N° d'ordre NNT : xxx

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

Membre de l'Université de Lyon

École Doctorale N° 488
SIS - SCIENCES INGÉNIERIE SANTÉ

Spécialité de doctorat: Informatique

Soutenue publiquement le 19/12/2025, par:

Viet-Sang Nguyen

Physical Attacks against Symmetric Cryptography Standards

Devant le jury composé de :

François-Xavier Standaert	Université catholique de Louvain	Rapporteur
Pierre-Alain Fouque	Université de Rennes	Rapporteur
Christina Boura	Université Paris Cité	Présidente du jury
Matthieu Rivain	CryptoExperts	Examinateur
Elisabeth Oswald	Université de Klagenfurt	Examinatrice
Romain Poussier	ANSSI	Invité
Pierre-Louis Cayrel	Université Jean Monnet	Directeur de thèse
Vincent Grosso	CNRS	Encadrant

Acknowledgements

I would never have imagined that one day I would achieve the title of Doctor. I used to be afraid of pursuing a PhD because I thought I was not suited for research, and I believed engineering would be more interesting. Along the way, however, I met many friends, had the chance to work with amazing people, and observed how true experts approach problems. This gradually changed the perspective in my mind. I began to realize how much better things can be done with the skills and knowledge gained through research. This was one of the main reasons I chose to pursue a PhD. Now, as I reach the end of my PhD journey, I would like to express my gratitude to all those, without whom, this thesis would not exist.

En premier lieu, je tiens à remiercier mon encadrant, Vincent Grosso, pour son soutien tout au long ma thèse, non seulement sur les aspects scientifiques mais aussi dans les démarches administratives. Il a toujours été à l'écoute de mes idées (parfois un peu "crazy"), et m'a constamment encouragé même lorsque moi-même je doutais de leur pertinence. Je le remercie également de m'avoir laissé la liberté d'explorer les sujets qui m'intéressaient, et d'avoir soutenu mes demandes, parfois inédites dans notre laboratoire.

Ensuite, j'aimerais remercier mon directeur de thèse, Pierre-Louis Cayrel, pour son aide dans les démarches administratives et pour la correction de mes articles. Je le remercie aussi pour la confiance qu'il m'a accordée en me laissant donner des séances de TP malgré mon niveau de français à l'époche. Cela m'a beaucoup aidé à progresser.

Je tiens à exprimer mes sincères remerciements à Pierre-Alain Fouque et François-Xavier Standaert qui ont accepté de rapporter mon manuscrit. Je les remercie d'avoir pris le temps de le relire et pour leurs précieuses remarques.

I would like to thank Elisabeth Oswald for accepting to be in the jury. It is a great honor to have the author of the classical textbook on power analysis attacks, which has been on my desk since the start of my PhD, as a member of the jury. Je remercie Christina Boura d'avoir accepté de faire le déplacement. Je suis très heureux qu'elle fasse partie du jury. Je remercie Romain Poussier pour ses remarques constructives lors de chaque commité de suivi, et pour avoir accepté de participer à mon jury.

Je souhaite remercier sincèrement Matthieu Rivain, avec qui j'ai eu le privilège de travailler. Grâce à la période passée à travailler avec lui en tant qu'ingénieur avant ma thèse, j'ai acquis ma première expérience de recherche avec le projet Obscure. Je suis très heureux de sa participation à mon jury. Je tiens à étendre mes remerciements à Darius Mercadier et Aleksei Udovenko, qui ont également travaillé sur le projet Obscure. J'ai appris beaucoup sur les compilateurs et la white-box crypto grâce à eux.

J'aimerais addresser mes remerciements à l'équipe SESAM, dont j'ai eu le privilège de faire partie pendant presque trois ans. Je remercie Lilian Bossuet pour ses indications concernant les missions, Brice Colombier pour son soutien avec le ChipWhisperer,

Paul Grandamme pour m'avoir ramené du IUT au laboratoire, Natalie Bochard et Éloïse Delolme pour leur chaleur humaine, Cédric Killian pour son sens de l'humour, Pierre-Antoine Tissot pour m'avoir passé le sujet de PFA, Alexandre Ortega pour les longues histoires, Simon Baissat-Chavent pour avoir cherché à manger ensemble le midi, Mathieu Descos pour sa magie, Jorge Hoyos Ramirez pour son énergie positive et les bonbons, Jose Bernal et William Pensec pour nos discussions sur des implémentations hardware, Natacha Muel Camenen pour sa sympathie, Tom Fleuranceau pour avoir chanté Les Champs-Élysées avec moi dans le bureau, Nicolas Vallet pour avoir laissé plein de petits visages souriants dans mon cahier, et Justine Paillet pour m'avoir montré les photos de son adorable chat. Je remercie aussi les autres membres de l'équipe que j'ai rencontrés: Florent Bernard, Viktor Fischer, Mateus Simões, Arturo Garay, Carlos Lara, Anis Fella-Touta, Raphaële Milan.

Je tiens à spécialement remercier Nicolas Vallet. Il m'a aidé à améliorer mon français grâce à sa patience et sa méticulosité lorsqu'il corrigeait mes phrases bizarres. Je remercie également Justine pour ses explications très claires des mots français, et Jorge pour avoir pris le temps de corriger le résumé de ce manuscrit. En parlant de langue, j'aimerais remercier ma professeure particulière, Van, qui m'enseigne depuis quatre ans.

Au laboratoire, j'ai reçu l'aide de plusieurs personnes à part mon équipe. Je remercie Julie Debiesse pour son aide dans les démarches administratives, François Martin pour son aide avec mon ordinateur, Thomas Gautrais pour son assistance avec le cluster (et pour l'organisation des tournois du foot), Stéphane Criedlich pour son aide dans mes missions, Fadoua Lafdil pour son aide avec l'école doctorale.

Je ne serais pas arrivé là où je suis aujourd'hui sans le soutien de Hieu Phan depuis mes premiers jours en France pour le master. Je tiens à le remercier sincèrement pour ses conseils tout au long de ma carrière. Je le remercie aussi d'avoir organisé des fêtes chez lui, où j'ai l'occasion de rencontrer les jeunes cryptographeurs vietnamiens comme Ky Nguyen, Ngoc Khanh Nguyen, Huyen Nguyen, ainsi que le hacker professionnel Thai Duong. *I would like to thank Thai Duong for his inspiring talks about real-world crypto prolems, and for his blog posts.*

(Em xin cảm ơn thầy Hiệu vì sự giúp đỡ, những lời khuyên trong những năm qua, và những bữa tiệc tại nhà thầy.)

I am so happy to have the friends who have been with me for six years, since my very first days in France for the master. I would like to thank Dr. Dung Bui, Dr. Hoa Nguyen, Dr. Linh Le (they have recently sucessfully defended their PhDs), and Huyen Tran. They have always supported me in many aspects of life, from encouragement after a rejection to answering random questions like where to buy things cheaper. A special thanks goes to Linh, who has been my teammate for six years in the NSUCRYPTO competition, from winning the third prizes to first prizes.

(Cảm ơn Linh, Dung, chị Hoa, chị Huyền vì đã đi cùng nhau, giúp đỡ nhau từ thời master đến bây giờ.)

I would like give a special thanks to my homie, Ky Nguyen, for our friendship since the bachelor in Vietnam, for hosting me whenever I stayed in Paris, and for all the (serious) advice. I still remember the night I called him at 0:30 AM to ask whether

I should accept this PhD offer.

(Cảm ơn Kỷ vì lời khuyên năm xưa, và cả những hôm uống thâu đêm.)

During my PhD, I had the chance to meet, make friends, discuss, and hang out with many people in international conferences. I would like to thank Trevor Yap for our discussion about my work, and Phuong Pham for sharing her PhD experience. I am also happy to have met and chatted with my dear "old" friends: Nicolas Bon, Thibauld Feneuil, and Abdul Rahman Taleb. I would like to thank Duyen Pay for our discussions about SCA and for the enjoyable moments exploring Kuala Lumpur together. I would also like to thank Tien Nguyen for the nice trip together during the summer school in Sophia-Antipolis.

(Cảm ơn chị Duyên vì đã cùng than thở về đề tài luận văn trong mấy năm qua. Cảm ơn anh Tiến đã cùng đi chuyến đi miền nam.)

I am much happier living in Saint-Étienne thanks to the many friends around me. I sincerely thank Ngan, Louis, la petite Pia, Trang, Fabien, Hai, Bastien, Mi, Doan and Duong (le chef) for the fun and memorable parties at Ngan and Louis' home. A special thanks goes to Ngan for the advice about relationships. Another special thanks goes to Hai for all the gossips we shared and for the spontaneous restaurant outings.

(Cảm ơn chị Ngân vì những bữa tiệc và những lời khuyên. Cảm ơn Hải vì cùng tám những câu chuyện ở nơi làm việc.)

La vie devient plus agréable et moins stressante chaque fois que je suis sur le terrain de foot. Je remercie mes amis du foot à Saint-Étienne: Mounir, Victor, Mouloud, Anas, Anders, Raza, Kaïs, Prashant, et bien d'autres joueurs occasionnels, pour avoir joué ensemble chaque vendredi soir. Je souhaite particulièrement remercier Mounir pour m'avoir ramené après les matches, et Victor pour nos échanges sur la SCA.

I sincerely thank my high school friends (known as the "CL group") for always giving me the warmest welcome, year after year, whenever I return home. I especially thank Hoang Anh for riding with me on motorbikes to visit the old corners, and for always being there to give me a hand.

(Cảm ơn tập thể Chuyên Lý đã luôn đón tiếp nồng hậu mỗi khi mình về. Cảm ơn Hoàng Anh vì tình bạn đã được 13 năm và những sự giúp đỡ tốt bụng mà không thể kể hết được.)

J'aimerais exprimer ma gratitude à Giang, qui m'a toujours écouté et fait l'effort de comprendre mon travail. Elle m'a accompagné et encouragé dans les moments de stress. Avec son énergie positive, elle m'a apporté beaucoup de bonheur dans ma vie et m'a aidé à ne pas trop m'inquiéter pour le futur.

I would like to wholeheartedly thank my family for their endless support. I am so graful to my parents, Cuc and Loan, for their incredible hard work in the coffee fields to raise me and my siblings. I thank my sister, Phuong, for taking care of our parents while my brother and I are working far from home. I thank my brother, Nam, for guiding my educational path since the early days of primary school. Without his guidance, I would probably have become a high school Physics teacher in my hometown.

(Con cảm ơn Ba Cúc và Mẹ Loan vì sự nuôi nấng, tình thương, sự hỗ trợ về vật chất và tinh thần suốt bao nhiêu năm qua. Em cảm ơn chị Hai (chị Phương) đã lo toan nhiều việc trong nhà để em có thể yên tâm làm việc ở một đất nước rất xa. Em cảm ơn anh Ba (anh Nam) vì những định hướng ngành học ngày xưa đã giúp em tiến xa được như ngày hôm nay.)

Abstract

The rise of embedded devices in the era of Internet of Things (IoT) has increased the demand for secure cryptographic algorithms. While standardized algorithms are proven to be secure in the black-box model, where an adversary has access only to inputs and outputs, this model is not sufficient to capture real-world threats. In practice, the adversary may have physical access to the devices and recover the secret key used in the algorithms by physical attacks, which are generally classified as Side-Channel Attacks (SCA) and Fault Attacks (FA). This thesis explores the threats of such attacks on symmetric cryptography standards, with a focus on the long-standing Advanced Encryption Standard (AES) and the newly standardized lightweight authenticated encryption Ascon-AEAD.

In the first part of this thesis, we focus on a prominent type of SCA, namely Correlation Power Analysis (CPA), targeting Ascon-AEAD. We analyze different approaches for choosing the selection function, which is a core factor of CPA, to provide insights into when and why an attack succeeds or fails. Our analysis is validated with a second-order CPA attack against a masked software implementation, which also provides the first practical indication of the data and computational cost for full key recovery in a protected implementation. We further propose extending the selection function from one bit to multiple bits, thereby increasing success rates.

In the second part of this thesis, we focus on two prominent types of FA, namely Statistical Ineffective Fault Analysis (SIFA) and Persistent Fault Analysis (PFA), targeting Ascon-AEAD and AES. For SIFA, we provide a deeper analysis of a generic attack strategy applied to nonce-based authenticated encryption schemes. Our results show that, although SIFA is powerful, it may fail in practice under instruction-skip faults. These findings are supported by demonstrations on Ascon-AEAD. For PFA, we show that persistent faults can be induced by an instruction skip, which is simpler than memory faulting techniques reported in the literature. We also introduce the first PFA targeting a constant other than S-box elements. Specifically, we demonstrate that faulting a round constant of AES can lead to full key recovery. Our findings are supported by experiments on the AES implementation in the widely used MbedTLS library.

Résumé

L'essor des dispositifs embarqués à l'ère de l'Internet des Objets a accru la demande en algorithmes cryptographiques sécurisés. Bien que les algorithmes standardisés soient prouvés sécurisés dans le modèle de boîte noire, où un adversaire a seulement accès aux entrées et aux sorties, ce modèle n'est pas suffisant pour représenter les menaces du monde réel. En pratique, l'adversaire peut avoir un accès physique aux dispositifs et récupérer la clé secrète utilisée dans les algorithmes au moyen d'attaques physiques. Ces attaques sont généralement classées en deux catégories : les attaques par canaux auxiliaires (*Side-Channel Attacks*, SCA) et les attaques par injection de fautes (*Fault Attacks*, FA). Cette thèse étudie les menaces posées par de telles attaques contre les standards de la cryptographie symétrique, en se concentrant sur l'*Advanced Encryption Standard* (AES), largement utilisé depuis de nombreuses années, ainsi que sur Ascon-AEAD, récemment standardisé comme schéma de chiffrement authentifié léger.

Dans la première partie de cette thèse, nous nous concentrons sur un type majeur de SCA, à savoir l'analyse de corrélation de consommation de puissance (*Correlation Power Analysis*, CPA), ciblant Ascon-AEAD. Nous étudions différentes approches pour choisir la fonction de sélection, un élément central de la CPA, afin de mieux comprendre les conditions dans lesquelles l'attaque réussit ou échoue. Notre analyse est validée par une attaque CPA d'ordre deux contre une implémentation logicielle masquée. Cela fournit également la première estimation pratique du coût en données et en calcul pour la récupération complète de la clé dans une implémentation protégée. De plus, nous proposons d'étendre la fonction de sélection d'un bit à plusieurs bits, ce qui permet d'augmenter le taux de réussite de l'attaque.

Dans la deuxième partie de cette thèse, nous nous concentrons sur deux types majeurs de FA, à savoir l'analyse statistique des fautes inefficaces (*Statistical Ineffective Fault Analysis*, SIFA) et l'analyse de fautes persistantes (*Persistent Fault Analysis*, PFA), ciblant Ascon-AEAD et AES. Pour la SIFA, nous proposons une analyse approfondie d'une stratégie d'attaque générique appliquée aux schémas de chiffrement authentifiés. Nos résultats montrent que, bien que la SIFA soit généralement puissante, elle peut échouer en pratique sous des fautes de type saut d'instruction. Ces conclusions sont illustrées par des démonstrations sur Ascon-AEAD. Pour la PFA, nous montrons que des fautes persistantes peuvent être induites par un saut d'instruction, ce qui est plus simple que les techniques d'altération de la mémoire rapportées dans la littérature. Nous présentons également la première PFA ciblant une constante autre que les éléments de la S-box. Spécifiquement, nous démontrons que l'altération d'une constante de tour d'AES peut conduire à la récupération complète de la clé. Nos résultats sont illustrés par des expériences sur l'implémentation AES de la bibliothèque MbedTLS largement utilisée.

Tóm tắt

Sự gia tăng của các thiết bị nhúng trong kỷ nguyên Internet vạn vật đã làm tăng nhu cầu về các thuật toán mật mã an toàn. Mặc dù các thuật toán chuẩn đã được chứng minh là an toàn trong mô hình hộp đen, nơi kẻ tấn công chỉ có thể truy cập đầu vào và đầu ra, mô hình này vẫn chưa đủ để phản ánh các mối nguy hiểm. Trên thực tế, kẻ tấn công có thể tiếp cận các thiết bị và lấy khóa bí mật được sử dụng trong các thuật toán thông qua các cuộc tấn công vật lý. Các cuộc tấn công này thường được phân loại thành tấn công kênh bên (Side-Channel Attacks, SCA) và tấn công lỗi (Fault Attacks, FA). Luận án này khảo sát các mối nguy hiểm từ các cuộc tấn công này trên các chuẩn mật mã đối xứng, bao gồm chuẩn mã hóa nâng cao (Advanced Encryption Standard, AES) đã được sử dụng lâu năm và Ascon-AEAD, một chuẩn mã hóa xác thực mới được chuẩn hóa.

Trong phần đầu của luận án, chúng tôi tập trung vào một loại tấn công kênh bên nổi bật nhắm vào Ascon-AEAD, đó là phân tích tương quan công suất tiêu thụ (Correlation Power Analysis, CPA). Chúng tôi phân tích các phương pháp khác nhau để thiết lập hàm lựa chọn, một yếu tố cốt lõi của CPA, nhằm giải thích khi nào và tại sao một cuộc tấn công thành công hay thất bại. Phân tích của chúng tôi được kiểm chứng bằng một cuộc tấn công CPA bậc hai trên một phần mềm đã được bảo vệ bằng phương pháp che giấu. Tấn công này đồng thời cung cấp một kết quả thực tế đầu tiên về chi phí dữ liệu và tính toán để lấy được khóa bí mật trong một phần mềm được bảo vệ. Tiếp theo đó, chúng tôi đề xuất mở rộng hàm lựa chọn từ một bit sang nhiều bit, từ đó tăng tỷ lệ thành công của cuộc tấn công.

Trong phần thứ hai của luận án, chúng tôi tập trung vào hai loại tấn công lỗi nổi bật nhắm vào Ascon-AEAD và AES, đó là phân tích thống kê lỗi không hiệu quả (Statistical Ineffective Fault Analysis, SIFA) và phân tích lỗi lâu dài (Persistent Fault Analysis, PFA). Đối với SIFA, chúng tôi thực hiện phân tích sâu hơn về một chiến lược tấn công áp dụng cho các sơ đồ mã hóa xác thực. Kết quả cho thấy, mặc dù SIFA là một tấn công có hiệu quả cao, nhưng có thể thất bại trong thực tế dưới các lỗi bỏ qua dòng lệnh. Những kết quả này được chứng minh trên Ascon-AEAD. Đối với PFA, chúng tôi chỉ ra rằng các lỗi lâu dài có thể được tạo ra bởi một lệnh bị bỏ qua. Điều này đơn giản hơn so với các kỹ thuật làm lỗi bộ nhớ được báo cáo trong các nghiên cứu trước đây. Chúng tôi cũng giới thiệu một tấn công PFA đầu tiên nhắm vào một hằng số khác ngoài các phần tử trong S-box. Cụ thể, làm lỗi một hằng số vòng lặp của AES có thể dẫn đến lấy được khóa bí mật. Các kết quả này được thí nghiệm trên phần mềm AES trong thư viện MbedTLS được dùng rộng rãi.

Contents

List of Figures

List of Tables

Abbreviations

AES	Advanced Encryption Standard
AEAD	Authenticated Encryption with Associated Data
CPA	Correlation Power Analysis
DFA	Differential Fault Analysis
DMR	Dual Modular Redundancy
DPA	Differential Power Analysis
ECC	Error Correction Codes
EM	ElectroMagnetic
EMFI	ElectroMagnetic Fault Injection
FA	Fault Attacks
HW	Hamming Weight
IFA	Ineffective Fault Analysis
IoT	Internet of Things
NIST	U.S. National Institute of Standards and Technology
PFA	Persistent Fault Analysis
RAM	Random-Access Memory
ROM	Read-Only Memory
SASCA	Soft Analytical Side-Channel Analysis
SCA	Side-Channel Attacks
SFA	Statistical Fault Analysis
SIFA	Statistical Ineffective Fault Analysis

Introduction en français

Ce chapitre introduit le contexte général et la motivation de notre étude. Nous présentons ensuite les principales contributions de cette thèse. Puis, nous donnons un aperçu de l'organisation du manuscrit.

Cadre et Motivation

Depuis très longtemps, la protection des communications entre un *émetteur* et un *récepteur* contre un *adversaire* constitue un problème essentiel. Ce problème remonte à l'histoire avec l'exemple du chiffrement de César et existe encore aujourd'hui dans les communications sur Internet. L'étude et la mise en œuvre de tels mécanismes de protection sont appelées *cryptographie*. L'adversaire, également appelé un *attaquant*, est une partie qui pourrait espionner les échanges, modifier les messages transmis, ou se faire passer pour un émetteur légitime. Dans ce contexte, les solutions cryptographiques visent à garantir les propriétés suivantes:

- Confidentialité: garantir que les messages sont incompréhensibles pour les récepteurs imprévus.
- Intégrité: garantir que les messages ne sont pas modifiés ou altérés lors de la transmission.
- Authenticité: garantir que les messages proviennent bien de l'émetteur prévu.

En cryptographie, la *cryptographie symétrique* est une branche importante dans laquelle l'émetteur et le récepteur partagent une *clé sécrete* commune. L'émetteur utilise cette clé pour *chiffrer* un message, appelé *texte en clair*, en un *chiffré*, qu'il envoie au récepteur. Le récepteur utilise ensuite la même clé pour déchiffrer le chiffré et récupérer le texte en clair original. Le secret de la clé détermine la sécurité des systèmes de communication.

