AES, using the same 3-round trails as in the previous section (see Figure 4.8):

$$|\bar{\mathcal{D}}^0_{\text{in}}| = |\mathcal{D}^0_{\text{in}}| = |\mathcal{D}^0_{\text{out}}| = 2^{32}, \quad |\mathcal{D}^0_{\text{mid}}| = 2^8, \quad \vec{p} = 2^{-24}, \quad \overleftarrow{p} = 2^{-24},$$
$$|\mathcal{D}^1_{\text{in}}| = |\mathcal{D}^1_{\text{out}}| = 2^{32}, \quad |\mathcal{D}^1_{\text{mid}}| = 2^8, \quad \overleftarrow{q} = 2^{-24}, \quad r = |\mathcal{D}^1_{\text{in}}|^{-1}.$$

Using the parameters of the key extraction, our analysis results in

$$\ell = |\mathcal{D}^0_{\text{mid}}|^2 \cdot |\mathcal{D}^1_{\text{mid}}|^2 \cdot |\mathcal{D}^0_{\text{in}}|^{-1} \cdot |\mathcal{D}^1_{\text{out}}|^{-1} = 2^{-32}, \quad \kappa = \log_2(|\mathcal{D}^0_{\text{in}}| \cdot |\mathcal{D}^1_{\text{out}}|) = 64,$$
$$p_b = \vec{p} \cdot \overleftarrow{p} \cdot \overleftarrow{q}^2 \times r = 2^{-128},$$
$$p_w = |\bar{\mathcal{D}}^0_{\text{in}}| \times 2^{-n} \times \ell \times 2^{-\kappa} = 2^{-192}, \qquad \tilde{\sigma} = 2^{64}.$$

Since $\tilde{\sigma} \gg 1$, we only need a few right quartets; with $\mu = 4$ we obtain

$$Q = \mu \times p_b^{-1} = 2^{130}, \qquad D = 2^{67}.$$

Time complexity. With these parameters, the attack complexity is dominated by the oracle queries. We use 8 structures of 2^{64} elements; in each structure we detect $2^{64} \times 2^{63} \times p_{\$} = 2^{31}$ pairs with $\bar{P} + \bar{P}' \in \bar{\mathcal{D}}^0_{\text{in}}$, resulting in $8 \times 2^{31} = 2^{34}$ candidate quartets in total. Each quartet suggests on average 2^{-32} candidates for 64 bits of key (for most of the quartets, there is no key compatible with both sides of the quartet). Finally, we expect 2^2 suggestions of wrong keys (each key is suggested 2^{-62} times on average), and $\mu = 4$ suggestions for the correct key. With high probability, the key with the most suggestions is the correct one.

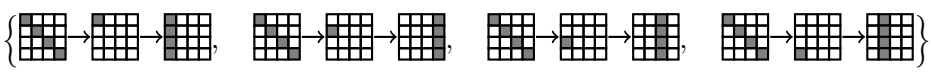
We implemented the attack on a reduced AES with 4-bit S-boxes, and it behaves as expected [BL23a].

4.4.3 Optimized Boomerang Attacks on 6-round AES

As shown by Biryukov [Bir04] and described in Section 4.3.1, boomerang attacks on AES can be optimized using multiple trails. We now present improved versions of our attacks using this technique, including a 6-round key-recovery attack with complexity 2^{61}. The improvement compared to the attack of Biryukov with complexity 2^{71} is due to the use of structures on the ciphertext side.

4.4.3.1 Optimized distinguisher

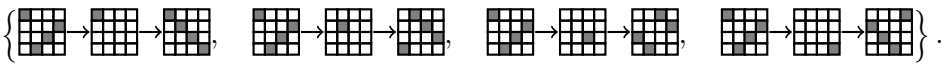
Instead of only considering the upper trail of Figure 4.8 with fixed positions for all the active bytes, we consider a collection of four different trails for upper part:



The collection can be considered as a truncated differential $\mathcal{D}^0_{\text{in}} \xrightarrow[E_0]{\vec{p}} \mathcal{D}^0_{\text{out}}$ with:

$$\vec{p} = 2^{-22}, \qquad |\mathcal{D}^0_{\text{in}}| = 2^{32}, \qquad |\mathcal{D}^0_{\text{out}}| = 2^{34}.$$

Similarly, we consider four trails for the lower part through E_1^{-1}:

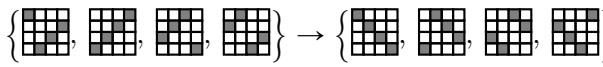


Again, this can be considered as a truncated differential $\mathcal{D}^1_{\text{out}} \xrightarrow[E_1^{-1}]{\overleftarrow{q}} \mathcal{D}^1_{\text{in}}$ with:

$$\overleftarrow{q} = 2^{-22}, \qquad |\mathcal{D}^1_{\text{in}}| = 2^{34}, \qquad |\mathcal{D}^1_{\text{out}}| = 2^{32}.$$

The analysis of the previous sections can be applied as-is with these trails. We obtain a better attack because we increase $\vec{p}$ and $\overleftarrow{q}$ by a factor 4, even though $|\mathcal{D}^1_{\text{in}}|$ increases by a factor 4; we obtain $p_b = 2^{-124}$ instead of 2^{-128}. The distinguisher is exactly the same because $\mathcal{D}^0_{\text{in}}$, $\bar{\mathcal{D}}^0_{\text{in}}$ and $\mathcal{D}^1_{\text{out}}$ are the same, but this improved analysis shows that the complexity of the distinguisher can be reduced to $T = D = 2^{91}$ (with $c = 4$, $\sigma = 2^{-28}$ and $Q = 2^{154}$).

Larger set $\mathcal{D}^1_{\text{out}}$. We further improve the distinguisher using a collection of 16 trails with the following input and output sets for the lower trail through E_1^{-1}:



This collection can be considered as a truncated differential $\mathcal{D}^1_{\text{out}} \xrightarrow[E_1^{-1}]{\overleftarrow{q}} \mathcal{D}^1_{\text{in}}$ with:

$$\overleftarrow{q} = 2^{-22}, \qquad |\mathcal{D}^1_{\text{in}}| = 2^{34}, \qquad |\mathcal{D}^1_{\text{out}}| = 2^{34}.$$

This does not affect the probability p_b, but generates larger structures; the complexity is reduced to $T = D = 2^{89}$ with $Q = 2^{154}$.

Different Set $\bar{\mathcal{D}}^0_{\text{in}}$ for Returning Pairs. Following the boomerang attack of Biryukov [Bir04], we use a larger set of accepted returning differences $\bar{\mathcal{D}}^0_{\text{in}}$ in order to increase the boomerang probability. We consider the same collection of 16 trails as above, corresponding to a truncated differential $\mathcal{D}^0_{\text{out}} \xrightarrow[E_0^{-1}]{\overleftarrow{p}} \bar{\mathcal{D}}^0_{\text{in}}$ with:

$$\overleftarrow{p} = 2^{-22}, \qquad |\bar{\mathcal{D}}^0_{\text{in}}| = 2^{34}, \qquad |\mathcal{D}^0_{\text{out}}| = 2^{34}.$$

This corresponds to keeping quartets with a single active diagonal in $\bar{P} + \bar{P}'$, but not necessarily the main one. This gives the following parameters:

$$p_b = \vec{p} \cdot \overleftarrow{p} \cdot \overleftarrow{q}^2 \times |\mathcal{D}^1_{\text{in}}|^{-1} = 2^{-122}, \qquad \sigma = 2^{-28},$$
$$p_\$ = |\bar{\mathcal{D}}^0_{\text{in}}|/2^n = 2^{-94}, \qquad Q = 2^{152}.$$

Finally, we obtain a distinguisher with complexity $T = D = 2^{87}$ (with $c = 4$).

4.4.3.2 Optimized key-recovery attack

Using multiple trails. For a key-recovery attack, we use the trails above, but we keep the set $\mathcal{D}^1_{\text{out}}$ active only in the first column. This gives:

$$\vec{p} = 2^{-22}, \quad |\mathcal{D}^0_{\text{in}}| = 2^{32}, \quad |\mathcal{D}^0_{\text{out}}| = 2^{34},$$
$$\overleftarrow{p} = 2^{-22}, \quad |\bar{\mathcal{D}}^0_{\text{in}}| = 2^{34},$$
$$\overleftarrow{q} = 2^{-22}, \quad |\mathcal{D}^1_{\text{in}}| = 2^{34}, \quad |\mathcal{D}^1_{\text{out}}| = 2^{32}.$$

When extracting the key, we recover some information on a diagonal of k_0 from (P, P') (but not $(\bar{P}, \bar{P}')$ since it not necessarely active on the same diagonal as (P, P')), and information about an anti-diagonal of k_6 from $(C, \bar{C})$ and $(C', \bar{C}')$:

$$\ell^0 = 2^{10}, \quad \kappa^0 = 32, \quad \ell^1 = 2^{-14}, \quad \kappa^1 = 32, \quad \ell = 2^{-4}, \quad \kappa = 64.$$

Therefore, we have the following parameters:

$$p_b = \vec{p} \cdot \overleftarrow{p} \cdot \overleftarrow{q}^2 \times |\mathcal{D}^1_{\text{in}}|^{-1} = 2^{-122},$$
$$p_w = |\bar{\mathcal{D}}^0_{\text{in}}| \times 2^{-n} \times \ell \times 2^{-\kappa} = 2^{-162}, \qquad \tilde{\sigma} = 2^{40}.$$

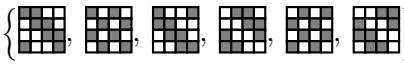
Since $\tilde{\sigma} \gg 1$, we only need a few right quartets; with $\mu = 4$ we obtain

$$Q = \mu \times p_b^{-1} = 2^{124}, \qquad D = 2^{62.5}.$$

Time complexity. The complexity is dominated by the oracle queries: for each structure of 2^{64} plaintexts/ciphertexts, we filter $2^{64} \times 2^{63} \times |\bar{\mathcal{D}}^0_{\text{in}}| \times 2^{-n} = 2^{33}$ candidate quartets, and the time to extract the key candidates is negligible.

After recovering a candidate for 64 bits of key (32 bits of k_0 and 32 bits of k_6), we repeat the attack with $\mathcal{D}^0_{\text{in}}$ in a different diagonal and use the partial knowledge of k_6 to increase the probability $\overleftarrow{q}$ (this has negligible complexity).

Using two active diagonals. In order to further reduce the complexity of this attack, we use truncated boomerang characteristics with lower signal-to-noise ratios, taking advantage of the additional filter provided by the key extraction. Following [Bir04], we modify the truncated trail on the returning side $(\bar{P}, \bar{P}')$ to allow any combination of two active diagonals in input, leading to the following parameters:

$$\bar{\mathcal{D}}^0_{\text{in}} = \left\{ \right.$$  $$\left. \right\},$$

$$\vec{p} = 2^{-22}, \quad |\mathcal{D}^0_{\text{in}}| = 2^{32}, \quad |\mathcal{D}^0_{\text{out}}| = 2^{34}, \quad |\mathcal{D}^0_{\text{mid}}| = 2^{10},$$
$$\overleftarrow{p} = 6 \times 2^{-16} = 2^{-13.4}, \quad |\bar{\mathcal{D}}^0_{\text{in}}| = 6 \times 2^{64},$$
$$\overleftarrow{q} = 2^{-22}, \quad |\mathcal{D}^1_{\text{in}}| = 2^{34}, \quad |\mathcal{D}^1_{\text{out}}| = 2^{32}, \quad |\mathcal{D}^1_{\text{mid}}| = 2^{10}.$$

When extracting the key, we recover information about the main diagonal of k_0 from (P, P'), and information about the first anti-diagonal of k_6 from $(C, \bar{C})$

and $(C', \bar{C}')$ (note that $(\bar{P}, \bar{P}')$ is not necessarily active in the main diagonal). Moreover, the key suggested by $(C, \bar{C})$ and $(C', \bar{C}')$ must lead to the same active byte in z_4, so that:

$$\ell^0 = 2^{10}, \qquad \kappa^0 = 32, \qquad \ell^1 = 2^{-14}, \qquad \kappa^1 = 32, \qquad \ell = 2^{-4}, \qquad \kappa = 64.$$

Using the previous analysis, we obtain:

$$p_b = \vec{p} \cdot \overleftarrow{p} \cdot \overleftarrow{q}^2 \times |\mathcal{D}^1_{\text{in}}|^{-1} = 2^{-113.4},$$
$$p_w = |\bar{\mathcal{D}}^0_{\text{in}}| \times 2^{-n} \times \ell \times 2^{-\kappa} = 2^{-129.4} \qquad \tilde{\sigma} = 2^{16}.$$

Since $\tilde{\sigma} \gg 1$, a few right quartets are sufficient for the success of this attack; we use $\mu = 8$, this corresponds to $Q = 2^{116.4}$ and we use a partial structure of $D = 2^{58.7}$ elements.

Success probability. We assume that the attacker keeps key candidates with counter values of at least 5. With $\tilde{\sigma} \gg 1$, we approximate the wrong key counters by Poisson distributions with $\lambda = Q \times p_w = 2^{-13}$, each of which equal 5 or more with probability $1 - e^{-\lambda}(1 + \lambda + \lambda^2/2 + \lambda^3/6 + \lambda^4/24) \approx 2^{-71.9}$; we do not expect to keep any wrong keys. On the other hand, the counter for the right key follows a Poisson distribution with $\lambda = \mu = 8$. It reaches a value of 5 or more with probability ≈ 0.9.

Time complexity. After recovering a candidate for 64 bits of key (32 bits of k_0 and 32 bits of k_6), we repeat the attack with $\mathcal{D}^0_{\text{in}}$ in a different diagonal and use the partial knowledge of k_6 to increase the probability $\overleftarrow{q}$. This step has a negligible complexity.

The time complexity is balanced between oracle queries and extracting key candidates. Indeed, we filter $2^{58.7} \times 2^{57.7} \times |\bar{\mathcal{D}}^0_{\text{in}}| \times 2^{-n} = 2^{55}$ candidates with $\bar{P} + \bar{P}' \in \bar{\mathcal{D}}^0_{\text{in}}$ using 6 hash tables indexed by each combination of two active columns. The complexity T_C to generate key candidates for a given quartet is essentially 4×2^{10} accesses to a small table; we approximate it as $T_C \approx 2^{5.4} T_E$ (since one encryption has 6×16 S-boxes). Finally, the time complexity is

$$T = 2^{58.7} T_E + 2^{55} T_C \approx 2^{60.8} T_E.$$

4.4.3.3 Key-recovery with secret S-boxes

The techniques described in Section 4.4.3.2 assume that the S-box and MDS matrix are known to the attacker in order to extract key information. However, it is also possible to extract key information with an unknown S-box under some conditions. Following [GRR16], we assume that all S-boxes in a column are identical, and that the MDS matrix has two identical coefficients in each row.

As a concrete example, we consider the AES MixColumns matrix:

$$\mathsf{MC} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}.$$

We consider a pair $(C, \bar{C})$ following the truncated trail of Figure 4.8. According to the trail, the difference before the last round (w_4) is in a set of 2^8 differences; in particular, the difference in cell 1 is equal to the difference in cell 2:

$$w_4 + \bar{w}_4 \in \left\{ \begin{bmatrix} 2\delta & 0 & 0 & 0 \\ \delta & 0 & 0 & 0 \\ \delta & 0 & 0 & 0 \\ 3\delta & 0 & 0 & 0 \end{bmatrix} \;:\; \delta \in \{0,1\}^8 \right\} = \left\{ \mathsf{MC} \cdot \begin{bmatrix} \delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \;:\; \delta \in \{0,1\}^8 \right\}.$$

Moreover, we assume that the differences in cells 13 and 10 of the ciphertext are equal (they are moved to cells 1 and 2 by ShiftRows):

$$C + \bar{C} = \begin{bmatrix} \alpha & 0 & 0 & 0 \\ 0 & 0 & 0 & \beta \\ 0 & 0 & \beta & 0 \\ 0 & \gamma & 0 & 0 \end{bmatrix}.$$

In this case, S-boxes 1 and 2 in the last round follow the same transition $\delta \to \beta$. With high probability, this implies that the unordered pairs of input/output are equal; in particular $\{C[13] + k_6[13], \bar{C}[13] + k_6[13]\} = \{C[10] + k_6[10], \bar{C}[10] + k_6[10]\}$. This suggests two key candidates:

$$k_6[13] + k_6[10] \in \left\{ C[13] + C[10], C[13] + \bar{C}[10] \right\}.$$

In order to use this property in a truncated boomerang attack, we use the multiple upper trails of Section 4.4.3.1, and a single lower trail with a restricted $\mathcal{D}^1_{\text{out}}$ of size 2^{24} to ensure that $C + \bar{C}$ and $C' + \bar{C}'$ have the required properties for all quartets considered:

$$\mathcal{D}^1_{\text{out}} = \left\{ \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & 0 & 0 & b \\ 0 & 0 & b & 0 \\ 0 & c & 0 & 0 \end{bmatrix} \;:\; a, b, c \in \{0,1\}^8 \right\}.$$

The corresponding parameters are:

$$\begin{aligned} &\vec{p} = 2^{-22}, && |\mathcal{D}^0_{\text{in}}| = 2^{32}, && |\mathcal{D}^0_{\text{out}}| = 2^{34}, \\ &\overleftarrow{p} = 2^{-22}, && |\bar{\mathcal{D}}^0_{\text{in}}| = 2^{34}, \\ &\overleftarrow{q} = 2^{-24}, && |\mathcal{D}^1_{\text{in}}| = 2^{32}, && |\mathcal{D}^1_{\text{out}}| = 2^{24}. \end{aligned}$$

For each quartet, the pair $(C, \bar{C})$ suggests two values for $k_6[13] + k_6[10]$, and $C', \bar{C}'$ also suggests two values. Therefore a quartet suggests on average $\ell = 2^{-6}$ values for $\kappa = 8$ bits of key. Using the truncated boomerang analysis, we obtain:

$$\begin{aligned} &p_b = \vec{p} \cdot \overleftarrow{p} \cdot \overleftarrow{q}^2 \times |\mathcal{D}^1_{\text{in}}|^{-1} = 2^{-124}, && \tilde{\sigma} = 2^{-16}, \\ &p_w = |\bar{\mathcal{D}}^0_{\text{in}}| \times 2^{-n} \times \ell \cdot 2^{-\kappa} = 2^{-108}, && Q = \mathcal{O}(2^{140}). \end{aligned}$$

To obtain a high probability of success we use $Q = 2^{145}$, *i.e.* $D = 2^{90}$. Since $\tilde{\sigma} \ll 1$, the counter distribution of the right key can be approximated to the normal distribution $\mathcal{N}(2^{37} + 2^{21}, 2^{37})$ while wrong key counters distributions can be approximated to $\mathcal{N}(2^{37}, 2^{37})$. We expect the correct key to be ranked first with very high probability ($P_S > 0.99$ using the formula from [Sel08]).

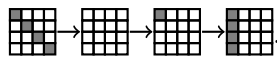
The time complexity is dominated by the oracle queries: for each structure of 2^{56} plaintexts/ciphertexts, we filter $2^{56} \times 2^{55} \times |\bar{\mathcal{D}}^0_{\text{in}}| \times 2^{-n} = 2^{17}$ candidate quartets with $\bar{P} + \bar{P}' \in \bar{\mathcal{D}}^0_{\text{in}}$, and the time to extract the key candidates is negligible. We can repeat the attack to recover up to 16 key bytes in different positions, with a complexity of $D = T = 2^{94}$ (but only 12 recovered bytes are linearly independent).

4.4.4 Application to 8-round Kiasu-BC

Kiasu-BC [JNP14a] is an instance of the TWEAKEY framework [JNP14b], reusing the AES round function in a tweakable block cipher. Its specification is presented in Section 2.3.4.3. The 6-round boomerang attack on the AES can be extended to 8-round Kiasu-BC by taking advantage of the tweak input to cancel state differences in order to have one inactive round in the upper and lower trails. Indeed, the best known attack on Kiasu-BC is an 8-round attack with complexity 2^{103} in data and time [DL17] following this idea; the corresponding boomerang is represented in Figure 4.9. Following our framework, we improve this attack with a better use of structures.

4.4.4.1 Truncated boomerang

Since we use a tweak difference Δ_{tw}, we slightly generalize our truncated differential framework to allow a set of tweak differences $\mathcal{D}^0_{\text{tw}}$. We start from a 4-round truncated trail $(\mathcal{D}^0_{\text{in}}, \mathcal{D}^0_{\text{tw}}) \xrightarrow[4R]{\vec{p}} \mathcal{D}_{\text{out}}$ with probability 2^{-32}, similar to the 3-round trail of previous sections:



$\mathcal{D}^0_{\text{tw}}$ is the set of differences active in the first cell of the tweak; following the tweakey schedule of Kiasu-BC, this results in a tweakey difference in $\mathcal{D}^0_{\text{tw}}$ at each round. We use the same truncated trail (in reverse order) for the bottom and the returning parts. We obtain an 8-round boomerang with two 4-round differentials:

$$\vec{p} = 2^{-32}, \quad \overleftarrow{p} = 2^{-32}, \quad |\bar{\mathcal{D}}^0_{\text{in}}| = |\mathcal{D}^0_{\text{in}}| = 2^{32}, \quad |\mathcal{D}^0_{\text{out}}| = 2^{32}, \quad |\mathcal{D}^0_{\text{tw}}| = 2^8,$$
$$\overleftarrow{q} = 2^{-32}, \quad |\mathcal{D}^1_{\text{in}}| = 2^{32}, \quad |\mathcal{D}^1_{\text{out}}| = 2^{32}, \quad |\mathcal{D}^1_{\text{tw}}| = 2^8.$$

Following the analysis of Section 4.4.2.3, we deduce on average $\ell = 2^{-32}$ candidates of $\kappa = 64$ key bits per quartet. Therefore, we obtain

$$p_b = \vec{p} \cdot \overleftarrow{p} \cdot \overleftarrow{q}^2 \times |\mathcal{D}^1_{\text{in}}|^{-1} = 2^{-160},$$
$$p_w = |\mathcal{D}^0_{\text{in}}|/2^n \times \ell \times 2^{-\kappa} = 2^{-192}, \qquad \tilde{\sigma} = 2^{32}.$$

Since $\tilde{\sigma} \gg 1$, we only need a few right quartets. Taking $\mu = 4$, we obtain an attack with $Q = 2^{162}$ quartets. We take advantage of the tweak to build larger structures (iterating over the tweak and data inputs), of size $|\mathcal{D}^0_{\text{in}}| \cdot |\mathcal{D}^0_{\text{tw}}| \cdot |\mathcal{D}^1_{\text{out}}| \cdot |\mathcal{D}^1_{\text{tw}}| = 2^{80}$. Thus we only need 8 structures, with data complexity $D = 2^{83}$. In each structure of 2^{80} elements, we expect 2^{63} quartets with $\bar{P} + \bar{P}' \in \bar{\mathcal{D}}^0_{\text{in}}$, therefore the time complexity for the key-recovery is negligible, and $T = D = 2^{83}$. Note that unlike in the attacks against AES, we can not use muliple differential trails for the upper and lower parts, because the tweakey difference is in a predetermined position, and therefore the byte position of the differences in w_0, x_2, w_4, x_6 is fixed.

Success Probability. There are 2^{66} quartets with $\bar{P} + \bar{P}' \in \bar{\mathcal{D}}^0_{\text{in}}$, suggesting on average 2^{-32} key candidates each; hence a total of 2^{34} candidates for 64 bits of key.

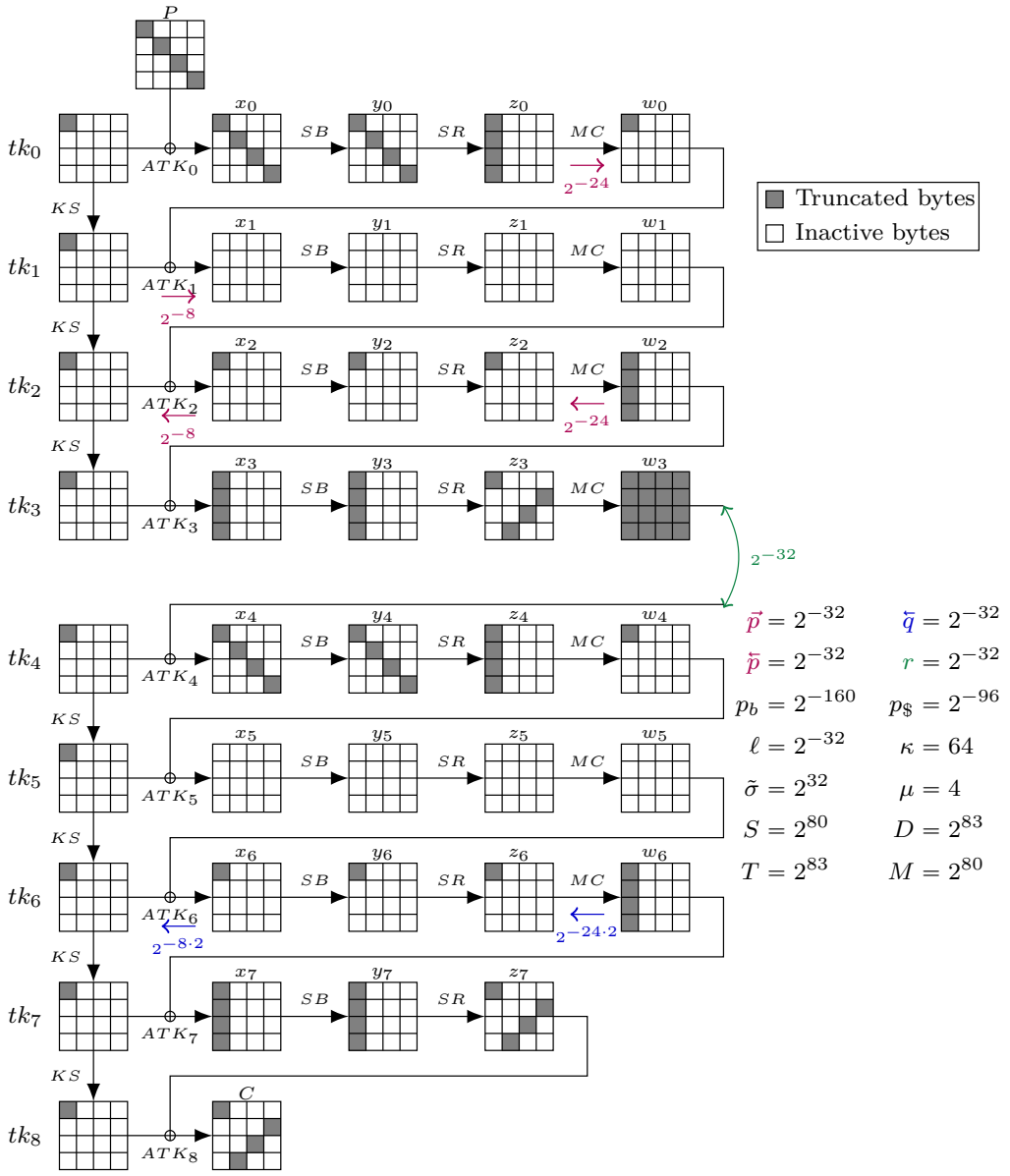


Figure 4.9: Truncated boomerang trail on 8-round Kiasu-BC, along with the key-recovery attack parameters. S is the structure size.

We keep key candidates whose counter reaches 2 or more. Modeling counters for wrong keys with a Poisson distribution with $\lambda = 2^{-30}$, the probability for a specific wrong key counter to be at least 2 is $1 - e^{-\lambda}(1+\lambda) \approx 2^{-61}$; therefore we expect to keep 8 wrong keys. On the other hand, the counter for the right key follows a Poisson distribution with $\lambda = 4$. It reaches a value of 2 or more with probability ≈ 0.9.

As in the AES attacks, we recover the full key by repeating the attack with $\mathcal{D}^0_{\text{in}}$ in a different diagonal. Taking advantage of the recovered values of the last round key, this adds a negligible complexity.

4.4.5 Application to TNT-AES

TNT-AES is an AES-based tweakable block cipher published in EUROCRYPT 2020 by Bao *et al.* [BGG+20], described in Section 2.3.4.4. We recall that TNT-AES uses three 128-bit keys K_0, K_1, K_2, and a 128-bit tweak, and is composed of the following operations:

$$\tilde{E}_{K_0,K_1,K_2} : P, T \mapsto C = \mathsf{AES}^6_{K_2}\Big(T + \mathsf{AES}^6_{K_1}\big(T + \mathsf{AES}^6_{K_0}(P)\big)\Big),$$

where AES^6_K denotes 6 AES rounds under the key K.

We present a marginal distinguisher on TNT-AES, that is far from reaching the complexity of the best known generic attack against TNT-AES, with 2^{69} operations [JKN+24]. However, our attack has the advantage of distinguishing between TNT-AES and TNT with a PRP.

4.4.5.1 Truncated boomerang

Our attack focuses on the middle cipher E_{K_1}, between both tweak additions. In order to skip the initial and final ciphers E_{K_0} and E_{K_2}, we introduce differences in the tweak, instead of introducing them in the plaintext and ciphertext. We fix a plaintext P, and consider four tweaks $T, T', \bar{T}, \bar{T}'$ to create quartets as follows:

1. Query $C = \tilde{E}(P, T)$ and $C' = \tilde{E}(P, T')$.
2. Query $\bar{P} = \tilde{E}^{-1}(C, \bar{T})$ and $\bar{P}' = \tilde{E}^{-1}(C', \bar{T}')$.
3. Detect when $\bar{P} = \bar{P}'$.

We denote the inputs and outputs of E_{K_1} as X and Y, with $Y = E_{K_1}(X)$:

$$X = E_{K_0}(P) + T, \quad X' = E_{K_0}(P) + T', \quad \bar{X} = E_{K_0}(\bar{P}) + \bar{T}, \quad \bar{X}' = E_{K_0}(\bar{P}') + \bar{T}',$$
$$Y = E^{-1}_{K_2}(C) + T, \quad Y' = E^{-1}_{K_2}(C') + T', \quad \bar{Y} = E^{-1}_{K_2}(C) + \bar{T}, \quad \bar{Y}' = E^{-1}_{K_2}(C') + \bar{T}'.$$

When $\bar{P} = \bar{P}'$, we have a boomerang quartet for E_{K1} with differences

$$X + X' = T + T' = \Delta_{\text{in}}, \qquad \bar{X} + \bar{X}' = \bar{T} + \bar{T}' = \Delta'_{\text{in}},$$
$$Y + \bar{Y} = T + \bar{T} = \nabla_{\text{out}}, \qquad Y' + \bar{Y}' = T' + \bar{T}' = \nabla'_{\text{out}}.$$

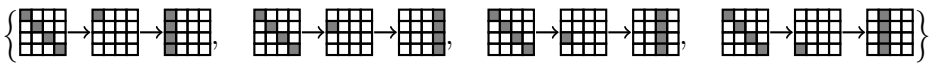
When using a truncated boomerang trail (with a fixed P and a set of tweaks), there are two important limitations compared to the previous attacks:

- We only detect when the difference $\bar{X} + \bar{X}'$ matches exactly $\bar{T} + \bar{T}'$, instead of detecting a set of differences $\bar{\mathcal{D}}^0_{\text{in}}$. This is equivalent to saying that $|\bar{\mathcal{D}}^0_{\text{in}}| = 1$. The boomerang trail probability decreases.
- We necessarily have $\Delta_{\text{in}} + \Delta'_{\text{in}} = \nabla_{\text{out}} + \nabla'_{\text{out}}$. For the 6-round AES truncated boomerang trail of Figure 4.8, this implies $\Delta_{\text{in}} = \Delta'_{\text{in}}$ and $\nabla_{\text{out}} = \nabla'_{\text{out}}$. Therefore, we cannot take advantage of structures on the ciphertext side.

Nonetheless, truncated boomerangs can be used with structures of tweaks on the plaintext side, and the analysis of the middle rounds as truncated differentials significantly reduces the complexity compared to the analysis of [BGG+20].

Our truncated boomerang attack against TNT-AES follows the characteristic of Figure 4.10, detailed below.

Upper Differential. We use the same collection of 4 upper trails as in our optimized attack on AES:



Note that the only accepted difference for $\bar{P} + \bar{P}'$ is exactly Δ_{in}, and no multi-trail can be performed in the backward upper trail. This gives the following parameters:

$$\vec{p} = 2^{-22}, \qquad |\mathcal{D}^0_{\text{in}}| = 2^{32}, \qquad |\mathcal{D}^0_{\text{out}}| = 2^{34},$$
$$\overleftarrow{p} = 2^{-56}, \qquad |\bar{\mathcal{D}}^0_{\text{in}}| = 2^0.$$

Lower Differential. Since we cannot use structures on the ciphertext side, we use a fixed value ∇_{out} to maximize the probability of the trail. We consider ∇_{out} active only on the main anti-diagonal, and analyze the S-box layer of the last round. The output difference of the S-box layer is equal to $\mathsf{ShiftRows}^{-1}(\mathsf{MixColumns}^{-1}(\nabla_{\text{out}}))$, and we want the input difference to be of one of the following types:

$$\begin{bmatrix} 2\alpha & 0 & 0 & 0 \\ \alpha & 0 & 0 & 0 \\ \alpha & 0 & 0 & 0 \\ 3\alpha & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} \alpha & 0 & 0 & 0 \\ \alpha & 0 & 0 & 0 \\ 3\alpha & 0 & 0 & 0 \\ 2\alpha & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} \alpha & 0 & 0 & 0 \\ 3\alpha & 0 & 0 & 0 \\ 2\alpha & 0 & 0 & 0 \\ \alpha & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 3\alpha & 0 & 0 & 0 \\ 2\alpha & 0 & 0 & 0 \\ \alpha & 0 & 0 & 0 \\ \alpha & 0 & 0 & 0 \end{bmatrix}.$$

We experimentally tested all possible differences on the main diagonal of ∇_{out}, and counted the number of pairs satisfying the transition. More precisely, we are interested in the joint probability that two different pairs reach an input difference of the same type, therefore we count the number of quartets of each type (using the DDT of the S-box for each possible input and output difference).

We found that in the best case, there are $2^{22.35}$ quartets that satisfy this transition (with the same type for both pairs), compared to 2^{18} quartets expected

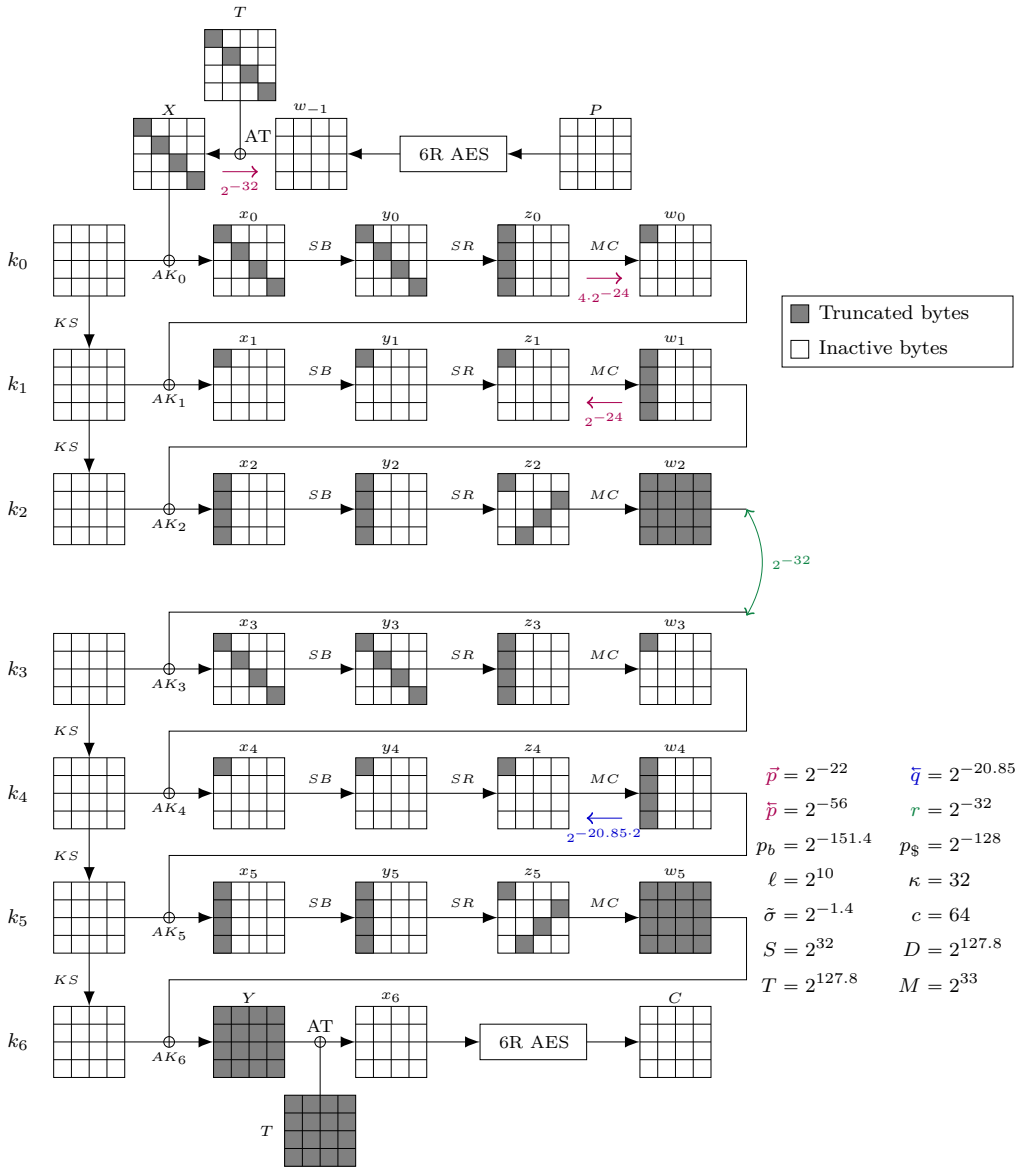


Figure 4.10: Scheme of our boomerang attack on the full TNT-AES. Although this only shows a unique boomerang trail which would imply $p_b = 2^{-151.7}$, the attack parameters are shown for $p_b = 2^{-151.4}$, following a thorough multi-trail analysis.

on average (for each type of input difference, we expect on average one pair for each of the 2^8 differences, therefore 2^{8+8} quartets). There are 4×255 choices of ∇_{out} reaching this maximum, and we found they they all have a special form: $\mathsf{ShiftRows}^{-1}(\mathsf{MixColumns}^{-1}(\nabla_{\text{out}}))$ is of one of the following types, with L the linear transform inside the AES S-box:

$$\begin{bmatrix} L(\beta/2) & 0 & 0 & 0 \\ L(\beta) & 0 & 0 & 0 \\ L(\beta) & 0 & 0 & 0 \\ L(\beta/3) & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} L(\beta) & 0 & 0 & 0 \\ L(\beta) & 0 & 0 & 0 \\ L(\beta/3) & 0 & 0 & 0 \\ L(\beta/2) & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} L(\beta) & 0 & 0 & 0 \\ L(\beta/3) & 0 & 0 & 0 \\ L(\beta/2) & 0 & 0 & 0 \\ L(\beta) & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} L(\beta/3) & 0 & 0 & 0 \\ L(\beta/2) & 0 & 0 & 0 \\ L(\beta) & 0 & 0 & 0 \\ L(\beta) & 0 & 0 & 0 \end{bmatrix}.$$

This special form can be explained by the structure of the AES S-box; we recall that the AES S-box is defined on F_{2^8} as:

$$S(x) = \begin{cases} A(x^{-1}) & \text{if } x \neq 0, \\ A(0) & \text{else.} \end{cases}$$

In particular, $S(x) = A(x^{254})$ holds for all $x \in \mathbb{F}_{2^8}$. Let us define L the linear part of A: $L(x) = A(x) - A(0)$. We have the following property:

Property 4.1. *For any $\alpha, \beta \neq 0$, $\mathsf{DDT}[\alpha, L(\beta)] = \mathsf{DDT}[\alpha\beta, L(1)]$.*

Proof.

$$\begin{aligned} \mathsf{DDT}[\alpha, L(\beta)] &= \left|\left\{x \,:\, L\left((x)^{254}\right) + L\left((x+\alpha)^{254}\right) = L(\beta)\right\}\right| \\ &= \left|\left\{x \,:\, (x)^{254} + (x+\alpha)^{254} = \beta\right\}\right| \\ &= \left|\left\{x \,:\, (\beta x)^{254} + (\beta x + \alpha\beta)^{254} = 1\right\}\right| \\ &= \left|\left\{x \,:\, (x)^{254} + (x+\alpha\beta)^{254} = 1\right\}\right| \\ &= \left|\left\{x \,:\, L\left((x)^{254}\right) + L\left((x+\alpha\beta)^{254}\right) = L(1)\right\}\right| \\ &= \mathsf{DDT}[\alpha\beta, L(1)]. \end{aligned}$$

□

Because of this property, S-box transitions of the form

$$(2\alpha, \alpha, \alpha, 3\alpha) \to (L(\beta/2), L(\beta), L(\beta), L(\beta/3))$$

have a higher probability than expected because the transition probability of the S-boxes are not independent: for a given α and β either all transitions are possible simultaneously, or none of the transitions are possible.

For instance, with $\beta = 1$, we obtain the following transitions through $\mathsf{MixColumns}^{-1} \circ \mathsf{SubBytes}^{-1}$:

$$\begin{aligned} \Pr\left[(\mathtt{14}, \mathtt{1f}, \mathtt{1f}, \mathtt{18}) \to (*, 0, 0, 0)\right] &= 2272/2^{32} \approx 2^{-20.85}, \\ \Pr\left[(\mathtt{14}, \mathtt{1f}, \mathtt{1f}, \mathtt{18}) \to (0, *, 0, 0)\right] &= 256/2^{32}, \\ \Pr\left[(\mathtt{14}, \mathtt{1f}, \mathtt{1f}, \mathtt{18}) \to (0, 0, *, 0)\right] &= 256/2^{32}, \\ \Pr\left[(\mathtt{14}, \mathtt{1f}, \mathtt{1f}, \mathtt{18}) \to (0, 0, 0, *)\right] &= 256/2^{32}. \end{aligned}$$

Using the trail $(\texttt{14}, \texttt{1f}, \texttt{1f}, \texttt{18}) \rightarrow (*, 0, 0, 0)$, we obtain $\overleftarrow{q} = 2^{-20.85}$. When combining the four boomerang trails, the probability that two pairs follow the same trail is increased from $\overleftarrow{q}^2 = 2^{-41.7}$ to:

$$\left(2272/2^{32}\right)^2 + 3 \times \left(256/2^{32}\right)^2 \approx 2^{22.35}/2^{64} = 2^{41.65}.$$

Refinement of r and $\overleftarrow{p}$. As such, our booomerang attack requires more than 2^{128} data to distinguish TNT-AES. We therefore need to perform a more detailled analysis to save a fraction of bits in the complexity attack.

In the lower trail, our analysis assumes that if $(C, \bar{C})$ and $(C', \bar{C}')$ both follow the trail, with the same pattern in z_4, then the differences $E_1^{-1}(C) + E_1^{-1}(\bar{C})$ and $E_1^{-1}(C') + E_1^{-1}(\bar{C}')$ are equal with probability $|\mathcal{D}_{\text{in}}^1|^{-1} = 2^{-32}$. Actually, the differences are not uniformly distributed in $\mathcal{D}_{\text{in}}^1$, and this increases the probability that the differences are equal.

Indeed, we can split this analysis into two disjoint cases: either the differences in x_4 are equal ($x_4 + \bar{x}_4 = x_4' + \bar{x}_4'$), or they are different. If they are equal, then the difference in y_3 is the same for both pairs; therefore there are only 127 possible differences in each active byte of x_3.

$$\begin{aligned}
&\Pr\left[E_1^{-1}(C) + E_1^{-1}(\bar{C}) = E_1^{-1}(C') + E_1^{-1}(\bar{C}')\right] \\
&= \overleftarrow{q}^2 \times \left(\frac{254}{255} \times 2^{-32} + \frac{1}{255} \times 127^{-4}\right) \\
&\approx \overleftarrow{q}^2 \times 2^{-31.915}.
\end{aligned}$$

This increases r from 2^{-32} to $2^{-31.915}$.

In the upper trail, there are similar effects. Our analysis assumes that the pair $(\bar{P}, \bar{P}')$ follows the truncated trail with probability $\overleftarrow{p}$ independently of the pair (P, P'). However, both pairs have the same differences at the input and output of the trail and the trail does not cover the sets $\mathcal{D}_{\text{in}}^0$ and $\mathcal{D}_{\text{out}}^0$ uniformly. Let us consider a pair $\bar{C}, \bar{C}'$ with $E_1^{-1}(\bar{C}) + E_1^{-1}(\bar{C}') = E_1^{-1}(C) + E_1^{-1}(C')$. The difference in y_2 (after the S-boxes of the third round) are the same for both pairs; the truncated trail allows a set 255 differences in x_2 (before the S-boxes). In general the probability of a transition through four active S-boxes is 2^{-32}, but we know that one of the 255 differences was followed by the pair P, P', therefore all differences are compatible, and the probability increases to 127^{-4}. Finally the probability of having a difference in x_2 compatible with the trail is higher than 2^{-24}:

$$1 \times 127^{-4} + 254 \times 2^{-32} \approx 2^{-23.92}.$$

In the first round, we have the same analysis as in the lower trail, and the probability to obtain the same difference as the pair P, P' is higher than 2^{-32}:

$$\frac{254}{255} \times 2^{-32} + \frac{1}{255} \times 127^{-4} \approx 2^{-31.915}.$$

This increases $\overleftarrow{p}$ from 2^{56} to $2^{-23.92} \cdot 2^{-31.915} \approx 2^{55.84}$.

Finally, we obtain the following paramters:

$$\vec{p} = 2^{-22}, \qquad |\mathcal{D}^0_{\text{in}}| = 2^{32}, \qquad |\mathcal{D}^0_{\text{out}}| = 2^{34},$$
$$\overleftarrow{p} = 2^{-55.84}, \qquad |\bar{\mathcal{D}}^0_{\text{in}}| = 2^0,$$
$$r = 2^{-31.915}, \qquad \overleftarrow{q}^2 = 2^{-41.65}, \qquad |\mathcal{D}^1_{\text{out}}| = 1.$$

4.4.5.2 Boomerang trail probability

We obtain:

$$p_b = \vec{p} \cdot \overleftarrow{p} \cdot \overleftarrow{q}^2 \times r = 2^{-151.4}, \qquad p_\$ = |\bar{\mathcal{D}}^0_{\text{in}}|/2^n = 2^{-128}.$$

It is not possible to recover actual key material with this attack because X is unkown. However, we can use $E_{K_0}(P) + K_1$ as an equivalent subkey if all queries are made with the same P. Using the pair (X, X') we extract $\ell = 2^{10}$ candidates for $\kappa = 32$ key bits. Unfortunately, we cannot use the pairs $(Y, \bar{Y})$ for filtering on the ciphertext side since the unkown value Y is different in each quartet. Similarly, the pair $(\bar{X}, \bar{X}')$ is unusable for key extraction. Therefore,

$$p_b = 2^{-151.4}, \qquad p_w = p_\$ \times \ell \times 2^{-\kappa} = 2^{-150}, \qquad \tilde{\sigma} = 2^{-1.4}.$$

With $\tilde{\sigma} < 1$, we need $Q = c \cdot \tilde{\sigma}^{-1} \cdot p_b^{-1}$ with a small constant c; we take $c = 64$, $Q = 2^{158.8}$. We have plaintext structures of size 2^{32}, and reuse each queried structure with different values of ∇_{out}, so that the data complexity is dominated by the number of decryptions, this corresponds to $D = Q \times 2^{-31} = 2^{127.8}$ ($2^{127.8}/255$ encryption queries and $2^{127.8}$ decryption queries).

4.4.5.3 Distinguisher

With $2^{127.8}$ queries we obtain a distinguisher between TNT-AES (using 6-round AES as the building block) and a PRP (or TNT using a PRP). This obviously does not threaten the security of TNT-AES, but we believe that it is an interesting use case showing that a 6-round boomerang distinguisher can be extended to a larger scheme, even if the attack is marginal.

After collecting the quartets, we expect that the counter corresponding to the right key follows the distribution $\mathcal{N}(2^{8.8} + 2^{7.4}, 2^{8.8})$ while counters for the wrong keys follow the distribution $\mathcal{N}(2^{8.8}, 2^{8.8})$ (the distance between the expected values is 8 times the standard deviation).

We obtain a distinguisher by observing whether the maximum counter is higher than a threshold $t = 2^{8.8} + 7 \times 2^{4.4}$. The probability that all counters for wrong keys are lower than t is $\Phi(7)^{2^{32}} \approx 0.995$, therefore the probability of false positive is 0.005. The probability that the counter for the right key is higher than t is $\Phi(1) = 0.84$ so the probability of false negative is 0.16.

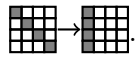
Finally, we can increase the success rate by running three attacks in parallel using three input sets $\mathcal{D}^0_{\text{in}}$, $\mathcal{D}^0_{\text{in}}{}'$, $\mathcal{D}^0_{\text{in}}{}''$ on three different diagonals. Using super-structures of 2^{96} values, we run all three attacks with the same queries, and

generate counters for three sets of 2^{32} equivalent keys. Using a threshold of $t = 2^{8.8} + 7.1 \times 2^{4.4}$, we keep the probability of false positive below 1%, while the probability that at least one of the three counters corresponding to right keys is higher than the threshold increases to 99%.

4.4.5.4 TNT with 5-round AES

If TNT is used with 5-round AES, we can build efficient boomerang attacks with a probability-1 truncated differential in one part. We re-use the analysis performed on TNT with 6-round AES to refine the different parameters of the attack.

Distinguisher. For a distinguisher, we split E_{K_1} into a 2-round E_0 and 3-round E_1. In E_0, we use a 2-round truncated trail with forward probability 1:



We have the following parameters:

$$\vec{p} = 1, \qquad \overleftarrow{p} = 2^{-32}, \qquad |\mathcal{D}^0_{\text{in}}| = 2^{32}, \qquad |\bar{\mathcal{D}}^0_{\text{in}}| = 2^0, \qquad |\mathcal{D}^0_{\text{out}}| = 2^{32}.$$

In the lower part E_1, we use the same trail as in the 6-round attack, with a fixed difference:

$$\nabla_{\text{out}} = \mathsf{MixColumns} \circ \mathsf{ShiftRows} \left(\begin{bmatrix} L(\beta/2) & 0 & 0 & 0 \\ L(\beta) & 0 & 0 & 0 \\ L(\beta) & 0 & 0 & 0 \\ L(\beta/3) & 0 & 0 & 0 \end{bmatrix} \right).$$

This gives:

$$r = 2^{31.915}, \qquad \overleftarrow{q}^2 = 2^{-41.65}, \qquad |\mathcal{D}^1_{\text{out}}| = 2^0.$$

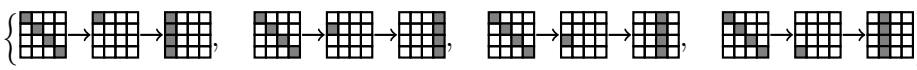
We obtain:

$$p_b = \vec{p} \cdot \overleftarrow{p} \cdot \overleftarrow{q}^2 \times r = 2^{-105.6}, \qquad p_\$ = |\bar{\mathcal{D}}^0_{\text{in}}|/2^n = 2^{-128}, \qquad \sigma = 2^{22.4}.$$

With $\sigma \gg 1$, we do not expect any false positive with $\overline{P} = \overline{P}'$, and we obtain a simple distinguisher with $Q = \mu \times p_b^{-1}$ for a small μ. For instance, with $Q = 2^{107}$, corresponding to $D = 2^{76}$, we have a success rate of 0.93.

Key-recovery. For a key-recovery attack, we use the same ideas as in the 6-round attack, in order to recover $E_{K_0}(P) + K_1$. This provides known plaintext/ciphertext pairs for E_{K_0}, and we use a low data complexity attack on 5-round AES to recover the master key. However, we have to split E_{K_1} into a 3-round E_0 and 2-round E_1 in order to extract information about $E_{K_0}(P) + K_1$.

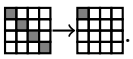
We use the same collection of 4 upper trails as in the 6-round attack:



We have the following parameters for the upper trail:

$$\vec{p} = 2^{-22}, \qquad |\mathcal{D}^0_{\text{in}}| = 2^{32}, \qquad |\mathcal{D}^0_{\text{out}}| = 2^{34},$$
$$\overleftarrow{p} = 2^{-55.84}, \qquad |\bar{\mathcal{D}}^0_{\text{in}}| = 2^0.$$

In the lower part, we use a 2-round truncated trail with backward probability 1, with a fixed (arbitrary) output difference:



We have the following parameters

$$r = 2^{-32}, \qquad \overleftarrow{q} = 1, \qquad |\mathcal{D}^1_{\text{out}}| = 2^0.$$

Using a single active output byte ensures that we do not have quartets that follow an alternative boomerang with a 2-round trail in the upper part with probability 1, and a 3-round trail in the lower part. We could increase the value of r with a more detailled analysis comparable to that of the attack on TNT with 6-round AES, but we chose not to do it to keep the analysis simple.

We obtain:

$$p_b = \vec{p} \cdot \overleftarrow{p} \cdot \overleftarrow{q}^2 \times r \approx 2^{-110}, \qquad p_\$ = |\bar{\mathcal{D}}^0_{\text{in}}|/2^n = 2^{-128}, \qquad \sigma = 2^{18}.$$

As in the 6-round attack, we recover $\ell = 2^{10}$ candidates for the main diagonal of $E_{K_0}(P) + K_1$ from each candidate quartet ($\kappa = 32$ bits of information). If we have at least two candidate quartets, the correct equivalent key is ranked first with high probability. Therefore we use $Q = 2^{113}$, so that the probability to have at least two candidate quartets is higher than 0.99. This corresponds to an attack with $D = 2^{82}$.

In order to recover the master key, we repeat the boomerang attack on the four diagonals to recover the full value of $E_{K_0}(P) + K_1$. Therefore, we obtain a known plaintext/ciphertext pair for E_{K_0}, and we use a low data complexity attack on 5-round AES. Using the attack from [Der13], we only need 8 chosen plaintexts to attack 5-round AES with a complexity of 2^{64}. Therefore, the total attack has a complexity of $8 \times 4 \times 2^{82} = 2^{87}$.

4.4.6 Modeling the Framework using MILP

We now introduce a MILP model to search for truncated boomerang attacks on AES-based ciphers, based on [CHP+17]. It uses our analysis of truncated boomerangs to evaluate the data complexity of an attack, and minimizes it.

Mixed Integer Linear Programming (MILP) is a mathematical optimization tool that minimizes a linear objective function of constrained variables. The variables can be either integer or rational, and the constraints are given as linear inequalities. In the last years, it has proven to be a useful tool to evaluate the security of cryptographic primitives, due to the facility of encoding cryptographic properties

as MILP problems, and the availability of high performance solvers. For more information on MILP modeling, refer to Section 2.4.

This method was applied to the search of boomerang distinguisher on Deoxys [JNP+21] by Cid *et al.* [CHP+17]. Their MILP model encodes the activity of each state byte with a binary variable that equals 1 if its corresponding byte is active and 0 if not, and that is constrained depending on the activity pattern of Deoxys operations. In order to build a boomerang trail, their model includes two separate differential trails with two overlapping rounds in the middle (in order to account for the ladder switch and the BCT analysis). The objective function to minimize is the number of active S-boxes, i.e. the sum of all variables representing the activity of S-box input (or output) bytes, which roughly corresponds to the attack complexity.

After generating the optimal boomerang template, they instantiate active bytes with concrete differences that maximize S-box transition probabilities. An important contribution of their work is an analysis of the degrees of freedom of the tweakey differences. Their MILP model counts the number of linear relations between the tweakey differences and ensures that at least one degree of freedom remains in the final trail, otherwise it is unlikely to find concrete differences for the tweakey.

In 2019, Zhao et al. [ZDJ19; ZDJ+19] improved this MILP model by adding two extra rounds at the end of the lower trail, containing truncated differences.

4.4.6.1 Our MILP model

Previous works [CHP+17; Sas18a; ZDJ+19] showed that the best attack is not always obtained with the best distinguisher. Therefore we follow the same high-level approach as in [QDW+21]: our main objective is to cover the full boomerang attack with the MILP model. The code for the MILP model is available in [BL23a].

Our model is not symmetric when switching plaintext and ciphertext[3], but for simplicity we only describe the attack starting from the plaintext, though it works similarly the other way around. Also, in order to have only two different trails, we consider than $\bar{\mathcal{D}}^0_{\text{in}} = \mathcal{D}^0_{\text{in}}$ and that the pairs follow the same upper differential trails in the forward and backward directions.

The model of Cid *et al.* [CHP+17] considers only two types of internal differences: either it is inactive with a zero difference, or it is active with a fixed non-zero difference. In order to model truncated trails, we consider four types of differences for all intermediate state variables (for the tweakey variables, we use only the first two types for simplicity):

- inactive, with a zero difference, denoted as □;
- active with a *fixed* non-zero difference, denoted as ■;

[3]The inverse AES round can be re-written with the same form as the encryption, but this requires to use equivalent round keys, and interacts badly with the introduction of sparse differences through the tweakey schedule.

- active with an *unknown* (truncated) difference, denoted as ■.
- active with an *equal* (but unknown) difference for both pairs, denoted as ■.

From these possible states, we can deduce the parameters of the attacks, including the structure size, which allows us to ask the MILP solver to minimize the formula for the data complexity of the attack given in Section 4.4.2.

Bytes with *equal* differences encode relations between the two different pairs that follow the same trail, rather than properties of a trail by itself. This allows the MILP model to capture trails like the 6-round AES boomerang of Figure 4.8 or the 8-round Kiasu boomerang of Figure 4.9; Figure 4.11 shows how they are captured by this model. The model does not encode linear relations between active bytes (*e.g.* the set of differences for w_2 of Figure 4.8 is active on all bytes but has size 2^{32}), but using this type of constraint is sufficient in many cases because it is propagated through the linear layer. In terms of analysis, we treat them specially: for the first pair they are considered as truncated bytes, but for the second pair they are considered as fixed differences (fixed to the value given by the first pair). Therefore, we explain the rest of the MILP model assuming that each trail has been duplicated, and *equal* differences have been replaced.

4.4.6.2 Constraints

We have constraints for each operation:

SubBytes. With truncated differences, the variables before and after the S-box layer do not necessarily have the same type. However, an S-box output is active (truncated or not) if and only if the input is active.

ShiftRows. ShiftRows only moves the bytes, so we have trivial equalities between the corresponding state variables.

MixColumns. The MixColumns operation operates on each column, multiplying it with an MDS matrix. Because of the MDS property, each column is either completely inactive, or has at least 5 active bytes (truncated or not) on input and output. Moreover, we have the same property with truncated bytes: either no byte is truncated, or at least 5 bytes are truncated.

For Deoxys-BC, we also reuse the constraints for the degrees of freedom given in [CHP+17] to avoid over-defining the fixed differences (for these constraints, the *equal* bytes are considered as truncated).

AddTweaKey. The AddTweaKey operation is just a XOR with the subkey. In our model, the subkey is not truncated, so it is either inactive, or active with a fixed difference. Therefore, the input state is truncated if and only if the output state is truncated. Otherwise, we use the constraints of Cid *et al.* to model XOR for the active bytes and the activity of the tweakey bytes.

Key Schedule. For Deoxys-BC, we follow the approach of [CHP+17] to model the key schedule. The permutation h just moves the bytes, and the LFSR construction ensures that each byte is either completely inactive, or inactive in at most $i-1$ rounds in the TK_i model. For Kiasu-BC, the 64-bit tweak is represented with 8 activity variables. For AES-128, the subkeys are inactive.

Additional Constraint. To avoid the trivial truncated trail with probability 1, we constrain the trail to have at most 3 active truncated columns in a single state, except for the first and last rounds.

4.4.6.3 Objective function

Using the results from the previous sections, we estimate the data complexity of an attack as:

$$\begin{aligned} D &= \max(\sqrt{2Q}, 2Q \times |\mathcal{D}^0_{\text{in}}|^{-1} \times |\mathcal{D}^1_{\text{out}}|^{-1}), && \text{with} \\ Q &\approx \max(p_b^{-1}, \tilde{\sigma}^{-1} \times p_b^{-1}), && \tilde{\sigma} \approx p_b/(p_\$ \times \ell \times 2^{-\kappa}), \\ p_b &= \vec{p} \cdot \overleftarrow{p} \cdot \overleftarrow{q}_1 \cdot \overleftarrow{q}_2 \times r, && p_\$ = |\mathcal{D}^0_{\text{in}}|/2^n. \end{aligned}$$

Since all variables are represented logarithmically by the MILP model, these formulas only involve additions and maximums, and are easily expressed with the MILP variables:

- $|\mathcal{D}^0_{\text{in}}|$ and $|\mathcal{D}^1_{\text{out}}|$ are obtained by counting truncated input/output bytes.
- $\vec{p}$, $\overleftarrow{p}$, $\overleftarrow{q}_1$ and $\overleftarrow{q}_2$ are computed from SubBytes and MixColumns transitions. Due to the *equal* bytes, the probability of the lower trail is computed twice: $\overleftarrow{q}_1$ after replacing them with truncated bytes, and $\overleftarrow{q}_2$ after replacing them with fixed bytes. Similarly $\vec{p}$ is computed after replacing them with truncated bytes, and $\overleftarrow{p}$ after replacing them with fixed bytes (see Table 4.4);
- r is computed from the boomerang connection probability of each S-box, as described in Table 4.4;
- $\ell \times 2^{-\kappa}$ is estimated as $\vec{p}_0{}^2 \cdot \overleftarrow{q}_0{}^2$ following Section 4.4.2.3, where the probability $\vec{p}_0$ for the first round and $\overleftarrow{q}_0$ for the last round are evaluated from the trail.

Trail Probability. With truncated differences, both MixColumns and SubBytes operations contribute to the probability of a trail. If at least one byte of a column is truncated, the probability of a MixColumns transition is 2^{-8t} where t is the number of non truncated bytes (active or not) in the output of MixColumns.

The SubBytes probability is computed by multiplying the transition probability of all the S-boxes, following the transition probabilities given in Table 4.4.

Table 4.4: Transition probability (DDT) and connection probability (BCT) for the AES S-box. For the transition probability of *equal* differences, we distinguish the first and the second pair. We omit the cases where the probability is 0.

Legend
□ $\delta = 0$
■ $\delta \neq 0$ fixed
■ unknown δ
■ *equal* δ

Transition probability	1st pair	2nd pair
$\Pr(□ \to □) = 1$	$\Pr(■ \to ■) \overset{1}{=} 1$	$\Pr(■ \to ■) \overset{2}{=} 2^{-7}$
$\Pr(■ \to ■) = 2^{-6}$	$\Pr(■ \to ■) \overset{1}{=} 1$	$\Pr(■ \to ■) \overset{2}{=} 2^{-8}$
$\Pr(■ \to ■) = 1$	$\Pr(■ \to ■) \overset{1}{=} 2^{-8}$	$\Pr(■ \to ■) \overset{2}{=} 2^{-7}$
$\Pr(■ \to ■) = 2^{-8}$	$\Pr(■ \to ■) \overset{1}{=} 1$	$\Pr(■ \to ■) \overset{2}{=} 1$
$\Pr(■ \to ■) = 1$	$\Pr(■ \to ■) \overset{1}{=} 1$	$\Pr(■ \to ■) \overset{2}{=} 2^{-7}$

Boomerang connection probability			
$\Pr(□ \to □) = 1$	$\Pr(■ \to □) = 1$	$\Pr(■ \to □) = 1$	$\Pr(■ \to □) = 1$
$\Pr(□ \to ■) = 1$	$\Pr(■ \to ■) = 2^{-6}$	$\Pr(■ \to ■) = 1$	$\Pr(■ \to ■) = 2^{-8}$
$\Pr(□ \to ■) = 2^{-8}$	$\Pr(■ \to ■) = 2^{-8}$	$\Pr(■ \to ■) = 1$	$\Pr(■ \to ■) = 2^{-8}$
$\Pr(□ \to ■) = 1$	$\Pr(■ \to ■) = 2^{-8}$	$\Pr(■ \to ■) = 1$	$\Pr(■ \to ■) = 2^{-8}$

Boomerang Connection Probability. In Section 4.4.1 and Section 4.4.2, we evaluated the connection probability as $r = |\mathcal{D}_{\text{in}}^1|^{-1}$. This is sufficient for AES boomerangs, but when targeting Deoxys, we obtain more accurate results using the Boomerang Connectivity Table (BCT) [CHP+18], as detailed in Section 4.2.3. Instead of splitting the cipher in two parts $E = E_1 \circ E_0$, we split it in three parts $E = E_1 \circ E_m \circ E_0$ with E_m an S-box layer. Given fixed differences on both sides of E_m, the probability that the boomerang connects is given by multiplying the BCT probabilities of each S-box. Because the MILP does not handle fixed difference instantiation, our MILP program uses the optimistic probabilities given in Table 4.4.

This analysis can be improved using the ladder switch [BK09]. Instead of splitting the cipher with E_m a full S-box layer, we use the Super-Box representation of the two middle rounds: from one S-box layer to the next, the AES round operates as four independent parallel transformations. Each of those four transformations can independently be considered as part of the upper or lower trail. We obtain E_0 and E_1 with partial rounds in the middle, and E_m corresponds to S-boxes of different rounds. We model the ladder switch using a binary variable for each Super-Box in the middle, encoding whether it is part of the upper trail or the lower trail. Depending on these variables, the S-boxes and MDS matrices in the middle are counted as part of E_0, E_m, or E_1.

To instantiate boomerang trails generated by the MILP solver, we iterate over all possible differences in the tweakey following the tweakey difference pattern given by the MILP solver, determine the fixed differences in the trail, and finally compute the exact trail probability with the tables DDT, BCT [CHP+18], UBCT, LBCT and EBCT, introduced in [SQH19; WP19; DDV20], presented in Section 2.2.2

and in Section 4.2.3. We then select the tweakey difference with the highest trail probability.

Limitations of the MILP model. Our model handles only fixed differences in the tweakey. More importantly, although our model takes into account the ladder switch to compute the boomerang connection probability, it does not accurately compute the connection probability. Some boomerang trails given in a preprint version of our paper [BL22] were even found incompatible by Yang *et al.* [YSS+22]. More generally, some boomerang trails returned by the MILP are not instantiable. When that happens in practice, we modify the boomerang trail squeleton or we generate a different trail using the MILP solver.

The reasons behind the incompatibilities found by Yang *et al.* are subtle, and in order to ensure that the trails are not incompatible, we verified experimentally the probability of the middle rounds [BL23a].

For each trail, we indicate the rounds that have been checked, with the theoretical probability p_{th} for the middle rounds, and the experimental value observed p_{exp}. In some attacks, the experimental probability slightly differs from the theoretical one; we deduce an adjusted trail probability $\tilde{p}_b$ that is used to calculate the complexity of the attack.

Resource Usage. For the search, we use Gurobi [Gur23] with 96 threads on a machine with two AMD EPIC 7352 CPUs and 256 GB of RAM. The solving time varies from a few minutes to several days, but the best trail is in general found much faster than the optimality proof. Memory shortage (exceeding 256 GB) is an issue for some of the largest models.

4.4.6.4 Results on AES-128 and Kiasu-BC

We use the MILP model to search for attacks on AES-128 and Kiasu-BC and compare them with the results of the previous sections.

On AES-128, the model returns the trail of Figure 4.11, which corresponds to the attack of Section 4.4.2.4 with a full *equal* state in w_2. This confirms the optimality of our truncated trail within our framework. However, the model does not handle multiple trails, so that it cannot suggest the improved attack of Section 4.4.3.

On Kiasu-BC, the model cannot find the attack of Section 4.4.4, because it does not handle truncated differences in the tweak. With fixed difference in the tweak, the best trail found by the solver is unfortunately not instantiable, because of incompatibilities in the middle rounds. The solver nevertheless ensures that no attack with complexity less than 2^{80} exist in that framework with fixed tweak differences.

4.4.7 Application to Deoxys-BC

Deoxys-BC [JNP+21] is a tweakable block cipher following the TWEAKEY framework [JNP14b] based the AES round function, on which the best known

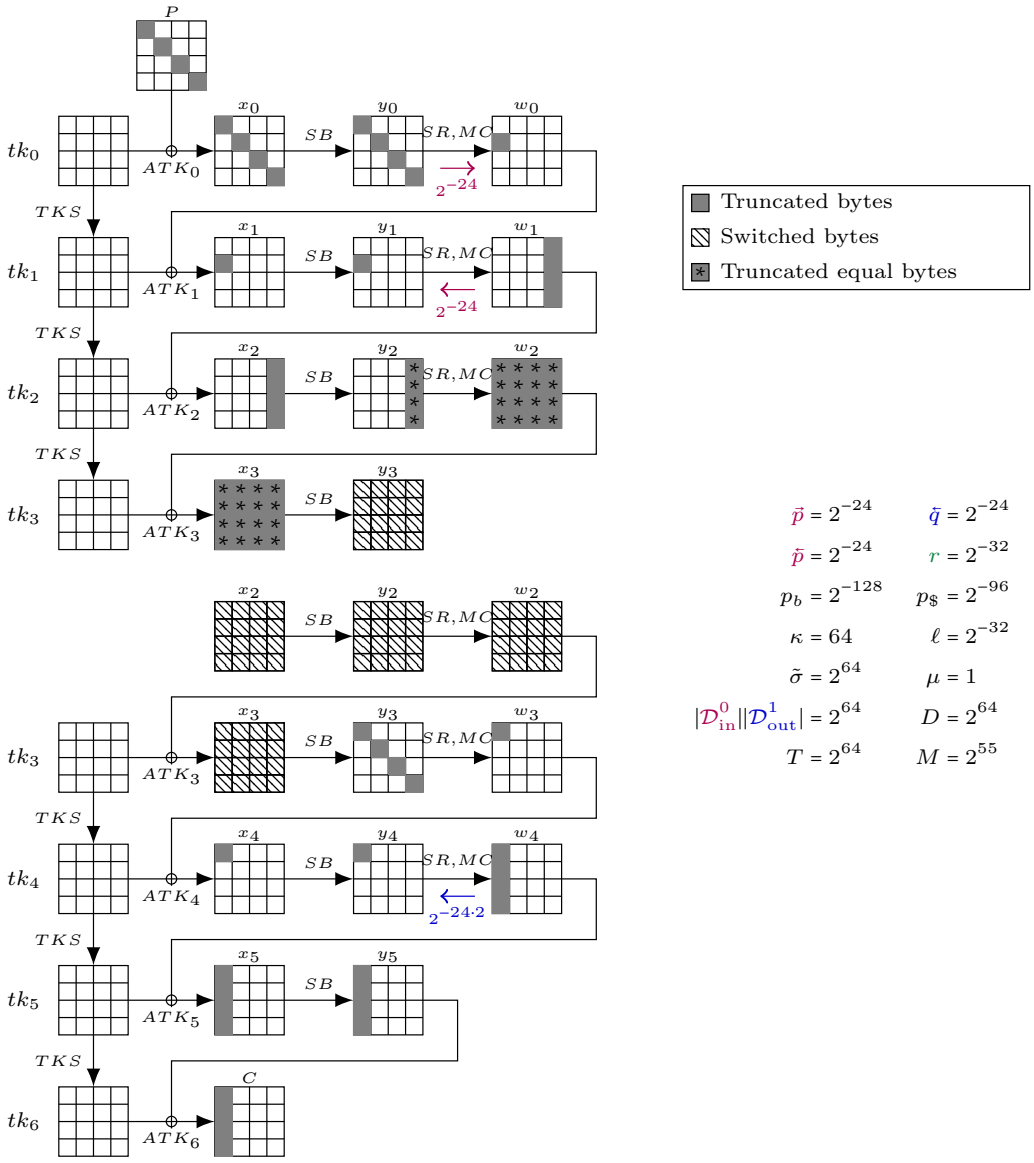


Figure 4.11: Truncated boomerang trail on 6-round AES, starting from the plaintext side, returned by the MILP model with *equal* differences in the middle.

attacks are based on boomerangs [CHP+17; Sas18a; ZDJ19; ZDJ+19]. Deoxys-BC is described in Section 2.3.4.2. Due to the large choice of tweakey differences, finding the best truncated boomerang trails manually is a tedious work. Instead, we use our MILP model of Section 4.4.6.