De nos jours, à l'ère de l'Internet des objets, l'utilisation des dispositifs embarqués, tels que les capteurs, les cartes à puces, les clés à puces et les passeports électroniques, devient de plus en plus fréquente. On estime qu'il y a environ 30 dispositifs embarqués par personne [Cen14]. Afin d'assurer une securité fiable, les algorithmes cryptographiques standardisés par le *National Institute of Standards and Technology* (NIST) des États-Unis sont souvent utilisés dans ces dispositifs.

En particulier, les deux standards de la cryptographie symétrique sont probalement largement déployés dans les dispositifs embarqués: l'*Advanced Encryption Standard* (AES) [DR02] et l'Ascon-*based Authenticated Encryption with Associated Data* (Ascon-AEAD) [DEM+21]. Le premier est reconnu pour sa robustesse depuis sa standardisation en 2001. Le deuxième, qui a été sélectionné pour la standardisation en 2023, est spécialement conçu pour être légér dans les scénarios où l'AES est trop gourmand en ressources [MBT+17]. Dans les dispositifs embarqués, les clés secrètes sont stockées dans leurs mémoires. Il est donc très important de les protéger contre

l'exposition à des adversaires potentiels.

Ayant été qualifiés lors des compétitions du NIST, les deux standards symétriques ont été analysés de manière approfondie. Cependent, ces analyses considèrent souvant un *modèle de boîte noire*, dans lequel l'adversaire n'a accès qu'aux entrées et sorties. Un tel modèle n'est pas toujours suffisant pour garantir la sécurité en pratique. Lorsqu'ils sont implémentés et déployés dans des dispositifs embarqués, l'adversaire peut y avoir un accès physique. Dans ce scénario, les algorithmes cryptographiques peuvent être vulnérables aux attaques physiques, qui sont généralement classées en deux groupes: les attaques par canaux auxiliaires (*Side-Channel Attacks*, SCA) et les attaques par injection de fautes (*Fault Attacks*, FA). Ces attaques sont reconnues comme des ménaces significatives depuis les travaux séminales sur l'analyse differentielle de puissance (*Differential Power Analysis*) de Kocher *et al.* [KJJ99] et sur l'analyse differentielle de fautes (*Differential Fault Analysis*) de Biham et Shamir [BS97].

Parce que les SCA et FA posent des risques réels en pratique, la compréhension des menaces potentielles nous aide à développer des implémentations plus robustes pour un déploiement à grande échelle, ainsi qu'à éviter des compromissions de sécurité catastrophiques. Il est donc très important d'étudier les attaques physiques, ce qui est particulièrement très pertinent pour Ascon-AEAD, récemment standardisé. Motivée par ce besoin, cette thèse étudie les menaces posées par les SCA et FA sur les deux standards de la cryptographie symétrique: AES et Ascon-AEAD.

Principales Contributions

Dans cette thèse, nous présentons deux parties principales de nos contributions, l'une portant sur les SCA et l'autre sur les FA.

Dans la première partie (Part I), nous étudions un type remarquable de SCA ciblant Ascon-AEAD, appelé l'analyse de corrélation de puissance (*Correlation Power Analysis*, CPA). Cette étude donne lieu aux deux publications suivantes:

- **Attaque CPA d'ordre deux avec une fonction de sélection appropriée [NGC25c]**: Dans ce travail, nous menons une analyse complète des approches differentes dans la littérature pour choisir la fonction de sélection, utilisée pour calculer les valeurs intermédiaires dans les attaques CPA. Par des explications théoriques et des validations expérimentales, nous montrons comment ces choix influencent le succès des attaques. Profitant des connaissances acquises grâce à notre analyse, nous réalisons la première attaque d'ordre deux réussie contre une implémentation masquée de Ascon-AEAD en logiciel.
- **Fonction de sélection à plusieurs bits pour l'attaque CPA [NGC25a]**: Dans ce travail, nous étendons la fonction de sélection à plusieurs bits au lieu d'un seul. Nous présentons des résultats expérimentaux pour montrer les avantages de cette approche en termes de taux de réussite et du nombre de traces nécessaires pour les attaques CPA.

Dans la deuxième partie (Part II), nous étudions deux types remarquables de FA

ciblant Ascon-AEAD et AES, appelées l'analyse statistique des fautes inefficaces (*Statistical Ineffective Fault Analysis*, SIFA) et l'analyse des fautes persistentes (*Persistent Fault Analysis*, PFA). Cette étude donne lieu aux deux publications suivantes:

- **Analyse statistique des fautes inefficaces sur Ascon-AEAD [NGC25d]**: Dans ce travail, nous présentons une analyse approfondie de la stratégie d'attaque utilisée dans la littérature pour appliquer la SIFA aux schémas de chiffrement authentifié. Nous montrons que cette stratégie peut échouer lorsqu'elle est appliquée à certaines implémentations, pour deux raisons principales: la faible probabilité d'occurrence d'une faute inefficace requise pour l'attaque, et l'uniformité de la valeur intermédiaire choisie. Nous validons nos résultats par des simulations sur une implémentation 8 bits de Ascon-AEAD.
- **Attaques par fautes persistantes sur AES avec saut d'instruction [NGC25b]**: Dans ce travail, nous montrons que les fautes persistentes peuvent être injectées non seulement en altérant les éléments de la S-box en mémoire, comme vu dans la littérature, mais aussi par une méthode d'injection très courante, appelée saut d'instruction. Nous introduisons ensuite la première attaque PFA, qui cible une constante autre que les éléments de la S-box. Plus précisément, nous montrons qu'une faute injectée sur une constante de tour est suffisante pour permettre la récupéreration de la clé secrète. Nos résultats sont validés expérimentalement avec l'implémentation de l'AES dans la bibliotèque MbedTLS.

Organisation du Manuscrit

Ce manuscrit est structuré en sept chapitres, dont quatre correspondent aux contributions présentées ci-dessus. Les chapitres sont organisés comme suit:

- Le premier chapitre (ce chapitre, Chapter 1) introduit la motivation et les contributions de cette thèse.
- Le deuxième chapitre (Chapter 2) présente les connaissances nécessaires sur l'AES et Ascon-AEAD, ainsi que sur les attaques SCA et FA, et définit les notations utilisées dans ce manuscrit.
- Le troisième chapitre (Chapter 3) présente notre contribution sur l'analyse de la fonction de sélection et l'attaque CPA d'ordre deux contre une implémentation protégée de Ascon-AEAD. Cela correspond à la publication [NGC25c].
- Le quatrième chapitre (Chapter 4) présente notre contribution sur l'extension de la fonction de sélection dans l'attaque CPA contre Ascon-AEAD, passant d'un bit à plusieurs bits. Cela correspond partiellement à la publication [NGC25a].
- Le cinquième chapitre (Chapter 5) présente notre contribution sur l'analyse de SIFA lorsqu'elle est appliquée aux schémas de chiffrement authentifié. Cela correspond à la publication [NGC25d].
- Le sixième chapitre (Chapter 6) présente notre contribution sur l'attaque PFA contre AES. Cela correspond à la publication [NGC25b].

- Le dernier chapitre (Chapter 7) conclut cette thèse et présente des perspectives pour les travaux futurs.

1

Introduction

Contents

We begin this thesis by presenting its scope and motivation. We then introduce the main contributions achieved during the doctoral studies, followed by an outline of the thesis structure.

1.1. Scope and Motivation

For a very long time, protecting communication between a sender and a receiver from an eavesdropping third party has been a critical problem, from the historical example of Caesar's cipher to today's secure communication over the Internet. The study and practice of such protection is called *cryptography*. The eavesdropping party is referred to as an *adversary* or *attacker*, who may spy on the communication, modify exchanged messages, or impersonate a legitimate sender. In this context, cryptographic solutions aim to guarantee the following properties:

- Confidentiality: ensuring that messages are unintelligible to unintended receivers.
- Integrity: ensuring that messages are not be modified or altered during transmission.
- Authenticity: ensuring that messages originate from the intended sender.

In cryptography, *symmetric cryptography* is an important branch in which the sender and receiver share a common secret *key*. The sender uses this key to *encrypt* a message, called the *plaintext*, into a *ciphertext* and sends the ciphertext to the receiver. The receiver then uses the same key to *decrypt* the ciphertext and recover the original plaintext. The secrecy of the key determines the security of communication systems.

Nowadays, in the era of Internet of Things (IoT), the use of small computing devices such as Radio-Frequency IDentification (RFID) tags, sensors, smart cards, smart keys, and electronic passports is increasingly common. It is estimated that there are about 30 embedded computing devices per person in developed countries [Cen14]. Figure 1.1 shows an example of smart cards in real world. To provide

trusted security, cryptographic algorithms standardized by the U.S. National Institute of Standards and Technology (NIST) are typically employed in these devices. In particular, two symmetric cryptography standards are likely to be widely deployed in resource-constrained IoT devices: the Advanced Encryption Standard (AES) [DR02] and the Ascon-based Authenticated Encryption with Associated Data (Ascon-AEAD) [DEM+21]. The former has been time-proven to be a robust cipher since its standardization in 2001. The latter, selected for standardization in 2023, is specifically designed to be lightweight for scenarios where AES is too resource-intensive [MBT+17]. In embedded devices, secret keys of these cryptographic algorithms are usually stored in memory. It is therefore crucial to protect them from exposure to potential adversaries.

Figure 1.1.: A smart card used in public transportation.

Having qualified through NIST's competitions, these two ciphers have been thoroughly analyzed. However, this analysis typically considers a *black-box model*, where the adversary has access only to the inputs and outputs. Such model is not always sufficient to guarantee security in practice. Especially when implemented and deployed in embedded devices, the adversary may have physical access to the devices. In this scenario, cryptographic algorithms can be vulnerable to physical attacks, which are generally classified into two categories: Side-Channel Attacks (SCA) and Fault Attacks (FA). These attacks have been recognized as significant threats since the seminal works on Differential Power Analysis by Kocher *et al.* [KJJ99] and Differential Fault Analysis by Biham and Shamir [BS97].

Because SCA and FA pose real risks in practice, understanding potential threats helps develop more robust implementations for large-scale deployment and helps prevent catastrophic security compromises. Therefore, studying physical attacks is of critical importance, and particularly relevant for the newly standardized cipher Ascon-AEAD. Motivated by this need, this thesis explores the threats posed by SCA and FA on the two symmetric cryptography standards: AES and Ascon-AEAD.

1.2. Main Contributions

In this thesis, we present two main parts of our contributions, one on Side-Channel Attacks (SCA) and the other on Fault Attacks (FA).

In Part I, we study a prominent type of SCA, namely Correlation Power Analy-

sis (CPA), targeting Ascon-AEAD. This study has resulted in the following two publications:

- **Second-Order CPA Attack with Proper Selection Function [NGC25c]**: In the first work of this part, we conduct a comprehensive analysis of different approaches from the literature for choosing the selection function used to compute intermediate values in CPA attacks against Ascon-AEAD. Through both theoretical explanation and experimental validation, we demonstrate how these choices influence the success of the attack. Leveraging insights from our analysis, we perform the first successful second-order CPA attack on a masked software implementation of Ascon-AEAD.
- **Multi-bit Selection Function for CPA Attack [NGC25a]**: In the second work, we extend the *1-bit* selection function from the literature to a *multi-bit* selection function. We provide experimental results that highlight the benefits of this approach, particularly in terms of success rates and number of traces required for the CPA attacks against Ascon-AEAD.

In Part II, we study two prominent types of FA, namely Persistent Fault Analysis (PFA) and Statistical Ineffective Fault Analysis (SIFA), targeting AES and Ascon-AEAD. This study has resulted in the following two publications:

- **Statistical Ineffective Fault Analysis on Ascon-AEAD [NGC25d]**: In the first work of this part, we present an in-depth analysis of the attack strategy used in the literature to apply SIFA on nonce-based authenticated encryption schemes. We show that this attack strategy can fail when applied to certain implementations due to two main reasons: the low probability of obtaining an ineffective fault required for the attack, and the uniformity of the chosen intermediate value. We validate our findings through simulations on an 8-bit implementation of Ascon-AEAD.
- **Persistent Fault Attacks on AES with Instruction Skip [NGC25b]**: In the second work, we show that *persistent faults* can be induced not only through tampering with S-box elements stored in memory, as seen in the literature, but also through a common fault method, namely instruction skip. We then introduce the first PFA that targets a constant other than the S-box elements. Specifically, we show that faulting a round constant is also sufficient to recover the secret key. Our findings are experimentally validated using the AES implementation in the MbedTLS library.

1.3. Thesis Outline

This thesis is structured into seven chapters, four of which correspond to the main contributions listed above. The chapters are organized as follows:

- Chapter 1 (this chapter) presents the motivation and contributions of this thesis.
- Chapter 2 provides background on AES and Ascon-AEAD, introduces SCA

and FA, and defines the notations used throughout the thesis.

- Chapter 3 presents the contribution on the analysis of selection functions and the second-order CPA attack on Ascon-AEAD, which corresponds to the publication [NGC25c].
- Chapter 4 presents the contribution on extending the selection function for CPA from one bit to multiple bits, which partially corresponds to the publication [NGC25a].
- Chapter 5 presents the contribution on the analysis of SIFA when applied to nonce-based authenticated encryption schemes, which corresponds to the publication [NGC25d].
- Chapter 6 presents the contribution on PFA on AES, which corresponds to the publication [NGC25b].
- Chapter 7 concludes this thesis and provides perspectives for future work.

2

Background

Contents

Before going to the main contributions, this chapter provides the necessary background and introduces the notations used throughout the thesis.

2.1. Symmetric Cryptography

In symmetric cryptography, a *cipher* consists of a pair of algorithms, called *encryption* and *decryption*. Both algorithms use the same secret key, which must be kept confidential to ensure security. The encryption algorithm transforms a plaintext into a ciphertext that appears statistically indistinguishable from random data, while the decryption algorithm reverses this process to recover the original plaintext from the ciphertext. Figure 2.1 illustrates the basic encryption and decryption processes.

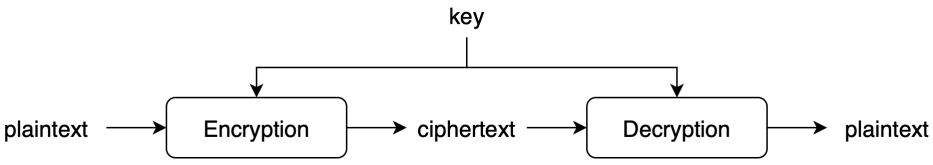


Figure 2.1.: Illustration of encryption and decryption.

Block ciphers, which operate on fixed-size blocks of data, are the classical primitives of symmetric cryptography. The most widely known block cipher is probably the Advanced Encryption Standard (AES) [DR02], which operates on 128-bit blocks. However, using directly a block cipher as a standalone encryption algorithm in practice leads to some problems. First, it can only process data in fixed-size blocks. Second, if two plaintext blocks are the same, their ciphertext blocks will also be the same, which can reveal information. Third, a block can be removed, reordered, or duplicated without being detected, which could break authenticity.

For these reasons, practical applications typically use a block cipher within a *mode of operation* to construct an authenticated encryption scheme. Such a scheme consists of two components, an encryption algorithm to provide confidentiality and a Message Authentication Code (MAC) to provide integrity and authenticity. In this context, the MAC is also referred to as an *authentication tag*. During decryption, this tag is recomputed and compared with the received tag to verify the authenticity and integrity of the message. If the verification fails, the decryption process returns an error and does not output a plaintext. This prevents the acceptance of potentially forged message. Figure 2.2 illustrates the concept of an authenticated encryption scheme.

Authenticated Encryption with Associated Data (AEAD) is an extension of this concept, in which additional cleartext data (*associated data* and *nonce*) is also included in the computation of the authentication tag. Among the various constructions, AES-GCM is one of the most widely used authenticated encryption schemes. It combines AES block cipher [DR02] with the Galois/Counter Mode (GCM) of operation [MV04].

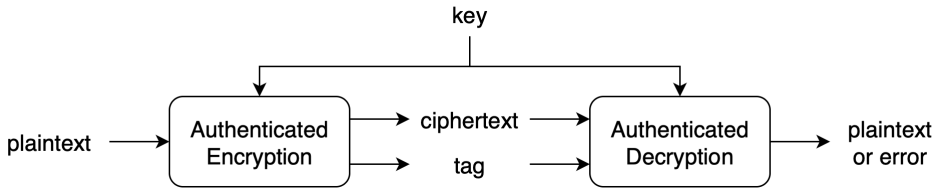


Figure 2.2.: Illustration of authenticated encryption.

Another way to construct an authenticated encryption scheme is by using a *permutation* together with the *sponge* construction with duplex mode [BDP+12]. In this approach, the internal state of the permutation is divided into two parts, called the *rate* and the *capacity*. The rate part is accessible to the adversary, while the capacity is inaccessible. The new lightweight cryptography standard Ascon-AEAD was designed based on this construction.

Since this thesis investigates physical attacks on the two symmetric cryptography standards, AES block cipher and Ascon-AEAD, we begin by introducing their designs. In this section, we provide detailed descriptions of both schemes.

2.1.1. Advanced Encryption Standard

The Advanced Encryption Standard (AES) [DR02] was selected by NIST as the winner in a competition in 2001 to replace the Data Encryption Standard (DES) [77].

It is a block cipher with a block size of 128 bits. A block (also called a *state*) is an array of 4×4 bytes (128-bit block) indexed from 0 to 15. The AES algorithm is a repetition of `Nr` rounds as depicted in Figure 2.3. A round is weak individually, but combining multiple rounds provides strong security [DR02]. There are three variants of AES with key sizes of 128, 192, or 256 bits, which correspond to a `Nr` $= 10$, `Nr` $= 12$, and `Nr` $= 14$ rounds, respectively. Each round is the composition of the following transformations:

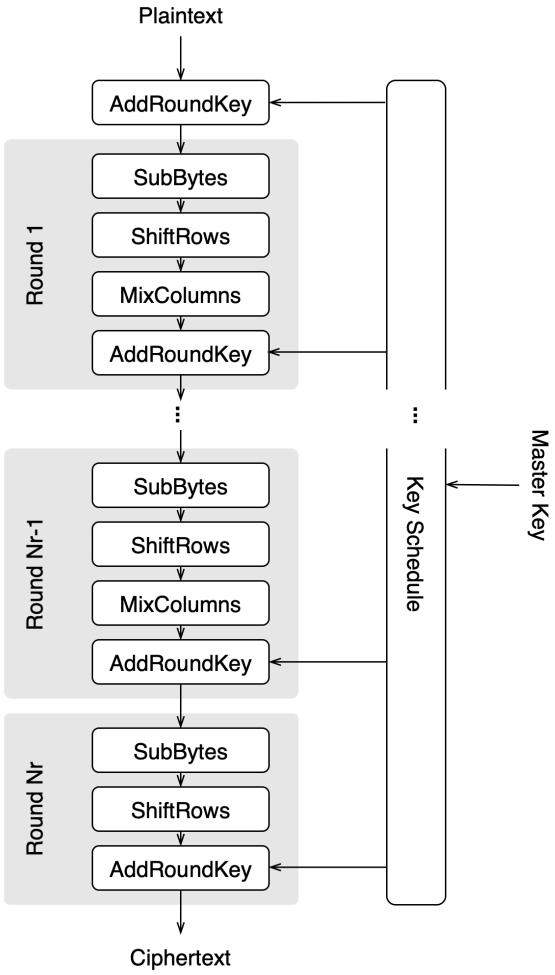


Figure 2.3.: Illustration of AES encryption.

- `SubBytes` is the substitution step, where each byte is replaced using an S-box to introduce non-linearity. This S-box is defined by a multiplicative inversion in the Galois field $\mathbb{F}_2^8$, followed by an affine transformation.
- `ShiftRows` is the transposition step, where rows of the state are cyclically shifted by a certain number of positions to introduce diffusion.
- `MixColumns` is the mixing step, where each column is transformed by a linear mapping over $\mathbb{F}_2^8$ to further diffuse the state. This mapping consists of multiplying the column vector by a Maximum Distance Separable (MDS) matrix.
- `AddRoundKey` is the key addition step, where a 16-byte round key derived from the original master key is XOR-ed with the state.

The process of deriving round keys from the original key is known as the *key schedule*. It generates a total of $4 \times (\texttt{Nr} + 1)$ columns, each of 4 bytes, from `Nk`-column master key ($\texttt{Nk} = 4, 6, 8$ for 128-, 192-, and 256-bit keys, respectively). The master key is first divided into `Nk` 4-byte columns. An iterative process is applied to derive the remaining columns. For the first column in each group of `Nk`, the process begins by taking the last column of the previous group and applying a series of transformations: `RotWord`, which cyclically shifts the bytes of the column, and `SubWord`, which substitutes each byte using the S-box. A round constant is then XOR-ed with the first byte of the column. For $\texttt{Nk} = 8$, an extra `SubWord` is also applied to the fourth column in each group. The result is then XOR-ed with the column located `Nk` positions earlier to produce the new column. For the remaining columns in the group, the process is simpler. Each column is generated by XOR-ing the previous column with the column from `Nk` positions earlier. This process continues until all round keys are derived. An illustration of the key schedule for AES-128 can be found in Figure 6.2.