In the single tweakey model, the analysis is exactly that of the AES, and the best known boomerang attack is given in Section 4.4.3.

In the related tweakey model, the attacker can insert differences in some of the tweakey words TK^i. Depending on the tweak size and differences used, this can be either a single-key attack with chosen tweaks, or a related-key attack. We denote as RTK_j a model with differences in j 128-bit states, corresponding to:

- RTK_1 : single-key attacks on any variant with at least 128 bits of tweak.
- RTK_2 : single-key attacks on Deoxys-BC-384 with 256 bits of tweak, or related-key attacks on Deoxys-BC-256.
- RTK_3 : related-key attacks on Deoxys-BC-384.

For 13-round Deoxys-BC in the RTK_3 model, we selected a non-optimal trail in terms of data complexity, which was better in time complexity. For 8-round and 9-round Deoxys-BC in the RTK_1 model and 10-round Deoxys-BC in the RTK_2 model, we modified the squeleton of the trail returned by the MILP because the original one was not instantiable. During other difference instantiations, we sometimes applied slight manual improvements. For instance, for minor gains, we introduced state changing bytes: fixed on the forward trail but truncated on the return trail.

4.4.7.1 Description of the attacks

In the related-tweakey model, the attacker queries two sets of $|\mathcal{D}^0_{\text{in}}|$ plaintexts (under tweaks T and T'), and each ciphertext is shifted $|\mathcal{D}^1_{\text{out}}|$ times with a new tweak ($\bar{T}$ and $\bar{T}'$ respectively). In total a structure of size $S = |\mathcal{D}^0_{\text{in}}| \times |\mathcal{D}^1_{\text{out}}|$ requires $2|\mathcal{D}^0_{\text{in}}| + 2S \approx 2S$ queries (or $4S$ if $|\mathcal{D}^1_{\text{out}}| = 1$) and generates S^2 quartets.

Deoxys-BC is not perfectly symmetrical, and on some instances the best attack captured by the MILP model starts from the ciphertext side. In this case, we keep the same analysis by replacing respectively $\mathcal{D}^0_{\text{in}}, \mathcal{D}^1_{\text{out}}, \vec{p}, \bar{p}, \bar{q}$ by $\mathcal{D}^1_{\text{in}}, \mathcal{D}^0_{\text{out}}, \bar{q}, \vec{q}, \vec{p}$.

The values of ℓ and κ mentioned on the figures are the one used in our attacks, corresponding either to a 1-round or to a 2-round key-recovery. Each attack recovers a partial key, aiming for a success rate of $1/2$, comparable to previous analysis; we assume that the rest of the key can be recovered efficiently afterwards. When $\tilde{\sigma} \gg 1$, the number μ of right quartets required varies from 1 to 4. In particular, if $p_b \gg \ell \cdot p_\$$, we expect no wrong quartet and $\mu = 1$ suffices, else several right quartets are needed to get the correct key ranked first.

In this section, we describe the attacks found in various settings, summarized in Table 4.3.

4.4.7.2 8-round Deoxys-BC in the RTK$_1$ model (Figure 4.12)

Query 2^{15} structures of 2^{72} elements. On average, $\mu = 2^{15} \cdot 2^{72} \cdot 2^{72} \cdot \tilde{p_b} = 2$ quartets follow the trail and $2^{15} \cdot 2^{72} \cdot 2^{72} \cdot p_\$ = 2^{79}$ random quartets are detected. For each quartet, retrieve on average 2^{-16} values of $tk_8[4,6]$ in a few table accesses. $2^{79} \cdot 2^{-16} = 2^{63}$ quartets remain. For each remaining quartet, retrieve 2^{-24} values of $tk_0[1,6,12]$. For each of the $2^{63} \times 2^{-24} = 2^{39}$ remaining quartets, deduce on average 2^{-16} key candidate for $tk_8[0,1,2,3]$ and $tk_7^{eq}[1,3]$. For each of the 2^{23} remaining quartets, increase the counter of the retrieved 88-bit key candidate. Only the right counter is expected to be greater than 2. The complexity is dominated by the encryption queries. This gives $(D,T,M) = (2^{88}, 2^{88}, 2^{73})$.

4.4.7.3 9-round Deoxys-BC in the RTK$_1$ model (Figure 4.13)

Query 2^6 full structures of 2^{128} plaintexts. Each structure is built with a different tweak. If we are in the related key model rather than the related tweak model, we ask for structures under different linked keys. On average, $\mu = 2^6 \cdot 2^{128} \cdot 2^{128} \cdot \tilde{p_b} = 4$ quartets follow the trail. For each element of the structure, deduce on average 1 candidate k for 38 bits of key on the ciphertext side: 1 candidate for $tk_9[0,1,2,4]$ and 1 representant of the 4 possible candidates for $tk_9[6]$.

Guess 5 key bytes: $tk_0[0,5,10,15]$ and $tk_1[1]$, so that $w_0[0,1,2,3]$ and $y_1[1]$ can be evaluated from each plaintext. Set $\delta = \texttt{0x3b000000} \| \texttt{0x000000} \| \texttt{0x000000} \| \texttt{0x80} \| \texttt{0x80} \| 0$ and look for collisions between:

$$v = \big(\bar{P}[2,7,8,13] \ \| \ w_0[0,2,3] \| \ \bar{w}_0[0,2,3] \| \ y_1[1] \| \ \bar{y}_1[1] \| \ k\big),$$
$$v' = \big(\bar{P}'[2,7,8,13] \| \ w_0'[0,2,3] \| \ \bar{w}_0'[0,2,3] \| \ y_1'[1] \| \ \bar{y}_1'[1] \| \ k'\big) \ + \delta.$$

Since we match on 134 bits, we expect $2^{6+128+128-134} = 2^{128}$ remaining quartets for each guess, or 2^{168} in total, with a complexity of $2^{6+128+40} = 2^{174}$.

Extract more key information from the remaining quartets. First, retrieve 2^{-16} candidates for $tk_0[3,4,9,14]$; this requires about 2^9 table accesses per quartet, which is less than to 2^6 encryptions, and therefore does not dominate the time complexity; 2^{152} quartet remain. Then retrieve 2^{-16} candidates for $tk_0[1,6,11,12]$, and 2^{-16} candidates for $tk_1[6,12]$. 2^{120} candidate quartets remain. For each remaining quartet, for all possible $tk_9[3]$, deduce 2^{-6} representant of the 4 possible $tk_8^{eq}[1]$ and 2^{-7} representant for the 2 possible $tk_8^{eq}[3]$. This step allows to deduce $2^{8-6-7} = 2^{-5}$ 21-bit key canditates. We end up with 2^{115} candidates for 179 bits of key.

We model the counters for wrong keys as a Poisson distribution with $\lambda = 2^{-64}$; they reach 4 or more with probability $2^{-260.6}$, therefore the right key is expected to be ranked first. This gives $(D,T,M) = (2^{135}, 2^{174}, 2^{129})$.

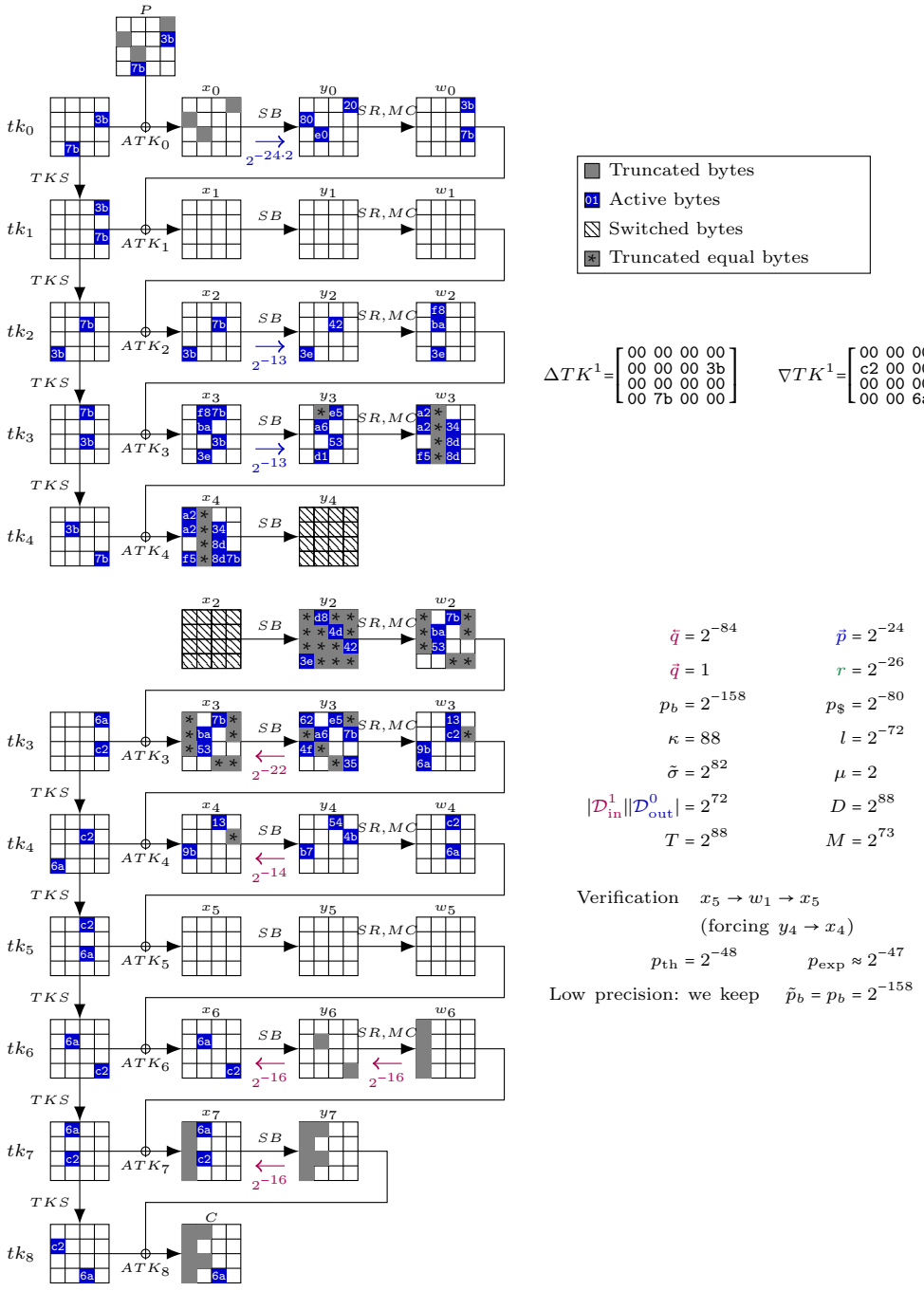


Figure 4.12: Truncated boomerang attack on 8-round Deoxys-BC in the RTK_1 model, starting from the ciphertext side. This attack succeeds with probability 1/2. Middle rounds are analyzed with UBCT, LBCT and EBCT (probabilities on the trail).

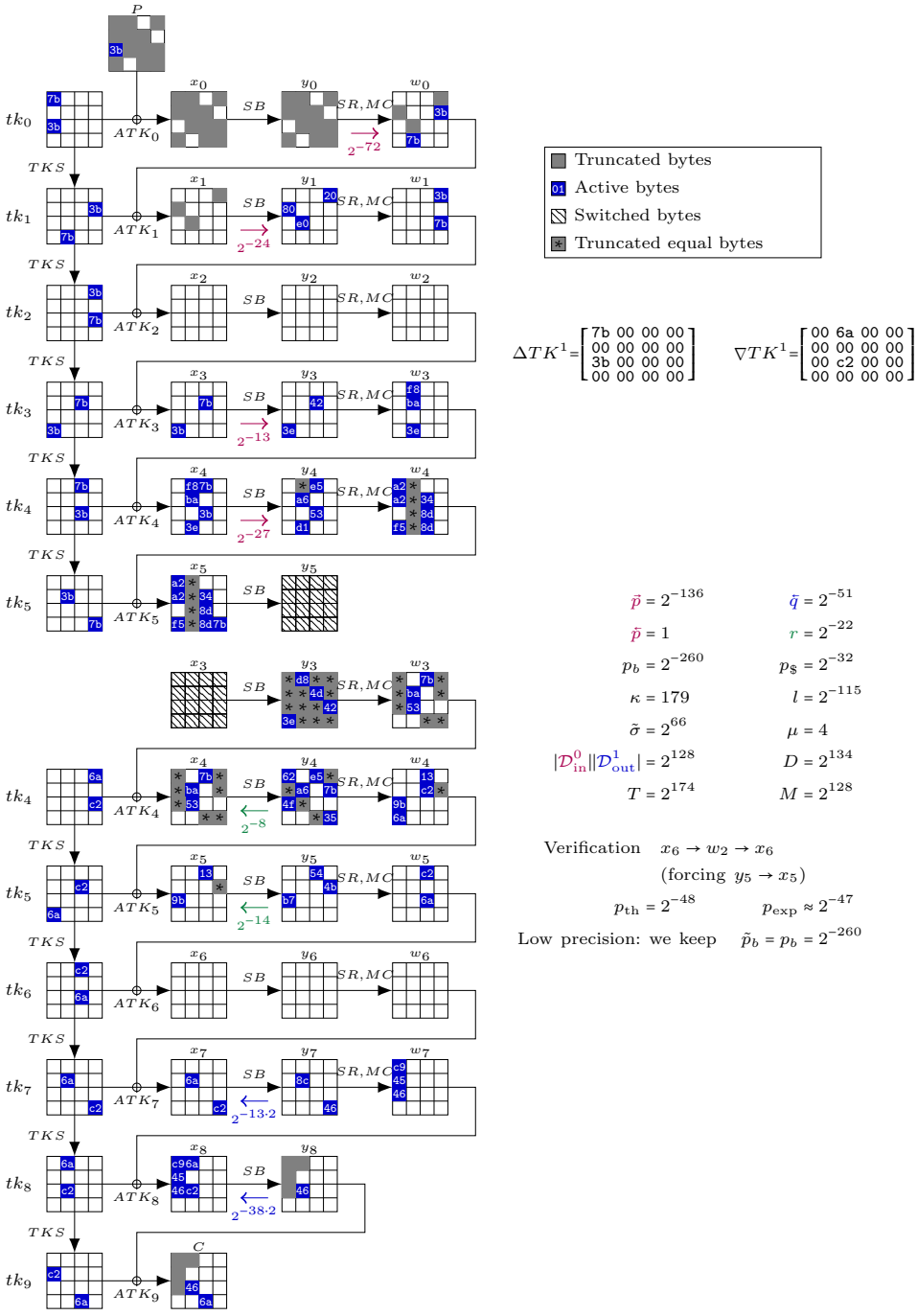


Figure 4.13: Truncated boomerang attack on 9-round Deoxys-BC in the RTK_1 model, starting from the plaintext side. This attack succeeds with probability 1/2. Middle rounds are analyzed with UBCT, LBCT and EBCT (probabilities on the trail).

4.4.7.4 8-round Deoxys-BC in the RTK$_2$ model (Figure 4.14)

Query a partial structure of 2^{25} ciphertexts. The only detected quartet is a right quartet. Thus, $(D, T, M) = (2^{27}, 2^{27}, 2^{27})$.

4.4.7.5 9-round Deoxys-BC in the RTK$_2$ model (Figure 4.15)

Query a partial structure of $2^{54.2}$ ciphertexts. On average, $\mu = 2^{54.2} \cdot 2^{54.2} \cdot \tilde{p_b} = 1$ quartets follow the trail and $2^{54.2} \cdot 2^{54.2} \cdot p_\$ = 2^{20.4}$ random quartets are detected. For each quartet, retrieve 2^{-52} values of $tk_0[0, 5, 10, 15]$ and $tk_9[0, 1, 2, 3, 5]$. This step is of negligible complexity, and with high probability no wrong quartet remains. Thus, $(D, T, M) = (2^{55.2}, 2^{55.2}, 2^{55.2})$.

4.4.7.6 10-round Deoxys-BC in the RTK$_2$ model (Figure 4.16)

Query one partial structure of $2^{93.2}$ ciphertexts, so that on average, $\mu = 2^{93.2+93.2} \cdot \tilde{p_b} = 2$ quartets follow the trail. For each element of the structure, deduce on average 1 candidate for 30 bits of key on the plaintext side: 1 candidate for $tk_0[4, 9, 14]$ and 1 representant of the 4 possible candidates each for $tk_0[3]$[4]. In total, there are on average $2^{93.2+93.2-56-30} = 2^{100.4}$ candidate quartets matching on the ciphertext bytes with a known difference and on the key candidate.

For each quartet, retrieve 2^{-8} candidates for $tk_{10}[9]$ with 2 table accesses. This costs $2^{101.4}$ table accesses, and since an encryption makes $10 \times 16 = 2^{7.3}$ S-box calls, this step costs $2^{94.1}$ equivalent encryptions. For each of the $2^{100.4-8} = 2^{92.4}$ remaining quartets, retrieve on average 2^{-32} candidates of $tk_{10}[0, 1, 2, 3, 4, 5, 6, 7]$. Finally, recover 2^{-16} candidates for $tk_9^{eq}[1, 6]$. There remains $2^{92.4-32-16} = 2^{44.4}$ quartets with a 118-bit key candidate. The only candidate suggested twice is expected to be the right candidate. The time complexity is $2^{94.2} + 2^{94.1} \approx 2^{95.2}$, thus $(D, T, M) = (2^{94.2}, 2^{95.2}, 2^{94.2})$.

4.4.7.7 11-round Deoxys-BC in the RTK$_2$ model

The MILP solver did not return a pertinent trail for this key setting. Instead, we use the 10-round trail and append a round at the beginning. First, query the full encryption codebook with $\bar{T}, \bar{T}'$ and store it. Then, guess the full tk_0. Perform the 10-round attack, by using the same ciphertext structure for each guess of tk_0 and simulating encryption queries with fetches in the codebook. We chose $\mu = 4$ and for each key guess, the 10-round attack with partial structures of $2^{93.7}$ elements gives $2^{45.4}$ candidates for 118 bits, for a time complexity of $2^{95.1}$. If we suppose that a fetch to the codebook costs an encryption in time complexity, we end up with $T = 2^{128}(2^{94.7} + 2^{95.1}) = 2^{223.9}$. The probability that one of the counters is at least 4 is $2^{-295+116+128} = 2^{-51}$, so on average, the correct key is ranked first. This gives $(D, T, M) = (2^{129}, 2^{223.9}, 2^{129})$.

[4] S-box 3 on the plaintext side has two pairs $(x, x + \delta), (x', x' + \delta)$ following the transition fixed by the trail. Instead of listing four key candidates, we identify one of the 2^6 cosets of $\langle \delta, x + x' \rangle$.

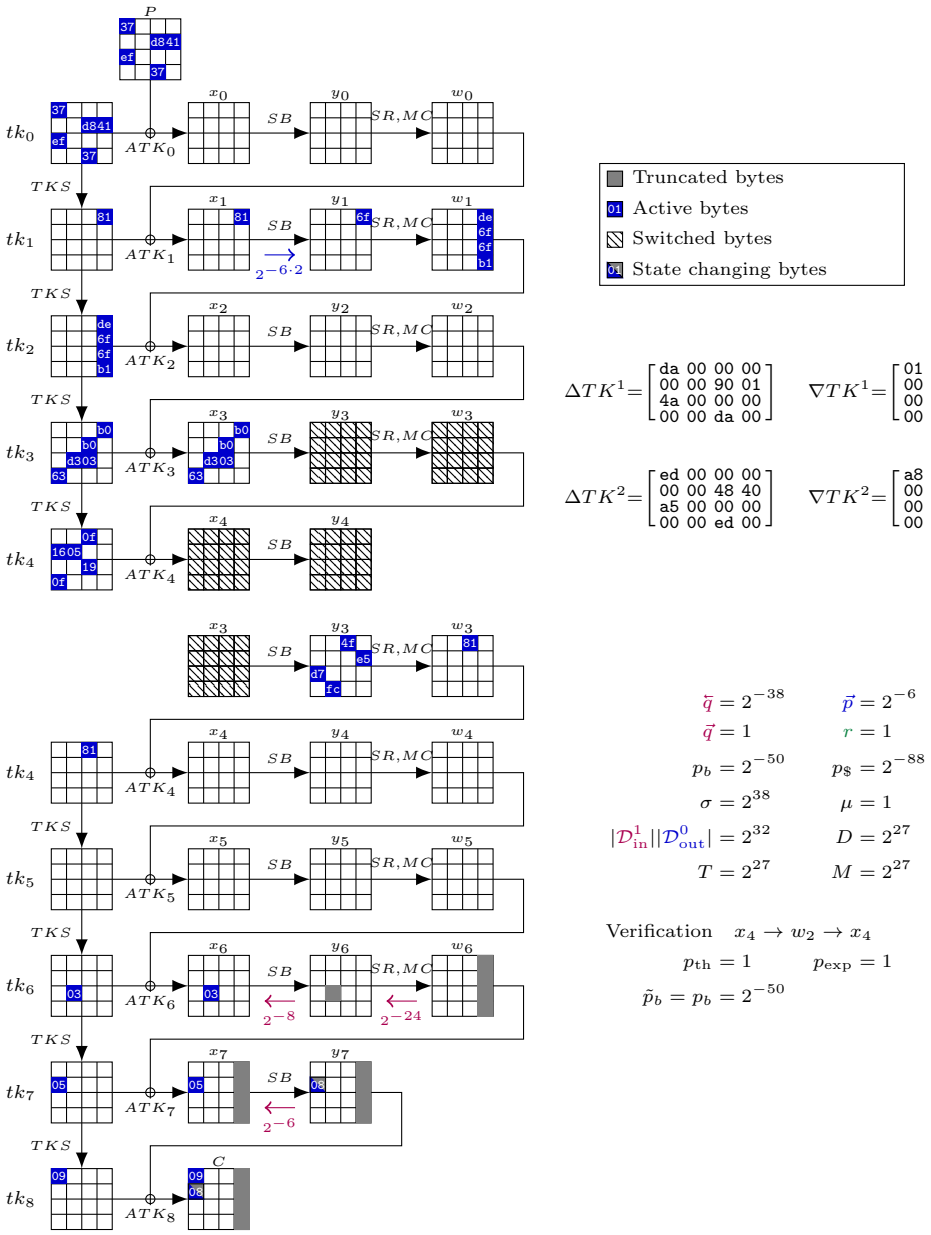
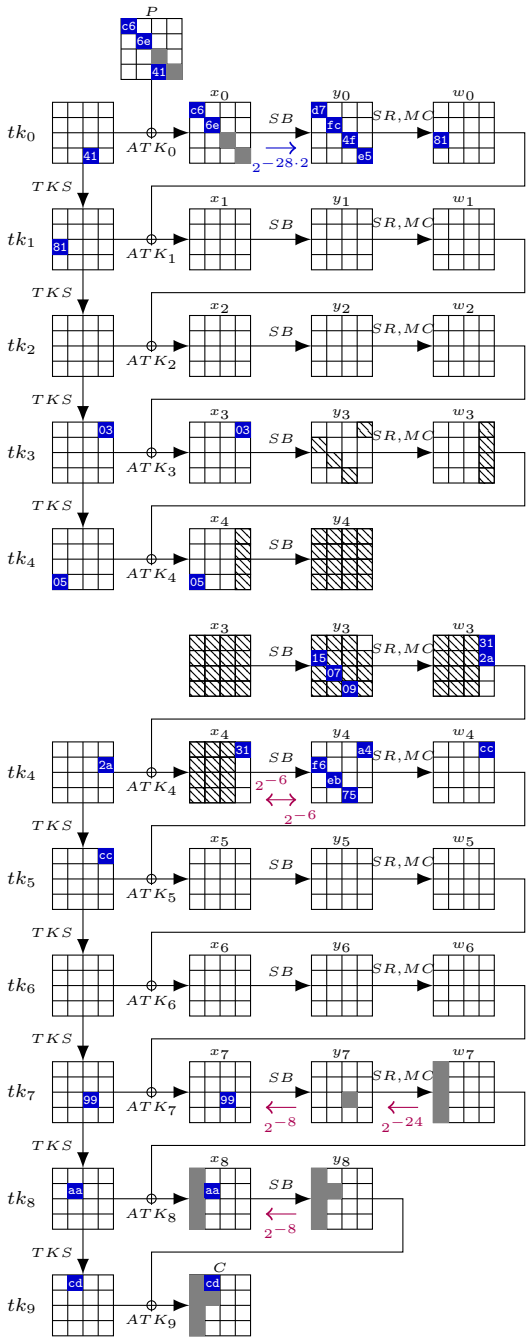


Figure 4.14: Truncated boomerang attack on 8-round Deoxys-BC in the RTK_2 model, starting from the ciphertext side. This attack succeeds with probability 1/2. Middle rounds are analyzed with the ladder switch and single BCT (probability r).



Truncated bytes
Active bytes
Switched bytes

$$\Delta TK^1 = \begin{bmatrix} 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 01 & 00 \end{bmatrix} \quad \nabla TK^1 = \begin{bmatrix} 00 & 00 & 00 & 00 \\ 00 & 88 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \end{bmatrix}$$

$$\Delta TK^2 = \begin{bmatrix} 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 40 & 00 \end{bmatrix} \quad \nabla TK^2 = \begin{bmatrix} 00 & 00 & 00 & 00 \\ 00 & 0a & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \end{bmatrix}$$

$$\tilde{q} = 2^{-46} \qquad \vec{p} = 2^{-28}$$
$$\vec{q} = 2^{-6} \qquad r = 1$$
$$p_b = 2^{-108} \qquad p_\$ = 2^{-88}$$
$$\kappa = 72 \qquad \ell = 2^{-52}$$
$$\tilde{\sigma} = 2^{104} \qquad \mu = 1$$
$$|\mathcal{D}^1_{\text{in}}||\mathcal{D}^0_{\text{out}}| = 2^{56} \qquad D = 2^{55}$$
$$T = 2^{55} \qquad M = 2^{55}$$

Verification $x_5 \to w_2 \to x_5$

$$p_{\text{th}} = 2^{-12} \qquad p_{\text{exp}} = 2^{-12.4}$$
$$\tilde{p}_b = 2^{-108.4}$$

Figure 4.15: Truncated boomerang attack on 9-round Deoxys-BC in the RTK_2 model, starting from the ciphertext side. This attack succeeds with probability 1/2. Middle rounds are analyzed with the ladder switch and single BCT (probability r).

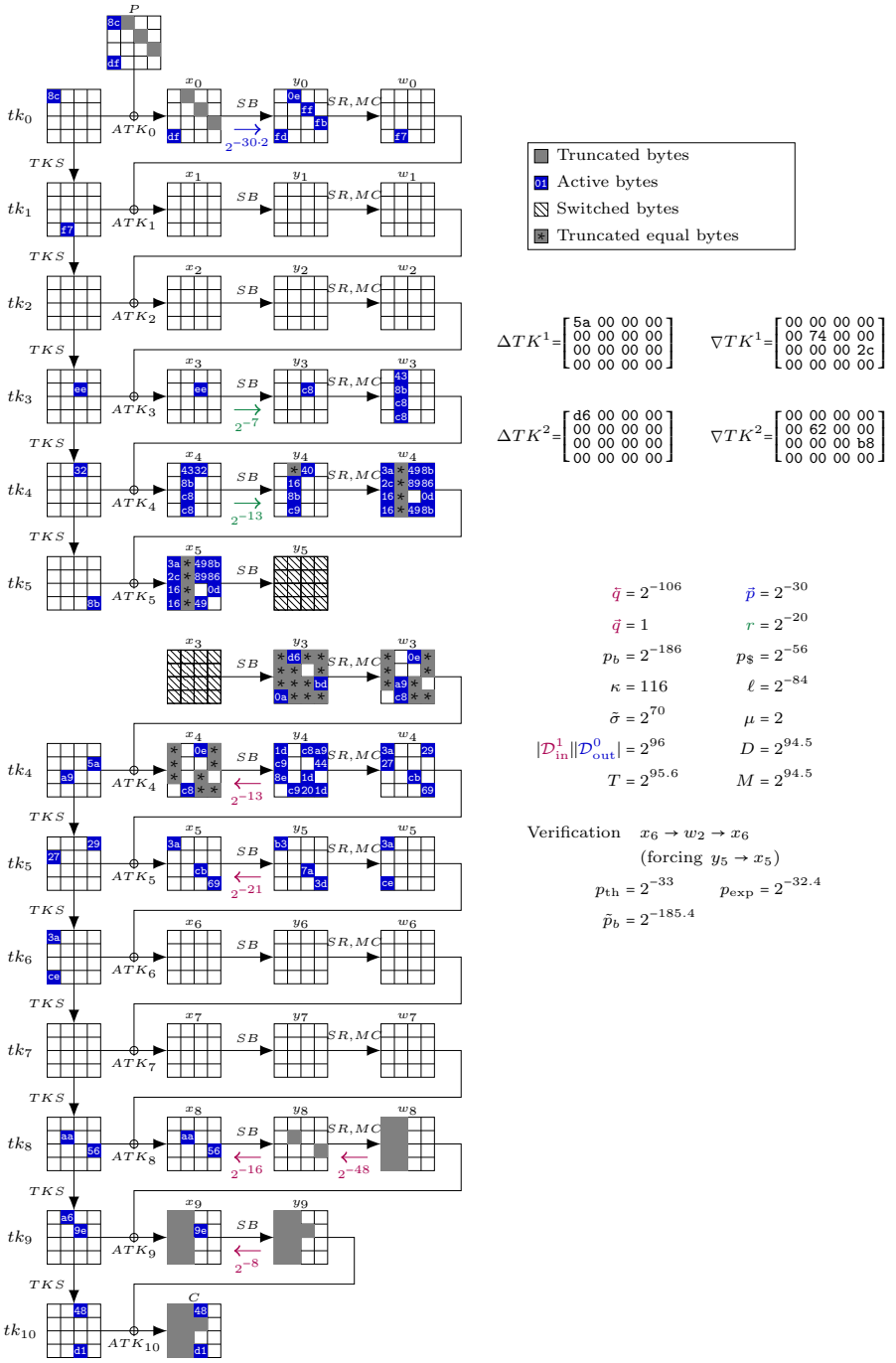


Figure 4.16: Truncated boomerang attack on 10-round Deoxys-BC in the RTK_2 model, starting from the ciphertext side. This attack succeeds with probability 1/2. Middle rounds are analyzed with UBCT, LBCT and EBCT (probabilities on the trail).

4.4.7.8 10-round Deoxys-BC in the RTK$_3$ model (Figure 4.17)

Query $2^{1.4}$ structures of 2^{16} ciphertexts. The only detected quartet is a right quartet. $(D, T, M) = (2^{19.4}, 2^{19.4}, 2^{18})$. This attack is equivalent to the attack given in [Sas18a], but the complexity was wrongly estimated as 2^{22}.

4.4.7.9 11-round Deoxys-BC in the RTK$_3$ model (Figure 4.18)

Query a partial structure of $2^{30.7}$ elements. The only detected quartet is a right quartet. $(D, T, M) = (2^{32.7}, 2^{32.7}, 2^{32.7})$.

4.4.7.10 12-round Deoxys-BC in the RTK$_3$ model (Figure 4.19)

Query $2^{2.4}$ structures of 2^{64} ciphertexts. On average, $\mu = 2^{2.4} \cdot 2^{64} \cdot 2^{64} \cdot \tilde{p_b} = 2$ quartets follow the trail and $2^{2.4} \cdot 2^{64} \cdot 2^{64} \cdot p_\$ = 2^{58.4}$ random quartets are detected. For each quartet, retrieve on average 2^{-32} key candidates for $tk_0[4]$ and $tk_{12}[0, 2, 3]$ in a few table accesses. Then, for each of the $2^{58.4} \cdot 2^{-32} = 2^{26.4}$ remaining quartets, deduce on average $2^{24} \cdot 2^{24} \cdot 2^{-32} = 2^{16}$ candidates for $tk_{12}[12, 13, 14, 15]$. For each candidate, compute the values of $x_{11}[12, 13, 14, 15]$ and the differences in state y_{10}. From the transition $x_{10} \rightarrow y_{10}$, retrieve 2^{-24} key candidates for $tk_{11}^{eq}[1, 11, 12]$. Thus, $2^{26.4} \cdot 2^{16} \cdot 2^{-24} = 2^{18.4}$ quartets remain with 1 average key candidate of 88 bits. For each remaining quartet, increase the counter of the corresponding key candidate. Only the right counter is expected to be greater than 2. This gives $(D, T, M) = (2^{67.4}, 2^{67.4}, 2^{65})$.

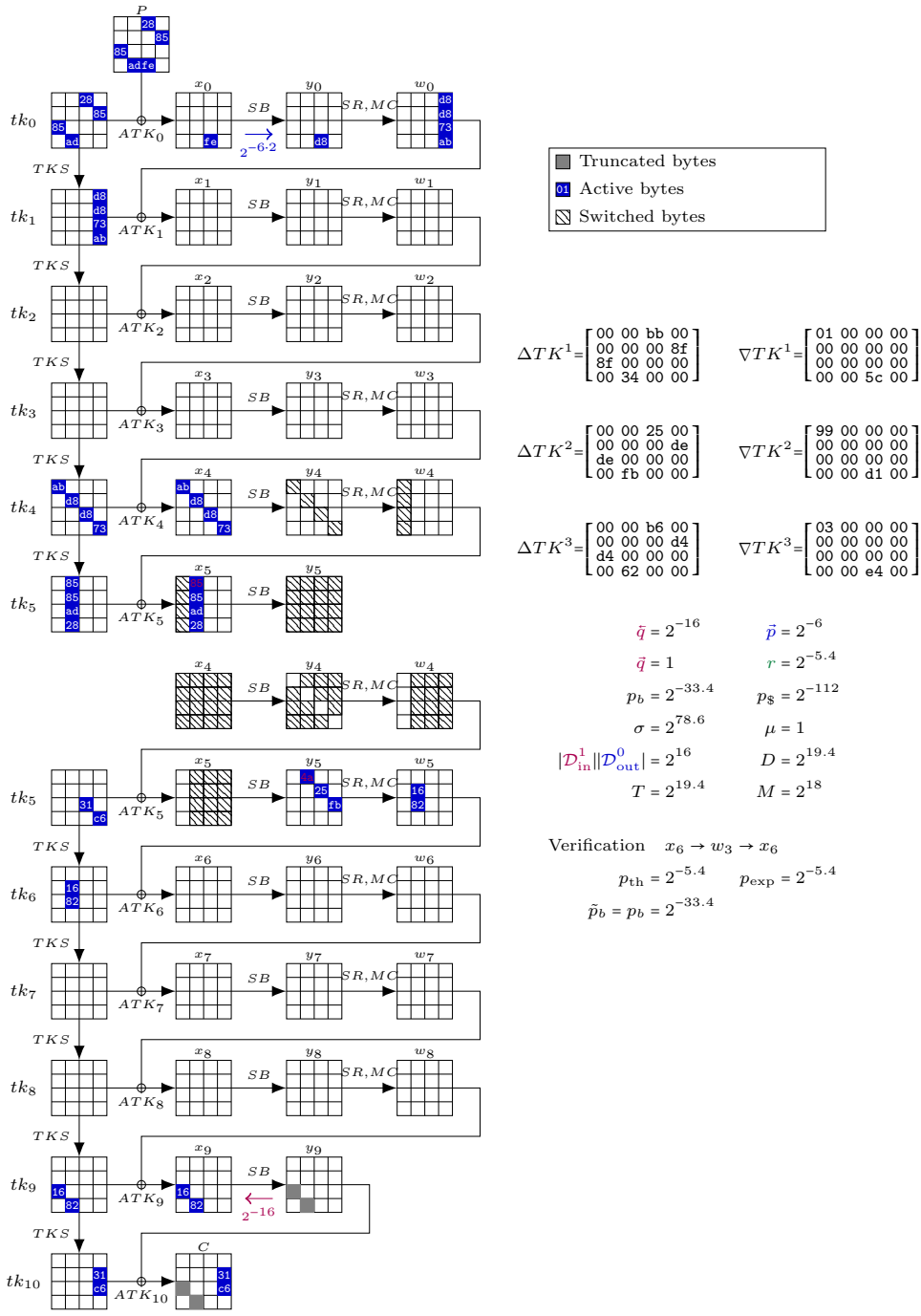


Figure 4.17: Truncated boomerang attack on 10-round Deoxys-BC in the RTK_3 model, starting from the ciphertext side. This attack succeeds with probability 1/2. Middle rounds are analyzed with the ladder switch and single BCT (probability r).

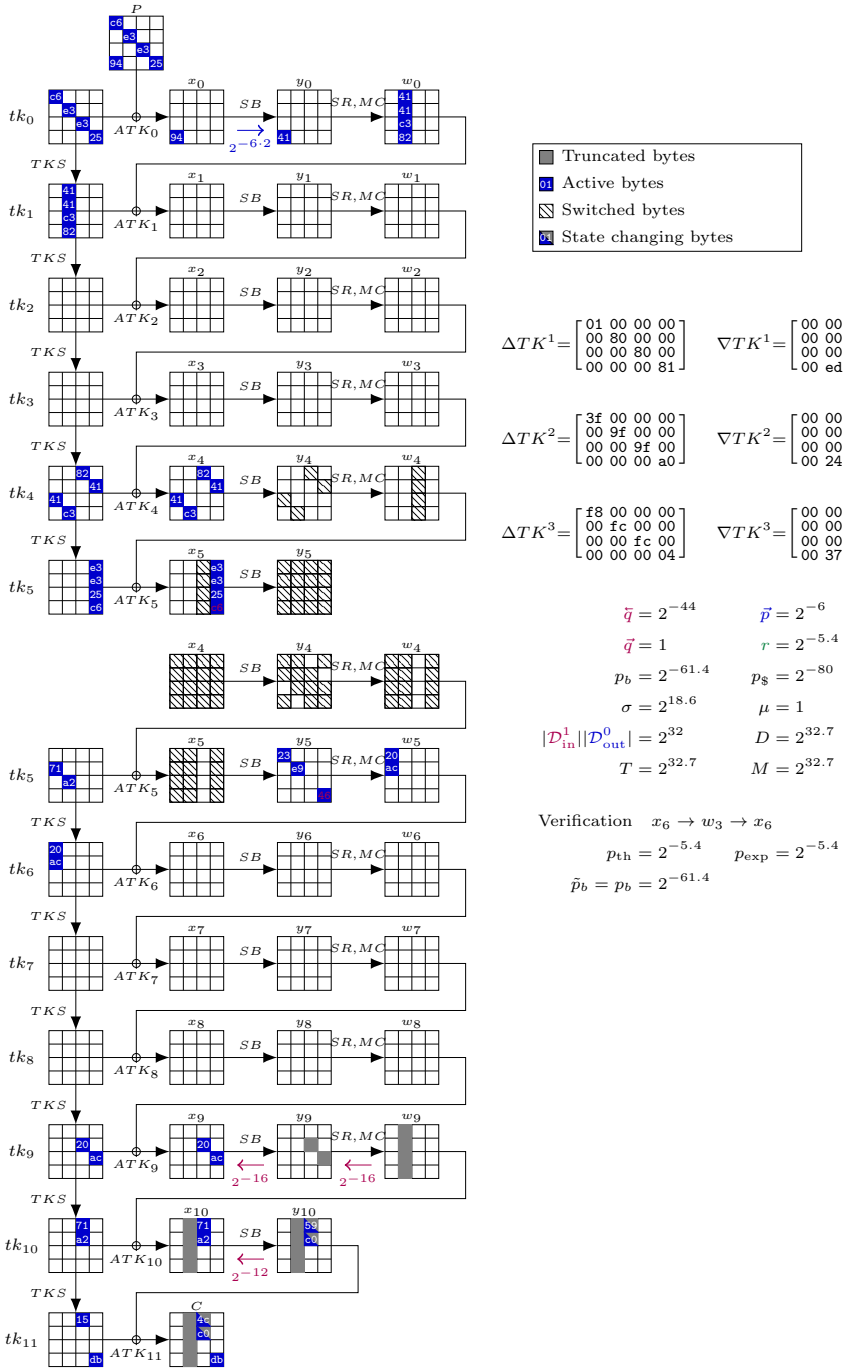


Figure 4.18: Truncated boomerang attack on 11-round Deoxys-BC in the RTK_3 model, starting from the ciphertext side. This attack succeeds with probability 1/2. Middle rounds are analyzed with the ladder switch and single BCT (probability r).

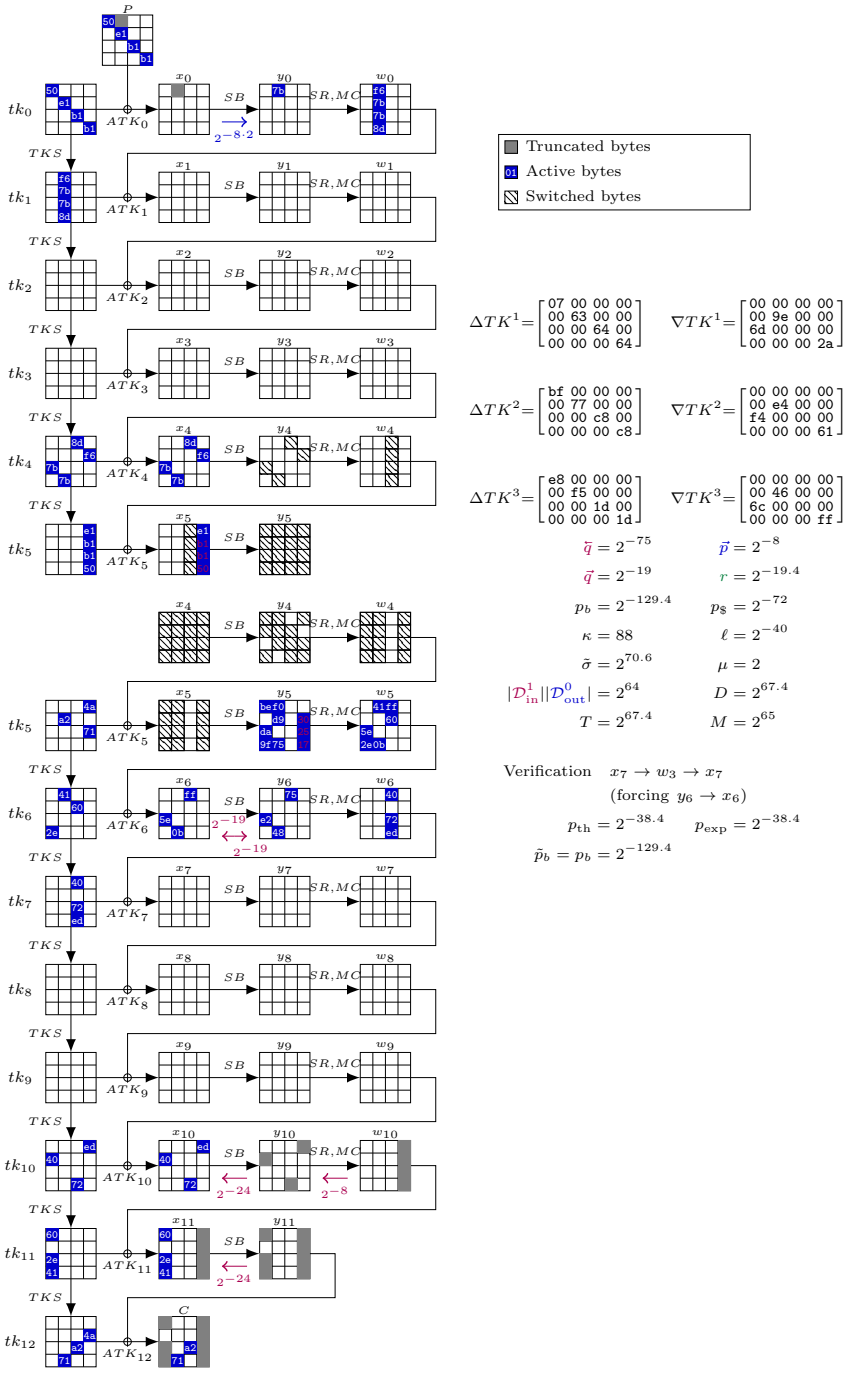


Figure 4.19: Truncated boomerang attack on 12-round Deoxys-BC in the RTK_3 model, starting from the ciphertext side. This attack succeeds with probability 1/2. Middle rounds are analyzed with the ladder switch and single BCT (probability r).

4.4.7.11 13-round Deoxys-BC in the RTK$_3$ model (Figure 4.20)

Query a partial structure of $2^{125.65}$ plaintexts. On average, $\mu = 2^{125.65}.2^{125.65}.\tilde{p}_b = 4$ quartets follow the trail.

1. For each element of the structure, retrieve the representant k of the 2^6 possible key values of $tk_{13}[13, 14, 15]$ that satisfy the transition $y_{12} \rightarrow x_{12}$. k defines 18 key bits.

2. Guess the value of the tweakey material $tk_0[2, 7, 8, 13]$. Set $\delta =$ `0x7e42c465` and $\delta_{\text{in}} =$ `0x00007a00`, and look for collisions between:

$$\begin{aligned} v &= y_0[2, 7, 8, 13] \quad \| \; \bar{y_0}[2, 7, 8, 13] \quad \| \; \bar{P}[0, 5, 10, 15] \quad \| \; k \\ v' &= y'_0[2, 7, 8, 13] + \delta \; \| \; \bar{y'_0}[2, 7, 8, 13] + \delta \; \| \; \bar{P'}[0, 5, 10, 15] + \delta_{\text{in}} \; \| \; k'. \end{aligned}$$

   This step costs $2^{32} \cdot 2 \cdot 2^{125.65} = 2^{158.65}$ in time complexity. On average, $2^{125.65} \cdot 2^{125.65} \cdot 2^{-114} = 2^{137.3}$ quartets remain for each $tk_0[2, 7, 8, 13]$ ($2^{169.3}$ in total).

3. For each quartet, retrieve $2^{7+7-32} = 2^{-18}$ values of $tk_0[3, 4, 9, 14]$ such that the difference in $w_0[4]$ is compatible with the S-box transition in the next round. In order to minimize the complexity, first deduce the $2^{7+7-8} = 2^6$ pairs of column differences compatible with a key candidate for $tk_0[3]$, by only checking the first S-box. Then, deduce the $2^{6-8} = 2^{-2}$ pairs of columns compatible with a key candidate for $tk_0[4]$ with the second S-box. Finally deduce $tk_0[9, 14]$.

   This step requires $2^8 + 2^7 = 2^{8.6}$ table accesses per quartet, therefore a total of $2^{8.6+169.3} = 2^{177.9}$ accesses; and $2^{32+137.3-18} = 2^{151.3}$ quartets remain.

4. For each quartet, retrieve $2^{7+7-32} = 2^{-18}$ values of $tk_0[1, 6, 11, 12]$ and $2^{24+24-32} = 2^{16}$ key candidates for $tk_{13}[8, 9, 10, 11]$. Recover $x_{12}[8, 9, 10, 11]$ and the difference in $y_{11}[2, 7, 8]$. Retrieve 2^{-24} candidates for $tk_{12}^{eq}[2, 7, 8]$. $2^{151.3-18+16-24} = 2^{125.3}$ quartets remain.

5. For each quartet, recover the difference in $x_1[4, 14]$ and the value of $w_0[4, 14]$ from the known key bytes of tk_0. Retrieve $2 \cdot 2 \cdot 2^{-8} = 2^{-6}$ values of $tk_1[4]$ and 2^{-6} values for $tk_1[14]$ (2 candidates are deduced per pair because the differences are already compatible). $2^{125.3} \cdot 2^{-12} = 2^{113.3}$ quartets remain.

6. Eventually, each of the $2^{113.3}$ quartets determines on average 1 candidate of $18 + 32 + 32 + 32 + 32 + 24 + 16 = 186$ bits. We model a wrong counter with a poisson distribution with $\lambda = 2^{-72.7}$. The probability that any wrong counter is at least 3 is $(1 - e^{-\lambda}(1 + \lambda + \lambda^2/2)) \cdot 2^{184} \approx 2^{-34.7}$. The correct counter follows the poisson distribution with $\lambda = 4$ and it is at least 3 with probability 0.76. Therefore, the success probability of this attack is 0.76.

Complexity analysis. The time complexity is dominated by the $2^{177.9}$ table accesses of step 3. An encryption of 13-round Deoxys-BC has 16×13 S-boxes, so the time complexity is equivalent to $2^{177.9}/208 = 2^{170.2}$ encryptions. Thus $(D, T, M) = (2^{126.7}, 2^{170.2}, 2^{126.7})$.

4.4.7.12 14-round Deoxys-BC in the RTK$_3$ model (Figure 4.21)

We did not manage to find a 14-round trail with the MILP solver, but the 13-round attack can be extended by adding a round at the end, and guessing most of the last subkey.

We start by querying the decryption oracle over the full codebook with tweaks $\bar{T}$ and $\bar{T}'$, and storing the results in memory. Then we guess 104 bits of the last round key: $k_{14}[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]$.

This allows us to essentially simulate the 13-round attack using queries from the 14-round oracle. Since 13 bytes of k_{14} are known, we can partially decrypt the corresponding S-boxes in the last round, and the MixColumns in the first three columns (after replacing tk_{13} by an equivalent key).

We build the same type of structure as used in the 13-round attack. For each plaintext P_i, we query $C_i = E(P_i, T)$ and we partially decrypt the final round. Then we generate the set of 2^{40} values with the required difference in y_{12}, and partially encrypt them to obtain the corresponding ciphertexts $\bar{C}_i^j$. Finally, we use the stored decryption values to obtain the corresponding $\bar{P}_i^j$.

We start with a full structure of 2^{128} plaintexts, so that we expect $\mu = 2^{128+128} \cdot \tilde{p}_b = 2^{2.7}$ good quartets. As in the 13-round attack (steps 1 and 2), we match elements on 114 bits: we expect $2^{128+128-114} = 2^{142}$ candidate quartets for each guess of $104 + 32 = 136$ bits of key, or 2^{278} quartets in total.

Following the 13-round attack (steps 3,4, and 6), we extract on average 2^{-48} candidates for 80 additional key bits . Finally, we use the constraints of the MixColumns operation of round 11: there are only 2^{16} possible differences in w_{11} from which we deduce $2^{16+16-24} = 2^8$ candidates for $tk_{13}^{eq}[2, 5, 8]$. We end up with 2^{238} suggestions for 258 bits of key.

We keep all key candidates suggested at least 6 times. Modeling the counters for wrong keys as following a Poisson distribution with $\lambda = 2^{-20}$, we expect wrong keys to be kept with probability $2^{-129.5}$. We expect $2^{2.7} = 6.5$ good quartets, so the right key candidate is at least 6 with probability more than 1/2. Finally we do an exhaustive search over the 126 key bits remaining, for a cost of $2^{126+128.5} = 2^{254.5}$.

The bottleneck of the attack is the extraction of key candidates for 2^{278} quartets. Following the analysis of the 13-round attack, we estimate that it requires about $2^{8.6}$ table accesses, equivalent to $2^{0.8}$ encryption. The full attack has complexity $(D, T, M) = (2^{129}, 2^{278.8}, 2^{129})$.

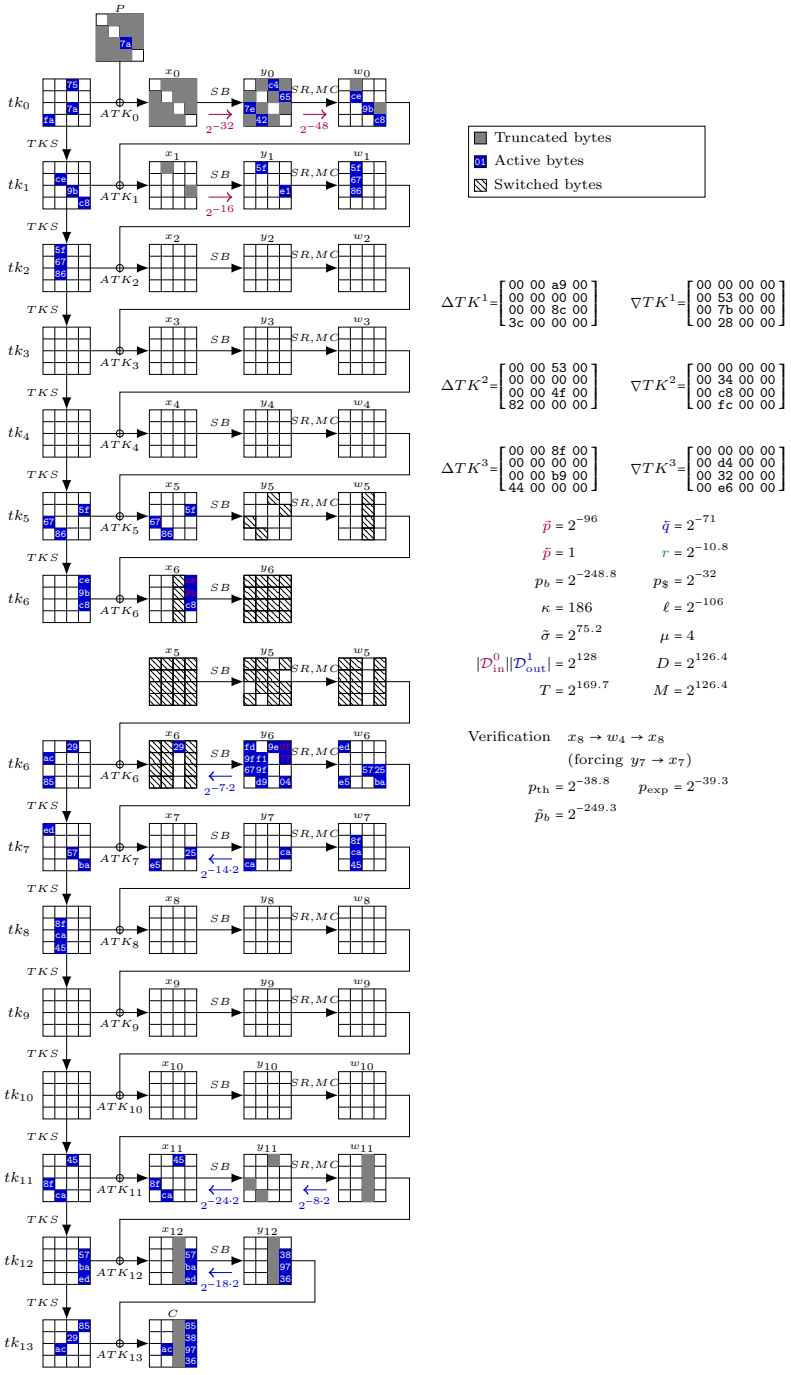


Figure 4.20: Truncated boomerang attacks on 13-round Deoxys-BC in the RTK_3 model, starting from the plaintext side. This attack succeeds with probability 0.76. Middle rounds are analyzed with the ladder switch and single BCT (probability r).

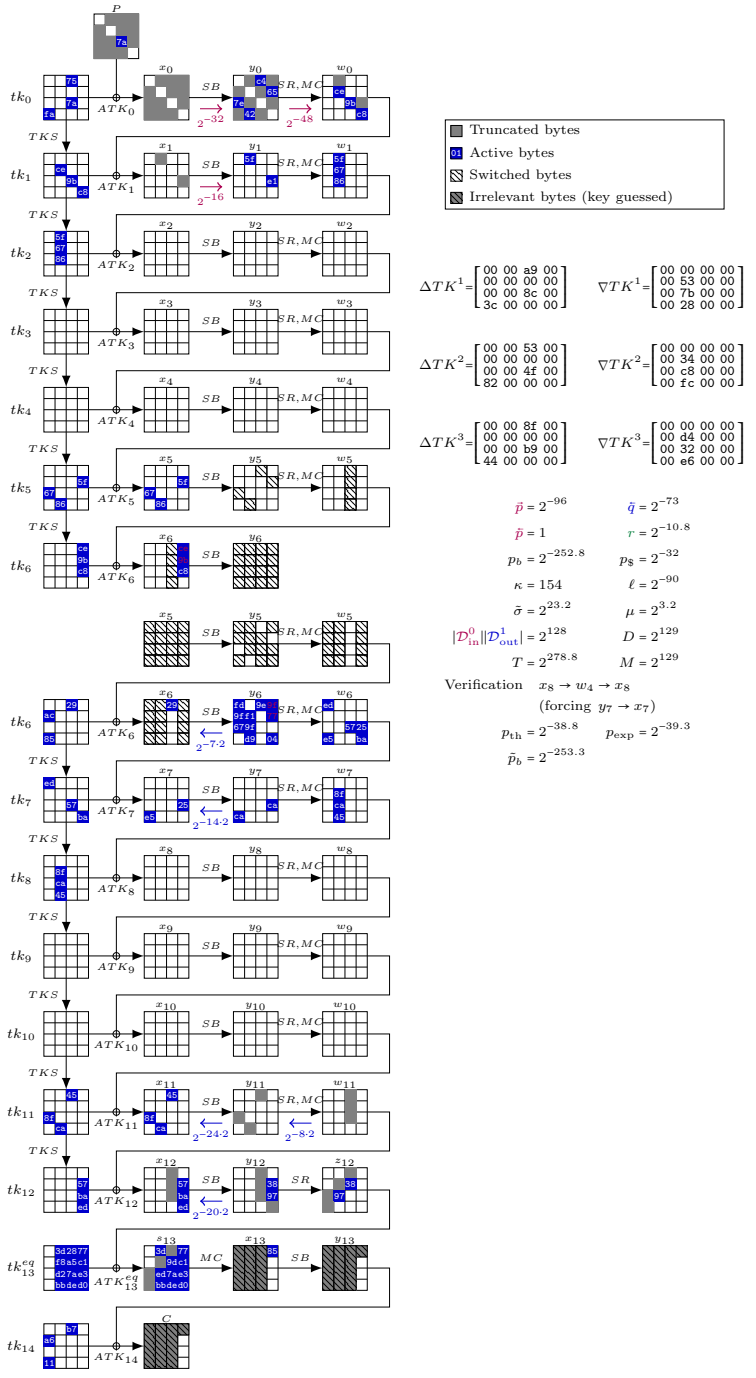


Figure 4.21: Truncated boomerang attack on 14-round Deoxys-BC in the RTK_3 model, starting from the plaintext side. Middle rounds are analyzed with the ladder switch and single BCT (probability r).

4.5 Improved Boomerang Attacks on AES

In this section, we present two improved boomerang attacks on 6-round AES, outperforming previous boomerang-based key-recovery attacks on 6-round AES. This is a joint work with Orr Dunkelman, Nathan Keller, Gaëtan Leurent and Victor Mollimard. The results are compared to the literature in Table 4.1 and recalled in Table 4.5.

Rounds	Type	Data	Time	Mem	Ref
6	Boomerang	$2^{50.9}$	$2^{67.7}$	2^{32}	Section 4.5.1
6	Boomerang	2^{51}	$2^{66.4}$	2^{42}	Section 4.5.1
6	Boomerang	$2^{57.1}$	$2^{60.9}$	2^{33}	Section 4.5.2

Table 4.5: Summary of our improved key-recovery attacks on 6-round AES.

This attack is based on an observation regarding the 6-round retracing boomerang attack (presented in Section 4.3.3): the attack requires much more decryptions (2^{55}) than encryptions (2^{20}). This implies that we may discard some ciphertext pairs (C, C') at the cost of additional encryptions without increasing the overall time or data complexity. With that idea in mind, we filter ciphertext pairs and only consider pairs (C, C') which collide on two anti-diagonals of ciphertext, rather than a single byte at the end of the fifth round. For each ciphertext collision, we can then build a structure of 2^{32} shifted ciphertexts $C + \nabla_i$ (∇_i takes all possible values in one of the inactive anti-diagonals), following the shifting retracing boomerang framework. Note that unlike in the truncated boomerang attack, we do not build a structure for each ciphertext, but only for ciphertexts colliding on two inactive anti-diagonals. The plaintext structures and ciphertext structures, each of size 2^{32}, are such that asking a collision on 64 bits of ciphertext does not bring any significant complexity overhead. Indeed, with 2^{32} plaintext queries, we expect roughly 1 pair colliding on 64 bits, leading to 2^{32} decryption queries, and 2^{32} potential quartets. On the other hand, given a collision on two anti-diagonals of ciphertext, the pair (P, P') is more likely to be a *right forward pair*.

Compared to the retracing boomerang attack, we additionally use truncated trails in the backward trail of $(C, \bar{C})$ and $(C', \bar{C'})$ without fixing the value of ∇_i, which improves the probability of the boomerang characteristic.

In the following attacks, the time complexity bottleneck comes from the key-recovery step. We therefore present slightly different boomerang characteristics and key-recovery strategies, leading to different trade-offs between time, data and memory complexities. Our results are summarized in Table 4.5.

Notation. Like in the retracing or truncated boomerang attacks, we use multiple differential trails. However, parts of the returning trail are fixed by the forward trail. To indicate byte (resp. column/diagonal/anti-diagonal) indexes in some states of the trail, we use letters $a \in [\![0, 15]\!]$ and $\ell, m \in [\![0, 3]\!]$. Bytes are indexed with the

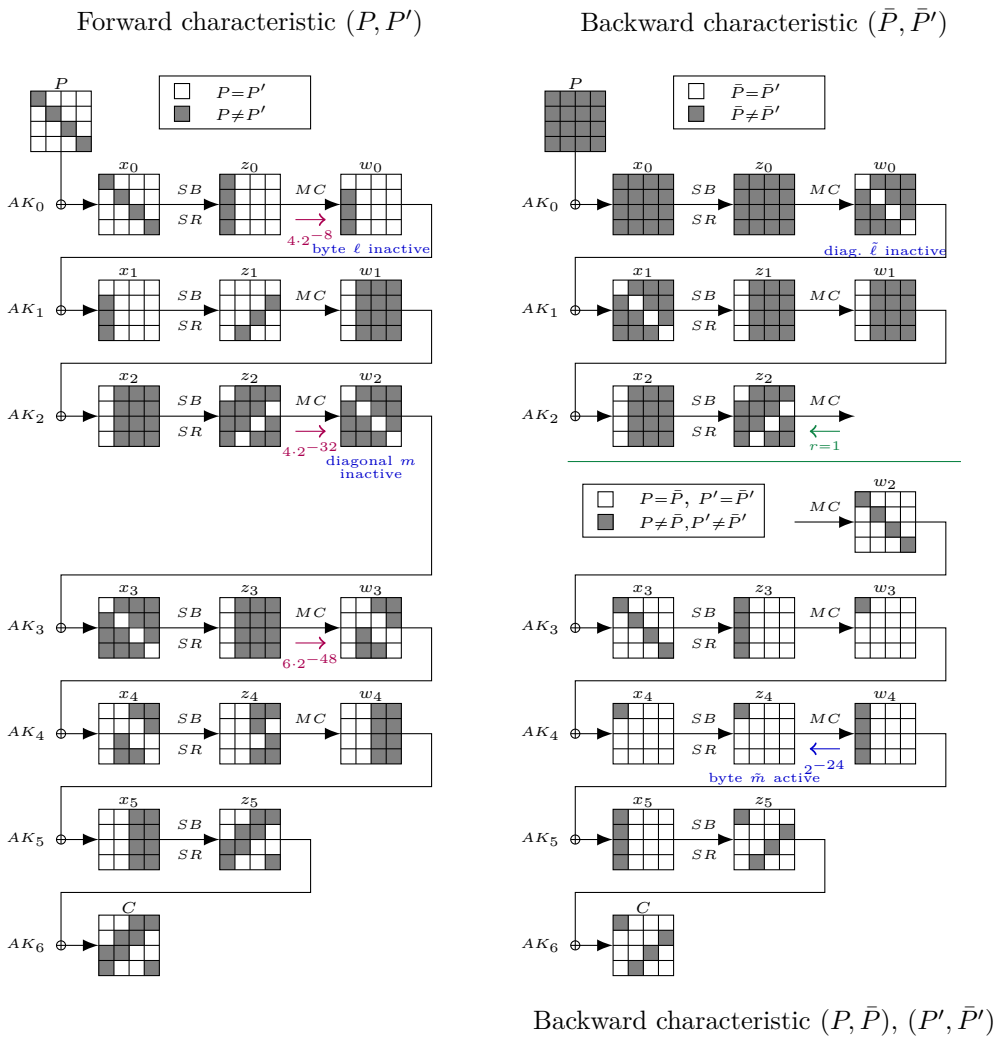


Figure 4.22: Boomerang characteristic on 6-round AES with low data complexity.

conventional order presented in Section 2.3.1. Diagonals and anti-diagonals are indexed with the index of their active column in the first row. We denote $\tilde{a} = -a \mod 4, \tilde{\ell} = -\ell \mod 4, \tilde{m} = -m \mod 4$.

4.5.1 A Key-Recovery Attack With Low Data Complexity

The first key-recovery attack that we propose is based on the boomerang depicted in Figure 4.22, and described in Algorithm 4.4.

Boomerang analysis. Starting from a pair (P, P') with an active diagonal, we consider that it is a *right forward pair* if one byte in position $l \in \{0, 1, 2, 3\}$ is inactive at the end of the first round (probability $4 \cdot 2^{-8}$), one diagonal in position

$m \in \{0, 1, 2, 3\}$ is inactive at the end of the third round (probability $4 \cdot 2^{-32}$), and two diagonals are inactive at the end of the fourth round (probability $6 \cdot 2^{-48}$). Therefore a forward pair (P, P') is right with probability $\vec{p} = 2^{-81.4}$. This implies a collision in two anti-diagonal of the ciphertext. The probability that a random ciphertext pair collides on two anti-diagonals is $6 \cdot 2^{-64} = 2^{-61.4}$.

In the backward direction, $\bar{C}$ and $\bar{C}'$ are obtained by shifting C and C' with a difference in one of the inactive anti-diagonals. This corresponds to a shifting retracing boomerang with a single active Super-box. Therefore, the active S-box transitions from z_5 to x_5 and from z_4 to x_4 are the same for the pairs $(C, \bar{C})$ and $(C', \bar{C}')$. We assume that the difference collapses to a single byte in position $\tilde{m}$ of the state z_4 (after one round), corresponding to the inactive diagonal m in $w_2 + w_2'$; this happens with probability $\overleftarrow{q} = 2^{-24}$. In practice, the attacker will guess the value of key bytes $k_6[0, 7, 10, 13]$ in order to directly construct values $\bar{C}$ and $\bar{C}'$ with this property.

As in the retracing boomerang attack, we have $(z_3 + z_3')[0, 1, 2, 3] = 0$ and $(\bar{z}_3 + \bar{z}_3')[0, 1, 2, 3] = 0$, which in turn implies $(x_3 + x_3')[0, 5, 10, 15] = 0$ and $(\bar{x}_3 + \bar{x}_3')[0, 5, 10, 15] = 0$. Therefore, we obtain $x_3 + \bar{x}_3 = x_3' + \bar{x}_3'$ and $z_2 + \bar{z}_2 = z_2' + \bar{z}_2'$ with probability $r = 1$. When considering the pair $(\bar{C}, \bar{C}')$, we deduce that the difference in z_2 is the same as in the forward pair: $\bar{z}_2 + \bar{z}_2' = z_2 + z_2'$. In particular, anti-diagonal $\tilde{\ell}$ is inactive in z_2; this implies that diagonal $\tilde{\ell}$ of w_0 is inactive with probability $\overleftarrow{p} = 1$.

We then proceed as in the retracing boomerang attack. We obtain many quartets following the characteristic simultaneously: assuming that (P, P') is a right forward pair, and that the pairs $(\bar{C}, \bar{C}')$ are constructed such that $z_4 + \bar{z}_4$ is active only in byte $\tilde{m}$, then with probability 1, diagonal $\tilde{\ell}$ of w_0 is inactive for each pair $(\bar{C}, \bar{C}')$.

Comparison with the retracing boomerang attack. The probability that a pair (P, P') is a right pair, and that a shifted pair $(\bar{C}, \bar{C}')$ follows the returning trail is $p_b = \vec{p} \cdot \overleftarrow{q} \cdot \overleftarrow{p} \cdot r = 2^{-81.4-24} = 2^{-105.4}$. This is much lower than the 6-round AES retracing boomerang attack, that has a boomerang probability of $2^{-38} \times 2^{-24} = 2^{-62}$ if we intepret the lower characteristic as a truncated trail instead of a key guess. However, the probability for a wrong forward pair to have the right ciphertext difference is $p_w = 2^{-61.4}$ (compared to $p_w = 2^{-8}$ for the retracing boomerang, corresponding to the event $z_4[0] + \bar{z}_4[0] = 0$). In order to collect a single right quartet, there are $2^{62-8} = 2^{54}$ wrong quartets in the retracing boomerang, but only $2^{105.4-61.4} = 2^{44}$ wrong quartets remain in our attack. This gain in signal-to-noise ratio p_b/p_w allows to significantly reduce the time complexity of the key-recovery, since the key-recovery procedure needs to be applied to fewer quartets.

Attack description. The attack consists of the following steps:

1. Ask for the encryption of structures of 2^{32} plaintexts with different values on the main diagonal.

2. Among each structure, look for pairs of ciphertexts (C, C') colliding on any two anti-diagonals (for simplicity, we consider them as the first two). Let us denote $\bar{C}_i = C + \nabla_i$ and $\bar{C}'_i = C' + \nabla_i$ for $1 \leqslant i \leqslant 2^{32}$, where ∇_i takes the value i on the first anti-diagonal and the value 0 on other anti-diagonals. Ask for the decryptions $\bar{P}_i = E^{-1}(\bar{C}_i)$ and $\bar{P}'_i = E^{-1}(\bar{C}'_i)$ for all $1 \leqslant i \leqslant 16 \times 2^{24} = 2^{28}$.

3. For each candidate K for $k_6[0, 7, 10, 13]$, and each $\tilde{m} \in \{0, 1, 2, 3\}$, fetch 8 pairs $(P, \bar{P}_i)$ (already queried) such that under the key K, the first column of z_4 of $(P, \bar{P}_i)$ is active only in position $\tilde{m}$, and consider the 8 corresponding quartets $(P, P', \bar{P}_i, \bar{P}'_i)$.

   For each $\tilde{\ell} \in [\![0, 3]\!]$, assume that diagonal $\tilde{\ell}$ of w_0 is inactive for all quartets simultaneously and deduce key k_0. If a candidate k_0 is compatible with all quartets, return it as the correct key.

Algorithm 4.4: Low data complexity boomerang attack on 6-round AES.

loop ▷ The expected number of iteration is $2^{18.4}$
  Query the encryption of a structure of 2^{32} plaintexts
  for all pairs (C, C') colliding on two anti-diagonals **do**
    for $0 \leqslant i < 2^{28}$ **do**
      Define $\bar{C}_i = C + \nabla_i$, $\bar{C}'_i = C' + \nabla_i$,
      Query $\bar{P}_i = E^{-1}(\bar{C}_i), \bar{P}'_i = E^{-1}(\bar{C}'_i)$
    for all $k_6[0, 7, 10, 13]$ **do**
      for $0 \leqslant m < 4$ **do**
        Gather 8 quartets $(P, P', \bar{P}_i, \bar{P}'_i)$ s.t. $z_4 + \bar{z}_4$ is active only on position $\tilde{m}$
        for $0 \leqslant \tilde{\ell} < 4$ **do**
          Assume diagonal $\tilde{\ell}$ of w_0 is inactive for (P, P') and all $(\bar{P}_i, \bar{P}'_i)$
          Deduce key candidates for k_0

Complexity. The attack queries new structures of plaintexts until a structure includes a right forward pair. Since the probability of the forward characteristic is $2^{-81.4}$, we expect $2^{32+31-81.4} = 2^{-18.4}$ right pairs per structure, and we expect to iterate over $2^{18.4}$ structures on average before finding a right pair. This corresponds to $2^{18.4+32+31-61.4} = 2^{20}$ candidates (C, C') at step 2. When a right pair is present, it passes the filtering of step 2, and the attack succeeds for the correct guess of $k_6[0, 7, 10, 13]$, $\tilde{\ell}$ and $\tilde{m}$.

We now give an analysis of Step 3. We first show that we can fetch the pairs $(P, \bar{P}_i)$ efficiently. Then, we explain in details the most technical part: the recovery of k_0 candidates from friend quartets, which we slightly improve compared to the meet-in-the-middle technique from [DKR+20].

Fetching the quartets. In Step 3, given a candidate K for $k_6[0,7,10,13]$ and a position $\tilde{m} \in [\![0,3]\!]$, we need to fetch 8 pairs $(\bar{P}_i, \bar{P}'_i)$ (leading to 8 quartets $(P, P', \bar{P}_i, \bar{P}'_i)$) such that the state z_4 of pair $(P, \bar{P}_i)$ is active only in position $\tilde{m}$ of the active column under the key K. This can be done efficiently: start from $C = E(P)$, compute the first column of z_4 from the anti-diagonal of C (ignoring the key addition AK_5), add the 256 possible byte values at position $\tilde{m}$ of the first column of z_4, and compute the corresponding values in the ciphertext side. This gives 256 different values for the first anti-diagonal in the ciphertexts. On average, this corresponds to $256 \times \frac{2^{28}}{2^{32}} = 16$ ciphertexts $\bar{C}_i$ of which the decryption was queried during Step 2. The probability that at least 8 such ciphertexts were queried during Step 2 can be approximated by $\Pr(\text{Poisson}(16) \geqslant 8) \geqslant 0.99$. The ciphertext $\bar{C}'_i$ is defined as $\bar{C}'_i = C + C' + \bar{C}_i$. With $256 \times 4 \times 2 = 2^{11}$ S-box calls, we fetch the 8 pairs; This is negligible compared to the complexity of getting the candidates for $k_0[0,5,10,15]$ as explained below.

Meet-In-The-Middle procedure to recover key candidates. We start by examining two candidate quartets, and extracting on average 2^8 candidates $k_0[0,5,10,15]$ for each $\tilde{\ell}$. Indeed, given two quartets $(P, P', \bar{P}_i, \bar{P}'_i)$ and $(P, P', \bar{P}_j, \bar{P}'_j)$, we have three pairs to use for filtering key candidates: (P, P'), $(\bar{P}_i, \bar{P}'_i)$, $(\bar{P}_j, \bar{P}'_j)$, and each pair provides an 8-bit filtering.

Following [DKR+20], we use a meet-in-the-middle procedure, considering independently 2^{16} values of $k_0[0,5]$ and 2^{16} values of $k_0[10,15]$.

1. Create 8 tables $T_0, T_1, T_2, T_3, T'_0, T'_1, T'_2, T'_3$.
2. For all 2^{16} values of $k_0[0,5]$, compute state $z_0[0,1]$ of $P + P'$ (denoted $a[0,1]$), $\bar{P}_i + \bar{P}'_i$ (denoted $b[0,1]$), and $\bar{P}_j + \bar{P}'_j$ (denoted $c[0,1]$). For each ℓ, store the 24-bit value
$$\mathsf{MC}(a[0], a[1], 0, 0)[\ell] \parallel \mathsf{MC}(b[0], b[1], 0, 0)[\ell] \parallel \mathsf{MC}(c[0], c[1], 0, 0)[\ell]$$
in table T_ℓ.
3. For all 2^{16} values of $k_0[10,15]$, compute state $z_0[2,3]$ of $P + P'$ (denoted $a[2,3]$), $\bar{P}_i + \bar{P}'_i$ (denoted $b[2,3]$), and $\bar{P}_j + \bar{P}'_j$ (denoted $c[2,3]$). For each ℓ, store the 24-bit value
$$\mathsf{MC}(0, 0, a[2], a[3])[\ell] \parallel \mathsf{MC}(0, 0, b[2], b[3])[\ell] \parallel \mathsf{MC}(0, 0, c[2], c[3])[\ell]$$
in table T'_ℓ.
4. Look for collisions between T_ℓ and T'_ℓ, for $\ell \in \{0,1,2,3\}$ (or equivalently for $\tilde{\ell} \in \{0,1,2,3\}$).

Steps 2 and 3 each require roughly the computation of $3 \times 4 \times 2^{16}$ AES S-boxes. However, since the same pair (P, P') is used for multiple quartets (with different key guesses for k_6), we can precompute the values $\mathsf{MC}(a[0], a[1], 0, 0)$ for all keys

$k_0[0,5]$, and $\mathsf{MC}(0,0,a[2],a[3])$ for all keys $k_0[10,15]$, reducing the complexity to $2 \times 4 \times 2^{16} = 2^{19}$ AES S-boxes. Step 4. requires 2^{18} sequential lookups to match the lists, equivalent to 2^{18} AES S-boxes. In total we obtain a complexity equivalent to $2^{20} + 2^{18} = 2^{20.3}$ AES S-boxes, or equivalently $2^{13.7}$ 6-round AES encryptions, to recover 2^{10} candidates for $k_0[0,5,10,15]$ and $\tilde{\ell}$.

Then, we can further filter the candidates using the remaining quartets. Indeed, each quartet provides an 8-bit filtering, and testing a key candidate requires only the evaluation of 2 AES columns (this adds negligible terms to the complexity). Given a wrong candidate pair (C, C'), the probability that there exists a choice of $\tilde{m}, \tilde{\ell}, k_6[0,7,10,13], k_0[0,5,10,15]$ compatible with 8 quartets is $2^{2+2+32+32-9\times 8} = 2^{-4}$. For the remaining candidates we apply the same procedure, to recover the other diagonals of k_0; this eliminates wrong candidates, and returns the full key k_0 for the right forward pair.

Finally, we obtain an attack with time complexity equivalent to $2^{20} \times 2^{32} \times 4 \times 2^{20.3} = 2^{74.3}$ AES S-boxes, or $2^{67.7}$ 6-round AES encryptions.

The attack requires on average $2^{32+18.4} = 2^{50.4}$ encryptions and $2 \times 2^{20+28} = 2^{49}$ decryptions, for a total of $2^{50.9}$ queries.

The memory complexity is bounded by the storage of each structure, i.e. 2^{32} 128-bit states. Therefore we obtain:

$$(D, T, M) = (2^{50.9}, 2^{67.7}, 2^{32}).$$

A time-memory trade-off. The time complexity of this attack can be slightly decreased by remarking that in the attack, we apply the MiTM procedure multiple times to each quartet $(P, P', \bar{P}_i, \bar{P}'_i)$. Indeed, given a pair (P, P') with two colliding anti-diagonals in output, we perform 4×2^{32} MiTM procedures on 2^{28} existing quartets (for each activity position $\tilde{m}$ in z_4, and each candidate for $k_6[0,7,10,13]$). Instead, it is possible to precompute 2^{18} candidates for $k_0[0,5,10,15]$ induced by each quartet with one MiTM procedure (with a variant using a single quartet) for each the 2^{28} quartets.

Then Step 3 is performed by recovering the list of 2^{18} candidates for $k_0[0,5,10,15]$ corresponding to two different quartets, and intersecting them. Therefore the time complexity becomes 2 table lookups recovering 2^{18} candidates each, and a matching step. It is hard to compare the cost of memory accesses to AES rounds, but since we make a large number of sequential accesses, we approximate it as equivalent to an S-box computation, resulting in a complexity of $2 \times 2^{20+2+32+18}$ S-box evaluations, or $2^{66.4}$ AES evaluations.

The memory complexity is however increased. Naively, we store 2^{28} sets of 2^{18} candidates, but it is possible to instead store only the sets of candidates for 2^{26} quartets: on average, 4 quartets among the 2^{26} chosen quartets correspond to each candidate $k_6[0,7,10,13]$ and activity position $\tilde{m}$ in z_4, and the MiTM procedure only needs to be applied to 2 of them (the 6 other quartets are used afterwards to filter the remaining candidates). Given $k_6[0,7,10,13]$ and $\tilde{m}$, the probability that at least 2 corresponding quartets are in the set of chosen quartets

can be approximated to $\Pr(\text{Poisson}(4) \geqslant 2) = 0.91$; if this is not the case, with probability 0.09, we discard the candidate $(k_6[0, 7, 10, 13], \tilde{m})$. This increases the expected number of plaintext structures to consider by a factor 1/0.91, but does not affect the time complexity because we do not process candidates with fewer than 2 precomputed quartets. This gives a memory complexity of $2^{26+18} = 2^{44}$ 32-bit states, equivalent to 2^{42} 128-bit states. In total, this gives:

$$(D, T, M) = (2^{51}, 2^{66.4}, 2^{42}).$$

4.5.2 A Key-Recovery Attack With Low Time Complexity

This attack reuses the boomerang of Section 4.5.1 with a slight change: the input pair (P, P') and returning pair $(\bar{P}, \bar{P}')$ are required to be inactive in a byte, in position $a \in \{0, 5, 10, 15\}$, i.e. in the main diagonal. This reduces the size of plaintext structures compared to the previous attack, but makes the key-recovery procedure more time-efficient. Indeed, for pairs following this characteristic, the first-round transitions depend on three key bytes (e.g. $k_0[5, 10, 15]$ if $a = 0$) rather than four. The attack is depicted in Figure 4.23 and described in Algorithm 4.5.

Boomerang analysis. Similarly to the previous attack, the probability that a pair is a *right forward pair* is $\vec{p} = 2^{-81.4}$, while the random probability that two ciphertexts collide in two anti-diagonals is $2^{-61.4}$.

In the backward direction, we assume that the difference collapses to a single byte in anti-diagonal $\tilde{m}$ of state z_4 (after one round); this happens with probability $\overleftarrow{q} = 2^{-24}$. This implies that diagonal $\tilde{\ell}$ of w_0 is inactive for the pair $(\bar{C}, \bar{C}')$ with probability $r = 1$, with the same analysis as in the previous attacks. Moreover, we require $\bar{P} + \bar{P}'$ to be inactive in byte a; this happens with probability $\overleftarrow{p} = 2^{-8}$.

As in the previous attacks, we consider multiple quartets, but the additional quartets are not required to collide in byte a of $\bar{P} + \bar{P}'$. Therefore, we use the same property as previously: assuming that (P, P') is a right forward pair, and that $(\bar{C}, \bar{C}')$ are constructed such that $z_4 + \bar{z}_4$ is active only in anti-diagonal m, then with probability 1, diagonal ℓ of w_0 is inactive for the pair $(\bar{C}, \bar{C}')$.

Comparison with the first attack. The boomerang probability is $p_b = \vec{p} \cdot \overleftarrow{q} \cdot \overleftarrow{p} \cdot r = 2^{-81.4-24-8} = 2^{-113.4}$ compared to $2^{-105.4}$ in the first attack, and the probability p_w that a wrong quartet is considered decreases by a factor 2^8 (corresponding to the condition $\bar{P}[0] = \bar{P}'[0]$). Therefore, this increases the data complexity but not the key-recovery time complexity, since the signal-to-noise ratio p_b/p_w, corresponding to the number of quartets on which the key-recovery is performed, is unchanged. Additionally, the structure size is reduced by the same factor 2^8 ; more plaintext pairs are needed to find a right forward pair. This also increases the data complexity, but not the key-recovery time complexity, since this does not affect the signal-to-noise ratio.

Attack description.

1. Ask for the encryption of structures of 2^{32} plaintexts with different values on the main diagonal.

2. Among each structure, consider pairs (P, P') with an inactive byte in the main diagonal of input; denote the inactive byte position as a. Look for corresponding pairs of ciphertexts (C, C') colliding on two anti-diagonals (for simplicity we consider them as the first two). Let us denote $\bar{C}_i = C + \nabla_i$ and $\bar{C}'_i = C' + \nabla_i$ for $1 \leqslant i \leqslant 2^{32}$, where ∇_i takes the value i on the first anti-diagonal and the value 0 on other anti-diagonals. Query and store the decryptions $\bar{P}_i = E^{-1}(\bar{C}_i)$ and $\bar{P}'_i = E^{-1}(\bar{C}'_i)$ for all $1 \leqslant i \leqslant 2^{32}$.

   For each $\ell \in \{0, 1, 2, 3\}$, assume that byte ℓ of w_0 is inactive for the pair (P, P'). Compute the set S_ℓ of 2^{16} candidates for $k_0[5, 10, 15]$ suggested by (P, P') under this assumption.

3. Consider plaintext pairs $(\bar{P}_i, \bar{P}'_i)$ that collide on byte a (we consider $a = 0$ in the rest of our analysis).

   For each $\ell \in \{0, 1, 2, 3\}$, consider the 2^{16} candidates for $k_0[5, 10, 15]$ in S_ℓ, and verify whether byte ℓ of w_0 is inactive for the pair $(\bar{P}_i, \bar{P}'_i)$. On average, 2^8 candidates should remain for each ℓ.

   Assume that state z_4 of $(P, \bar{P}_i)$ is active on byte $\tilde{m} \in \{0, 1, 2, 3\}$ of the first column and deduce 2^{10} candidates for $k_6[0, 7, 10, 13] \parallel \tilde{m}$. In total, a quartet suggests 2^{10} candidates for $k_0[5, 10, 15] \parallel \ell$ and 2^{10} candidates for $k_6[0, 7, 10, 13] \parallel \tilde{m}$ (2^{20} candidates in total).

4. For each quartet and each of its 2^{10} candidates for $k_6[0, 7, 10, 13] \parallel \tilde{m}$, consider 7 other pairs $(\bar{C}_j, \bar{C}'_j)$ such that the first column of z_4 of $(C, \bar{C}_j)$ is active only in position $\tilde{m}$; fetch their corresponding plaintexts $(\bar{P}_j, \bar{P}'_j)$ (stored during Step 2.).

   For each of the 2^{10} candidates $k_0[5, 10, 15] \parallel \ell$, there is on average a single value for $k_0[0]$ that is compatible with the first pair $(\bar{P}_j, \bar{P}'_j)$. Finally, verify the 2^{10} candidates $k_0[0, 5, 10, 15] \parallel \ell$ with the remaining pairs $(\bar{P}_j, \bar{P}'_j)$. If a candidate k_0 is compatible with all the quartets, return it as the correct key.

Complexity. The attack queries new strucures of plaintexts until a structure includes a right forward pair, and at least one quartet follows the backward characteristic. Each forward pair defines 2^{32} quartets, and the backward probability is 2^{-32}, therefore a right forward pair is detected with probability $1 - 1/e \approx 0.63$. When a right forward pair passes this condition, the attack succeeds for the correct guess of ℓ and $\tilde{m}$.

A structure of 2^{32} plaintexts contains $2^{32+31-6} = 2^{57}$ pairs with one colliding byte in the main diagonal of input. Therefore, we expect $2^{57-81.4} = 2^{-24.4}$ right pairs per structure, and we expect to iterate over $2^{24.4}/0.63 \approx 2^{25}$ structures on

Algorithm 4.5: Low time complexity boomerang attack on 6-round AES.

loop ▷ The expected number of iteration is 2^{25}
  Query the encryption of a strucuture of 2^{32} plaintexts
  for all pairs (P, P') colliding on one input byte, and two output anti-diagonals **do**
    Denote the inactive byte position of $P + P'$ as a
    for $0 \leqslant \ell < 4$ **do**
      Assume $w_0[\ell]$ is inactive for pairs (P, P')
      Deduce a set S_ℓ of 2^{16} candidates for $k_0[5, 10, 15]$
    for $0 \leqslant i < 2^{32}$ **do**
      Define $\bar{C}_i = C + \nabla_i$, $\bar{C}'_i = C' + \nabla_i$,
      Query $\bar{P}_i = E^{-1}(\bar{C}_i), \bar{P}'_i = E^{-1}(\bar{C}'_i)$
    for all pairs $(\bar{P}_i, \bar{P}'_i)$ colliding on input byte a **do**
      for $0 \leqslant \ell < 4$ **do**
        Assume $w_0[\ell]$ is inactive for pairs (P, P') and $(\bar{P}_i, \bar{P}'_i)$
        Filter the set S_ℓ to keep 2^8 candidates for $k_0[5, 10, 15]$
      for $0 \leqslant \tilde{m} < 4$ **do**
        Assume z_4 of pair $(P, \bar{P}_i)$ is active only on position $\tilde{m}$
        Deduce 2^8 key candidates for $k_6[0, 7, 10, 13]$
      for all candidates for $\tilde{m} \parallel k_6[0, 7, 10, 13]$ **do**
        Gather 7 extra quartets s.t. z_4 of pair $(P, \bar{P}_i)$ is active only on position m
        for all candidates for $\ell \parallel k_0[5, 10, 15]$ **do**
          Recover one candidate for $k_0[0]$ using one extra quartet
          Use remaining quartets to verify $k_0[0, 5, 10, 15]$

average before finding a right quartet. This corresponds to $2^{25+57-61.4} = 2^{20.6}$ candidates (P, P') at Step 2. and $2^{20.6+32-8} = 2^{44.6}$ candidates $(\bar{P}_i, \bar{P}'_i)$ at Step 3.

In Step 3, we consider on average $2^{20.6}$ pairs (P, P'), and each of the 2^{24} pairs $(\bar{P}_i, \bar{P}'_i)$ colliding in input byte a. First, we iterate over 2^{10} candidates $k_0[5, 10, 15]$, and verify that byte ℓ is inactive in w_0 for $(\bar{P}_i, \bar{P}'_i)$. This requires 6 S-box evaluations for each candidate. Then, we consider 2^{10} state differences in z_4 of $(P, \bar{P}_i)$ and deduce 2^{10} corresponding candidates for $k_6[0, 7, 10, 13]$. This requires 4 table lookups in the DDT for each candidate. In total, Step 3. requires on average $2^{20.6+24} \times (6 \times 2^{10} + 4 \times 2^{10}) = 2^{57.9}$ lookups.

In Step 4, fetching another pair $(\bar{P}_j, \bar{P}'_j)$ compatible with the characteristic requires 4 AES S-box calls and 1 lookup in the plaintext/ciphertext table of Step 2: compute the first column in z_4 of C from the first anti-diagonal of the ciphertext C (only once, this cost is amortized), shift by any value on byte of position $\tilde{m}$, and compute back the corresponding ciphertext $\bar{C}_j$. Since all 2^{32} ciphertexts $\bar{C}_i$ were queried during the decryption phase, $\bar{C}_j$ belongs to the table of queried ciphertexts. In total, this costs $2^{20.6+24} \times 4 \times 2^{10} = 2^{56.6}$ calls and $2^{20.6+24+10} = 2^{54.6}$ lookups

in the plaintext/ciphertext table, for each of the 8 additional ciphertexts, i.e. $2^{59.6}$ S-box calls, and $2^{20.6+24+10+3} = 2^{57.6}$ lookups in the plaintext/ciphertext table.

The most costly part of the attack is the recovery of 1 average value of $k_0[0]$ from another pair $(\bar{P}_j, \bar{P}'_j)$, given $k_0[5, 10, 15] \parallel \ell$. To do so, we compute the difference in $y_0[5, 10, 15]$ for the pair $(\bar{P}_j, \bar{P}'_j)$ with 6 S-box evaluations, recover the only difference $y_0[0]$ yielding the correct difference pattern in w_0, and fetch the 1 average value $k_0[0]$ satisfying the transition $\bar{P}_j[0] + \bar{P}'_j[0] \rightarrow y_0[0]$. In total, this requires 7 lookups per pair. Then, given another pair $(\bar{P}_j, \bar{P}'_j)$, we verify that the difference in byte ℓ of w_0 is inactive with 8 S-box lookups for each of the 2^{10} candidates $k_0[0, 5, 10, 15] \parallel \ell$. We expect on average 2^2 remaining candidates for $k_0[0, 5, 10, 15] \parallel \ell$, and the rest of the filtering is of negligible complexity. In total, this requires $(7 + 8) \times 2^{20.6+24+10+10} = 2^{68.5}$ lookups, which corresponds to $2^{61.9}$ encryptions.

Further optimization. We observe that we process several pairs corresponding to the same candidate (P, P') and the same guess of $\tilde{m} \parallel k_6[0, 7, 10, 13]$. Indeed, for each (P, P'), we consider 2^{24} pairs $(\bar{P}_i, \bar{P}'_i)$, and 2^{10} values of $\tilde{m} \parallel k_6[0, 7, 10, 13]$ for each. On average, there is one pair $(\bar{P}_i, \bar{P}'_i)$ for each guess of $\tilde{m} \parallel k_6[0, 7, 10, 13]$, but we expect only 0.63×2^{34} distinct values. Pairs corresponding to repeated values of $\tilde{m} \parallel k_6[0, 7, 10, 13]$ are not necessary: assuming that (P, P') is a right forward pair, we only need one pair $(\bar{P}_i, \bar{P}'_i)$ corresponding to the correct guess of $\tilde{m} \parallel k_6[0, 7, 10, 13]$ for the attack to succeed. Therefore, we reduce the complexity by creating a table of size 2^{34} at the beginning of Step 3, and marking candidates $\tilde{m} \parallel k_6[0, 7, 10, 13]$ at the beginning of Step 4 to avoid processing the same guess twice. This reduces the complexity by a factor 0.63, leading to a total complexity of $0.63 \times 2^{61.9} = 2^{61.2}$. The storage of the table of 2^{34} bits corresponds to 2^{27} 128-bit states; it is not the memory bottleneck.

The complexity can potentially be further reduced by requiring the first two pairs $(\bar{P}_j, \bar{P}'_j)$ to be inactive on one byte of the main diagonal of $\bar{P}_j + \bar{P}'_j$. This increases the cost of locating the pairs by a factor 2^6, resulting in $2^{63.6}$ DDT lookups, and $2^{61.6}$ plaintext/ciphertext lookups in total to gather the first two pairs. If this is a negligible in comparison with the rest of attack[5], this reduces the number of lookups required to test a key candidate from $7 + 8$ to $5 + 6$ because one S-box is inactive. This reduces the total complexity to $(5+6) \times 2^{20.6+24+10+10} \times 0.63 = 2^{67.5}$ lookups, which corresponds to $2^{60.9}$ encryptions.

Total complexity. The data complexity corresponds to 2^{57} encryptions and $2 \times 2^{20.6+32} = 2^{53.6}$ decryptions, for a total of $2^{57.1}$ queries. The memory complexity is bounded by the storage of the 2×2^{32} plaintexts $\bar{P}_i, \bar{P}'_i$, for $0 \leqslant i < 2^{32}$ from a forward pair (P, P').

[5]This assumption actually depends on the architecture on which the attack is implemented: if the time required for a lookup in a table of size 2^{32} is large, this technique does not result in a better attack.

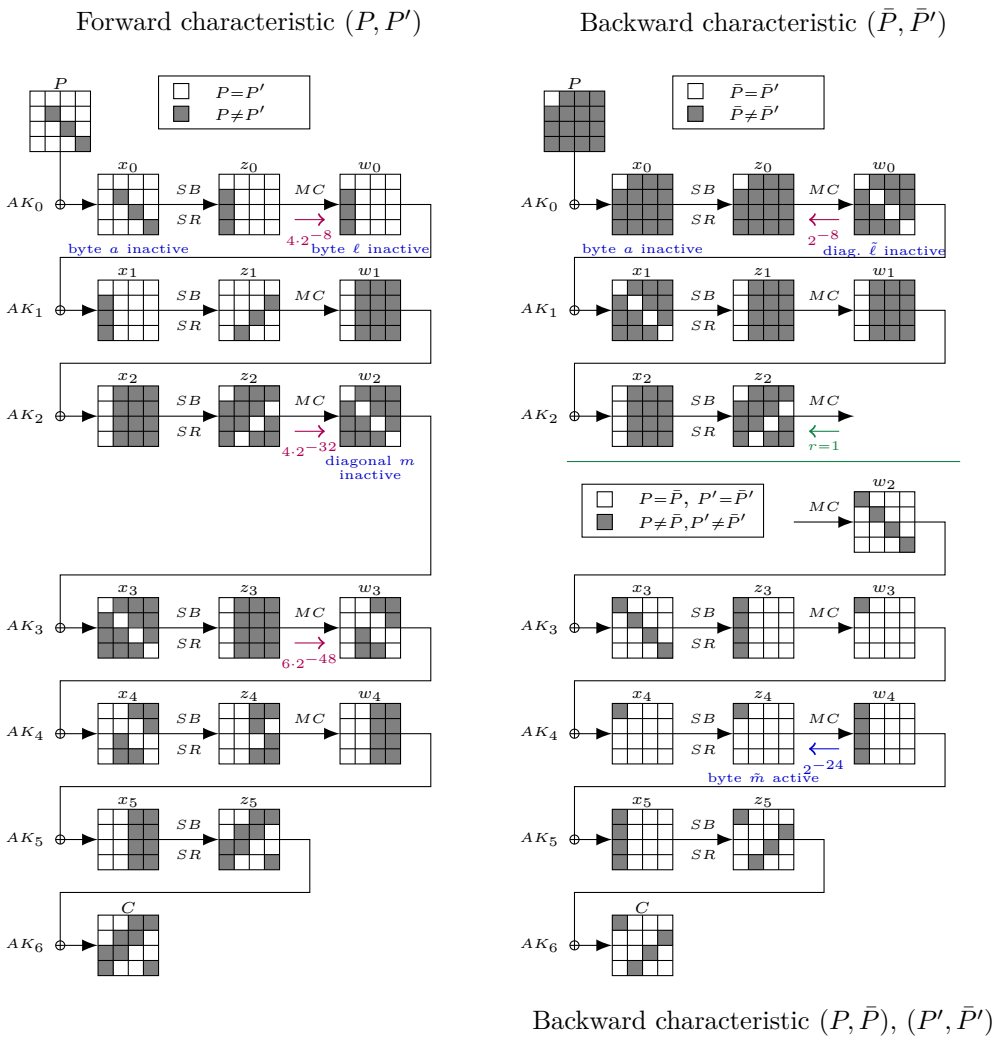


Figure 4.23: Boomerang characteristic on 6-round AES with low time complexity.

This gives:

$$(D, T, M) = (2^{57.1}, 2^{60.9}, 2^{33}).$$

4.5.3 Incompatibility in a 6-Round Distinguisher

At first glance, it seems that the boomerang characteristics presented in the previous sections can easily be modified to obtain a distinguishing attack with a complexity of about 2^{76}. 'All one has to do' is to modify the forward characteristic such that after the first round the difference is active only in a single byte, in order to improve the filtering in the backward direction. The resulting characteristic is presented in Figure 4.24.

However, a closer inspection shows that this 'distinguisher' is in fact flawed. The reason is that the characteristic is actually *impossible* due to an internal inconsistency in the forward trail between z_2 and w_2, as is shown in the figure.

We note that the same flaw appears in the *shifting retracing boomerang* attack on 5-round AES presented in [DKR+19, App. C3] and described in Section 4.3.3. In that attack, one considers pairs of plaintexts for 5-round AES with non-zero difference only in bytes $\{0, 5, 10, 15\}$ and filters out all pairs except for those whose output difference in bytes $\{0, 7, 10, 13\}$ is equal to 0 or δ_L (for some fixed δ_L). Then, the remaining pairs are further analyzed, and it is claimed that the proportion of analyzed pairs is 2^{-31} of the number of initial pairs. However, in fact there is no point in analyzing pairs with zero ciphertext difference in bytes $\{0, 7, 10, 13\}$ since such pairs cannot have non-zero difference in a single byte after the first round (due to the same internal inconsistency as the one shown in Figure 4.24), and so they cannot be part of right boomerang quartets. As a result, the number of right quartets is reduced by a factor of 2, and to compensate for this, the data, time and memory complexities of the attack should be increased by a factor of $2^{0.5}$.

4.6 Conclusion

In this section, we described the boomerang attack, overviewed the existing boomerang-like attacks on AES, and presented new boomerang attacks on AES and some AES-based ciphers. Although the boomerang attack only reaches 6 rounds of AES (the best 7-round attacks reach 7 AES rounds), and is not the best attack against 6-round AES, it is still interesting to study the technique as it may leave room for future improvements. In addition, it is the most efficient existing attack against Deoxys-BC and Kiasu-BC, two AES-based ciphers.

We introduced two main types of boomerang attacks against 6-round AES: the truncated boomerang attack and an improvement to the retracing boomerang attack. The truncated boomerang attack is very generic, so that it can easily be applicable to other AES-based ciphers, and reaches a relatively low data (2^{59}) and time (2^{61}) complexity, but has an unpractical memory complexity (2^{59}). On the other hand, the boomerang attack of Section 4.5.2 has similar data (2^{57}) and time (2^{61}) complexity, but has a practical memory complexity (2^{33}). However, it can hardly be applied to other AES-based ciphers.

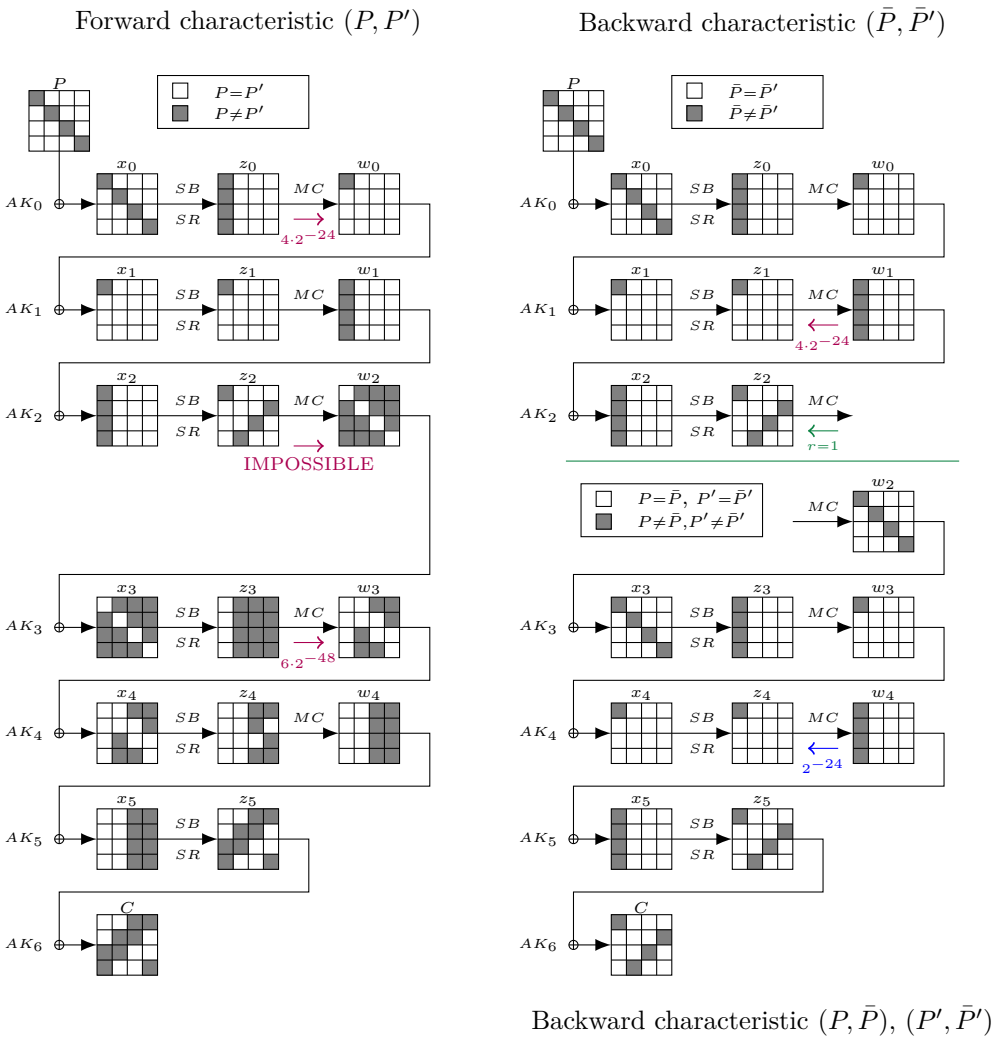


Figure 4.24: An impossible forward pattern for a distinguishing attack.

Chapter 5
Design of Fast AES-based UHFs and MACs

In this chapter, we present a joint work with Jules Baudrin, Gaëtan Leurent, Clara Pernot, Léo Perrin, and Thomas Peyrin, submitted to ToSC 2024. We design a new framework for AES-based Universal Hash Functions, whose candidates reach outstanding performance. We build two MACs based on this framework, LeMac and PetitMac, the first of which is the fastest MAC in the literature, with a throughput of 0.068 cycles per Byte on an Intel Ice Lake (Xeon Gold 5320).

Contents

5.1 Introduction

The AES block cipher [Aes] has had a significant impact on the development of symmetric-key primitives, for two main reasons. First, its inner structure makes its differential and linear properties well-understood. Second, hardware accelerated AES instructions (AES-NI) [Gue08] were introduced in modern CPUs, thus significantly increasing the speed of AES encryptions and decryptions in software. In particular, hardware acceleration of the AES round-function is today widespread in most computing platforms, from high-end Intel/AMD CPUs to microcontrollers for mobile devices, and AES-NI have become even more helpful over the processors generations, with reduced latency and increased throughput. Those reasons led many designers to reuse full AES [MV04; RBB03; KR21b], or only the AES round-function [BBG+08; WP14; Nik14; JNP+21; SLN+21; NFI24], as building blocks for cryptographic constructions.

These technological advances allowed many symmetric-key primitives to eventually reach performances under 1 cycle per byte (c/B), but new use-cases arise. In particular, sixth-generation mobile communication systems (6G) plan to deliver transmissions with an impressive throughput between 100 Gbps and 1 Tbps. This puts a lot of pressure on the encryption/authentication performances and AES-NI-based solutions seem very natural. This is the direction taken by the Authenticated Encryption with Additional Data (AEAD) algorithm Rocca [SLN+21; SLN+22] and its updated version Rocca-S [NFI24], currently the fastest AEAD on AES-NI platforms and under submission at IETF. However, it remains unknown if Rocca's construction is indeed close to the optimal throughput when leveraging AES-NI.

We remarked that despite the numerous AES-based constructions existing in the literature, few AES-based MACs exist. The main example that we are aware of is PC-MAC [MT06][1]. Based on the analysis of the the Maximum Expected Differential Probability (MEDP) of 4-round AES by Keliher and Sui [KS07], they consider 4-round AES as an ε-AXU family with $\varepsilon \approx 1.18 \cdot 2^{-110}$ (under the hypothesis that the round keys are independent). Using this as a building block, they construct a MAC with 4 AES rounds per 128-bit block of plaintext, with provable security.

Moreover, the collision-resistant round functions of Jean & Nikolić [JN16] and the Additional Data phase of Rocca [SLN+21] both achieve 128-bit collision security with 2 AES rounds per 128-bit message; the drawback is that the security is heuristic and not provable. Our goal is to design a new MAC construction with an even better performance than those.

We want to point out that many popular MACs, such as poly1305 [Ber05] are based on arithmetic operations, and are designed for modern processors enabling fast arithmetic. Those processors often support AES-NI instructions as well; we can therefore directly compare the software performance of AES-based MACs to state-of-the-art arithmetic MACs.

[1] Some AEAD constructions with no encrypted data may also be used as MACs, but they are not presented as MAC constructions.

Contributions. In this chapter, we present a family of ε-AU UHFs faster than state-of-the-art UHFs, by exploiting the extremely high throughput of AES-NI instructions, with flexible parameters that can be adapted to future computing platforms. Our construction uses a novel design strategy compared to previous UHFs or collision resistant round functions, with a (potentially large) internal state separated into two parts: one part updated with AES-round functions and 128-bit XORs, and one part storing linear combinations of messages. Each round, new message blocks are injected in the second part, and the second part is injected in the first part. The general idea is that a difference in a single message block influences many inner state AES rounds, and that the collision security remains easily computable using MILP. This separation strategy draws inspiration from recent (tweakable) block cipher designs, where the tweakey schedule is linear, or the Panama [DC98] hash function.

Although this family is too big to be exhausted in practice, we propose a process to iterate over candidates of the family and select some fast and secure ones. We implemented a tool that, given any candidate of this family, automatically computes the number of active S-boxes in the best differential trail, using MILP. The MILP model exactly discards linear incompatibilities, improving on heuristic approaches to avoid linear incompatibilities [CHP+17]. In addition, our tool can compile on-the-fly candidates of the family and benchmark them, in order to automatically measure their speed. To our knowledge, this is the first time that on-the-fly benchmarking is performed to filter candidates in an AES-based framework.

To showcase the relevance of our approach, we present an ε-AU UHF candidate whose round function reaches a speed of 0.067 cycles per byte on Intel Tiger Lake (i5-1135G7). In addition, we show the first candidate with less than 2 AES rounds per 128-bit message block and 128-bit collision security, namely with 1.75 AES rounds per 128-bit message block.

From this family of ε-AU UHFs, we present two new MAC constructions: LeMac and PetitMac. The former is as of today the fastest MAC on modern desktop/server processors, reaching of speed of 0.068 cycles per byte on Intel Ice Lake (Xeon Gold 5320) for 256 kB messages vs. 0.113 cycles per byte for a MAC based on the round function of [JN16], the fastest state-of-the-art MAC according to our benchmarks. PetitMac is slower, but of a smaller size, thus more suitable for lightweight applications.

Outline. We start with a detailed description of our design goals, and their interactions with the state-of-the-art, in Section 5.2. In light of this discussion, we decided to focus our efforts on a specific family of UHFs which we present in Section 5.3. Since this family very large, we reduce the search space using equivalence classes (see Section 5.4), and we automate the security analysis using MILP-based methods described in Section 5.5. The results of our search are presented in Section 5.6. Finally, we use these results to build concrete primitives in Section 5.7, while Section 5.8 concludes the paper.

5.2 Design Goals and First Observations

While several AES-based constructions exist, we identified places where there remains substantial room for improvement. Below, we describe the goals that our family of UHFs are intended to fulfil; the family itself will be described in Section 5.3.

5.2.1 AES-based Round Functions

As already mentioned in introduction, many designs rest upon the AES round function and the 128-bit XOR to be both secure and efficient, thanks to the AES-NI instructions set in modern processors. Among them, the CAESAR candidates Tiaoxin [Nik14] and AEGIS [WP14] (the latter was selected in the final high-performance portfolio) are competitive AEAD schemes. In terms of throughput, they are outperformed by the building blocks designed by Jean & Nikolić [JN16] and later Nikolić [Nik17]. Recently, the AEAD proposals Rocca [SLN+21; SLN+22] and Rocca-S [NFI24] target 6G requirements in terms of speed and security. All of those constructions aim at minimizing the so-called *rate* [JN16], that is, the number of AES rounds per 128-bit message block. Rocca (during Additional Data processing) and one of the schemes of Jean & Nikolić achieve a rate of 2 for 128-bit security. We will adopt a similar strategy and minimize the rate of the round function.

Goal 1. *Our ε-AU families should use AES rounds as internal components for high software performance, and preferably at the lowest rate.*

5.2.2 Instruction Scheduling

Modern processors are superscalar processors with out-of-order execution. They can execute several instructions simultaneously, and schedule instructions as soon as the input operands are ready. Moreover the execution units are pipelined: some instructions take several cycle to process, and the execution unit can start processing a new instruction at every clock cycle, with the output being ready some cycles later.

There are two main metrics to measure the performance of an instruction I:

Latency: the number of clock cycles between the beginning of I to the return of its result. We denote $L(I)$ the latency of I.

Throughput: the number of instructions that can be processed in a given amount of time. We usually consider the reciprocal throughput, measured in cycles. We denote $T(I)$ the throughput of I.

Processors are composed of several execution units, accessed by ports denoted $P_1 \dots P_k$. Each port P_i accepts a certain set of instructions S_i. At cycle t, each execution unit P_i can process an instruction $I \in S_i$, and returns its result $L(I)$

Architecture	Instr	Latency	Throughput	P_0	P_1	P_2	P_3	P_4	P_5	P_6
Intel Haswell	XOR	1	0.33	x	x				x	
	AESENC	7	1						x	
Intel Skylake	XOR	1	0.33	x	x				x	
	AESENC	4	1	x						
Intel Ice Lake	XOR	1	0.33	x	x				x	
	AESENC	3	0.5	x	x					
Intel Tiger Lake	XOR	1	0.33	x	x				x	
	AESENC	3	0.5	x	x					
AMD Zen 1/2/3/4	XOR	1	0.25	x	x	x	x			
	AESENC	4	0.5	x	x					

Table 5.1: Scheduling of AESENC and XOR instructions on modern processors.

cycles later. At cycle $t+1$, the execution unit P_i might process another instruction $I' \in S_i$ (with potentially $I' = I$), even though the instruction I of cycle t has not returned its result yet. The throughput of an instruction I usually corresponds to the number of ports which can process I.

For the AES-based ciphers mentioned in Section 5.2.1, two types of instructions are extensively used: AES rounds instructions (e.g. AESENC), and 128-bit XORs. Each instruction has its own throughput and latency on modern processors, but we cannot exploit the full throughput of both types of instructions at the same time, because they share ports, as illustrated by Table 5.1. In particular, on modern Intel processors (Ice Lake and higher) and AMD processors (Zen1 and higher), AES-NI instructions operate on two ports P_0 and P_1, while the 128-bit XOR operates on three or four: P_0, P_1, P_5 (or P_0, P_1, P_2, P_3).

On the number of XOR instructions in AES-based constructions. In the case of Intel Processors (Ice Lake and higher), the throughput of AESENC encryption is 0.5, and the throughput of XOR is 0.33; AESENC uses ports 0 or 1, and XOR uses ports 0, 1, or 5. In order to fully exploit the throughput of the AESENC instruction, we need to feed ports 0 and 1 only with AESENC instructions at each clock cycle, thus they become unavailable for XOR instructions, and XOR instructions can only be assigned to port 5. Consequently, if x AESENC instructions are executed at full throughput, there can be at most $x/2$ XOR instructions at the same time. Similar observations apply to recent processors, and unfortunately, minmizing the number XOR of AES-based constructions is not a systematic approach[2]. For example, Jean & Nikolić [JN16] present rate-2 candidates with 6 AES per rounds and 9 128-bit XORs, and Rocca's rate-2 round function uses 4 128-bit XORS and 4 AES rounds instructions. As a consequence, regardless of implementation tricks, a full throughput on current modern Intel processors will always remain out of reach for these algorithms.

[2]To the best of our knowledge, minimizing the number of XOR has only been considered in the permutation design of Gueron and Mouha [GM16], which consider zero XOR instruction outside the AESENC instruction.

Dependency chains. In addition to the throughput analysis, dependency chains affect the performance of AES-based constructions. As an example, let us denote x_i the first state element of an AES-based construction at round i. If x_{i+1} depends on x_i, the latency to compute x_{i+1} from x_i is the sum of the latency of each involved instruction. Then, the latency of the round function is at least the latency of computing x_{i+1} from x_i. In the decryption mode of Rocca [SLN+21], we found a cycle of dependency with 1 AESENC and 3 XOR instructions. Indeed, using the notation of [SLN+21, Figure 1], we found the following dependency chain composed of:

$$S^{new}[4] = \mathsf{XOR}(\mathsf{XOR}(C_i^1, \mathsf{AESENC}(\mathsf{XOR}(S[0], S[4]), S[2])), S[3]). \tag{5.1}$$

We notice that this dependancy chain only appears in the decryption mode, since the value X_1 of [SLN+21, Figure 1] needs to be computed from $S[4]$. On Ice Lake and Tiger Lake architectures, this dependency chain first appears to have a latency of $1 \times 3 + 3 \times 1 = 6$ cycles, corresponding to 0.19 cycles per byte, but in practice we measured a speed of around 0.34 cycles per byte, which we believe corresponds to a latency of around 10 cycles. We try to explain this gap in the following paragraph.

Bypass delay. As far as we can tell, there is an additional delay (a *bypass delay*) of two cycles when the output of an AES instruction is used as input of a non-AES instruction. In particular, the latency of the dependency chain of Equation 5.1 increases by 4 cycles (to a total of 10 cycles of latency): the two XOR instructions on the left of Equation 5.1 each have one input (respectively $S[3]$ and $\mathsf{AESENC}(\mathsf{XOR}(S[0], S[4]), S[2])$) that is the output of an AESENC instruction. The latency can be reduced to 8 cycles with a fake XOR instruction added at the beginning of the round : $S[3] \leftarrow \mathsf{XOR}(S[3], 0x00)$, so that at the beginning of each round, $S[3]$ is not the direct result of an AESENC encryption. This was tested experimentally and increases the speed to around 0.25 cycles per byte on Tiger Lake (i5-1135G7). It seems that these 8 cycles of latency can not be reduced, and we therefore believe that Rocca in the decryption mode can not run faster than $8/(2 \times 16) = 0.25$ cycles per bytes on Tiger Lake processors.

In addition to the number of XOR instructions and the dependency chains, there are a lot of processor subtleties, which are difficult to exhaustively consider. As a general guideline, we aim at avoiding any pipelining issue and at being as efficient as possible on modern processors.

Goal 2. *The instruction scheduling in modern processors should be favorable.*

Goal 2 is very reasonable, but is hard to guarantee with pen-and-paper analysis because of the always-evolving, complex, and well-optimized scheduling of modern processors. In fact, one way to directly evaluate the performance with state-of-the-art instruction scheduling algorithms is to compile and benchmark candidates on-the-fly. This strategy exploits advanced techniques from compilers (e.g. modern gcc) or processors, and remains future-proof, since it can easily be adapted to future processors.

Goal 3. *Our tool should automatize the benchmarking of candidates. The automatic benchmarking should be adaptable to all processors.*

5.2.3 Security

We first recall the security definitions of universal hash functions.

Definition 5.1 (ε-AU)**.** A family of functions $H_K : A \to B$ for $K \in \mathcal{K}$ is ε-almost-universal if:

$$\forall m \neq m' \in A,\ |\{K \in \mathcal{K} : H_K(m) = H_K(m')\}| \leqslant \varepsilon|\mathcal{K}|.$$

Definition 5.2 (ε-AXU)**.** A family of functions $H_K : A \to \mathbb{F}_2^b$ for $K \in \mathcal{K}$ is ε-almost-XOR-universal if:

$$\forall m \neq m' \in A,\ \forall d \in \mathbb{F}_2^b,\ |\{K \in \mathcal{K} : H_K(m) \oplus H_K(m') = d\}| \leqslant \varepsilon|\mathcal{K}|.$$

Our strategy is to design a UHF family with the ε-AU security. Therefore, we are only interested in collision resistance. In order to facilitate the security analysis of our candidates, we consider that the output of one of our ε-AU UHFs is not of a single word, but rather the entire state composed of multiple 128-bit words. In addition, we consider that the initial state is the key, so that values of the inner states cannot be exploited to build collisions. Thus, in order to ensure collision resistance, it is sufficient to prevent the existence of high probability differentials leading to collisions. We then rely on the following assumption to investigate these.

Assumption 1. *The highest probability of a differential trail is a good indication of the highest probability of a differential.*

Thanks to 1, estimating the security level can be done by modeling the differential propagation with a MILP model and this is now a widespread practice [JN16; SLN+21].

Goal 4. *A lower bound on the number of active Sboxes in the differential trails of a candidate should be easily computed with computer-aided tools, such as MILP solvers.*

Section 5.5 will be fully dedicated to our MILP modeling and its optimizations.

5.2.4 A Roadmap to Achieve these Goals

All those guiding principles lead us toward the family of UHF that we describe in the next section. Our goal are in line with previous works [JN16; SLN+21]: we want a primitive that favors parallel AES calls to optimize scheduling. However, properly taking this into account means carefully considering the number of 128-bit XORs, and in fact minimizing it—the authors of Rocca already observed the negative impact that AES and XOR used "in a cascade way" could have. As a consequence, we limit ourselves to sparse linear layers.

To compensate the slower diffusion implied by the sparse linear layer, and to broaden our search space, we consider more sophisticated injection techniques inspired by the design of (tweak-)key schedules. This could increase the cost of each round (in particular in terms of memory), but it indeed enables the safe use of very simple round functions. This overall structure is similar to that of Panama [DC98], a hash function attacked in [RVP+02]. It was based on a large "buffer" and a smaller "inner state", the former being linearly updated using message blocks, and the latter being non-linearly updated using data extracted from the buffer. The separation between buffer and inner state was quickly set aside as several algorithms adopted a similar structure that nevertheless involved a datapath from the inner state to the buffer, e.g. RadioGatùn [BDP+06] and Lux [NBK08].

Our whole construction is presented in the next section.

5.3 A Specific Family of Universal Hash Functions

In light of the discussion presented in the previous section, we have settled on a specific family of UHFs, that is both large enough to contain algorithms that are both fast and secure, and that is small enough that we can practically explore significant subsets of it.

The idea is to separate the (potentially large) state into two subparts with different roles: an inner part updated with AES rounds and a linear layer, and an outer part updated only with a linear layer and new message blocks. Each round, words of the outer state are XORed to the inner state (but not the other way). The aim is that each message block is XORed several times into the inner state, so that short differential trails leading to collisions do not exist. This construction is similar to many sponge-like constructions, but in our case the linear outer state allows to save many AES round calls (while sponge-like designs will apply the same function to the full state), and to be easily modelable in MILP. This also resembles a large tweakable block cipher with a large tweak, and a linear tweakey schedule. We chose this structure as it has the potential to offer both high throughput (thanks to its reliance on the AES rounds, the expensive operations being restrained to one subpart of the state, and the potentially low rate) and high security (thanks to the sparsity of the round which makes it easier to use automated tools to check for differential attacks).

5.3.1 Overall Structure

The UHF family we consider is described in Figure 5.1. Each *wire* on the figure represents a 128-bit value. The inner state is on the left-hand side of Figure 5.1, and the outer linear message-schedule with memory on the right. Overall, our approach can be seen like a standard Substitution Permutation Network (SPN): the inner state (alternatively denoted X, Y, Z) is iteratively updated through a

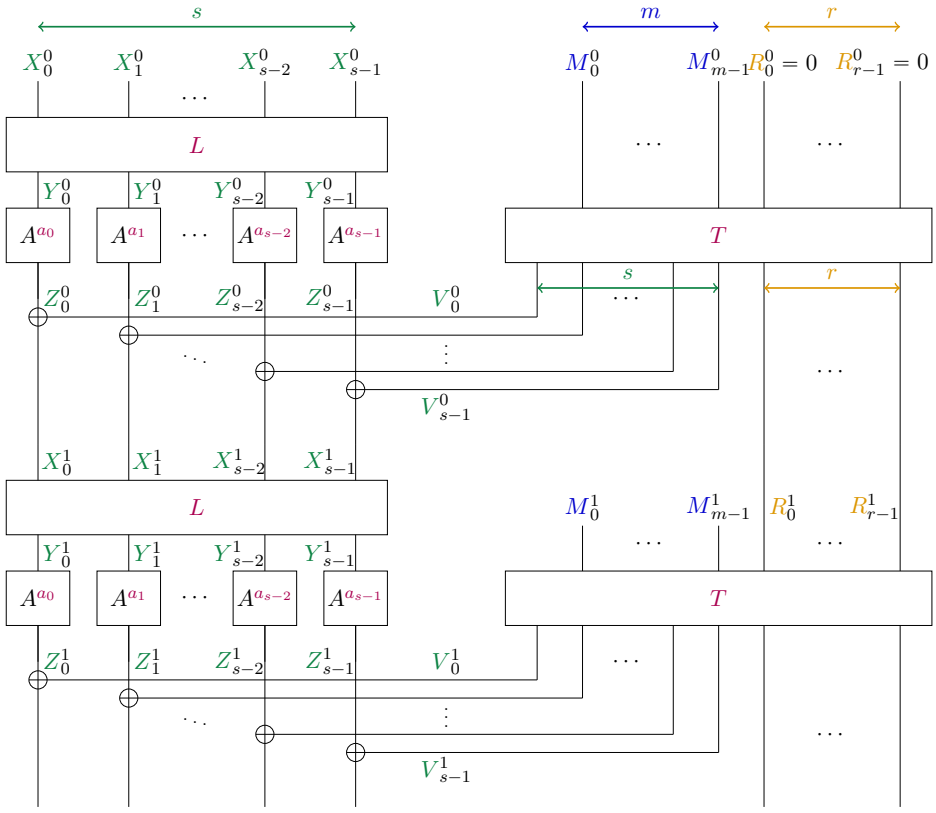


Figure 5.1: Overview of our UHF family. A stands for a key-less AES round. Each choice of the size parameters s, m, r, the Boolean values a_i, and the linear matrices L, T defines an instance of the framework.

round function built by composing a linear layer with a non-linear one. Between each round, the linear message-schedule ingests several blocks of the input message, and produces an *injected value* V which is added to Z to yield X. The memory registers of the linear message-schedule, that we denote R, keep linear information on previous input message blocks.

Parameters. From now on, by *size*, we always mean the number of 128-bit blocks. Thus, each member of the family is parameterized by the sizes s, m, r of the inner state X, Y, Z, of the input message M, and of the memory R, that can be chosen freely. Note that s also corresponds to the size of the injected value V. Once these sizes are fixed, we define a specific instance by choosing the vector a and the matrices L, T.

The Boolean vector $a := (a_0, \cdots, a_{s-1})$, of size s, indicates whether a state wire goes through an AES round or not. For any i such that $a_i = 1$, the i-th wire of the state is called an *AES wire*.

The $s \times s$ invertible sparse matrix $L \in \mathcal{M}_{s\times s}(\mathbb{F}_{2^{128}})$ is used as linear layer. By design, we restrict the coefficients of L to $\{0, 1\}$, so that L can be viewed as a

matrix of $\mathcal{M}_{s\times s}(\mathbb{F}_2)$. In particular, the output of the linear layer is only composed of copies and XORs of the 128-bit input words.

Finally, T is the $(s+r)\times(m+r)$ message-schedule *transition matrix*. T indicates how to compute the s-word injected-value V and how to update the memory R (of size r). Both are linearly computed using the current memory R and m fresh message words, M. Similarly to L, we restrict by design the coefficients of T to $\{0,1\}$: $T \in \mathcal{M}_{(s+r)\times(m+r)}(\mathbb{F}_2)$.

Notation 1 (Time stamp, coordinates and sequences)**.** *As the values of the blocks vary over time, we use superscript to indicate the clock (with $t = 0$ as initial clock) while subscripts are reserved for coordinates: for instance R_i^t stands for the i-th coordinate of R^t, that is, R at time t. We keep plain characters for generic purposes: e.g. the memory R, and use calligraphic letters to denote the sequence throughout time: e.g. $\mathcal{V} := (V^t)_{t\in\mathbb{N}}$. Finally, for any finite subsets $I \subset \mathbb{N}$, $J \subset [\![0, s-1]\!]$ and $t \in \mathbb{N}$, we denote sub-sequences and sub-vectors as: $\mathcal{V}^I := (V^t)_{t\in I}$ and $V_J^t := (V_j^t)_{j\in J}$.*

5.3.2 Round Function and Message-schedule

Round function. It is applied on the inner state, and is composed of three layers:

- a linear layer $Y^t := L(X^t)$.
- an AES-round layer where an AES round A (composed of SubBytes, ShiftRows and MixColumns, but *without* AddKey) is applied in parallel to each AES wire: $Z_i^t := A^{a_i}(Y_i^t)$; where $A^0 := \mathrm{Id}$ and $A^1 := A$.
- and an injected-value addition layer where the injected value V^t of round t, generated by the message-schedule, is added to the state: $X^{t+1} := Z^t + V^t$.

In the AES round layer, the AddKey step is omitted. Thus, the AESENC instructions perform both the AES-round layer and the addition layer.

Message-Schedule. The linear message-schedule has a memory R of size r. Each register contains a linear combination of previous message words. At round t, m new message words are ingested, the s-long injected value V^t is output and the memory R^t is updated, in a single transition step:

$$\forall t \geqslant 0, \quad \begin{pmatrix} R^{t+1} \\ V^t \end{pmatrix} = T \begin{pmatrix} R^t \\ M^t \end{pmatrix}. \tag{5.2}$$

As highlighted by the previous equation, it is convenient to decompose T as a block matrix.

Notation 2 (T decomposition)**.** *In the following, given a transition matrix T, we will intensively use the following decomposition and notation:*

$$T := \begin{array}{c} \overset{r}{\longleftrightarrow}\ \overset{m}{\longleftrightarrow} \\ \begin{pmatrix} T_{00} & T_{01} \\ T_{10} & T_{11} \end{pmatrix} \begin{array}{l} \updownarrow\ r \\ \updownarrow\ s \end{array} \end{array} \tag{5.3}$$

Taking advantages of Equation 5.2 and Equation 5.3, we can easily express the injected-values as (recursive) linear combinations of input messages blocks:

$$\forall t \geqslant 0, \quad R^{t+1} = T_{00}R^t + T_{01}M^t, \qquad V^t = T_{10}R^t + T_{11}M^t. \tag{5.4}$$

Remark 5.1. Let $I \subset \mathbb{N}$, $u := max(i|i \in I)$. $\mathcal{V}^I$ can therefore be viewed as a family of $|I|s$ linear combinations, or equivalently as a $|I|s \times um$ matrix where each column represents one of the um message blocks that can appear in the $|I|s$ combinations. We will often prefer the latter point of view.

A given injected-value sequence $\mathcal{V}$ can be obtained from infinitely many matrices T. For instance, infinitely many unused memory registers could be added. It is thus necessary to limit as much as possible this redundancy while exploring the transition matrices T. In the next section, we start by finding a "normal form" for the transition matrices we will study. We then limit our search by defining an equivalence relation between injected-value sequences and finally present and justify our search space.

5.4 A Searchable Space of UHFs

5.4.1 A Normal Form for Transition Matrices

The first notable point about transition matrices is that, at clock t, only the space spanned by the memory registers (and not the register themselves) matters. Indeed, the same information can be recovered from two different spanning families, only in different representation systems. This is illustrated by the following proposition.

Proposition 5.1 (Change of basis for memory registers)**.** *Let T be a transition matrix. Let $P \in \mathrm{GL}_r(\mathbb{F}_2)$. Let us define $T_P \in \mathcal{M}_{(s+r)\times(m+r)}(\mathbb{F}_2)$ such that:*

$$T_P = \begin{pmatrix} PT_{00}P^{-1} & PT_{01} \\ T_{10}P^{-1} & T_{11} \end{pmatrix}. \tag{5.5}$$

Then T_P produces the same sequence $\mathcal{V}$ as the original matrix T.

Proof. Let us denote for any $t \geqslant 0$, R_P^t, V_P^t the respective memory registers and round-message at clock t produced by T_P. By adapting Equation 5.4 to T_P, we obtain:

$$\forall t \geqslant 0, \quad R_P^{t+1} = PT_{00}P^{-1}R_P^t + PT_{01}M^t, \qquad V_P^t = T_{10}P^{-1}R_P^t + T_{11}M^t. \tag{5.6}$$

By design, $R^0 = 0$ and $R_P^0 = 0$ because the memory is initialized as such. In particular $R_P^0 = PR^0$. Let $t \geqslant 0$ and let suppose that $R_P^t = PR^t$. Then by injecting $R_P^t = PR^t$ into Equation 5.6 and simplifying we get,

$$\begin{aligned} R_P^{t+1} &= PT_{00}P^{-1}R_P^t + PT_{01}M^t \\ &= PT_{00}P^{-1}PR^t + PT_{01}M^t \\ &= PT_{00}R^t + PT_{01}M^t \\ &= P(T_{00}R^t + T_{01}M^t) = PR^{t+1}. \end{aligned}$$

This proves by induction that $R_P^t = PR^t$ for any $t \geqslant 0$.

Let $t \geqslant 0$. According to Equation 5.6, $V_P^t = T_{10}P^{-1}R_P^t + T_{11}M^t$. Replacing R_P^t by PR^t, we obtain:

$$V_P^t = T_{10}P^{-1}PR^t + T_{11}M^t = T_{10}R^t + T_{11}M^t = V^t;$$

which proves the announced equality for any $t \geqslant 0$. □

For fixed sizes r, m, s, Proposition 5.1 in particular states that it is sufficient to explore a single representative per similarity class for the top-left block T_{00}. We recall the following classical results about similarity that can be found for instance in [DF04, Thm. 14, p. 476] or [Gan90, p. 192].

Definition 5.3 (Similarity)**.** Let $n \in \mathbb{N}\backslash\{0\}$. Let $\mathbb{F}$ be a field. Let $M, N \in \mathcal{M}_n(\mathbb{F})$. M is *similar* to N if there exists a matrix $P \in \mathrm{GL}_n(\mathbb{F})$ such that $M = PNP^{-1}$.

Proposition 5.2 (Normal Form for similarity)**.** *Let $M \in \mathcal{M}_n(\mathbb{F})$. Let us denote C_Q the companion matrix associated to a polynomial $Q \in \mathbb{F}[X]$.*

1. *Similarity is an equivalence relation over $\mathcal{M}_n(\mathbb{F})$. We denote it $\sim$.*
2. *There exists a unique family $(Q_0, \cdots, Q_{\ell-1})$ of polynomials such that:*

$$Q_{\ell-1}|\cdots|Q_1|Q_0 \quad \textit{and} \quad M \sim \mathrm{Diag}(C_{Q_0}, \cdots, C_{Q_{\ell-1}}).$$

This representative is called Frobenius Normal Form *or* Rational Canonical Form.

According to Proposition 5.2, it is thus sufficient to exhaust all possible Frobenius Normal Forms rather than all $r \times r$ matrices for the top left-hand corner. This decreases the search space by a significant factor: for $r = 4$, there are 2^{16} matrices in $\mathrm{GL}_4(\mathbb{F}_2)$, but only $34 \approx 2^5$ equivalence classes.

On top of that, Proposition 5.1 also allows to get rid of redundant memory registers, as presented by the following corollary.

Corollary 5.1. *Let T be a transition matrix. Let us denote $d = \mathsf{rank}(T_{00}|T_{01})$. Then, there exists an instance using d memory-registers which generates the same sequence $\mathcal{V}$.*

Proof. If $d = r$ then T generates $\mathcal{V}$ and has d memory-registers. Let us now suppose that $d < r$. In that case, we can find $P \in \mathrm{GL}_r(\mathbb{F}_2)$ such that the first $r-d$ rows of $(PT_{00}|PT_{01}) = P(T_{00}|T_{01})$ are all-0. $(PT_{00}P^{-1}|PT_{01})$ naturally shares the same property, and according to Proposition 5.1, T_P produces the same round-message sequence. But the $r-d$ first empty rows in T_P indicates that the first $r-d$ memory registers will be zero at any time $t \geqslant 0$, and therefore will never impact the output sequence. T_P can thus be adapted by removing the $r-d$ null rows in the upper half, and removing the corresponding $r-d$ columns in the left-hand half. The obtained matrix T' generates the same sequence with d memory registers. □

Corollary 5.1 states that after choosing a Frobenius Normal Form for T_{00}, and any value for T_{01}, one can immediately look at the rank of the top half $(T_{00}|T_{01})$. If the top half has not full rank, the study of the matrix comes back to the study of an instance with strictly less memory (a smaller r). If the search is done by increasing values of r, one can restrict the search to a top half with a full rank.

5.4.2 Equivalent Injected-value Sequences

Even if we limit redundancies thanks to Proposition 5.1 and Corollary 5.1, for most of values of r, m, s, the associated space of message-schedules remains too big. In particular, it cannot be exhaustively searched, especially if a MILP problem needs to be optimized *for each instance*.

To reduce further the explored space, we first restrict ourselves to matrices T for which $\mathsf{rank}(T_{11}) = m$. Indeed, if $\mathsf{rank}(T_{11}) < m$, only a strict subspace of the messages at round t impacts the injected values at this round. This does not directly generate collisions, since the unused messages can be stored in memory and used in later rounds. However, this requires extra registers whose only purpose is to store the unused injected messages of previous rounds, increasing the memory size r without increasing the security. Precisely, after a few rounds, such an instance behaves as if exactly m message blocks impacted the injected values at each round; the message blocks sequence being slightly slid. So from now on, we consider $\mathsf{rank}(T_{11}) = m$, and in particular, $s \geqslant m$.

Secondly, we take into account our adversary in a scenario where it has a full control over the input differences in message blocks (such as a chosen-plaintext scenario). From this point-of-view, the implementation does not matter, only the *actual* decompositions of all V_i^t as linear combinations of $M_j^{t'}, i, j \in [\![0, m-1]\!], t' \leqslant t$ do. In particular, with n degrees of freedom, such an adversary can choose the differences of n independent V_i^t, rather than just the differences of n message blocks M_i^t. We thus study injected-value sequences up to linear change of variables of the inputs.

Definition 5.4 (Linearly-equivalent injected-values sequences)**.** Let $\mathcal{V} = (V^t)_{t\in\mathbb{N}}$ and $\mathcal{W} = (W^t)_{t\in\mathbb{N}}$ be sequences of linear combinations such that, for any i, t, V_i^t depends only on $M_j^{t'}$, where $j \in [\![0, m-1]\!], t' \leqslant t \in \mathbb{N}$. Then, $\mathcal{V}$ is *linearly-equivalent* to $\mathcal{W}$ if:

$$\forall\ t \in \mathbb{N}\backslash\{0\},\ \exists\ P^t \in \mathrm{GL}_{tm}(\mathbb{F}_2), \quad V^{[\![0,t-1]\!]} = W^{[\![0,t-1]\!]}P^t;$$

where P^t is a *lower triangular block matrix*[3] whose blocks are of size $m \times m$.

Remark 5.2. Let $t \in \mathbb{N}\backslash\{0\}$. The lower triangular form of P^t implies that the equivalence relation preserves the fact that only variables $M_i^{t'}$, $t' \leqslant t$ appear in both V^t and W^t.

Proposition 5.3. *Linear equivalence of injected-value sequences, as defined in Definition 5.4, is an equivalence relation.*

Proof. The invertible lower triangular matrices is a sub-group of $\mathrm{GL}_{tm}(\mathbb{F}_2)$. Let $t > 0$ and $V^{[\![0,t-1]\!]} = W^{[\![0,t-1]\!]}P^t$, $W^{[\![0,t-1]\!]} = X^{[\![0,t-1]\!]}Q^t$. Reflexivity is proved with Id, symmetry using $(P^t)^{-1}$ and transitivity using P^tQ^t. □

Proposition 5.4. *Let T be a transition matrix such that $\mathsf{rank}(T_{11}) = m$. Then, up to a wire permutation of the inner state, T produces a sequence $\mathcal{V}$ which is linearly-equivalent to the sequence produced by $\widetilde{T}$, where:*

$$\widetilde{T} = \begin{array}{c} \overset{r}{\longleftrightarrow}\ \overset{m}{\longleftrightarrow} \\ \left(\begin{array}{cc} \star & \star \\ \hline 0 & \mathrm{Id}_m \\ \star & \star \end{array}\right) \begin{array}{l} \updownarrow r \\ \updownarrow m \\ \updownarrow s-m \end{array} \end{array}$$

Proof. By hypothesis, $\mathsf{rank}(T_{11}) = m$, so at each round, the information of the m independent new message blocks is fully contained in m of the round-value blocks. In other words, there exists m indices $I = \{i_0, \cdots, i_{m-1}\}$ such that for any t, $V_I^t = \begin{pmatrix} F & C \end{pmatrix} \times \mathcal{M}^{[\![0,t]\!]}$, where $C \in \mathrm{GL}_m(\mathbb{F}_2)$ (and $F \in \mathcal{M}_{m\times(t-1)m}(\mathbb{F}_2)$). *Up to a wire permutation of the inner state*, let us assume that $I = [\![0, m-1]\!]$. In that case, $(T_{10}|T_{11})$ can be decomposed, such that C appears in it:

$$\begin{pmatrix} T_{10} & T_{11} \end{pmatrix} = \begin{array}{c} \overset{r}{\longleftrightarrow}\ \overset{m}{\longleftrightarrow} \\ \begin{pmatrix} B & C \\ D & E \end{pmatrix} \begin{array}{l} \updownarrow m \\ \updownarrow s-m \end{array} \end{array} \tag{5.7}$$

Now, let $\ell \in \mathbb{N}\backslash\{0\}$ and let us consider the following change of variables:

$$\forall t \in [\![0, \ell-1]\!], \quad \widetilde{M}^t := BR^t + CM^t \iff C^{-1}(\widetilde{M}^t - BR^t) = M^t.$$

Because R^t is a linear combination of $M_i^{t'}$ where $t' < t$, this change of variables corresponds to a lower triangular block matrix P^t (whose diagonal is only made of C blocks).

Decomposing V^t as $V^t = (V^t_{[\![0,m-1]\!]} \mid V^t_{[\![m,s-1]\!]})$, we can rewrite the linear relations in Equation 5.4 using the decomposition of $(T_{10}|T_{11})$ given in Equation 5.7:

$$R^0 = 0, \quad \forall t \geqslant 0,\ R^{t+1} = T_{00}R^t + T_{01}M^t,$$
$$\forall t \geqslant 0,\ V^t_{[\![0,m-1]\!]} = BR^t + CM^t, \quad V^t_{[\![m,s-1]\!]} = DR^t + EM^t.$$

[3] $V^{[\![0,t-1]\!]}$ and $W^{[\![0,t-1]\!]}$ are viewed as matrices of dimension $ts \times tm$, see Remark 5.1.