Focus of physical attacks. In physical attacks, we typically choose an attack point in the cipher where its computation can be derived from accessible data and a small part of the key (which is to be guessed). This is known as the divide-and-conquer strategy. In AES, two locations are commonly targeted for such attacks: the beginning and the end of the cipher. This is because plaintexts and/or ciphertexts are usually accessible to attackers, and the first and last round keys can be partially guessed.

2.1.2. Ascon Authenticated Encryption

Ascon [DEM+21] is a family of algorithms that provide Authenticated Encryption with Associated Data (AEAD), hash function, and eXtendable Output Function (XOF). The design of these primitives are based on the duplex sponge construction [BDP+12]. In 2023, NIST selected Ascon as the winner of the lightweight cryptography competition [TMK+25]. It is designated to be a lightweight alternative to AES for resource-constrained environments such as IoT devices and embedded systems.

In this thesis, we focus on the authenticated encryption Ascon-AEAD. Figure 2.4 illustrates the encryption process of Ascon-AEAD. Its inputs include a 128-bit key K, a 128-bit nonce N, a constant initialization vector $\texttt{IV} = \texttt{00001000808c0001}$ (in hexadecimal), associated data $A_1, \ldots, A_s$, each of r bits, and plaintexts $P_1, \ldots, P_t$, each of r bits. It produces as output a tag T of 128 bits and ciphertexts $C_1, \ldots, C_t$, each of r bits. The tag T is used during the decryption to verify the authenticity of the ciphertexts and associated data.

The permutations p^a and p^b, consisting of a and b rounds, form the core of the Ascon-AEAD construction. The version standardized by NIST has $a = 12$, $b = 8$ and $r = 128$. These permutations operate on a 320-bit state, which is divided into five 64-bit words. Within a word, the rightmost bit (indexed as 0) is the least significant bit. A word can be stored in a single 64-bit register or several smaller-sized registers, depending on the device architectures. This design facilitates the transition from the

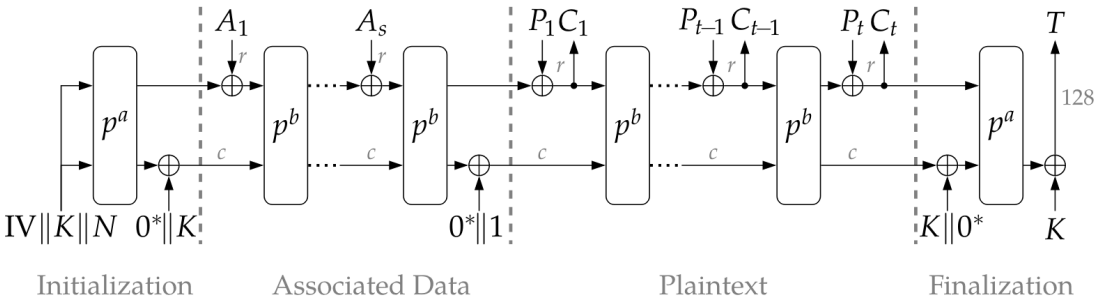


Figure 2.4.: Encryption in Ascon-AEAD [DEM+21].

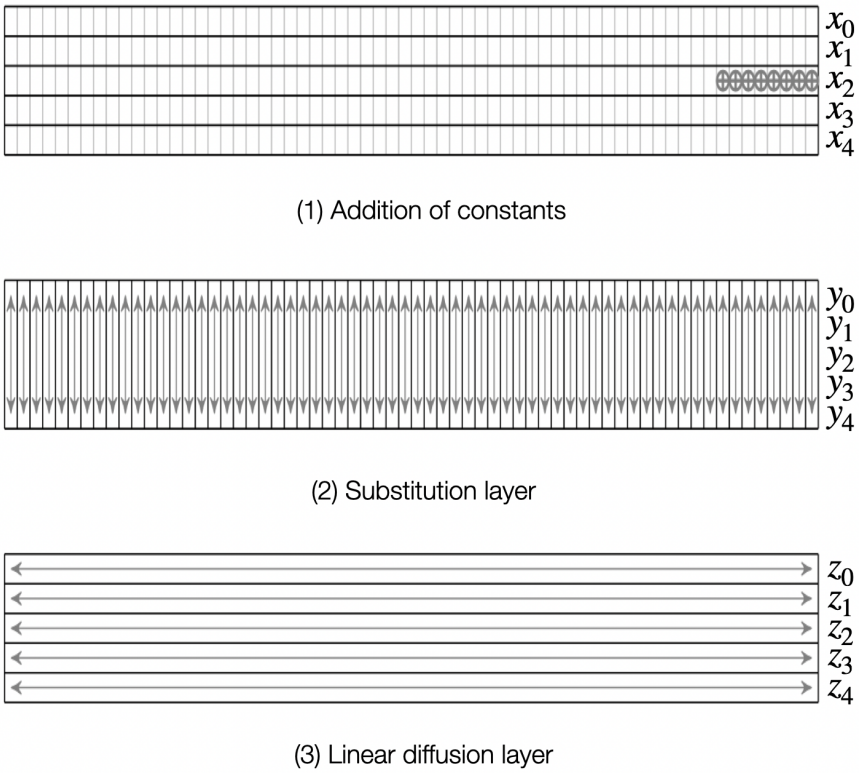


Figure 2.5.: Three steps of a round in Ascon-AEAD [DEM+21].

mathematical description to practical and efficient implementations.[1]

Each round in the permutations is composed of three steps as depicted in Figure 2.5:

- Constant-addition layer: an 8-bit round constant is XOR-ed into the rightmost bits of the third word of the state.
- Substitution layer: the state is updated column-wise with 64 parallel applications of a 5-bit S-box to introduce non-linearity.
- Linear diffusion layer: each word is rotated by fixed offsets and XOR-ed with the original word.

We now introduce notation to describe the computations in each step more concretely,

[1] Implementations for 8-bit, 32-bit, 64-bit architectures can be found at https://github.com/ascon/ascon-c

which will facilitate the analyses in the following chapters. Let x_0, x_1, x_2, x_3, x_4 represent the five 64-bit words of the round input as shown in Figure 2.5. In the first step, a round constant is added to the least significant byte of x_2. Since the constant addition step is not important in our analyses, we simplify the notation by continuing to denote the output of the first step as x_0, x_1, x_2, x_3, x_4.

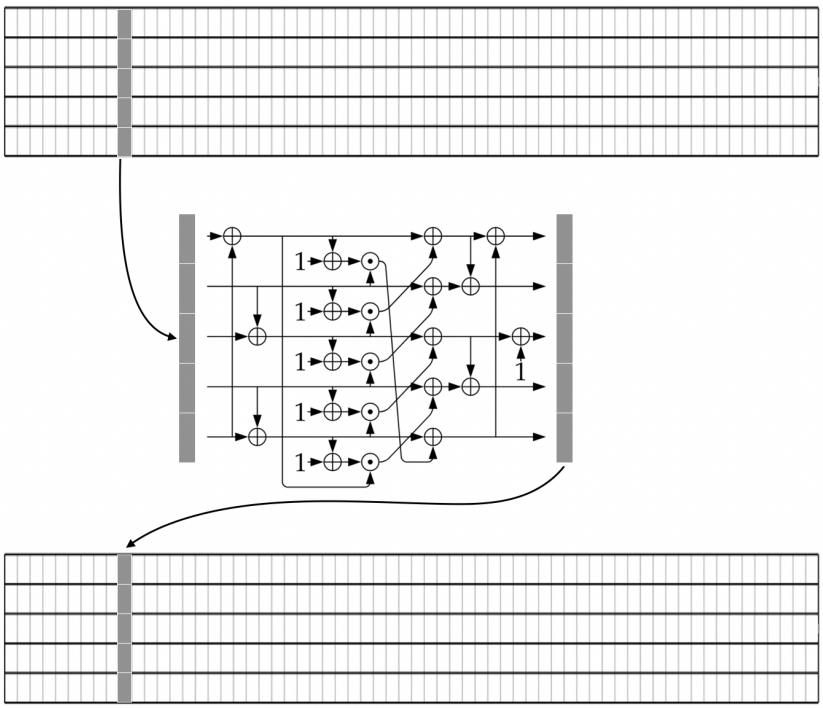


Figure 2.6.: An S-box computation in Ascon-AEAD.

The second step involves a non-linear transformation (S-box) applied to five bits, with one bit taken from each word of the first step's output. Figure 2.6 depicts an S-box computation. Let y_0, y_1, y_2, y_3, y_4 represent the output state of the S-box, and let $\mathbf{1}$ (in bold) denote a word filled with 64 bit ones. The algebraic normal form (ANF) of the S-box, with all operations performed on the full 64-bit words (in bitsliced form) can be expressed as:

$$\begin{aligned} y_0 &= x_4x_1 \oplus x_3 \oplus x_2x_1 \oplus x_2 \oplus x_1x_0 \oplus x_1 \oplus x_0, \\ y_1 &= x_4 \oplus x_3x_2 \oplus x_3x_1 \oplus x_3 \oplus x_2x_1 \oplus x_2 \oplus x_1 \oplus x_0, \\ y_2 &= x_4x_3 \oplus x_4 \oplus x_2 \oplus x_1 \oplus \mathbf{1}, \\ y_3 &= x_4x_0 \oplus x_4 \oplus x_3x_0 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0, \\ y_4 &= x_4x_1 \oplus x_4 \oplus x_3 \oplus x_1x_0 \oplus x_1. \end{aligned} \tag{2.1}$$

The third step, linear diffusion layer, applies a rotation to each word at the S-box output twice. The rotated words are then XOR-ed with the original one. Let z_0, z_1, z_2, z_3, z_4 denote the output of the linear diffusion layer. The linear functions

applied to each word are defined as:

$$\begin{aligned} z_0 &= y_0 \oplus (y_0 \ggg 19) \oplus (y_0 \ggg 28), \\ z_1 &= y_1 \oplus (y_1 \ggg 61) \oplus (y_1 \ggg 39), \\ z_2 &= y_2 \oplus (y_2 \ggg 1) \oplus (y_2 \ggg 6), \\ z_3 &= y_3 \oplus (y_3 \ggg 10) \oplus (y_3 \ggg 17), \\ z_4 &= y_4 \oplus (y_4 \ggg 7) \oplus (y_4 \ggg 41). \end{aligned} \tag{2.2}$$

Focus of physical attacks. In physical attacks, an attack point is typically chosen so that it can be computed from accessible data and a small part of the key (which is to be guessed). In Figure 2.4, the key K appears in four locations: before and after the initialization, and before and after the finalization. Among these, only the first location allows full control over the state, since the public nonces are accessible and the key can be partially guessed. In contrast, in the other locations, either only part of the state is controllable (after the finalization) or the state is not controllable at all (the remaining two locations). For this reason, the first initialization round is an ideal target for physical attacks.

In this thesis, the analyses presented in the following chapters concentrate extensively on the first initialization round. At the beginning of the initialization (Figure 2.4), the 64-bit initialization vector $\texttt{IV}$ is stored in x_0, the two 64-bit halves of the key $(k_0, k_1) = K$ are stored in x_1 and x_2, and the two 64-bit halves of the nonce $(n_0, n_1) = N$ are stored in x_3 and x_4. The S-box computation during the first round of the initialization phase can thus be written as follows:

$$\begin{aligned} y_0 &= n_1k_0 \oplus n_0 \oplus k_1k_0 \oplus k_1 \oplus k_0\texttt{IV} \oplus k_0 \oplus \texttt{IV}, \\ y_1 &= n_1 \oplus n_0k_1 \oplus n_0k_0 \oplus n_0 \oplus k_1k_0 \oplus k_1 \oplus k_0 \oplus \texttt{IV}, \\ y_2 &= n_1n_0 \oplus n_1 \oplus k_1 \oplus k_0 \oplus \mathbf{1}, \\ y_3 &= n_1\texttt{IV} \oplus n_1 \oplus n_0\texttt{IV} \oplus n_0 \oplus k_1 \oplus k_0 \oplus \texttt{IV}, \\ y_4 &= n_1k_0 \oplus n_1 \oplus n_0 \oplus k_0\texttt{IV} \oplus k_0. \end{aligned} \tag{2.3}$$

Equation 2.3 will be discussed frequently in our analyses in the subsequent chapters.

2.2. Side-Channel Attacks

In 1996, Kocher [Koc96] published the first Side-Channel Attack (SCA) against a cryptographic implementation. In this attack, secret keys in classical implementations of RSA and other public-key cryptosystems could be recovered by exploiting differences in execution time. Later, in 1999, Kocher *et al.* [KJJ99] published another attack that exploited differences in power consumption during the execution of cryptographic implementations, namely Differential Power Analysis (DPA). Since then, power analysis attacks have become a major research area in cryptography. Subsequent works [CKN01; May00; BCO04] proposed using Pearson correlation to exploit power traces (an example trace is shown in Figure 2.7). This approach was shown to be very effective and became well known as Correlation Power Analysis (CPA). Over the years, CPA has remained a powerful attack, applicable to many modern cryptographic schemes [KT23; WGL+24] and ciphers [SD17; BDG16].

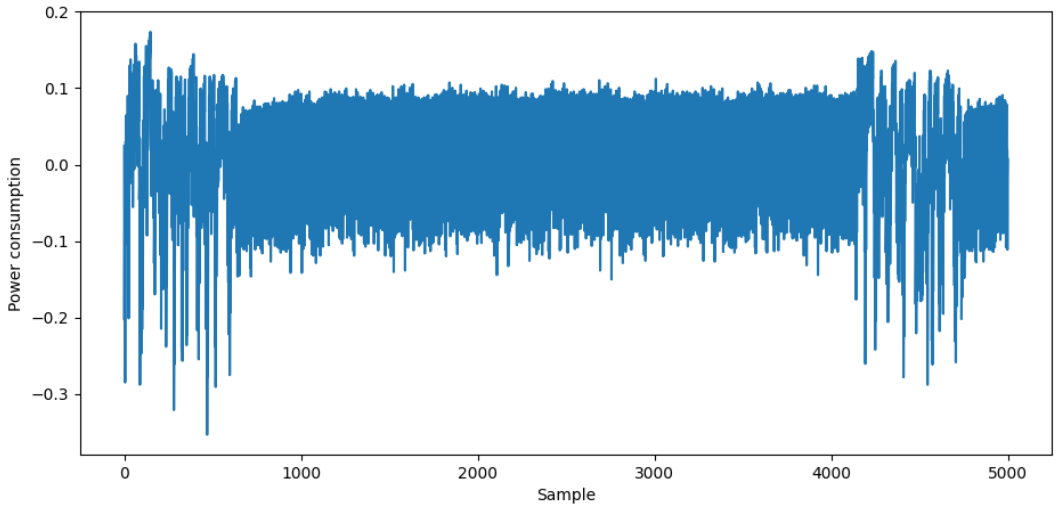


Figure 2.7.: A power trace acquired during an Ascon-AEAD execution.

Apart from CPA, many other attack techniques have been proposed. Some examples are Template Attacks [CRR03], Mutual Information Analysis [BGP+11], Soft Analytical Side-Channel Analysis [VGS14], Deep Learning Side-Channel Attacks [MPP16]. However, in Part I of this thesis, we focus on the practical aspects of CPA attacks on the new standard Ascon-AEAD. We therefore provide in this section the background on CPA and its common countermeasure, namely masking.

2.2.1. Correlation Power Analysis

Correlation Power Analysis (CPA) is a statistical analysis that exploits key-dependent power leakages to recover cryptographic keys. It is a *non-profiled* attack, requiring only power traces rather than detailed knowledge about the target device. A CPA attack considers only one point in the time dimension of the traces (*univariate*). The attack evaluates the correlation between power consumption at specific moments and the processed data. The procedure is depicted in Figure 2.8. It generally follows five steps:

Step 1: Choose an intermediate variable of the executed algorithm as the attack point. This intermediate variable needs to be a function $f(d, k)$, called *selection function*, of a part of the key k and the known non-constant data d (*e.g.*, plaintext, ciphertext, nonce).

Step 2: Measure the power consumption of the device while it executes the cryptographic algorithm ℓ times. For each execution, we record the data value d involved in the selection function and a power trace of s samples. Then, ℓ data values are written as a vector $\mathbf{d} = (d_1, \ldots, d_\ell)$, and ℓ power traces are written as a matrix $\mathbf{T}$ of size $\ell \times s$. It is important to note that the traces must be correctly aligned.

Step 3: Calculate hypothetical intermediate values for every possible candidate of k. Let $\mathbf{k} = (k_1, \ldots, k_p)$ be the vector of p possible candidates for k, also usually referred to as key hypotheses. For each key hypothesis, we use the selection func-

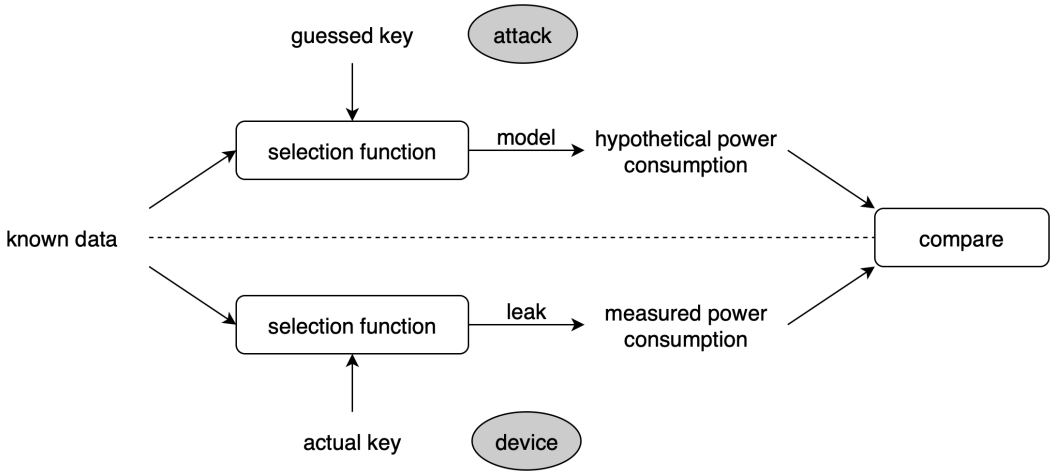


Figure 2.8.: Description of a CPA attack.

tion $f(d, k)$ to calculate the hypothetical intermediate values corresponding to the vector $\mathbf{d}$. Performing this calculation for all key hypotheses results in a matrix of hypothetical intermediate values, denoted by $\mathbf{V}$, of size $\ell \times p$.

Step 4: Map hypothetical intermediate values to hypothetical power consumption values. We choose a *leakage model* to estimate the power consumption exposed by the device when processing a value. In this thesis, we use the Hamming weight model. Each value in $\mathbf{V}$ is then mapped to a corresponding hypothetical power consumption value, resulting in a hypothetical power consumption matrix $\mathbf{H}$ of size $\ell \times p$.

Step 5: Compare the hypothetical power consumption values with the power traces. We use the Pearson's correlation coefficient to examine the linear correlation between the hypothetical power consumption values of each key candidate with the measured traces at every position. Specifically, we calculate the correlation coefficient between each column $\mathbf{h}_i$ of the matrix $\mathbf{H}$ and each column $\mathbf{t}_j$ of the matrix $\mathbf{T}$, resulting the element $r_{i,j}$ of the matrix $\mathbf{R}$ of size $p \times s$, where:

$$r_{i,j} = \frac{\sum_{u=1}^{\ell} \left(h_{u,i} - \overline{h}_i\right) \left(t_{u,j} - \overline{t}_j\right)}{\sqrt{\sum_{u=1}^{\ell} \left(h_{u,i} - \overline{h}_i\right)^2} \sqrt{\sum_{u=1}^{\ell} \left(t_{u,j} - \overline{t}_j\right)^2}}.$$

In the above equation, the values $h_{u,i}$ and $t_{u,j}$ (resp. $\overline{h}_i$ and $\overline{t}_j$) denote the u-th elements (resp. mean values) of the columns $\mathbf{h}_i$ and $\mathbf{t}_j$. The i-th row of $\mathbf{R}$ is called the *correlation trace* corresponding to the key candidate k_i.

The higher value $r_{i,j}$ indicates a stronger linear correlation between the columns $\mathbf{h}_i$ and $\mathbf{t}_j$, which suggests a better match under the assumed leakage model. In other words, the Hamming weight of the intermediate value is highly correlated to the power consumption and the Hamming weight model is appropriate.