By substituting M^t in the previous equations we obtain:

$$R^0 = 0, \quad \forall t \geqslant 0,\ R^{t+1} = T_{00}R^t + T_{01}C^{-1}(\widetilde{M}^t - BR^t),$$
$$V^t_{[\![0,m-1]\!]} = BR^t + CC^{-1}(\widetilde{M}^t - BR^t), \quad V^t_{[\![m,s-1]\!]} = DR^t + EC^{-1}(\widetilde{M}^t - BR^t);$$

which, once simplified and reorganized, become:

$$R^0 = 0, \quad \forall t \geqslant 0,\ R^{t+1} = (T_{00} - T_{01}C^{-1}B)R^t + T_{01}C^{-1}\widetilde{M}^t,$$
$$V^t_{[\![0,m-1]\!]} = \widetilde{M}^t, \quad V^t_{[\![m,s-1]\!]} = (D - EC^{-1}B)R^t + EC^{-1}\widetilde{M}^t.$$

Thus, the sequence $\mathcal{V}$, is linearly-equivalent to the sequence generated by the transition matrix $\widetilde{T}$ defined as:

$$\widetilde{T} := \left(\begin{array}{c|c} T_{00} - T_{01}C^{-1}B & T_{01}C^{-1} \\ \hline 0 & \mathrm{Id}_m \\ D - EC^{-1}B & EC^{-1} \end{array}\right). \tag{5.8}$$

$\widetilde{T}$ has the announced form. □

We can now present the chosen form for the explored transition matrices.

Theorem 5.1. *Let T be a transition matrix such that $\mathsf{rank}(T_{11}) = m$. Then, up to a wire permutation of the inner state, T produces a sequence $\mathcal{V}$ which is linearly-equivalent to the sequence produced by a matrix $\widetilde{T}$ of the following form:*

$$\widetilde{T} = \begin{array}{c} \overset{r}{\longleftrightarrow}\ \overset{m}{\longleftrightarrow} \\ \left(\begin{array}{cc} F & \star \\ \hline 0 & \mathrm{Id}_m \\ \star & \star \end{array}\right) \begin{array}{l} \updownarrow r \\ \updownarrow m \\ \updownarrow s-m \end{array} \end{array} \tag{5.9}$$

where F is a Frobenius Normal Form matrix.

Proof. First using Proposition 5.4, we obtain, up to a wire permutation of the inner state, a transition matrix $\widetilde{T}$ which produces a linearly-equivalent sequence, as in Equation 5.8. We can now use Proposition 5.1, in order to put the top left-hand block in Frobenius Normal Form. The multiplication of the lower-left part by P^{-1} does not change the fact that the first row of this block are all-0. The lower-right block is not modified, so Id_m still appears on its first rows. T' has thus the announced form. □

The class of matrices presented in Theorem 5.1 is not only chosen to make the search more efficient, but also for its *sparsity* to guarantee a small implementation cost. Indeed, the Frobenius Normal Form constitutes a very sparse representative of a similarity class: it is a sparse matrix (a diagonal block matrix) with sparse non-empty blocks (companion blocks). The chosen form for the lower half is also quite sparse with the 0 and Id blocks.

5.4.3 Constraints on the Linear Layer

Regarding the linear diffusion matrix L, it should be implementable with a low number of XORs. However, we must ensure that each inner state block at round i will eventually influence all of them. To this end, we use a simple heuristic.

Definition 5.5. Let $\hat{L}$ be a matrix identical to a binary matrix L, except that its coefficients are integers. The *diffusion time* of L is the smallest integer i such that all coefficients in $(\hat{L})^i$ are non-zero. If no such integer exists, we set it to $+\infty$.

We consider integers rather than binary field elements so that additions do not cancel out. Intuitively, this number tells how many rounds are needed to ensure full diffusion in the inner part. In our search space, we generate matrices L under weight constraints, often with a weight of $s+1$ or $s+2$ so that L can be implemented with 1 or 2 XORs and ignore matrices with high diffusion time: we mostly use a value of around $2 \times s$, although we show a candidate with rate 1.75 and infinite diffusion time in Section 5.6.

5.4.4 The Actual Explored Space

The search method presented above is optimized but heuristic : we stress that we do not assure the minimal sparsity of the studied transition matrices. Still, the explored space contains promising candidates (see Section 5.6), that could be further-optimized later on. Nevertheless, exhaustive search remains unreachable. Equivalence relations on a and L could be used, but would (and in practice do) interfere with the previous ones. Instead, we restrict the weight of a and L, as described in Section 5.4.3 and further in Section 5.6.

5.5 Turning Collision Resistance into a MILP Problem

The search space being established, we now focus on assessing the security of the potential UHF candidates, by building an adapted MILP model and then solving it thanks to an optimizer. An overview of MILP is performed in Section 2.4.

5.5.1 Prior Works

The use of MILP modeling for searching differential trails with the highest probability was set to light by Mouha, Wang, Gu & Preneel in 2011 [MWG+11]. Several approaches exist depending on the needed level of precision and the available computational power. In theory, by using one MILP variable for each bit of the state at each round, all the non-linear differential transitions could be modeled (at the cost of *many* constraints). This approach is in practice very costly. For byte-aligned (resp. nibble-aligned) primitives, it is much faster and practical to rather affect a MILP variable to each nibble (resp. byte) of the state. Yet less

precise, such a model enables (if it can be efficiently solved) to determine the minimum number of active Sboxes, from which an upper bound on the probability of the best differential trail can easily be estimated. In the case of AES-based ciphers, this method has become standard, as highlighted by Rocca [SLN+21] or Deoxys-BC [JNP+21]. Following their lead, we consider the byte-wise approach.

To do so, we extend Notation 1 so that the byte position appears.

Notation 3. *The second subscript indicates the byte position:* $X^i_{j,\ell}$ *is the* ℓ*-th byte of* X^i_j.

5.5.2 Our Model

From now on, a candidate has been chosen: s, m, r and a, T, L are now fixed. To these constants, we add ρ, the number of rounds of the primitive to model.

Variables. Let $i \in [\![0, \rho-1]\!]$ be a round number, $j \in [\![0, B-1]\!]$ be a word number (where the bound $B \in \{s, m, r\}$ depends on the register we look at) and $\ell \in [\![0, 15]\!]$ be a byte position. We track the differential activeness of every byte throughout the rounds by modeling each byte of the state as a *binary variable*, that is equal to 0 if the byte is inactive and 1 if it is active. We use lowercase to denote the binary variables. More precisely $x^i_{j,\ell}, y^i_{j,\ell}, z^i_{j,\ell}, r^i_{j,\ell}, m^i_{j,\ell}, v^i_{j,\ell}$ respectively represents the bytes $X^i_{j,\ell}, Y^i_{j,\ell}, Z^i_{j,\ell}, R^{j,\ell}_i, M^{j,\ell}_i, V^i_{j,\ell}$.

Objective. Our goal is to minimize the number of active Sboxes, represented by the variables $y^i_{j,l}$ on AES wires (i.e. $j \in \mathrm{Supp}(a)$):

$$\mathrm{Obj} := \sum_{i=0}^{\rho-1} \sum_{j \in \mathrm{Supp}(a)} \sum_{\ell=0}^{15} y^i_{j,l}.$$

Now, we present constraints that will appear in the definition of more advanced ones.

Multiple-XOR. It models the relation $\bigoplus_{i=0}^{N-1} U_i = 0$ where $(U_i)_{i \in [\![0,N-1]\!]}$ is a list of N bytes represented by N binary variables $(u_i)_{i \in [\![0,N-1]\!]}$. To do so, we introduce an auxiliary binary variable α, and two constraints:

$$\alpha N \geqslant \sum_{i=0}^{N-1} u_i \quad \text{and} \quad \sum_{i=0}^{N-1} u_i \geqslant 2\alpha.$$

In that way, depending on $\alpha \in \{0, 1\}$, either 0 or at least 2 bytes are active.

MDS constraint. It models the relation between an input column of bytes, represented as the binary variables $(y_i)_{i \in [\![0,3]\!]}$, and an output one, represented

as $(z_i)_{i\in[\![0,3]\!]} \in \{0,1\}^4$, through the AES MDS matrix. With an auxiliary binary variable α, and the two constraints:

$$10\alpha \geqslant \sum_{i=0}^{3} y_i + z_i \quad \text{and} \quad \sum_{i=0}^{3} y_i + z_i \geqslant 5\alpha;$$

either 0 or at least 5 bytes are active.

Remark 5.3. In the above constraints, Σ corresponds to an integer sum, *not a modulo-2 sum.*

We can now create constraints for each layer of the round function. Let $i \in [\![0, \rho-1]\!]$.

Linear layer. The transition through L is naturally expressed by linear relations between bytes. Denoting $L = (L_{j,k})_{j,k\in[\![0,\cdots,s-1]\!]}$ (where $L_{j,k} \in \mathbb{F}_2$ for any j,k), it holds that:

$$\forall i \in [\![0,\rho-1]\!], j \in [\![0,s-1]\!], \ell \in [\![0,15]\!], \qquad Y^i_{j,\ell} = \sum_{k=0}^{s-1} L_{j,k} X^i_{k,\ell}.$$

These constraints can therefore be modeled using a Multiple-XOR constraint.

AES-round layer. Let $j \in \mathrm{Supp}(a)$ so that an AES round is applied on the j-th wire. The S-box layer does not change the activity pattern, but the linear layer (ShiftRows and MixColumns) needs to be modeled. For any round $i \in [\![0,\rho-1]\!]$, and column index $t \in [\![0,3]\!]$, the t-th diagonal of Y^i_j is linked by a MDS relation together with the t-th column Z^i_j. Those relations require a MDS constraint.

When $j \notin \mathrm{Supp}(a)$, we simply add the constraints $y^i_{j,\ell} = z^i_{j,\ell}$ for all i,ℓ.

Message-schedule. The 128-bit linear relations between R^i, M^i, R^{i+1}, V^t given by Equation 5.4 can be modeled with 16 Multiple-XOR constraints (one for each byte).

Injected-value addition. For all i,j,ℓ, $X^{i+1}_{j,\ell} = Z^i_{j,\ell} + V^i_{j,\ell}$ is modeled as a Multiple-XOR.

Finally, we add constraints on the inputs/outputs of the UHF, and constraints to take advantage of the inherent symmetries of the AES round function.

Input constraints. At clock $t = 0$, the state and memory are fully inactive. Thus,

$$\forall \ell \in [\![0,15]\!], j \in [\![0,s-1]\!], j' \in [\![0,r-1]\!], \qquad x^0_{j,\ell} = 0, \quad r^0_{j',\ell} = 0.$$

Message constraints. If a trail with an inactive first round exists, shifting it by 1 round makes it still a valid trail. Moreover, in the AES, any column (resp. row) plays the same role, so any trail can be shifted so that the first difference appears in the byte of index $\ell = 0$. By forcing at least one 0-index first-round-message byte to be active, we facilitate the solving process, without leaving any trail aside. Hence the *symmetry constraint*:

$$\sum_{j=0}^{m-1} m_{j,0}^{0} \geqslant 1.$$

This model will be referred as our *basic model*. Additionally, we can add to this model some output and/or linear incompatibilities constraints.

Output constraints. We can force the state to be fully inactive at the end:

$$\forall \ell \in [\![0, 15]\!], j \in [\![0, s-1]\!] \qquad x_{j,\ell}^{\rho} = 0.$$

This constraint highly reduces the MILP solution space. However it is a too-strong constraint when ρ is small: a differential trail over more rounds but with less active S-boxes cannot be captured by the model. In practice, we iteratively increase ρ to capture more and more trails, until a sufficient number of rounds is reached.

Removing linear incompatibilities. With the basic model, some obtained activity patterns may not be instantiable into differential trails because of linear incompatibilities, similar to the ones observed on AES [FJP13] or on Deoxys-BC [CHP+17]. Unlike those ciphers, our message-schedule is acting on 128-bit words which enables us to model the linear incompatibilities with exact constraints[4]. In our case, for an AES wire of index j, we observe that $\mathsf{MC}\circ\mathsf{SR}\circ\mathsf{SB}(Y_j^i)\oplus V_j^i = X_j^{i+1}$. Introducing, the variables

$$\widehat{X}_j^i := \mathsf{LIN}^{-1}(X_j^i), \quad \widehat{V}_j^i := \mathsf{LIN}^{-1}(V_j^i), \quad \widehat{M}_j^i = \mathsf{LIN}^{-1}(M_j^i), \quad \widehat{R}_j^i = \mathsf{LIN}^{-1}(R_j^i), \tag{5.10}$$

where $\mathsf{LIN} := \mathsf{MC} \circ \mathsf{SR}$, we can rewrite it as:

$$\forall j \in \mathrm{Supp}(a), i \in [\![0, \rho-1]\!], \qquad \mathsf{SB}(Y_j^i) \oplus \widehat{V}_j^i = \widehat{X}_j^{t+1}. \tag{5.11}$$

Getting rid of linear incompatibilities in the model precisely means taking Equation 5.11 into account. To do so, we introduce the binary MILP variables $\hat{x}_{j,\ell}^i, \hat{v}_{j,\ell}^i, \hat{m}_{j,\ell}^i, \hat{r}_{j,\ell}^i$ corresponding to the bytes of $\widehat{X}_j^i, \widehat{M}_j^i, \widehat{R}_j^i, \widehat{V}_j^i$. We then build constraints corresponding to Equation 5.10 using MDS-constraints (SR only consists in a renumbering). The bytes of $\widehat{R}_j^i, \widehat{M}_j^i, \widehat{R}_j^{i+1}$ and $\widehat{V}_j^i$ are linearly-dependent with respect to Equation 5.4; this is encoded using multiple-XOR constraints. Finally,

[4] On the contrary, the tweakey-schedule of Deoxys-BC includes a byte permutation. This is the reason why, Cid *et al.* used heuristic constraints based on degrees of freedom.

because $\mathsf{Sbox}(Y^i_{j,\ell})$ is active if and only if $Y^i_{j,\ell}$ is, Equation 5.11 is encoded byte-wise with 3-XORs between $y^i_{j,\ell}$ (no hat), $\hat{v}^i_{j,\ell}$, and $\hat{x}^i_{j,\ell}$.

In practice the model solving is severely slowed by taking these extra constraints into account. It however often increases the minimal number of active S-boxes. Because of the pros and cons of each of these additional constraints, we parameterize our model depending on them. In Section 5.6, we explain how we parameterized the models to converge toward promising candidates.

Notation 4. *Model(ρ, lin=True/False, output=True/False) is the model with ρ rounds and with (no) output (resp. linear incompatibilities) constraints depending on lin (resp. output).*

A word on solutions. As already mentioned, a solution to these models consists in an activity pattern, which, *if it is instantiable*, minimizes the number of active S-boxes. There is however no *a priori* guarantee that it actually can be instantiated as an actual differential trail. Nevertheless, if it is instantiable, and if all transitions can occur with maximal probability, then the instantiated trail would have a probability of p^N, where N is the number of active S-boxes, and $p = \delta 2^{-k}$ is the probability associated to the differential uniformity δ of the k-bit S-box. Thanks to Assumption 1, this upper bound on the probability of the best differential trail enables to estimate the level of security of any candidate (once the solver terminates). Section 5.6 presents our experimental results.

5.6 Experimental Results of the Search for Good Candidates

5.6.1 Search Strategy

In order to find good candidates, we proceed as follows.

1. First, we fix some numerical values for s, m, r, and $w := |\mathrm{Supp}(a)|$, a maximum number of XORs to implement L and T, and a diffusion time threshold for L.

2. Then, we generate random candidates for a, L, and for T according to Section 5.4.

3. For each generated candidate, we solve Model(ρ, lin=False, output=True) using Gurobi [Gur23], with increasing ρ.

4. For each candidate with a sufficient number of active S-boxes (i.e. more than 24), we generate C code corresponding to the round function, compile it and benchmark it on a recent CPU. If the software performance is high enough, we keep the candidate.

5. Finally, we select one of the final candidates based on performance/security trade-off, and solve Model(ρ, lin=True, output=False) with high ρ to guarantee the security of the candidate.

In Step 2, in practice, a is generated randomly among vectors of s elements with hamming weight m, L is generated from a random element of the symmetric group $S_s(\mathbb{F}_2)$, of which k 0s replaced by 1s (the implementation of L thus requires k XORs at most), and T is generated by looping over the set of possible Frobenius Normal Forms, until the XOR-cost is less than j. The rest of the matrix T is generated, line by line, by making sure that the XOR-cost constraint is satisfied. In our search, k and j are empirically randomized.

In Step 3, by making ρ bigger, we go from very restrictive and quickly-solved models to more complete but slower ones. Optionally, Step 4 can be executed before increasing ρ, to discard non-performing candidates and avoid time-consuming MILP solving.

Running the Search. In practice, we select $\rho \in \{2, 3, 4, 12, 20\}$. At each point, if the number of active S-boxes falls under a *security threshold*, the candidate is discarded. The security threshold is fixed to 20 active S-boxes, but by using lin=True in a later step, the minimal number of active S-boxes might increase. Between the runs with $\rho = 12$ and $\rho = 20$, we automatically generate a C implementation, compile it on-the-fly and benchmark it. If the speed (in cycles per byte) of the candidate, falls under a *speed threshold*, the candidate is discarded. This threshold depends on the parameters (chosen in Step 1) and of the processor used in the benchmark.

Because our candidates rely on AES-NI instructions, they require at least $u \times v$ cycles per 128-bit, where u is the throughput of AESENC, and v the rate (see Section 5.2.1). Thus, candidates with a speed close to this bound are considered promising. In our case, we benchmark on an Intel 11th Gen Core i5-1135G7 (Tiger Lake family), with a throughput $u = 0.5$, and we mainly target round functions with rate $v = 2$. Those round functions cannot go faster than 1 cycle per 128-bit state, i.e. 0.0625 cycles per byte. For rate-2 candidates, if $w = 8$, we set the speed threshold to 0.08 cycles per byte; if $w < 8$, very few candidates are faster than 0.08 cycles per bytes, so the threshold is increased accordingly.

For each remaining candidate, we finally solve Model(20, lin=True, output=False) in order to obtain a final bound on the number of active S-boxes.[5] This heuristic finds candidates with both good performance and security but may not be the fastest approach. Still, it is much faster than simpler approaches such as solving the slow-but-accurate Model(high ρ, lin=True, output=False) directly, or benchmarking every candidate before running the fastest Model(low ρ, lin=False, output=True).

[5] When output=False, reducing ρ increases the solution space. Thus, a final solve with a smaller ρ would also give a legitimate, but potentially loose lower bound, and may thus decrease the solving time.

Table 5.2: Table of the retained candidates over different parameters sets. Speeds were measured on a Intel 11th Gen Core i5-1135G7 (Tiger Lake) for different sizes of message.

								Speed (cy/B)		
Rate	w	m	s	r	XOR-cost	Diffusion	Security	16 kB	256 kB	Descr.
2	8	4	9	4	4	15	26	0.074	0.067	Figure 5.3
1.75	7	4	10	5	5	∞	23	0.079	0.068	Equation 5.12
2	6	3	7	4	4	11	25	0.086	0.080	Equation 5.13
2	4	2	6	4	3	9	24	0.104	0.099	Equation 5.14
2	2	1	4	3	4	5	23	0.180	0.175	Equation 5.15
2	1	0.5[1]	1	5	3/1[2]	-	26	0.374	0.371	Figure 5.4

[1] A message is added every other round.
[2] There is 1 inherent XOR in the transition matrix. Every other rounds, the message accounts for 2 additional XORS.

Choosing the numerical parameters. We chose numerical parameters, based on the following experimental observations. First, for a fixed rate, increasing w (and therefore m) tends to improve the performance. Moreover, when other parameters are fixed, increasing r or s tends to increase the security. Finally, we limited the sizes of the state to $r + s < 16$.

We thus looked for candidates with a high w and multiple memory registers. We also explored lighter candidates, and propose good candidates for smaller values of w, typically $w \in \{2, 4, 6\}$, which are not as fast but might be parallelizable in some scenarios. Finally, for $w = 1$, $s = 1$, $m = 1$, we looked at a rate-2 construction that lies slightly outside the scope of our family by replacing any message block M_0^t with odd t by 0.

5.6.2 Results of the Search

The results of our search are given in Table 5.3. For each set of numerical parameters, we give the total number of candidates we considered ("Total"), the number among them that satisfied the security threshold ("After Sec."), and the number of among those that also satisfied the speed threshold ("After Speed"). As we can see, the vast majority of the candidates do not satisfy our demanding criteria, but a broad-enough search allows us to find promising candidates. The case of $w = 1$ is peculiar as such candidates are inherently slower (they are not parallelizable), which is why the bottom right cell of the table is left empty. The properties of the most promising candidates are given in Table 5.2. Interestingly, for a fixed rate, a higher weight w usually means a higher speed.

$$L = \begin{bmatrix} 0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0 \\ 0&0&1&0&0&0&0&0&0&1 \\ 0&1&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0 \end{bmatrix}, \quad T = \left[\begin{array}{cccc|cccc} 0&0&0&0&0&0&1&0 \\ 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&1 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ \hline 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&1&0&0 \\ 0&1&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0 \end{array}\right], \tag{5.12}$$

$$A = (1100111110), \quad r = 5, \quad s = 10, \quad m = 4\ .$$

$$L = \begin{bmatrix} 0&1&0&0&0&0&0 \\ 0&1&0&1&0&0&0 \\ 1&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1 \\ 0&0&1&0&0&0&0 \end{bmatrix}, \quad T = \left[\begin{array}{cccc|ccc} 0&0&0&0&0&1&1 \\ 1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&1&0&0&0&0 \\ \hline 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&1 \\ 0&1&0&0&0&0&0 \end{array}\right], \quad A = (1011111), \quad r = 4, \quad s = 7, \quad m = 3\ . \tag{5.13}$$

$$L = \begin{bmatrix} 1&1&0&0&0&0 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \\ 1&0&0&0&0&0 \\ 0&0&1&0&0&0 \\ 0&0&0&1&0&0 \end{bmatrix}, \quad T = \left[\begin{array}{cccc|cc} 0&0&0&1&1&0 \\ 1&0&0&0&0&0 \\ 0&1&0&0&0&0 \\ 0&0&1&0&0&0 \\ \hline 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \\ 0&0&0&0&1&0 \\ 0&0&0&0&1&0 \\ 1&0&0&0&0&0 \\ 0&0&0&0&0&0 \end{array}\right], \quad A = (011110), \quad r = 4, \quad s = 6, \quad m = 2\ . \tag{5.14}$$

$$L = \begin{bmatrix} 0&0&1&0 \\ 1&0&0&0 \\ 0&0&1&1 \\ 0&1&0&0 \end{bmatrix}, \quad T = \left[\begin{array}{ccc|c} 0&0&1&1 \\ 1&0&0&0 \\ 0&1&0&0 \\ \hline 0&0&0&1 \\ 0&0&0&1 \\ 0&0&0&1 \\ 0&0&0&1 \end{array}\right], \quad A = (1100), \quad r = 3, \quad s = 4, \quad m = 1\ . \tag{5.15}$$

Figure 5.2: Specification of some UHF candidates.

Table 5.3: Number of tested and passed candidates for different settings. Candidates were generated so that they satisfy the diffusion threshold. The search time is given in core hours. The search was performed on Cascade Lake Intel Xeon 5218.

Meta parameters			Thresholds			Candidates			
Rate	w	m	Diff.	Time Security	Speed	Search time	Total	After Sec.	After Speed
2	8	4	15-22	22-24	0.07	>300k	8.0M	346	13
2	6	3	12-16	23	0.09	28k	1.2M	154	4
2	4	2	10-12	23	0.105	18k	2.2M	187	3
2	2	1	4-8	23	0.19	30k	1.2M	1,165	13
2	1	0.5	2	21/23[1]	-	1,152	908[2]	9	-

[1]We first used lin=False with a security threshold of 21, then lin=True and a threshold of 23.
[2]We exhausted all candidates with $r \leqslant 5$ up to tweakey sequence equivalence.

5.7 Concrete MAC Instances

Using the results of our search, we propose two new AES-based MACs: LeMac and PetitMac. We first present the common framework we use to turn our UHF round functions first into full-fledged UHFs, and second into MACs (Section 5.7.1). We then provide more detailed benchmarks and compare them with the state-of-the-art in Section 5.7.2.

5.7.1 Specifications

To turn our fast universal hash function into a MAC, we use the following strategy:

1. Using one of the round functions we obtained in our search, we get an ε-AU family H taking an arbitrary message as input with a $128 \cdot s$-bit output. The family is indexed by the secret initial state, and we conjecture that it is a 2^{-128}-AU family based on our MILP analysis.

2. We compose H with an ε-AXU family C taking a $128 \cdot s$-bit input with a 128-bit output. We obtain an ε'-AXU family $C \circ H$.

3. We use the EWCDM construction [CS16], with the ε'-AXU family $C \circ H$, and the AES block cipher.

We thus obtain a MAC whose security relies only on the PRP security of AES and the ε-AU security of H, the former being a standard assumption, and the latter being a consequence of our MILP-based analysis.

ε-AXU family C. We build the family C using the sum hashing construction from [CW79, Proposition 4]. Given two ε-AXU family $H_1 : A_1 \to B$ and $H_2 : A_2 \to B$, this construction yields an ε-AXU family $G : A_1 \times A_2 \to B$ defined as $G = \{m \mapsto (h_1(m) + h_2(m)) : h_1 \in H_1, h_2 \in H_2\}$. Concretely, we take the AES

block cipher as an ε-AXU family (the ε-AXU security of AES is a consequence of its the security as a PRP), and define the family C as:

$$C : x_0, x_1, \ldots x_{s-1} \mapsto \bigoplus_{i=0}^{s-1} \mathrm{AES}_{k_i}(x_i),$$

where each AES is keyed independently.

C is a 2^{-128}-AXU family assuming that the AES is a secure PRP, and the composition of the 2^{-128}-AU family H and the 2^{-128}-AXU family C yields a 2^{-127}-AXU family $C \circ H$ using the composition result from [Sti92, Theorem 5.6].

EWCDM. The MAC itself follows the EWCDM construction by Cogliati and Seurin [CS16]:

$$\mathrm{EWCDM}[H, E]_{k_1,k_2,k_3}(M, N) = E_{k_3}\big(H_{k_1}(M) + E_{k_2}(N) + N\big).$$

This construction uses a nonce to obtain high security, but it still provides security up to $2^{n/2}$ queries if the nonces are repeated (or omitted).

When used with unique nonces, EWCDM was initially proved secure up to $2^{2n/3}$ queries, but a more recent result proved security up to essentially 2^n queries: assuming that the adversary makes less than $2^n/67$ queries, Mennink and Neves [MN17] proved that:

$$\mathbf{Adv}^{\mathrm{MAC}}_{\mathrm{EWCDM}[H,E]} \leqslant \mathbf{Adv}^{\mathrm{PRP}}_{F} + \frac{q}{2^n} + \frac{q^2\varepsilon}{2^n} + 2^{-n}.$$

We use the EWCDM construction because it provides significantly higher security than the more common Wegman-Carter-Shoup construction

$$\mathrm{WCS}[H, E]_{k_1,k_2}(M, N) = H_{k_1}(M) + E_{k_2}(N).$$

Indeed, Wegman-Carter-Shoup only provides $2^{n/2}$ security with unique nonces, and fails completely when nonces are repeated.

Initialization. While the family is indexed by the secret initial state, we suggest to derive it as follows: the branch with index i is initialized to $E_{\mathsf{Kinit}}(i)$, where Kinit is 128-bit secret key, and E is the AES-128 block cipher.

LeMac. It is our ultra-fast MAC algorithm. It takes as input a 128-bit nonce and a 128-bit key, and returns 128-bit digest. It is based on the round function summarized in Figure 5.3, which corresponds to the fastest promising candidate we found for $w = 8$.

The state is initialized as state above. During the UHF finalization, each branch of the inner state goes through 10 independent AES rounds with subkeys that were derived as encrypted counters. We derive only 18, and use those in a rolling fashion in each different branch. The idea is to save space by not having to store $10 \times s = 90$ different 128-bit subkeys for this final step. Its specification is given in Algorithm 5.1.

Algorithm 5.1: LeMac, with 128-bit master key K, 128-bit nonce N, and 128-bit message blocks $M_0, ..., M_{4\ell-1}$.

▷ Key derivation
$\mathsf{Kinit} \leftarrow (\mathsf{AES}_K(0), ...\mathsf{AES}_K(8))$
$\mathsf{Kfinal} \leftarrow (\mathsf{AES}_K(9), ...\mathsf{AES}_K(26))$
$k_2 \leftarrow \mathsf{AES}_K(27)$
$k_3 \leftarrow \mathsf{AES}_K(28)$
▷ UHF
$X^0 \leftarrow \mathsf{Kinit}$ ▷ Start of initialization
$R^0 \leftarrow (0, 0, 0)$
▷ Padding
$\ell \leftarrow \lceil(\text{bitlen}(M) + 1)/512\rceil$
$M_0, ..., M_{4\ell-1} \leftarrow M \| 10^*$
$M_{4\ell}, ..., M_{4\ell+11} \leftarrow 0$
for all $0 \leqslant i < \ell + 3$ **do** ▷ Start of absorption (see Figure 5.3)
  $X_0^{i+1} \leftarrow X_0^i + X_8^i$
  for all $1 \leqslant j < 9$ **do**
    $X_j^{i+1} \leftarrow A(X_{j-1}^i)$
  $X_0^{i+1} \leftarrow X_0^{i+1} + M_{4i+2}$
  $X_1^{i+1} \leftarrow X_1^{i+1} + M_{4i+3}$
  $X_2^{i+1} \leftarrow X_2^{i+1} + M_{4i+3}$
  $X_3^{i+1} \leftarrow X_3^{i+1} + R_1^i + R_2^i$
  $X_4^{i+1} \leftarrow X_4^{i+1} + M_{4i}$
  $X_5^{i+1} \leftarrow X_5^{i+1} + M_{4i}$
  $X_6^{i+1} \leftarrow X_6^{i+1} + M_{4i+1}$
  $X_7^{i+1} \leftarrow X_7^{i+1} + M_{4i+1}$
  $X_8^{i+1} \leftarrow X_8^{i+1} + M_{4i+3}$
  $R_0^{i+1} \leftarrow M_{4i+2}$
  $R_1^{i+1} \leftarrow R_0^i + M_{4i+1}$
  $R_2^{i+1} \leftarrow R_1^i$
for all $0 \leqslant j < 9$ **do** ▷ Start of finalization
  for all $0 \leqslant i < 10$ **do**
    $X_j^{\ell+3+i+1} \leftarrow A\left(X_j^{\ell+3+i} + \mathsf{Kfinal}_{i+j}\right)$
$h \leftarrow \sum_{j=0}^{8} X_j^{\ell+13}$
▷ EWCDM
return $\mathsf{AES}_{k_3}(h + \mathsf{AES}_{k_2}(N) + N)$

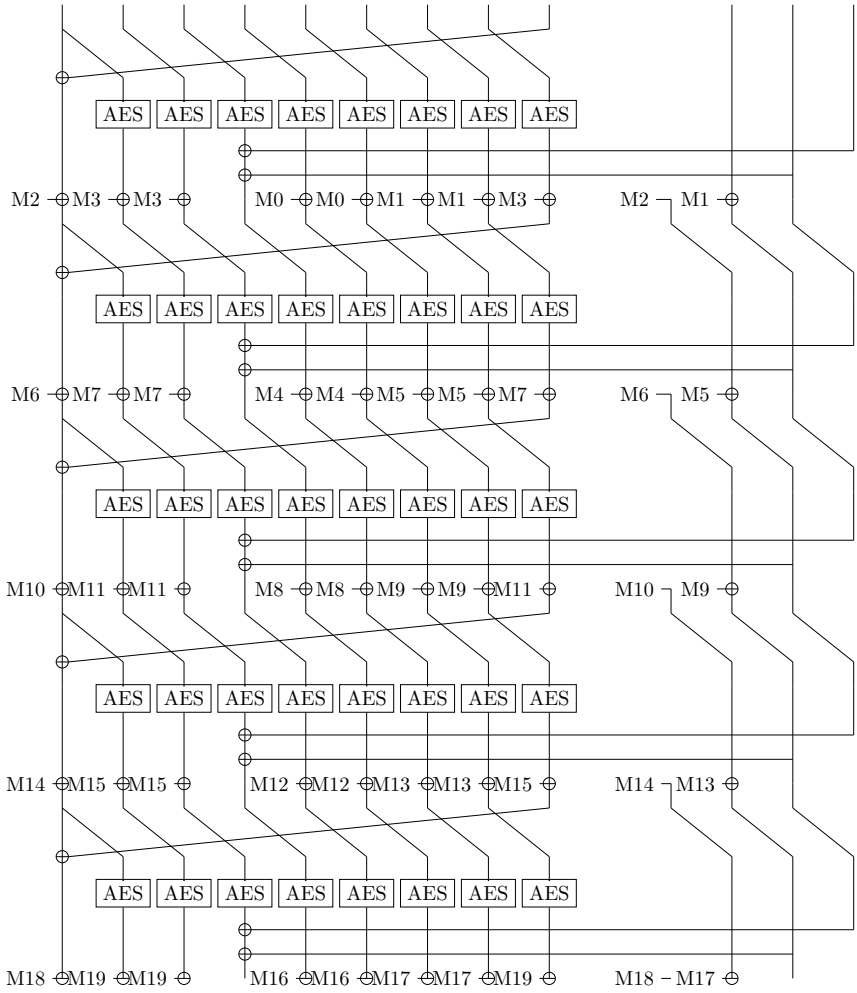


Figure 5.3: Five rounds of the UHF used in LeMac.

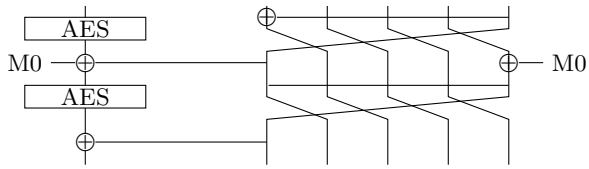


Figure 5.4: Processing one message block in the UHF used in PetitMac.

PetitMac. For cases where the high parallel potential of LeMac might not be an advantage (e.g. on smaller processors), we propose instead PetitMac, which is based on the promising candidate we found for $w = 1$ (see Table 5.2, and Figure 5.4 for its round function). It has a rate of 2, and ensures the activation of at least 26 S-boxes during absorption. Its specifications are given in Algorithm 5.2.

Algorithm 5.2: PetitMac, with 128-bit master key K, 128-bit nonce N, and 128-bit message blocks $M_0, ..., M_{\ell-1}$.

▷ Key derivation
$\mathsf{Kinit} \leftarrow \mathsf{AES}_K(0)$
$\mathsf{Kfinal} \leftarrow (\mathsf{AES}_K(1), ...\mathsf{AES}_K(15))$
$k_2 \leftarrow \mathsf{AES}_K(16)$
$k_3 \leftarrow \mathsf{AES}_K(17)$
▷ UHF
$X^0 \leftarrow \mathsf{Kinit}$ ▷ Start of initialization
$R^0 \leftarrow (0, 0, 0, 0, 0)$
▷ Padding
$\ell \leftarrow \lceil (\text{bitlen}(M) + 1)/128 \rceil$
$M_0, ..., M_{\ell-1} \leftarrow M \| 10^*$
for all $0 \leqslant i < \ell$ **do** ▷ Start of absorption (see Figure 5.4)
$t \leftarrow A(X^i) + M_i + R_4^i$
$R_0^{i+1} \leftarrow M_i + R_3^i$
$R_1^{i+1} \leftarrow R_4^i + R_0^{i+1}$
$R_2^{i+1} \leftarrow R_4^i + R_0^i$
$R_3^{i+1} \leftarrow R_1^i$
$R_4^{i+1} \leftarrow R_2^i$
$X^{i+1} \leftarrow A(t) + R_0^{i+1}$
▷ Finalization
for all $0 \leqslant i < 10$ **do** ▷ Start of finalization
$X^{\ell+i+1} \leftarrow A\left(X^{\ell+i} + \mathsf{Kfinal}_i\right)$
for all $0 \leqslant j < 5$ **do**
for all $0 \leqslant i < 10$ **do**
$R_j^{\ell+i+1} \leftarrow A\left(R_j^{\ell+i} + \mathsf{Kfinal}_{i+j+1}\right)$
$h \leftarrow X^{\ell+10} + \sum_{j=0}^{4} R_j^{\ell+10}$
▷ EWCDM
return $\mathsf{AES}_{k_3}\left(h + \mathsf{AES}_{k_2}(N) + N\right)$

5.7.2 Benchmarks

In order to evaluate the performance of LeMac, we have performed comparative benchmarks on several recent hardware architectures from Intel and AMD. We compare LeMac with the following constructions: Rocca [SLN+21] and Rocca-S [NFI24]; AEGIS128 [WP14] and AEGIS128L [WP13]; Tiaoxin-346 v2 [Nik14]; The rate-2 round function of Jean and Nikolić [JN16], with the same initialisation and finalization as LeMac. Rocca, Rocca-S, AEGIS128, AEGIS128L and Tiaoxin-346 are authenticated encryption algorithms, therefore they provide more features than LeMac, but we believe they still provide a reasonable comparison point when used in their associated data-processing mode.

The results are shown in Table 5.4. We observe that LeMac essentially reaches the maximal possible performance for a rate-2 scheme on these CPU architectures: the Haswell and Skylake architectures compute at most 1 AES round per cycle, corresponding a limit of 0.125 cycle per byte, while the Tiger Lake and Zen 3 architectures compute at most 2 AES rounds per cycle, corresponding to a limit of 0.0625 cycle per byte. Tiaoxin, Rocca, and the Jean-Nikolić round function also have rate 2, but they don't allow enough parallelism to reach this bound.

We have also implemented and benchmarked PetitMac on a microcontroller setting. More precisely, our benchmarks were run on the STM32F407VG microcontroller, which is based on the ARM Cortex-M4 processor. For the AES round implementation we used the T-table-based one written in ARMv7-M assembly from [SS16] while we implemented the round function in C code. The code was compiled using `arm-none-eabi-gcc 10.3.1` with the `-O3` optimization flag and the processor was clocked at 24MHz to take advantage of zero wait-states. Processing 16384-byte messages required 299509 clock cycles (witout the initialization and finalization), leading to 18.3 c/B. This performance places PetitMac as a very competitive MAC on microcontrollers, even though it was not directly designed for that platform (AES round is probably not the best starting point).

As expected, PetitMac is not competitive on high-end desktop, because we have to perform two sequential AES rounds per input block, and the latency of the AES instruction is the bottleneck.

5.8 Conclusion

In this chapter, we introduced a novel family of extremely fast UHFs, optimized for servers/desktop computers with AES-NI. Our general construction is large enough to offer high granularity to contain interesting security/performance tradeoffs, while ensuring a manageable automated security analysis with MILP. Our strategy to search for good candidates within this family is fully automated and adaptable to the performance profiles of future processors. We showcased the validity of our approach by proposing concrete UHFs and corresponding MACs, largely improving over the state-of-the-art on recent processors. Notably, our proposal LeMac is currently the fastest MAC on the high-profile use-case of AES-NI platforms.

CPU	Cipher	Speed (c/B) 1kB	16kB	256kB
Intel Haswell (Xeon E5-2630 v3)	GCM (AD only)	1.138	0.700	0.605
	Rocca (AD only)	0.602	0.225	0.201
	Rocca-S (AD only)	0.660	0.290	0.269
	AEGIS128L (AD only)	0.542	0.299	0.285
	Tiaoxin-346 v2 (AD only)	0.489	0.207	0.190
	Jean-Nikolić	0.455	0.149	0.159
	LeMac	0.498	0.148	0.131
	PetitMac	1.116	0.890	0.876
Intel Skylake (Xeon Gold 6130)	GCM (AD only)	0.817	0.396	0.370
	Rocca (AD only)	0.573	0.190	0.167
	Rocca-S (AD only)	0.568	0.213	0.192
	AEGIS128L (AD only)	0.505	0.267	0.253
	Tiaoxin-346 v2 (AD only)	0.473	0.206	0.189
	Jean-Nikolić	0.389	0.142	0.130
	LeMac	0.422	0.144	0.126
	PetitMac	0.792	0.635	0.626
Intel Ice Lake (Xeon Gold 5320)	GCM (AD only)	0.699	0.311	0.286
	Rocca (AD only)	0.528	0.171	0.149
	Rocca-S (AD only)	0.478	0.172	0.151
	AEGIS128L (AD only)	0.416	0.208	0.195
	Tiaoxin-346 v2 (AD only)	0.328	0.131	0.121
	Jean-Nikolić	0.307	0.126	0.113
	LeMac	0.289	0.082	0.068
	PetitMac	0.521	0.384	0.376
AMD Zen1 (EPYC 7301)	GCM (AD only)	0.932	0.567	0.538
	Rocca (AD only)	0.431	0.147	0.127
	Rocca-S (AD only)	0.438	0.159	0.142
	AEGIS128L (AD only)	0.376	0.177	0.181
	Tiaoxin-346 v2 (AD only)	0.358	0.142	0.129
	Jean-Nikolić	0.338	0.125	0.112
	LeMac	0.330	0.088	0.076
	PetitMac	0.670	0.511	0.501
AMD Zen3 (EPYC 7513)	GCM (AD only)	0.794	0.470	0.451
	Rocca (AD only)	0.393	0.139	0.124
	Rocca-S (AD only)	0.417	0.157	0.141
	AEGIS128L (AD only)	0.358	0.183	0.173
	Tiaoxin-346 v2 (AD only)	0.311	0.121	0.109
	Jean-Nikolić	0.312	0.111	0.098
	LeMac	0.309	0.085	0.072
	PetitMac	0.655	0.510	0.501

Table 5.4: Benchmark results.

Chapter 6
Algebraic Cryptanalysis of Arithmetization-Oriented Primitives

This chapter presents some algebraic attacks on recent class of ciphers, named arithmetization-oriented ciphers. We first give an introduction to algebraic attacks, and then present some attacks on real-world ciphers, that were introduced in a joint work with Clémence Bouvier, Gaëtan Leurent, Léo Perrin published at ToSC 2022 [BBL+22], and in a preprint on my own [Bar23]. These attacks first represent the target ciphers as a systems of polynomial equations, and then solve the systems. In these attacks, we present some new modelings that reduce the number of equations in the polynomial system of several ciphers, using algebraic theory on Gröbner bases [Buc76]. In the next chapter, Chapter 7, we introduce a new type of algebraic attack; so-called FreeLunch attacks.

Contents

6.1 Arithmetization-Oriented Ciphers

6.1.1 Context

Some recent advanced protocols, such as Zero-Knowledge proofs (ZK), Multi-Party Computation (MPC), or Fully-Homomorphic Encryption (FHE) have emerged as a rising field in modern cryptography. Most ZK-proof systems, MPC and FHE protocols operate on large finite fields $\mathbb{F}_q$, q being a prime or a power of 2 typically at least equal to 2^{63} [BCC+12; DPS+12; DZ13; BBH+18].

The use of cryptographic permutations or symmetric key ciphers within ZK and MPC protocols has various applications. Indeed, cryptographic permutations are a solid building block for hash functions, which are extensively used in ZK protocols. In MPC, the multi-party evaluation of symmetric key ciphers allows to reduce the memory requirements when the protocol is paused: the shared data is encrypted under a shared key, the ciphertext is then recovered and stored on a public server, while each party keeps its share of the key until the protocol resumes.

Measuring the cost of an AES implementation in new MPC protocols has become a standard for measuring the efficiency of a MPC protocol [PSS+09; DKL+12; KSS13; DLT14]. However, in order to use a traditional cipher (such as the AES) within some finite field based protocols, its implementation needs to be converted into sequences of finite field operations. The performance of the cipher thus highly depends on its resulting arithmetic circuit.

The term *arithmetization-oriented (AO)* describes primitives that are *not* optimized for time and memory complexities on the usual platforms, but rater to have an efficient representation as an arithmetic circuit, where the arithmetic operations considered are the addition and the multiplication in a large finite field $\mathbb{F}_q$. *Arithmetization-oriented ciphers (AOC)* are commonly defined as ciphers that operate directly in the native finite field $\mathbb{F}_q$ of a protocol, with large q. It is important to note that AOC are only defined as families of ciphers, since there exist as many variants of the cipher as the number of allowed field size q in the target protocol.

In 2015, LowMC [ARS+15] was the first attempt to propose a design aiming at minimizing the arithmetic circuit cost by minimizing the boolean multiplicative complexity [BPP00] of the cipher. In 2016, Albretch *et al.* designed the first AOC family, MiMC [AGR+16], a family of cryptographic functions operating directly on the native field $\mathbb{F}_q$ of the protocol. Multiple other AOC families were later introduced, such as Jarvis [AD18], Vision, Rescue [AAB+20], Poseidon [GKR+21], Ciminion [DGG+21b], `Anemoi` [BBC+23], Griffin [GHR+23], Hydra [GØS+23], XHash8, XHash12 [AKM23] and Arion [RST23].

MPC, FHE and ZK protocols all differ in substance, so the cost metric of a design may only be considered relative to a specific protocol. In general, the cost of a cipher depends mostly on the number of field multiplications, but the subtleties of the arithmetization protocol considered impacts the cost of each operation, and may in fact enable the use of more sophisticated operations (for example, `Reinforced Concrete` [GKL+22] uses the ability of Plonk [GWC19] to evaluate lookup tables). As a first approximation, AOC should have low degree round functions. This was the idea behind the designs Feistel–MiMC [AGR+16], Poseidon [GKR+21], and Ciminion [DGG+21b]. Later, a more sophisticated approach was introduced by the designers of Rescue [AAB+20; SAD20], who remarked that for some MPC and ZK protocols, such a round function also had an efficient implementation if the *inverse* of the round function had a low degree. The designers of `Anemoi` [BBC+23] went further by showing that in some ZK protocols, the cost of the round function in the protocol can be reduced to the cost of any CCZ-equivalent [CCZ98] of the round function. Similar strategies were used in the design of Griffin [GHR+23], XHash8, XHash12 [AKM23] and Arion [RST23].

6.1.2 Security

All AO primitives use field multiplications to provide non-linearity, and many primitives use power-mapping S-boxes $x \to x^\alpha$, invertible when $\alpha \nmid q-1$. Both the multiplication of two unkown elements and the power-mapping S-box have strong differential and linear properties. For instance, the differential uniformity of the power mapping $x \to x^\alpha$ is $\alpha - 1$. Indeed, the equation $(x+\delta_i)^\alpha - x^\alpha = \delta_o$ is of degree exactly $\alpha - 1$ in x when $\delta_i, \alpha \neq 0$, and therefore has at most $\alpha - 1$ solutions for x, when δ_i and are fixed. This implies that the probability of an active differential trail going through a component $x \to x^\alpha$ is at maximum $\frac{\alpha-1}{q}$, which is very low for values of α and q used in practice ($\alpha < 20$ and $q > 2^{63}$). This tends to show that AOC are often strong against traditional attacks, but it should not be generalized: Zhang *et al.* [ZLL+23], showed linear trails with correlation 1 exist on Aiminion, a variant of Ciminion under weak round constants. Subspace-based attacks against Poseidon [GKR+21] were also revealed in [BCD+20; KR21a].

Because of their rather simple algebraic representation, AOC are particularly vulnerable to a class of attacks called *algebraic attacks*. These attacks exploit low degree algebraic relations between the inputs, outputs and (if applicable) the key. They can take different forms, and some well-known algebraic attacks are applicable to non-AO ciphers. Among the most common algebraic attacks, we can list cube attacks [DS08], interpolation attack [JK97], the division property [Tod15] and Gröbner basis attacks [Buc76; SS21].

The AOC Jarvis [AD18] was found to be vulnerable to some algebraic attacks [ACG+19]: it turns out that the system of equations used by the designers to model its round function could be greatly simplified systematically. The initial security margin of the AO block cipher MiMC [AGR+16] against higher-order differentials was also re-assessed twice, first by Eichlseder *et al.* [EGL+20],

then by Bouvier *et al.* [BCP22]. Beyne *et al.* [BCD+20] presented higher-order differentials against the permutations the permutations GMiMC [AGP+19] and Poseidon [GKR+21].

A common method to analyse AO permutations is to consider the constrained-input constrained-output (CICO) problem, presented in Problem 6.1. It was initially proposed by the designers of Keccak [BDP+09] as a crucial problem for estimating the security of permutations used in sponge constructions. A natural instance in the context of algebraic cryptanalysis may be stated in the following form.

Problem 6.1 (CICO problem for AO permutations)**.** *Let* $F : \mathbb{F}_q^t \mapsto \mathbb{F}_q^t$ *be a permutation and* $1 \leqslant \ell < t$ *an integer. The goal is to find* $x \in \{0\}^\ell \times \mathbb{F}_q^{t-\ell}$ *such that* $F(x) \in \{0\}^\ell \times \mathbb{F}_q^{t-\ell}$.

In this thesis, we only focus on the case $\ell = 1$ for AO ciphers, and suppose that q is large enough so that the generic method for finding a CICO problem is unfeasible.

The pressing need for a better understanding of the security of AO hash functions has pushed the Ethereum Foundation to put forward a bounty, published on 01/11/2021 at `https://www.zkhashbounties.info/`, rewarding with thousands of dollars the best *practical* attacks against round-reduced versions of the permutations underlying several sponge-based AO hash functions, namely `Reinforced Concrete` [GKL+22], Feistel–MiMC [AGR+16], Poseidon [GKR+21], and Rescue–Prime [AAB+20]. The challenges set out by the Etheureum Foundation are described in Table 6.1.

6.2 Algebraic Attacks

As their name suggests, algebraic attacks exploit the algebraic representation of a cipher to recover sensitive information. They can apply to a wide range of primitives, such as permutations, hash functions or encryption algorithms. The attacks can be divided in several families of attack, though they might need a slight adaptation depending on the application. In this section, we present the different classes of algebraic attacks. Some of these attacks use advanced algorithms for polynomial manipulation, which we desribe in a first place.

Advanced algorithms for univariate polynomial manipulation. Let P, Q be two polynomials of degree d over a field $\mathbb{F}$. Naïvely, the element-wise multiplication $P \times Q$ costs d^2 multiplications over $\mathbb{F}$. However, faster algorithms exist for polynomial multiplication, using Discrete Fast Fourrier Transform (DFT) [NN82].

Proposition 6.1 ([CK91])**.** *Two polynomials of degree* d *over a field* $\mathbb{F}$ *can be multiplied with* $\mathcal{O}(d\log(d))$ *multiplications and* $\mathcal{O}(d\log(d)\log(\log(d)))$ *additions in* $\mathbb{F}$.

Table 6.1: Sets of challenges proposed by the Ethereum Foundation.

Cipher	Parameters	Security ($\log_2$)[1]	Complexity of our attack ($\log_2$)	Solved in practice
Rescue–Prime [AAB+20]	$N = 4, m = 3$	38	43	✓
	$N = 6, m = 2$	38	53	
	$N = 7, m = 2$	44	62	
	$N = 5, m = 3$	45	57	
	$N = 8, m = 2$	50	72	
Feistel–MiMC [AGR+16] (old challenges)	$r = 6$	18	19	✓
	$r = 10$	30	26	✓
	$r = 14$	44	33	✓
	$r = 18$	56	40	✓
	$r = 22$	68	47	
Feistel–MiMC [AGR+16] (new challenges)[2]	$r = 22$	36	47	
	$r = 25$	40	52	
	$r = 30$	48	60	
	$r = 35$	56	69	
	$r = 40$	64	77	
Poseidon [GKR+21]	$R_P = 3$	45	26	✓
	$R_P = 8$	53	35	✓
	$R_P = 13$	61	44	✓
	$R_P = 19$	69	54	
	$R_P = 24$	77	62	
`Reinforced Concrete` [GKL+22]	$\log_2(p) \approx 48$	48	-	
	$\log_2(p) \approx 56$	56	-	
	$\log_2(p) \approx 64$	64	-	

[1] As expected by the Ethereum Foundation.

[2] On 23/11/2021, the Ethereum Fundation reevaluated the security of Feistel–MiMC and proposed a new set of challenges for Feistel–MiMC to replace the old one, after we sent the solutions to the first three old challenges.

The fast polynomial multiplications can even be speed up to $\mathcal{O}(d\log(d))$ operations if a primitive root of unity is known in $\mathbb{F}$, as discussed in [CK91]. Many other polynomial operations achieve quasi-linear complexities with the same techniques. They are all based on fast polynomial multiplication, which is why a factor $\log(\log(d))$ is added to the complexity, in the case where no primitive root of unity is known in $\mathbb{F}$.

Proposition 6.2 ([Str75])**.** *The Euclidian division of two polynomials of degree d over a field $\mathbb{F}$ can be performed in $\mathcal{O}(d\log(d)\log(\log(d)))$ operations.*

Proposition 6.3 ([Moe73, Corollary 2])**.** *The Greatest Common Divisor (GCD) of two polynomials of degree d over a field $\mathbb{F}$ can be computed in $\mathcal{O}(d\log(d)^2\log(\log(d)))$ operations.*

Proposition 6.4 ([BM74])**.** *The coefficients of a polynomial of degree d over a field $\mathbb{F}$ can be interpolated with $O(d\log(d)^2\log(\log(d)))$ operations.*

These results will be useful for complexity estimations in some algebraic attacks. Let us present the main types of algebraic attacks.

6.2.1 Interpolation Attacks

The traditional interpolation attack [JK97] is an attack against block ciphers that represents some public data (e.g. ciphertext...) as a polynomial Q_K, of degree d, of public data (e.g plaintext...), where the coefficients of Q_K depend on a key K. For instance, in the arithmetization-oriented world over a single element of $\mathbb{F}_q$:

$$C = Q_K(P) = \sum_{i=0}^{d} Q_i(K) \times P^i.$$

Lagrange Interpolation. The Lagrange interpolation formula allows to recover the polynomial Q when $d+1$ distinct values of inputs/outputs P_i, C_i $(0 \leqslant i \leqslant d)$ are known:

$$Q_K(P) = \sum_{i=0}^{i=d} \left(C_i \prod_{\substack{j=0 \\ j \neq i}}^{j=d} \frac{P - P_j}{P_i - P_j} \right).$$

If the attacker choses well the plaintexts P_i, he can use the fast interpolation algorithm of Proposition 6.4 to recover the data coefficients of Q_K in $\mathcal{O}(d \log(d)^2 \log(\log(d)))$ operations.

If d is low enough, the attacker collects $d+1$ data, performs the Lagrange interpolation and recovers all coefficients $Q_i(K)$. This allows the attacker to exactly know the mapping $Q_K : P \to C$. In some cases, this also allows to recover the key.

Variants of the interpolation attack. The interpolation attack finds other applications with nonce-based keystream-cipher. Let us take the example of an arithmetization-oriented stream cipher on $\mathbb{F}_q$ and suppose that we can write a keystream element S_0 as a polynomial on the nonce N:

$$S_0 = Q_K(N).$$

If the degree d of Q_K is low enough, the attacker can recover its coefficients with $d+1$ data and a Lagrange interpolation.

In all cases, the interpolation requires a relatively high data complexity.

6.2.2 Cube Attacks

The cube attack is an attack that affects low-degree traditional keyed ciphers operating on $\mathbb{F}_2$, presented by Dinur and Shamir at EUROCRYPT 2009 [DS09]. Since elements of in $\mathbb{F}_q$ with $q = 2^n$ can be represented as n Boolean variables, the cube attack is a threat against arithmetization-oriented ciphers, as highlighted by

the work of Eichlseder *et al.* [EGL+20]. The cube attack consists in an offline step, followed by an online step. Let us denote public variables (IV bits, nonce bits, plaintext bits...), as elements of $\mathbb{F}_2$,can $x_0 \dots x_{n-1}$, and private variables (key bits, subkey bits) $y_0 \dots y_{m-1}$. Let us consider an output bit (ciphertext bit, keystream bit...); it can be expressed as a function Q of the variables: $Q = Q(x, y)$ with $x = (x_0 \dots x_{n-1})$ and $y = (y_0 \dots y_{m-1})$.

Offline phase. Select a set $I \subseteq \{0 \dots n-1\}$. Q can be uniquely decomposed as

$$Q(x, y) = t_I f_I(x, y) + r(x, y),$$

where $t_I = \Pi_{i \in I} x_i$, and where for all monomials m in $r(x, y)$, $t_I \nmid m$. In the offline phase, the attacker computes formally $f_I(x, y)$ as a function of $x_i, i \notin I$, and y. For the cube attack to work, the set I should be chosen so that the polynomial f_I is sparse and easy to compute.

Online phase. Generate the data corresponding to a $|I|$-dimensional cube C_I: fix all variables x_i for $i \notin I$, and loop through all $2^{|I|}$ values of x_i for $i \in I$. Then, sum all the output values of Q, and the following holds:

$$\sum_{x \in C_I} Q(x, y) = f_I(x, y).$$

The attacker gains the knowledge of $f_I(x, y)$, and may repeat the online phase with other output bits or other cubes C_I (by fixing x_i, $i \notin I$ to different values). Each step gives information on the private variables y. The attacker then finalizes by combining the pieces of information to recover y.

Similarly to interpolation attacks, the cube attack requires a large amount of data.

6.2.3 The GCD Attack

The GCD attack is an attack using the low degree representation of a cipher presented in the security analysis of MiMC [AGR+16], a cipher $E_K : \mathbb{F}_q \mapsto \mathbb{F}_q$ indexed on a key $K \in \mathbb{F}_q$. Let us suppose that the ciphertext C of the cipher can be represented as a polynomial of the plaintext P and of the key K:

$$C = Q(P, K).$$

The attacker generates two plaintexts P_0, P_1 and asks for $C_0 = Q(P_0, K)$ and $C_1 = Q(P_1, K)$. We should note that the value K is a root of both polynomials $T_i(X) = Q(P_i, X) - C_i$ $(i = 0, 1)$. The GCD attack consists in computing the GCD of $T_0(X)$ and $T_1(X)$. This GCD is usually of very small degree and it is very cheap to compute its roots. Using fast polynomial multiplication of Proposition 6.3, this attack costs $\mathcal{O}(d \log^2(d) \log(\log(d)))$ field operations, where d is the degree of Q in K.

Remark. In this case, the attacker can also use the univariate root finding on the polynomial $Q(P_0, K) - C_0$, presented in Section 6.4.1, which costs $\mathcal{O}\big(d\log(d)\big(\log(d) + \log(q)\big)\log(\log(d))\big)$. The GCD attack is cheaper than the univariate root finding, but only by a logarithmic factor $\log(q)/\log(d)$ when q is sufficiently larger than d.

6.2.4 Polynomial Solving Attacks

Polynomial solving attacks is a type of algebraic attack against ciphers which are very efficient against arithmetization-oriented ciphers. The attack against an arithmetization-oriented cipher on $\mathbb{F}_q$ consists in two steps:

- **Modeling:** the attacker represents the cipher as a system $\mathcal{P}$ of m polynomial equations over the ring $R = \mathbb{F}_q[x_0, \ldots, x_{n-1}]$:

$$\mathcal{P} = \begin{cases} p_0(x_0, \ldots x_{n-1}) = 0 \\ p_1(x_0, \ldots x_{n-1}) = 0 \\ \qquad\qquad\qquad\vdots \\ p_{m-1}(x_0, \ldots x_{n-1}) = 0. \end{cases}$$

  $\mathcal{P}$ should have the following properties:

  1. $\mathcal{P}$ has a finite number of solution in the algebraic closure of $\mathbb{F}_q^n$.
  2. $\mathcal{P}$ has a low number of solution in $\mathbb{F}_q^n$.
  3. There exists a solution of $\mathcal{P}$ which directly leads to a secret or unwanted property of the cipher: e.g. the key, a subkey, a CICO solution, or a preimage of a given hash.

- **Polynomial solving:** the attacker solves $\mathcal{P}$ to recover the solutions in the base field $\mathbb{F}_q$. In some cases, only one solution is required, such as a solution to the CICO problem or a preimage. Several techniques exist to solve the system. Depending on the chosen method, the complexity of this step might vary drastically. A common way to solve the system is to use Gröbner basis algorithms [Buc76]. This step is detailed in Section 6.4.

In this thesis, we mount multiple polynomial attacks against several AO ciphers. We mainly consider the case where the polynomial system is *well-defined*, i.e. $m = n$. Note that the property 2. of the polynomial system is hard to verify beforehand, but in practice the property has always been verified in concrete implementations.

Overall, the univariate solving tends to be much more efficient. However, it cannot be applied to all algorithms as there are efficient methods to prevent its applicability, as was done by the designers of Rescue–Prime.

Before diving into an in-depth analysis of the polynomial solving step, we need to introduce a few notions in algebraic geometry.

6.3 Background in Algebraic Geometry

In this section, we present relevant knowledge on algebraic geometry, required for this chapter and for Chapter 7. This section may be hard to read on its own; its purpose is principally to serve as reference to later results. However, the notions presented are very common in algebraic geometry, and many introductions to the subject exist in the literature. We refer readers interested in a further analysis of these notions to [CLO97].

Throughout this chapter and Chapter 7, we denote the base field by $\mathbb{F}_q$. We also use the notions of polynomials and polynomial mappings interchangeably. We study properties of the multivariate polynomial ring $R = \mathbb{F}_q[x_0, \ldots, x_{n-1}]$, its polynomial systems and ideals.

Let us denote $\mathcal{P} = \{p_0(x_0, \ldots x_{n-1}), \ldots p_{n-1}(x_0, \ldots x_{n-1})\}$ a set of polynomials of R, of which we want to find the common zeros. We denote d_i the degree p_i.

Definition 6.1. A set of polynomials

$$\mathcal{S} = \{p_0(x_0, \ldots x_{n-1}), \ldots p_{k-1}(x_0, \ldots x_{n-1})\}$$

defines an ideal $I = \langle \mathcal{S} \rangle \subseteq R$ s.t. $p \in I$ if there exists $(z_0, \ldots z_{k-1}) \in R^k$ s.t.

$$p = \sum_{i=0}^{k-1} z_i p_i.$$

In particular, we denote $I = \langle \mathcal{P} \rangle$ the ideal defined by our polynomial system $\mathcal{P}$.

In order to study this ideal, we need the notion of *monomial order*, for which we first define *monomials*.

Definition 6.2. A *monomial* is a polynomial of R of the form $\prod_{i=0}^{n-1} x_i^{\alpha_i}$ for some $(\alpha_0 \ldots \alpha_{n-1}) \in \mathbb{N}^n$.

In particular, any polynomial of R can be expressed as a weighted sum of monomials.

Definition 6.3. A *monomial order* $\prec$ is a total order on the set of monomials of R such that *i*) for any monomial $m \in R$ we have $1 \prec m$; and *ii*) for any three monomials $m_1, m_2, t \in R$ we have

$$m_1 \prec m_2 \implies t \cdot m_1 \prec t \cdot m_2 \ .$$

We now define three very common monomial orders.

Definition 6.4. The *lexicographical (lex)* order with $x_0 > x_1 \cdots > x_{n-1}$ is defined by $m_0 = x_0^{\alpha_0} \ldots x_{n-1}^{\alpha_{n-1}} \prec_{lex} m_1 = x_0^{\beta_0} \ldots x_{n-1}^{\beta_{n-1}}$ if the first non-zero element in $(\beta_0 - \alpha_0, \ldots, \beta_{n-1} - \alpha_{n-1})$ is positive.

Definition 6.5. The *graded lexicographical (deglex)* order with $x_0 > x_1 \cdots > x_{n-1}$ is defined by $m_0 = x_0^{\alpha_0} \ldots x_{n-1}^{\alpha_{n-1}} \prec_{deglex} m_1 = x_0^{\beta_0} \ldots x_{n-1}^{\beta_{n-1}}$ if $\sum \alpha_i < \sum \beta_i$, or if $\sum \alpha_i = \sum \beta_i$ and the first non-zero term in $(\beta_0 - \alpha_0, \ldots, \beta_{n-1} - \alpha_{n-1})$ is positive.

Definition 6.6. The *graded reverse lexicographical (grevlex)* order with $x_0 > x_1 \cdots > x_{n-1}$ is defined by $m_0 = x_0^{\alpha_0} \dots x_{n-1}^{\alpha_{n-1}} \prec_{grevlex} m_1 = x_0^{\beta_0} \dots x_{n-1}^{\beta_{n-1}}$ if $\sum \alpha_i < \sum \beta_i$, or if $\sum \alpha_i = \sum \beta_i$ and the last non-zero term in $(\beta_0 - \alpha_0, \dots, \beta_{n-1} - \alpha_{n-1})$ is negative.

We now define a *weighted* order which we use throughout Chapter 7.

Definition 6.7. Consider a weight vector $\boldsymbol{w} = (w_0, \dots, w_{n-1}) \in \mathbb{R}^n$, where $w_0 \neq 0$, and $x_0 < x_1 < \ldots < x_{n-1}$. We say that $\boldsymbol{w}$ is associated with the *weighted graded lexicographical (wdeglex)* order $\prec_{\boldsymbol{w}}$, defined by $m_0 = x_0^{\alpha_0} \dots x_{n-1}^{\alpha_{n-1}} \prec_{\boldsymbol{w}} m_1 = x_0^{\beta_0} \dots x_{n-1}^{\beta_{n-1}}$ if

$$\sum_{i=0}^{n-1} w_i \alpha_i < \sum_{i=0}^{n-1} w_i \beta_i,$$

or if

$$\sum_{i=0}^{n-1} w_i \alpha_i = \sum_{i=0}^{n-1} w_i \beta_i,$$

and the last non-zero term in $(\beta_0 - \alpha_0, \dots, \beta_{n-1} - \alpha_{n-1})$ is positive.

Note that we chose the variable ordering $x_0 < x_1 < \ldots < x_{n-1}$ for practical reasons, as this can be directly applied in Chapter 7.

Definition 6.8. The *leading monomial* of a nonzero polynomial $f \in R$, relative to a monomial order $\prec$, is the largest monomial contained in f according to $\prec$. It is denoted $\mathrm{LM}(f)$. The *leading coefficient* of f, $\mathrm{LC}(f)$, is the coefficient associated with $\mathrm{LM}(f)$. Finally, the *leading term* of f, $\mathrm{LT}(f)$, is the product of its leading monomial and coefficient.

If $S = \{p_0, p_1, \ldots\} \subseteq R$, then we can extend the above definitions to the set S, e.g., $LT(S) = \{LT(p_0), LT(p_1), \ldots\}$. We may now define the notion of a Gröbner basis of an ideal of R.

Definition 6.9 (Gröbner basis [Buc76])**.** Let I be an ideal of R. A finite set of polynomials $G \subset I$ is a *Gröbner basis* with respect to $\prec$ if the leading monomial of every polynomial in I is a multiple of the leading monomial of some polynomial in G. A Gröbner basis G is said to be *reduced* if for all $g \in G$, no monomial in g is divisible by an element of $\mathrm{LT}(G) \backslash \{\mathrm{LT}(g)\}$ and $\mathrm{LC}(G) = \{1\}$.

An ideal I always contains a Gröbner basis. For a fixed $\prec$ there are usually many Gröbner bases, but only one reduced Gröbner basis. We crucially rely on the following results throughout Chapter 7.

Proposition 6.5 ([CLO97, Chapter 2, §9, Prop 4 and Thm 3])**.** *Let G be a set of polynomials of R, $G = \{g_1, ..., g_m\}$. If the leading monomials of g_i and g_j are relatively prime for all $1 \leqslant i \neq j \leqslant m$, then G is a Gröbner basis for $\langle G \rangle$.*

Proposition 6.6 ([CLO97, Chapter 2 §6 Prop 1])**.** *Let I be an ideal, $\prec$ a monomial order, G a Gröbner basis of I w.r.t. $\prec$, and $f \in R$. There exists a unique $r \in R$ such that:*

- $\mathrm{LT}(r)$ *is not divisible by any element of* $\mathrm{LT}(G)$.
- $g = f - r \in I$.

The polynomial r is called the remainder *or* normal form *of f w.r.t. I and $\prec$.*

In the following, the Gröbner basis of an ideal I refers to the unique reduced Gröbner basis. The computation of the remainder r of f can be performed in a finite number of steps from G, by iteratively eliminating its leading terms with multiples of elements of G.

The Quotient Ring. We can define the quotient ring R/I, where each class has a unique representative r such that $\mathrm{LT}(r)$ is not divisible by any element of $\mathrm{LT}(I)$. This is known as Macaulay's theorem [Eis13, Theorem 15.3]. The monomial order $\prec$ does not affect the quotient ring R/I, but determines the representative of each class. The quotient ring R/I forms a $\mathbb{F}_q$-vector space

Definition 6.10. Let I be an ideal of R. We say that I is *zero-dimensional* if $\dim_{\mathbb{F}_q}(R/I)$ is finite. In this case, its *ideal degree* D_I is $\dim_{\mathbb{F}_q}(R/I)$.

First, given a zero-dimensional ideal I, it is possible to bound its ideal degree from the degrees of its generating polynomials.

Proposition 6.7. *Let $p_0 \dots p_{n-1}$ be polynomials of R, of degree respectively $d_0 \dots d_{n-1}$. Let $I = \langle p_0 \dots p_{n-1} \rangle$ be the zero-dimensional ideal generated by the polynomials p_i. The Bezout bound states that*

$$D_I \leqslant \prod_{i=0}^{n-1} d_i.$$

The Bezout bound is very useful in cryptanalysis, as it allows to bound the ideal degree of an ideal, and thus to bound the complexities of some Gröbner basis algorithms.

The quotient ring R/I has a canonical basis with respect to $\prec$ denoted $\mathcal{B}_\prec(R/I)$, where the basis elements are given by all the monomials in R that are not in the ideal $\langle \mathrm{LM}(G) \rangle$, for G a Gröbner basis for $(I, \prec)$. If I is zero-dimensional, we have $|\mathcal{B}_\prec(R/I)| = D_I$. Each element r of R/I can then be written as a vector in the basis $\mathcal{B}_\prec(R/I)$, which we call $\mathrm{NormalForm}(r)$. This allows us to define the linear matrix $T_j : R/I \to R/I$ corresponding to the multiplication by x_j.

Definition 6.11. The *multiplication matrix* T_j of x_j relative to a zero-dimensional ideal I, a monomial order $\prec$, and the basis $\mathcal{B}_\prec(R/I) = (\epsilon_1, \dots, \epsilon_{D_I})$ is defined as the square matrix which has each column defined as $C_i = \mathrm{NormalForm}(\epsilon_i \times x_j)$ represented in the basis $\mathcal{B}_\prec(R/I)$.

The following result is well-known in the literature, but we have not been able to find a reference that holds for finite fields (e.g., it is derived as Corollary 4.6 in [CLO98] over $\mathbb{C}$). For completeness, we provide a short proof that works over any field.

Proposition 6.8. *Let I be a zero-dimensional ideal of R, $\prec$ a monomial order, and T_0 the multiplication matrix of the variable x_0 with respect to $\prec$. We have $\det(x_0\boldsymbol{I} - T_0) \in I$, where $\boldsymbol{I}$ is the identity matrix.*

Proof. Let $C(X) = \det(X\boldsymbol{I} - T_0) = \sum_{i=0}^{D_I} c_i X^i$ be the characteristic polynomial of T_0. By the Cayley-Hamilton theorem we have $C(T_0) = \sum_{i=0}^{D_I} c_i T_0^i = \mathbf{0}$, where $\mathbf{0}$ is the $D_I \times D_I$ zero-matrix. Letting ϵ denote the column vector representing the constant polynomial 1 in R/I, we then have $C(T_0)\epsilon = 0$. As $T_0^i\epsilon$ is the representation of NormalForm(x_0^i), this implies that NormalForm$(C(x_0)) = 0$, hence $C(x_0) \in I$. □

We here present an interesting property of the multiplication matrix, due to the algebraic structure of the ring R/I:

Definition 6.12 (Sparsity indicator of a matrix)**.** The *sparsity indicator* t of a square matrix M of dimensions $d \times d$ in a field $\mathbb{F}$ is the average number of non-zero values in each column. It is given with the following formula:

$$t = \frac{1}{d} \sum_{(i,j) \in [\![1,d]\!]^2} \mathbb{1}_{M[i,j] \neq 0}.$$

Property 6.1 ([FM17, Corollary 6.10])**.** *Let $R = \mathbb{F}_q[x_0 \dots x_{n-1}]$, I an ideal generated from a random set of polynomials of degree d, and the ideal degree D_I. Let T_0 be the multiplication matrix w.r.t. x_0 and t its sparsity indicator. As $d \to \infty$, if the Moreno-Socías conjecture [MS91] holds,*

$$t \sim D_I \times \sqrt{\frac{6}{n\pi}}/d.$$

The next two definitions are a standard hypothesis for an ideal when implementing Gröbner basis polynomial solving algorithms.

Definition 6.13 ([BND22], Definition 2.1)**.** An ideal I of R satisfies the *stability property* with respect to x_0, a monomial order $\prec$ and its corresponding Gröbner basis G if for all $m \in \mathrm{LM}(G)$ such that $x_0 | m$, $\frac{mx_i}{x_0} \in \mathrm{LM}(G)$ for all $i \in \{1, \dots, n-1\}$.

Definition 6.14 ([FM17], Definition 3.1)**.** An ideal I of R is in *shape position* if its reduced Gröbner basis in the lexicographical order has the following form:

$$G = \{f_0(x_0),\ x_1 - f_1(x_0),\ \dots,\ x_{n-1} - f_{n-1}(x_0)\}.$$

If the ideal I is in shape position, it follows straightforwardly that $D_I = \deg(f_0)$, and the cost of finding common zeros for I, given its lexicographic Gröbner basis, reduces to the problem of finding the roots of f_0.

The following definition of regularity is an important hypothesis on in the complexity estimations of many Gröbner basis algorithms.

Definition 6.15. Let $\mathcal{P} = (p_0 \dots p_{n-1})$ be a polynomial system over $R = \mathbb{F}_q[x_0, \dots, x_{n-1}]$. $\mathcal{P}$ is said to be *regular* if for all $0 \leqslant i \leqslant n-1$,

$$g p_i \in \langle p_0, \dots, p_{i-1} \rangle \Rightarrow g \in \langle p_0, \dots, p_{i-1} \rangle.$$

The Fröberg conjecture states random polynomial systems [Frö85] are regular, and polynomial systems in cryptography are often considered regular, partly because the study of non-regular systems is difficult and very system-dependant.

Dubois et al. in [DG10, Section 2.2] give a definition of the degree of regularity D_{reg} of I. The definition involves non trivial notions that are not necessary in this thesis, so we simply refer to their definition. Indeed, we only use D_{reg} as a bound for some Gröbner basis algorithm complexity estimations. The following result holds:

Proposition 6.9 (Macaulay bound)**.** *The* Macaulay bound *gives an upper bound to the degree of regularity of a polynomial system. The bound is reached in particular if the system is regular*[1]*:*

$$D_{\text{reg}} \leqslant 1 + \sum_{i=1}^{n} (d_i - 1).$$

6.4 Solving Polynomial Systems

The notions presented in Section 6.3 allow us to go over the state-of-the-art algorithms of polynomial system solving. Two types of polynomial systems must be treated separately: univariate and multivariate systems.

6.4.1 Solving Univariate Systems

In the univariate case, we have a system with a single equation and a single variable:

$$P(x) = 0.$$

Solving the system is equivalent to finding the roots of the polynomial $P \in \mathbb{F}_q[x]$. We denote $d (= D_I)$ the degree of P.

Using the following method, finding the roots requires only

$$\mathcal{O}\big(d \log(d) \big(\log(d) + \log(q) \big) \log(\log(d)) \big)$$

field operations:

[1] Note that the upper bound does not require the system to be regular.

1. Compute $Q = x^q - x \bmod P$.
   The computation is performed with a double-and-add algortihm. We multiply two polynomials of degree d with $\mathcal{O}(d\log(d)\log(\log(d)))$ field operations (Proposition 6.1), and compute the remainder of a polynomial of degree $2d$ by a polynomial of degree d in $\mathcal{O}(d\log(d)\log(\log(d)))$ field operations (Proposition 6.2). In total, this step costs $\mathcal{O}(d\log(q)\log(d)\log(\log(d)))$ field operations.

2. Compute $R = \gcd(P, Q)$.
   R has the same roots as P in the field $\mathbb{F}_q$ since $R = \gcd(P, x^q - x)$, but its degree is much lower (it is exactly the number of roots in $\mathbb{F}_q$).
   This requires $\mathcal{O}(d\log^2(d)\log(\log(d)))$ field operations (Proposition 6.3).

3. Factor R.
   In general, R is of very low degree because P has few roots in the field, and this step is of negligible complexity.

Note that finding the roots inside the field is significantly easier than factoring the polynomial. In particular, one of the "six worlds of Gröbner basis cryptanalysis" of Koschatko *et al.* [KLR24], i.e. the theoretical complexity of the univariate solving, can be improved by considering the complexity computed in this section rather than the complexity of polynomial factoring.

For practical instances, we use the NTL library [Sho], a C++ library for number theory, and the computation is feasible for a degree up to roughly $2^{32}(\approx 3^{20})$ in a prime field $\mathbb{F}_p$ with $p \approx 2^{64}$. We present some benchmarks in Section 6.5.5 for polynomials given by round-reduced versions of Feistel–MiMC, of Poseidon and for random polynomials.

6.4.2 Solving Multivariate Systems

In the multivariate case, we only consider well-defined systems, i.e. systems with as many variables as equations; the system to solve is composed of n polynomials in $\mathbb{F}_q[x_1, \ldots, x_n]$. The complexity of solving this type of system largely depends on the underlying structure. In some cases, some variables x_i can be substituted in the system, and the solving can be reduced to the problem of solving a system with fewer equations and fewer variables. A thorough analysis of specific non regular systems is very complex for the designers, therefore a common practice is to estimate the complexity of the Gröbner basis attack for an equivalent regular system, and to add extra rounds to account for the non-regularity of the system. The regularity of a system is an expected property of random systems, under the Fröberg conjecture [Frö85], but it should be noted that many systems based on cryptographic problems are not regular, as discussed below.

Let us denote $\mathcal{P} = \{p_0 \ldots p_{n-1}\}$ the multivariate system to solve, d_i the degree of the polynomial p_i, D_I the degree of the ideal $\mathcal{I} = \langle p_1, \ldots, p_n \rangle$, and D_{reg} its degree of regularity.

The goal of the Gröbner basis attack is to recover the solutions of the sytem, and this can be done by computing its *lex* Gröbner basis. If the ideal is zero-dimensional, the first element of the *lex* Gröbner basis is univariate in the first variable, and its roots can be recovered with univariate solving. In practice, directly computing the Gröbner basis in *lex* order is prohibitively expensive. Instead, the traditional Gröbner basis attack consists in the following three steps:

1. First compute the Gröbner basis in the *grevlex* order. The current fastest algorithm for this step is F5 [Fau02], with a complexity of $\mathcal{O}\left(\binom{n+D_{reg}}{D_{reg}}\right)^{\omega}$ [BFS04], where $2 \leqslant \omega \leqslant 2.38$ is the coefficient of fast linear algebra. This bound is however not tight.

2. Apply a change of order algorithm to produce a *lex* Gröbner basis from the *grevlex* Gröbner basis. Different algorithms exist, with different complexity formulas. FGLM was first designed in 1993 [FGL+93], with a complexity in $\mathcal{O}(nD_I^3)$ under no assumption. It was then improved to $\mathcal{O}(D_I^\omega)$ [FGH+14a; NS20] and to $\mathcal{O}(tD_I^2)$ [FM17], where t is the sparsity indicator of the multiplication matrix T_1, under the shape and stability assumptions. Variables can of course be reordered to select the sparsest matrix T_i. Recently, Berthomieu *et al.* improved the change of order algorithm using Hermite Normal Forms (HNF) [BND22], under the shape and stability assumptions, lowering the complexity to $\tilde{\mathcal{O}}(t^{\omega-1}D_I)$. A comparison of the strengths and limits of these algorithms is given in Section 7.1.2. In this section, most systems do not possess the stability property, therefore we only use the complexity $\mathcal{O}(nD_I^3)$ for our attacks, unlike what was performed in our original paper [BBL+22].

3. The *lex* Gröbner basis is of the form:

$$\left\{f_0(x_0), f_{1,0}(x_0,x_1), \dots f_{1,k_1}(x_0,x_1), \dots f_{n-1,k_{n-1}}(x_0,\dots,x_{n-1})\right\} .$$

   Solutions for x_0 can be found by applying an efficient univariate root finding algorithm to the polynomial f_0 (of degree at most D_I), in $\tilde{\mathcal{O}}(D_I)$. Then, we can efficiently recover the values of $x_1, \dots, x_n$ by progressively substituting the x_i and solving univariate equations. The complexity of this step is negligible compared to the first two.

Theoretical complexity. For random systems, the Macaulay and Bezout bounds are reached, and the complexity bounds are tight. In this case, the complexity of the first step ($F5$) dominates the complexity of the second step (Fast-FGLM or HNF). For instance, when the polynomials have the same degrees $d_i = \delta$, we have the following: $D_I = \delta^n$, $D_{\text{reg}} = n \times \delta - n + 1$. This implies

$$\binom{n + D_{\text{reg}}}{D_{\text{reg}}} = \binom{n\delta + 1}{n} = \frac{n\delta+1}{n} \cdot \frac{n\delta}{n-1} \cdot \frac{n\delta-1}{n-2} \cdot \dots \cdot \frac{n\delta-n+2}{1} \geqslant \delta^n .$$

Which in turn implies:

$$\binom{n + D_{\text{reg}}}{D_{\text{reg}}}^{\omega} \geqslant D_I^{\omega} \geqslant t^{\omega-1} D_I.$$

Remarks. In practice however, the systems corresponding to symmetric ciphers are often not regular. The non-regularity affects most strongly the degree of regularity, but the ideal degree can also be smaller than expected, as demonstrated by the implementation of Ciminion multivariate modeling of Section 6.6.3 and by the modeling of `Anemoi` [BBC+23]. In the worst case scenario, Gröbner basis can be directly computed from the modeling of the cipher. For instance, the Flurry and Curry ciphers [BPW06b] were designed to possess such a property. This property was also discovered on AES [BPW06a] and later on MiMC [ACG+19]. In Chapter 7, we show that a Gröbner basis under weighted monomial orders can be freely expressed from the algebraic representations of several ciphers, such as `Anemoi` [BBC+23], Arion [RST23], Griffin [GHR+23], and XHash8 [AKM23].

6.5 CICO Cryptanalysis of some AO Hash Functions

In this section, we report our theoretical and experimental results in solving the CICO problem for the challenges put forward by the Ethereum Foundation. This section puts forward some of the results obtained in the work [BBL+22]. The theoretical results against Feistel–MiMC are presented in Section 6.5.1. Section 6.5.2 presents a technique to bypass the first two rounds of SPN ciphers (or steps in the sense of Rescue–Prime), while Section 6.5.3 and Section 6.5.4 apply this technique on respectively Poseidon and Rescue–Prime. The corresponding experimental results are detailed in Section 6.5.5.

These ciphers all operate on $\mathbb{F}_p^m$ with p prime and $m \geqslant 2$. The complexities of some of our attacks are in line with the designers' claims, although there were wrongful claims from the Ethereum Fondation. However, we have found that, in practice, the exact claimed security level for a given number of rounds is not always clear. Furthermore, breaking challenges in practice gives a more concrete understanding of the security of reduced versions (we focus on upper bounds rather than lower bounds). Besides, in the designer's analysis some optimistic complexity assumptions are made (eg. ignoring log factors, or taking a small omega) but in this chapter we propose a more accurate analysis.

6.5.1 Attacks Against Round-Reduced Feistel–MiMC

Design description. Feistel–MiMC is a Feistel network, based on the simple structure of MiMC, introduced by Albretch *et al.* at Asiacrypt 2016 [AGR+16]. It operates on $\mathbb{F}_p^2$ ($m = 2$) using a basic r-round Feistel structure with the i-th round

function being $x \mapsto (x + c_i)^\alpha$ (where in this chapter, we take $\alpha = 3$, as fixed by the author of the Ethereum's challenges).

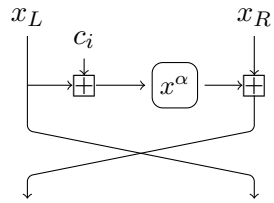


Figure 6.1: Round i of Feistel–MiMC.

Attack description. In order to build a polynomial system representing the CICO problem, we consider an input state $(p_0, q_0) = (x, 0)$, with x the indeterminate of a polynomial ring. Then we evaluate the round function iteratively, as polynomials in $\mathbb{F}_p[x]$:

$$\begin{aligned} p_0 &= x, & q_0 &= 0, \\ p_i &= q_{i-1} + (p_{i-1} + c_i)^3, & q_i &= p_{i-1}\ . \end{aligned}$$

The CICO problem becomes $q_r = 0$: we just have to find the roots of $q_r = p_{r-1}$.

In practice, we use `SageMath` to generate the polynomial, and we compute the roots either directly from `SageMath`, or with an external program using NTL.

Complexity Analysis. Since the round function has degree 3, we obtain a univariate polynomial p_{r-1} of degree $d = 3^{r-1}$ after $r - 1$ rounds. We can estimate the complexity of finding the roots as:

$$d \log(d)\big(\log(d) + \log(p)\big) \log(\log(d)) \ \approx\ 3^{r-1} \times (r-1) \times 1.58 \times 64 \times \log_2(r-1).$$

We give explicit values for the proposed challenges in Table 6.2. Parameters have changed while we were working on it, so "old" (Table 6.2a) and "new parameters" (Table 6.2b) are two sets of parameters proposed by the Ethereum Fondation, the first ones being less secure than the latter ones.

We observe that the initial security claims from the Ethereum Foundation are close to 3^{2r}. This likely corresponds to an estimation of the complexity of a Gröbner base attack using r equations of degree 3 in r variables: the corresponding complexity would be $3^{\omega r} \geqslant 3^{2r}$.

Besides, the original specification of Feistel–MiMC states that Lagrange interpolation attacks are expected to have a complexity of $r \cdot 3^{2r-3}$, while GCDs attacks are expected to have a complexity of $r^2 \cdot 3^{r/2-3}$. As the GCD attack only applies for keyed variants of Feistel–MiMC, it does not apply in our context. However, we have chosen to put both in Table 6.2 for a fairer comparison.

r	Authors claims *Lagrange*	Authors claims *GCD*	Ethereum claims	d	complexity
6	2^{16}	2^{5}	2^{18}	3^{5}	$\mathbf{2^{19}}$
10	2^{30}	2^{9}	2^{30}	3^{9}	$\mathbf{2^{26}}$
14	2^{43}	2^{13}	2^{44}	3^{13}	$\mathbf{2^{33}}$
18	2^{56}	2^{17}	2^{56}	3^{17}	$\mathbf{2^{40}}$
22	2^{69}	2^{21}	2^{68}	3^{21}	2^{47}

(a) **Old parameters**.

r	Authors claims *Lagrange*	Authors claims *GCD*	Ethereum claims	d	complexity
22	2^{69}	2^{21}	2^{36}	3^{21}	2^{47}
25	2^{79}	2^{24}	2^{40}	3^{24}	2^{52}
30	2^{95}	2^{28}	2^{48}	3^{29}	2^{60}
35	2^{111}	2^{33}	2^{56}	3^{34}	2^{69}
40	2^{127}	2^{37}	2^{64}	3^{39}	2^{77}

(b) **New parameters**.

Table 6.2: Complexity of our attack against Feistel–MiMC, compared with the security claims given by the authors and by the challenges. Complexity figures in bold correspond to attacks that we have implemented in practice.

6.5.2 Bypassing SPN Steps

Let $\pi = \pi_0 \circ \pi_1$ be a permutation of $\mathbb{F}_p^m$, and $\mathcal{Z}$ be the vector space $\mathbb{F}_p^{m-1} \times \{0\}$. Suppose that there exists two vectors V and G in $\mathbb{F}_p^m$ such that

$$\pi_0^{-1}(xV + G) \in \mathcal{Z}$$

for all $x \in \mathbb{F}_p$. In this case, we write all the intermediate variables of π_1 as polynomials in x, starting from the state $xV + G$, and evaluating round operations one by one as polynomials. Then we can find r such that $\pi_1(rV + G) \in \mathcal{Z}$ by finding a root r of the polynomial corresponding to the last coordinate of the output. Finally, setting $y = \pi_0^{-1}(rV + G)$ yields a solution to the CICO problem, while the solver has to handle a polynomial based on π_1 rather than the full π. This approach is summarized in Figure 6.2, and we used it against both Poseidon (see Section 6.5.3) and Rescue–Prime (see Section 6.5.4).

Let us describe this trick in more detail. First, for the sake of consistency, we use steps when referring to the constant addition, the S-box, and the linear part. Then one round of Poseidon consists of one step, and one round of Rescue–Prime of two steps: one using S as S-box, the other using S^{-1}.

We consider π_0 to be two steps of an SPN construction without the final linear layer: addition of rounds constants, S-box layer S_1, linear layer consisting of a multiplication by an MDS matrix, and S-box layer S_2. We require the S-boxes to

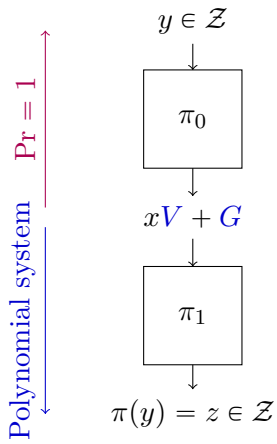


Figure 6.2: A 2-staged trick.

be monomial functions, so that $S(Ax) = S(A)S(x)$. The question of whether the attack can be adapted to the case where the condition is not verified is an open problem.

We use c_i^r to denote the i-th round constant used in step r. We let the linear layer M be such that:

$$M^{-1} = \begin{bmatrix} \alpha_{0,0} & \alpha_{1,0} & \dots & \alpha_{m-1,0} \\ \alpha_{0,1} & \alpha_{1,1} & \dots & \alpha_{m-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{0,m-1} & \alpha_{1,m-1} & \dots & \alpha_{m-1,m-1} \end{bmatrix}.$$

Case $m = 3$. We start with the special case $m = 3$, and we denote the state after π_0 as (x, y, z), with three variables. As seen in Figure 6.3, we have $\pi_0^{-1}(x, y, z) \in \mathcal{Z}$ if and only if

$$\begin{aligned} S_1(c_2^0) &= \alpha_{0,2}(S_2^{-1}(x) - c_0^1) + \alpha_{1,2}(S_2^{-1}(y) - c_1^1) + \alpha_{2,2}(S_2^{-1}(z) - c_2^1) \\ &= \alpha_{0,2}S_2^{-1}(x) + \alpha_{1,2}S_2^{-1}(y) + \alpha_{2,2}S_2^{-1}(z) - (\alpha_{0,2}c_0^1 + \alpha_{1,2}c_1^1 + \alpha_{2,2}c_2^1). \end{aligned}$$

In order to simplify the equation, we fix z to a constant value g with: $g = S_2(\alpha_{2,2}^{-1}(\alpha_{0,2}c_0^1 + \alpha_{1,2}c_1^1 + \alpha_{2,2}c_2^1 + S_1(c_2^0)))$. We obtain:

$$\begin{aligned} \pi_0^{-1}(x, y, g) \in \mathcal{Z} &\iff \alpha_{0,2}(S_2^{-1}(x)) = -\alpha_{1,2}(S_2^{-1}(y)) \\ &\iff S_2(\alpha_{0,2})x = S_2(-\alpha_{1,2})y. \end{aligned}$$

Therefore, we obtain an affine space with $\pi_1(xV + G) \in \mathcal{Z}$ by choosing:

$$V = (1,\ S_2(\alpha_{0,2})/S_2(-\alpha_{1,2}),\ 0) \quad \text{and} \quad G = (0, 0, g)\ .$$

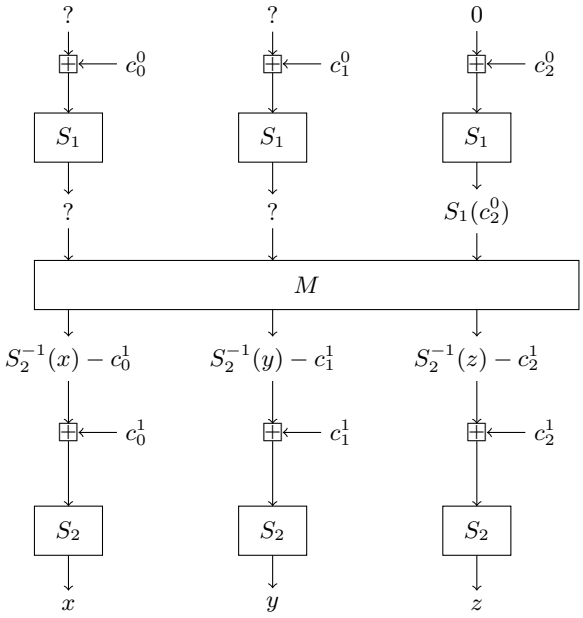


Figure 6.3: Bypassing Two SPN Steps ($m = 3$).

General case ($m \geqslant 3$). In general, we take $V = (S_2(A_0), \ldots, S_2(A_{t-2}), 0)$ and $G = (0, \ldots, 0, g)$, such that we can consider an input state after the S-box layer of the second step of the form $(S_2(A_0)x, \ldots, S_2(A_{t-2})x, g)$, and study the first two steps as shown in Figure 6.4.

Following Figure 6.4, the value $S_1(c_{m-1}^0)$ must satisfy

$$\begin{aligned} S_1(c_{m-1}^0) &= \sum_{j=0}^{m-2} \alpha_{j,2}(A_j S_2^{-1}(x) - c_j^1) + \alpha_{m-1,2}(S_2^{-1}(g) - c_{m-1}^1) \\ &= S_2^{-1}(x) \left(\sum_{j=0}^{m-2} \alpha_{j,2} A_j \right) + \alpha_{m-1,2} S_2^{-1}(g) - \sum_{j=0}^{m-1} \alpha_{j,2} c_j^1 \ . \end{aligned}$$

It is the case provided for instance that:

$$\begin{cases} A_{t-2} &= -\sum_{j=0}^{m-3} \frac{\alpha_{j,2}}{\alpha_{s-2,2}} A_j \\ g &= S_2 \left(\frac{1}{\alpha_{m-1,2}} \sum_{j=0}^{m-1} \alpha_{j,2} c_j^1 + S_1(c_{m-1}^0) \right) \ . \end{cases} \tag{6.1}$$

Thus, if we find a value x such that the image of $(S_2(A_0)x, \ldots, S_2(A_{t-2})x, g)$ through $R-2$ steps of the primitive is equal to $(*, \ldots, *, 0)$, then we are always able to deduce an input $(x_0, x_1, \ldots, x_{m-2}, 0)$ for R steps of the primitive that is mapped to $\mathcal{Z}$.

6.5.3 Application to Round-Reduced Poseidon

Design description. Poseidon [GKR+21] is a family of hash functions, based on the HADES design strategy [GLR+20]. The internal permutation is composed of

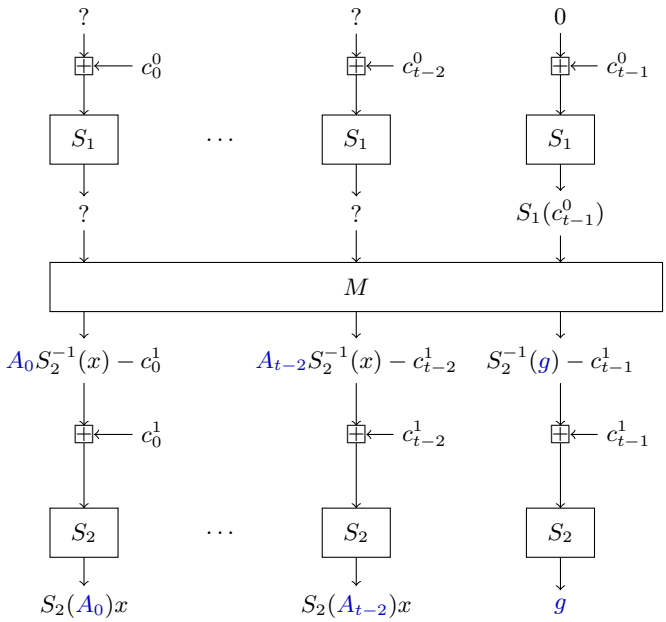


Figure 6.4: Bypassing Two SPN Steps (general case).

$r = \mathsf{RF} + \mathsf{RP}$ rounds of two different types: full rounds have m S-box functions, and partial rounds have only 1 S-box and $m-1$ identity functions. Each round function consists of adding the round constants[2], applying partial or full S-box layers S, and then multiplying the state by an MDS matrix (M). The permutation starts with $\mathsf{Rf} = \mathsf{RF}/2$ full rounds, followed by RP partial rounds, and finally $\mathsf{Rf} = \mathsf{RF}/2$ full rounds.

The challenges from the Ethereum Foundation use $m = 3$, the S-box is $x \mapsto x^3$ and $\mathsf{RF} = 8$ is fixed, while RP varies according to the security level required.

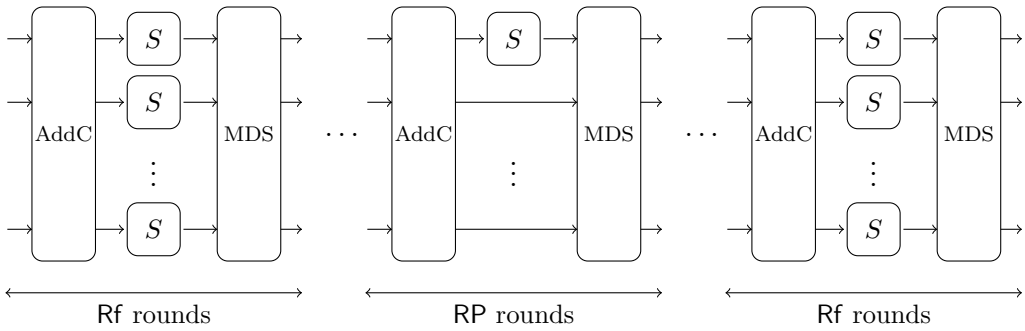


Figure 6.5: Overview of the construction of Poseidon.

[2]For the sake of consistency of the different hash functions presented in this chapter, we note the addition of constants: "AddC".

Attack description. A basic encoding of Poseidon into equations can be solved quickly for a small number of rounds. In fact, it was sufficient for us to be able to claim the first bounty offered by the Ethereum Foundation for this algorithm. However, we needed to use the technique described in Section 6.5.2 for subsequent challenges. The idea is to decrease the degree and the complexity of the polynomial system by more carefully choosing its variables.

Let $m = 3$, and S_1, S_2 such that $S_1(x) = S_2(x) = x^3$. Then, applying our trick for SPN rounds, we consider an input state after the S-box layer of the second round of the form $({A_0}^3 x, {A_1}^3 x, g)$ (*i.e.* we use $V = ({A_0}^3, {A_1}^3, 0)$ and $G = (0, 0, g)$). We obtain

$$\begin{cases} A_1 &= -\frac{\alpha_{0,2}}{\alpha_{1,2}} A_0 \\ g &= \left(\frac{1}{\alpha_{2,2}} \left(\alpha_{0,2} c_0^1 + \alpha_{1,2} c_1^1 \right) + c_2^1 + (c_2^0)^3 \right)^3 . \end{cases} \tag{6.2}$$

Therefore, we evaluate the permutation as polynomials in $\mathbb{F}_p[x]$ starting from the state $({A_0}^3 x, {A_1}^3 x, g)$ with A_0, A_1, g satisfying Equation 6.2, and the CICO problem is equivalent to finding the root of the polynomial corresponding to the rightmost branch of the output.

In practice, we use `SageMath` to generate the polynomial, and we compute the roots either directly from `SageMath`, or with an external program using NTL.

Complexity Analysis. Poseidon has $r = \mathsf{RF} + \mathsf{RP}$ rounds in total, but we skip the first two rounds using the trick. Therefore, we obtain a univariate polynomial of degree $d = 3^{r-2}$, and we can estimate the complexity of finding the roots as:

$$d \log(d) \big(\log(d) + \log(p) \big) \log(\log(d))\big) \approx 3^{r-2} \times (r-2) \times 1.58 \times 64 \times \log_2(r-2).$$

We give explicit values for the proposed challenges in Table 6.3, along with the corresponding security claims. For the challenges issued by the Ethereum foundation, the claim was that an attack would require at least 2^{37+s} steps, where s is a "security" level specified in bits, and is equal to $8, 16, 34, 32$ and 40 when RP is equal to $3, 8, 13, 19$ and 24 respectively.

The original specification of Poseidon states that interpolation attacks are expected to have a complexity similar to the one of our attacks, namely about $\alpha^{\mathsf{RP}+\mathsf{RF}}$ [GKR+21, Equation (3)]. However, the challenges of the Ethereum Foundation[3] appear to claim a higher security level.

6.5.4 Application to Round-Reduced Rescue–Prime

Design description. Rescue is a family of AO hash functions, that was first proposed as part of Marvellous designs [AAB+19]. Rescue has the particularity of using both a low degree S-box and its inverse. Indeed, each round of Rescue, consists of two steps: while the first one involves an S-box S, an MDS matrix M,

[3] We observe that this claim is close to $3^{3\mathsf{RF}+\mathsf{RP}}$, but it is unclear which attack it corresponds to.

RP	Authors claims	Ethereum claims	d	Complexity
3	2^{17}	2^{45}	3^9	$\mathbf{2^{26}}$
8	2^{25}	2^{53}	3^{14}	$\mathbf{2^{35}}$
13	2^{33}	2^{61}	3^{19}	$\mathbf{2^{44}}$
19	2^{42}	2^{69}	3^{25}	2^{54}
24	2^{50}	2^{77}	3^{30}	2^{62}

Table 6.3: Complexity of our attack against Poseidon, compared with the security claims given by the authors and by the challenges, with $\mathsf{RF} = 8$. Complexity figures in bold correspond to attacks that we have implemented in practice.

and the addition of the round constants, the second one is quite similar but replaces S with its inverse S^{-1}. The two steps of each round are described in Figure 6.6.

For our study, we use the specifications of Rescue–Prime [SAD20], which means in particular that in each round, we first apply S and then S^{-1} (rather than the contrary as described in the original paper [AAB+19]).

The challenges from the Ethereum Foundation use $m = 3$ or $m = 2$, and the S-boxes are $x \mapsto x^3$ and its inverse $x \mapsto x^{1/3}$.

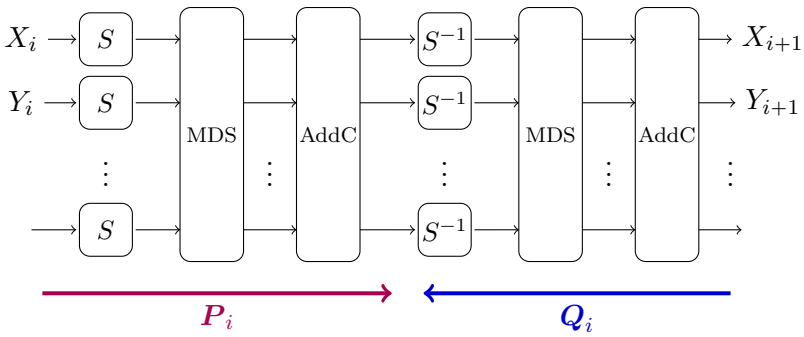


Figure 6.6: Round i of Rescue–Prime.

Attack description. Rescue–Prime cannot be efficiently written as a univariate polynomial system, because it uses both the S-boxes $x \mapsto x^3$ and $x \mapsto x^{1/3}$. Each S-box has a low univariate degree in one direction, but a high degree in the other direction. Therefore, we add intermediate variables so that each S-box can be described with a low-degree equation, and we build a multivariate system.

More precisely, let us consider Rescue–Prime with a t-element state ($m = 2$ or $m = 3$) and N rounds. We use variables $(x_0, y_0, \ldots)$ to represent the input and $(x_i, y_i, \ldots)$ to represent the internal state after the i-th round ($m(N+1)$ variables in total). As shown in Figure 6.6, we can write m equations linking the m variables at the input and output of round i, using only the direct S-box $x \mapsto x^3$. Therefore, we have degree-3 equations:

$$\forall j \in \{1, \ldots, m\},\ p_{i,j}(x_i, y_i, \ldots) - q_{i,j}(x_{i+1}, y_{i+1}, \ldots) = 0\ .$$

If we add equations $x_0 = 0$ and $x_N = 0$, we obtain a system of polynomial equations representing the CICO problem. We observe that the input variables can be removed. Indeed, each $S(x_0), S(y_0), \ldots$ can be written as degree-3 polynomial of $x_1, y_1, \ldots$. Given that $S(x_0) = 0$, it follows that we can only keep the corresponding polynomial equal to 0, and then remove the input variables. We can also remove x_N because it is fixed to zero, and we obtain a system of $m(N-1)+1$ equations and $mN-1$ variables.

With $m = 2$, we have the same number of equations and variables. However, with $m \geqslant 3$ we have more variables than equations, and we can use the trick of Section 6.5.2 to obtain a smaller system corresponding to a subset of the solutions with one solution on average.

Bypassing the First Round when m = 3. Let us repeat the idea described in Section 6.5.3 and apply it to Rescue–Prime.

Let $m = 3$, and S_1, S_2 such that $S_1(x) = x^3$, and $S_2(x) = S_1^{-1}(x) = x^{1/3}$. We consider an input state after the S-box layer of the second round of the form $({A_0}^{1/3}x, {A_1}^{1/3}x, g)$ (*i.e.* we use $V = ({A_0}^{1/3}, {A_1}^{1/3}, 0)$ and $G = (0, 0, g)$).

We first notice that we can switch the order of the multiplication by the MDS matrix and the addition of the constants. Let

$$\begin{pmatrix} C_0^0 \\ C_1^0 \\ C_2^0 \end{pmatrix} = M^{-1} \begin{pmatrix} c_0^0 \\ c_1^0 \\ c_2^0 \end{pmatrix} .$$

In particular, we have:

$$C_2^0 = \alpha_{0,2}c_0^0 + \alpha_{1,2}c_1^0 + \alpha_{2,2}c_2^0 .$$

As a consequence, using the same notations as above, the value C_2^0, in Figure 6.7, must satisfy

$$\begin{aligned} C_2^0 &= \alpha_{0,2}A_0x^3 + \alpha_{1,2}A_1x^3 + \alpha_{2,2}g^3 \\ &= x^3\left(\alpha_{0,2}A_0 + \alpha_{1,2}A_1\right) + \alpha_{2,2}g^3 . \end{aligned}$$

It is the case provided for instance when:

$$\begin{cases} A_1 &= -\frac{\alpha_{0,2}}{\alpha_{1,2}}A_0 \\ g &= \left(\frac{1}{\alpha_{2,2}}\left(\alpha_{0,2}c_0^0 + \alpha_{1,2}c_1^0\right) + c_2^0\right)^{1/3} \end{cases} . \tag{6.3}$$

Recalling that one round corresponds to two steps, it follows that, if we find a value x such that the image of $({A_0}^{1/3}x, {A_1}^{1/3}x, g)$ through $R-1$ rounds of Rescue–Prime (and a linear layer) is equal to $(*, *, 0)$, then the corresponding input through R-round Rescue–Prime is in $\mathcal{Z}$.

Then, for the remaining $R-1$ rounds, Figure 6.6 shows how we generate the following polynomial equations to avoid the inverse S-box.

$$\forall j \in \{0,1,2\},\ p_{i,j}(x_i, y_i, z_i) - q_{i,j}(x_{i+1}, y_{i+1}, z_{i+1}) = 0 .$$

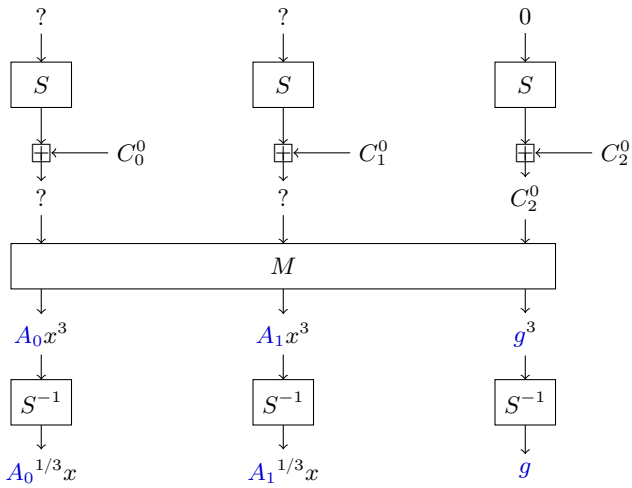


Figure 6.7: How to bypass the first round of Rescue–Prime.

Finally, this results in the following system of polynomial equations:

$$\begin{cases} \forall\, 1 \leqslant i \leqslant N-1, \forall\, j \in \{0,1,2\}, \\ p_{i,j}(x_i, y_i, z_i) - q_{i,j}(x_{i+1}, y_{i+1}, z_{i+1}) = 0\ , \end{cases} \tag{6.4}$$

where $z_N = 0$ and

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} = M \begin{pmatrix} {A_0}^{1/3}x \\ {A_1}^{1/3}x \\ g \end{pmatrix} + \begin{pmatrix} c_0^1 \\ c_1^1 \\ c_2^1 \end{pmatrix}\ .$$

This system has $m(N-1)$ variables and $m(N-1)$ equations. As before, we used `SageMath` to generate our system of equation. However, we used `Magma` to find the solutions of the corresponding multivariate system.

Complexity Analysis. With $m = 3$ branches and N rounds, we obtain a system of $3(N-1)$ degree-3 equations with the same number of variables. In our experiments, the system has $d = 3^{3(N-1)}$ solutions in the algebraic closure of the field, and the $F4$ step is much faster than the FGLM step. Therefore, we estimate the complexity of solving the system by the complexity of running FGLM, which is approximately:

$$d^3 = 3^{9(N-1)}.$$

With $m = 2$ branches and N rounds, we obtain a system of $2N-1$ degree-3 equations with the same number of variables. Therefore, $d = 3^{2N-1}$ and the complexity of solving the system is approximately:

$$d^3 = 3^{6N-3}.$$

Besides, the original paper of Rescue–Prime [SAD20, Section 2.5] claims a lower bound on the complexity of Gröbner basis attacks:

$$\binom{(0.5(\alpha-1)+1)t(\mathsf{N}-1)+3}{t(\mathsf{N}-1)+1}^2.$$

We give explicit values for the proposed challenges in Table 6.4.

N	t	Authors claims	Ethereum claims	d	complexity
4	3	2^{36}	$2^{37.5}$	3^9	$\mathbf{2^{43}}$
6	2	2^{40}	$2^{37.5}$	3^{11}	2^{53}
7	2	2^{48}	$2^{43.5}$	3^{13}	2^{62}
5	3	2^{48}	2^{45}	3^{12}	2^{57}
8	2	2^{56}	$2^{49.5}$	3^{15}	2^{72}

Table 6.4: Complexity of our attack against Rescue–Prime, compared with the security claims given by the authors and by the challenges. Complexity figures in bold correspond to attacks that we have implemented in practice.

6.5.5 Experimental Results

In order to better understand the behaviour of the root-finding tools we relied on in our attacks, we performed additional benchmarks on top of our attacks against the Ethereum challenges. We treat the cases of univariate and multivariate equations separately.

6.5.5.1 Univariate solving

For root-finding of univariate polynomials, we investigated the FLINT [tea23], version 2.8.3 and NTL 11.5.1 C libraries. Both support operations related to big polynomials in finite fields, but NTL was considerably faster for different sizes of toy polynomials, therefore we chose to benchmark only NTL. In order to work with high degree polynomials with NTL, we need to apply a small patch to the library source files to increase the value of `NTL_FFTMaxRoot`.

Table 6.5 presents our experimental results, with $p = 18446744073709551557 = 2^{64} - 59$. Given a degree $d = 3^k$, we generate the polynomial modeling the CICO problem for Poseidon with $(\mathsf{RP}, \mathsf{RF}) = (k-6, 8)$ and for Feistel–MiMC with $k+1$ rounds , and random polynomials (each of the $d+1$ coefficients is taken randomly in $\mathbb{F}_p$). For each instance and degree, we launched $\min(32, 2^{18-k})$ jobs, with varying random polynomials. For all instances, the standard deviation of the memory consumption is negligible (on average 10^{-5} times the average value), and the standard deviation of time stays under 3 percent of the average value.

The data is represented in Figure 6.8, and we performed a linear regression of the time and memory usage of random polynomials root finding. We notice that

the structure of the polynomials of Feistel–MiMC and Poseidon does not offer a significant speed up to the root finding compared to random polynomials.

Because the theoretical complexity is quasi-linear, the linear regression should be treated cautiously. In addition, the benchmarks apply only on 1 core and do not account for parallelization. We expect to speed up the univariate root finding with NTL parallelization (which, officially, is supported), but some tests showed that NTL CPU usage does not exceed 300%, even with more than 3 threads.

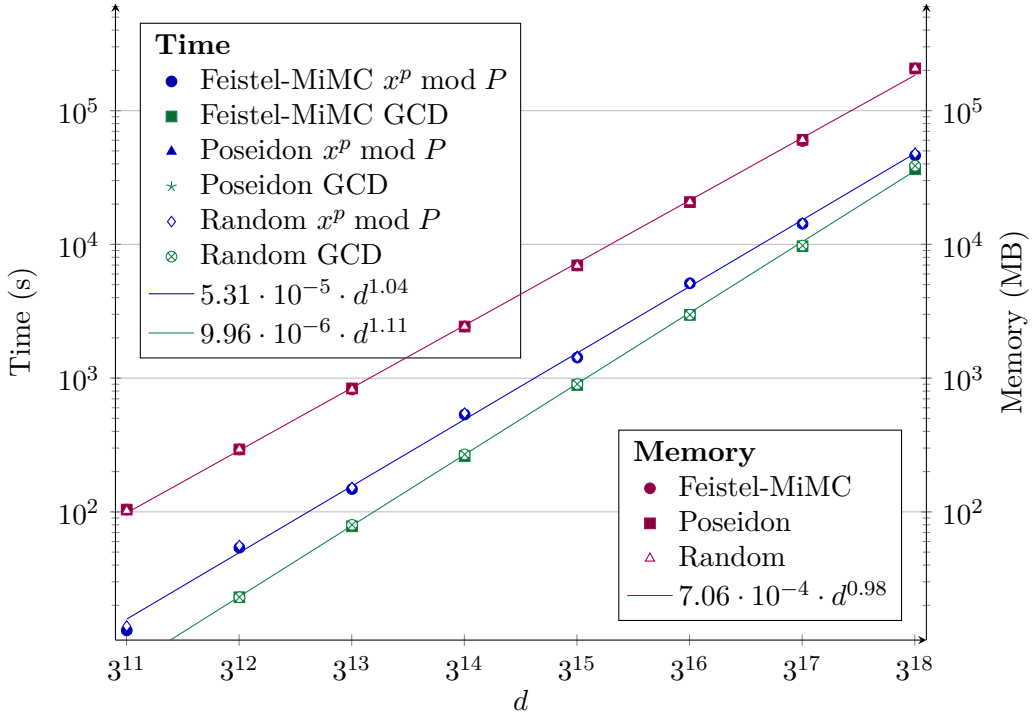


Figure 6.8: Benchmarks of univariate root finding with NTL (using 1 core of an Intel Xeon E7-4860).

6.5.5.2 Multivariate solving

For benchmarks of multivariate solving of the Rescue–Prime system, we chose to use `Magma` [BCP97] 2.21. We compare the resources needed for the resolution of the Rescue–Prime polynomial system to the resources needed for random equivalent systems. It should be noted that `Magma` implements the F4 algorithm [FGH+14b] to find the grevlex Gröbner basis, and the FGLM algorithm [FGL+93] in cubic complexity for the change of ordering. Also, there seems to be a fixed memory overhead of 32 MB when using `Magma` for Gröbner basis, therefore we did not take into account low-memory points in the linear regressions.

Table 6.6 and Figure 6.9 present the results for the resolution of a k-round Rescue–Prime instance, $k = 3, 4$, with $m = 3$ and $p = 18446744073709551557 = 2^{64} - 57$. These are compared to random equivalent systems of n equations of degree

3 on n variables, generated randomly by affecting a random coefficient of $\mathbb{F}_p$ to each possible monomial of degree $\leqslant 3$. In order to give a better insight on the evolution of the resources consumption, we chose to additionally benchmark random systems with $n = 5, 7, 8$, which do not correspond to any version of Rescue–Prime.

For systems of n equations, we launched $\min(2^{2(9-n)}, 64)$ jobs. The standard deviation of time and memory consumption never exceeds 2% of the average value. The program was cut after 7 days for random systems with $n = 9$. The F4 step finished, but the FGLM step did not finish (after approximately 44 hours). The linear regression of Rescue system FGLM time might be biased compared to Random FGLM time, because only Rescue possesses a data point with $n = 9$, which demands heavy resources in memory, potentially causing some overhead.

The results highlight several properties:

- We observe that the theoretical maximal ideal degree is reached, for all systems: 3^n for n equations of degree 3.
- The F4 run time varies considerably between the Rescue–Prime system and a random system. For 4-round Rescue–Prime ($n = 9$), there is almost a factor 50 between the run time of F4 on the Rescue–Prime system and on a random system.
- F4 slightly dominates in time for random system, but FGLM heavily dominates for Rescue–Prime systems.
- The case with 6 equations is the only point of comparison between the Rescue–Prime system and the random system, but on this data point, FGLM is faster on Rescue–Prime than on a random system, despite having the same ideal degree (729).
- The memory consumption seems to essentially follow the same linear regression for both Rescue–Prime systems and Random systems.

6.6 Algebraic cryptanalysis of Ciminion

In this section, we present some attacks on the arithmetization-oriented stream cipher Ciminion [DGG+21b], that we describe in Section 6.6.1. The attacks described in this section come from two works [BBL+22; Bar23]. The first attack is a multivariate attack that breaks Ciminion for very large security levels ($s > 776$). Unlike the original paper [BBL+22], we perform a generic linear change of coordinates to obtain the stability property, and bound the cost of the attack by the $F5$ step. It is presented in Section 6.6.2. The second attack is an univariate attack on an instance of Ciminion proposed by the designers and breaks the security claims of the designers of Ciminion for security levels $s \geqslant 93$, presented in Section 6.6.4. Eventually, we present experimental results of the multivariate attack in Section 6.6.3.

Table 6.5: Benchmarks of univariate root finding with NTL (using 1 core of an Intel Xeon E7-4860), for Poseidon, Feistel–MiMC and random polynomials of several degrees, with $p = 18446744073709551557 \approx 2^{64}$. Times are given in seconds and memory usage in MegaBytes.

System	Degree	3^{11}	3^{12}	3^{13}	3^{14}	3^{15}	3^{16}	3^{17}	3^{18}
Feistel–MiMC	$x^p \bmod P$ time	13	54	148	535	1,426	5,119	14,243	46,256
	GCD time	7	23	78	261	889	2,970	9,687	36,451
	Total time	20	77	226	796	2,315	8,089	23,930	82,707
	Memory	104	293	822	2,431	6,967	20,696	59,227	$2.07 \cdot 10^5$
Poseidon	$x^p \bmod P$ time	13	54	148	534	1,454	5,083	14,241	47,963
	GCD time	8	23	78	262	893	2,964	9,699	38,541
	Total time	21	77	226	796	2,347	8,047	23,940	86,504
	Memory	104	293	838	2,431	6,968	20,696	60,538	$2.07 \cdot 10^5$
Random	$x^p \bmod P$ time	14	56	152	547	1,433	5,117	14,406	47,964
	GCD time	7	23	80	269	903	2,976	9,790	38,693
	Total time	21	79	232	816	2,336	8,093	24,196	86,657
	Memory	102	293	822	2,431	6,935	20,696	60,538	$2.07 \cdot 10^5$

Table 6.6: Benchmarks of multivariate root finding with `Magma` using 1 CPU core of an Intel Xeon Gold 5218, for Rescue and Rescue-like systems. Times are given in seconds and memory usage in MegaBytes.

System	Number of equations	5	6	7	8	9
Rescue–Prime	Rounds		3			4
	F4 time		1.41			8,500
	FGLM time		7.77			$2.5 \cdot 10^5$
	Memory		112			58,675
	Ideal degree		729			19,683
Random	F4 time	0.25	8.23	299	11,120	$4.46 \cdot 10^5$
	FGLM time	0.58	11.15	263	6,490	
	Memory	32	134	936	7,945	
	Ideal degree	243	729	2,187	6,561	19,683

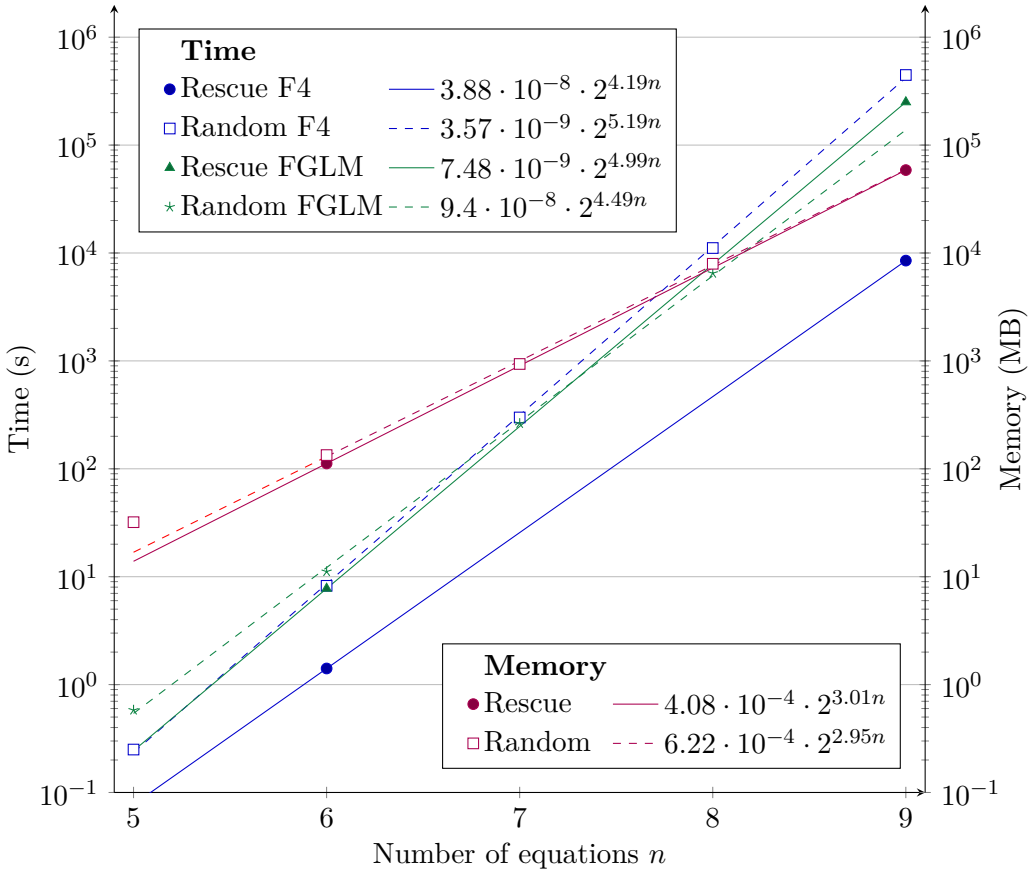


Figure 6.9: Benchmarks of multivariate root finding with `Magma` using 1 CPU core of an Intel Xeon Gold 5218, for Rescue–Prime and Rescue–Prime-like systems (n equations of degree 3), with $p = 18446744073709551557$ and $m = 3$. 3-round and 4-round Rescue-prime respectively correspond to $n = 6$ and $n = 9$.

6.6.1 Specification and Security Analysis of Ciminion

Ciminion is an Arithmetization-Oriented stream cipher published at Eurocrypt 2021 [DGG+21b] operating on $\mathbb{F}_q$ with $q \geqslant 2^{64}$. Ciminion takes as input a nonce $\aleph \in \mathbb{F}_q$ and a key $K \in \mathbb{F}_q$, and produces a sequence of key stream elements $\alpha_1, \alpha_2, \ldots$ that are added to the plaintext blocks to yield ciphertext blocks. Unlike Feistel–MiMC, Poseidon and Rescue–Prime, Ciminion does not use a power map S-box (such as $x \to x^3$) and the non-linear diffusion instead comes from the use of Toffoli gates $(a, b, c) \mapsto (a, b, c + ab)$. This implies that the forward and inverse round function both are quadratic. In addition, Ciminion uses a light linear layer instead of an MDS matrix. Ciminion's encryption scheme is presented in Figure 6.10. Two permutations using the same round function, presented in Figure 6.11b, are employed: p_C and p_E of respectively N and R rounds. The round function for round i is denoted f_i. It uses four round constants RC_ℓ, with $\ell = i$ for p_C , and $\ell = i + N - R$ for p_E. $RC4$ is assumed to be different from 0 or 1. rol is a quadratic

rolling function described in Figure 6.11a. There is a key schedule that generates keystream elements $K_1, K_2 \ldots$ from K, and we do not exploit it in our attacks; we therefore consider that the round keys K_i are unrelated round keys, and the objective of an attacker is to recover K_1 and K_2.

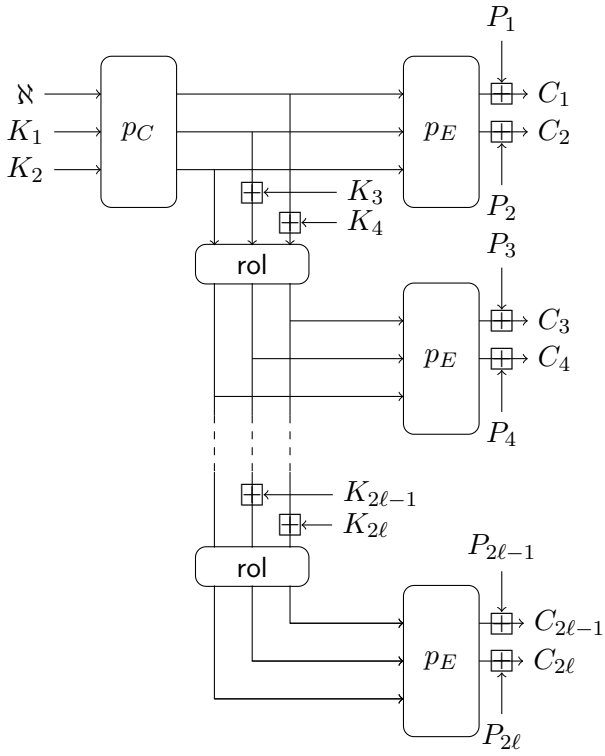
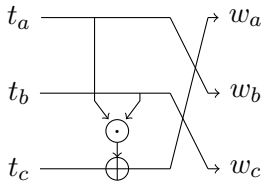
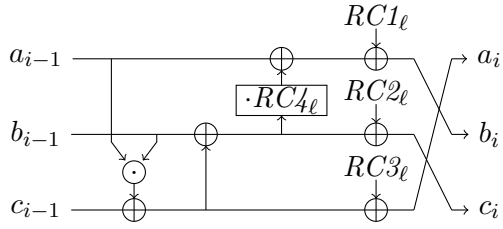


Figure 6.10: The Ciminion encryption over $\mathbb{F}_p$ (replace + by $\oplus$ over $\mathbb{F}_{2^n}$).



(a) rolling function rol.



(b) Ciminion round function.

Figure 6.11: Components of Ciminion.

The security of Ciminion comes from the truncation of one output element of each permutation p_E, to prevent the recovery of intermediate states. Indeed, the knowledge of a truncated element would allow to recover the intermediate states by simply inverting the corresponding p_E, and to recover the round keys K_1 and K_2 by inverting p_C. Therefore, a simple guess of a truncated element would recover

the first round keys, so the security of Ciminion cannot exceed $\log_2(q)$ by design. There are three instances of Ciminion proposed by the designers, with different number of rounds for p_E and p_C: the standard and conservative instances, with no data restriction for the attacker, and the data limit instance, where the attacker can only query up to $2^{s/2}$ data. The number of rounds in each instance is presented in Table 6.7, for a security level s.

Instance	N	R
Standard	$s+6$	$\lceil \frac{s+37}{12} \rceil$
Data limit $2^{s/2}$	$\lceil \frac{2(s+6)}{3} \rceil$	$\lceil \frac{s+37}{12} \rceil$
Conservative	$s+6$	$\lceil \frac{3}{2} \cdot \frac{s+37}{12} \rceil$

Table 6.7: Number of rounds N and R (of p_C and p_E respectively) in the instances proposed by the designers of Ciminion, for a security level of $64 \leqslant s \leqslant q$.

The designers of Ciminion performed a thorough security analysis of Ciminion, exploring all cryptanalysis techniques, such as linear cryptanalysis, differential cryptanalysis, higher-order differentials, interpolation and Gröbner basis attacks. Zhang *et al.* later showed some trivial linear trails under weak round constants [ZLL+23]. We first present the designers analysis on interpolation and Gröbner basis attacks:

Security against Interpolation attacks. The interpolation attack is described in Section 6.2.1. When applied to Ciminion, the attacker models an output keystream $\alpha_1 = C_1 - P_1$ as a polynomial $Q(\aleph)$ of degree d in $\aleph$. Then, the attacker collects $d+1$ keystream blocks α_1 under different nonces, and interpolates Q to recover its coefficients. This way, he gains the knowledge of the function mapping $\aleph$ to the first keystream block α_1.

The number of rounds of p_E and p_C are chosen so that the degree of the polynomial Q exceeds 2^s. It can easily be seen that r rounds of the round function has degree 2^{r-2} from any input to any output. For extra security, the designers chose the number of rounds of p_C to be larger than s: for the standard instance, they chose $N = s+6$, and $R = \lceil \frac{s+37}{12} \rceil$ rounds for P_E.

However, the interpolation attack requires a large amount of data. Because of this very reason, the designers of Ciminion suggested a variant of Ciminion if the attacker is limited to $2^{s/2}$ cipher calls: they decreased the number of rounds in a manner that ensures that the degree of the polynomial still exceeds $2^{s/2}$. This results a *limited data* variant of Ciminion shown in Table 6.7. As a security margin, they chose respectively $N = \lceil \frac{2(s+6)}{3} \rceil$ and $R = \lceil \frac{s+37}{12} \rceil$ as the number of rounds of p_C and p_E. In this settings, the polynomial $P_K(\aleph)$ is of degree $2^{\lceil \frac{2(s+6)}{3} \rceil + \lceil \frac{s+37}{12} \rceil - 1}$, which is larger than $2^{s/2}$, but less than 2^s for large security levels s.

Security against Gröbner basis attack. The designers of Ciminion were not able to generate an algebraic representation of the cipher involving only p_E that

was not underdetermined [DGG+21a, Appendix B]. Therefore, they studied a modified Ciminion, Aiminion, that they conjectured to be weaker than the real Ciminion. In the modified Ciminion, they came up with a system of 6 equations of degrees $\{2^{R-1}, 2^R, 2^R, 2^{R+1}, 2^{R+1}, 2^{R+2}\}$ over 6 variables, where R is the number of rounds of p_E.

The value of R was chosen so that this attack has complexity at least 2^s. More precisely, the authors estimated the complexity of the F5 algorithm with parameters

$$n = 6, \qquad D_{\text{reg}} = 21 \cdot 2^{R-1} - 5 \approx 2^{R+3.4} .$$

Following [BFS04], they estimated the complexity as

$$\binom{n + D_{\text{reg}}}{D_{\text{reg}}}^{\omega} \leqslant \left(\frac{(D_{\text{reg}} + n)^n}{n!}\right)^{\omega} \approx 2^{(6R+10.9)\omega} .$$

The designers took $\omega = 2$ as a lower bound, obtaining a minimum number of rounds $R \geqslant \lceil \frac{s-21.8}{12} \rceil$, and added 5 rounds as a security margin.

6.6.2 Multivariate Algebraic Attack on Ciminion

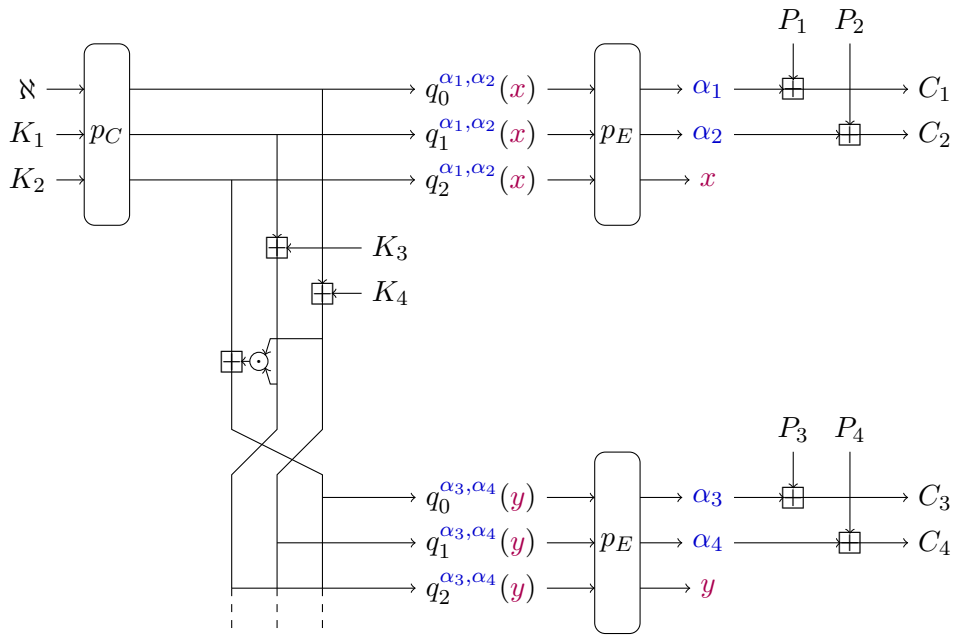


Figure 6.12: A new multivariate modelization of Ciminion.

Instead of looking at a system of equations resulting from a presumed weaker scheme (Aiminion), we study the real scheme and propose a new way to set up a system of equations.

For a given nonce $\aleph$, we consider the first two output blocks. We denote $\alpha_i = C_i - P_i$ and $\alpha'_i = C'_i - P'_i$, for $i = 1 \dots 4$, and introduce two variables $x, y \in \mathbb{F}_q$

for the missing output words (not given as part of the ciphertext) after the first and second permutations p_E (see Figure 6.12). The output of the first permutation p_E is (α_1, α_2, x), therefore, we can write the input as polynomials in x:

$$(q_0^{\alpha_1,\alpha_2}(x), q_1^{\alpha_1,\alpha_2}(x), q_2^{\alpha_1,\alpha_2}(x)) = p_E^{-1}(\alpha_1, \alpha_2, x).$$

Similarly, the output of the second permutation p_E is (α_3, α_4, y), and we can write the corresponding input as polynomials in y:

$$(q_0^{\alpha_3,\alpha_4}(y), q_1^{\alpha_3,\alpha_4}(y), q_2^{\alpha_3,\alpha_4}(y)) = p_E^{-1}(\alpha_3, \alpha_4, y).$$

Then, we write equations linking the input of the first two p_E through the rol function:

$$\begin{aligned}
q_0^{\alpha_1,\alpha_2}(x) &= q_1^{\alpha_3,\alpha_4}(y) - K_4, \\
q_1^{\alpha_1,\alpha_2}(x) &= q_2^{\alpha_3,\alpha_4}(y) - K_3, \\
q_2^{\alpha_1,\alpha_2}(x) &= q_0^{\alpha_3,\alpha_4}(y) - q_1^{\alpha_3,\alpha_4}(y) \odot q_2^{\alpha_3,\alpha_4}(y).
\end{aligned}$$

Finally, taking two nonces $\aleph, \aleph'$, we eliminate the keys K_3, K_4 and obtain a system of four equations in the four variables (x, x', y, y'), using two blocks of ciphertexts from each nonce:

$$\begin{cases}
q_0^{\alpha_1,\alpha_2}(x) - q_0^{\alpha'_1,\alpha'_2}(x') &= q_1^{\alpha_3,\alpha_4}(y) - q_1^{\alpha'_3,\alpha'_4}(y') \\
q_1^{\alpha_1,\alpha_2}(x) - q_1^{\alpha'_1,\alpha'_2}(x') &= q_2^{\alpha_3,\alpha_4}(y) - q_2^{\alpha'_3,\alpha'_4}(y') \\
q_2^{\alpha_1,\alpha_2}(x) &= q_0^{\alpha_3,\alpha_4}(y) - q_1^{\alpha_3,\alpha_4}(y) q_2^{\alpha_3,\alpha_4}(y) \\
q_2^{\alpha'_1,\alpha'_2}(x') &= q_0^{\alpha'_3,\alpha'_4}(y') - q_1^{\alpha'_3,\alpha'_4}(y') q_2^{\alpha'_3,\alpha'_4}(y') \ .
\end{cases} \tag{6.5}$$

Solving this system allows to recover the full internal state, and to deduce the keys K_1, K_2, K_3, K_4. In order to solve the system, we use the approach explained in Section 6.4.2.

Solving complexity. Let us denote p_i the polynomial corresponding to the i-th row of the system, such that we have for all $i = 0 \dots 3$, $p_i(x, x', y, y') = 0$. Given that $\deg(q_0) = 2^{R-1}$, $\deg(q_1) = 2^{R-1}$, and $\deg(q_2) = 2^R$, the degree of the p_i polynomials in x, x', y and y' are:

	x	x'	y	y'
p_0	2^{R-1}	2^{R-1}	2^{R-1}	2^{R-1}
p_1	2^{R-1}	2^{R-1}	2^R	2^R
p_2	2^R	0	$3 \cdot 2^{R-1}$	0
p_3	0	2^R	0	$3 \cdot 2^{R-1}$

In particular, system (6.5) has 2 equations of degree $3 \cdot 2^{R-1}$, 1 of degree 2^R, and 1 of degree 2^{R-1}. Therefore, we have the following parameters:

$$n = 4, \qquad D_{\text{reg}} \leqslant 1 + \sum_{i=1}^{R}(d_i - 1) \approx 2^{R+2.2}, \qquad D_I \leqslant \Pi_{i=1}^{n} d_i \approx 2^{4R+0.2} \ .$$

We can deduce upper bounds on the cost of the steps required to solve the system. In order to use the bounds of change of order algorithms that require the stability and the shape position assumptions, such as [BND22], we perform a generic linear change of coordinates (the base field is large enough and the ideal is assumed to be radical) [BND22].

Computing a Gröbner basis with respect to the *grevlex* order using Faugère's F5 algorithm has asymptotic complexity:

$$\begin{aligned}\binom{n + D_{\text{reg}}}{D_{\text{reg}}}^{\omega} &= \binom{2^{R+2.2} + 4}{2^{R+2.2}}^{\omega} \\ &\leqslant \left(\frac{(2^{R+2.2} + 3)^4}{4!}\right)^{\omega} \approx 2^{(4R+4.2)\omega} \ .\end{aligned}$$

On the other hand, performing the change of order with the variant fast-FGLM [FGH+14a] has asymptotic complexity

$$D_I^{\omega} \approx 2^{(4R+0.2)\omega} \ .$$

HNF [BND22] could also be used for the change of order, but this step is not the bottleneck: the theroretical upper bound on the complexity of $F5$ is higher that the complexity of Fast-FGLM, and the practical experiments of Section 6.6.3 were unfortunately performed without the linear change of coordinate. We therefore use the bound on the complexity of $F5$ to estimate the cost of the attack.

From an attacker point of view, we assume that linear algebra is implemented with Strassen's algorithm, resulting in $\omega = 2.807$ (asymptotically, the best algorithm known has $\omega < 2.373$, but only for implausibly large sizes). Taking the designer's recommended number of rounds $R = \lceil \frac{s+37}{12} \rceil$, this attack is slightly faster than 2^s for large values of s, with a time complexity of:

$$2^{(4R+4.2)\omega} = 2^{(4\lceil \frac{s+37}{12} \rceil + 4.2)\omega} \approx 2^{\frac{4\omega}{12}s + 46.5} \approx 2^{0.94s + 46.5} \ .$$

Theoretically, this breaks the security claims for $s \geqslant 776$. For practical values of s however, the attack is not faster than 2^s, but this shows that the design has much less security margin than anticipated by the designers. In particular, if we take an optimistic value $\omega = 2$ as in the security analysis of the designers, we obtain an attack with complexity roughly $2^{120.4}$ for the 128-bit security version recommended by the designers with 14 rounds.

6.6.3 Experimental Results

For the sake of simplicity, in our experiments, we keep the same prime number $p = 18446744073709551557 = 2^{64} - 59$ as in Section 6.5.5, although it is less than 2^{64} (Ciminion normally requires a prime $p > 2^{64}$).

For r rounds of Ciminion, we present in Section 6.6.2 a modelization of the cryptosystem with 4 equations on 4 variables, of degrees respectively $2^{r-1}, 2^r, 3 \cdot 2^{r-1}$, and $3 \cdot 2^{r-1}$. Unfortunately, these experiments do not perform the generic linear change of coordinates to use the Fast FGLM algorithm. Anyway, the documentation of `Magma` does not mention which algorithm is used for the change of order.

More data points. We chose to add the concept of half rounds to increase the number of data points: $r + 0.5$ rounds of Ciminion is Ciminion where the first branch p_E has undergone $r + 1$ rounds while the second branch p_E has only been through r rounds. With the same technique, we can represent this $r + 0.5$ instance of Ciminion with a system of 4 equations on 4 variables, of degrees $2^r, 2^{r+1}, 2^{r+1}$, and 3×2^r. We compare the Ciminion systems to random systems of 4 equations on 4 variables with the same degrees, where a random coefficient of $\mathbb{F}_p$ is assigned to every possible multivariate monomial of degree equal or less than the degree of the corresponding equation. The 4-round Ciminion was cut off due to memory insufficiency ($\geqslant 192$ GB). Table 6.8 and Figure 6.13 present the results. We did not take into account the memory points with $r = 2$ in the linear regression, because it seems to be a fixed overhead when solving Gröbner bases with `Magma` (regardless of their sizes).

The results allow us to make the following observations.

- The Ciminion system of equation does not reach the maximal ideal degree. We did not succeed to find a simple reduction of the system to explain this fact. This is surprising and not accounted for in the security analysis of the designers of Ciminion.
- The FGLM step heavily dominates the time complexity for Ciminion-like systems.
- We observe a factor 20 between the time complexity of the F4 step of the Ciminion system and random equivalent systems.
- We observe a factor at least 2.5 between the time complexity of the FGLM step of the Ciminion system and random equivalent systems. This is partially due to the lower ideal degree of the Ciminion system.
- After a generic linear change of coordinates, the system would be expected to behave as a random system. This however increases the complexity of FLGM in practice; this is however maybe due to the suboptimal implementation of FGLM in `Magma` 2.21. It is probably possible to reduce the FGLM complexity for random systems with a more performant change of order algorithm [BND22].