Let $\mathtt{ck}$ be the index of the correct key $k_{\mathtt{ck}}$ (*i.e.*, the key that is used in the device) in the vector $\mathbf{k}$, and $\mathtt{ct}$ be the index of the power consumption values $\mathbf{t}_{\mathtt{ct}}$ that depend on the intermediate values $\mathbf{v}_{\mathtt{ck}}$. The columns $\mathbf{h}_{\mathtt{ck}}$ and $\mathbf{t}_{\mathtt{ct}}$ should be strongly correlated. Thus, the highest value $r_{\mathtt{ck},\mathtt{ct}}$ in the matrix $\mathbf{R}$ reveals the indexes of the

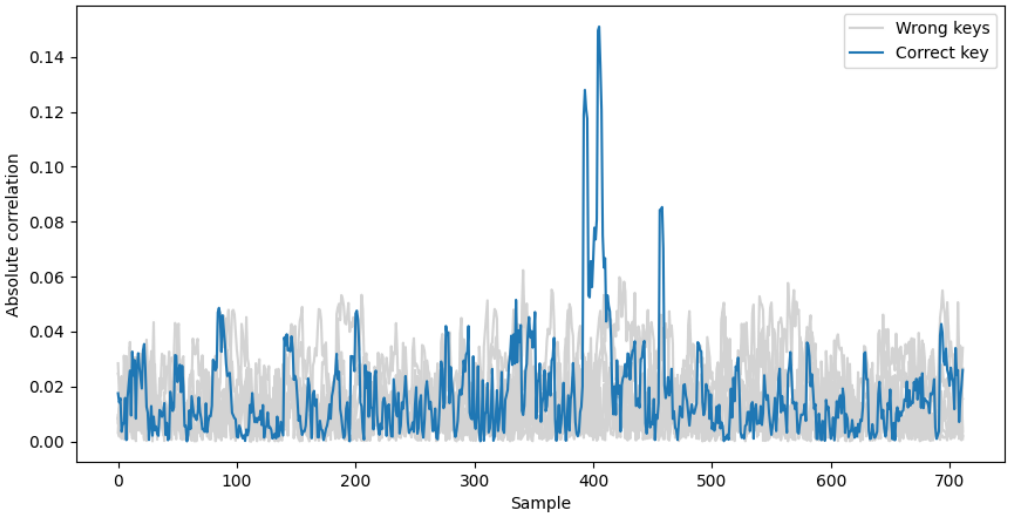


Figure 2.9.: Correlation traces in attack on an unprotected Ascon-AEAD.

correct key `ck` and the position `ct`. In this case, high peaks appear in the correlation trace corresponding to $k_{\texttt{ck}}$, while no such peaks are observed for other wrong key candidates. Figure 2.9 shows an example of correlation traces when performing CPA on Ascon-AEAD.

2.2.2. Masking

To protect against power analysis attacks, *masking* [CJR+99; GP99] is one of the most widely studied countermeasures. The goal of masking is to make the power consumption independent of the intermediate value. Its core idea is to split a *sensitive variable* x into $q+1$ *shares* $x_0, x_1, \ldots, x_q$ such that combining these $q+1$ shares yields x. Any subset of at most q shares is statistically independent of x. The parameter q is called the *masking order*. The computation is then carried out on these shares instead of on the sensitive variable.

For ciphers in which XOR is an elementary operation such as Ascon-AEAD and AES, *Boolean masking* is often used. In Boolean masking, q shares are drawn uniformly and the last one is computed such that the XOR combination of $q+1$ shares must satisfy:

$$x_0 \oplus x_1 \oplus \ldots \oplus x_q = x.$$

Note that masking does not provide complete protection but instead increases the attack complexity. In particular, the number of traces required to attack a masking scheme grows exponentially with the masking order [CJR+99]. A common method of attacking a masked scheme is to combine the power leakages of the individual shares and perform the analysis on the aggregated leakages. Such attacks are referred to as *higher-order* attacks [JPS05; Mes00; WW04]. The smallest statistical moment of the leakage distribution exploited by the attack is defined as the *attack order*. For an unprotected scheme, the leakage is usually contained in the mean, which corresponds to the first moment. A t-order attack is based on the t-th order moment. Theoretically, a q-order masking scheme is always vulnerable to a $(q+1)$-order attack.

This is because the joint leakages of $q + 1$ shares depend on the sensitive variable.

In practice, due to the trade-off between security and efficiency, cryptographic implementations are often masked with two shares ($q = 1$) or three shares ($q = 2$). Attacking those implementations is costly, but could still be practical. In Chapter 3, we will demonstrate a second-order CPA attack on a masked implementation of Ascon-AEAD with two shares.

2.3. Fault Attacks

In 1997, Boneh *et al.* [BDL97; BDL01] published the first idea of attacking cryptographic implementations by fault injection. In their demonstration, they showed how to attack certain implementations of RSA and other signature and authentication schemes. Soon after, Biham and Shamir [BS97] extended fault attacks to symmetric cryptography with a well-known technique called Differential Fault Analysis (DFA). Since then, fault attacks have emerged as a significant threat to cryptographic implementations, especially for embedded devices, where the adversary often has physical access.

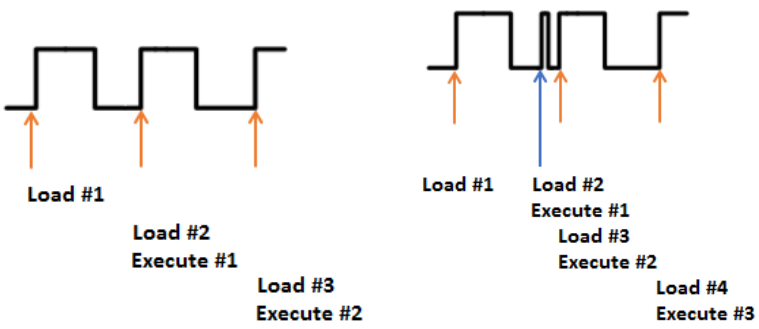
The goal of fault attacks is to take advantage of errors in the execution of cryptographic implementations, caused by intentional fault injection, to extract the secret key. A fault attack typically involves two main steps: *fault injection* and *fault analysis*. In the first step, faults are injected into the target device to disrupt the cryptographic execution. In the second step, faulty outputs are collected from the device and analyzed to recover the secret key.

In Part II of this thesis, we investigate the practical aspects of fault attacks on the new standard Ascon-AEAD and the long-standing standard AES. In this section, we provide background on the two steps of a fault attack and its common countermeasures.

2.3.1. Fault Injection

The most common fault injection techniques include clock glitches [CPH+21], voltage glitches [ABF+03], laser [VTM+17], electromagnetic pulses [MDH+13], X-rays [ABC+17], and rowhammer technique [MK19]. Following a fault injection, different effects may occur. For example, a laser fault can flip one or multiple bits in memory (*bit flip*) [ADM+10]. An electromagnetic fault can change a bit value to 1 (*bit set*) or 0 (*bit reset*) [MDH+13]. A voltage glitch can alter a byte to a random value (*random byte*) [TSS17]. In particular, an instruction can be skipped (*instruction skip*) with most techniques, from clock glitches [CPH+21] to electromagnetic pulses [DO22] to laser pulses [BJC15]. Instruction skip is also widely used to bypass security mechanisms in practice, such as PIN verification. For more details on common fault injection techniques and their practicality, we refer interested readers to the works of Bar-El *et al.* [BCN+06] and Breier and Hou [BH22].

Unlike SCA, where the leakage can be modeled, the effects of faults are much more difficult to model. In this thesis, our fault attacks use clock glitches to induce in-



a) Normal clock. b) Glitched clock.

Figure 2.10.: Example of instruction skip by a clock glitch in a pipelined system.

struction skips. Therefore, we provide a more detailed explanation of how a clock glitch can cause such behavior. Figure 2.10 illustrates how an injected clock glitch can force an instruction to be skipped in a pipelined system.[2] The glitch inserts a fast clock cycle between two ordinary clock cycles. As a consequence, Execution #1 is skipped since the system does not have enough time to perform the instruction.

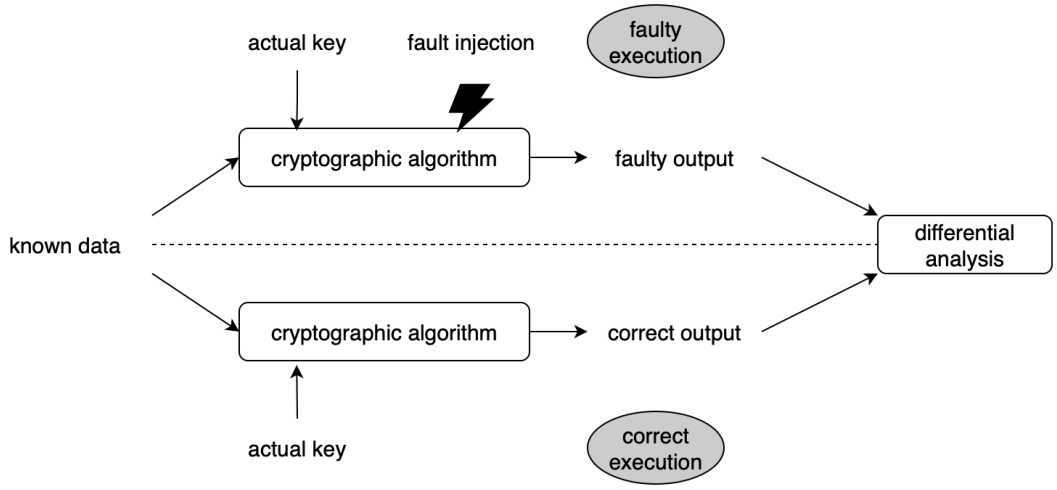
2.3.2. Fault Analysis

Since the seminal work of Biham and Shamir [BS97], Differential Fault Analysis (DFA) has become a prominent attack. Many subsequent works extended DFA to other ciphers, for example, AES [PQ03; DLV03; Gir05], DES [Riv09], Triple-DES [Hem04], Grain [BMS12], Trivium [HR08]. The principle of DFA is to recover the secret key by exploiting differences between outputs obtained from correct and faulty executions. Each pair of correct and faulty outputs must be collected from two executions with the same input: one with the fault injected and one without. The key recovery relies on carefully analyzing how the fault propagates through the rounds from the point of injection, and how the key can be inferred. This procedure is depicted in Figure 2.11a.

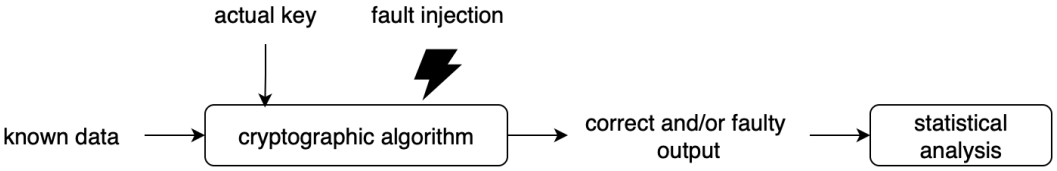
Apart from DFA, many other analysis methods have been published over time. Notably, methods based on statistical analysis are among the most recent and have attracted significant attention from the research community. These include Statistical Fault Analysis (SFA) [FJL+13a], Statistical Ineffective Fault Analysis (SIFA) [DEK+18], and Persistent Fault Analysis (PFA) [ZLZ+18]. The procedure of these analyses is depicted in Figure 2.11b. Their common principle is to collect a large number of outputs from executions with injected faults and apply a statistical or probabilistic analysis to infer the secret key. They are particularly effective against masked or countermeasure-protected implementations, where DFA may fail.

We briefly recall above the principles of DFA, SIFA, and PFA, as they are directly related to our contributions presented in Part II. In the literature, there are many other fault analysis methods that we do not cover here. Some examples are Differential Fault Intensity Analysis (DFIA) [GYT+14], Fault Sensitivity Analysis (FSA) [LSG+10], Fault Template Analysis (FTA) [SBR+20].

[2]Figure taken from https://github.com/newaetech/chipwhisperer-jupyter



a) A fault attack based on differential analysis.



b) A fault attack based on statistical analysis.

Figure 2.11.: Description of fault attacks.

2.3.3. Countermeasures

Along with the advancement of fault attacks, countermeasures are also increasingly needed for protection. In the literature, the main techniques used to protect cryptographic implementations against fault attacks include redundancy, sensors, and algorithmic protections.

Redundancy. In hardware, the most basic redundancy countermeasure is to duplicate the circuit, feed both circuits with the same input, and compare their outputs to ensure that they are identical. For software, this can be done by running the execution twice, either sequentially or in parallel on multiple processors, and checking for consistency. A variant of this redundancy is to verify an encryption by decrypting the resulting ciphertext and confirming that the obtained plaintext matches the original input. Moreover, a majority voting with a triplication can also be an option for redundancy countermeasure [BKH+20]. Instruction skips can be detected by an intra-instruction redundancy [PYG+16]. Code-based redundancy is also be an use to detect and possibly correct faults [JWK04; GKK+20; SMG16].

Sensors. In modern chips, some sensors are usually integrated to detect abnormalities in voltage and frequency [ADG+21]. This detection is aimed to protect against voltage and clock glitches. There also exist sensors that detect electromagnetic faults [BBH17] and laser faults [HBB16].

Algorithmic protections. An emerging research direction to protect against fault attacks is to design ciphers with built-in countermeasures. Examples of such ciphers are CRAFT [BLM+19], DEFAULT [BBB+21], and FRIET [SBD+20]. In these examples, the countermeasures in the first two ciphers are based on coding theory, while the countermeasure in the last cipher relies on an S-box with linear structures.

2.4. Experimental Setup

In this thesis, we provide experimental results for almost all of our power attacks and fault attacks. In both cases, the experiments are conducted using the ChipWhisperer toolbox [OC14]. This toolbox provides the necessary the hardware (analog capture hardware and target device) and software (for data capture and analysis) required for embedded security research. Its hardware and software designs are fully open-source and maintained in a public repository.[3]

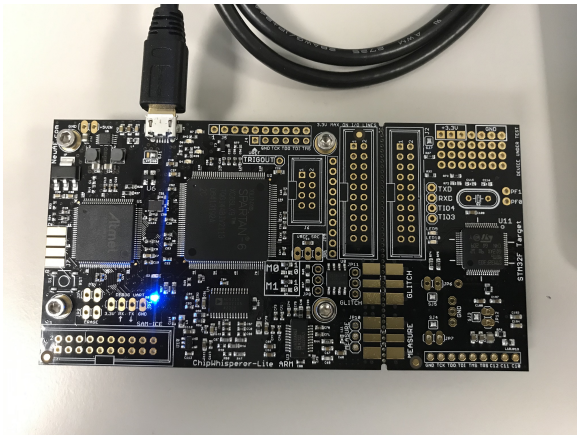


Figure 2.12.: ChipWhisperer Lite used in our experiments.

For our experiments, we use a ChipWhisperer-Lite board integrated with a 32-bit ARM target microcontroller (STM32F303), as shown in Figure 2.12. The device operates at its default clock frequency of 7.37 MHz. In power analysis attacks, the board is used to record power traces during the execution of cryptographic algorithms on the target. In fault injection attacks, the board is used to insert a fast clock cycle, creating a glitch that can cause the target to skip an instruction.

The ChipWhisperer board is connected to a MacBook Air M1 with 16 GB of RAM via USB. All subsequent analyses presented in this thesis are also performed on this computer.

[3] https://github.com/newaetech/chipwhisperer

Part I.

Correlation Power Attacks on Ascon-AEAD

3

Second-Order Attack with Proper Selection Function

Contents

In this chapter, we present the first contribution regarding our study on the Correlation Power Analysis (CPA) attacks on Ascon-AEAD. This includes an in-depth analysis of the selection functions and the first successful second-order attack against a masked implementaion.

3.1. Context and Motivation

After being selected by NIST, Ascon-AEAD [DEM+21] is expected to be widely deployed in embedded devices, where power leakages pose a significant threat. Hence, studying power analysis attacks on Ascon-AEAD is of critical importance. So far, there has not been much attention on this research area for Ascon-AEAD. Samwel and Daemen [SD17] introduced the first successful CPA attack on a noisy hardware implementation. In their work, the authors constructed an effective selection function for computing their chosen intermediate value. Using the same selection function, Roussel *et al.* [RPD+23] and Weissbart and Picek [WP23] also successfully performed CPA attacks on a hybrid CMOS/MRAM hardware implementation and an ARMv7m software implementation, respectively. Ramezanpour *et al.* [RAD+20] conducted DPA and CPA attacks with a different selection function, but reported that they failed to recover the key. The authors then proposed a deep learning-based power

analysis, which succeeded in key recovery. You *et al.* [YKS+23] introduced an efficient template attack on a 32-bit software implementation. Lou *et al.* [LWL+22] presented a Soft Analytical Side-Channel Analysis (SASCA) attack with simulated traces for an 8-bit implementation.

In the literature, several masked software implementations were published by the Ascon team.[1] Weissbart and Picek [WP23] attempted to perform a second-order CPA on an implementation with two shares, but reported a failure. An implicit reason that can be inferred from their work is that the number of traces was insufficient, however, this is not stated explicitly. Therefore, it is unclear whether this was the actual cause of failure or if the issue is in another step of the CPA process. The authors then proposed a successful deep learning-based power analysis, which was later improved by Rezaeezade *et al.* [RBW+23]. To our knowledge, no successful second-order CPA on Ascon-AEAD has been reported to date. This raises an open question regarding the practical cost of mounting such an attack. Addressing this question is important for assessing the security level of masked implementations and for determining the trade-off between security and efficiency in practice.

We also observe that most of the state-of-the-art power attacks on (protected) implementations of Ascon-AEAD, including template attack [YKS+23], SASCA attack [LWL+22], deep learning attacks [WP23; RBW+23], are *profiled attacks*. In these attacks, the powerful adversary is assumed to have full control on a copy of the targeted device and can obtain *a priori* knowledge about the implementation details. In contrast, the CPA attack, which is a *non-profiled attack* corresponding to a weaker adversary as presented in Section 2.2, has received less attention. Moreover, existing CPA attacks have different approaches of choosing the selection function, leading to either a success [SD17] or a failure [RAD+20] in key recovery. One approach relies on heuristics, such as using the S-box computation as the selection function [RAD+20], which has been well-studied in CPA attacks on AES. Another approach derives from observing of how processed data leaks into the power consumption [SD17]. In both cases [SD17; RAD+20], whether successful or not, there is a notable lack of analysis regarding the impact of these choices of the selection function on the success of the CPA attack.

In this chapter, we conduct a more in-depth study of the CPA attack against Ascon-AEAD. Specifically, we present the following results:

- First, we provide a comprehensive analysis of the selection functions used in the literature. Through both theoretical explanation and experimental validation, we demonstrate that different choices of the selection function can determine the success or failure of CPA attacks on Ascon-AEAD.
- Second, we determine the optimal number of CPA runs to reduce the effort for full key recovery. We show how the problem can be formalized as a *set cover problem* and solved using a SAT solver.
- Third, leveraging insights from our analysis, we present the first successful second-order CPA attack against a masked software implementation. We detail how the attack can be performed with modest resource requirements and

[1] See https://github.com/ascon/simpleserial-ascon

validate it on the 32-bit ARMv6 masked implementation of the Ascon team.

For the sake of reproducibility, we publish the source code of the experiments at:

https://github.com/nvietsang/socpa-ascon.

The results presented in this chapter was published and presented in the international conference *Constructive Approaches for Security Analysis and Design of Embedded Systems (CASCADE 2025)* [NGC25c].

3.2. Choices of Selection Function

The first and important step of CPA is to select an intermediate value as the attack point. This intermediate value must be the output of a selection function that takes as input a small portion of the key and known non-constant data. As evidenced in certain prior works [SD17; WP23; RAD+20; RPD+23], an intermediate value in the first round of the initialization phase seems well-suited for this purpose. This is due to the fact that the first round's inputs are the key and the nonce (see Figure 2.4), where the nonce can be regarded as the known non-constant data.

In the literature, to our knowledge, there exist two approaches of choosing the intermediate value and the selection function for CPA attacks on Ascon-AEAD. The first approach is to straightforwardly choose the S-box output as the intermediate value and the S-box computation as the selection function. It was used by Ramezanpour *et al.* [RAD+20] and is similar to the choice of S-box output as the attack point in many well-studied CPA attacks on AES. Using this approach, Ramezanpour *et al.* reported a failure for their attack (before introducing a successful attack based on deep learning), but did not provide any explanation. The second approach is to choose the linear diffusion layer output as the intermediate value and *fine-tune* the S-box computation for the selection function. It was proposed by Daemen and Samwel [SD17] and later used in [WP23; RPD+23]. Using this approach, the attacks in [SD17; WP23; RPD+23] successfully recovered the key. Daemen and Samwel provided the rationale behind their adjustment in the S-box computation to derive the selection function, but did not analyze how it impacts the attack's success. In other words, the authors did not explain why it is necessary to fine-tune the S-box computation instead of using it directly as the selection function.

In this section, we take a closer look into the two approaches. For each of them, we begin with a brief description of the selection function, and then analyze its impact on the attack's success. To simplify distinction, we refer the first approach as using the *pure* S-box computation, and the second approach as using the *fine-tuned* S-box computation, as the selection function. We also discuss whether we can go further into the second round to choose an intermediate value.

3.2.1. Pure S-box Computation as Selection Function

In Equation 2.3, the S-box computation is written in a bitsliced form in which 64 parallel applications of the 5-bit S-box (corresponding to the entire 64-bit words) are performed at once. For analysis, we consider a single application of the S-box. Let

the superscript j denote the index of the j-th bit of a 64-bit word, where $0 \leq j \leq 63$. The computation of the five S-box output bits $y_0^j, y_1^j, y_2^j, y_3^j$, and y_4^j is written as:

$$\begin{aligned}
y_0^j &= n_1^j k_0^j \oplus n_0^j \oplus k_1^j k_0^j \oplus k_1^j \oplus k_0^j \mathtt{IV}^j \oplus k_0^j \oplus \mathtt{IV}^j, \\
y_1^j &= n_1^j \oplus n_0^j k_1^j \oplus n_0^j k_0^j \oplus n_0^j \oplus k_1^j k_0^j \oplus k_1^j \oplus k_0^j \oplus \mathtt{IV}^j, \\
y_2^j &= n_1^j n_0^j \oplus n_1^j \oplus k_1^j \oplus k_0^j \oplus 1, \\
y_3^j &= n_1^j \mathtt{IV}^j \oplus n_1^j \oplus n_0^j \mathtt{IV}^j \oplus n_0^j \oplus k_1^j \oplus k_0^j \oplus \mathtt{IV}^j, \\
y_4^j &= n_1^j k_0^j \oplus n_1^j \oplus n_0^j \oplus k_0^j \mathtt{IV}^j \oplus k_0^j.
\end{aligned} \tag{3.1}$$

It can be observed that $y_0^j, y_1^j, y_2^j, y_3^j$ take as input two bits of the nonce (n_0^j, n_1^j) that are known non-constant data, and two bits of the key (k_0^j, k_1^j) that need to be guessed in our attack. The fifth bit is from the constant initialization vector `IV`. A small difference in y_4^j is that only one bit of the key (k_0^j) is involved.

A single S-box output bit as the intermediate value. We now analyze the impact on the success of attacks if one chooses an output bit among $y_0^j, y_1^j, y_2^j, y_3^j, y_4^j$ as the intermediate value and the S-box computation as the selection function. Let us first consider the computation of y_0^j. There are 4 possible key candidates for (k_0^j, k_1^j), and 4 possible values of the known non-constant data (n_0^j, n_1^j). Table 3.1 presents the distribution of y_0^j corresponding to every possible key candidate. In the CPA attack on Ascon-AEAD, we input (random) nonces into the algorithm and measure the corresponding power traces. Suppose that the values of nonce bits are uniformly distributed. Then, the intermediate value y_0^j follows the distribution presented in Table 3.1.

We examine the linear correlation between these distributions by computing Pearson's correlation coefficient between the distribution vectors for each key pair. For example, the vectors (0, 0, 1, 1) and (1, 1, 0, 0) represent the distributions when $(k_0^j, k_1^j) = (0, 0)$ and $(k_0^j, k_1^j) = (0, 1)$, respectively. These two vectors are correlated with a correlation coefficient of -1. Table 3.1 presents two correlation coefficients corresponding to the distribution vectors in red and blue. As observed, two key pairs result in distributions that are *fully correlated* with a coefficient of -1 or 1.

Table 3.1 presents the correlation coefficients for only two key pairs. We then extend this analysis by performing similar calculations for all possible key pairs. Table 3.2 (top left) summarizes the correlation coefficients between their corresponding distributions for y_0^j. Suppose 1 among 4 possible key candidates is the correct key. This analysis implies that, if y_0^j is chosen as the intermediate value, there will always exist a wrong key candidate whose hypothetical leakages are as highly correlated with the power traces as those of the correct key. Consequently, the correct key and this wrong key cannot be distinguished.

As shown in Table 3.1, such pairs of correct and wrong keys, for example, are $(0, 0)$ and $(0, 1)$, $(1, 0)$ and $(1, 1)$. We see that only one-bit information can be recovered (the first bit in those pairs), whereas we would expect to obtain two bits. Figure 3.1 illustrates the experiment for this analysis. It can be observed that the correlation traces for the candidate pairs $(0, 0)$ and $(0, 1)$ (in red) as well as $(1, 0)$ and $(1, 1)$ (in blue) are identical. Table 3.1 also shows that the value of the `IV` bit has no impact

	(k_0^j, k_1^j)			
(n_0^j, n_1^j)	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	0	1	1	1
(0,1)	0	1	0	0
(1,0)	1	0	0	0
(1,1)	1	0	1	1
Correlation	-1		1	

	(k_0^j, k_1^j)			
(n_0^j, n_1^j)	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	1	0	1	1
(0,1)	1	0	0	0
(1,0)	0	1	0	0
(1,1)	0	1	1	1
Correlation	-1		1	

Table 3.1.: Distribution of y_0^j corresponding to every possible key candidate when $\texttt{IV}^j = 0$ (top) and $\texttt{IV}^j = 1$ (bottom). In each table, the correlation value in red (resp. blue) is computed using Pearson's correlation applied to the two vectors in red (resp. blue).

on the correlation score. This is expected, because it either negates all outputs of the selection function when $\texttt{IV}^j = 1$ (vectors in red), or has no effect on them (vectors in blue) when $\texttt{IV}^j = 0$.

Note that the keys in those pairs are also not distinguishable in a Differential Power Analysis (DPA) attack, where the values of y_0^j are used to divide the traces into two sets, one for $y_0^j = 0$ and the other for $y_0^j = 1$. For instance, the division into two sets will be identical for the two key candidates $(0, 0)$ and (0,1), or $(1, 0)$ and $(1, 1)$, as the resulting distributions of y_0^j for these candidates are identical, as shown in Table 3.1. As a consequence, the difference of means of the two sets will also be identical for the two candidates. In the DPA attack of Ramezanpour *et al.* [RAD+20], y_0^j is chosen as the intermediate value. The authors reported that their DPA attack failed to find the correct key with more than 40K traces, but did not provide the reason. According to our analysis, their choice of the intermediate value and the selection function could be the explanation for this failure.

We conduct a similar analysis for each of y_1^j, y_2^j, y_3^j and y_4^j, and present the correlations between the distributions of all possible key pairs for each case in Table 3.2. The results are similar to those of y_0^j, except for y_4^j. Specifically, for any given correct key in cases of y_1^j, y_2^j and y_3^j, there will always be an wrong key that produces the same or negated distribution, making the correct key indistinguishable from the wrong one in DPA and CPA attacks. It is even worse in the cases of y_2^j and y_3^j, represented by the table of full correlations with values of 1 in Table 3.2. In these cases, all other wrong key candidates yield identical or opposite distributions. This arises from the fact that

(k_0^j, k_1^j)	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	1	1	-	-
(0,1)	1	1	-	-
(1,0)	-	-	1	1
(1,1)	-	-	1	1

y_0^j

(k_0^j, k_1^j)	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	1	-	-	1
(0,1)	-	1	1	-
(1,0)	-	1	1	-
(1,1)	1	-	-	1

y_1^j

(k_0^j, k_1^j)	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	1	1	1	1
(0,1)	1	1	1	1
(1,0)	1	1	1	1
(1,1)	1	1	1	1

y_2^j and y_3^j

k_0^j	0	1
0	1	-
1	-	1

y_4^j

Table 3.2.: Absolute correlations of distributions associated to all possible key pairs for $y_0^j, y_1^j, y_2^j, y_3^j, y_4^j$ when $\texttt{IV}^j = 0$. The entries with value 0 are indicated by "-" to facilitate reading. An interpretation example: the value 1 at row (0,0) and column (0,1) in the top-left table indicates a correlation coefficient of 1, meaning that the distribution vectors corresponding to the key candidates $(k_0^j, k_1^j) = (0, 0)$ and $(k_0^j, k_1^j) = (0, 1)$ are fully correlated.

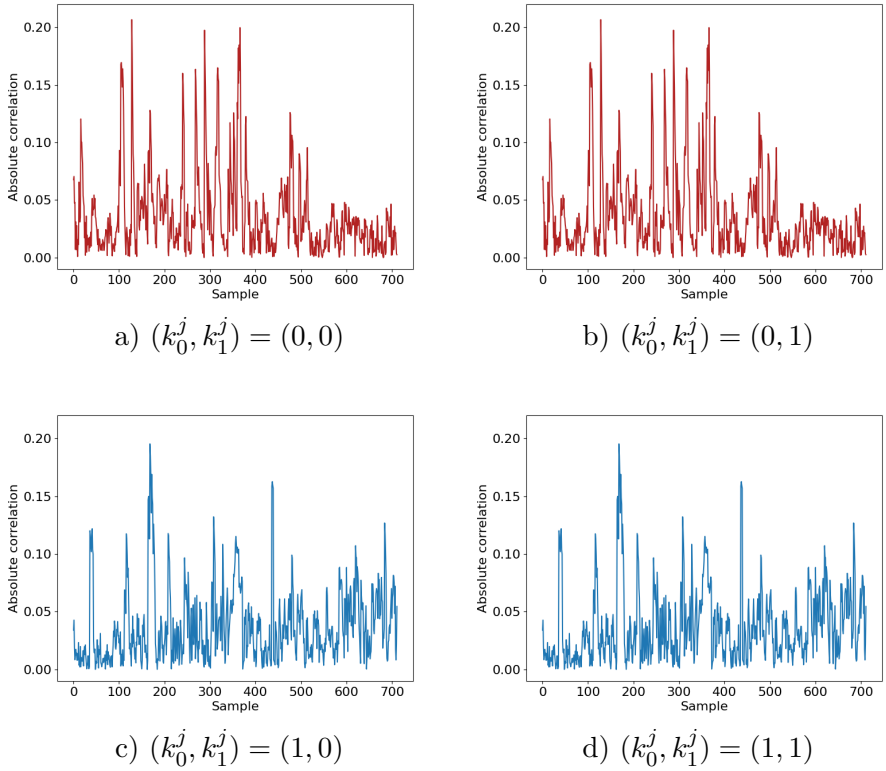
the key bits only have a linear influence on the computations of y_2^j and y_3^j. Thus, we conclude that CPA and DPA attacks using y_0^j, y_1^j, y_2^j or y_3^j as the intermediate value will not succeed in obtaining a unique correct key, regardless the number of traces.

We observe an exception in the case of y_4^j (Table 3.2, bottom right). The distributions of the two possible key candidates (since only k_0^j is involved in y_4^j) are uncorrelated. This implies that y_4^j, when used as a selection function, could yield a unique key candidate given enough traces. This also suggests that if we fix, for example, $k_1^j = 0$ in the y_0^j's table (*i.e.*, removing the first and third columns as well as the first and third rows in the top-left table), the table reduces to one similar to y_4^j. A similar fix applies to the y_1^j's table. Using y_0^j and y_1^j with these fixes might also lead to a unique key candidate.

Later, in Subsection 3.2.2, we will show that y_4^j or y_0^j and y_1^j with the fixes are similar to the fine-tuning in the second approach. These fixes will play an important role in the success of CPA attacks on Ascon-AEAD.

Hamming weight of S-box output as leakage model. We now consider the case where the Hamming weight of the 5-bit S-box output is used to model the leakages and the computations in Equation 3.1 as the selection function.[2] Employing the

[2]This is specific to the hardware implementation design where a register stores a 5-bit S-box output, with 1 bit from each of the 5 words. In other words, the register is designed to operate along the vertical dimension in Figure 2.5. This design, adopted by Ramezanpour *et al.* [RAD+20], differs from the intent of the reference implementation, where a register is meant to store an



a) $(k_0^j, k_1^j) = (0, 0)$ b) $(k_0^j, k_1^j) = (0, 1)$

c) $(k_0^j, k_1^j) = (1, 0)$ d) $(k_0^j, k_1^j) = (1, 1)$

Figure 3.1.: Correlation traces when using y_0^j as the intermediate value. The calculations use 1000 traces recorded from the execution of the reference implementation.

Hamming weight of the S-box output, as done by Ramezanpour *et al.* [RAD+20] in their attack on Ascon-AEAD, is a very common approach in CPA attacks on AES. Recall that there are still 4 possible key candidates for (k_0^j, k_1^j) and 4 possible nonce values for (n_0^j, n_1^j). As in the above analysis, we calculate the the Hamming weight distributions of the S-box output for every key candidate:

$$\mathrm{HW}(y_0^j | y_1^j | y_2^j | y_3^j | y_4^j),$$

where $\mathrm{HW}(\cdot)$ denotes the Hamming weight and $|$ denotes the concatenation. We then calculate the correlation between distributions generated by all possible key pairs, as shown in Table 3.3. We see that no key pairs with fully correlated distributions are observed. There are, however, still some very high correlations, for example, 0.90 between (1,0) and (1,1) in the left table, 0.93 between (0,1) and (0,0) in the right table. This suggests that the hypothetical power consumption values corresponding to some wrong key candidates (besides the correct one) may also be highly correlated to the power traces, making it difficult to distinguish the correct key. Especially in practical scenarios where the traces are heavily affected by noise, the CPA may fail or require a very large number of traces to find the correct key. This was also noted in the seminal work of Brier *et al.* [BCO04].

Our analysis shows that using the Hamming weight of the S-box output as the leakage model is not effective for CPA attacks against Ascon-AEAD. In [RAD+20],

entire word or part of a word, corresponding to the horizontal dimension in Figure 2.5.

(k_0^j, k_1^j)	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	1.00	0.15	0.89	0.87
(0,1)	0.15	1.00	0.48	0.09
(1,0)	0.89	0.48	1.00	0.90
(1,1)	0.87	0.09	0.90	1.00

(k_0^j, k_1^j)	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	1.00	0.93	0.52	0.17
(0,1)	0.93	1.00	0.48	0.27
(1,0)	0.52	0.48	1.00	0.09
(1,1)	0.17	0.21	0.09	1.00

Table 3.3.: Absolute correlations between the Hamming weight distributions of the S-box output for each key pair when $\mathtt{IV}^j = 0$ (left) and $\mathtt{IV}^j = 1$ (right).

Ramezanpour *et al.* adopted this approach for their CPA attack. The authors reported that their attack failed to recover the correct key even after using more than 40K traces, but they did not provide any justification. Since we do not have access to their implementation, we cannot determine the precise cause of the failure. However, we believe that the insights from our analysis here may help explain their results.

3.2.2. Fine-tuned S-box Computation as Selection Function

In the attack by Daemen and Samwel [SD17], the authors chose the output of the linear diffusion layer in their hardware implementation as the attack point, corresponding to the location of the registers. The activity of these registers at the end of each round (load/store) is assumed to leak information through power consumption. A notable contribution of their work is the adjustment applied to S-box computation before using it as the selection function. We now recall this adjustment and then analyze its impact on the success of CPA attacks.

As in [SD17], we only consider y_0^j, y_1^j and y_4^j in Equation 3.1 as their computations contain non-linear terms between the key and the nonce. For y_2^j and y_3^j, we observe that the key bits have only linear influence through XOR operations. As a consequence, y_2^j and y_3^j have the same distribution for all key candidates (not considering its sign). Hence, the correct key guess cannot be distinguished if one of these bits is chosen as the intermediate value.

Let us focus on y_0^j as an example. Its computation in Equation 3.1 is rewritten as follows:

$$y_0^j = k_0^j(n_1^j \oplus 1) \oplus n_0^j \oplus k_0^j k_1^j \oplus k_0^j \mathtt{IV}^j \oplus k_1^j \oplus \mathtt{IV}^j.$$

Following Bertoni *et al.* [BDD+12], the term $k_0^j k_1^j \oplus k_0^j \mathtt{IV}^j \oplus k_1^j \oplus \mathtt{IV}^j$ can be removed because, for a fixed correct key in the device, it is independent of the nonce and contributes a constant amount to the activity that leaks power consumption of the register containing y_0. Intuitively, this corresponds to vertical shift of the traces by a constant value. Note that this removal is equivalent to fixing $k_1^j = 0$ (and $\mathtt{IV}^j = 0$), as discussed before about the reduction for y_0^j in Table 3.2. The fine-tuned version of y_0^j, denoted by $\tilde{y}_0^j$, is:

$$\tilde{y}_0^j = k_0^j(n_1^j \oplus 1) \oplus n_0^j. \tag{3.2}$$

As the attack point is the activity of the registers at the linear diffusion layer output (not at the S-box output), we take the operation of this layer into account. Recall

from Equation 2.1 that the 64-bit output word z_0 of this layer is computed as:

$$z_0 = y_0 \oplus (y_0 \ggg 19) \oplus (y_0 \ggg 28).$$

The computation of the j-th bit of z_0 $(0 \leq j \leq 63)$ thus is:

$$z_0^j = y_0^j \oplus y_0^{j+19} \oplus y_0^{j+28}. \tag{3.3}$$

The additions $j + 19$ and $j + 28$ are implicitly taken modulo 64. The bit at index 0 is the word's rightmost bit. Applying Equation 3.2 to Equation 3.3 results in the fine-tuned version of z_0^j, denoted by $\tilde{z}_0^j$, which is used as the selection function to recover k_0 (three bits at a time):

$$\begin{aligned} \tilde{z}_0^j = &\left(k_0^j(n_1^j \oplus 1) \oplus n_0^j\right) \\ &\oplus \left(k_0^{j+19}(n_1^{j+19} \oplus 1) \oplus n_0^{j+19}\right) \\ &\oplus \left(k_0^{j+28}(n_1^{j+28} \oplus 1) \oplus n_0^{j+28}\right). \end{aligned} \tag{3.4}$$

Figure 3.2 illustrates the bits involved in the computation of the selection function of $\tilde{z}_0^j$, with an example for $j = 2$.

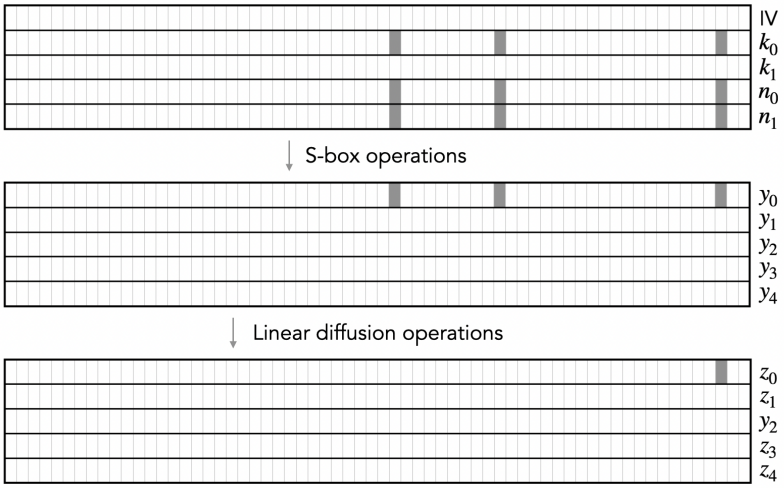


Figure 3.2.: Bits involved in selection function of $\tilde{z}_0^j$ from Equation 3.4 (illustrated for $j = 2$).

Similarly, we can derive the selection functions for recovering k_0 by fine-tuning y_4^j, and for recovering k_1 by fine-tuning y_1^j. The detailed derivation steps are provided in Appendix A. Here, we present the fine-tuned version of y_1^j and z_1^j, denoted by $\tilde{y}_1^j$ and $\tilde{z}_1^j$, which is used as the selection function to recover k_1 (three bits at a time):

$$\begin{aligned} \tilde{y}_1^j &= n_0^j(k_0^j \oplus k_1^j \oplus 1) \oplus n_1^j, \\ \tilde{z}_1^j &= \left(n_0^j(k_{01}^j \oplus 1) \oplus n_1^j\right) \\ &\quad \oplus \left(n_0^{j+61}(k_{01}^{j+61} \oplus 1) \oplus n_1^{j+61}\right) \\ &\quad \oplus \left(n_0^{j+39}(k_{01}^{j+39} \oplus 1) \oplus n_1^{j+39}\right), \end{aligned} \tag{3.5}$$

where $k_{01}^j = k_0^j \oplus k_1^j$. Note that k_1^j is not directly recovered, instead, k_{01}^j is recovered when $\tilde{z}_1^j$ is used as the selection. Then, k_1^j is derived as $k_1^j = k_{01}^j \oplus k_0^j$, with k_0^j recovered from the CPA using $\tilde{z}_0^j$ as the selection function.

Impact of fine-tuning. Let us analyze the selection function of $\tilde{z}_0^j$. A similar analysis applies to $\tilde{z}_1^j$ and we present here the results for both $\tilde{z}_0^j$ and $\tilde{z}_1^j$. We begin by examining the core of $\tilde{z}_0^j$, which is $\tilde{y}_0^j$ (Equation 3.2). As before, we calculate the distribution of $\tilde{y}_0^j$ for all possible candidates for k_0^j (2 candidates in total) in Table 3.4. It can be seen that the distributions produced by the two key candidates are uncorrelated with each other.

	k_0^j	
(n_0^j, n_1^j)	0	1
(0,0)	0	1
(0,1)	0	0
(1,0)	1	0
(1,1)	1	1
Correlation	0	

	k_{01}^j	
(n_0^j, n_1^j)	0	1
(0,0)	0	0
(0,1)	1	1
(1,0)	1	0
(1,1)	0	1
Correlation	0	

Table 3.4.: Distribution of $\tilde{y}_0^j$ (left) and $\tilde{y}_1^j$ (right) corresponding to every possible key candidate.

We then extend this calculation to the selection function of $\tilde{z}_0^j$. Table 3.5 presents the correlations between distributions of all possible key pairs. Note that 3 key bits and 6 nonce bits involve in $\tilde{z}_0^j$. We thus have 8 key candidates. As we can see, the distribution associated with an arbitrary key is uncorrelated with that of any other key. This makes the correlation between the hypothetical power consumption associated with the correct key and the power traces stand out those of the wrong keys. Figure 3.3 illustrates the experimental result for this analysis. It can be seen that the prominent peaks appear exclusively in the correlation trace of a single (correct) key candidate $(0, 0, 1)$. This explains the success of the attacks in [SD17], as opposed to the failure of the attack in [RAD+20], which relied on using the pure S-box computation as the selection function.