- Extrapolating the results to $s = 64$ and $r = 9$ (since $q \approx 2^{64}$ it does not make sense to consider larger security levels), the expected time complexity of the FGLM step is $9 \cdot 10^{-8} \cdot 2^{10.52 \cdot 9} = 2^{60.8}$ seconds (and not operations), which gives a comfortable security margin: our modelization does not break Ciminion for this security level.

6.6.4 Univariate Algebraic Attack on Ciminion

The attack in this section targets the *limited data* variant, in which the attacker can not query more than $2^{s/2}$ data. This attack is a known-plaintext attack using two key stream blocks α_1 and α_2, as highlighted by Figure 6.14. The variable x represents the third truncated output of the first final permutation layer p_E.

Recovery of K_1 and K_2. We represent the nonce $\aleph$ as a polynomial on x. The nonce $\aleph$ and x are separated by the following number of rounds:

$$r = N + R = \lceil \frac{2(s+6)}{3} \rceil + \lceil \frac{s+37}{12} \rceil \leqslant \frac{3s}{4} + 10.$$

As shown in previous papers [ZLL+23; BBL+22], the first input element of a r-round Ciminion permutation as a function of the output elements is of degree 2^{r-1}. This implies that:

$$\deg(q^{\alpha_1,\alpha_2}(x)) = 2^{\lceil \frac{2(s+6)}{3} \rceil + \lceil \frac{s+37}{12} \rceil - 1} \approx 2^{0.75s+6.1}.$$

The truncated element x is a root of the polynomial $q^{\alpha_1,\alpha_2}(x) - \aleph$, we can therefore use a univariate root finding algorithm.

As shown in Section 6.4.1, the univariate root finding algorithm is quasi-linear in the degree of the polynomial. It implies that for large security levels s, we expect the root finding algorithm to run faster than 2^s operations. Let us take the case of $s = \log_2(q) = 256$. q^{α_1,α_2} is of degree $2^{174+25-1} = 2^{198}$, and the root finding algorithm is of complexity $\mathcal{O}(d \log(d)(\log(d) + \log(q)) \log(\log(d))) \approx c \times 2^{198} \times 198 \times (198 + 256) \times 7.6 \approx c \times 2^{217.4}$ field operations, where c is the constant behind the $\mathcal{O}$.

In the case of $s = log(q) = 128$, q^{α_1,α_2} is of degree $2^{90+14-1} = 2^{103}$. The root finding algorithm costs approximately $\mathcal{O}(d \log(d)(\log(d) + \log(q)) \log(\log(d))) = c \times 2^{103} \times 103 \times (103 + 128) \times 6.7 \approx c \times 2^{120.3}$ field operations.

Assuming that a Ciminion encryption costs approximately c operations, this attack breaks the security claims for $s = \log(q) \geqslant 93$.

For now, this attack leads to the recovery of the subkeys K_1 and K_2. If we want to recover subsquent subkeys, we can proceed as explained in the following paragraph, using univariate solving of negligible complexity compared to this attack.

Table 6.8: Benchmarks of multivariate root finding with `Magma` 2.21 using 1 CPU core of an Intel Xeon Gold 5218, for Ciminion and Ciminion-like systems. Times are given in seconds and memory usage in MegaBytes.

System	Expected Ideal degree	288	1024	4608	16384	73728
Ciminion	Rounds	2	2.5	3	3.5	4
	F4 Time	$2 \cdot 10^{-2}$	0.41	4.6	127	5,624
	FGLM Time	0.23	6.7	209.1	13,848	
	Memory	32	125.66	1,279	19,040	
	Ideal degree	170	680	2,736	10,944	43,840
Random	F4 time	0.1	2.36	92.9	3,030	
	FGLM time	0.74	18.96	1,011	32,069	
	Memory	32	226	3,480	47,444	
	Ideal degree	288	1,024	4,608	16,384	

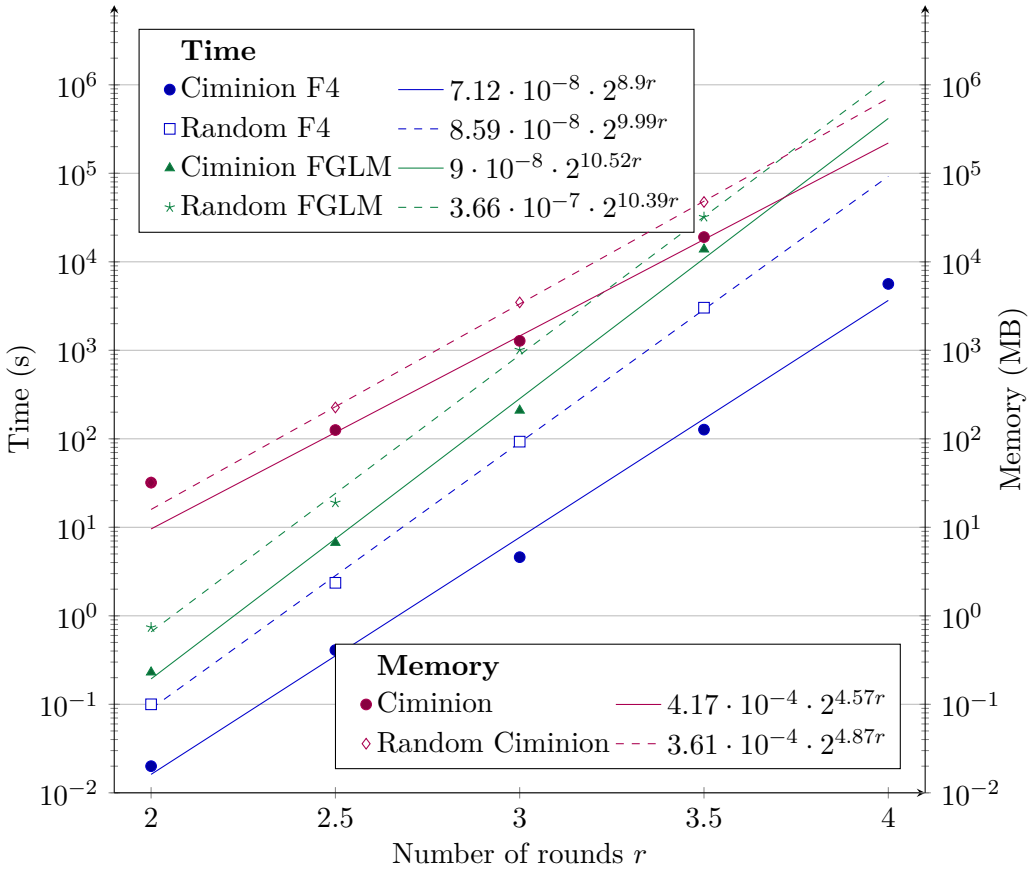


Figure 6.13: Benchmarks of multivariate root finding with `Magma` 2.21 using 1 CPU core of an Intel Xeon Gold 5218, for Ciminion and Ciminion-like systems (4 equations of high degree).

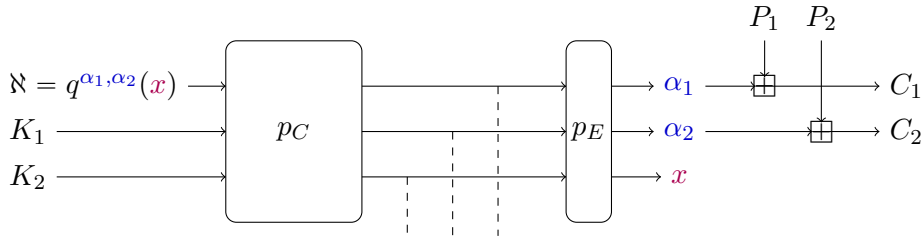


Figure 6.14: A new univariate modelization of Ciminion.

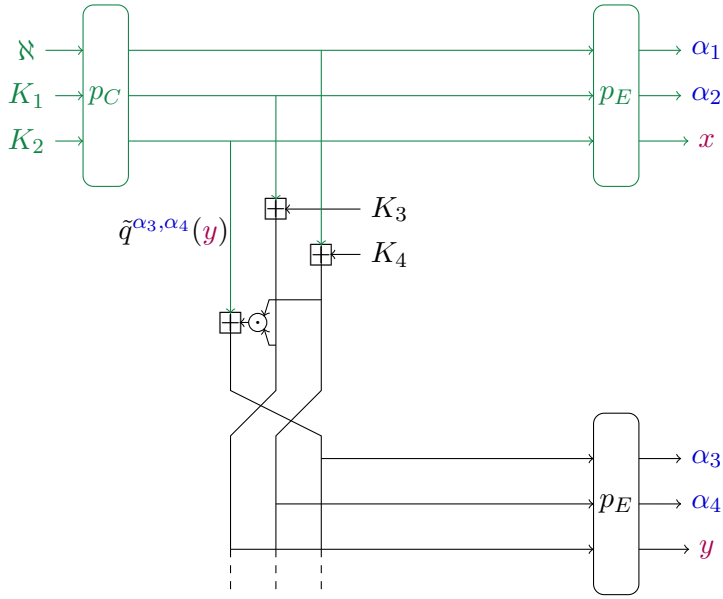


Figure 6.15: Recovery of keystream elements K_3 and K_4. The green wires denote the known internal state elements from the recovery of K_1 and K_2.

Recovery of K_i for $i \geqslant 3$. After the recovery of K_1 and K_2, we query further keystream elements α_i for $i \geqslant 3$. Since $\aleph, K_1, K_2$ are known, the attacker knows the value of three wires at the output of p_C, as depicted in green in Figure 6.15. We denote y the truncated output of the second permutation p_E. We can compute the polynomial $\tilde{q}^{\alpha_3,\alpha_4}(y)$ representing the first inner state element. This inner state element is not dependant on K_3 and K_4 and its value is known to the attacker. We denote it β. The truncated output of the second p_E is a root of $\tilde{q}^{\alpha_3,\alpha_4}(y) - \beta$, which is a polynomial of degree $2^R + 2^{R-1} \approx 2^{R+0.6}$ where $R = \lceil \frac{s+37}{12} \rceil$ is the number of rounds of p_E. The recovery of y is of negligible complexity compared to the first step. This allows to recover the inner state before the second p_E permutation and therefore to recover K_3 and K_4. Ultimately, (K_{2i+1}, K_{2i+2}) for $i \geqslant 2$ can be recovered in a similar manner using a longer keystream.

Aiminion. Since Aiminion uses a key addition before the keystream, it is impossible to express the nonce only from the keystreams α_1, α_2 and the truncated

element x. Instead, the unkown subkeys K_3 and K_4 are involved in the formula. We did not manage to overcome this difficulty to mount an attack.

Chapter 7
The Algebraic FreeLunch: Efficient Gröbner Basis Attacks Against Arithmetization-Oriented Primitives

In this section, we introduce a new type of polynomial solving attack, which we call the FreeLunch attack. The attack modelizes AO ciphers in such a way that the polynomial system to solve is directly a Gröbner basis under a weighted order. The attack is applicable on several AO ciphers published in top-tier cryptography conference, such as `Anemoi` [BBC+23], Arion [RST23], Griffin [GHR+23], and XHash8 [AKM23]. This is a joint work with Aurélien Boeuf, Axel Lemoine, Irati Manterola Ayala, Morten Øygarden, Léo Perrin, and Håvard Raddum [BBL+24b]. This chapter requires the notions on algebraic theory presented in Section 6.3, and reuses the notations defined in Chapter 6.

Contents

7.1 The Algebraic FreeLunch Attack

In this section, we present the theory behind the FreeLunch attack. For this attack to work, we first need to describe our problem using a system of polynomial equations, as in classical polynomial solving attacks. We start with a description of the general form of systems for which a Gröbner basis can be obtained for free (see Section 7.1.1). Then, we show how to deduce a univariate polynomial from this system in Section 7.1.2. Means to create these polynomial systems for various primitives are discussed in Section 7.1.3 and Section 7.1.4, where we reuse and generalize an encoding technique introduced by the authors of `Griffin` in a way that can be applied to iterated functions. The entire solving strategy is summarized in Section 7.1.5, and the FreeLunch attack is described in Algorithm 7.1.

Algorithm 7.1: Overview of the FreeLunch Attack.

1. `sysGen`: Generate a FreeLunch system (Section 7.1.4).
2. `matGen`: Compute the multiplication matrix T_0 (Section 7.1.2).
3. `polyDet`: Compute $f(x_0) = \det\left(x_0^{\alpha_0}\boldsymbol{I}_{D_H} + \sum_{i=0}^{\alpha_0-1} x_0^i M_i\right)$ (Section 7.1.2).
4. `uniSol`: Solve $f(x_0) = 0$.

7.1.1 FreeLunch Systems

We saw in Proposition 6.5 that there is a class of polynomial systems that admits a simple Gröbner basis. We recall this proposition here:

Proposition. *Let G be a set of polynomials of R, $G = \{g_1, ..., g_m\}$. If the leading monomials of g_i and g_j are relatively prime for all $1 \leqslant i \neq j \leqslant m$, then G is a Gröbner basis for $\langle G \rangle$.*

This is the motivation for the following definition.

Definition 7.1 (FreeLunch System)**.** Let R be the ring $\mathbb{F}_q[x_0, \ldots, x_{n-1}]$ and $P = \{p_0, \ldots, p_{n-1}\}$ be a sequence of polynomials of R. We say that P is a **FreeLunch system** if there exists a monomial order $\prec$ and integers $(\alpha_0, \ldots, \alpha_{n-1})$ such that for all $i \in [\![0, n-1]\!]$, $\mathrm{LM}_{\prec}(p_i) = x_i^{\alpha_i}$. Any monomial order $\prec$ that verifies this property is said to be a FreeLunch order.

Note that this is not the first time Proposition 6.5 has been used in cryptography. In [BPW06a], the authors describe a polynomial modeling for AES that can be said to be a FreeLunch system in a *graded lex* order. However, the ensuing change of order computation to a *lex* order is too costly to threaten the security of AES. The following properties are now easy to verify, and were also used in [BPW06a].

Proposition 7.1. *A FreeLunch system P is a Gröbner basis for the ideal $I = \langle P \rangle$ with respect to any of its FreeLunch orders. Moreover, I is zero-dimensional and of ideal degree $D_I = \prod_{i=0}^{n-1} \alpha_i$.*

Proof. The first statement follows directly from Proposition 6.5. For the latter statement, note that the canonical basis of R/I (w.r.t. a FreeLunch order $\prec$) is

$$\mathcal{B}_{\prec}(R/I) = \{x_0^{i_0} \cdots x_{n-1}^{i_{n-1}} \mid 0 \leqslant i_j < \alpha_j, \text{ for } 0 \leqslant j \leqslant n-1\}.$$

Counting all these basis elements yields D_I. □

We conceived a dedicated algorithm for the resolution of FreeLunch systems, which has a competitive time complexity.

7.1.2 Extracting a Univariate Equation from a FreeLunch System

In this section, we aim at proving the following theorem.

Theorem 7.1. *Given a FreeLunch system P, a FreeLunch order $\prec$, and the associated multiplication matrix T_0 of the variable x_0, there exists an algorithm to compute a solution for x_0 with time complexity:*

$$\tilde{\mathcal{O}}\left(\alpha_0 \left(\prod_{i=1}^{n-1} \alpha_i\right)^{\omega}\right) = \tilde{\mathcal{O}}\left(\frac{D_I^{\omega}}{\alpha_0^{\omega-1}}\right) .$$

While a FreeLunch system is already a Gröbner basis, it is typically only so under specific monomial orders, as we will see in later sections. To easily retrieve the solutions of a FreeLunch system, we look for a univariate polynomial belonging to the ideal spawned by the system. To do so, a common approach is to compute a Gröbner basis in the *lex* order. Given an initial Gröbner basis, computing a *lex* Gröbner basis can be performed using a *change of order* algorithm.

Existing change of order algorithms. The FGLM algorithm [FGL+93] provides an efficient method for changing the monomial orders of Gröbner bases of zero-dimensional ideals, with a running time of $\mathcal{O}(nD_I^3)$ operations and no conditions on the Gröbner bases, on the monomial order or on the ideal. Note that this cost includes computing the multiplication matrix T_0.

Later algorithms [FM17; FGH+14a; NS20; BND22] significantly improve upon this running time, but require various assumptions on the input basis and underlying ideal. For instance, [FM17; FGH+14a; BND22] assume that the multiplication matrix T_0 is either given, or can be efficiently computed. Note that the latter is a consequence of the stability property (see Definition 6.13), which is assumed in some of these works. Unfortunately, FreeLunch systems do not generally satisfy this property. In fact, the authors of [BND22] state that when the base field is large enough and the ideal under consideration is radical, the stability property

can be ensured through a generic linear change of coordinates. The issue is that doing so might transform the FreeLunch system into a different type of system that is not a Gröbner basis.

We briefly recall the effectiveness of the change of order algorithms assuming that: i) T_0 is given; ii) I is in shape position. In this case, the algorithm of [FGH+14a] runs in $\mathcal{O}(D_I^\omega \log(D_I))$ and supposes that the input order is *grevlex* and the output order is *lex*. [NS20] runs in $\mathcal{O}(nD_I^\omega \log(D_I))$ with no additional hypothesis, and in $\mathcal{O}(D_I^\omega \log(D_I))$ when the ideal is in shape position. In our case, we are particularly interested in some algorithms that benefit from the sparsity of T_0, represented by its sparsity indicator t. The algorithm of [FM17, Theorem 3.2] for example runs in $\mathcal{O}(tD_I^2)$. The algorithm of [BND22] achieves an even better time complexity, of $\tilde{\mathcal{O}}(t^{\omega-1}D_I)$, if the input order is *grevlex* and the output order is *lex*. However, it is not clear to us if the ideas as presented in [BND22] can be directly generalized to our setting, i.e. with a weighted input monomial order, even if T_0 is given. Instead, we will develop a dedicated resolution algorithm from a basis in a FreeLunch order when T_0 is given, whose running time happens to coincide with that of [BND22] ($\tilde{\mathcal{O}}(t^{\omega-1}D_I)$).

A new approach. These reasons led us to design a new dedicated algorithm for finding a univariate polynomial $f_0(x_0)$ belonging to the ideal, exploiting the sparsity of the multiplication matrix T_0.

Let P be a FreeLunch system, I be its zero-dimensional ideal of $R = \mathbb{F}_q[x_0, \ldots, x_{n-1}]$, $\prec$ one of its FreeLunch orders, $\mathcal{B}_\prec = (\epsilon_1, \ldots, \epsilon_{D_I})$ the canonical basis of R/I, and T_0 the multiplication matrix corresponding to the variable x_0. Let H be the subspace of R/I containing the classes h of R/I where the normal form of h with respect to $\prec$ does not contain the variable x_0. Let D_H be the dimension of H and $\mathcal{B}_\prec^H = [\phi_1, \ldots, \phi_{D_H}]$ be a canonical basis for the subspace H. It is clear that $\mathcal{B}_\prec^H$ exactly consists of the monomials m of $\mathcal{B}_\prec$ such that $x_0 \nmid m$. Thus, it holds that $D_H = \prod_{i=1}^{n-1} \alpha_i = D_I/\alpha_0$. We order the basis $\mathcal{B}_\prec$ specifically as

$$[\phi_1, \ldots, \phi_{D_H},\ x_0\phi_1, \ldots, x_0\phi_{D_H},\ x_0^2\phi_1, \ldots, x_0^2\phi_{D_H},\ \ldots,\ x_0^{\alpha_0-1}\phi_1, \ldots, x_0^{\alpha_0-1}\phi_{D_H}]\,,$$

and identify any polynomial $f \in R/I$ with its coefficient vector v_f of length D_I. The coefficient vector for the polynomial $x_0 f \in R/I$ can then be computed as a matrix/vector multiplication $T_0 v_f^\top$ for a fixed matrix T_0. The following lemma gives the structure of T_0.

Lemma 7.1. *Under the basis $\mathcal{B}_\prec$, the matrix T_0 is of the following form, represented as a block matrix with block sizes $D_H \times D_H$:*

$$T_0 = \begin{pmatrix} 0 & 0 & \ldots & 0 & -M_0 \\ \boldsymbol{I} & 0 & \ldots & 0 & -M_1 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \ldots & 0 & \boldsymbol{I} & -M_{\alpha_0-1} \end{pmatrix}.$$

The block matrices $M_0, \ldots, M_{\alpha_0-1}$ are a representation of the reduction of $x_0^{\alpha_0}\mathcal{B}_\prec^H$ modulo I. The exact entries in the M_i matrices depend on the particular

polynomials making up the Gröbner basis for the FreeLunch system. We call `matGen` the procedure which, given a basis $\mathcal{B}_\prec$, returns T_0.

Overview of `matGen` We note that FGLM [FGL+93] gives an algorithm with complexity $\mathcal{O}(nD_I^3)$ to compute the multiplication matrix. In practice, we instead use a custom algorithm for the `matGen` step. `matGen` consists of the reductions of $x_0^{\alpha_0}\phi_i$ for $i \in \{1, \dots D_H\}$ by the FreeLunch system. The bound on the number of steps in a reduction by a Gröbner basis in a weighted monomial order is however not clear. In order to give an estimation of the complexity of this step, we implemented `matGen` along with the FreeLunch attack in Section 7.2 and Section 7.3, and benchmarked it. In our implementation, we proposed a variant to the naïve approach: we remarked that the computation of NormalForm$(x_0^{\alpha_0}\phi_i)$ can be speed up if ϕ_i is not a single variable x_i. If $\phi_i = a \times b$ (a and b being non-trivial monomials), we compute NormalForm$(x_0^{\alpha_0}a)$, and then NormalForm(NormalForm$(x_0^{\alpha_0}a)b$). The intermediary normal form corresponds to another columns of T_0 and can be considered free if the columns of T_0 are computed in the right order. In our implementations, we chose $b = x_i$ with α_i as low as possible; this seemed to be the fastest approach.

From Proposition 6.8 it follows that $\det(x_0\boldsymbol{I}_{D_I} - T_0)$ is a univariate polynomial belonging to the ideal I. Computing this determinant and using a root-finding algorithm to solve $\det(x_0\boldsymbol{I}_{D_I} - T_0) = 0$ will finally give us a value for x_0 that solves the CICO problem. The following lemma shows that computing this determinant of this particularly structured matrix T_0 can be done with much lower complexity than for a generic matrix of dimension D_I.

Lemma 7.2. *Let $M_0, \dots, M_{\alpha_0-1}$ be the matrices defined in Lemma 7.1. We have*

$$\det(x_0\boldsymbol{I}_{D_I} - T_0) = \pm \det\left(x_0^{\alpha_0}\boldsymbol{I}_{D_H} + \sum_{i=0}^{\alpha_0-1} x_0^i M_i\right).$$

Proof.

$$\det(x_0\boldsymbol{I}_{D_I} - T_0) = \det\begin{pmatrix} x_0\boldsymbol{I} & 0 & \dots & 0 & M_0 \\ -\boldsymbol{I} & x_0\boldsymbol{I} & \dots & 0 & M_1 \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & 0 & -\boldsymbol{I} & x_0\boldsymbol{I} & M_{\alpha_0-2} \\ 0 & 0 & 0 & -\boldsymbol{I} & x_0\boldsymbol{I} + M_{\alpha_0-1} \end{pmatrix}.$$

The rows of this matrix can be split into a set of α_0 blocks of D_H rows each. Denote these blocks as $L_0, \dots, L_{\alpha_0-1}$ from top to bottom. We now do elementary row operations block-wise, from bottom to the top, with $L_i = L_i + x_0 L_{i+1}$, for $i = \alpha_0 - 2, \dots, 0$. This does not change the value of the determinant and after these row operations the resulting determinant to compute is:

$$\det\begin{pmatrix} 0 & 0 & \dots & 0 & x_0^{\alpha_0}\boldsymbol{I} + \sum_{i=0}^{\alpha_0-1} x_0^i M_i \\ -\boldsymbol{I} & 0 & \dots & 0 & x_0^{\alpha_0-1}\boldsymbol{I} + \sum_{i=0}^{\alpha_0-2} x_0^i M_{i+1} \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \dots & -\boldsymbol{I} & 0 & x_0^2\boldsymbol{I} + \sum_{i=0}^{1} x_0^i M_{i+\alpha_0-2} \\ 0 & \dots & 0 & -\boldsymbol{I} & x_0\boldsymbol{I} + M_{\alpha_0-1} \end{pmatrix}.$$

In this block matrix representation, the determinant of the full matrix is the determinant of the top right matrix, up to the sign $(-1)^{\alpha_0+1}$. □

Complexity Analysis. We call `polyDet` the procedure returning the polynomial $\det(x_0 \boldsymbol{I}_{D_I} - T_0)$ using Lemma 7.2. The determinant of a polynomial matrix of dimensions $m \times m$ and degree d can be computed with $\tilde{\mathcal{O}}(dm^\omega)$ [GJV03, Theorem 4.4]. With our notations, this gives a complexity of $\tilde{\mathcal{O}}(D_I D_H^{\omega-1}) = \tilde{\mathcal{O}}(\alpha_0 D_H^\omega)$. Note that this is the complexity that would be obtained with the algorithm of [BND22] if the system satisfied the *stability* and *shape position* properties. In order to estimate the logarithmic factors in the complexity formula, we bound the complexity with the formula from [GJV03, Theorem 4.4]:

$$\mathcal{O}\left(\sum_{i=0}^{\log(m)}\left(\sum_{j=0}^{i\log(d)} 2^j \mathrm{MM}(2^{-i}n, 2^{i-j}d)\right)\right),$$

where $\mathrm{MM}(m,d)$ is the complexity of a multiplying two $m \times m$ polynomial matrices of degree d. We use a polynomial matrix multiplication algorithm of complexity $\mathcal{O}(m^\omega d\log(d) + m^2 d\log(d)\log(\log(d))) \approx \mathcal{O}(m^\omega d\log(d))$ [CK91, Section 3] to bound the complexity of the determinant computation by $\mathcal{O}(m^\omega d\log(d)^2)$ when m is large. This way, we bound the number of operations of `polyDet` with:

$$\mathcal{O}(\alpha_0 \log(\alpha_0)^2 D_H^\omega)\ . \tag{7.1}$$

The remaining task to show Theorem 7.1 is to recover the roots of a univariate polynomial of degree D_I, a step we refer to as `uniSol`. This costs $\tilde{\mathcal{O}}(D_I)$ operations and is thus negligible in comparison with the `polyDet` step.

We want to highlight that the complexity of the `matGen` step can only be bounded by $\mathcal{O}(nD_I^3)$ [FGL+93, Proposition 3.1] (recall that T_0 is assumed known in Theorem 7.1), and that `matGen` is in theory costlier than `polyDet`, as confirmed with later experiments.

7.1.3 Ordering a FreeLunch

Having seen how to efficiently find solutions for FreeLunch systems, we will focus in the next two subsections on the problem of actually finding them. Recall that FreeLunch systems rely on the existence of specific monomial orders. How can we figure out if such an order exists (and thus, if a system is a FreeLunch)? In general, answering this question is not trivial. However, the systems we will be concerned with in Section 7.2 and Section 7.3 naturally have a deeper structural property that allows for a procedural approach to this problem.

Definition 7.2 (Quasi-triangular System)**.** Let $P = (p_1, \ldots, p_{n-1}, g)$ be a polynomial system in $\mathbb{F}_q[x_0, x_1, \ldots, x_{n-1}]$. We say that P is a ***quasi-triangular* system** if there exists polynomials $q_0, q_1, \ldots, q_{n-1}$, integers $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$, and

$c_0, \ldots, c_{n-1} \in \mathbb{F}_q \backslash \{0\}$ such that

$$\begin{cases} p_i &= c_i x_i^{\alpha_i} + q_i(x_0, \ldots, x_{i-1}) \qquad \text{for } \ 1 \leqslant i \leqslant n-1 \ , \\ g &= c_0 x_0^{\alpha_0} + q_0(x_0, \ldots, x_{n-1}) \ . \end{cases}$$

A quasi-triangular system P can be assigned the following monomial order that is naturally motivated by the FreeLunch definition.

Construction 1. *For a quasi-triangular system P, we define its quasi-triangular order, $\prec_T$, as the monomial order from Definition 6.7 associated with the weight vector defined recursively by:*

$$\begin{cases} \mathrm{wt}(x_0) &= 1 \ , \\ \mathrm{wt}(x_i) &= \mathrm{wt}\left(\mathrm{LM}_{\prec_T}\left(q_i(x_0, x_1, \ldots, x_{i-1})\right)\right)/\alpha_i \qquad \text{for } \ 1 \leqslant i \leqslant n-1 \ . \end{cases}$$

The recursion is well-defined since the leading monomial of q_i and its associated weight are only dependent on the weights of x_j for $j < i$. The definition ensures that $\mathrm{LM}_{\prec_T}(p_i) = x_i^{\alpha_i}$ for $1 \leqslant i \leqslant n-1$. Hence, a quasi-triangular system P is a FreeLunch system with respect to $\prec_T$ if the leading monomial of g is univariate in x_0, which gives the following Proposition:

Proposition 7.2 (Ordering a FreeLunch)**.** *Let P be a quasi-triangular system, and $\prec_T$ be its quasi-triangular order. If $\alpha_0 > \mathrm{wt}(\mathrm{LM}_{\prec_T}(q_0(x_0, \ldots, x_{n-1})))$ then P is a FreeLunch system and $\prec_T$ is one of its FreeLunch orders.*

As we will see below, such systems naturally occur when investigating some cryptographic permutations.

7.1.4 FreeLunch Systems From Iterated Functions

The permutations we target share the same structure: a composition of a number of round functions. The input and output of every round is a state of t elements[1] from $\mathbb{F}_q$ and the round functions typically consist of a limited number of multiplications and α-th roots in $\mathbb{F}_q$. Writing them out directly as polynomial functions yields polynomials of high degree, owing to the α-th root operations. A natural modeling strategy introduces a new variable for each of them to keep the degree growth manageable, as $x = y^\alpha$ is of much lower degree than $y = x^{1/\alpha}$ when $\alpha \in \{3, 5, ..., 257\}$ and $|\mathbb{F}|$ is large. In this section, we take inspiration from an encoding suggested by the authors of `Griffin` [GHR+23] and show how to model this class of primitives as polynomials that form a low degree FreeLunch system.

7.1.4.1 Toy example

Let us start with a toy SPN of two rounds, where the round function F is given by $F = S \circ A : \mathbb{F}_q^2 \to \mathbb{F}_q^2$, for an invertible affine layer A and a non-linear layer

[1] We say that the permutation has t *branches*, or as we like to think of them, *brunches*.

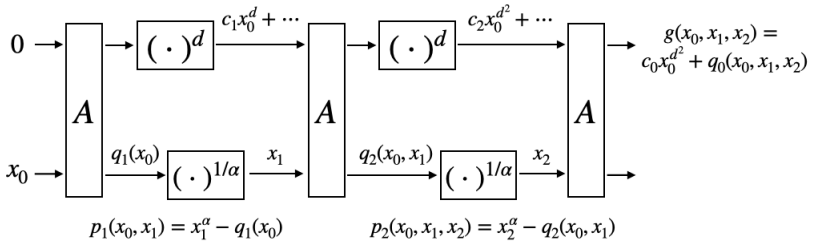


Figure 7.1: Quasi-triangular system for a simple SPN with two branches and two rounds.

S. Moreover, we write $S = (S_1, S_2)$ where $S_1(y) = y^d$, for a small integer d, and $S_2(y) = y^{1/\alpha}$. This simple construction is shown in Figure 7.1, where we also label the branches with variables and polynomials at different points. We consider a CICO problem with input $(0, x_0)$.

As we assume $d \ll p$, we note that the round function F can only achieve a high degree as a polynomial function $\mathbb{F}_q^2 \to \mathbb{F}_q^2$ due to the map S_2. Thus, we introduce new variables x_1 and x_2 for the output of S_2 in the first and second rounds, respectively. The polynomials p_1, p_2 relate the symbolic input and output of the two S_2-functions, where q_1 is an affine polynomial in x_0 that is input to S_2 in the first round, and $q_2(x_0, x_1)$ is the input to S_2 in the second round and has degree d in the x_0-variable and degree 1 in the x_1-variable. Finally, we let $g(x_0, x_1, x_2)$ represent the first output of the construction that is required to be 0 by the CICO-problem. We can now write g as

$$g(x_0, x_1, x_2) = c_0 x_0^{d^2} + q_0(x_0, x_1, x_2),$$

for a suitable constant $c_0 \in \mathbb{F}_q$, and where $q_0(x_0, x_1, x_2)$ has degree $< d^2$ in x_0, degree d in x_1 and degree 1 in x_2. If $c_0 \neq 0$ we observe that $P = \{p_1, p_2, g\}$ forms a quasi-triangular system (Definition 7.2), whose solutions yield a solution to the specified CICO-problem. The weight vector of $\prec_T$ from Construction 1 is $(1, 1/\alpha, d/\alpha)$, and it is straightforward to verify that P satisfies the condition of Proposition 7.2. Hence, P is a FreeLunch system.

7.1.4.2 General case

The above example shows the core idea for how a FreeLunch system can be made from a round function that relies on the functional inverse of low degree function to achieve a high degree. Let us generalize this insight. Let $F_i : \mathbb{F}_q^t \to \mathbb{F}_q^t$, $\boldsymbol{z}_{i-1} \mapsto \boldsymbol{z}_i$, denote the i-th round of a primitive, where $\boldsymbol{z}_{i-1} = (z_{i-1,0}, \ldots, z_{i-1,t-1})$ is the state after $i-1$ rounds. Recall that F_i may itself have a high degree (in $\boldsymbol{z}_{i-1}$), but suppose there exists a set of variables $\boldsymbol{x}_i = \{x_{i,0}, \ldots, x_{i,\ell_i-1}\}$ satisfying

$$x_{i,j}^{\alpha_{i,j}} = \mathcal{L}_{i,j}(\boldsymbol{z}_{i-1}), \text{ for } 0 \leqslant j < \ell_i, \tag{7.2}$$

where $\alpha_{i,j}$ is an integer and $\mathcal{L}_{i,j}$ an affine function. Moreover, suppose that there exists a polynomial function $G_i : \mathbb{F}_q^{t+\ell_i} \to \mathbb{F}_q^t$ of low degree d_i, satisfying

$$F_i(\boldsymbol{z}_{i-1}) = \{G_i(\boldsymbol{z}_{i-1}, \boldsymbol{x}_i) \mid \boldsymbol{x}_i \text{ satisfies (7.2)}\}. \tag{7.3}$$

In other words, while F_i and G_i are different as polynomial functions, they yield the same output when $\boldsymbol{x}_i$ is restricted by (7.2). For instance, in the toy example above, we used

$$G_1(\boldsymbol{z}_0, x_1) = \left((A_1(\boldsymbol{z}_0))^d, x_1\right), \qquad G_2(\boldsymbol{z}_1, x_2) = \left((A_1(\boldsymbol{z}_1))^d, x_2\right),$$
$$x_1^\alpha = A_2(\boldsymbol{z}_0), \qquad x_2^\alpha = A_2(\boldsymbol{z}_1),$$

where A_1 (resp. A_2) denotes the first (resp. second) output of A.

Polynomial Modeling.

We now have an iterated function of t branches where each round can be described using the functions $\mathcal{G} = \{G_1, \ldots, G_r\}$ and satisfying (7.2) and (7.3), and where G_i is of degree d_i. We introduce the shorthand $d_{\leqslant i} = d_1 d_2 \cdots d_i$, and we require that the exponents d_i are small enough to ensure that their composition will not exceed the maximal degree determined by the finite field, *i.e.* $d_{\leqslant r} < |\mathbb{F}_q| - 1$.

With this in place, we give the following blueprint for constructing a polynomial system. Recall that we focus on the variant of the CICO-problem where a single input in $\mathbb{F}_q$ is unknown, which we will symbolically denote by x_0, and the output of the first branch should be 0. The initial state is written as $\boldsymbol{z}_0(x_0)$, which consists of t affine polynomials in x_0. The following state is defined as $\boldsymbol{z}_1(x_0, \boldsymbol{x}_1) = G_1(\boldsymbol{z}_0, \boldsymbol{x}_1)$, where we note that $\boldsymbol{z}_1$ is now t polynomials of at most degree d_1 in the variables $x_0, \boldsymbol{x}_1$. Furthermore, we define functions $\boldsymbol{p}_1 = \{p_{1,0}, \ldots, p_{1,\ell_1-1}\}$ to encode the relations (7.2) that we encounter in this step. That is, for $\boldsymbol{x}_1 = \{x_{1,0}, \ldots, x_{1,\ell_1-1}\}$, we construct the polynomials

$$p_{1,j} = x_{1,j}^{\alpha_{1,j}} - \mathcal{L}_{1,j}(\boldsymbol{z}_0), \text{ for } 0 \leqslant j < \ell_1 \ .$$

This process of updating the state $\boldsymbol{z}_i$ and constructing polynomials[2] $\boldsymbol{p}_i$ is repeated for all rounds up to $r-1$. In the last round, we generate polynomials $\boldsymbol{p}_r$ as before, but instead of updating the state, we compute the final polynomial

$$g(x_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_r) = [G_r(\boldsymbol{z}_{r-1}, \boldsymbol{x}_r)]_1,$$

where $[\cdot]_1$ means the first polynomial of $G_r(\boldsymbol{z}_{r-1}, \boldsymbol{x}_r)$. This construction yields the polynomial system $P_{\mathcal{G}} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_r, g\}$ over the ring $\mathbb{F}_q[x_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_r]$.

[2] If a single variable is introduced in a round, we will ease notation by writing $\boldsymbol{x}_i = x_i$, $\boldsymbol{p}_i = p_i$ and α_i.

$P_{\mathcal{G}}$ as a FreeLunch system.

It is easy to verify that $P_{\mathcal{G}}$ is a quasi-triangular system if g contains a univariate monomial in x_0. In fact, this is a stronger case than the generic quasi-triangular systems considered in Section 7.1.3, since we are also able to bound the degrees of the polynomials in $P_{\mathcal{G}}$ by round degrees $d_1, \ldots, d_r$. This allows us to give an analogous variant of Proposition 7.2 for $P_{\mathcal{G}}$. Instead of a condition on the entire system that could be computationally expensive to verify, we reduce the assumption to the condition of a single monomial in g.

Proposition 7.3. *Let $P_{\mathcal{G}}$ be a polynomial system as constructed above, where all $\alpha_{i,j}$ from (7.2) are at least 2, and the functions $\mathcal{G} = \{G_1, \ldots, G_r\}$ are of degrees $d_1, \ldots, d_r \geqslant 2$. Then $P_{\mathcal{G}}$ is a FreeLunch system if g contains the monomial $x_0^{d_{\leqslant r}}$.*

Before proving the proposition, we start by defining $\prec_{\mathcal{G}}$, which is the monomial order from Definition 6.7 whose weight vector is given by

$$\begin{cases} \mathrm{wt}(x_0) &= 1 \ , \\ \mathrm{wt}(x_{i,j}) &= d_{\leqslant i-1}/\alpha_{i,j} \qquad \text{for } \ 1 \leqslant i \leqslant r \ \text{ and } \ 1 \leqslant j \leqslant \ell_i \ , \end{cases}$$

where we define $d_{\leqslant 0} = 1$. Recall that $\boldsymbol{z}_i$ denotes the i-th state represented by t polynomials in $x_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_i$. We will write $\mathrm{wt}(\mathrm{LM}(\boldsymbol{z}_i))$ for the maximal weight among the monomials of these t polynomials.

Lemma 7.3. *Let $\boldsymbol{z}_i$ be the i-th state associated with a system $\mathcal{G}$ that satisfies the conditions of Proposition 7.3. Then the following inequality holds for $\prec_{\mathcal{G}}$:*

$$\mathrm{wt}\left(\mathrm{LM}(\boldsymbol{z}_i)\right) \leqslant d_{\leqslant i} \ .$$

Proof. We proceed by induction. The base case of $i = 0$ is immediate since $\boldsymbol{z}_0$ is affine in x_0, and $d_{\leqslant 0} = 1$ by definition. For the induction step, we recall that $\boldsymbol{z}_i = G_i(\boldsymbol{z}_{i-1}, \boldsymbol{x}_i)$, where G_i has degree d_i. Thus we have

$$\mathrm{wt}(\mathrm{LM}(\boldsymbol{z}_i)) \leqslant d_i \cdot \max\{\mathrm{wt}(\mathrm{LM}(\boldsymbol{z}_{i-1})), \mathrm{wt}(x_{i,1}), \ldots, \mathrm{wt}(x_{i,\ell_i})\} \ .$$

Now we have $\mathrm{wt}(x_{i,j}) < d_{\leqslant i-1}$, and $\mathrm{wt}(\mathrm{LM}(\boldsymbol{z}_{i-1})) \leqslant d_{\leqslant i-1}$ by the induction hypothesis. Hence

$$\mathrm{wt}(\mathrm{LM}(\boldsymbol{z}_i)) \leqslant d_i d_{\leqslant i-1} = d_{\leqslant i} \ .$$

□

The proof of this lemma also implies that $\prec_{\mathcal{G}}$ coincides with $\prec_T$ from Construction 1 if all functions $\mathcal{L}_{i,j}(\boldsymbol{z}_{i-1})$ achieve their maximal weight $d_{\leqslant i-1}$. We now have all we need to show Proposition 7.3.

Proof. (Proposition 7.3). From Lemma 7.3 we observe

$$\begin{aligned} \mathrm{wt}\left(x_{i,j}^{\alpha_{i,j}}\right) &= \alpha_{i,j}\mathrm{wt}\left(x_{i,j}\right) = d_{\leqslant i-1} \\ &\geqslant \mathrm{wt}(\mathrm{LM}(\boldsymbol{z}_{i-1})) \geqslant \mathrm{wt}(\mathrm{LM}(\mathcal{L}_{i,j}\left(\boldsymbol{z}_{i-1}\right))). \end{aligned}$$

Hence $\mathrm{LM}(f_{i,j}) = x_{i,j}^{\alpha_{i,j}}$. Moreover, Lemma 7.3 also guarantees that

$$\mathrm{wt}(\mathrm{LM}(g)) \leqslant \mathrm{wt}(\mathrm{LM}(\boldsymbol{z}_r)) \leqslant d_{\leqslant r}.$$

Due to the fact that $\alpha_{i,j} \geqslant 2$, the factor $1/\alpha_{i,j}$ that appears in the weight of all variables $\boldsymbol{x}_i$, $i \geqslant 1$, the above equality can only be achieved by the monomial $x_0^{d_{\leqslant r}}$. It then follows from the assumption that $\mathrm{LM}(g) = x_0^{d_{\leqslant r}}$, which makes $P_{\mathcal{G}}$ a FreeLunch system. □

7.1.4.3 Computing a reduced Gröbner basis for $\langle P_{\mathcal{G}} \rangle$ (`sysGen`)

We have just seen that computing a Gröbner basis for a given FreeLunch system $P_{\mathcal{G}}$ is – as the name suggests – free. There are, however, two practical concerns worth addressing. Firstly, while $P_{\mathcal{G}}$ is itself a Gröbner basis, it is generally not the unique reduced Gröbner basis w.r.t. any of its FreeLunch orders. Secondly, generating the polynomials in $P_{\mathcal{G}}$ may itself be hard.

In practice we do not generate the polynomials in $P_{\mathcal{G}}$ in the direct manner outlined earlier. Rather, we construct a related polynomial system iteratively while reducing as many monomials as possible along the way. More formally, for a polynomial h and an ordered sequence of polynomials H, we let $\mathrm{Red}(h, H)$ denote the operation of reducing h by H (according to a specified monomial order). That is, $\mathrm{Red}(h, H)$ is the remainder after performing multivariate division of h by H (see [CLO97, Ch. 2, §3]). Because H is not necessarily a Gröbner basis, the remainder may change if the order of the functions in H changes. For a tuple of polynomials $\boldsymbol{h} = (h_1, \ldots, h_t)$, we write $\mathrm{Red}(\boldsymbol{h}, H) = (\mathrm{Red}(h_1, H), \ldots, \mathrm{Red}(h_t, H))$. Now fix a monomial order, and define $\boldsymbol{z}'_0 = \boldsymbol{z}_0$. We generate $\boldsymbol{p}'_i = (p'_{i,1}, \ldots, p'_{i,\ell_i})$ and the reduced states $\boldsymbol{z}'_i$ recursively as follows for $1 \leqslant i \leqslant r$ and $1 \leqslant j \leqslant \ell_i$.

$$p'_{i,j} \in \mathrm{Red}\left(x_{i,j}^{\alpha_{i,j}} - \mathcal{L}_{i,j}(\boldsymbol{z}'_{i-1}), \{\boldsymbol{p}'_1 \ldots, \boldsymbol{p}'_{i-1}\}\right),$$
$$\boldsymbol{z}'_i \in \mathrm{Red}\left(G_i(\boldsymbol{z}'_{i-1}, \boldsymbol{x}_i), \{\boldsymbol{p}'_1, \ldots, \boldsymbol{p}'_i\}\right),$$

where $\mathcal{L}_{i,j}$ is the polynomial from (7.2). Finally, we define

$$g' \in \mathrm{Red}\left([G_r(\boldsymbol{z}'_{r-1}, \boldsymbol{x}_r)]_1, \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_r\}\right),$$

and write $P'_{\mathcal{G}} = \{\boldsymbol{p}'_1, \ldots, \boldsymbol{p}'_r, g'\}$. Since the construction of $P'_{\mathcal{G}}$ only differs from that of $P_{\mathcal{G}}$ by reductions with generators in the ideal $I_{\mathcal{G}} = \langle P_{\mathcal{G}} \rangle$ their ideals should, intuitively speaking, be identical. This intuition is confirmed by the following lemma.

Lemma 7.4. *For any fixed monomial order we have*

$$I_{\mathcal{G}} = \langle P_{\mathcal{G}} \rangle = \langle P'_{\mathcal{G}} \rangle .$$

Proof. For any polynomial h and polynomial sequence H, we can write the reduction operation as $\mathrm{Red}(h, H) = h + W$, for some polynomial $W \in \langle H \rangle$. Since the G_i's

used in the construction of $\boldsymbol{p}'_i$ and $\boldsymbol{z}'_i$ are polynomial functions, one can show by induction that

$$p'_{i,j} = p_{i,j} + \langle\{\boldsymbol{p}_1 \ldots, \boldsymbol{p}_{i-1}\}\rangle, \qquad z'_{i,j} = z_{i,j} + \langle\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_i\}\rangle \tag{7.4}$$

holds for all $1 \leqslant i \leqslant r$ and $1 \leqslant j \leqslant \ell_i$. In particular, we have $g' = g + \langle\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_r\}\rangle$. Thus it is clear that $P_{\mathcal{G}}$ and $P'_{\mathcal{G}}$ generate the same polynomial ideal. □

The following result relates $P'_{\mathcal{G}}$ and $P_{\mathcal{G}}$ when Proposition 7.3 holds. Recall that we write $d_{\leqslant i} = d_1 \cdots d_i$, where $d_i = \deg(G_i)$.

Proposition 7.4. *Let $P_{\mathcal{G}}$ satisfy the condition of Proposition 7.3. Then constructing $P'_{\mathcal{G}}$ w.r.t. $\prec_{\mathcal{G}}$ is also a FreeLunch system. Moreover, replacing g' in $P'_{\mathcal{G}}$ with $g'/LC(g')$ yields the unique reduced Gröbner basis for $I_{\mathcal{G}}$ w.r.t. $\prec_{\mathcal{G}}$.*

Proof. By definition of polynomial division, we have $\mathrm{wt}(\mathrm{LM}(\mathrm{Red}(h, H))) \leqslant \mathrm{wt}(\mathrm{LM}(h))$. Since $\mathrm{LM}(p_{i,j})'$ cannot be reduced by $\{\boldsymbol{p}'_1 \ldots, \boldsymbol{p}'_{i-1}\}$ under $\prec_{\mathcal{G}}$, it follows from (7.4) and the prior discussion that $\mathrm{LM}(p'_{i,j}) = \mathrm{LM}(p_{i,j})$. For the similar statement on $\mathrm{LM}(g')$, we note that the condition $\mathrm{LM}(g) = x_0^{d_{\leqslant r}}$ can only hold if for every i there exist a j_i such that $\mathrm{LM}(z_{i,j_i}) = x_0^{d_{\leqslant i}}$. By construction of $\prec_{\mathcal{G}}$, this monomial will not be reduced by $\{\boldsymbol{p}'_1 \ldots, \boldsymbol{p}'_{i-1}\}$. Again, it follows from (7.4) that $\mathrm{LM}(z'_{i,j_i}) = x_0^{d_{\leqslant i}}$. In particular, $\mathrm{LM}(g') = \mathrm{LM}(g)$, hence $P'_{\mathcal{G}}$ is also a FreeLunch system.

For the last assertion, one observes from the way $p_{i,j}$ only depends on the variables $x_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, x_{i,j}$ that

$$\mathrm{Red}(p'_{i,j}, P'_{\mathcal{G}} \backslash \{p'_{i,j}\}) = \mathrm{Red}(p'_{i,j}, \{\boldsymbol{p}_1 \ldots, \boldsymbol{p}_{i-1}\}) \ ,$$

holds for $\prec_{\mathcal{G}}$. Hence $P'_{\mathcal{G}}$ is already fully reduced, and replacing g' with $g'/\mathrm{LC}(g')$ makes all polynomials monic. □

Remark 7.1. Recall from Proposition 6.6 that if H is a Gröbner basis for $\langle H\rangle$, then $\mathrm{Red}(h, H)$ does not depend on the order of the sequence H. It follows from Proposition 7.4 that if $P_{\mathcal{G}}$ satisfies the condition of Proposition 7.3, then the reductions in the construction of $p'_{i,j}$, $\boldsymbol{z}'_i$ and g' are independent of the order of the sequence $\{\boldsymbol{p}_1, \ldots \boldsymbol{p}_i\}$, w.r.t. $\prec_{\mathcal{G}}$.

Complexity of computing $P'_{\mathcal{G}}$. We are left with bounding the complexity of computing $P'_{\mathcal{G}}$, which will yield our estimate for the `sysGen` step. In the setting we will be interested in, this is expected to be dominated by the cost of applying the last round function G_r to compute g', and its reduction by $\{\boldsymbol{p}'_1 \ldots, \boldsymbol{p}'_r\}$. Our insight is that the reductions involved in the `sysGen` process are cheaper than the reductions required in `matGen`, since the reductions are performed on a smaller Gröbner basis; but we do not have a proof for such a statement. However, it is possible to bound the cost of the multiplications performed on the state $\boldsymbol{z}'_{r-1}$ when applying G_r. Let m denote the number of these multiplication, where we recall that m is typically

small by design. We reduce by $\{\boldsymbol{p}'_1 \dots, \boldsymbol{p}'_r\}$ after each multiplication, and will assume that this reduction is negligible compared to the cost of the multiplications themselves. Thus we have m multiplications of multivariate polynomials of maximal degree $d_{\leqslant r}$ in x_0[3] and $\alpha_{i,j} - 1$ in $x_{i,j}$, for $1 \leqslant i \leqslant r$, $1 \leqslant j \leqslant \ell_i$. We can then use the Kronecker trick presented by Moenck [Moe76, Section 3.4] to perform these multiplications in an efficient manner. In short, the Kronecker trick starts by transforming the multivariate polynomials to univariate polynomials. This allows us to perform the multiplication using an efficient univariate multiplication algorithm, before converting the result back to a multivariate polynomial. Moenck describes the algorithm and proves its correctness for any bound on the degree of each variable in both polynomials in the input of multiplication, but only gives a complexity estimate when all bounds are equal. It is, however, easy to verify that the complexity formula for the multivariate multiplication algorithm in our setting will be:

$$\tilde{\mathcal{O}}(d_{\leqslant r} \prod_{\substack{1 \leqslant i \leqslant r \\ 1 \leqslant j \leqslant \ell_i}} 2\alpha_{i,j}) \ ,$$

when applying either the Fast Fourier Transform, or Schönhage & Strassen's algorithm to perform the univariate multiplication [GG13, Chapter 8]. Repeating this m times yields our estimate for cost of multiplications in the `sysGen` step:

$$\tilde{\mathcal{O}}(m d_{\leqslant r} \prod_{\substack{1 \leqslant i \leqslant r \\ 1 \leqslant j \leqslant \ell_i}} 2\alpha_{i,j}) \ .$$

In comparison, recall that the `polyDet` step of our analysis is expected by Theorem 7.1 to require

$$\tilde{O}(d_{\leqslant r}(\prod_{\substack{1 \leqslant i \leqslant r \\ 1 \leqslant j \leqslant \ell_i}} \alpha_{i,j})^{\omega})$$

operations in $\mathbb{F}$. Thus, when m remains small, we do not expect the multiplications in `sysGen` to be the bottleneck of the overall attack.

Use in Experiments. We implemented the Kronecker trick for the experiments we ran with the Flint library [tea23], using the NTL library [Sho] for the univariate multiplication; the mapping between flint and NTL polynomial representations was performed by hand. The multivariate multiplications performed for experiments with `Magma` and `SageMath` used their own built-in functionalities.

7.1.5 Summary of the FreeLunch Attack

The strategy of the attack presented in this section is summarized in Algorithm 7.1.

The initial condition is that there exists a FreeLunch system associated with the target primitives. Methods for constructing this FreeLunch system were presented in Section 7.1.3 and Section 7.1.4, and the complexities for `sysGen` using

[3] This follows from Lemma 7.3 with $i = r$, and $\mathrm{LM}(\boldsymbol{z}'_r) \prec_{\mathcal{G}} \mathrm{LM}(\boldsymbol{z}_r)$.

these methods were discussed in Section 7.1.4.3. A different way of generating a FreeLunch system will also be shown in Section 7.3. We will estimate the complexity of `polyDet` by Equation 7.1, but we do not have a clear estimate for `matGen`. The final step `uniSol` recovers the roots of a univariate polynomial of degree D_I. This costs $\tilde{\mathcal{O}}(D_I)$ operations and is thus negligible in comparison with the earlier steps. We expect the complexity of the attack as a whole to be dominated by either `matGen` or `polyDet` for the primitives we have investigated. This is in line with our experiments (see Section 7.4.1), where `matGen` seems to be the dominating step for larger instances.

The numbers for the complexity of the `polyDet` step in our attacks against several AO permutations are shown are in Table 7.1[4]. Details of how we obtained them will be provided further in further sections.

Name	α/e	Number of branches						
		2	3	4	5	6	8	$\geqslant 12$
`Griffin`	3	∅	120 (16)	112 (15)	∅	∅	76 (11)	64 (10)
	5	∅	141 (14)	110 (11)	∅	∅	81 (9)	74 (9)
`Arion`	3	∅	128 (6)	134 (6)	114 (5)	119 (5)	98 (4)	∅
	5	∅	132 (6)	113 (5)	118 (5)	122 (5)	101 (4)	∅
α-`Arion`	3	∅	104 (5)	84 (4)	88 (4)	92 (4)	98 (4)	∅
	5	∅	83 (4)	87 (4)	91 (4)	94 (4)	101 (4)	∅
`Anemoi`	3	118 (21)	∅	-	∅	-	-	-
	5	156 (21)	∅	-	∅	-	-	-
	7	174 (20)	∅	-	∅	-	-	-
	11	198 (19)	∅	-	∅	-	-	-

Table 7.1: Theoretical time complexity ($\log_2$) of `polyDet` in FreeLunch-based attacks against some full-round algorithms (aiming at 128-bit security). Number of rounds in parentheses, ∅ corresponds to undefined algorithms. The α/e column reports α for `Griffin` and `Anemoi`; and e for the `Arion` variants.

7.2 Using FreeLunch Systems Directly

Experimental Verification. In this section and in Section 7.3, we support theoretical attacks with practical experiments on reduced-round versions. All experiments are performed on 1 core of AMD EPYC 7352 (2.3GHz) with 250 GB of memory, and on $\mathbb{F}_p$ with $p =$ `0x64ec6dd0392073`. The `sysGen` step is performed with `SageMath` [Sag9522], `Magma` [BCP97] or the NTL [Sho] and Flint [tea23] libraries, the `matGen` step is performed with Flint, and the `polyDet` step is perfomed with the Polynomial Matrix Library [The23; HNS19].

[4]The complexities correspond to the number of basic $\mathbb{F}_q$ operations; writing them as number of calls to the primitive would yield lower but hard to compute numbers.

7.2.1 A Detailed Example: `Griffin`

7.2.1.1 Specification of `Griffin`

`Griffin` [GHR+23] is a family of sponge hash and compression functions proposed by Grassi *et al.* at CRYPTO 2023 designed to be used in Zero-Knowledge applications. As such, it makes use of the internal permutation `Griffin-`π, which is defined over the finite field $\mathbb{F}_q$.

Each round function of `Griffin-`π is composed of a non-linear layer, the addition of a round constant, and a linear layer defined by multiplication by an MDS matrix.The specific features of `Griffin` impose that the primitive is only suitable for $\mathbb{F}_q^t$ where $t = 3$ or t is a multiple of four.

Definition 7.3 (Non-linear layer of `Griffin-`π)**.** Let $\alpha \in \{3, 5, 7, 11\}$ be the smallest integer such that $\gcd(\alpha, p-1) = 1$, $p > 2^{63}$ and let t be the number of branches. For $0 \leqslant i \leqslant t-1$, let $(\delta_i, \mu_i) \in \mathbb{F}_q^2 \backslash \{(0,0)\}$ be pairwise distinct such that $\delta_i^2 - 4\mu_i$ is a quadratic nonresidue modulo p. Then, the non-linear layer of `Griffin-`π is $S(x_0, \ldots, x_{t-1}) = (y_0, \ldots, y_{t-1})$, where each y_i is defined by the equations:

$$y_i := \begin{cases} x_0^{1/\alpha} & \text{if } i = 0, \\ x_1^{\alpha} & \text{if } i = 1, \\ x_2 \cdot (L_2(y_0, y_1, 0)^2 + \delta_2 \cdot L_2(y_0, y_1, 0) + \mu_2) & \text{if } i = 2, \\ x_i \cdot (L_i(y_0, y_1, x_{i-1})^2 + \delta_i \cdot L_i(y_0, y_1, x_{i-1}) + \mu_i) & \text{otherwise,} \end{cases}$$

for $L_i(z_0, z_1, z_2) = (i-1) \cdot z_0 + z_1 + z_2$.

Definition 7.4 (`Griffin-`π)**.** Let r be the number of rounds, and for $1 \leqslant i \leqslant r-1$ let $\mathbf{c}^{(i)} \in \mathbb{F}_q^t$ be a constant vector (we assume $\boldsymbol{c}^{(r)} = 0$). Then `Griffin-`π $\mathcal{G}^\pi : \mathbb{F}_q^t \to \mathbb{F}_q^t$ is defined as

$$\mathcal{G}^\pi(\cdot) := \mathcal{F}_r \circ \cdots \circ \mathcal{F}_2 \circ \mathcal{F}_1(M \times \cdot),$$

where for $1 \leqslant i \leqslant r$, the i-th round function $\mathcal{F}_i$ is defined as

$$\mathcal{F}_i(\cdot) = \boldsymbol{c}^{(i)} + M \times S(\cdot),$$

for $M \in \mathbb{F}_q^{t \times t}$ a matrix, and S the non-linear layer of `Griffin-`π.

The first round function of `Griffin-`π for $t = 4$ is depicted in Figure 7.2 where, to simplify the construction, we denote by F_i the last two equations of Definition 7.3. The authors proposed various instances with a 128-bit security claim. The number of branches varies from 3 to 24 (though not all values are possible), and the number of rounds is computed for different degrees α based on the complexity of finding a Gröbner basis using the basic encoding as it was the most efficient attack they could find.

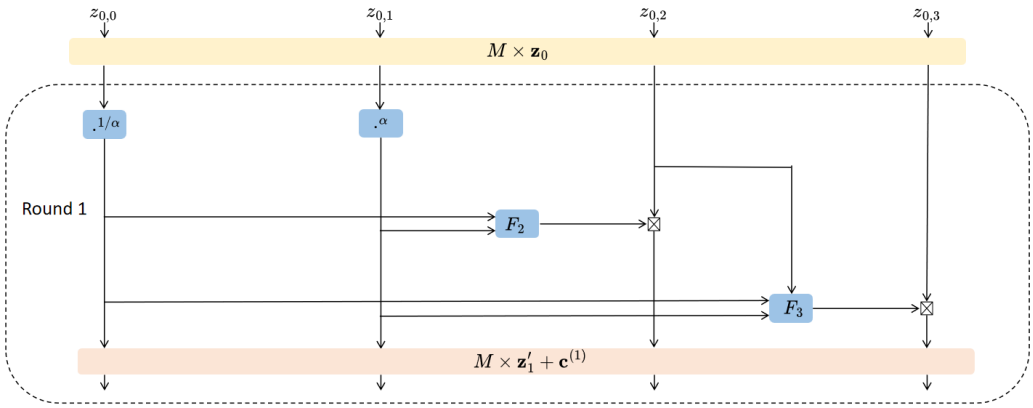


Figure 7.2: First round function of `Griffin`-π with $t = 4$.

7.2.1.2 A FreeLunch system for `Griffin`

We observe that the round function of `Griffin` readily lends itself to a naïve construction of the system $P_{\mathcal{G}}$, as described in Section 7.1.4. Indeed, for each round i we can simply define $\boldsymbol{z}_i = G_i(\boldsymbol{z}_{i-1}, x_i)$ by $\boldsymbol{z}_i = \boldsymbol{c}^{(i)} + M \times \boldsymbol{z}'_i$, where $z'_{i,0} = x_i$ and $z'_{i,j}$ given as y_j, for $1 \leqslant j < t$, in Definition 7.3 of the i-th round. Note that G_i will be of degree at most $d_i = 2\alpha + 1$. Under the assumption that the polynomial g in $P_{\mathcal{G}}$ satisfies the monomial property of Proposition 7.3, we get an associated ideal degree of $(\alpha\,(2\alpha + 1))^r$.

Remark 7.2. Note that the naïve modeling $P_{\mathcal{G}}$ given above for `Griffin` is not new; in fact, it was proposed by the authors of this algorithm for their initial security analysis [GHR+23, Section 6.2]. However, the authors did not attempt to compute a Gröbner basis for $\langle P_{\mathcal{G}} \rangle$ in a FreeLunch order, but rather in the usual *grevlex* order. They estimate that computing a Gröbner basis in this latter monomial order well exceeds the security level for the suggested number of rounds.

7.2.1.3 Bypassing several rounds

A further improvement is constructing an affine input in x_0 for the CICO problem that is tailored to bypass the inversion operation for a few initial rounds. This effectively means that fewer variables x_i are necessary, which in turn has a significant impact on the resulting ideal degree. The only difference is that we choose a different sequence of polynomial functions $\mathcal{G}^*$, where G_1^* effectively spans several rounds but only depends on $\boldsymbol{z}_0$, and does not necessitate the introduction of a new variable x_1. The ensuing functions G_i^*, $i \geqslant 2$, can be constructed following the above approach (though there will be fewer of them). The exact number of initial rounds we can bypass will depend on t, where a larger t generally allows us to bypass more rounds[5]. For $t \geqslant 12$ branches, we can find an easily computable set of input

[5] A similar observation of bypassing rounds was already considered in [GHR+23, Section 6.2]. However, the authors only describe a method for bypassing a single round for $t = 3$ and do not

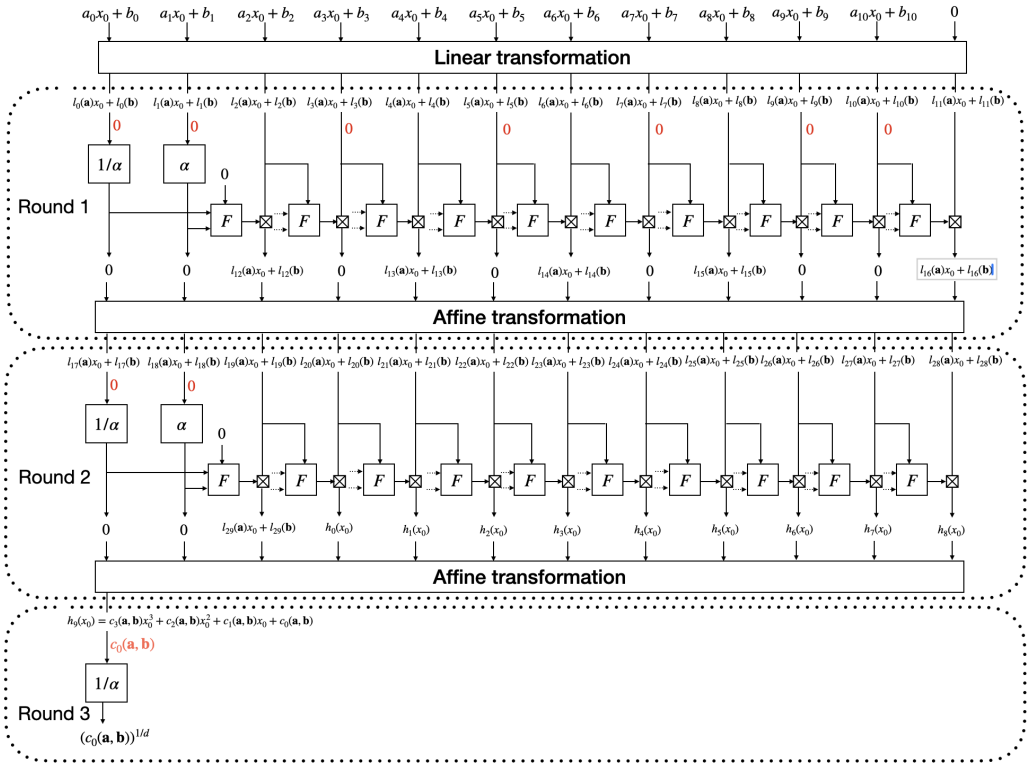


Figure 7.3: Evolution of chosen set of input states to `Griffin` with 12 branches. Red values give conditions on the a_i and b_j such that the input of $x^{1/\alpha}$ in the third round becomes a known constant independent of x_0.

states that allows to bypass the first three rounds of `Griffin`, so that the full fourth round state can be expressed as low degree polynomials in x_1. We explain in detail how this can be done for $t = 12$, and the result can trivially be extended to $t \in \{16, 20, 24\}$, and more subtly to $t < 12$.

Denote the input state to `Griffin` as follows:

$$(a_0x_0 + b_0, a_1x_0 + b_1, a_2x_0 + b_2, \ldots, a_{10}x_0 + b_{10}, 0).$$

The a_i and b_i are constants in $\mathbb{F}_q$ that we now proceed to determine. We look for values of a_i and b_i such that the input to the $x^{1/\alpha}$ function in the three first rounds does not depend on x_0. Figure 7.3 illustrates the evolution of one of the chosen input states up to the start of round 3.

The values of a_i and b_i can be determined as follows. After the initial linear transformation before the first round, all branches can be expressed as $l_i(\boldsymbol{a})x_0 + l_i(\boldsymbol{b})$ for $0 \leqslant i \leqslant 11$, where $l_i(\cdot)$ is a known linear combination. To get 0 on the branches indicated in Figure 7.3, the a_i's and b_j's need to satisfy the following linear equations

consider the effect of having a larger t.

$$\begin{aligned} l_0(\boldsymbol{a}) &= 0, & l_0(\boldsymbol{b}) &= 0, \\ l_1(\boldsymbol{a}) &= 0, & l_1(\boldsymbol{b}) &= 0, \\ l_3(\boldsymbol{a}) &= 0, & l_3(\boldsymbol{b}) &= 0, \\ l_5(\boldsymbol{a}) &= 0, & l_5(\boldsymbol{b}) &= 0, \\ l_7(\boldsymbol{a}) &= 0, & l_7(\boldsymbol{b}) &= 0, \\ l_9(\boldsymbol{a}) &= 0, & l_9(\boldsymbol{b}) &= 0, \\ l_{10}(\boldsymbol{a}) &= 0, & l_{10}(\boldsymbol{b}) &= 0. \end{aligned}$$

With 0 on any two adjacent branches, the input to F will either be all 0, with $F(0,0,0)$ being equal to a constant, or the output of F will be multiplied with 0, making sure the value on the branch remains 0. This ensures that the algebraic expressions on the branches stay linear in $x_0, \boldsymbol{a}$ and $\boldsymbol{b}$ after the affine transformation at the start of the second round. The need to have input 0 to $x^{1/\alpha}$ and x^{α} in the second round gives four more linear constraints

$$\begin{aligned} l_{17}(\boldsymbol{a}) &= 0, & l_{17}(\boldsymbol{b}) &= 0, \\ l_{18}(\boldsymbol{a}) &= 0, & l_{18}(\boldsymbol{b}) &= 0, \end{aligned}$$

where the γ_i are known constants.

Before the affine transformation in the second round, most branches will have cubic polynomials in x_0 as their values (the $h_i(x_0)$ in Figure 7.3). These are again linearly mixed in the affine transformation at the end of round two, producing the cubic polynomial

$$h_9(x_0) = c_3(\boldsymbol{a},\boldsymbol{b})x_0^3 + c_2(\boldsymbol{a},\boldsymbol{b})x_0^2 + c_1(\boldsymbol{a},\boldsymbol{b})x_0 + c_0(\boldsymbol{a},\boldsymbol{b})$$

on the first branch. We want to enforce that $c_3(\boldsymbol{a},\boldsymbol{b}) = c_2(\boldsymbol{a},\boldsymbol{b}) = c_1(\boldsymbol{a},\boldsymbol{b}) = 0$ such that the input to the $x^{1/\alpha}$ function in round three becomes a known constant independent from x_0. The expressions for the coefficients are cubic in the a_i and b_j, but note that all the polynomials $h_i(x_0)$ for $0 \leqslant i \leqslant 8$ are made as products of linear factors as

$$(l_i(\boldsymbol{a})x_0 + l_i(\boldsymbol{b}))(l_j(\boldsymbol{a})x_0 + l_j(\boldsymbol{b}))(l_k(\boldsymbol{a})x_0 + l_k(\boldsymbol{b})),$$

and that $h_9(x_0)$ is a sum of these. By calculating the coefficients for the x_0^3, x_0^2, and x_0 terms, we see that $c_3(\boldsymbol{a},\boldsymbol{b})$ is cubic in $\boldsymbol{a}$, but does not contain $\boldsymbol{b}$ at all. Similarly, $c_2(\boldsymbol{a},\boldsymbol{b})$ is quadratic in $\boldsymbol{a}$ and linear in $\boldsymbol{b}$ and $c_1(\boldsymbol{a},\boldsymbol{b})$ is linear in $\boldsymbol{a}$ and quadratic in $\boldsymbol{b}$.

We can now use the 9 linear equations in $\boldsymbol{a}$ introduced above to eliminate $a_2, \ldots, a_{10}$ from $c_3(\boldsymbol{a})$. This leaves c_3 as $c_3(a_0, a_1)$, a cubic expression in a_0 and a_1. Next we fix a_1 to an arbitrary non-zero value (to avoid the trivial solution $a_0 = \ldots = a_{10} = 0$) and solve for $c_3(a_0) = 0$ using a root-finding algorithm for univariate polynomials. With a_0 and a_1 fixed, all the other a_i gets fixed as well from the linear constraints from rounds 1 and 2.