$(k_0^j, k_0^{j+19}, k_0^{j+28})$	(0,0,0)	(0,0,1)	(0,1,0)	(0,1,1)	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)
(0,0,0)	1	-	-	-	-	-	-	-
(0,0,1)	-	1	-	-	-	-	-	-
(0,1,0)	-	-	1	-	-	-	-	-
(0,1,1)	-	-	-	1	-	-	-	-
(1,0,0)	-	-	-	-	1	-	-	-
(1,0,1)	-	-	-	-	-	1	-	-
(1,1,0)	-	-	-	-	-	-	1	-
(1,1,1)	-	-	-	-	-	-	-	1

Table 3.5.: Absolute correlations of distributions associated to all possible key pairs using the selection function of $\tilde{z}_0^j$. The entries with value 0 are indicated by "-" to facilitate reading. The table for $\tilde{z}_1^j$ with the key tuple $(k_{01}^j, k_{01}^{j+61}, k_{01}^{j+39})$ is the same.

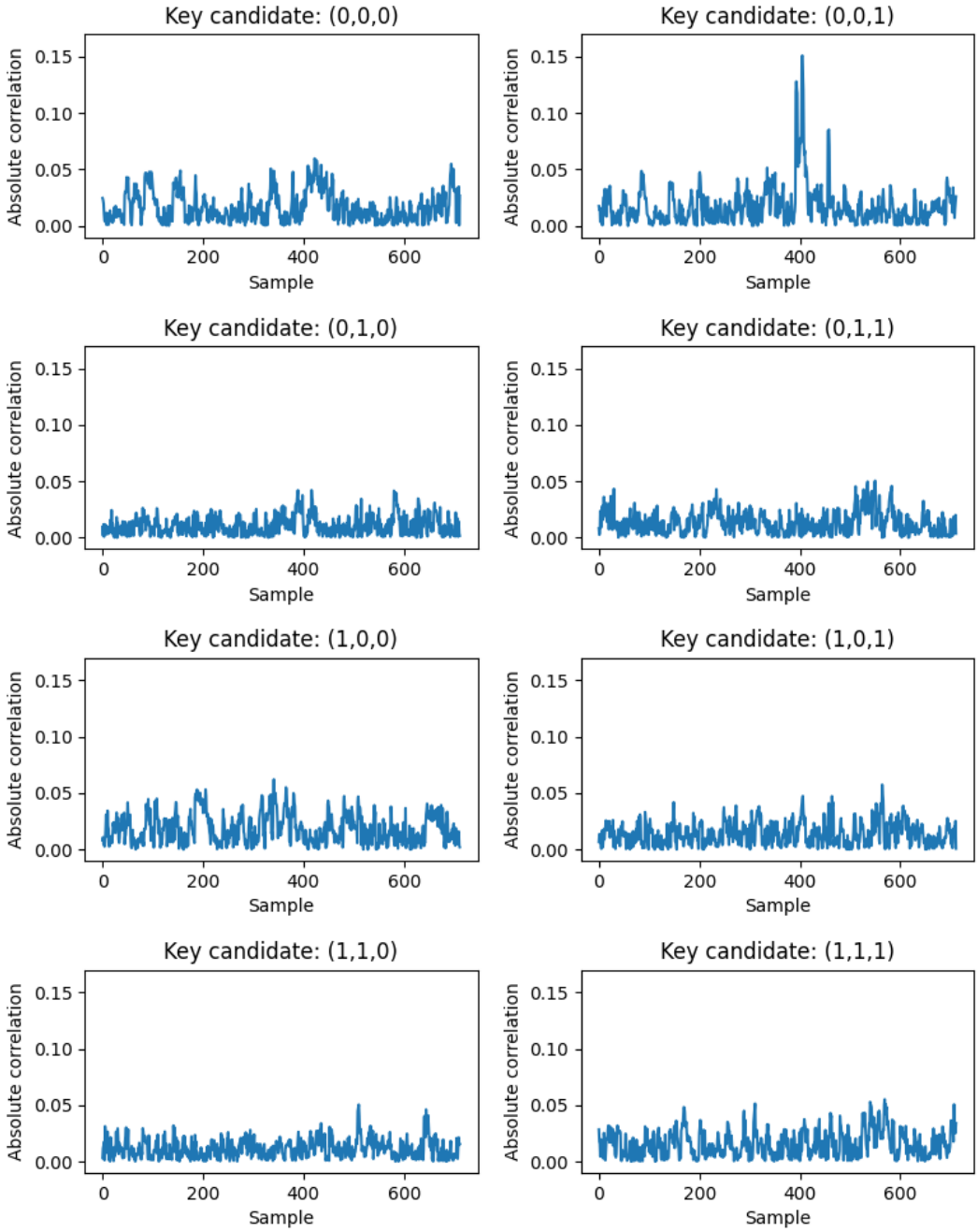


Figure 3.3.: Correlation traces for all key candidates when using $\tilde{z}_0^j$ as the intermediate value. Peaks appear in the correlation trace corresponding to the correct key candidate $(k_0^j, k_0^{j+19}, k_0^{j+28}) = (0, 0, 1)$. The calculations use 1000 traces recorded from the execution of the reference implementation.

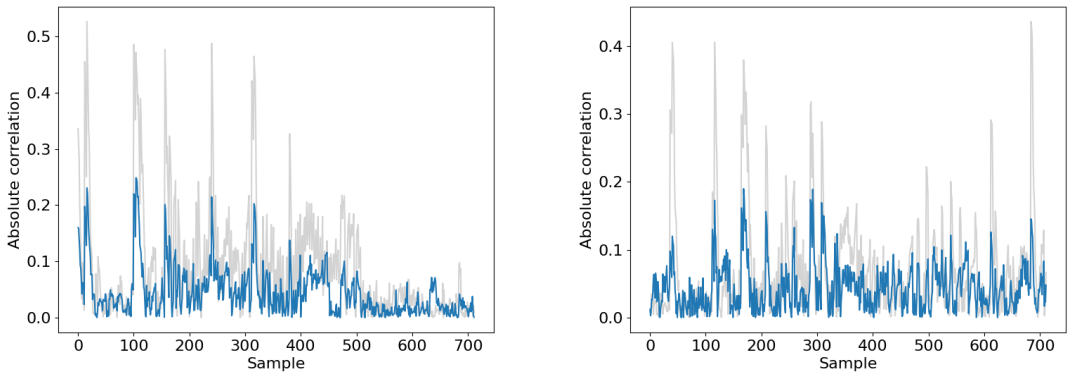
Impact of linear diffusion layer. Recall that Daemen and Samwel [SD17] choose the linear diffusion layer output as the attack point since it is where the registers locate in their hardware implementation. Note that in hardware, flip-flops used in registers usually consume more power than combinational logic [MOP07]. As a result, targeting the registers is generally easier. In software, since every computation is stored in registers, any intermediate value can be targeted.

Previously, we demonstrate that the pure S-box output $(y_0^j, y_1^j, y_2^j, y_3^j)$ produce distributions that are correlated to each other for some pairs of key candidates (Table 3.2). This leads to the fact that employing one of $y_0^j, y_1^j, y_2^j, y_3^j$ as the intermediate value results in multiple key candidates ranking equally with the correct key. We then show that the fine-tuned S-box functions $(\tilde{y}_0^j, \tilde{y}_1^j, \tilde{y}_4^j)$ yield distributions that are uncorrelated for all possible pairs of key candidates (Table 3.4). Now, we are interested in investigating whether $\tilde{y}_0^j, \tilde{y}_1^j, \tilde{y}_4^j$ are also good choices for the intermediate value (in addition to $\tilde{z}_0^j, \tilde{z}_1^j, \tilde{z}_4^j$) in software implementations. In other words, we aim to determine whether accounting for the linear operations really impacts the success of CPA attacks, or is just primarily relevant for attacks targeting hardware implementations with registers at the output of the linear diffusion layer (as in [SD17]).

Let us consider $\tilde{y}_0^j$ from Equation 3.2 with two possible key candidates, $k_0^j = 0$ and $k_0^j = 1$. Below are the results of $\tilde{y}_0^j$ for each key candidate:

$$\tilde{y}_0^j = k_0^j(n_1^j \oplus 1) \oplus n_0^j = \begin{cases} n_0^j & \text{if } k_0^j = 0, \\ n_0^j \oplus n_1^j \oplus 1 & \text{if } k_0^j = 1. \end{cases}$$

We will use the visualization of peaks in correlation traces for explanation. First, if $k_0^j = 0$ then $\tilde{y}_0^j = n_0^j$, meaning that the values of the intermediate value $\tilde{y}_0^j$ are identical to the values of the nonce bit n_0^j. As a result, the values of $\tilde{y}_0^j$ become correlated with the power consumption caused by the activity of the registers containing n_0^j (in addition to that of the registers containing y_0^j). Consequently, many peaks appear in the correlation trace for $k_0^j = 0$, as the blue peaks shown in Figure 3.4a. To support this explanation, we determine the locations where the activity of the registers containing n_0^j cause the power consumption, as illustrated by the light gray peaks in Figure 3.4a.[3]



a) $k_0^j = 0$ b) $k_0^j = 1$

Figure 3.4.: Correlation traces when using $\tilde{y}_0^j$ as the intermediate value. The calculations use 1000 traces recorded from the execution of the reference implementation.

Second, if $k_0^j = 1$ then $\tilde{y}_0^j = n_0^j \oplus n_1^j \oplus 1$, meaning that the values of the intermediate

[3]For each of the recorded nonces, we extract a byte value from the 64-bit word n_0 that contains the bit n_0^j. We then compute the correlation between the Hamming weights of those values and the power traces.

value $\tilde{y}_0^j$ are the inverse of $n_0^j \oplus n_1^j$ (correlation coefficient of -1). We examine the bitsliced implementation of the Ascon-AEAD S-box.[4] Listing 3.1 shows the generic approach to the S-box implementation. It can be observed that there does exist the operation $n_0^j \oplus n_1^j$ in line 2 of Listing 3.1. As a result, the values of $\tilde{y}_0^j$ become correlated with the power consumption caused by this operation, leading to the appearance of many peaks in the correlation trace for $k_0^j = 1$, as the blue peaks shown in Figure 3.4b. Similarly to before, we identify the locations where the activity of the operation $n_0^j \oplus n_1^j$ causes the power consumption, as illustrated by the light gray peaks in Figure 3.4b, to support our argument.[5]

```
1  x0 = x0 ^ x4
2  x4 = x4 ^ x3  // n0 ^ n1 in the first initialization round
3  x2 = x2 ^ x1
4  // Start of keccak S-box
5  u0 = x0 ^ (~x1 & x2)
6  u1 = x1 ^ (~x2 & x3)
7  u2 = x2 ^ (~x3 & x4)
8  u3 = x3 ^ (~x4 & x0)
9  u4 = x4 ^ (~x0 & x1)
10 // End of keccak S-box
11 y0 = u0 ^ u4
12 y1 = u1 ^ u0
13 y2 = ~u2
14 y3 = u3 ^ u2
15 y4 = u4
```

Listing 3.1: Bitsliced implementation in C syntax of Ascon-AEAD's S-box.

To sum up, high peaks appear in correlation traces for all key candidates, making the CPA prone to failure in recovering the correct key. The analyses on $\tilde{y}_4^j$ and $\tilde{y}_1^j$ yield analogous results. In contrast, peaks appear only for the correct key candidate when the linear diffusion layer is accounted for, as in Figure 3.3. This demonstrates that the linear diffusion layer in the selection functions of $\tilde{z}_0^j$ (Equation 3.4) and $\tilde{z}_1^j$ (Equation 3.5) plays an important role in the success of the CPA attacks, even in software implementations.

3.2.3. Intermediate Value in Second Round

We now turn to the question whether it is possible to go further to choose an intermediate value. In particular, we investigate whether a bit at the S-box output of the second initialization round can be an intermediate value. To this end, we examine the computation of this bit from the nonce and the key.

Let t_0, t_1, t_2, t_3 and t_4 denote the five 64-bit words at the output of the S-box in the second round. We consider the bit at index 0 of t_0, denoted by t_0^0. Since Ascon-AEAD's S-box has algebraic degree two, this computation has algebraic degree four, as described below:

[4] See https://github.com/ascon/ascon-c

[5] Similarly to before, but for $n_0 \oplus n_1$ instead of n_0.

$$t_0^0 = \underbrace{k_0^0 k_1^0 n_0^0 n_1^0 \oplus k_0^0 k_1^0 k_0^7 n_1^7 \oplus \ldots \oplus k_1^{10} \oplus k_0^{10}}_{589 \text{ monomials}}. \tag{3.6}$$

In Equation 3.6, 22 key bits and 22 nonce bits are involved in the computation. It consists of 589 monomials, 119 of which depend only on key bits. Following the strategy of Samwel and Daemen [SD17], we remove these 119 monomials, leaving 470 monomials that remain. However, we encounter another problem with the remaining monomials due to the high degree.

Let us take the first two monomials in Equation 3.6 as an example. If one of the key bits k_0^0 or k_0^1 is guessed to be 0, these two monomials vanish regardless the values of the other bits. In other words, $(k_0^0, k_0^1) = (0, 0)$, $(0, 1)$, or $(1, 0)$ all make these two monomials equal to 0. As a consequence, many key candidates will give the same correlation between the hypothetical power consumption and the measured traces. As the computation still contains 470 monomials, analyzing it by hand is difficult. Finding a way to process this computation into an effective selection function is an interesting problem. However, we leave this as an open question for now.

3.3. Optimal Number of CPA Runs

Each application of the CPA recovers three bits of k_0, indexed by $(j, j+19, j+28)$, or three bits of k_1, indexed by $(j, j+61, j+39)$, where $0 \leq j \leq 63$ (with additions implicitly modulo 64). To recover the full 128-bit key, the CPA must be applied multiple times to different tuples of key bit indexes. These tuples must collectively cover all indexes from 0 to 63 for both k_0 and k_1. Minimizing the number of such tuples is crucial to reduce the effort required for the attack. For example, performing the CPA three times to recover the key bits of k_0 at the tuples of indexes $(0, 19, 28)$, $(45, 0, 9)$ and $(36, 55, 0)$ (corresponding to $j = 0$, $j = 45$ and $j = 36$, respectively) results in the bit at index 0 being recovered three times, which is redundant.

To address this, we investigate the minimum number of index tuples. Weissbart and Picek [WP23] reported that 30 CPA runs are needed to recover k_0 and 33 runs to recover k_1, totaling 63 runs. In this work, we show that the full 128-bit key can be recovered with fewer CPA runs. Specifically, we find the optimal number of CPA runs by formalizing the problem as a *set cover problem* and solving it using a SAT solver.

Let o_0 and o_1 be two offsets, $(o_0, o_1) = (19, 28)$ for k_0 and $(o_0, o_1) = (61, 39)$ for k_1. The problem is stated as follows: given a *universe* of elements $\mathcal{U} = \{0, 1, \ldots, 63\}$ and a collection of 64 tuples $\mathcal{S} = \{(j, j+o_0, j+o_1) | j \in [0, 63]\}$ (with additions implicitly modulo 64), find the *smallest* sub-collection $\mathcal{S}' \subseteq \mathcal{S}$ whose union covers the universe,

$$\bigcup_{\texttt{tup} \in \mathcal{S}'} \texttt{tup} = \mathcal{U}.$$

Although the set cover problem is known as NP-hard, it is conceivable that a solution may be found for problems of smaller scale. To this end, we represent the problem as

a SAT problem. Each tuple $\texttt{tup}_j = (j, j+o_0, j+o_1) \in \mathcal{S}$ is represented by a Boolean variable b_j, where $b_j = 1$ (true) if $\texttt{tup}_j \in \mathcal{S}'$ and $b_j = 0$ (false) otherwise. To ensure full coverage of the universe, we impose that, for each $i \in [0, 63]$, at least one variable corresponding to the tuples that contains i should be true, *i.e.*,

$$\bigvee_{i \in \texttt{tup}_j} b_j = 1.$$

Additionally, we enforce a cardinality constraint to limit the number of selected tuples,

$$\sum_{j=0}^{63} b_j < \texttt{MAX},$$

where `MAX` is a parameter, and the sum is over integers. We transform this constraint into SAT clauses following [Sin05] and employ the SAT solver CryptoMiniSat[6] to find solutions for various values of `MAX`. The smallest `MAX` satisfying the SAT problem determines the minimum number of CPA runs. The Python script for this optimization is provided in Appendix A.

23 tuples for k_0 (j,j+19,j+28)		24 tuples for k_1 (j,j+61,j+39)	
(1,20,29)	(32,51,60)	(0,61,39)	(32,29, 7)
(3,22,31)	(35,54,63)	(2,63,41)	(37,34,12)
(5,24,33)	(38,57, 2)	(4, 1,43)	(38,35,13)
(8,27,36)	(42,61, 6)	(9, 6,48)	(39,36,14)
(11,30,39)	(45, 0, 9)	(11, 8,50)	(40,37,15)
(15,34,43)	(48, 3,12)	(13,10,52)	(45,42,20)
(18,37,46)	(49, 4,13)	(19,16,58)	(47,44,22)
(21,40,49)	(52, 7,16)	(20,17,59)	(49,46,24)
(22,41,50)	(55,10,19)	(21,18,60)	(51,48,26)
(25,44,53)	(59,14,23)	(23,20,62)	(56,63,31)
(28,47,56)	(62,17,26)	(28,25, 3)	(57,54,32)
(30,49,58)		(30,27, 5)	(58,55,33)

Table 3.6.: Tuples of indexes for each 3-bit key recovery. The number of tuples should be minimized while the range from 0 to 63 must be covered.

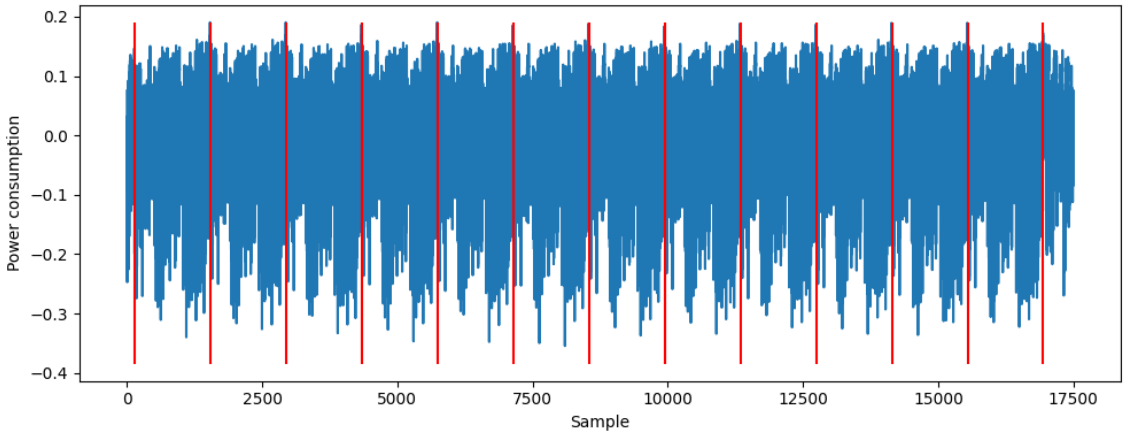
Our results are shown in Table 3.6. Only 23 CPA runs are needed for recovering k_0 and 24 for k_1, reducing the total to 47 runs. These results are obtained within a few seconds after executing the Python script. Note that imposing a smaller cardinality constraint (*i.e.*, setting `MAX` to a value below 23 or 24) leads to unsatisfiable problems within seconds. Therefore, the reported number of CPA runs is optimal.

3.4. Second-Order Attack

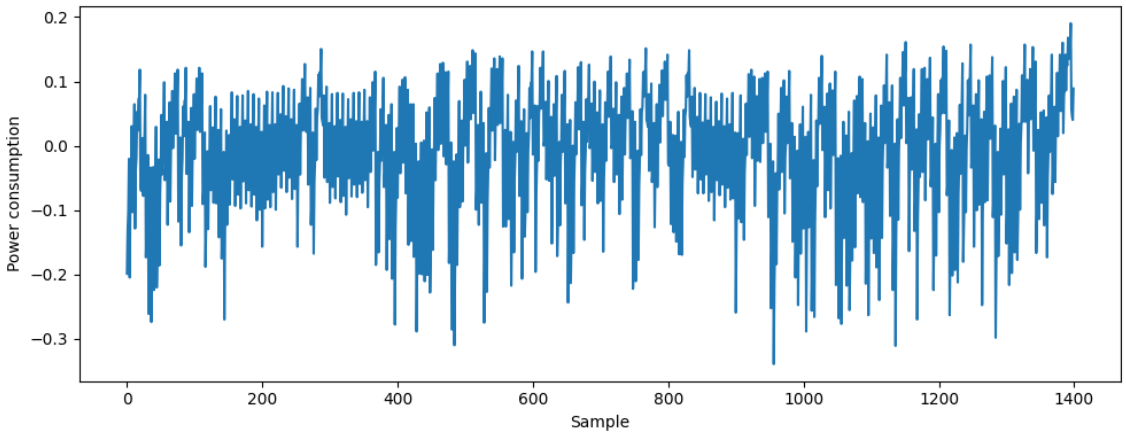
In the previous section, we thoroughly analyzed various approaches for choosing the intermediate value and the selection function, along with their impact on the success

[6] https://github.com/msoos/cryptominisat

of CPA attacks. Our analysis shows that $\tilde{z}_0^j$ (Equation 3.4) and $\tilde{z}_1^j$ (Equation 3.5) are effective choices for the intermediate values to recover the first and the second halves of the key. In this section, we apply these two selection functions to perform a second-order CPA attack on a masked software implementation with two shares. For our experiment, we use the 32-bit ARMv6 implementation[7] submitted to the call for protected software implementations of finalists in the NIST lightweight cryptography standardization process[8] by the Ascon team.



a) Power consumption of the first 12 rounds.



b) Power consumption of the first round.