Once all a_i have been found, $c_2(\boldsymbol{a},\boldsymbol{b}) = 0$ just becomes a linear equation in $\boldsymbol{b}$. Using this linear equation together with the 9 from above, we can eliminate $b_1, \ldots, b_{10}$ from the last coefficient $c_1(\boldsymbol{a},\boldsymbol{b})$. With all the a_i fixed, c_1 then just becomes $c_1(b_0)$, a quadratic expression in b_0 and we easily solve $c_1(b_0) = 0$. This determines all the values for the b_i.

With the a_i and b_j now fixed, we know that the input state from our chosen set will generate polynomials in x_0 of degree $6\alpha + 3$ on the branches at the start of round 4. We can then start the basic attack from there, adapting the weighted order of the variables accordingly. When the number of x_i-variables is reduced by 3 and with the degree of x_0 bounded to $6\alpha + 3$ until the fourth round, the dimension of the Gröbner basis ideal becomes much smaller, which again reduces the overall attack complexity significantly.

When there are more than 12 branches we can do the exact same trick as explained above. The only difference is that there will be more values of a_i and b_j that can be chosen arbitrarily when solving for $c_3(\boldsymbol{a}, \boldsymbol{b}) = c_2(\boldsymbol{a}, \boldsymbol{b}) = c_1(\boldsymbol{a}, \boldsymbol{b}) = 0$. When there are less than 12 branches, there is not enough degrees of freedom to make it through the third round. For $t = 8$ we can bypass the two first rounds, so x_1 only needs to be introduced in round 3, and for $t = 3, 4$ it is possible to bypass the first round and introduce x_1 in round 2.

Summary. For $t = 3, 4$, we can bypass one round with linear functions in $\boldsymbol{z}_0$: we get $r-1$ equations with respective leading terms $x_1^\alpha, \dots x_{r-1}^\alpha$ and one equation with leading term $x_0^{(2\alpha+1)^{r-1}}$. For $t = 8$, we are able to bypass two rounds with cubic functions in $\boldsymbol{z}_0$: we get $r-2$ equations with respective leading terms $x_1^\alpha, \dots x_{r-2}^\alpha$ and one equation with leading term $x_0^{3(2\alpha+1)^{r-1}}$. For $t \geqslant 12$, three rounds can be bypassed with $\deg(\boldsymbol{z}_0) = 6\alpha + 3$: we get $r-3$ equations with respective leading terms $x_1^\alpha, \dots x_{r-3}^\alpha$ and one equation with leading term $x_0^{3(2\alpha+1)^{r-2}}$. We therefore get the following parameters:

$$D_{I,t} = \begin{cases} (\alpha\,(2\alpha+1))^{r-1}\,, & \text{for } t = 3, 4, \\ 3\,(\alpha\,(2\alpha+1))^{r-2}\,, & \text{for } t = 8, \\ (6\alpha+3)\,(\alpha\,(2\alpha+1))^{r-3}\,, & \text{for } t \geqslant 12\,. \end{cases} \tag{7.5}$$

$$D_{H,t} = \begin{cases} \alpha^{r-1}, & \text{for } t = 3, 4, \\ \alpha^{r-2}, & \text{for } t = 8, \\ \alpha^{r-3}, & \text{for } t \geqslant 12\,. \end{cases} \tag{7.6}$$

7.2.1.4 Complexity analysis and experimental results

We can now use the machinery described in Section 7.1.2 to solve the above-described FreeLunch system for `Griffin`. As noted in Section 7.1.2, it is hard to theoretically estimate the complexity of `matGen` where one computes the multiplication matrix T_0. On the other hand, based on previous analysis, we estimate the complexity of `polyDet`as $\tilde{\mathcal{O}}(D_{I,t} D_{H,t}^{\omega-1}) = \tilde{\mathcal{O}}(D_{I,t}(D_{I,t}/\alpha_0)^{\omega-1})$. As a consequence, the running time for `polyDet` becomes

$$\tilde{\mathcal{O}}(D_{I,t} D_{H,t}^{\omega-1}) = \begin{cases} \tilde{\mathcal{O}}((\alpha^\omega\,(2\alpha+1))^{r-1}), & \text{for } t = 3, 4, \\ \tilde{\mathcal{O}}(3\,(\alpha^\omega\,(2\alpha+1))^{r-2}), & \text{for } t = 8, \\ \tilde{\mathcal{O}}((6\alpha+3)\,(\alpha^\omega\,(2\alpha+1))^{r-3}), & \text{for } t \geqslant 12\,. \end{cases} \tag{7.7}$$

Table 7.2: Expected time complexity of `polyDet` for the different full-round instances of `Griffin`, where $\omega = 2.81$. Number of rounds in parentheses.

Branches	Complexity ($\log_2$)	
	$\alpha = 3$	$\alpha = 5$
3	120 (16)	141 (14)
4	112 (15)	110 (11)
8	76 (11)	81 (9)
12,16,20,24	64 (10)	74 (9)

Table 7.3: Experimental results on `Griffin` with $(t, \alpha) = (12, 3)$. `sysGen` uses Flint and NTL with the fast multivariate multiplication algorithm of Section 7.1.4.3. The complexity of `polyDet` is estimated using the bounds of Theorem 7.1 with the logarithmic factors.

Number of rounds	Complexity of `polyDet`	Time (s)			Memory (MB)
		`sysGen`	`matGen`	`polyDet`	
5	26	0.17	0.02	0.53	14
6	34	4.0	6.67	50.78	471
7	41	2,558	3,361	5,727	27,600

The logarithmic factors can be included to give a complexity of approximately $D_{I,t}(D_{I,t}/\alpha_0)^{\omega-1}\log(\alpha_0)^2$ in the general case. The resulting estimated time complexities (including the logarithmic factors) of running `polyDet`for the proposed instances of `Griffin` are listed in Table 7.2. Experimental results are presented in Table 7.3 and discussed in Section 7.4.1.

7.2.2 Applicability Beyond `Griffin`: `ArionHash`

7.2.2.1 Specification of `ArionHash`

`ArionHash` [RST23] is an arithmetization-oriented hash function proposed by Roy *et al.* that, much like `Griffin`, uses a permutation as its core primitive. Called `Arion`-π, this permutation utilizes in each round a polynomial of very high degree in one branch and low degree polynomials in the remaining branches to significantly decrease the number of necessary rounds to achieve the desired security.

Definition 7.5 (Non-linear layer of `Arion`-π)**.** Let $p \geqslant 5$ be a prime, t the number of branches, e the smallest positive integer be such that $\gcd(e, p-1) = 1$, and $121 \leqslant \alpha \leqslant 257$ an integer such that $\gcd(\alpha, p-1) = 1$.

For $0 \leqslant i \leqslant t-2$, let $\delta_{i,1}, \delta_{i,2}, \mu_i \in \mathbb{F}_q$ be such that $g_i(x) = x^2 + \delta_{i,1} \cdot x + \delta_{i,2}$ is a quadratic function without zeroes in $\mathbb{F}_q$ and define $h_i(x) = x^2 + \mu_i \cdot x$. Then the *non-linear layer of `Arion`-π* is $\mathcal{S} = \{f_0, \ldots, f_{t-1}\}$, where each f_i is defined

"from-right-to-left" by the equations:

$$f_{t-1}(y_0, \ldots, y_{t-1}) = y_{t-1}^{1/\alpha},$$
$$f_i(y_0, \ldots, y_{t-1}) = y_i^e \cdot g_i(\sigma_{i,t}) + h_i(\sigma_{i,t}), \qquad t-2 \geqslant i \geqslant 0,$$

where $\sigma_{i,t}$ represents the sum of all previously computed inputs and outputs

$$\sigma_{i,t} = \sum_{j=i+1}^{t-1} y_j + f_j(y_0, \ldots, y_{t-1}) \ .$$

Definition 7.6 (`Arion`-π)**.** Let r be the number of rounds, and for $1 \leqslant i \leqslant r$ let $\mathbf{c}_i \in \mathbb{F}_q^t$ be a constant vector. Then `Arion`-π is defined as the following composition over $\mathbb{F}_q^t$:

$$\texttt{Arion-}\pi : (y_0, \ldots, y_{t-1}) \mapsto (L_{c_r} \circ \mathcal{S}_r) \circ \cdots \circ (L_{c_1} \circ \mathcal{S}_1) \circ L_0(y_0, \ldots, y_{t-1}),$$

where L_{c_i} is the affine map of [RST23, Definition 3] and $\mathcal{S}_i$ is the non-linear layer of `Arion`-π, for $1 \leqslant i \leqslant r$.

We illustrate the construction of the first round of `Arion`-π in Figure 7.4 for $t = 4$ where, for the sake of clarity, we only represent the function f_i of the Generalized Triangular Dynamical System (GTDS) without the details of g_i and h_i.

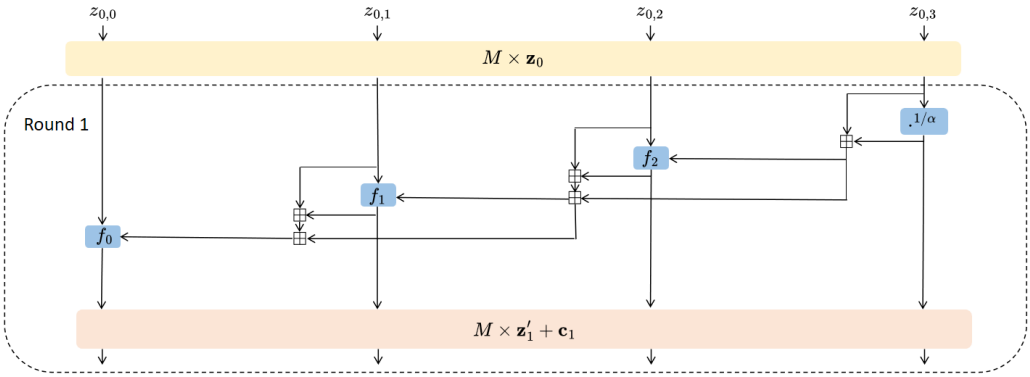


Figure 7.4: First round function of `Arion`-π with $t = 4$.

We provide the parameters for `Arion`-π and `ArionHash` as well as for their additionally proposed aggressive versions α-`Arion` and α-`ArionHash` with $e = 3, 5$ and $\alpha = 121$ in Table 7.4 (number of rounds are in parenthesis). The authors claim 128-bit security for each parameter set.

7.2.2.2 FreeLunch system for `ArionHash`

Due to the similarities in construction between `Arion`-π and `Griffin`-π, it comes as no surprise that the round function of `Arion`-π also fits the naïve construction of the

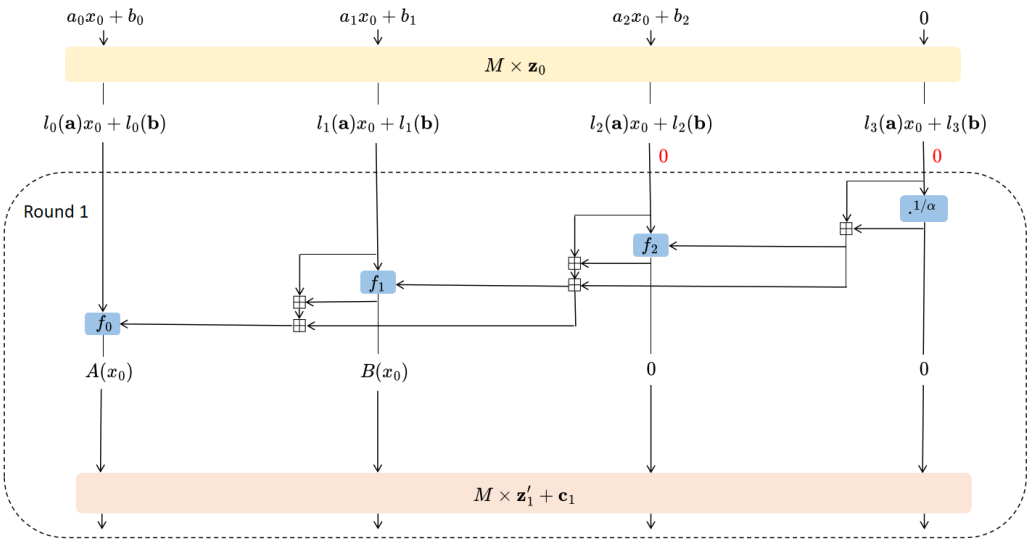


Figure 7.5: Evolution of chosen set of input states to `Arion`-π with 4 branches. Red values give conditions on the a_i and b_j.

system $P_{\mathcal{G}}$ described in Section 7.1.4. In this case, for each round i we define $\boldsymbol{z}_i = G_i(\boldsymbol{z}_{i-1}, x_i)$ by $\boldsymbol{z}_i = L_{c_i}(\boldsymbol{z}_i')$, where $z_{i,t-1}' = x_i$ and $z_{i,j}' = f_j(z_{i-1,0}, \ldots, z_{i-1,t-2}, x_i^\alpha)$ for $t-2 \geqslant j \geqslant 0$. Note that each component of $\boldsymbol{z}_i'$ (and thus of $\boldsymbol{z}_i$) will have degree at most $d_i = (2^{t-1}(e+1) - e)^i$ in x_0. Assuming that the polynomial g in $P_{\mathcal{G}}$ satisfies the monomial property of Proposition 7.3, we get an associated ideal degree of $\left(\alpha\left(2^{t-1}(e+1) - e\right)\right)^r$.

In addition, one can further improve this technique by generating a set of input states constructed so that the inversion operation for the first round is bypassed, reducing the number of necessary variables and, consequently, the associated ideal degree.

Denote the input state to `Arion` as follows:

$$(a_0x_0 + b_0, a_1x_0 + b_1, a_2x_0 + b_2, \ldots, a_{t-2}x_0 + b_{t-2}, 0).$$

The a_i and b_i are constants in $\mathbb{F}_q$ that will be determined. We look for a_i and b_i such that the input to the first $x^{1/\alpha}$ does not depend on x_0. Figure 7.5 illustrates the evolution of one of the chosen input states up to the start of round 2.

The values of a_i and b_i can, in general, be determined as follows. After the initial matrix multiplication, all branches can be expressed as $l_i(\boldsymbol{a})x_0 + l_i(\boldsymbol{b})$ for $0 \leqslant i \leqslant t-1$, where $l_i(\boldsymbol{a})$ and $l_i(\boldsymbol{b})$ are known linear combinations. To get 0 on the last $t-2$ branches, the a_i's and b_j's need to satisfy the following linear equations

$$\begin{aligned} l_2(\boldsymbol{a}) &= 0, & l_2(\boldsymbol{b}) &= 0, \\ &\vdots & &\vdots \\ l_{t-1}(\boldsymbol{a}) &= 0, & l_{t-1}(\boldsymbol{b}) &= 0. \end{aligned}$$

With $2t-4$ equations on $2t-2$ variables, these constraints leave two degrees of freedom for the variables in $\boldsymbol{a}$ and $\boldsymbol{b}$. Naïvely, one could think of additionally

imposing the constraints $l_1(\boldsymbol{a}) = 0$ and $l_1(\boldsymbol{b}) = 0$ such that only the first branch is nonzero and the degree on x_0 is further reduced. However, the unique solution to this system is the trivial solution $(\boldsymbol{a}, \boldsymbol{b}) = (0, 0)$, which is not of interest. Thus, we avoid this by instead imposing arbitrary conditions for two variables a_k and b_l (as long as a_k is set to be a non-zero value to avoid the trivial solution). With a_k and b_l fixed, all the other variables get fixed, too, from the previous linear constraints. For simplicity, one could fix the values $a_0 = 1$ and $b_0 = 0$, leading to an input state of the form $(x_0, a_1x_0, a_2x_0, \ldots, a_{t-2}x_0, 0)$, where all a_i's are fixed.

With 0 on the last $t-2$ input branches, the output of the non-linear layer of `Arion`-π will be of the form $(A(x_0), B(x_0), 0, \ldots, 0)$, where A and B are polynomials in x_0 of degree $3e$ and e, respectively. Thus, the input state from our chosen set will generate polynomials in x_0 of degree $3e$ on the branches after the affine transformation in round 1. We can then start the basic attack from there, adapting the weighted order of the variables accordingly. When the number of x_i-variables is reduced by 1 and with the degree of x_0 bounded to $3e$ until the second round, the dimension of the Gröbner basis ideal becomes smaller, which again reduces the overall attack complexity.

Summary. For `Arion` we are only able to bypass a single round with $\deg(\boldsymbol{z}_0) = 3e$, independent of t. We get $r-1$ equations with respective leading terms $x_1^{\alpha}, \ldots x_{r-1}^{\alpha}$, and one equation of leading term $x_0^{3e(2^{t-1}(e+1)-e)^{r-1}}$. This gives the following parameters:

$$\begin{aligned} D_I &= 3e\left(\alpha\left(2^{t-1}(e+1)-e\right)\right)^{r-1}, \\ D_H &= \alpha^{r-1}. \end{aligned}$$

7.2.2.3 Complexity analysis and experimental results

We can now apply the new methods introduced in Section 7.1.2 to solve the FreeLunch system for `ArionHash`. Based on the general complexity analysis of the attack, we list the estimated time complexities of `polyDet` for the different proposed `ArionHash` parameters in Table 7.4. Note that here $D_H = D_I/\alpha_0 = \alpha^{r-1}$, so that the running time for `polyDet` becomes

$$\tilde{\mathcal{O}}(D_I D_H^{\omega-1}) = \tilde{\mathcal{O}}\left(3e\left(\alpha^{\omega}\left(2^{t-1}(e+1)-e\right)\right)^{r-1}\right).$$

Experimental results are presented in Table 7.5 and discussed in Section 7.4.1.

7.2.3 Last Example: `XHash8`

`XHash8` is a permutation proposed by Ashur, Kindi and Mahzoun in [AKM23]. Along with `XHash12`, it is a follow-up of Rescue-Prime Optimized (RPO) [AKM+22], itself a follow-up of Rescue-Prime [AKM+22].

	Arion-π & ArionHash		α-Arion& α-ArionHash	
Branches	Complexity ($\log_2$)		Complexity ($\log_2$)	
	$e = 3$	$e = 5$	$e = 3$	$e = 5$
3	128 (6)	132 (6)	104 (5)	83 (4)
4	134 (6)	113 (5)	84 (4)	87 (4)
5	114 (5)	118 (5)	88 (4)	91 (4)
6	119 (5)	122 (5)	92 (4)	94 (4)
8	98 (4)	101 (4)	98 (4)	101 (4)

Table 7.4: Expected time complexity ($\log_2$) of `polyDet` for the different full-round instances of `ArionHash`, where $\alpha = 121$ and $\omega = 2.81$. Number of rounds in parenthesis.

Number of branches	Complexity of `polyDet`	Time (s) `sysGen`	`matGen`	`polyDet`	Memory (MB)
3	32	1.31	< 0.01	6.8	3,387
4	33	1.46	0.07	18.7	7,551
5	35	9.54	0.08	64.5	15,903
6	36	247	0.31	215	32,626
8	39	24,872	4.86	2,545	134,165

Table 7.5: Experimental results on 2-round `Arion`, with $(e, \alpha) = (3, 121)$. `sysGen` is performed using `SageMath`. `polyDet` uses an evaluation/interpolation algorithm of pml [The23] since the algorithm of [LNZ17] implemented in pml does not work for the non-generic polynomial matrix in input of `polyDet`.

7.2.3.1 Description of `XHash8`

`XHash8` is an SPN with nonlinear S-boxes, multiplication by a fixed MDS matrix M, and addition by round constants C_i. Its state contains $t = 12$ elements in $\mathbb{F}_p$ where $p = 2^{64} - 2^{32} + 1$. The rate is fixed to 8 and capacity 4. There are 3 rounds in total, and each round consists of 3 steps, for a total of 9 steps (plus the initial affine layer (I)). With the cipher state denoted as $\boldsymbol{z} = (z_0, \ldots, z_{11})$, one round of `XHash8` is constructed from the following functions (excluding $(P3)^{(k)}$ which is specified below):

$$
\begin{aligned}
(I) &: \boldsymbol{z} \mapsto M \times (C_0 + \boldsymbol{z}),\\
(F)^{(k)} &: \boldsymbol{z} \mapsto C_{3k} + M \times (z_0^7, \ldots, z_{11}^7),\\
(B')^{(k)} &: \boldsymbol{z} \mapsto C_{3k+1} + (z_0^{\frac{1}{7}}, z_1, z_2^{\frac{1}{7}}, z_3^{\frac{1}{7}}, z_4, z_5^{\frac{1}{7}}, z_6^{\frac{1}{7}}, z_7, z_8^{\frac{1}{7}}, z_9^{\frac{1}{7}}, z_{10}, z_{11}^{\frac{1}{7}}).
\end{aligned}
$$

The last step of a round, $(P3)^{(k)}$, consists of naturally mapping $\boldsymbol{z}$ to a state of four elements in a cubic expansion $\mathbb{F}_{p^3}$, denoted $(S_{0,1,2}, S_{3,4,5}, S_{6,7,8}, S_{9,10,11})$, and then computing $S_{i,i+1,i+2}^7$ and mapping the result back to $\mathbb{F}_p$. After that, like with $(F)^{(k)}$, an MDS layer is applied, and the round constant C_{3k+2} is added. Effectively,

$(P3)^{(k)}$ is equivalent to mapping each z_{3q+r} to a multivariate polynomial of degree 7 in $z_{3q}, z_{3q+1}, z_{3q+2}$ (see also the detailed description in [AKM23, Appendix A]), which is the way we modelize it.

The steps are applied in the following order, from left to right:

$$(I)\ (F)^{(1)}(B')^{(1)}(P3)^{(1)}(F)^{(2)}(B')^{(2)}(P3)^{(2)}(F)^{(3)}(B')^{(3)}(P3)^{(3)}.$$

One round preceded by (I) is shown in figure 7.6, taken from [AKM23].

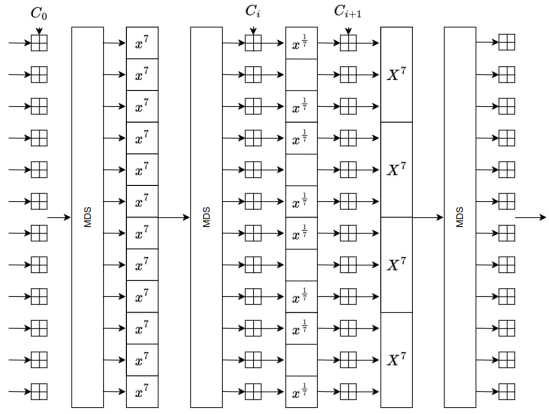


Figure 7.6: Round i of `XHash8` preceded by an (I) step: $(I)(F)^{(i)}(B')^{(i)}(P3)^{(i)}$.

7.2.3.2 A FreeLunch system for `XHash8`

Our resolution allows us to solve the CICO problem on one branch. However, since the size of one branch is roughly 64 bits, this CICO problem could simply be solved by making 2^{64} queries to the permutation, for which our solving algorithm does not give us an advantage. On top of that, the real capacity of `XHash8` is $c = 4$ for a security claim of 128 bits. Rather than claiming a full attack on `XHash8`, we show a special case where a FreeLunch system can be easily extracted. However, the later solving steps, in particular the `polyDet` step, will still have a very high complexity.

Following the construction of FreeLunch systems from Section 7.1 we define the initial state as $\boldsymbol{z}_0 = (x_0, 0, \ldots, 0)$ and add a new variable $x_{i,j}$ for $0 \leqslant i \leqslant 2$ and $j \in \{0, 2, 3, 5, 6, 8, 9, 11\}$ after every $(\cdot)^{1/7}$. All other nonlinear operations can be represented as polynomials of degree 7, fixing the weights of the introduced variables to

$$\mathrm{wt}(x_0) = 1, \qquad \mathrm{wt}(x_{i,j}) = 7^{2i} + 1\ .$$

We end up with 25 polynomials in 25 variables; 24 of these polynomials have $x_{i,j}^7$ as leading monomials and the last polynomial has $x_0^{7^6}$ as a leading monomial. The coefficient of the $x_0^{7^6}$-term in the last polynomial will be non-zero with a very high probability, ensuring we get a FreeLunch system, with $D_H = 7^{24}$ and $\alpha_0 = 7^6$.

7.2.3.3 Complexity and impact on security analysis

We can solve the system using the algorithm described in Section 7.1. The complexity of `matGen` is hard to estimate precisely. The complexity of the `polyDet` step is:

$$\mathcal{O}(D_H^\omega \alpha_0 \log(\alpha_0)^2) \approx 2^{214}$$

when $\omega = 2.81$. As this is significantly higher than brute force for the chosen p, we conclude that `XHash8` seems very secure against the FreeLunch attack.

That said, we note that the current security estimates for `XHash8` are (conservatively) extrapolated from scaled-down experiments with $t = 3$ using a single unknown input [AKM23, Appendix B]. While the FreeLunch framework cannot currently be extrapolated in a similar manner for the full construction, we still hope it could provide a basis for future insights into the security of `XHash8`. We note that the related constructions XHash12, RPO and Rescue-Prime all contain a layer of inversion operations in all branches, and hence we cannot directly obtain a FreeLunch from them.

7.3 Forcing the Presence of a FreeLunch for `Anemoi`

We have just seen three examples where the FreeLunch machinery of Section 7.1.4 could be readily applied. `Anemoi` is another class of permutations that rely on the inverse of low degree monomials in a finite field to achieve a high degree and so it would, a-priori, seem like another candidate where we can apply the FreeLunch techniques. However, we will see that this is not as straightforward as it may appear because a direct application of the technique creates a polynomial system $P_{\mathcal{G}}$ where g does not satisfy the assumption of Proposition 7.3. Instead, we will show how to compute a modified polynomial system $P_{\mathcal{G}*}$ that retains the valid solution to the CICO problem, which will turn out to be a FreeLunch system. This comes at the cost of a somewhat larger, yet still comparable, ideal degree than what was given in Conjecture 2 of [BBC+23]. We start by describing `Anemoi`.

7.3.0.1 Description of `Anemoi`.

The `Anemoi` permutations [BBC+23] operate on $\mathbb{F}_q^{2\ell}$ for $\ell \geqslant 1$, and either $q = 2^n$ with n odd, or $q = p$ for any prime $p \geqslant 3$. There are differences between the operations for the odd and even characteristic cases that will impact our later modeling. Thus, we focus on the setting of $\ell = 1$ and p prime, leaving the even case as future work. In odd characteristic, `Anemoi` takes a parameter α such that $x \mapsto x^\alpha$ is a permutation of $\mathbb{F}_q$, usually $\alpha = 3, 5, 7$ or 11. The original paper gives two specific hash function instances based on `Anemoi` with $\ell = 1$: `AnemoiSponge-BN-254`, with a 254-bit prime p, `AnemoiSponge-BLS12-381`, with a 381-bit prime p. 127 bits of security are claimed for both of these.

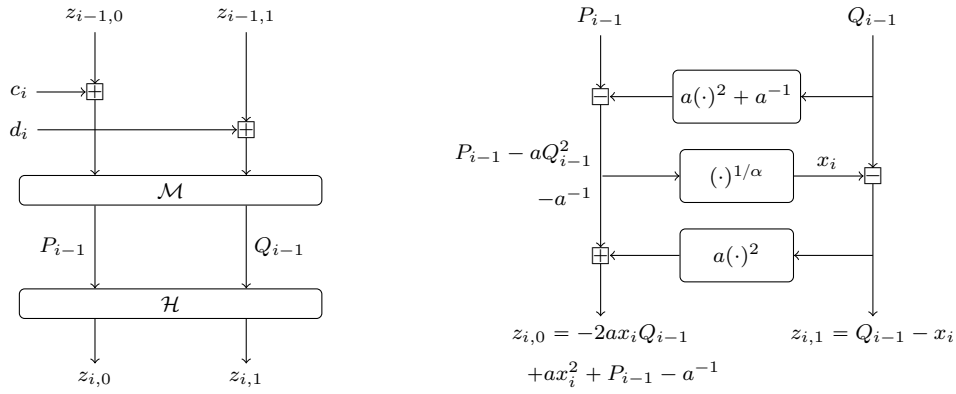
Definition 7.7 (Odd Anemoi with $\ell = 1$)**.** For a given p, α and number of rounds r, Anemoi is a permutation of $\mathbb{F}_p^2$ defined as

$$\texttt{Anemoi}_{p,\alpha,r}(x, y) = \mathcal{M} \circ \mathsf{R}_r \circ \cdots \circ \mathsf{R}_1(x, y).$$

For $1 \leqslant i \leqslant r$, the i-th round function R_i is defined as

$$\mathsf{R}_i(x, y) = \mathcal{H} \circ \mathcal{M}(x + c_i, y + d_i) \qquad \text{and} \qquad \mathcal{M}(x, y) = (2x + y, x + y),$$

for constants $c_i, d_i \in \mathbb{F}_p$. $\mathcal{H}$ is the nonlinear operation over $\mathbb{F}_p^2$ that is described in Figure 7.7b for a non-zero constant $a \in \mathbb{F}_p$.



(a) Anemoi round function. (b) $\mathcal{H}$ in odd characteristic.

Figure 7.7: Description of Anemoi over prime fields with $\ell = 1$.

7.3.0.2 Failure of the Direct FreeLunch Approach.

As a starting point, we consider the following slight modification[6] of the polynomial system $P_{\mathcal{G}}$ for Anemoi, for $1 \leqslant i \leqslant r$.

$$p_i(x_0, \ldots, x_i) = x_i^{\alpha} + aQ_{i-1}(x_0, \ldots, x_{i-1})^2 - P_{i-1}(x_0, \ldots, x_{i-1}) + a^{-1}, \tag{7.8}$$
$$g(x_0, \ldots, x_r) = P_r(x_0, \ldots, x_r)\ . \tag{7.9}$$

A first observation is that $z_{i,0}$ has a larger leading term than $z_{i,1}$ under any monomial order. Since this leading term gets distributed to both branches under $\mathcal{M}$ (without the possibility of cancelling the leading term), we have $\mathrm{LM}(Q_i) = \mathrm{LM}(P_i)$. Now note from the output shown in Figure 7.7b that in the computation of $z_{i,0}$, the terms aQ_{i-1}^2 and $-aQ_{i-1}^2$ both occur and cancel each other. Hence, the leading monomial of g must be either $x_r\mathrm{LM}(Q_{r-1})$ or x_r^2, so there is no possible choice of monomial order where g will have a leading monomial in only x_0.

[6]The only difference from the description in Section 7.1.4 is that we allow $\mathcal{L}_{i,j}$ from (7.2) to be quadratic, due to the term Q_{i-1}^2.

In order to circumvent this issue, we will multiply g by suitable monomials in $x_1, \ldots, x_r$ that leads to a reduction by the polynomials $p_1, \ldots, p_r$. This process will ultimately lead to a new polynomial g^*, whose leading monomial will be univariate in x_0. To briefly illustrate the idea, we consider the first step of this procedure. Writing out g in terms of P_{r-1}, Q_{r-1} and x_r, we have

$$\begin{aligned} g(P_{r-1}, Q_{r-1}, x_r) &= (1 - 4ax_r)Q_{r-1} + (2ax_r - 1)x_r + 2(P_{r-1} - a^{-1}) \\ &= (-4aQ_{r-1} + 2ax_r - 1)x_r + Q_{r-1} + 2P_{r-1} - 2a^{-1} \ , \end{aligned}$$

taking into account the final $\mathcal{M}$-transformation. In order to cancel out the product $x_r Q_{r-1}$ using p_r, we construct the following polynomial:

$$\begin{aligned} g' &= x_r^{\alpha-1} g + (4aQ_{r-1} - 2ax_r + 1)p_r \\ &= x_r^{\alpha}(-4aQ_{r-1} + 2ax_r - 1) + x_r^{\alpha-1}(Q_{r-1} + 2P_{r-1} - 2a^{-1}) \\ &\quad + (4aQ_{r-1} - 2ax_r + 1)(x_r^{\alpha} + aQ_{r-1}^2 - P_{r-1} + a^{-1}) \\ &= x_r^{\alpha-1}(Q_{r-1} + 2P_{r-1} - 2a^{-1}) + 4a^2 Q_{r-1}^3 + aQ_{r-1}^2 \\ &\quad + 4Q_{r-1}(1 - aP_{r-1}) - P_{r-1} + a^{-1} - 2x_r(a^2 Q_{r-1}^2 - aP_{r-1} + 1) \ . \end{aligned}$$

Hence, we have successfully eliminated x_r from the leading monomial of g' under any monomial order that satisfies:

$$\begin{aligned} \mathrm{wt}\left(\mathrm{LM}\left(Q_{r-1}^3\right)\right) &> \mathrm{wt}\left(\mathrm{LM}\left(x_r^{\alpha-1} Q_{r-1}\right)\right) \ , \\ \mathrm{wt}\left(\mathrm{LM}\left(Q_{r-1}^3\right)\right) &> \mathrm{wt}\left(\mathrm{LM}\left(x_r^{\alpha-1} P_{r-1}\right)\right) \ , \\ \mathrm{wt}\left(\mathrm{LM}\left(Q_{r-1}^3\right)\right) &> \mathrm{wt}\left(\mathrm{LM}\left(x_r Q_{r-1}^2\right)\right) \ , \\ \mathrm{wt}\left(\mathrm{LM}\left(Q_{r-1}^3\right)\right) &> \mathrm{wt}\left(\mathrm{LM}\left(x_r P_{r-1}\right)\right) \ , \end{aligned}$$

which, since $\alpha \geqslant 3$ and $\mathrm{LM}(P_{r-1}) = \mathrm{LM}(Q_{r-1})$, can be simplified further to:

$$\mathrm{wt}\left(\mathrm{LM}\left(Q_{r-1}^2\right)\right) > \mathrm{wt}\left(x_r^{\alpha-1}\right) = (\alpha - 1)\mathrm{wt}(x_r) \ .$$

7.3.0.3 Constructing FreeLunch Systems From `Anemoi`.

We now turn our attention to the general construction of g^* that will allow us to apply the FreeLunch machinery for solving the CICO problem for `Anemoi`. Here, we will not only be interested in the leading monomials of the intermediate states and p_i, but also in the second and third monomials. To this end, we define $\prec_A$ to be the monomial order associated with the weight vector defined recursively by

$$\begin{cases} \mathrm{wt}(x_0) = 1, \\ \mathrm{wt}(x_i) = \frac{2}{\alpha}\mathrm{wt}(x_0 \cdots x_{i-1}), \text{ for } 1 \leqslant i \leqslant r \ . \end{cases}$$

Indeed, this choice of monomial order allows us to prove the following two lemmas. For a polynomial h, we let $\mathrm{Mon}_j(h)$ denote the j-th monomial of h according to

$\prec_A$. As usual, we write $\mathrm{LM}(h) = \mathrm{Mon}_1(h)$. To avoid pathological cases, we always consider an affine input in x_0 for the CICO-problem such that x_0 is not eliminated after the initial linear operation $\mathcal{M}$. Finally, remember that two monomials may have equal weight, and only get sorted by their lexicographic order.

Lemma 7.5. *Let $Q_i(x_0, \ldots, x_i)$ be as defined in Figure 7.7a and ordered according to $\prec_A$. Then the following holds for $\alpha \geqslant 3$.*

$$\mathrm{LM}(Q_i) = x_0 \cdots x_i, \qquad \text{and} \qquad \mathrm{wt}(\mathrm{LM}(Q_i)) > \mathrm{wt}(\mathrm{Mon}_2(Q_i)) \ .$$

Proof. We proceed by induction. The statements are clearly true for $i = 0$, as Q_0 is an affine polynomial in x_0 by our CICO setting. Now assume it holds for $i - 1$. As mentioned above, leading terms cannot be canceled under $\mathcal{M}$, and the leading terms come from the first output of $\mathcal{M}$. Thus, we can restrict ourselves to the two largest monomials in the first output from $\mathcal{M}$, that is $2ax_iQ_{i-1} + ax_i^2 + P_{i-1} - a^{-1}$. From the induction hypothesis we have $\mathrm{LM}(x_iQ_{i-1}) = x_0 \cdots x_i$, and it follows from the definition of $\prec_A$ that this has a strictly higher weight than x_i^2 when $\alpha \geqslant 3$. □

Lemma 7.6. *Let $p_i(x_0, \ldots, x_i)$ be as defined in (7.8) and ordered according to $\prec_A$, and let $\alpha \geqslant 3$. Then for all $1 \leqslant i \leqslant r$ the following holds.*

1. $\mathrm{LM}(p_i) = x_i^\alpha$.
2. $\mathrm{Mon}_2(p_i) = (x_0 \cdots x_{i-1})^2$.
3. $\mathrm{wt}(\mathrm{LM}(p_i)) = \mathrm{wt}(\mathrm{Mon}_2(p_i)) > \mathrm{wt}(\mathrm{Mon}_3(p_i))$.

Proof. We see from the definition of p_i that $\mathrm{LM}(p_i)$ must be either x_i^α or $\mathrm{LM}(Q_i^2)$. From Lemma 7.5, we have $\mathrm{wt}(\mathrm{LM}(Q_i^2)) = 2\mathrm{wt}(x_0 \cdots x_{i-1})$, so these two monomials have the same weight by definition of $\prec_A$. $\mathrm{LM}(p_i) = x_i^\alpha$ then follows from Definition 6.7. Finally, $\mathrm{Mon}_3(p_i) = \mathrm{Mon}_2(Q_i^2)$ and thus has a strictly smaller weight than the initial two monomials (Lemma 7.5). □

Before we can define g^*, we also need a way to predict the powers of x_i we will use in the multiplication of g prior to the reductions by $p_1, \ldots, p_r$. This is handled by the following integer sequences.

Definition 7.8. We define two integer sequences $\{u_i\}_{0 \leqslant i \leqslant r}$ and $\{k_j\}_{1 \leqslant j \leqslant r}$, where $u_r = 1$, and the remaining sequences are recursively defined as follows:

- k_i is the unique integer $0 \leqslant k_i < \alpha$ such that $k_i \equiv -u_i \mod \alpha$;
- $u_{i-1} = u_i + 2(u_i + k_i)/\alpha$.

In the following, we will denote $u = u_0$.

Note that in the above definition, $u_i + k_i$ is always a multiple of α; hence u_{i-1} is indeed an integer.

For a polynomial h and sequence of polynomials H, we write $\mathrm{Red}(h, H)$ to denote the reduction of h by H w.r.t. $\prec_A$.More specifically, $\mathrm{Red}(h, H)$ is the

remainder after performing multivariate division of h by H (see [CLO97, Ch. 2, §3]). We are now in a position to define g^*. Let $g'_r = g$, and recursively define

$$g'_{i-1} = \mathrm{Red}(x_i^{k_i} g'_i, \{p_i, p_{i+1}, \ldots, p_r\}), \text{ for } i = r, r-1, \ldots, 1 \ .$$

We set $g^* = g'_0$, and $I_A = \langle P_{\mathcal{G}^*} \rangle$, where $P_{\mathcal{G}^*} = \{p_1, \ldots, p_r, g^*\}$. It is clear from the construction that I_A is a subideal of $\langle P_{\mathcal{G}} \rangle$. The following result guarantees that $P_{\mathcal{G}^*}$ is a FreeLunch system generating this subideal.

Proposition 7.5. *The polynomial system $P_{\mathcal{G}^*}$ is a FreeLunch system, where $\mathrm{LM}_{\prec_A}(g^*) = x_0^u$. Moreover, the variety of the associated ideal I_A contains all valid solutions of the underlying instance of* `Anemoi`*.*

Proof. By Lemma 7.6 we have $\mathrm{LM}(p_i) = x_i^\alpha$, so we need only show that $\mathrm{LM}(g^*)$ is a univariate monomial in x_0 to guarantee that $P_{\mathcal{G}^*}$ is a FreeLunch system. To this end, we will show by induction on descending i that $\mathrm{LM}(g'_i) = (x_0 \cdots x_i)^{u_i}$, and $\mathrm{wt}(\mathrm{LM}(g'_i)) > \mathrm{LM}(\mathrm{Mon}_2(g'_i))$.

For $i = r$, we have $g'_r = g = Q_r$, and the statement holds by Lemma 7.5. Suppose the hypothesis holds for a given i and consider $i-1$. If we denote $s_i = (u_i + k_i)/\alpha$, we have $\mathrm{LM}(x_i^{k_i} g'_i) = (x_0 \cdots x_{i-1})^{u_i} x_i^{\alpha s_i}$. We now reduce this monomial by p_i. Write c for the leading coefficient of $x_i^{k_i} g'_i$. From Lemma 7.6, we have that the first two monomials in p_i have the same weight, while all other monomials have smaller weights. Moreover, the induction hypothesis ensures that $\mathrm{Mon}_j(x_i^{k_i} g'_i)$ will a have smaller weight than $\mathrm{LM}(x_i^{k_i} g'_i)$ for $j \geqslant 2$. Hence,

$$\mathrm{LM}\left(x_i^{k_i} g'_i - c(x_0 \cdots x_{i-1})^{u_i} x_i^{\alpha(s_i-1)} p_i\right) = (x_0 \cdots x_{i-1})^{u_i+2} x_i^{\alpha(s_i-1)},$$

following from the fact that $\mathrm{wt}((x_0 \cdots x_{i-1})^{u_i+2} x_i^{\alpha(s_i-1)}) = \mathrm{wt}(\mathrm{LM}(x_i^{k_i} g'_i)) = \mathrm{wt}\left(\mathrm{LM}\left((x_0 \cdots x_{i-1})^{u_i} p_i\right)\right)$, and the weight of all other monomials are guaranteed to be strictly smaller. Repeating this process $s_i - 1$ times, we get $\mathrm{LM}(g'_{i-1}) = (x_0 \cdots x_{i-1})^{u_i+2s_i}$, which proves the induction statement. Since this implies that $\mathrm{LM}(g^*) = \mathrm{LM}(g'_0) = x_0^u$, it also concludes the proof of the first part of the proposition. The second part holds since I_A is a subideal of $\langle P_{\mathcal{G}} \rangle$, where the variety of the latter ideal will contain all solutions of `Anemoi`. □

Ideal Degree.

Based on experiments, the authors of `Anemoi` conjectured a tight upper bound on the ideal degree of one modeling of the CICO-problem to be $(\alpha+2)^r$ [BBC+23, Conjecture 2]. As I_A is a FreeLunch system, we have $D_{I_A} = \alpha^r u$, where we recall that u is an integer depending on r and α. As I_A is a subideal of $\langle P_{\mathcal{G}} \rangle$, we generally expect D_{I_A} to be strictly larger than $(\alpha+2)^r$. The following result guarantees that D_I can at most differ by a factor close to α, which in practical instances will be a small constant.

Proposition 7.6. *Let u be as defined in Definition 7.8 for integers $r, \alpha \geqslant 1$. Then*

$$\left(\frac{\alpha+2}{\alpha}\right)^r \leqslant u \leqslant (\alpha+1)\left(\frac{\alpha+2}{\alpha}\right)^r - \alpha \ .$$

Proof. Recall from Definition 7.8 that u is defined as $u = u_0$ through the sequence $\{u_i\}_{0 \leqslant i \leqslant r}$. To simplify the exposition, we will work with the sequence $\{v_i\}_{0 \leqslant i \leqslant r}$ defined by $v_0 = 1$, and

$$v_{i+1} = v_i + 2\left\lceil \frac{v_i}{\alpha} \right\rceil, \text{ for } 0 \leqslant i < r.$$

Note that $v_i = u_{r-i}$ and, in particular, $v_r = u$. Define two more integer sequences $\{a_i\}_{0 \leqslant i \leqslant r}$ and $\{b_i\}_{0 \leqslant i \leqslant r}$ defined by $a_0 = b_0 = 1$ and for $0 \leqslant i < r$

$$a_{i+1} = \frac{\alpha + 2}{\alpha} a_i, \qquad b_{i+1} = \frac{\alpha + 2}{\alpha} b_i + 2.$$

As a first step, we will prove $a_i \leqslant v_i \leqslant b_i$. This is clearly true for $i = 0$. Supposing it holds up to some i, then using the identity $x \leqslant \lceil x \rceil < x + 1$, we have

$$\frac{\alpha + 2}{\alpha} v_i \leqslant v_i + 2\left\lceil \frac{v_i}{\alpha} \right\rceil < \frac{\alpha + 2}{\alpha} v_i + 2,$$

Thus, using the induction hypothesis and the definitions of $\{a_i\}$ and $\{b_i\}$,

$$a_{i+1} \leqslant v_{i+1} \leqslant b_{i+1}.$$

Observe that $a_i = \left(\frac{\alpha+2}{\alpha}\right)^i$, which proves the left-hand side of the inequality in the proposition. For the right-hand side we note that $\{b_i\}$ can be written as $b_i = (\alpha + 1)\left(\frac{\alpha+2}{\alpha}\right)^i - \alpha$, when $i \geqslant 1$. Indeed, this can be verified for $i = 1$. Supposing it holds up to some i, then

$$\begin{aligned} b_{i+1} &= \frac{\alpha + 2}{\alpha} b_i + 2 \\ &= (\alpha + 1)\left(\frac{\alpha + 2}{\alpha}\right)^{i+1} - \frac{\alpha(\alpha + 2)}{\alpha} + 2 \\ &= (\alpha + 1)\left(\frac{\alpha + 2}{\alpha}\right)^{i+1} - \alpha, \end{aligned}$$

and the bounds on u stated in Proposition 7.6 follows. □

From experiments for $\alpha = 3$ and large r, we find $u \approx 2.1\,(5/3)^r$.

Summary. For Anemoi, we get a polynomial system with r equations of respective leading terms $x_1^\alpha, \ldots x_r^\alpha$ and one equation of leading term x_0^u. This gives the following parameters:

$$\begin{aligned} D_I &= \alpha^r u, \\ D_H &= \alpha^r. \end{aligned}$$

Security claim	$\alpha = 3$	$\alpha = 5$	$\alpha = 7$	$\alpha = 11$
128	118 (21)	156 (21)	174 (20)	198 (19)
256	203 (37)	270 (37)	307 (36)	358 (35)

Table 7.6: Expected time complexity of `polyDet` for the different full-round instances of `Anemoi` over $\mathbb{F}_p$ and $\ell = 1$. Number of rounds in parentheses.

Number of rounds	Complexity of `polyDet`	Time (s) `sysGen`	Time (s) `matGen`	Time (s) `polyDet`	Memory (MB)
3	20	< 0.01	< 0.01	0.02	< 400
4	26	< 0.01	0.34	0.24	< 400
5	32	0.07	23.3	7.6	< 400
6	37	2.52	2,127	292	2,863
7	43	128	156,348	10,725	42,337

Table 7.7: Experimental results on `Anemoi` with $(\ell, \alpha) = (1, 3)$. `sysGen` is performed with `Magma` and refers to the generation of the polynomial system $P_{\mathcal{G}*}$ from scratch, including the computation of $P_{\mathcal{G}}$.

7.3.0.4 Complexity Analysis and Experimental Results.

The FreeLunch system $P_{\mathcal{G}*}$ consists of r polynomials of degree α and one polynomial of degree u. The algorithm of Section 7.1.2 has a complexity of $\tilde{\mathcal{O}}(\alpha^{r\omega}u)$. We plugged in the numbers for odd `Anemoi` ($\ell = 1$), see Table 7.6. We also ran experiments for `Anemoi` with $(\ell, \alpha) = (1, 3)$ and different number of rounds to verify the theory presented above. The results are presented in Table 7.7.

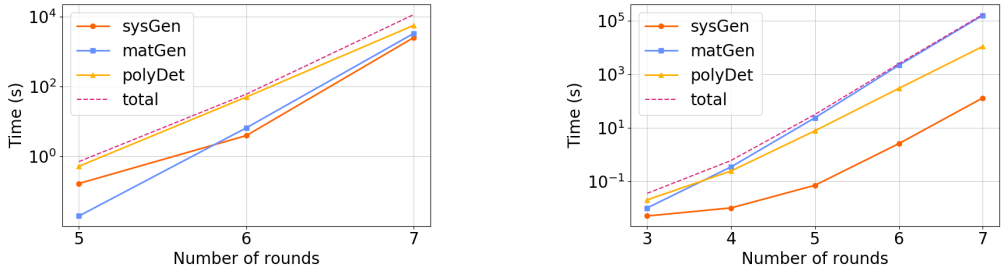
7.4 Discussion on the FreeLunch Attack

We have presented the FreeLunch approach, an algebraic attack particularly efficient against arithmetization-oriented permutations. We conclude this chapter with some comments regarding our experiments as well the consequences of our results, in particular regarding the areas we believe worth investigating further.

7.4.1 Discussion on Experimental Results

Figure 7.8 depicts the runtimes of each step of our attack that we obtained experimentally when targeting `Griffin` and `Anemoi`. A first observation is that the running time of a full FreeLunch-based attack is hard to predict: there are three steps (`sysGen`, `matGen`, and `polyDet`), and we experimentally found situations where each of them was the slowest. The case of `sysGen` is a bit peculiar: using `SageMath`, `Magma` or Flint/NTL yields very different results and a deeper understanding seems out of our grasp. We nevertheless would argue (see Section 7.1.4.3) that, should their implementations use similar tools, `sysGen` will always be of lower complexity than that of the rest of the attack.

Assuming that the dominating step is either `matGen` or `polyDet`, it then seems easy to extrapolate: as we can see in Figure 7.8, their logarithm increases linearly with the number of rounds. Even better: for `Griffin`, Equation 7.7 predicts that adding a round multiplies the complexity of `polyDet` by $\alpha^{\omega}(2\alpha+1) \approx 109.5$, which closely matches our observations as $5727/50.79 \approx 112.7$. For `matGen`, we see that adding a round multiplies the time complexity by about 500. Extrapolating from this, an attack against full-round `Griffin` should take about $4.2 \cdot 10^{11}$s on a single CPU (around 13,000 years), or around 2^{70} clock cycles at 2.3 GHz. Similarly, for `Anemoi` with $\mathbb{F}_p$ and $(l, \alpha) = (1, 3)$, adding a round multiplies the time complexity of `matGen` by about 75. Extrapolating gives respectively 2^{104} seconds (or 2^{135} clock cycles) and 2^{204} seconds (2^{235} clock cycles) for full-round `Anemoi` with 128 and 256 bits of security.



(a) `Griffin` complexity (from Table 7.3). (b) `Anemoi` complexity (from Table 7.7).

Figure 7.8: Experimental time complexity of our attacks on `Griffin` and `Anemoi`.

7.4.2 Preventing the FreeLunch Attack

Our attack breaks full-round instances of symmetric primitives built using state-of-the-art security arguments, which consequently must be revisited: one must learn how to prevent the relevant applicability of the FreeLunch approach.

At the Primitive Level. An obvious but perhaps costly countermeasure consists of simply adding more rounds. This is particularly tempting as we are able to tightly estimate the complexity of `polyDet`, a step which we have found to often be the most expensive in practice. Choosing a number of rounds high enough to prevent it would be a simple yet convincing argument. Primitive designers must also be mindful of "classical" tricks, i.e., symmetric cryptanalysis techniques (a priori) unrelated to root finding that can be used to enhance its efficiency. In the case of 12-branch `Griffin`, the fact that we can bypass 3 out of 10 rounds using some kind of subspace trail is a problem we deem worth studying.

At the Mode of Operation Level. The FreeLunch systems are multivariate, but a single variable (x_0) plays an inherently different role, meaning that they have

a univariate *flair*. This makes them particularly well suited to CICO instances whereby a single output word has to be set to 0, but they will not work if more zeroes are needed in the output. Thus, a simple countermeasure against the FreeLunch approach (and univariate ones) consists of forcing the capacity of the sponge to have at least two words set to 0, even if one word would a priori be enough.

7.4.3 Open Problems for Future Work

Time Taken by Polynomial Reductions. A roadblock in our complexity estimates is the number of operations needed to perform certain reductions of a polynomial modulo an ideal. This is crucial for understanding the complexity of the `matGen` step and, to a lesser degree, `sysGen`. A tighter estimate for these computations would greatly benefit our analysis: we would be able to figure out which step of our attack is the actual bottleneck without the need for experiments or assumptions, and designers could then be able to use fewer rounds to achieve a given security level against FreeLunch-based attacks. For instance, estimating the complexity of a reduction by a FreeLunch quasi-triangular system (Definition 7.2) would be a big step forward.

Other Custom Approaches. The FreeLunch approach is, to the best of our knowledge, the first "custom" root finding method designed specifically for use in symmetric cryptanalysis. There is, of course, no reason to believe that it is the only one possible, and we consider it a direction worth pursuing. As a first step, a multivariate variant of FreeLunch where several variables play the role of x_0, and where several words need to be set to 0, would be an interesting target.

Bibliography

[AAB+19] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. *Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols*. Cryptology ePrint Archive, Report 2019/426. https://eprint.iacr.org/2019/426. 2019 (cit. on pp. 220, 221).

[AAB+20] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. "Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols". In: *IACR Transactions on Symmetric Cryptology* 2020.3 (2020), pp. 1–45. ISSN: 2519-173X. DOI: 10.13154/tosc.v2020.i3.1-45 (cit. on pp. xv, xvi, 200–203).

[ACG+19] Martin R. Albrecht, Carlos Cid, Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, and Markus Schofnegger. "Algebraic Cryptanalysis of STARK-Friendly Designs: Application to MARVELlous and MiMC". In: *Advances in Cryptology – ASIACRYPT 2019, Part III*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11923. Lecture Notes in Computer Science. Kobe, Japan: Springer, Heidelberg, Germany, 2019, pp. 371–397. DOI: 10.1007/978-3-030-34618-8_13 (cit. on pp. 201, 214).

[AD18] Tomer Ashur and Siemen Dhooghe. *MARVELlous: a STARK-Friendly Family of Cryptographic Primitives*. Cryptology ePrint Archive, Report 2018/1098. https://eprint.iacr.org/2018/1098. 2018 (cit. on pp. xv, 200, 201).

[Ada97a] Carlisle Adams. *The CAST-128 encryption algorithm*. Tech. rep. 1997 (cit. on p. 18).

[Ada97b] Dr. Carlisle Adams. *The CAST-128 Encryption Algorithm*. RFC 2144. May 1997. DOI: 10.17487/RFC2144 (cit. on p. 86).

[Aes] *Advanced Encryption Standard (AES)*. National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce. Nov. 2001 (cit. on p. 170).

[AGP+19] Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. "Feistel Structures for MPC, and More". In: *ESORICS 2019: 24th European Symposium on Research in Computer*

Security, Part II. Ed. by Kazue Sako, Steve Schneider, and Peter Y. A. Ryan. Vol. 11736. Lecture Notes in Computer Science. Luxembourg: Springer, Heidelberg, Germany, 2019, pp. 151–171. DOI: 10.1007/978-3-030-29962-0_8 (cit. on p. 202).

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity". In: *Advances in Cryptology – ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer, Heidelberg, Germany, 2016, pp. 191–219. DOI: 10.1007/978-3-662-53887-6_7 (cit. on pp. xv, xvi, 200–203, 205, 214).

[AKM+22] Tomer Ashur, Al Kindi, Willi Meier, Alan Szepieniec, and Bobbin Threadbare. *Rescue-Prime Optimized*. Cryptology ePrint Archive, Report 2022/1577. https://eprint.iacr.org/2022/1577. 2022 (cit. on p. 261).

[AKM23] Tomer Ashur, Al Kindi, and Mohammad Mahzoun. *XHash8 and XHash12: Efficient STARK-friendly Hash Functions*. Cryptology ePrint Archive, Paper 2023/1045. https://eprint.iacr.org/2023/1045. 2023 (cit. on pp. xv, 200, 201, 214, 239, 261, 263, 264).

[ALP+19a] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. *ForkAE v.1*. Submission to the NIST Lightweight Cryptography standardization process. 2019 (cit. on pp. 47, 80).

[ALP+19b] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. "Forkcipher: A New Primitive for Authenticated Encryption of Very Short Messages". In: *Advances in Cryptology – ASIACRYPT 2019, Part II*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11922. Lecture Notes in Computer Science. Kobe, Japan: Springer, Heidelberg, Germany, 2019, pp. 153–182. DOI: 10.1007/978-3-030-34621-8_6 (cit. on pp. xi, 47).

[AMP+14] Jean-Philippe Aumasson, Willi Meier, Raphael C.-W. Phan, and Luca Henzen. *The Hash Function BLAKE*. Information Security and Cryptography. Springer, Heidelberg, Germany, 2014. ISBN: 978-3-662-44757-4. DOI: 10.1007/978-3-662-44757-4 (cit. on p. 18).

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. "Ciphers for MPC and FHE". In: *Advances in Cryptology – EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Sofia, Bulgaria: Springer, Heidelberg, Germany, 2015, pp. 430–454. DOI: 10.1007/978-3-662-46800-5_17 (cit. on pp. xv, 200).

[ARV+18] Elena Andreeva, Reza Reyhanitabar, Kerem Varici, and Damian Vizár. *Forking a Blockcipher for Authenticated Encryption of Very Short Messages*. Cryptology ePrint Archive, Report 2018/916. https://eprint.iacr.org/2018/916. 2018 (cit. on pp. xi, 47–49).

[AST+17] Ahmed Abdelkhalek, Yu Sasaki, Yosuke Todo, Mohamed Tolba, and Amr M. Youssef. "MILP Modeling for (Large) S-boxes to Optimize Probability of Differential Characteristics". In: *IACR Transactions on Symmetric Cryptology* 2017.4 (2017), pp. 99–129. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i4.99-129 (cit. on pp. 40, 41).

[BAK98] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. "Serpent: A New Block Cipher Proposal". In: *Fast Software Encryption – FSE'98*. Ed. by Serge Vaudenay. Vol. 1372. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, 1998, pp. 222–238. DOI: 10.1007/3-540-69710-1_15 (cit. on p. 18).

[Bar19] Navid Ghaedi Bardeh. *A Key-Independent Distinguisher for 6-round AES in an Adaptive Setting*. Cryptology ePrint Archive, Report 2019/945. https://eprint.iacr.org/2019/945. 2019 (cit. on pp. xiii, 84, 96, 109).

[Bar23] Augustin Bariant. *Algebraic Cryptanalysis of Full Ciminion*. Cryptology ePrint Archive, Paper 2023/1283. https://eprint.iacr.org/2023/1283. 2023 (cit. on pp. xv, 199, 226).

[BBC+23] Clémence Bouvier, Pierre Briaud, Pyrros Chaidos, Léo Perrin, Robin Salen, Vesselin Velichkov, and Danny Willems. "New Design Techniques for Efficient Arithmetization-Oriented Hash Functions: ttAnemoi Permutations and ttJive Compression Mode". In: *Advances in Cryptology – CRYPTO 2023, Part III*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14083. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2023, pp. 507–539. DOI: 10.1007/978-3-031-38548-3_17 (cit. on pp. xv, xvii, 200, 201, 214, 239, 264, 268).

[BBG+08] Ryad Benadjila, Olivier Billet, Henri Gilbert, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw, and Yannick Seurin. *SHA-3 Proposal: ECHO*. Submission to NIST SHA-3 Cryptographic Hash Algorithm Competition. Available at https://ehash.iaik.tugraz.at/uploads/9/91/Echo.pdf. 2008 (cit. on pp. 37, 170).

[BBH+18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. *Scalable, transparent, and post-quantum secure computational integrity*. Cryptology ePrint Archive, Report 2018/046. https://eprint.iacr.org/2018/046. 2018 (cit. on p. 200).

[BBJ+19] Subhadeep Banik, Jannis Bossert, Amit Jana, Eik List, Stefan Lucks, Willi Meier, Mostafizar Rahman, Dhiman Saha, and Yu Sasaki. "Cryptanalysis of ForkAES". In: *ACNS 19: 17th International Conference on Applied Cryptography and Network Security*. Ed. by Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung. Vol. 11464. Lecture Notes in Computer Science. Bogota, Colombia: Springer, Heidelberg, Germany, 2019, pp. 43–63. DOI: 10.1007/978-3-030-21568-2_3 (cit. on pp. xi, 51–53).

[BBL+22] Augustin Bariant, Clémence Bouvier, Gaëtan Leurent, and Léo Perrin. "Algebraic Attacks against Some Arithmetization-Oriented Primitives". In: *IACR Transactions on Symmetric Cryptology* 2022.3 (2022), pp. 73–101. DOI: 10.46586/tosc.v2022.i3.73-101 (cit. on pp. xv, 199, 213, 214, 226, 235).

[BBL+24a] Augustin Bariant, Jules Baudrin, Gaëtan Leurent, Clara Pernot, Léo Perrin, and Thomas Peyrin. *Fast AES-based Universal Hash Functions and MACs*. Accepted at ToSC 2024.2. 2024.

[BBL+24b] Augustin Bariant, Aurélien Boeuf, Axel Lemoine, Irati Manterola Ayala, Morten Øygarden, Léo Perrin, and Håvard Raddum. *The Algebraic Freelunch Efficient Gröbner Basis Attacks Against Arithmetization-Oriented Primitives*. Cryptology ePrint Archive, Paper 2024/347. https://eprint.iacr.org/2024/347. 2024 (cit. on pp. xvii, 239).

[BC18] Christina Boura and Anne Canteaut. "On the Boomerang Uniformity of Cryptographic Sboxes". In: *IACR Transactions on Symmetric Cryptology* 2018.3 (2018), pp. 290–310. ISSN: 2519-173X. DOI: 10.13154/tosc.v2018.i3.290-310 (cit. on p. 62).

[BC20] Christina Boura and Daniel Coggia. "Efficient MILP Modelings for Sboxes and Linear Layers of SPN ciphers". In: *IACR Transactions on Symmetric Cryptology* 2020.3 (2020), pp. 327–361. ISSN: 2519-173X. DOI: 10.13154/tosc.v2020.i3.327-361 (cit. on p. 41).

[BCC+12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again". In: *ITCS 2012: 3rd Innovations in Theoretical Computer Science*. Ed. by Shafi Goldwasser. Cambridge, MA, USA: Association for Computing Machinery, 2012, pp. 326–349. DOI: 10.1145/2090236.2090263 (cit. on p. 200).

[BCD+20] Tim Beyne, Anne Canteaut, Itai Dinur, Maria Eichlseder, Gregor Leander, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Yu Sasaki, Yosuke Todo, and Friedrich Wiemer. "Out of Oddity - New Cryptanalytic Techniques Against Symmetric Primitives Optimized for Integrity Proof Systems". In: *Advances in Cryptology – CRYPTO 2020, Part III*. Ed. by Daniele Micciancio and Thomas

Ristenpart. Vol. 12172. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2020, pp. 299–328. DOI: 10.1007/978-3-030-56877-1_11 (cit. on pp. 201, 202).

[BCP22] Clémence Bouvier, Anne Canteaut, and Léo Perrin. *On the Algebraic Degree of Iterated Power Functions*. Cryptology ePrint Archive, Report 2022/366. https://eprint.iacr.org/2022/366. 2022 (cit. on p. 202).

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. "The Magma algebra system. I. The user language". In: *J. Symbolic Comput.* 24.3-4 (1997). Computational algebra and number theory (London, 1993), pp. 235–265. ISSN: 0747-7171. DOI: 10.1006/jsco.1996.0125 (cit. on pp. 225, 252).

[BD08] Eli Biham and Orr Dunkelman. *The SHAvite-3 Hash Function*. Submission to NIST SHA-3 Cryptographic Hash Algorithm Competition. Available at https://www.cs.rit.edu/~ark/20090927/Round2Candidates/SHAvite-3.pdf. 2008 (cit. on p. 37).

[BDK01] Eli Biham, Orr Dunkelman, and Nathan Keller. "The Rectangle Attack - Rectangling the Serpent". In: *Advances in Cryptology – EUROCRYPT 2001*. Ed. by Birgit Pfitzmann. Vol. 2045. Lecture Notes in Computer Science. Innsbruck, Austria: Springer, Heidelberg, Germany, 2001, pp. 340–357. DOI: 10.1007/3-540-44987-6_21 (cit. on p. 91).

[BDK02] Eli Biham, Orr Dunkelman, and Nathan Keller. "New Results on Boomerang and Rectangle Attacks". In: *Fast Software Encryption – FSE 2002*. Ed. by Joan Daemen and Vincent Rijmen. Vol. 2365. Lecture Notes in Computer Science. Leuven, Belgium: Springer, Heidelberg, Germany, 2002, pp. 1–16. DOI: 10.1007/3-540-45661-9_1 (cit. on p. 91).

[BDK+18] Achiya Bar-On, Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. "Improved Key Recovery Attacks on Reduced-Round AES with Practical Data and Memory Complexities". In: *Advances in Cryptology – CRYPTO 2018, Part II*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10992. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2018, pp. 185–212. DOI: 10.1007/978-3-319-96881-0_7 (cit. on p. 84).

[BDK+24] Augustin Bariant, Orr Dunkelman, Nathan Keller, Gaëtan Leurent, and Victor Mollimard. *Improved Boomerang Attacks on 6-Round AES*. In submission. 2024.

[BDL20] Augustin Bariant, Nicolas David, and Gaëtan Leurent. "Cryptanalysis of Forkciphers". In: *IACR Transactions on Symmetric Cryptology* 2020.1 (2020), pp. 233–265. ISSN: 2519-173X. DOI: 10.13154/tosc.v2020.i1.233-265 (cit. on pp. xi, 47, 66).

[BDP+06] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. *RadioGatún, a belt-and-mill hash function*. Cryptology ePrint Archive, Report 2006/369. https://eprint.iacr.org/2006/369. 2006 (cit. on p. 176).

[BDP+07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Sponge functions". In: *ECRYPT hash workshop*. Vol. 2007. 9. Citeseer. 2007 (cit. on pp. 16, 23).

[BDP+09] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Keccak sponge function family main document". In: *Submission to NIST (Round 2)* 3.30 (2009), pp. 320–337 (cit. on pp. 23, 202).

[BDP+12] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications". In: *SAC 2011: 18th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Ali Miri and Serge Vaudenay. Vol. 7118. Lecture Notes in Computer Science. Toronto, Ontario, Canada: Springer, Heidelberg, Germany, 2012, pp. 320–337. DOI: 10.1007/978-3-642-28496-0_19 (cit. on p. 19).

[Ber05] Daniel J. Bernstein. "The Poly1305-AES Message-Authentication Code". In: *Fast Software Encryption – FSE 2005*. Ed. by Henri Gilbert and Helena Handschuh. Vol. 3557. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, 2005, pp. 32–49. DOI: 10.1007/11502760_3 (cit. on pp. 20, 170).

[Ber08a] Daniel J Bernstein. "ChaCha, a variant of Salsa20". In: *Workshop record of SASC*. Vol. 8. 1. Citeseer. 2008, pp. 3–5 (cit. on pp. 18, 20).

[Ber08b] Daniel J. Bernstein. "The Salsa20 Family of Stream Ciphers". In: *New Stream Cipher Designs - The eSTREAM Finalists*. Ed. by Matthew J. B. Robshaw and Olivier Billet. Vol. 4986. Lecture Notes in Computer Science. Springer, 2008, pp. 84–97. DOI: 10.1007/978-3-540-68351-3_8 (cit. on pp. 18, 20).

[BFL11] Charles Bouillaguet, Pierre-Alain Fouque, and Gaëtan Leurent. "Security Analysis of SIMD". In: *SAC 2010: 17th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Alex Biryukov, Guang Gong, and Douglas R. Stinson. Vol. 6544. Lecture Notes in Computer Science. Waterloo, Ontario, Canada: Springer, Heidelberg, Germany, 2011, pp. 351–368. DOI: 10.1007/978-3-642-19574-7_24 (cit. on pp. 40, 89).

[BFS04] Magali Bardet, Jean-Charles Faugere, and Bruno Salvy. "On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations". In: *Proceedings of the International Conference on Polynomial System Solving*. 2004, pp. 71–74 (cit. on pp. 213, 231).

[BGG+20] Zhenzhen Bao, Chun Guo, Jian Guo, and Ling Song. "TNT: How to Tweak a Block Cipher". In: *Advances in Cryptology – EUROCRYPT 2020, Part II*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Heidelberg, Germany, 2020, pp. 641–673. DOI: 10.1007/978-3-030-45724-2_22 (cit. on pp. xii, 39, 83, 85, 86, 125, 126).

[BGG+23] Emanuele Bellini, David Gérault, Juan Grados, Rusydi H. Makarim, and Thomas Peyrin. "Boosting Differential-Linear Cryptanalysis of ChaCha7 with MILP". In: *IACR Trans. Symmetric Cryptol.* 2023.2 (2023), pp. 189–223. DOI: 10.46586/TOSC.V2023.I2.189-223 (cit. on p. 40).

[BGL20] Zhenzhen Bao, Jian Guo, and Eik List. "Extended Truncated-differential Distinguishers on Round-reduced AES". In: *IACR Transactions on Symmetric Cryptology* 2020.3 (2020), pp. 197–261. ISSN: 2519-173X. DOI: 10.13154/tosc.v2020.i3.197-261 (cit. on p. 84).

[BHK+99] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. "UMAC: Fast and Secure Message Authentication". In: *Advances in Cryptology – CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1999, pp. 216–233. DOI: 10.1007/3-540-48405-1_14 (cit. on p. 20).

[Bir04] Alex Biryukov. "The boomerang attack on 5 and 6-round reduced AES". In: *International Conference on Advanced Encryption Standard*. Springer. 2004, pp. 11–15 (cit. on pp. 84, 86, 89, 96, 98, 104, 109, 118–120).

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. "The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS". In: *Advances in Cryptology – CRYPTO 2016, Part II*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2016, pp. 123–153. DOI: 10.1007/978-3-662-53008-5_5 (cit. on pp. xi, 18, 38, 47–49, 89).

[BK09] Alex Biryukov and Dmitry Khovratovich. "Related-Key Cryptanalysis of the Full AES-192 and AES-256". In: *Advances in Cryptology – ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. Lecture Notes in Computer Science. Tokyo, Japan: Springer, Heidelberg, Germany, 2009, pp. 1–18. DOI: 10.1007/978-3-642-10366-7_1 (cit. on pp. 32, 136).

[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: *Cryptographic Hardware and Embedded Systems – CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Vol. 4727. Lecture Notes in Computer Science. Vienna, Austria: Springer, Heidelberg, Germany, 2007, pp. 450–466. DOI: 10.1007/978-3-540-74735-2_31 (cit. on p. 18).

[BKN09] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. "Distinguisher and Related-Key Attack on the Full AES-256". In: *Advances in Cryptology – CRYPTO 2009*. Ed. by Shai Halevi. Vol. 5677. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2009, pp. 231–249. DOI: 10.1007/978-3-642-03356-8_14 (cit. on p. 32).

[BL22] Augustin Bariant and Gaëtan Leurent. *Truncated Boomerang Attacks and Application to AES-based Ciphers*. Cryptology ePrint Archive, Report 2022/701. https://eprint.iacr.org/2022/701. 2022 (cit. on p. 137).

[BL23a] Augustin Bariant and Gaëtan Leurent. *Truncated Boomerang Attacks and Application to AES-based Ciphers — Additional data*. https://github.com/AugustinBariant/Truncated_boomerangs. 2023 (cit. on pp. 110, 118, 133, 137).