Figure 3.5.: Power consumption of initialization rounds.

We begin by configuring an encryption execution with the minimum number of rounds, using empty associated data and an empty plaintext (*i.e.*, associated data and plaintext with length of 0) to skip the internal permutation blocks (see Figure 2.4). The execution thus consists of 24 rounds: 12 rounds for the initialization phase and 12 rounds for the finalization phase. During an execution of these 24 rounds, we record a power trace. Variance calculations are then applied to determine the length and starting index of each round, based on the assumption that the correctly aligned

[7] https://github.com/ascon/simpleserial-ascon, in `protected_bi32_armv6`.
[8] https://cryptography.gmu.edu/athena/LWC/Call_for_Protected_Software_Implementations.pdf

frames will minimize variance (*i.e.*, 24 frames should align well when overlapped). Figure 3.5a shows a power trace for the first 12 rounds of the initialization phase, and Figure 3.5b illustrates the frame corresponding to the first round, consisting of 1400 samples. We focus on the power consumption of this first round, as our attack utilizes the linear diffusion layer outputs $\tilde{z}_0^j$ and $\tilde{z}_1^j$ as intermediate values.

3.4.1. First-Order Leakage Assessment

In the repository of the targeted masked software implementation,[7] the authors reported that their implementation might leak information due to potential collisions between the two shares in hardware. However, they introduced device-specific fixes to prevent first-order leakages, specifically by inserting an `MOV` instruction with a value of 0 at appropriate locations to avoid these collisions.

Using a similar hardware platform (STM32F303), we expect these fixes to remain effective in our experiment. To verify the absence of first-order leakages in the implementation under attack, we employ the widely used Test Vector Leakage Assessment (TVLA) [GJJ+11]. This methodology applies a non-specific, fixed vs. random t-test statistic on two sets of traces, one with a fixed input and the other with random inputs. The t-score at the i-th sample, denoted as $\tau[i]$, is computed as:

$$\tau[i] = \frac{\mu_{\mathrm{f}}[i] - \mu_{\mathrm{r}}[i]}{\sqrt{\frac{\sigma_{\mathrm{f}}^2[i]}{n_{\mathrm{f}}} + \frac{\sigma_{\mathrm{r}}^2[i]}{n_{\mathrm{r}}}}}$$

where $\mu_{\mathrm{f}}, \sigma_{\mathrm{f}}$, and n_{f} (resp. $\mu_{\mathrm{r}}, \sigma_{\mathrm{r}}$, and n_{r}) represent the estimated mean, standard deviation, and the number of traces for the fix-input set (resp. random-input set). Leakage is detected if the absolute t-score exceeds the commonly used threshold of 4.5, in which case the null hypothesis that the means of the two sets are similar is rejected.

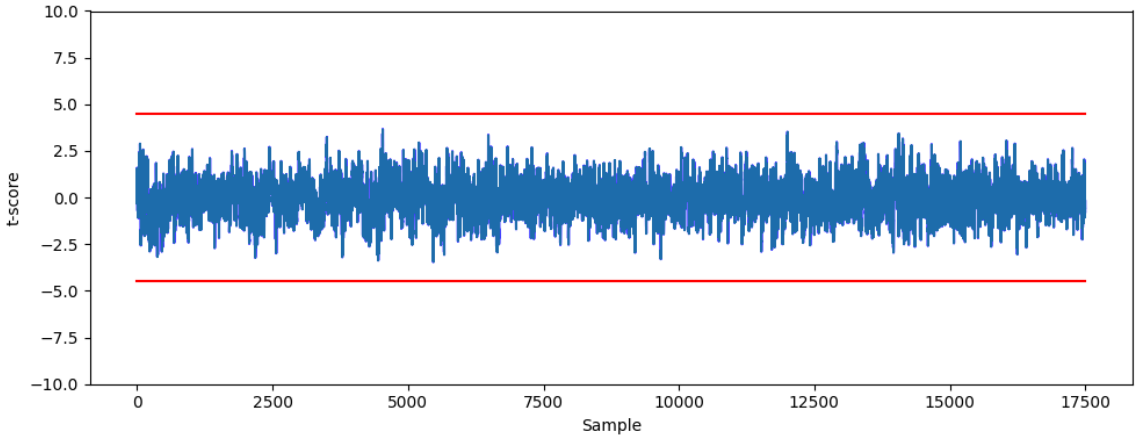


Figure 3.6.: Non-specific t-test on the first 12 rounds with 300,000 traces ($n_{\mathrm{f}} = n_{\mathrm{r}} = 150,000$).

Figure 3.6 shows the t-test results for all the time samples of the traces recorded during the execution of the first 12 rounds. As expected, no first-order leakages are observed, ensuring that we will not accidentally exploit them in our second-order CPA attack.

3.4.2. Pre-processing Power Traces

A second-order CPA attack consists of two phases: trace pre-processing and the standard CPA. In the first phase, samples within a trace are combined to produce a *pre-processed trace*. This combination can cause the length of each trace to increase quadratically, significantly raising the attack complexity. Therefore, we detail the pre-processing steps below to ensure that the attack remains time and memory efficient in practice.

Let $\mathbf{t} = [t_1, \ldots, t_s]$ represent a power trace containing s samples, where in our case, $s = 1400$. We can combine the samples within $\mathbf{t}$ using various methods, such as normalized product, absolute difference, or sum [PRB09; SVO+10]. Among these, the normalized product has been shown to be the most effective when applying Pearson's correlation coefficient with the Hamming weight leakage model [SVO+10]. In this work, we adopt the normalized product for trace pre-processing. According to this method, the sample $t'_{i,j}$ in the pre-processed trace $\mathbf{t}'$ derived from two power samples t_i and t_j $(1 \leq i, j \leq s)$ in $\mathbf{t}$, is calculated as:

$$t'_{i,j} = (t_i - \bar{t}_i)(t_j - \bar{t}_j),$$

where $\bar{t}_i$ (resp. $\bar{t}_j$) is the estimated mean computed over all the traces at the i-th (resp. j-th) sample. There are a total of $s(s+1)/2 = 980700$ possible pairs (i, j) from $s = 1400$ samples. This large number will significantly increase the time and memory cost of the CPA. However, we note that the computation of the shares of the first round output occurs within a limited time span. It is thus unnecessary to consider all possible pairs. We use a parameter called the *window size*, denoted w, which estimates the maximum distance between the leakages of the two shares in the trace $\mathbf{t}$. Thus, for the first $s - w$ samples ($i \in [1, s - w]$), there are w possible indexes j for each i. For the last w samples, there are a total of $w(w+1)/2$ possible pairs of (i, j). Consequently, the number of samples in the pre-processed trace $\mathbf{t}'$ (*i.e.*, the number of pairs (i, j)) becomes:

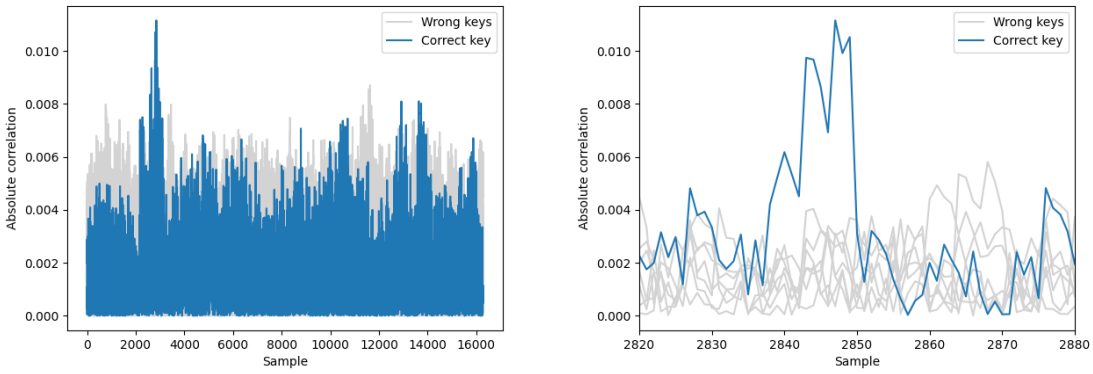
$$W = (s - w)w + \frac{w(w+1)}{2} = w\left(s - \frac{w-1}{2}\right).$$

To further reduce the number of samples s in $\mathbf{t}$, we make an educated guess that the S-box computation typically dominates the computation time in a round. Moreover, the less costly linear diffusion computation, where our attack point is located, occurs near the end of the round. Based on this insight, we focus on the last quarter of the samples in each trace. Specifically, we consider the last 350 samples (from 1050 to 1400 in Figure 3.5b). This reduces the value of s to 350. Additionally, we set the window size to $w = 50$. With these parameters, the number of samples in each pre-processed trace becomes $W = 16275$.

3.4.3. Results of Key Recovery

In the second phase of the attack, the standard CPA, as described in Section 2.2, is applied to the pre-processed traces. Since this is a conventional approach, we omit the details and instead present the results of the full key recovery.

To efficiently perform the CPA with a large number of traces, we implement an *incremental* second-order CPA as introduced by Bottinelli and Bos [BB17]. This approach enables to gradually compute Pearson's correlation coefficients with a smaller, tunable number of traces instead of processing the entire dataset at once. Additionally, the trace pre-processing is performed on-the-fly.



a) Full-length correlation traces b) Zoomed-in view of the peak

Figure 3.7.: Correlation traces for all key candidates. The calculations use 300K traces.

Figure 3.7 shows the results of the second-order CPA for recovering three bits of k_0 using $\tilde{z}_0^j$ as the selection function. Several peaks corresponding to the correct key are clearly visible at around sample 2848. Figure 3.8 illustrates how the correlation coefficients depend on the number of traces for recovering 3 key bits at indexes (1, 20, 29). It can be observed that around 100K traces are required for the second-order CPA to reliably distinguish the correct key from the others.

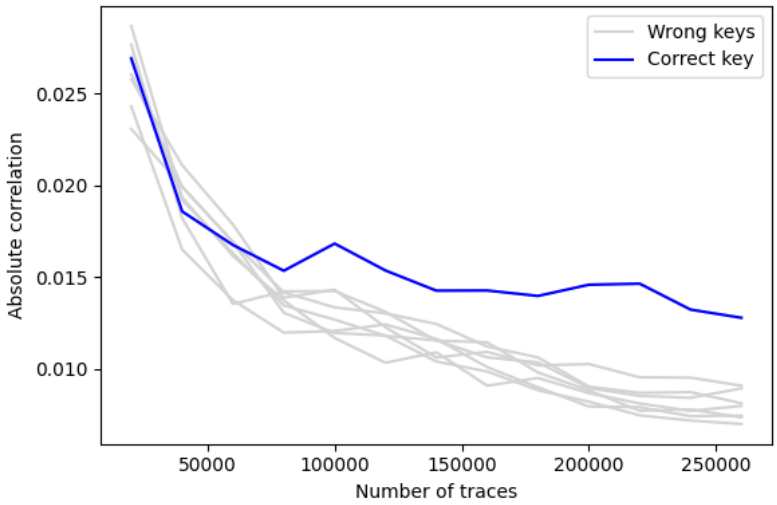


Figure 3.8.: Correlation for all key candidates depending on the number of traces.

We present the dependence between the success rate and the number of power traces for the full key recovery in Figure 3.9. Since directly measuring the success rate for

the full key recovery is time-consuming, we employ an estimation approach. Specifically, we measure the success rates of recovering three key bits of k_0 and k_1 by performing the CPA many times with different index tuples and different trace sets. These success rates obtained are then raised to the power of 23 and 24, respectively, reflecting the number of CPA runs needed to recover the full 128-bit key. Multiplying these two results provide an estimate of the success rate for full key recovery. As in Figure 3.9, about 360,000 traces are sufficient to achieve 100% success for the full key recovery. To validate this estimation, we perform an actual full key recovery experiment using 360,000 traces. As expected, we succeeded in recovering the full 128-bit key in about 4.7 hours.

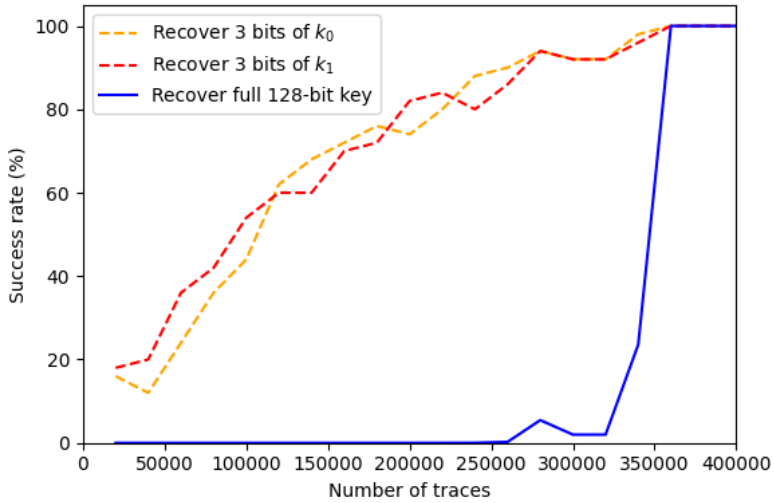


Figure 3.9.: Success rate of the full key recovery.

In our second-order CPA, the analysis phase is very efficient as we implement the incremental computation [BB17] to process a small number of traces at a time. For example, if we set this number to 20,000, the computation consumes 0.06 GB of RAM and takes about 30 seconds. To recover three bits of the key with 100% success rate, we need to process 360,000 traces, which thus takes 9 minutes, while the memory cost (0.06 GB) does not change thanks to the incremental computation. We can run several CPA on different tuples of key bits in parallel to accelerate the process. The most time-consuming phase of the attack is the collection of the power traces. With a collection speed of 448 traces per minute, acquiring 360,000 traces requires approximately 13.4 hours to ensure a 100% success rate.

3.5. Conclusion

In this chapter, we investigate various aspects of the CPA attack on (protected) software implementations of Ascon-AEAD. We show that choosing an appropriate selection function is a vital step toward a successful attack. We then determine the optimal number of CPA runs required to reduce the attack effort. Finally, we present the first results of a second-order CPA attack against a masked software implementation.

This chapter focuses exclusively on selection functions that work on one bit. In the next chapter, we extend them to multiple bits, allowing the hypothetical power consumption to be modeled by the Hamming weight of more than one bit.

4

Multi-bit Selection Function

Contents

In this chapter, we present the second contribution of our study on the Correlation Power Analysis (CPA) attacks on Ascon-AEAD. This includes extending the selection function to operate on multiple bits.

4.1. Context and Motivation

In the previous chapter, we have detailed the use of $\tilde{z}_0^j$ as the selection function. We observe that using $\tilde{z}_0^j$ as the selection function means exploiting the leakages of a single bit z_0^j in the 64-bit word z_0. These leakages are modeled as the Hamming weight of $\tilde{z}_0^j$, denoted by $\mathrm{HW}(\tilde{z}_0^j)$. Note that, since $\tilde{z}_0^j$ is a 1-bit value, its Hamming weight equals the value itself, *i.e.*, $\mathrm{HW}(\tilde{z}_0^j) = \tilde{z}_0^j$.

In software implementations, the 64-bit word z_0 may be stored in eight 8-bit registers, two 32-bit registers, or a single 64-bit register, depending on the device architecture. The activity of these registers (load/store) is known to leak information about their data through power consumption [MOP07]. In an ideal attack scenario, one should consider the entire register when making hypotheses about power consumption [THM+07]. This typically results in a higher correlation with the measured traces. For example, in CPA attacks on an 8-bit AES implementation, modeling power consumption with the Hamming weight of the 8-bit S-box output is more effective than using only the most significant bit [BCO04]. However, for large word sizes such as 32 bits, this approach becomes computationally intensive, since the CPA must be run a large number of times.

Considering Ascon-AEAD, the selection function proposed by Samwel and Daemen [SD17] is limited to one bit z_0^j (hereafter referred to as the *1-bit selection function*).

In other words, the leakage hypotheses are modeled solely from the value of a single bit within a machine register. This still works because, as pointed out by Brier *et al.* [BCO04], a partial correlation exists when modeling the leakages from only part of register data. More precisely, the partial correlation coefficient ρ_d computed from d independent bits out of m bits ($d \leq m$) and the correlation coefficient ρ_m computed from all m bits have the following relation:

$$\rho_d = \rho_m \sqrt{\frac{d}{m}}.$$

Modeling the leakages by $\tilde{z}_0^j$, as in the previous chapter, corresponds to the case $d = 1$. The value of m corresponds to the register bit-width, which can be 8, 32, or 64, depending on the device architecture and implementation. The equation above also indicates that increasing the value of d makes the partial correlation ρ_d higher and closer to ρ_m.

In this chapter, we explore the feasibility of using a *multi-bit selection function* ($d \geq 2$) to CPA attacks on Ascon-AEAD. Specifically, we present the following results:

- First, we extend the 1-bit selection function to work on multiple bits. Through experiments, we demonstrate the advantages of this extension in terms of success rates, the number of traces required for a successful CPA, and the number of CPA runs needed for full key recovery.
- Second, we present some cautionary notes that may lead to a failed CPA in practice. These notes aim to help avoid potential pitfalls and increase the likelihood of success.

For the sake of reproducibility, we publish the source code of the experiments at:

`https://github.com/nvietsang/multibitcpa-ascon`.

Part of the results presented in this chapter were published and presented in the *International Conference on Security and Cryptography (SECRYPT 2025)* [NGC25a].

4.2. Extending to Multi-bit Selection Functions

In this section, we describe the extension of the selection function from one bit to multiple bits. As before, we detail the analysis of the selection function $\tilde{z}_0^j$ in Equation 3.4. A similar analysis applies to other selection functions $\tilde{z}_1^j$ and $\tilde{z}_4^j$.

4.2.1. Extension Method

From the bitsliced computation of the S-box in Equation 2.1, the S-box output bit y_0^j, where $0 \leq j \leq 63$ denotes the index of the j-th bit of a 64-bit word, is computed as follows:

$$y_0^j = n_1^j k_0^j \oplus n_0^j \oplus k_1^j k_0^j \oplus k_1^j \oplus k_0^j \mathtt{IV}^j \oplus k_0^j \oplus \mathtt{IV}^j. \tag{4.1}$$

Recall that in Chapter 3, y_0^j is fine-tuned by removing the constant term $k_1^j k_0^j \oplus k_1^j \oplus k_0^j \mathtt{IV}^j \oplus k_0^j \oplus \mathtt{IV}^j$, following the work of Samwel and Daemen [SD17]. This yields $\tilde{y}_0^j$,

defined as:

$$\tilde{y}_0^j = n_1^j k_0^j \oplus n_0^j$$

A bit at the linear layer output, z_0^j, is chosen as the attack point, and the selection function is given in Equation 3.4. For convenience, we rewrite this selection function here:

$$\begin{aligned}\tilde{z}_0^j &= \tilde{y}_0^j \oplus \tilde{y}_0^{j+19} \oplus \tilde{y}_0^{j+28} \\ &= \left(k_0^j(n_1^j \oplus 1) \oplus n_0^j\right) \\ &\oplus \left(k_0^{j+19}(n_1^{j+19} \oplus 1) \oplus n_0^{j+19}\right) \\ &\oplus \left(k_0^{j+28}(n_1^{j+28} \oplus 1) \oplus n_0^{j+28}\right).\end{aligned} \tag{4.2}$$

Note that each variable involved in Equation 4.2 is a 1-bit variable, indexed at j, $j+19$, and $j+28$. Our goal is to extend Equation 4.2 to work on multi-bit variables, so that the hypothetical leakages can be modeled by the Hamming weight of several bits of $\tilde{z}_0$, rather than one bit at index j. Figure 4.1 depicts the main idea of this extension from a 1-bit to a multi-bit selection function.

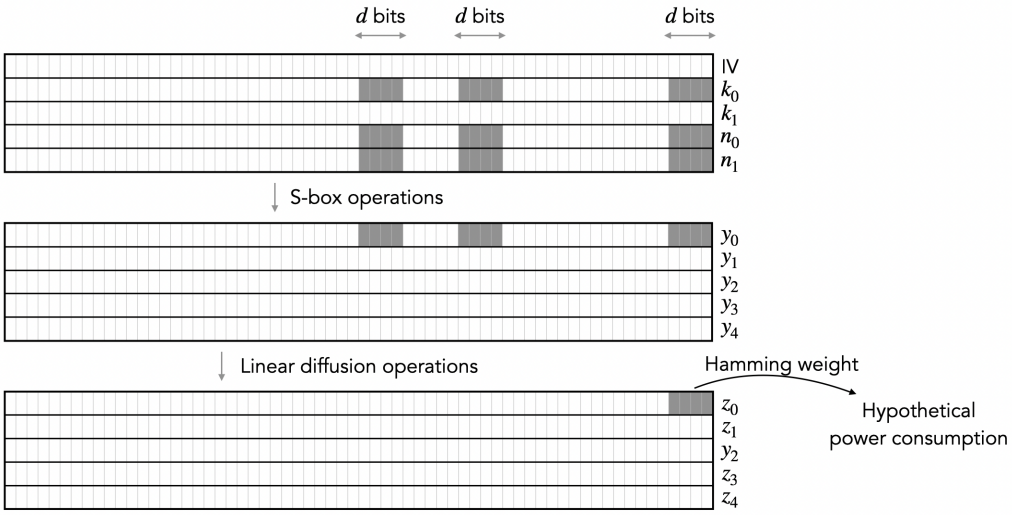


Figure 4.1.: Illustration of a d-bit selection function. In this example, $j = 0$ and $d = 4$.