[BL23b] Augustin Bariant and Gaëtan Leurent. "Truncated Boomerang Attacks and Application to AES-Based Ciphers". In: *Advances in Cryptology – EUROCRYPT 2023, Part IV*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14007. Lecture Notes in Computer Science. Lyon, France: Springer, Heidelberg, Germany, 2023, pp. 3–35. DOI: 10.1007/978-3-031-30634-1_1 (cit. on pp. xi, xii, 49, 51, 83, 89, 109, 110, 116).

[BL24] Xavier Bonnetain and Virginie Lallemand. *A Note on Related-Tweakey Impossible Differential Attacks*. Cryptology ePrint Archive, Paper 2024/563. https://eprint.iacr.org/2024/563. 2024 (cit. on pp. 67, 71, 72).

[BLL+22] Jannis Bossert, Eik List, Stefan Lucks, and Sebastian Schmitz. "Pholkos - Efficient Large-State Tweakable Block Ciphers from the AES Round Function". In: *Topics in Cryptology – CT-RSA 2022*.

Ed. by Steven D. Galbraith. Vol. 13161. Lecture Notes in Computer Science. Virtual Event: Springer, Heidelberg, Germany, 2022, pp. 511–536. DOI: 10.1007/978-3-030-95312-6_21 (cit. on p. 37).

[BLM+19] Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. "CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks". In: *IACR Transactions on Symmetric Cryptology* 2019.1 (2019), pp. 5–45. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i1.5-45 (cit. on p. 89).

[BLN+18] Christina Boura, Virginie Lallemand, María Naya-Plasencia, and Valentin Suder. "Making the Impossible Possible". In: *Journal of Cryptology* 31.1 (Jan. 2018), pp. 101–133. DOI: 10.1007/s00145-016-9251-7 (cit. on p. 37).

[BM74] Allan Borodin and R. Moenck. "Fast Modular Transforms". In: *J. Comput. Syst. Sci.* 8.3 (1974), pp. 366–386. DOI: 10.1016/S0022-0000(74)80029-2 (cit. on p. 203).

[BND22] Jérémy Berthomieu, Vincent Neiger, and Mohab Safey El Din. "Faster Change of Order Algorithm for Gröbner Bases under Shape and Stability Assumptions". In: *ISSAC '22: International Symposium on Symbolic and Algebraic Computation, Villeneuve-d'Ascq, France, July 4 - 7, 2022*. Ed. by Marc Moreno Maza and Lihong Zhi. ACM, 2022, pp. 409–418. DOI: 10.1145/3476446.3535484 (cit. on pp. 210, 213, 233, 234, 241, 242, 244).

[BNS14] Christina Boura, María Naya-Plasencia, and Valentin Suder. "Scrutinizing and Improving Impossible Differential Attacks: Applications to CLEFIA, Camellia, LBlock and Simon". In: *Advances in Cryptology – ASIACRYPT 2014, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. Lecture Notes in Computer Science. Kaoshiung, Taiwan, R.O.C.: Springer, Heidelberg, Germany, 2014, pp. 179–199. DOI: 10.1007/978-3-662-45611-8_10 (cit. on pp. 31, 80).

[BPF+23] Emanuele Bellini, Alessandro De Piccoli, Mattia Formenti, David Gérault, Paul Huynh, Simone Pelizzola, Sergio Polese, and Andrea Visconti. "Differential Cryptanalysis with SAT, SMT, MILP, and CP: A Detailed Comparison for Bit-Oriented Primitives". In: *CANS 23: 22th International Conference on Cryptology and Network Security*. Ed. by Jing Deng, Vladimir Kolesnikov, and Alexander A. Schwarzmann. Vol. 14342. Lecture Notes in Computer Science. Augusta, GA, USA: Springer, Heidelberg, Germany, 2023, pp. 268–292. DOI: 10.1007/978-981-99-7563-1_13 (cit. on p. 40).

[BPP00] Joan Boyar, René Peralta, and Denis Pochuev. "On the multiplicative complexity of Boolean functions over the basis (Λ,⊕,1)". In: *Theoretical Computer Science* 235.1 (2000), pp. 43–57. ISSN: 0304-3975. DOI: 10.1016/S0304-3975(99)00182-6 (cit. on p. 200).

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. "GIFT: A Small Present - Towards Reaching the Limit of Lightweight Encryption". In: *Cryptographic Hardware and Embedded Systems – CHES 2017*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, Heidelberg, Germany, 2017, pp. 321–345. DOI: 10.1007/978-3-319-66787-4_16 (cit. on p. 18).

[BPW06a] Johannes Buchmann, Andrei Pyshkin, and Ralf-Philipp Weinmann. "A Zero-Dimensional Gröbner Basis for AES-128". In: *Fast Software Encryption – FSE 2006*. Ed. by Matthew J. B. Robshaw. Vol. 4047. Lecture Notes in Computer Science. Graz, Austria: Springer, Heidelberg, Germany, 2006, pp. 78–88. DOI: 10.1007/11799313_6 (cit. on pp. 214, 240).

[BPW06b] Johannes Buchmann, Andrei Pyshkin, and Ralf-Philipp Weinmann. "Block Ciphers Sensitive to Gröbner Basis Attacks". In: *Topics in Cryptology – CT-RSA 2006*. Ed. by David Pointcheval. Vol. 3860. Lecture Notes in Computer Science. San Jose, CA, USA: Springer, Heidelberg, Germany, 2006, pp. 313–331. DOI: 10.1007/11605805_20 (cit. on p. 214).

[BR19] Navid Ghaedi Bardeh and Sondre Rønjom. "The Exchange Attack: How to Distinguish Six Rounds of AES with $2^{88.2}$ Chosen Plaintexts". In: *Advances in Cryptology – ASIACRYPT 2019, Part III*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11923. Lecture Notes in Computer Science. Kobe, Japan: Springer, Heidelberg, Germany, 2019, pp. 347–370. DOI: 10.1007/978-3-030-34618-8_12 (cit. on pp. 84, 96, 109).

[BR22a] Navid Ghaedi Bardeh and Vincent Rijmen. "New Key-Recovery Attack on Reduced-Round AES". In: *IACR Transactions on Symmetric Cryptology* 2022.2 (2022), pp. 43–62. DOI: 10.46586/tosc.v2022.i2.43-62 (cit. on pp. 37, 96, 109).

[BR22b] Tim Beyne and Vincent Rijmen. "Differential Cryptanalysis in the Fixed-Key Model". In: *Advances in Cryptology – CRYPTO 2022, Part III*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13509. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2022, pp. 687–716. DOI: 10.1007/978-3-031-15982-4_23 (cit. on p. 25).

[BS92a] Eli Biham and Adi Shamir. "Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer". In: *Advances in Cryptology – CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1992, pp. 156–171. DOI: 10.1007/3-540-46766-1_11 (cit. on p. 23).

[BS92b] Eli Biham and Adi Shamir. "Differential Cryptanalysis of the Full 16-Round DES". In: *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*. Ed. by Ernest F. Brickell. Vol. 740. Lecture Notes in Computer Science. Springer, 1992, pp. 487–496. DOI: 10.1007/3-540-48071-4_34 (cit. on p. 10).

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. "National Security Agency 9800 Savage Road, Fort Meade, MD 20755, USA". In: (2013) (cit. on p. 18).

[Buc76] Bruno Buchberger. "A theoretical basis for the reduction of polynomials to canonical forms". In: *SIGSAM Bull.* 10.3 (1976), pp. 19–29. DOI: 10.1145/1088216.1088219 (cit. on pp. xvi, 199, 201, 206, 208).

[CCZ98] Claude Carlet, Pascale Charpin, and Victor A. Zinoviev. "Codes, Bent Functions and Permutations Suitable For DES-like Cryptosystems". In: *Des. Codes Cryptogr.* 15.2 (1998), pp. 125–156. DOI: 10.1023/A:1008344232130 (cit. on p. 201).

[CHP+17] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. "A Security Analysis of Deoxys and its Internal Tweakable Block Ciphers". In: *IACR Transactions on Symmetric Cryptology* 2017.3 (2017), pp. 73–107. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i3.73-107 (cit. on pp. 89, 132–135, 139, 171, 187).

[CHP+18] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. "Boomerang Connectivity Table: A New Cryptanalysis Tool". In: *Advances in Cryptology – EUROCRYPT 2018, Part II*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10821. Lecture Notes in Computer Science. Tel Aviv, Israel: Springer, Heidelberg, Germany, 2018, pp. 683–714. DOI: 10.1007/978-3-319-78375-8_22 (cit. on pp. 62, 94, 136).

[CK91] David G. Cantor and Erich L. Kaltofen. "On Fast Multiplication of Polynomials over Arbitrary Algebras". In: *Acta Informatica* 28.7 (1991), pp. 693–701. DOI: 10.1007/BF01178683 (cit. on pp. 202, 203, 244).

[CLO97] David A. Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (2. ed.)* Undergraduate texts in mathematics. Springer, 1997. ISBN: 978-0-387-94680-1 (cit. on pp. 207–209, 249, 268).

[CLO98] David A. Cox, John B. Little, and Donal O'Shea. *Using Algebraic Geometry*. First. Vol. 185. Graduate Texts in Mathematics. Springer, 1998. DOI: 10.1007/978-1-4757-6911-1 (cit. on p. 210).

[CS16] Benoît Cogliati and Yannick Seurin. "EWCDM: An Efficient, Beyond-Birthday Secure, Nonce-Misuse Resistant MAC". In: *Advances in Cryptology – CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2016, pp. 121–149. DOI: 10.1007/978-3-662-53018-4_5 (cit. on pp. xiv, 192, 193).

[CW79] Larry Carter and Mark N. Wegman. "Universal Classes of Hash Functions". In: *J. Comput. Syst. Sci.* 18.2 (1979), pp. 143–154. DOI: 10.1016/0022-0000(79)90044-8 (cit. on pp. 20, 192).

[DC98] Joan Daemen and Craig S. K. Clapp. "Fast Hashing and Stream Encryption with PANAMA". In: *Fast Software Encryption – FSE'98*. Ed. by Serge Vaudenay. Vol. 1372. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, 1998, pp. 60–74. DOI: 10.1007/3-540-69710-1_5 (cit. on pp. 171, 176).

[DDH+20] Stéphanie Delaune, Patrick Derbez, Paul Huynh, Marine Minier, Victor Mollimard, and Charles Prud'homme. *SKINNY with Scalpel - Comparing Tools for Differential Analysis*. Cryptology ePrint Archive, Report 2020/1402. https://eprint.iacr.org/2020/1402. 2020 (cit. on p. 40).

[DDV20] Stéphanie Delaune, Patrick Derbez, and Mathieu Vavrille. "Catching the Fastest Boomerangs Application to SKINNY". In: *IACR Transactions on Symmetric Cryptology* 2020.4 (2020), pp. 104–129. ISSN: 2519-173X. DOI: 10.46586/tosc.v2020.i4.104-129 (cit. on pp. 40, 89, 95, 96, 136).

[De 06] Christophe De Cannière. "Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles". In: *ISC 2006: 9th International Conference on Information Security*. Ed. by Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel. Vol. 4176. Lecture Notes in Computer Science. Samos Island, Greece: Springer, Heidelberg, Germany, 2006, pp. 171–186 (cit. on p. 20).

[DEM16] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. "Square Attack on 7-Round Kiasu-BC". In: *ACNS 16: 14th International Conference on Applied Cryptography and Network Security*. Ed. by Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider. Vol. 9696. Lecture Notes in Computer Science. Guildford, UK: Springer, Heidelberg, Germany, 2016, pp. 500–517. DOI: 10.1007/978-3-319-39555-5_27 (cit. on pp. 52, 85).

[DEM+21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. "Ascon v1.2: Lightweight Authenticated Encryption and Hashing". In: *Journal of Cryptology* 34.3 (July 2021), p. 33. DOI: 10.1007/s00145-021-09398-9 (cit. on p. 11).

[Der13] Patrick Derbez. "Meet-in-the-Middle Attacks on AES". Theses. Ecole Normale Supérieure de Paris - ENS Paris, Dec. 2013 (cit. on p. 132).

[Des] *Data Encryption Standard (DES)*. Tech. rep. Federal Information Processing Standards Publication 46-3. U.S. Department of Commerce, National Institute of Standards and Technology, 1977, reaffirmed 1988,1993,1999, withdrawn 2005 (cit. on pp. 9, 18).

[DF04] David Steven Dummit and Richard M. Foote. *Abstract Algebra*. Third. Hoboken, NJ: John Wiley & sons, 2004. ISBN: 0-471-43334-9 (cit. on p. 180).

[DFJ13] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. "Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting". In: *Advances in Cryptology – EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. Lecture Notes in Computer Science. Athens, Greece: Springer, Heidelberg, Germany, 2013, pp. 371–387. DOI: 10.1007/978-3-642-38348-9_23 (cit. on pp. xi, 37, 52).

[DG10] Vivien Dubois and Nicolas Gama. "The Degree of Regularity of HFE Systems". In: *Advances in Cryptology – ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. Lecture Notes in Computer Science. Singapore: Springer, Heidelberg, Germany, 2010, pp. 557–576. DOI: 10.1007/978-3-642-17373-8_32 (cit. on p. 211).

[DGG+21a] Christoph Dobraunig, Lorenzo Grassi, Anna Guinet, and Daniël Kuijsters. *Ciminion: Symmetric Encryption Based on Toffoli-Gates over Large Finite Fields*. Cryptology ePrint Archive, Report 2021/267. https://eprint.iacr.org/2021/267. 2021 (cit. on p. 231).

[DGG+21b] Christoph Dobraunig, Lorenzo Grassi, Anna Guinet, and Daniël Kuijsters. "Ciminion: Symmetric Encryption Based on Toffoli-Gates over Large Finite Fields". In: *Advances in Cryptology – EUROCRYPT 2021, Part II*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12697. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Heidelberg, Germany, 2021, pp. 3–34. DOI: 10.1007/978-3-030-77886-6_1 (cit. on pp. xv, 200, 201, 226, 228).

[DGK+24] Orr Dunkelman, Shibam Ghosh, Nathan Keller, Gaëtan Leurent, Avichai Marmor, and Victor Mollimard. "Partial Sums Meet FFT: Improved Attack on 6-Round AES". In: *EUROCRYPT 2024* (2024) (cit. on pp. xiii, 37, 84, 86).

[DH76] Whitfield Diffie and Martin E. Hellman. "New directions in cryptography". In: *IEEE Trans. Inf. Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638 (cit. on pp. x, 10).

[DH77] Whitfield Diffie and Martin E. Hellman. "Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard". In: *Computer* 10.6 (1977), pp. 74–84. DOI: 10.1109/C-M.1977.217750 (cit. on p. 31).

[DKL+12] Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel P. Smart. "Implementing AES via an Actively/Covertly Secure Dishonest-Majority MPC Protocol". In: *SCN 12: 8th International Conference on Security in Communication Networks*. Ed. by Ivan Visconti and Roberto De Prisco. Vol. 7485. Lecture Notes in Computer Science. Amalfi, Italy: Springer, Heidelberg, Germany, 2012, pp. 241–263. DOI: 10.1007/978-3-642-32928-9_14 (cit. on p. 200).

[DKR+19] Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. *The Retracing Boomerang Attack*. Cryptology ePrint Archive, Report 2019/1154. https://eprint.iacr.org/2019/1154. 2019 (cit. on pp. 103, 104, 108, 166).

[DKR+20] Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. "The Retracing Boomerang Attack". In: *Advances in Cryptology – EUROCRYPT 2020, Part I*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Heidelberg, Germany, 2020, pp. 280–309. DOI: 10.1007/978-3-030-45721-1_11 (cit. on pp. xiii, 37, 84, 86, 96, 100, 103, 104, 158, 159).

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. "The Block Cipher Square". In: *Fast Software Encryption – FSE'97*. Ed. by Eli Biham. Vol. 1267. Lecture Notes in Computer Science. Haifa, Israel: Springer, Heidelberg, Germany, 1997, pp. 149–165. DOI: 10.1007/BFb0052343 (cit. on pp. xiii, 52, 84, 86).

[DKS10] Orr Dunkelman, Nathan Keller, and Adi Shamir. "A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony". In: *Advances in Cryptology – CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2010, pp. 393–410. DOI: 10.1007/978-3-642-14623-7_21 (cit. on pp. 93, 94).

[DKS15] Orr Dunkelman, Nathan Keller, and Adi Shamir. "Improved Single-Key Attacks on 8-Round AES-192 and AES-256". In: *Journal of Cryptology* 28.3 (July 2015), pp. 397–422. DOI: 10.1007/s00145-013-9159-4 (cit. on p. 32).

[DL17] Christoph Dobraunig and Eik List. "Impossible-Differential and Boomerang Cryptanalysis of Round-Reduced Kiasu-BC". In: *Topics in Cryptology – CT-RSA 2017*. Ed. by Helena Handschuh. Vol. 10159.

Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, 2017, pp. 207–222. DOI: 10.1007/978-3-319-52153-4_12 (cit. on pp. xi, xii, 52, 85, 86, 109, 110, 123).

[DLT14] Ivan Damgård, Rasmus Lauritsen, and Tomas Toft. "An Empirical Study and Some Improvements of the MiniMac Protocol for Secure Computation". In: *SCN 14: 9th International Conference on Security in Communication Networks*. Ed. by Michel Abdalla and Roberto De Prisco. Vol. 8642. Lecture Notes in Computer Science. Amalfi, Italy: Springer, Heidelberg, Germany, 2014, pp. 398–415. DOI: 10.1007/978-3-319-10879-7_23 (cit. on p. 200).

[DPS+12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryption". In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2012, pp. 643–662. DOI: 10.1007/978-3-642-32009-5_38 (cit. on p. 200).

[DQS+22] Xiaoyang Dong, Lingyue Qin, Siwei Sun, and Xiaoyun Wang. "Key Guessing Strategies for Linear Key-Schedule Algorithms in Rectangle Attacks". In: *Advances in Cryptology – EUROCRYPT 2022, Part III*. Ed. by Orr Dunkelman and Stefan Dziembowski. Vol. 13277. Lecture Notes in Computer Science. Trondheim, Norway: Springer, Heidelberg, Germany, 2022, pp. 3–33. DOI: 10.1007/978-3-031-07082-2_1 (cit. on pp. xii, 47, 67, 89).

[DR01] Joan Daemen and Vincent Rijmen. "The Wide Trail Design Strategy". In: *Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, December 17-19, 2001, Proceedings*. Ed. by Bahram Honary. Vol. 2260. Lecture Notes in Computer Science. Springer, 2001, pp. 222–238. DOI: 10.1007/3-540-45325-3_20 (cit. on p. 34).

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. ISBN: 3-540-42580-2. DOI: 10.1007/978-3-662-04722-4 (cit. on pp. x, 10, 18, 35, 83).

[DR06] Joan Daemen and Vincent Rijmen. "Understanding Two-Round Differentials in AES". In: *SCN 06: 5th International Conference on Security in Communication Networks*. Ed. by Roberto De Prisco and Moti Yung. Vol. 4116. Lecture Notes in Computer Science. Maiori, Italy: Springer, Heidelberg, Germany, 2006, pp. 78–94. DOI: 10.1007/11832072_6 (cit. on pp. 34, 102).

[DS08] Itai Dinur and Adi Shamir. *Cube Attacks on Tweakable Black Box Polynomials*. Cryptology ePrint Archive, Report 2008/385. https://eprint.iacr.org/2008/385. 2008 (cit. on p. 201).

[DS09] Itai Dinur and Adi Shamir. "Cube Attacks on Tweakable Black Box Polynomials". In: *Advances in Cryptology – EUROCRYPT 2009*. Ed. by Antoine Joux. Vol. 5479. Lecture Notes in Computer Science. Cologne, Germany: Springer, Heidelberg, Germany, 2009, pp. 278–299. DOI: 10.1007/978-3-642-01001-9_16 (cit. on pp. xvi, 31, 204).

[Dwo07a] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Tech. rep. NIST Special Publication 800-38D. National Institute of Standards and Technology, Nov. 2007 (cit. on p. 20).

[Dwo07b] Morris Dworkin. *Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality*. Tech. rep. National Institute of Standards and Technology, 2007 (cit. on p. 19).

[DZ13] Ivan Damgård and Sarah Zakarias. "Constant-Overhead Secure Computation of Boolean Circuits using Preprocessing". In: *TCC 2013: 10th Theory of Cryptography Conference*. Ed. by Amit Sahai. Vol. 7785. Lecture Notes in Computer Science. Tokyo, Japan: Springer, Heidelberg, Germany, 2013, pp. 621–641. DOI: 10.1007/978-3-642-36594-2_35 (cit. on p. 200).

[EGL+20] Maria Eichlseder, Lorenzo Grassi, Reinhard Lüftenegger, Morten Øygarden, Christian Rechberger, Markus Schofnegger, and Qingju Wang. "An Algebraic Attack on Ciphers with Low-Degree Round Functions: Application to Full MiMC". In: *Advances in Cryptology – ASIACRYPT 2020, Part I*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12491. Lecture Notes in Computer Science. Daejeon, South Korea: Springer, Heidelberg, Germany, 2020, pp. 477–506. DOI: 10.1007/978-3-030-64837-4_16 (cit. on pp. 201, 205).

[Eis13] David Eisenbud. *Commutative algebra: with a view toward algebraic geometry*. Vol. 150. Springer Science & Business Media, 2013 (cit. on p. 209).

[EJ00] Patrik Ekdahl and Thomas Johansson. "SNOW-a new stream cipher". In: *Proceedings of first open NESSIE workshop, KU-Leuven*. 2000, pp. 167–168 (cit. on p. 20).

[Fau02] Jean Charles Faugere. "A new efficient algorithm for computing Gröbner bases without reduction to zero (F 5)". In: *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*. 2002, pp. 75–83 (cit. on p. 213).

[Fei73] Horst Feistel. "Cryptography and computer privacy". In: *Scientific american* 228.5 (1973), pp. 15–23 (cit. on p. 18).

[FGH+14a] Jean-Charles Faugère, Pierrick Gaudry, Louise Huot, and Guénaël Renault. "Sub-cubic change of ordering for Gröbner basis: a probabilistic approach". In: *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*. Ed. by Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó. ACM, 2014, pp. 170–177. DOI: 10.1145/2608628.2608669 (cit. on pp. 213, 233, 241, 242).

[FGH+14b] Jean-Charles Faugère, Pierrick Gaudry, Louise Huot, and Guénaël Renault. "Sub-cubic change of ordering for Gröbner basis: a probabilistic approach". In: *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*. Ed. by Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó. ACM, 2014, pp. 170–177. DOI: 10.1145/2608628.2608669 (cit. on p. 225).

[FGL+93] Jean-Charles Faugère, Patrizia M. Gianni, Daniel Lazard, and Teo Mora. "Efficient Computation of Zero-Dimensional Gröbner Bases by Change of Ordering". In: *J. Symb. Comput.* 16.4 (1993), pp. 329–344. DOI: 10.1006/JSCO.1993.1051 (cit. on pp. xvii, 213, 225, 241, 243, 244).

[FJP13] Pierre-Alain Fouque, Jérémy Jean, and Thomas Peyrin. "Structural Evaluation of AES and Chosen-Key Distinguisher of 9-Round AES-128". In: *Advances in Cryptology – CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2013, pp. 183–203. DOI: 10.1007/978-3-642-40041-4_11 (cit. on p. 187).

[FKL+01] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. "Improved Cryptanalysis of Rijndael". In: *Fast Software Encryption – FSE 2000*. Ed. by Bruce Schneier. Vol. 1978. Lecture Notes in Computer Science. New York, NY, USA: Springer, Heidelberg, Germany, 2001, pp. 213–230. DOI: 10.1007/3-540-44706-7_15 (cit. on pp. xiii, 37, 84, 86).

[FLS+10] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. "The Skein hash function family". In: *Submission to NIST (round 3)* 7.7.5 (2010), p. 3 (cit. on p. 18).

[FM17] Jean-Charles Faugère and Chenqi Mou. "Sparse FGLM algorithms". In: *J. Symb. Comput.* 80 (2017), pp. 538–569. DOI: 10.1016/J.JSC.2016.07.025 (cit. on pp. 210, 213, 241, 242).

[Frö85] Ralf Fröberg. "An inequality for Hilbert series of graded algebras". In: *Mathematica Scandinavica* 56.2 (1985), pp. 117–144 (cit. on pp. 211, 212).

[FWG+16] Kai Fu, Meiqin Wang, Yinghua Guo, Siwei Sun, and Lei Hu. "MILP-Based Automatic Search Algorithms for Differential and Linear Trails for Speck". In: *Fast Software Encryption – FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. Lecture Notes in Computer Science. Bochum, Germany: Springer, Heidelberg, Germany, 2016, pp. 268–288. DOI: 10.1007/978-3-662-52993-5_14 (cit. on p. 40).

[Gam84] Taher El Gamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*. Ed. by G. R. Blakley and David Chaum. Vol. 196. Lecture Notes in Computer Science. Springer, 1984, pp. 10–18. DOI: 10.1007/3-540-39568-7_2 (cit. on p. 10).

[Gan90] Felix Ruvimovich Gantmacher. *The Theory of Matrices*. Second. Vol. 1. Chelsea Publishing Company, 1990 (cit. on p. 180).

[GG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra (3. ed.)* Cambridge University Press, 2013. ISBN: 978-1-107-03903-2 (cit. on p. 251).

[GGL+20] Chun Guo, Jian Guo, Eik List, and Ling Song. "Towards Closing the Security Gap of Tweak-aNd-Tweak (TNT)". In: *Advances in Cryptology – ASIACRYPT 2020, Part I*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12491. Lecture Notes in Computer Science. Daejeon, South Korea: Springer, Heidelberg, Germany, 2020, pp. 567–597. DOI: 10.1007/978-3-030-64837-4_19 (cit. on pp. xii, 39, 85).

[GHR+23] Lorenzo Grassi, Yonglin Hao, Christian Rechberger, Markus Schofnegger, Roman Walch, and Qingju Wang. "Horst Meets Fluid-SPN: Griffin for Zero-Knowledge Applications". In: *Advances in Cryptology – CRYPTO 2023, Part III*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14083. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2023, pp. 573–606. DOI: 10.1007/978-3-031-38548-3_19 (cit. on pp. xv, xvii, 200, 201, 214, 239, 245, 253, 254).

[GJV03] Pascal Giorgi, Claude-Pierre Jeannerod, and Gilles Villard. "On the complexity of polynomial matrix computations". In: *Symbolic and Algebraic Computation, International Symposium ISSAC 2003, Drexel University, Philadelphia, Pennsylvania, USA, August 3-6, 2003, Proceedings*. Ed. by J. Rafael Sendra. ACM, 2003, pp. 135–142. DOI: 10.1145/860854.860889 (cit. on p. 244).

[GK08] Shay Gueron and Michael E. Kounavis. "Vortex: A New Family of One-Way Hash Functions Based on AES Rounds and Carry-Less Multiplication". In: *ISC 2008: 11th International Conference on Information Security*. Ed. by Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee. Vol. 5222. Lecture Notes in

Computer Science. Taipei, Taiwan: Springer, Heidelberg, Germany, 2008, pp. 331–340 (cit. on p. 37).

[GKL+22] Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. "Reinforced Concrete: A Fast Hash Function for Verifiable Computation". In: *ACM CCS 2022: 29th Conference on Computer and Communications Security*. Ed. by Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi. Los Angeles, CA, USA: ACM Press, 2022, pp. 1323–1335. DOI: 10.1145/3548606.3560686 (cit. on pp. 201–203).

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. "Poseidon: A New Hash Function for Zero-Knowledge Proof Systems". In: *USENIX Security 2021: 30th USENIX Security Symposium*. Ed. by Michael Bailey and Rachel Greenstadt. USENIX Association, 2021, pp. 519–535 (cit. on pp. xv, xvi, 200–203, 218, 220).

[GLR+20] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. "On a Generalization of Substitution-Permutation Networks: The HADES Design Strategy". In: *Advances in Cryptology – EUROCRYPT 2020, Part II*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Heidelberg, Germany, 2020, pp. 674–704. DOI: 10.1007/978-3-030-45724-2_23 (cit. on p. 218).

[GM16] Shay Gueron and Nicky Mouha. "Simpira v2: A Family of Efficient Permutations Using the AES Round Function". In: *Advances in Cryptology – ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer, Heidelberg, Germany, 2016, pp. 95–125. DOI: 10.1007/978-3-662-53887-6_4 (cit. on pp. 37, 173).

[Gos] *GOST 28147-89*. Tech. rep. Federal Information Processing Standard - Cryptographic Algorithm, Russian National Bureau of Standards, 1989 (cit. on p. 18).

[GØS+23] Lorenzo Grassi, Morten Øygarden, Markus Schofnegger, and Roman Walch. "From Farfalle to Megafono via Ciminion: The PRF Hydra for MPC Applications". In: *Advances in Cryptology – EUROCRYPT 2023, Part IV*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14007. Lecture Notes in Computer Science. Lyon, France: Springer, Heidelberg, Germany, 2023, pp. 255–286. DOI: 10.1007/978-3-031-30634-1_9 (cit. on pp. xv, 200).

[GP10] Henri Gilbert and Thomas Peyrin. "Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations". In: *Fast Software Encryption – FSE 2010*. Ed. by Seokhie Hong and Tetsu Iwata.

Vol. 6147. Lecture Notes in Computer Science. Seoul, Korea: Springer, Heidelberg, Germany, 2010, pp. 365–383. DOI: 10.1007/978-3-642-13858-4_21 (cit. on p. 34).

[Gro96] Lov K. Grover. "A Fast Quantum Mechanical Algorithm for Database Search". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*. Ed. by Gary L. Miller. ACM, 1996, pp. 212–219. DOI: 10.1145/237814.237866 (cit. on p. 10).

[GRR16] Lorenzo Grassi, Christian Rechberger, and Sondre Rønjom. "Subspace Trail Cryptanalysis and its Applications to AES". In: *IACR Transactions on Symmetric Cryptology* 2016.2 (2016). https://tosc.iacr.org/index.php/ToSC/article/view/571, pp. 192–225. ISSN: 2519-173X. DOI: 10.13154/tosc.v2016.i2.192-225 (cit. on p. 121).

[GRR17] Lorenzo Grassi, Christian Rechberger, and Sondre Rønjom. "A New Structural-Differential Property of 5-Round AES". In: *Advances in Cryptology – EUROCRYPT 2017, Part II*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10211. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, 2017, pp. 289–317. DOI: 10.1007/978-3-319-56614-6_10 (cit. on pp. 36, 103, 113).

[Gue08] Shay Gueron. *Intel Advanced Encryption Standard (AES) New Instructions Set*. White Paper Rev 3.01 (09/2012). Intel Corporation, 2008 (cit. on pp. xiv, 37, 170).

[Gur23] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. https://www.gurobi.com. 2023 (cit. on pp. 40, 41, 137, 188).

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge*. Cryptology ePrint Archive, Report 2019/953. https://eprint.iacr.org/2019/953. 2019 (cit. on p. 201).

[HBD+22] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. *SPHINCS$^+$*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. National Institute of Standards and Technology, 2022 (cit. on p. 11).

[HBS21] Hosein Hadipour, Nasour Bagheri, and Ling Song. "Improved Rectangle Attacks on SKINNY and CRAFT". In: *IACR Transactions on Symmetric Cryptology* 2021.2 (2021), pp. 140–198. ISSN: 2519-173X. DOI: 10.46586/tosc.v2021.i2.140-198 (cit. on p. 89).

[HDE24] Hosein Hadipour, Patrick Derbez, and Maria Eichlseder. *Revisiting Differential-Linear Attacks via a Boomerang Perspective with Application to AES, Ascon, CLEFIA, SKINNY, PRESENT, KNOT, TWINE, WARP, LBlock, Simeck, and SERPENT*. Cryptology ePrint Archive, Paper 2024/255. https://eprint.iacr.org/2024/255. 2024 (cit. on p. 40).

[HGS+24] Hosein Hadipour, Simon Gerhalter, Sadegh Sadeghi, and Maria Eichlseder. "Improved Search for Integral, Impossible Differential and Zero-Correlation Attacks Application to Ascon, ForkSKINNY, SKINNY, MANTIS, PRESENT and QARMAv2". In: *IACR Trans. Symmetric Cryptol.* 2024.1 (2024), pp. 234–325. DOI: 10.46586/TOSC.V2024.I1.234-325 (cit. on pp. 47, 67).

[HNE22] Hosein Hadipour, Marcel Nageler, and Maria Eichlseder. "Throwing Boomerangs into Feistel Structures Application to CLEFIA, WARP, LBlock, LBlock-s and TWINE". In: *IACR Transactions on Symmetric Cryptology* 2022.3 (2022), pp. 271–302. DOI: 10.46586/tosc.v2022.i3.271-302 (cit. on p. 89).

[HNS19] S. G. Hyun, V. Neiger, and É. Schost. "Implementations of Efficient Univariate Polynomial Matrix Algorithms and Application to Bivariate Resultants". In: *Proceedings ISSAC 2019*. https://github.com/vneiger/pml. ACM, 2019, pp. 235–242. ISBN: 9781450360845. DOI: 10.1145/3326229.3326272 (cit. on p. 252).

[IAC+08] Sebastiaan Indesteege, Elena Andreeva, Christophe De Cannière, Orr Dunkelman, Emilia Käsper, Svetla Nikova, Bart Preneel, and Elmar Tischhause. *The Lane hash function*. Submission to NIST SHA-3 Cryptographic Hash Algorithm Competition. Available at https://www.cosic.esat.kuleuven.be/lane/index.shtml. 2008 (cit. on p. 37).

[IIL+23] Takanori Isobe, Ryoma Ito, Fukang Liu, Kazuhiko Minematsu, Motoki Nakahashi, Kosei Sakamoto, and Rentaro Shiba. "Areion: Highly-Efficient Permutations and Its Applications to Hash Functions for Short Input". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2023.2 (2023), pp. 115–154. DOI: 10.46586/tches.v2023.i2.115-154 (cit. on p. 37).

[JK97] Thomas Jakobsen and Lars R. Knudsen. "The Interpolation Attack on Block Ciphers". In: *Fast Software Encryption – FSE'97*. Ed. by Eli Biham. Vol. 1267. Lecture Notes in Computer Science. Haifa, Israel: Springer, Heidelberg, Germany, 1997, pp. 28–40. DOI: 10.1007/BFb0052332 (cit. on pp. xvi, 31, 201, 204).

[JKN+24] Ashwin Jha, Mustafa Khairallah, Mridul Nandi, and Abishanka Saha. "Tight security of tnt and beyond: Attacks, proofs and possibilities for the cascaded lrw paradigm". In: *EUROCRYPT 2024* (2024) (cit. on pp. 39, 85, 86, 125).

[JN16] Jérémy Jean and Ivica Nikolic. "Efficient Design Strategies Based on the AES Round Function". In: *Fast Software Encryption – FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. Lecture Notes in Computer Science. Bochum, Germany: Springer, Heidelberg, Germany, 2016, pp. 334–353. DOI: 10.1007/978-3-662-52993-5_17 (cit. on pp. 37, 170–173, 175, 197).

[JNP14a] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. "KIASU v1". In: *Submitted to the CAESAR competition* (2014) (cit. on pp. xi, xii, 39, 47–49, 83, 123).

[JNP14b] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. "Tweaks and Keys for Block Ciphers: The TWEAKEY Framework". In: *Advances in Cryptology – ASIACRYPT 2014, Part II*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. Lecture Notes in Computer Science. Kaoshiung, Taiwan, R.O.C.: Springer, Heidelberg, Germany, 2014, pp. 274–288. DOI: 10.1007/978-3-662-45608-8_15 (cit. on pp. 37, 49, 50, 123, 137).

[JNP+21] Jérémy Jean, Ivica Nikolic, Thomas Peyrin, and Yannick Seurin. "The Deoxys AEAD Family". In: *Journal of Cryptology* 34.3 (July 2021), p. 31. DOI: 10.1007/s00145-021-09397-w (cit. on pp. xii, xiv, 37–39, 83, 89, 133, 137, 170, 185).

[Ker83] A. Kerckhoffs. "La cryptographie militaire". In: *Journal des Sciences Militaires* (1883), pp. 161–191 (cit. on pp. ix, 8).

[KKS01] John Kelsey, Tadayoshi Kohno, and Bruce Schneier. "Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent". In: *Fast Software Encryption – FSE 2000*. Ed. by Bruce Schneier. Vol. 1978. Lecture Notes in Computer Science. New York, NY, USA: Springer, Heidelberg, Germany, 2001, pp. 75–93. DOI: 10.1007/3-540-44706-7_6 (cit. on p. 91).

[KLM+16] Stefan Kölbl, Martin M. Lauridsen, Florian Mendel, and Christian Rechberger. *Haraka - Efficient Short-Input Hashing for Post-Quantum Applications*. Cryptology ePrint Archive, Report 2016/098. https://eprint.iacr.org/2016/098. 2016 (cit. on p. 37).

[KLR24] Katharina Koschatko, Reinhard Lüftenegger, and Christian Rechberger. *Exploring the Six Worlds of Gröbner Basis Cryptanalysis: Application to Anemoi*. Cryptology ePrint Archive, Paper 2024/250. https://eprint.iacr.org/2024/250. 2024 (cit. on p. 212).

[Knu95] Lars R. Knudsen. "Truncated and Higher Order Differentials". In: *Fast Software Encryption – FSE'94*. Ed. by Bart Preneel. Vol. 1008. Lecture Notes in Computer Science. Leuven, Belgium: Springer, Heidelberg, Germany, 1995, pp. 196–211. DOI: 10.1007/3-540-60590-8_16 (cit. on p. 28).

[Kob87] Neal Koblitz. "Elliptic curve cryptosystems". In: *Mathematics of computation* 48.177 (1987), pp. 203–209 (cit. on p. 10).

[KR21a] Nathan Keller and Asaf Rosemarin. "Mind the Middle Layer: The HADES Design Strategy Revisited". In: *Advances in Cryptology – EUROCRYPT 2021, Part II*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12697. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Heidelberg, Germany, 2021, pp. 35–63. DOI: 10.1007/978-3-030-77886-6_2 (cit. on p. 201).

[KR21b] Ted Krovetz and Phillip Rogaway. "The Design and Evolution of OCB". In: *Journal of Cryptology* 34.4 (Oct. 2021), p. 36. DOI: 10.1007/s00145-021-09399-8 (cit. on pp. 37, 170).

[KS07] Liam Keliher and Jiayuan Sui. "Exact maximum expected differential and linear probability for two-round Advanced Encryption Standard". In: *IET Inf. Secur.* 1.2 (2007), pp. 53–57. DOI: 10.1049/IET-IFS:20060161 (cit. on p. 170).

[KSS13] Marcel Keller, Peter Scholl, and Nigel P. Smart. "An architecture for practical actively secure MPC with dishonest majority". In: *ACM CCS 2013: 20th Conference on Computer and Communications Security*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. Berlin, Germany: ACM Press, 2013, pp. 549–560. DOI: 10.1145/2508859.2516744 (cit. on p. 200).

[KT22] Andreas B. Kidmose and Tyge Tiessen. "A Formal Analysis of Boomerang Probabilities". In: *IACR Transactions on Symmetric Cryptology* 2022.1 (2022), pp. 88–109. DOI: 10.46586/tosc.v2022.i1.88-109 (cit. on p. 96).

[KW02] Lars R. Knudsen and David Wagner. "Integral Cryptanalysis". In: *Fast Software Encryption – FSE 2002*. Ed. by Joan Daemen and Vincent Rijmen. Vol. 2365. Lecture Notes in Computer Science. Leuven, Belgium: Springer, Heidelberg, Germany, 2002, pp. 112–127. DOI: 10.1007/3-540-45661-9_9 (cit. on p. 31).

[LDK+22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. *CRYSTALS-DILITHIUM*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. National Institute of Standards and Technology, 2022 (cit. on p. 11).

[LGS17] Guozhen Liu, Mohona Ghosh, and Ling Song. "Security Analysis of SKINNY under Related-Tweakey Settings (Long Paper)". In: *IACR Transactions on Symmetric Cryptology* 2017.3 (2017), pp. 37–72. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i3.37-72 (cit. on pp. 67, 72).

[LH94] Susan K. Langford and Martin E. Hellman. "Differential-Linear Cryptanalysis". In: *Advances in Cryptology – CRYPTO'94*. Ed. by Yvo Desmedt. Vol. 839. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1994, pp. 17–25. DOI: 10.1007/3-540-48658-5_3 (cit. on p. 30).

[LKH+16] HoChang Lee, HyungChul Kang, Deukjo Hong, Jaechul Sung, and Seokhie Hong. *New Impossible Differential Characteristic of SPECK64 using MILP*. Cryptology ePrint Archive, Report 2016/1137. https://eprint.iacr.org/2016/1137. 2016 (cit. on p. 40).

[LKK+08] Jiqiang Lu, Jongsung Kim, Nathan Keller, and Orr Dunkelman. "Improving the Efficiency of Impossible Differential Cryptanalysis of Reduced Camellia and MISTY1". In: *Topics in Cryptology – CT-RSA 2008*. Ed. by Tal Malkin. Vol. 4964. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, 2008, pp. 370–386. DOI: 10.1007/978-3-540-79263-5_24 (cit. on p. 79).

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. "Markov Ciphers and Differential Cryptanalysis". In: *Advances in Cryptology – EUROCRYPT'91*. Ed. by Donald W. Davies. Vol. 547. Lecture Notes in Computer Science. Brighton, UK: Springer, Heidelberg, Germany, 1991, pp. 17–38. DOI: 10.1007/3-540-46416-6_2 (cit. on p. 25).

[LNZ17] George Labahn, Vincent Neiger, and Wei Zhou. "Fast, deterministic computation of the Hermite normal form and determinant of a polynomial matrix". In: *J. Complex.* 42 (2017), pp. 44–71. DOI: 10.1016/J.JCO.2017.03.003 (cit. on p. 262).

[LP21] Gaëtan Leurent and Clara Pernot. "New Representations of the AES Key Schedule". In: *Advances in Cryptology – EUROCRYPT 2021, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Heidelberg, Germany, 2021, pp. 54–84. DOI: 10.1007/978-3-030-77870-5_3 (cit. on pp. 32, 37).

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. "Tweakable Block Ciphers". In: *Advances in Cryptology – CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2002, pp. 31–46. DOI: 10.1007/3-540-45708-9_3 (cit. on p. 15).

[LTX23a] Jiamei Liu, Lin Tan, and Hong Xu. "Improved related-tweakey rectangle attacks on round-reduced Deoxys-BC". In: *IET Inf. Secur.* 17.3 (2023), pp. 407–422. DOI: 10.1049/ISE2.12104 (cit. on p. 89).

[LTX23b] Jiamei Liu, Lin Tan, and Hong Xu. "New Related-Tweakey Boomerang Attacks and Distinguishers on Deoxys-BC". In: *Chinese Journal of Electronics* 33 (2023), pp. 1–11 (cit. on p. 89).

[LWL22] Chenmeng Li, Baofeng Wu, and Dongdai Lin. "Generalized Boomerang Connectivity Table and Improved Cryptanalysis of GIFT". In: *Information Security and Cryptology - 18th International Conference, Inscrypt 2022, Beijing, China, December 11-13, 2022, Revised Selected Papers*. Ed. by Yi Deng and Moti Yung. Vol. 13837. Lecture Notes in Computer Science. Springer, 2022, pp. 213–233. DOI: 10.1007/978-3-031-26553-2_11 (cit. on p. 89).

[LXC+23] Yong Liu, Zejun Xiang, Siwei Chen, Shasha Zhang, and Xiangyong Zeng. "A Novel Automatic Technique Based on MILP to Search for Impossible Differentials". In: *ACNS 23: 21st International Conference on Applied Cryptography and Network Security, Part I*. Ed. by Mehdi Tibouchi and Xiaofeng Wang. Vol. 13905. Lecture Notes in Computer Science. Kyoto, Japan: Springer, Heidelberg, Germany, 2023, pp. 119–148. DOI: 10.1007/978-3-031-33488-7_5 (cit. on p. 40).

[Mat94] Mitsuru Matsui. "Linear Cryptanalysis Method for DES Cipher". In: *Advances in Cryptology – EUROCRYPT'93*. Ed. by Tor Helleseth. Vol. 765. Lecture Notes in Computer Science. Lofthus, Norway: Springer, Heidelberg, Germany, 1994, pp. 386–397. DOI: 10.1007/3-540-48285-7_33 (cit. on pp. 10, 30).

[MDR+10] Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen, and Mahmoud Modarres-Hashemi. "Improved Impossible Differential Cryptanalysis of 7-Round AES-128". In: *Progress in Cryptology - INDOCRYPT 2010: 11th International Conference in Cryptology in India*. Ed. by Guang Gong and Kishan Chand Gupta. Vol. 6498. Lecture Notes in Computer Science. Hyderabad, India: Springer, Heidelberg, Germany, 2010, pp. 282–291 (cit. on p. 52).

[Mer91] Ralph C. Merkle. "Fast Software Encryption Functions". In: *Advances in Cryptology – CRYPTO'90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Vol. 537. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1991, pp. 476–501. DOI: 10.1007/3-540-38424-3_34 (cit. on p. 86).

[Mil85] Victor S. Miller. "Use of Elliptic Curves in Cryptography". In: *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*. Ed. by Hugh C. Williams. Vol. 218. Lecture Notes in Computer Science. Springer, 1985, pp. 417–426. DOI: 10.1007/3-540-39799-X_31 (cit. on p. 10).

[Miy91] Shoji Miyaguchi. "The FEAL Cipher Family (Impromptu Talk)". In: *Advances in Cryptology – CRYPTO'90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Vol. 537. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1991, pp. 627–638. DOI: 10.1007/3-540-38424-3_46 (cit. on p. 86).

[MN17] Bart Mennink and Samuel Neves. "Encrypted Davies-Meyer and Its Dual: Towards Optimal Security Using Mirror Theory". In: *Advances in Cryptology – CRYPTO 2017, Part III*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2017, pp. 556–583. DOI: 10.1007/978-3-319-63697-9_19 (cit. on p. 193).

[Moe73] Robert T. Moenck. "Fast Computation of GCDs". In: *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*. Ed. by Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong. ACM, 1973, pp. 142–151. DOI: 10.1145/800125.804045 (cit. on p. 203).

[Moe76] Robert T. Moenck. "Practical fast polynomial multiplication". In: *Proceedings of the third ACM Symposium on Symbolic and Algebraic Manipulation, SYMSAC 1976, Yorktown Heights, New York, USA, August 10-12, 1976*. Ed. by Richard D. Jenks. ACM, 1976, pp. 136–148. DOI: 10.1145/800205.806332 (cit. on p. 251).

[MP13] Nicky Mouha and Bart Preneel. *Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20*. Cryptology ePrint Archive, Report 2013/328. https://eprint.iacr.org/2013/328. 2013 (cit. on p. 40).

[MS02] Itsik Mantin and Adi Shamir. "A Practical Attack on Broadcast RC4". In: *Fast Software Encryption – FSE 2001*. Ed. by Mitsuru Matsui. Vol. 2355. Lecture Notes in Computer Science. Yokohama, Japan: Springer, Heidelberg, Germany, 2002, pp. 152–164. DOI: 10.1007/3-540-45473-X_13 (cit. on pp. 102, 112).

[MS91] Guillermo Moreno Socias. "Autour de la fonction de Hilbert-Samuel (escaliers d'idéaux polynomiaux)". PhD thesis. Palaiseau, Ecole polytechnique, 1991 (cit. on p. 210).

[MSR14] Marine Minier, Christine Solnon, and Julia Reboul. "Solving a symmetric key cryptographic problem with constraint programming". In: *ModRef 2014, Workshop of the CP 2014 Conference*. 2014, p. 13 (cit. on p. 40).

[MT06] Kazuhiko Minematsu and Yukiyasu Tsunoo. "Provably Secure MACs from Differentially-Uniform Permutations and AES-Based Implementations". In: *Fast Software Encryption – FSE 2006*. Ed. by Matthew J. B. Robshaw. Vol. 4047. Lecture Notes in Computer Science. Graz, Austria: Springer, Heidelberg, Germany, 2006, pp. 226–241. DOI: 10.1007/11799313_15 (cit. on p. 170).

[Mur11] Sean Murphy. "The Return of the Cryptographic Boomerang". In: *IEEE Trans. Inf. Theory* 57.4 (2011), pp. 2517–2521. DOI: 10.1109/TIT.2011.2111091 (cit. on pp. 88, 92).

[MV04] David A. McGrew and John Viega. *The Galois/Counter Mode of Operation (GCM)*. Submission to NIST Modes of Operation Process. Available at https://csrc.nist.rip/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf. Jan. 2004 (cit. on pp. 20, 37, 170).

[MWG+11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. "Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming". In: *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*. Ed. by Chuankun Wu, Moti Yung, and Dongdai Lin. Vol. 7537. Lecture Notes in Computer Science. Springer, 2011, pp. 57–76. DOI: 10.1007/978-3-642-34704-7_5 (cit. on pp. 35, 40, 89, 184).

[NBK08] Ivica Nikolić, Alex Biryukov, and Dmitry Khovratovich. *Hash family LUX - Algorithm Specifications and Supporting Documentation*. Submission to NIST SHA-3 Cryptographic Hash Algorithm Competition. Available at https://ehash.iaik.tugraz.at/uploads/f/f3/LUX.pdf. 2008 (cit. on p. 176).

[NFI24] Yuto Nakano, Kazuhide Fukushima, and Takanori Isobe. *Encryption algorithm Rocca-S*. Internet-Draft draft-nakano-rocca-s-05. Work in Progress. Internet Engineering Task Force, Jan. 2024. 25 pp. (cit. on pp. 37, 170, 172, 197).

[Nik14] Ivica Nikolić. *Tiaoxin–346*. Submission to CAESAR Competition. Available at https://competitions.cr.yp.to/round3/tiaoxinv21.pdf. 2014 (cit. on pp. 37, 170, 172, 197).

[Nik17] Ivica Nikolic. "How to Use Metaheuristics for Design of Symmetric-Key Primitives". In: *Advances in Cryptology – ASIACRYPT 2017, Part III*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10626. Lecture Notes in Computer Science. Hong Kong, China: Springer, Heidelberg, Germany, 2017, pp. 369–391. DOI: 10.1007/978-3-319-70700-6_13 (cit. on pp. 37, 172).

[NN82] Henri J Nussbaumer and Henri J Nussbaumer. *The fast Fourier transform*. Springer, 1982 (cit. on p. 202).

[NS20] Vincent Neiger and Éric Schost. "Computing syzygies in finite dimension using fast linear algebra". In: *J. Complex.* 60 (2020), p. 101502. DOI: 10.1016/J.JCO.2020.101502 (cit. on pp. 213, 241, 242).

[Oki12] Eiji Oki. "GLPK (GNU Linear Programming Kit)". In: 2012 (cit. on p. 41).

[PFH+22] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. *FALCON*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. National Institute of Standards and Technology, 2022 (cit. on p. 11).

[PSS+09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. "Secure Two-Party Computation Is Practical". In: *Advances in Cryptology – ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. Lecture Notes in Computer Science. Tokyo, Japan: Springer, Heidelberg, Germany, 2009, pp. 250–267. DOI: 10.1007/978-3-642-10366-7_15 (cit. on p. 200).

[PT22] Thomas Peyrin and Quan Quan Tan. "Mind Your Path: On (Key) Dependencies in Differential Characteristics". In: *IACR Transactions on Symmetric Cryptology* 2022.4 (2022), pp. 179–207. DOI: 10.46586/tosc.v2022.i4.179-207 (cit. on pp. 25, 92).

[QDW+21] Lingyue Qin, Xiaoyang Dong, Xiaoyun Wang, Keting Jia, and Yunwen Liu. "Automated Search Oriented to Key Recovery on Ciphers with Linear Key Schedule". In: *IACR Transactions on Symmetric Cryptology* 2021.2 (2021), pp. 249–291. ISSN: 2519-173X. DOI: 10.46586/tosc.v2021.i2.249-291 (cit. on pp. xii, 40, 47, 67, 89, 133).

[RBB03] Phillip Rogaway, Mihir Bellare, and John Black. "OCB: A block-cipher mode of operation for efficient authenticated encryption". In: *ACM Trans. Inf. Syst. Secur.* 6.3 (2003), 365–403. ISSN: 1094-9224. DOI: 10.1145/937527.937529 (cit. on pp. 37, 170).

[RBH17] Sondre Rønjom, Navid Ghaedi Bardeh, and Tor Helleseth. "Yoyo Tricks with AES". In: *Advances in Cryptology – ASIACRYPT 2017, Part I*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. Lecture Notes in Computer Science. Hong Kong, China: Springer, Heidelberg, Germany, 2017, pp. 217–243. DOI: 10.1007/978-3-319-70694-8_8 (cit. on pp. 35, 84, 96, 100, 103).

[RD01] Vincent Rijmen and Joan Daemen. "Advanced encryption standard". In: *Proceedings of federal information processing standards publications, national institute of standards and technology* 19 (2001), p. 22 (cit. on p. 32).

[Rog11] Phillip Rogaway. "Evaluation of some blockcipher modes of operation". In: *Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan* 630 (2011) (cit. on p. 15).

[RSA83] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems (Reprint)". In: *Commun. ACM* 26.1 (1983), pp. 96–99. DOI: 10.1145/357980.358017 (cit. on pp. x, 10).

[RSP21] Mostafizar Rahman, Dhiman Saha, and Goutam Paul. "Boomeyong: Embedding Yoyo within Boomerang and its Applications to Key Recovery Attacks on AES and Pholkos". In: *IACR Transactions on Symmetric Cryptology* 2021.3 (2021), pp. 137–169. ISSN: 2519-173X. DOI: 10.46586/tosc.v2021.i3.137-169 (cit. on pp. 84, 89, 96, 109).

[RST23] Arnab Roy, Matthias Johann Steiner, and Stefano Trevisani. "Arion: Arithmetization-Oriented Permutation and Hashing from Generalized Triangular Dynamical Systems". In: *CoRR* abs/2303.04639 (2023). DOI: 10.48550/ARXIV.2303.04639. arXiv: 2303.04639 (cit. on pp. xv, xvii, 200, 201, 214, 239, 258, 259).

[RVP+02] Vincent Rijmen, Bart Van Rompay, Bart Preneel, and Joos Vandewalle. "Producing Collisions for PANAMA". In: *Fast Software Encryption – FSE 2001*. Ed. by Mitsuru Matsui. Vol. 2355. Lecture Notes in Computer Science. Yokohama, Japan: Springer, Heidelberg, Germany, 2002, pp. 37–51. DOI: 10.1007/3-540-45473-X_4 (cit. on p. 176).

[SAB+22] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. *CRYSTALS-KYBER*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. National Institute of Standards and Technology, 2022 (cit. on p. 11).

[SAD20] Alan Szepieniec, Tomer Ashur, and Siemen Dhooghe. *Rescue-Prime: a Standard Specification (SoK)*. Cryptology ePrint Archive, Report 2020/1143. https://eprint.iacr.org/2020/1143. 2020 (cit. on pp. 201, 221, 224).

[Sag9522] The Sage Developers. *SageMath, the Sage Mathematics Software System*. Version 9.5. 2022. DOI: 10.5281/zenodo.6259615 (cit. on p. 252).

[Sas18a] Yu Sasaki. "Improved Related-Tweakey Boomerang Attacks on Deoxys-BC". In: *AFRICACRYPT 18: 10th International Conference on Cryptology in Africa*. Ed. by Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi. Vol. 10831. Lecture Notes in Computer

Science. Marrakesh, Morocco: Springer, Heidelberg, Germany, 2018, pp. 87–106. DOI: 10.1007/978-3-319-89339-6_6 (cit. on pp. xiii, 85, 89, 133, 139, 147).

[Sas18b] Yu Sasaki. "Integer Linear Programming for Three-Subset Meet-in-the-Middle Attacks: Application to GIFT". In: *IWSEC 18: 13th International Workshop on Security, Advances in Information and Computer Security*. Ed. by Atsuo Inomata and Kan Yasuda. Vol. 11049. Lecture Notes in Computer Science. Sendai, Japan: Springer, Heidelberg, Germany, 2018, pp. 227–243. DOI: 10.1007/978-3-319-97916-8_15 (cit. on p. 40).

[Sel08] Ali Aydin Selçuk. "On Probability of Success in Linear and Differential Cryptanalysis". In: *Journal of Cryptology* 21.1 (Jan. 2008), pp. 131–147. DOI: 10.1007/s00145-007-9013-7 (cit. on pp. 115, 116, 122).

[Sha45] Claude E Shannon. "A mathematical theory of cryptography". In: *Mathematical Theory of Cryptography* (1945) (cit. on p. 9).

[Sho] V. et al. Shoup. *NTL: A library for doing number theory*. https://libntl.org (cit. on pp. 212, 251, 252).

[Sho94] Peter W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*. IEEE Computer Society, 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700 (cit. on p. 10).

[Sho96] Victor Shoup. "On Fast and Provably Secure Message Authentication Based on Universal Hashing". In: *Advances in Cryptology – CRYPTO'96*. Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1996, pp. 313–328. DOI: 10.1007/3-540-68697-5_24 (cit. on p. 20).

[SLN+21] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. "Rocca: An Efficient AES-based Encryption Scheme for Beyond 5G". In: *IACR Transactions on Symmetric Cryptology* 2021.2 (2021), pp. 1–30. ISSN: 2519-173X. DOI: 10.46586/tosc.v2021.i2.1-30 (cit. on pp. 37, 170, 172, 174, 175, 185, 197).

[SLN+22] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. *Rocca: An Efficient AES-based Encryption Scheme for Beyond 5G (Full version)*. Cryptology ePrint Archive, Report 2022/116. https://eprint.iacr.org/2022/116. 2022 (cit. on pp. 170, 172).

[SMB18] Sadegh Sadeghi, Tahereh Mohammadi, and Nasour Bagheri. "Cryptanalysis of Reduced round SKINNY Block Cipher". In: *IACR Transactions on Symmetric Cryptology* 2018.3 (2018), pp. 124–162. ISSN: 2519-173X. DOI: 10.13154/tosc.v2018.i3.124-162 (cit. on pp. xii, 67, 71, 72, 74, 76).

[SQH19] Ling Song, Xianrui Qin, and Lei Hu. "Boomerang Connectivity Table Revisited". In: *IACR Transactions on Symmetric Cryptology* 2019.1 (2019), pp. 118–141. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i1.118-141 (cit. on pp. 95, 96, 136).

[SS16] Peter Schwabe and Ko Stoffelen. *All the AES You Need on Cortex-M3 and M4*. Cryptology ePrint Archive, Report 2016/714. https://eprint.iacr.org/2016/714. 2016 (cit. on p. 197).

[SS21] Jan Ferdinand Sauer and Alan Szepieniec. *SoK: Gröbner Basis Algorithms for Arithmetization Oriented Ciphers*. Cryptology ePrint Archive, Report 2021/870. https://eprint.iacr.org/2021/870. 2021 (cit. on p. 201).

[SSD+18] Danping Shi, Siwei Sun, Patrick Derbez, Yosuke Todo, Bing Sun, and Lei Hu. "Programming the Demirci-Selçuk Meet-in-the-Middle Attack with Constraints". In: *Advances in Cryptology – ASIACRYPT 2018, Part II*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11273. Lecture Notes in Computer Science. Brisbane, Queensland, Australia: Springer, Heidelberg, Germany, 2018, pp. 3–34. DOI: 10.1007/978-3-030-03329-3_1 (cit. on p. 40).

[Sti92] Douglas R. Stinson. "Universal Hashing and Authentication Codes". In: *Advances in Cryptology – CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1992, pp. 74–85. DOI: 10.1007/3-540-46766-1_5 (cit. on p. 193).

[Str75] Volker Strassen. "Die Berechnungskomplexität der Symbolischen Differentiation von Interpolationspolynomen". In: *Theor. Comput. Sci.* 1.1 (1975), pp. 21–25. DOI: 10.1016/0304-3975(75)90010-9 (cit. on p. 203).

[SYC+24] Ling Song, Qianqian Yang, Yincen Chen, Lei Hu, and Jian Weng. *Probabilistic Extensions: A One-Step Framework for Finding Rectangle Attacks and Beyond*. Cryptology ePrint Archive, Paper 2024/344. https://eprint.iacr.org/2024/344. 2024 (cit. on pp. 67, 80, 89).

[SZY+22] Ling Song, Nana Zhang, Qianqian Yang, Danping Shi, Jiahao Zhao, Lei Hu, and Jian Weng. "Optimizing Rectangle Attacks: A Unified and Generic Framework for Key Recovery". In: *Advances in Cryptology – ASIACRYPT 2022, Part I*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13791. Lecture Notes in Computer Science.

Taipei, Taiwan: Springer, Heidelberg, Germany, 2022, pp. 410–440. DOI: 10.1007/978-3-031-22963-3_14 (cit. on pp. 89, 92).

[TA14] Yosuke Todo and Kazumaro Aoki. "FFT Key Recovery for Integral Attack". In: *CANS 14: 13th International Conference on Cryptology and Network Security*. Ed. by Dimitris Gritzalis, Aggelos Kiayias, and Ioannis G. Askoxylakis. Vol. 8813. Lecture Notes in Computer Science. Heraklion, Crete, Greece: Springer, Heidelberg, Germany, 2014, pp. 64–81. DOI: 10.1007/978-3-319-12280-9_5 (cit. on p. xiii).

[TAY16] Mohamed Tolba, Ahmed Abdelkhalek, and Amr M. Youssef. "A Meet in the Middle Attack on Reduced Round Kiasu-BC". In: *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 99-A.10 (2016), pp. 1888–1890. DOI: 10.1587/TRANSFUN.E99.A.1888 (cit. on pp. 52, 85).

[tea23] The FLINT team. *FLINT: Fast Library for Number Theory*. Version 3.0.0. 2023. DOI: 10.1109/TC.2017.2690633 (cit. on pp. 224, 251, 252).

[The23] The PML team. *PML: Polynomial Matrix Library*. Version 0.3, https://github.com/vneiger/pml. 2023 (cit. on pp. 252, 262).

[TKK+15] Tyge Tiessen, Lars R. Knudsen, Stefan Kölbl, and Martin M. Lauridsen. "Security of the AES with a Secret S-Box". In: *Fast Software Encryption – FSE 2015*. Ed. by Gregor Leander. Vol. 9054. Lecture Notes in Computer Science. Istanbul, Turkey: Springer, Heidelberg, Germany, 2015, pp. 175–189. DOI: 10.1007/978-3-662-48116-5_9 (cit. on pp. xiii, 36, 84).

[Tod15] Yosuke Todo. "Structural Evaluation by Generalized Integral Property". In: *Advances in Cryptology – EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Sofia, Bulgaria: Springer, Heidelberg, Germany, 2015, pp. 287–314. DOI: 10.1007/978-3-662-46800-5_12 (cit. on p. 201).

[Vau03] Serge Vaudenay. "Decorrelation: A Theory for Block Cipher Security". In: *Journal of Cryptology* 16.4 (Sept. 2003), pp. 249–286. DOI: 10.1007/s00145-003-0220-6 (cit. on p. 86).

[Wag99] David Wagner. "The Boomerang Attack". In: *Fast Software Encryption – FSE'99*. Ed. by Lars R. Knudsen. Vol. 1636. Lecture Notes in Computer Science. Rome, Italy: Springer, Heidelberg, Germany, 1999, pp. 156–170. DOI: 10.1007/3-540-48519-8_12 (cit. on pp. xii, 31, 86, 87, 89, 111).

[WN95] David J. Wheeler and Roger M. Needham. "TEA, a Tiny Encryption Algorithm". In: *Fast Software Encryption – FSE'94*. Ed. by Bart Preneel. Vol. 1008. Lecture Notes in Computer Science. Leuven, Belgium: Springer, Heidelberg, Germany, 1995, pp. 363–366. DOI: 10.1007/3-540-60590-8_29 (cit. on p. 18).

[WN98] David J Wheeler and Roger M Needham. "Correction to xtea". In: *Unpublished manuscript, Computer Laboratory, Cambridge University, England* 1.2 (1998), p. 17 (cit. on p. 18).

[WP13] Hongjun Wu and Bart Preneel. *AEGIS: A Fast Authenticated Encryption Algorithm*. Cryptology ePrint Archive, Report 2013/695. https://eprint.iacr.org/2013/695. 2013 (cit. on p. 197).

[WP14] Hongjun Wu and Bart Preneel. "AEGIS: A Fast Authenticated Encryption Algorithm". In: *SAC 2013: 20th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Tanja Lange, Kristin Lauter, and Petr Lisonek. Vol. 8282. Lecture Notes in Computer Science. Burnaby, BC, Canada: Springer, Heidelberg, Germany, 2014, pp. 185–201. DOI: 10.1007/978-3-662-43414-7_10 (cit. on pp. xiv, 37, 170, 172, 197).

[WP19] Haoyang Wang and Thomas Peyrin. "Boomerang Switch in Multiple Rounds". In: *IACR Transactions on Symmetric Cryptology* 2019.1 (2019), pp. 142–169. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i1.142-169 (cit. on pp. 95, 96, 136).

[WSW+23] Libo Wang, Ling Song, Baofeng Wu, Mostafizar Rahman, and Takanori Isobe. *Revisiting the Boomerang Attack from a Perspective of 3-differential*. Cryptology ePrint Archive, Paper 2023/1689. https://eprint.iacr.org/2023/1689. 2023 (cit. on p. 96).

[XLJ+22] Zheng Xu, Yongqiang Li, Lin Jiao, Mingsheng Wang, and Willi Meier. *Do NOT Misuse the Markov Cipher Assumption - Automatic Search for Differential and Impossible Differential Characteristics in ARX Ciphers*. Cryptology ePrint Archive, Report 2022/135. https://eprint.iacr.org/2022/135. 2022 (cit. on p. 25).

[YSS+22] Qianqian Yang, Ling Song, Siwei Sun, Danping Shi, and Lei Hu. "New Properties of the Double Boomerang Connectivity Table". In: *IACR Transactions on Symmetric Cryptology* 2022.4 (2022), pp. 208–242. DOI: 10.46586/tosc.v2022.i4.208-242 (cit. on pp. 95, 137).

[YTX+24] Xueping Yan, Lin Tan, Hong Xu, and Wen-Feng Qi. "Improved mixture differential attacks on 6-round AES-like ciphers towards time and data complexities". In: *J. Inf. Secur. Appl.* 80 (2024), p. 103661. DOI: 10.1016/J.JISA.2023.103661 (cit. on p. 84).

[YZS+15] Gangqiang Yang, Bo Zhu, Valentin Suder, Mark D. Aagaard, and Guang Gong. "The Simeck Family of Lightweight Block Ciphers". In: *Cryptographic Hardware and Embedded Systems – CHES 2015*. Ed. by Tim Güneysu and Helena Handschuh. Vol. 9293. Lecture Notes in Computer Science. Saint-Malo, France: Springer, Heidelberg, Germany, 2015, pp. 307–329. DOI: 10.1007/978-3-662-48324-4_16 (cit. on p. 18).

[ZBL+15] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. "RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms". In: *Sci. China Inf. Sci.* 58.12 (2015), pp. 1–15. DOI: 10.1007/S11432-015-5459-7 (cit. on p. 18).

[ZCJ21] Jiyan Zhang, Ting Cui, and Chenhui Jin. "New Rectangle Attack Against SKINNY Block Cipher". In: *Wireless Algorithms, Systems, and Applications - 16th International Conference, WASA 2021, Nanjing, China, June 25-27, 2021, Proceedings, Part III*. Ed. by Zhe Liu, Fan Wu, and Sajal K. Das. Vol. 12939. Lecture Notes in Computer Science. Springer, 2021, pp. 399–409. DOI: 10.1007/978-3-030-86137-7_43 (cit. on p. 89).

[ZD19] Rui Zong and Xiaoyang Dong. "MILP-Aided Related-Tweak/Key Impossible Differential Attack and its Applications to QARMA, Joltik-BC". In: *IEEE Access* 7 (2019), pp. 153683–153693. DOI: 10.1109/ACCESS.2019.2946638 (cit. on p. 40).

[ZDJ19] Boxin Zhao, Xiaoyang Dong, and Keting Jia. "New Related-Tweakey Boomerang and Rectangle Attacks on Deoxys-BC Including BDT Effect". In: *IACR Transactions on Symmetric Cryptology* 2019.3 (2019), pp. 121–151. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i3.121-151 (cit. on pp. xiii, 40, 85, 133, 139).

[ZDJ+19] Boxin Zhao, Xiaoyang Dong, Keting Jia, and Willi Meier. "Improved Related-Tweakey Rectangle Attacks on Reduced-Round Deoxys-BC-384 and Deoxys-I-256-128". In: *Progress in Cryptology - INDOCRYPT 2019: 20th International Conference in Cryptology in India*. Ed. by Feng Hao, Sushmita Ruj, and Sourav Sen Gupta. Vol. 11898. Lecture Notes in Computer Science. Hyderabad, India: Springer, Heidelberg, Germany, 2019, pp. 139–159. DOI: 10.1007/978-3-030-35423-7_7 (cit. on pp. xiii, 85, 89, 133, 139).

[ZDM+20] Boxin Zhao, Xiaoyang Dong, Willi Meier, Keting Jia, and Gaoli Wang. "Generalized related-key rectangle attacks on block ciphers with linear key schedule: applications to SKINNY and GIFT". In: *Des. Codes Cryptogr.* 88.6 (2020), pp. 1103–1126. DOI: 10.1007/S10623-020-00730-1 (cit. on p. 89).

[ZDY19] Baoyu Zhu, Xiaoyang Dong, and Hongbo Yu. "MILP-Based Differential Attack on Round-Reduced GIFT". In: *Topics in Cryptology – CT-RSA 2019*. Ed. by Mitsuru Matsui. Vol. 11405. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, 2019, pp. 372–390. DOI: 10.1007/978-3-030-12612-4_19 (cit. on p. 40).

[ZLL+23] Lulu Zhang, Meicheng Liu, Shuaishuai Li, and Dongdai Lin. "Cryptanalysis of Ciminion". In: *Information Security and Cryptology*. Ed. by Yi Deng and Moti Yung. Cham: Springer Nature Switzerland, 2023, pp. 234–251. DOI: 10.1007/978-3-031-26553-2_12 (cit. on pp. 201, 230, 235).

[ZZD+19] Chunning Zhou, Wentao Zhang, Tianyou Ding, and Zejun Xiang. "Improving the MILP-based Security Evaluation Algorithm against Differential/Linear Cryptanalysis Using A Divide-and-Conquer Approach". In: *IACR Transactions on Symmetric Cryptology* 2019.4 (2019), pp. 438–469. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i4.438-469 (cit. on p. 40).

[ZZY+23] Jiahao Zhao, Nana Zhang, Qianqian Yang, Ling Song, and Lei Hu. "Improved Boomerang Attacks on Deoxys-BC". In: *Advances in Information and Computer Security - 18th International Workshop on Security, IWSEC 2023, Yokohama, Japan, August 29-31, 2023, Proceedings*. Ed. by Junji Shikata and Hiroki Kuzuno. Vol. 14128. Lecture Notes in Computer Science. Springer, 2023, pp. 59–76. DOI: 10.1007/978-3-031-41326-1_4 (cit. on p. 89).