To extend the selection function, we first revisit the fine-tuning process that derives the function of $\tilde{z}_0^j$, but from a different perspective. Let δ_0^j denote the constant term containing the key and initialization vector variables in Equation 4.1:

$$\delta_0^j = k_1^j k_0^j \oplus k_1^j \oplus k_0^j \mathtt{IV}^j \oplus k_0^j \oplus \mathtt{IV}^j.$$

Equation 4.1 is then rewritten as:

$$y_0^j = n_1^j k_0^j \oplus n_0^j \oplus \delta_0^j,$$

and the attack point z_0^j is rewritten as:

$$\begin{aligned} z_0^j &= y_0^j \oplus y_0^{j+19} \oplus y_0^{j+28} \\ &= \left(k_0^j(n_1^j \oplus 1) \oplus n_0^j\right) \\ &\oplus \left(k_0^{j+19}(n_1^{j+19} \oplus 1) \oplus n_0^{j+19}\right) \\ &\oplus \left(k_0^{j+28}(n_1^{j+28} \oplus 1) \oplus n_0^{j+28}\right) \\ &\oplus \Delta_0^j \\ &= \tilde{z}_0^j \oplus \Delta_0^j, \end{aligned} \tag{4.3}$$

where $\Delta_0^j = \delta_0^j \oplus \delta_0^{j+19} \oplus \delta_0^{j+28}$.

We observe that the selection function derivation of Samwel and Daemen [SD17], *i.e.*, the removal of the constant term δ_0^j in Equation 4.1, is equivalent to setting $\delta_0^j = 0$, and similarly, setting $\delta_0^{j+19} = \delta_0^{j+28} = 0$. Consequently, $\Delta_0^j = 0$ and $z_0^j = \tilde{z}_0^j$ in Equation 4.3. Modeling the leakages on $\tilde{z}_0^j$ does not affect the CPA success because Δ_0^j is a 1-bit variable, $\Delta_0^j \in \{0, 1\}$. If its actual value is 0, then $z_0^j = \tilde{z}_0^j$ and our modeling is correct. If its actual value is 1, then $z_0^j = \tilde{z}_0^j \oplus 1$, meaning that the distribution of z_0^j is negated, as demonstrated in Table 4.1. This negation just changes the sign of the correlation coefficient, but not the *absolute* correlation coefficient. In short, the *absolute* correlation coefficient is the same for $\Delta_0^j = 0$ and $\Delta_0^j = 1$.

	$\mathrm{HW}(z_0^j) = z_0^j = \tilde{z}_0^j \oplus \Delta_0^j$	
$\tilde{z}_0^j$	$\Delta_0^j = 0$	$\Delta_0^j = 1$
0	0	1
1	1	0

Table 4.1.: Distribution of the Hamming weight of z_0^j for different values of Δ_0^j.

However, we cannot directly apply the strategy of setting $\Delta_0^j = 0$ when considering a multi-bit selection function. Let $z_0^{j..j+d}$ denote d consecutive bits of the 64-bit word z_0, from index j to $j + d - 1$ ($d \geq 1$). For multi-bit variables, Equation 4.3 becomes:

$$\begin{aligned} z_0^{j..j+d} &= y_0^{j..j+d} \oplus y_0^{j+19..j+19+d} \oplus y_0^{j+28..j+28+d} \\ &= \left(k_0^{j..j+d}(n_1^{j..j+d} \oplus \mathbf{1}) \oplus n_0^{j..j+d}\right) \\ &\oplus \left(k_0^{j+19..j+19+d}(n_1^{j+19..j+19+d} \oplus \mathbf{1}) \oplus n_0^{j+19..j+19+d}\right) \\ &\oplus \left(k_0^{j+28..j+28+d}(n_1^{j+28..j+28+d} \oplus \mathbf{1}) \oplus n_0^{j+28..j+28+d}\right) \\ &\oplus \Delta_0^{j..j+d} \\ &= \tilde{z}_0^{j..j+d} \oplus \Delta_0^{j..j+d}. \end{aligned} \tag{4.4}$$

In Equation 4.4, we cannot set $\Delta_0^{j..j+d} = \mathbf{0}$ and model the leakages on $\tilde{z}_0^{j..j+d}$ (for $d \geq 2$) as before, because the distribution of $\mathrm{HW}(z_0^{j..j+d})$ does not simply negate as the single-bit case. Table 4.2 shows the distribution of $\mathrm{HW}(z_0^{j..j+d})$ when $d = 2$. Different values of $\Delta_0^{j..j+d}$ can lead to different absolute correlation coefficients when

correlated with the power traces. Therefore, in CPA, we need to guess the value of $\Delta_0^{j..j+d}$ if we model the power consumption on multiple bits (*i.e.*, when $d \geq 2$).

Nevertheless, it is not necessary to brute-force all 2^d possible values of the d-bit variable $\Delta_0^{j..j+d}$. Instead, we only need to guess value in the first half of the search space $[0, 2^{d-1} - 1]$. This is because of the following two reasons:

- The distribution produced when $\Delta_0^{j..j+d} = \mathtt{a}$ is correlated to that produced when $\Delta_0^{j..j+d} = \neg\mathtt{a}$ with a coefficient of -1, for $\mathtt{a} \in [0, 2^d - 1]$ and $\neg$ is a bitwise negation. This is formally stated and proven in Lemma 4.1.
- Bitwise negation ($\neg$) defines a bijection between the first half $[0, 2^{d-1} - 1]$ and the second half $[2^{d-1}, 2^d - 1]$ of the search space. This is formally stated and proven in Lemma 4.2.

Consequently, the absolute correlation coefficient between the hypothetical leakages and the measured power traces is the same when $\Delta_0^{j..j+d} = \mathtt{a}$ and $\Delta_0^{j..j+d} = \neg\mathtt{a}$. For example, the two distributions in red, corresponding to $\Delta_0^{j..j+2} = 00$ and $\Delta_0^{j..j+2} = 11$, are fully correlated in Table 4.2. The same holds for the two distributions in blue.

	$\mathrm{HW}(z_0^{j..j+2}) = \mathrm{HW}(\tilde{z}_0^{j..j+2} \oplus \Delta_0^{j..j+2})$			
$\tilde{z}_0^{j..j+2}$	$\Delta_0^{j..j+2} = 00$	$\Delta_0^{j..j+2} = 01$	$\Delta_0^{j..j+2} = 10$	$\Delta_0^{j..j+2} = 11$
00	0	1	1	2
01	1	2	0	1
10	1	0	2	1
11	2	1	1	0

Table 4.2.: Distribution of the Hamming weight of $z_0^{j..j+d}$ for different values of $\Delta_0^{j..j+d}$ when $d = 2$. Two columns in the same color are correlated with a coefficient of -1.

Lemma 4.1. *Let $\tilde{z}$ and Δ be two d-bit variable, and let $\mathtt{a} \in [0, 2^d - 1]$ be a d-bit value. The two distributions, $HW(\tilde{z} \oplus \mathtt{a})$ and $HW(\tilde{z} \oplus \neg\mathtt{a})$, corresponding to $\Delta = \mathtt{a}$ and $\Delta = \neg\mathtt{a}$, are fully correlated with coefficient of -1.*

Proof of Lemma 4.1. Let $\mathbf{1}$ denote the value of d bit ones. Since $\neg\mathtt{a} = \mathtt{a} \oplus \mathbf{1}$, we have

$$\tilde{z} \oplus \neg\mathtt{a} = \tilde{z} \oplus (\mathtt{a} \oplus \mathbf{1}) = (\tilde{z} \oplus \mathtt{a}) \oplus \mathbf{1} = \neg(\tilde{z} \oplus \mathtt{a}).$$

Therefore,

$$\mathrm{HW}(\tilde{z} \oplus \mathtt{a}) + \mathrm{HW}(\tilde{z} \oplus \neg\mathtt{a}) = \mathrm{HW}(\tilde{z} \oplus \mathtt{a}) + \mathrm{HW}(\neg(\tilde{z} \oplus \mathtt{a})) = d.$$

Let us define the two random variables:

$$X := \mathrm{HW}(\tilde{z} \oplus \mathtt{a}),$$
$$Y := \mathrm{HW}(\tilde{z} \oplus \neg\mathtt{a}).$$

Then, $Y = d - X$. In this proof, we compute the Pearson correlation through the covariance and variances of X and Y. The covariance is

$$\mathrm{Cov}(X, Y) = \mathrm{Cov}(X, d - X) = \mathrm{Cov}(X, d) - \mathrm{Cov}(X, X) = 0 - \mathrm{Var}(X) = -\mathrm{Var}(X),$$

and the variance is

$$\mathrm{Var}(Y) = \mathrm{Var}(d - X) = \mathrm{Var}(X).$$

The Pearson correlation thus is

$$\rho_{X,Y} = \frac{\mathrm{Cov(X,Y)}}{\sqrt{\mathrm{Var}(X)\mathrm{Var}(Y)}} = \frac{-\mathrm{Var}(X)}{\mathrm{Var}(X)} = -1.$$

Hence, the two distributions are fully correlated with coefficient of -1. □

Lemma 4.2. *Let $\mathcal{S}_0 = [0, 2^{d-1} - 1]$ and $\mathcal{S}_1 = [2^{d-1}, 2^d - 1]$ be the first and the second halves of the search space. The map $f : \mathcal{S}_0 \to \mathcal{S}_1$, where $f(x) = \neg x$, is a bijection.*

Proof of Lemma 4.2. The computation of bitwise negation can be written as

$$f(x) = \neg x = (2^d - 1) - x.$$

First, we prove that f is injective. Let $x_1, x_2 \in \mathcal{S}_0$. If $f(x_1) = f(x_2)$, then $(2^d - 1) - x_1 = (2^d - 1) - x_2 \implies x_1 = x_2$. Thus, no two different inputs map to the same output. Therefore, injectivity is proven.

Second, we prove that f is surjective. Let $y \in \mathcal{S}_1$. We find an $x \in \mathcal{S}_0$ such that $y = f(x)$. Given $y = (2^d - 1) - x$, we have $x = (2^d - 1) - y$. Since $y \in [2^{d-1}, 2^d - 1]$, it follows that $x = (2^d - 1) - y \in [0, 2^{d-1} - 1] = \mathcal{S}_0$. Therefore, surjectivity is proven.

As f is both injective and surjective, it is bijective. □

		Number of bits			
Intermediate value $z_0^{j..j+d}$	d	$d = 1$	$d = 2$	$d = 3$	$d = 4$
$\Delta_0^{j..j+d}$	$d - 1$	$0^{(*)}$	1	2	3
Key	$3d$	3	6	9	12
Nonce	$6d$	6	12	18	24
Number of CPA runs		47	28	19	15

Table 4.3.: Involved bits in each CPA run and number of CPA runs for the full key recovery. (*) There is only one possible value for Δ_0^j in this case, which is $\Delta_0^j = 0$.

In Table 4.3, we present the number of key and nonce bits involved in the computation of the intermediate value $z_0^{j..j+d}$ for different values of d. Figure 4.1 helps clarify this table. By modeling the leakages on a d-bit intermediate value, we can recover $3d$ bits of the key in each CPA run. As a result, the number of CPA runs required for full key recovery decreases significantly as d increases. We employ the SAT solver for the set cover problem presented in Section 3.3 to find the optimal number of CPA runs

for each $d \in \{1, 2, 3, 4\}$. The detailed results of this finding are provided in Table 3.6 and Table 4.4, and the total numbers are summarized in Table 4.3. As we can see, 47 CPA runs are required for $d = 1$ to recover the full 128-bit key, whereas only 15 runs are needed for $d = 4$.

$d = 2$	
14 tuples for k_0 (j..j+2,j+19..j+21,j+28..j+30)	14 tuples for k_1 (j..j+2,j+61..j+63,j+39..j+41)
(0, 1,19,20,28,29)	(5, 6, 2, 3,44,45)
(2, 3,21,22,30,31)	(7, 8, 4, 5,46,47)
(4, 5,23,24,32,33)	(15,16,12,13,54,55)
(8, 9,27,28,36,37)	(17,18,14,15,56,57)
(18,19,37,38,46,47)	(19,20,16,17,58,59)
(25,26,44,45,53,54)	(21,22,18,19,60,61)
(34,35,53,54,62,63)	(23,24,20,21,62,63)
(38,39,57,58, 2, 3)	(25,26,22,23, 0, 1)
(40,41,59,60, 4, 5)	(33,34,30,31, 8, 9)
(42,43,61,62, 6, 7)	(35,36,32,33,10,11)
(48,49, 3, 4,12,13)	(40,41,37,38,15,16)
(50,51, 5, 6,14,15)	(42,43,39,40,17,18)
(52,53, 7, 8,16,17)	(51,52,48,49,26,27)
(55,56,10,11,19,20)	(53,54,50,51,28,29)
$d = 3$	
10 tuples for k_0 (j..j+3,j+19..j+22,j+28..j+31)	9 tuples for k_1 (j..j+3,j+61..j,j+39..j+42)
(24,25,26,43,44,45,52,53,54)	(2, 3, 4,63, 0, 1,41,42,43)
(27,28,29,46,47,48,55,56,57)	(5, 6, 7, 2, 3, 4,44,45,46)
(29,30,31,48,49,50,57,58,59)	(14,15,16,11,12,13,53,54,55)
(32,33,34,51,52,53,60,61,62)	(21,22,23,18,19,20,60,61,62)
(34,35,36,53,54,55,62,63, 0)	(27,28,29,24,25,26, 2, 3, 4)
(37,38,39,56,57,58, 1, 2, 3)	(33,34,35,30,31,32, 8, 9,10)
(40,41,42,59,60,61, 4, 5, 6)	(40,41,42,37,38,39,15,16,17)
(51,52,53, 6, 7, 8,15,16,17)	(50,51,52,47,48,49,25,26,27)
(54,55,56, 9,10,11,18,19,20)	(59,60,61,56,57,58,34,35,36)
(57,58,59,12,13,14,21,22,23)	
$d = 4$	
8 tuples for k_0 (j..j+4,j+19..j+23,j+28..j+32)	7 tuples for k_1 (j..j+4,j+61..j+1,j+39..j+43)
(28,29,30,31,47,48,49,50,56,57,58,59)	(6, 7, 8, 9, 3, 4, 5, 6,45,46,47,48)
(32,33,34,35,51,52,53,54,60,61,62,63)	(16,17,18,19,13,14,15,16,55,56,57,58)
(36,37,38,39,55,56,57,58, 0, 1, 2, 3)	(24,25,26,27,21,22,23,24,63, 0, 1, 2)
(40,41,42,43,59,60,61,62, 4, 5, 6, 7)	(34,35,36,37,31,32,33,34, 9,10,11,12)
(44,45,46,47,63, 0, 1, 2, 8, 9,10,11)	(42,43,44,45,39,40,41,42,17,18,19,20)
(55,56,57,58,10,11,12,13,19,20,21,22)	(52,53,54,55,49,50,51,52,27,28,29,30)
(59,60,61,62,14,15,16,17,23,24,25,26)	(60,61,62,63,57,58,59,60,35,36,37,38)
(60,61,62,63,15,16,17,18,24,25,26,27)	

Table 4.4.: Tuples of indexes for each $3d$-bit key recovery, where $d \in \{2, 3, 4\}$.

The number of CPA runs in Table 4.3 also suggests that using $d \geq 2$ enables *second-order success*. After each CPA run, instead of choosing only the single top key candidate, we select the *top two* candidates with the highest correlations. A brute-force search is then performed over these candidates to determine the correct key.

The brute-force spaces are 2^{28} for $d = 2$, 2^{19} for $d = 3$, and 2^{15} for $d = 4$, all of which are computationally feasible in practice. In particular, for $d = 4$, *third-order success* remains possible with the brute-force space of $3^{15} \approx 2^{22.77}$. In contrast, the brute-force space for second-order success with $d = 1$ is 2^{47}, which is impractical on a classical computer.

Although it is possible to increase d to model the leakages on more bits, our analysis is limited to $d \in \{1, 2, 3, 4\}$. This limitation is due to the rapidly increasing time complexity for key recovery at higher values of d. We discuss this complexity in the next section.

It also worth noting that one could consider d non-consecutive bits. However, this would complicate the determination of the number of CPA runs compared to Table 4.4, and such determination may not be automatically solved by the SAT solver as before. Hence, in this chapter, we limit our study to d consecutive bits.

4.2.2. Experimental results

For our experiments, we use the ARMv6 implementation provided by the Ascon team,[1] and acquire power traces using a ChipWhisperer Lite. We perform the CPA attacks with a d-bit selection function, where $d \in \{1, 2, 3, 4\}$, and measure the success rates for each case over an increasing number of traces.

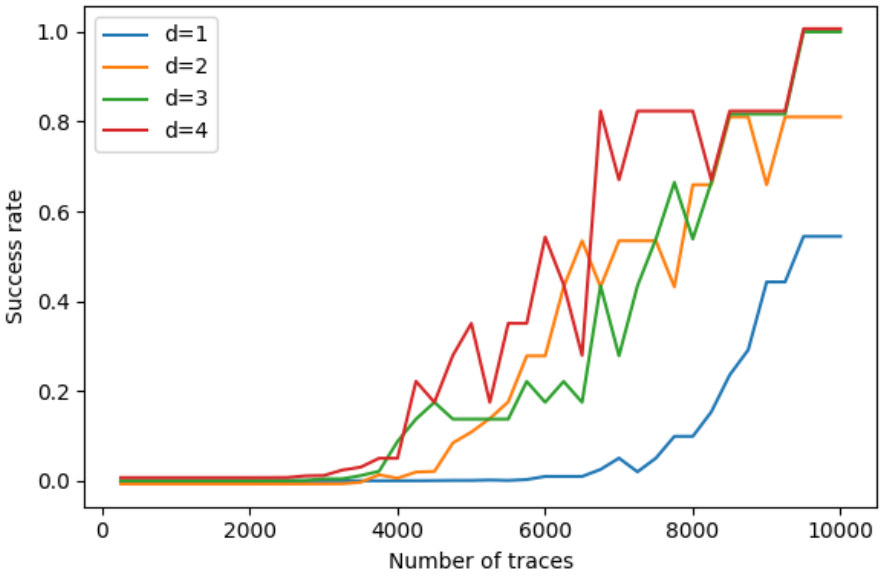


Figure 4.2.: Success rates of the full key recovery for different values of d.

Measuring success rates for full key recovery requires repeating the analysis numerous times (typically a hundred or more) for different numbers of traces, which is very time-consuming. Therefore, we employ an estimation instead. Specifically, we focus on the success rates of recovering $3d$ bits of k_0 and $3d$ bits of k_1. These success rates are measured by running the CPA many times with several sets of traces to recover different tuples of key bits. Table 3.6 and Table 4.4 present these tuples of indexes of

[1] See https://github.com/ascon/ascon-c/tree/main/crypto_aead/asconaead128/armv6

key bits for $d \in \{1, 2, 3, 4\}$. The results are raised to the power corresponding to the number of CPA runs needed, and then multiplied together to estimate the success rates for full key recovery.

Figure 4.2 shows the success rates for full key recovery for different values of d. As expected, the success rates for $d \geq 2$ are higher than those for $d = 1$. In other words, fewer traces are required for the CPA to achieve the same success rates when $d \geq 2$ compared to $d = 1$. This confirms that modeling leakages on multiple bits can improve success rates of the attack.

However, it can also be observed in Figure 4.2 that the differences in success rates for $d = 2$, $d = 3$, and $d = 4$ are not significant. This may be due to the partial correlation between the distributions produced by different key candidates. We discuss this behavior in more detail in the next section.

Number of bits for $z_0^{j..j+d}$	d	$d = 1$	$d = 2$	$d = 3$	$d = 4$
Number of candidates (key and $\Delta_0^{j..j+d}$)	2^{4d-1}	2^3	2^7	2^{11}	2^{15}
Measured time (sequential recovery)		3m	10m	2.7h	34.7h
Measured time (parallel recovery)				8m	2.3h

Table 4.5.: Measured time for full key recovery. The number of power traces is fixed to 10000, and each trace has 1000 samples. In sequential recovery, the tuples of key bits are sequentially recovered. In parallel recovery, they are recovered in parallel.

To compare the time complexity of full key recovery, we measure the elapsed time of the CPA for each value of d and report the results in Table 4.5. As d increases, the number of key candidates grows exponentially, making full key recovery more time-consuming. For example, it takes 34.7h for $d = 4$. In this case, recovering the tuples of key bits in parallel can significantly reduce the time. We execute 15 CPA runs simultaneously in a cluster for full key recovery, which reduces the time to 2.3h.

4.3. Cautionary Notes for Practical Attacks

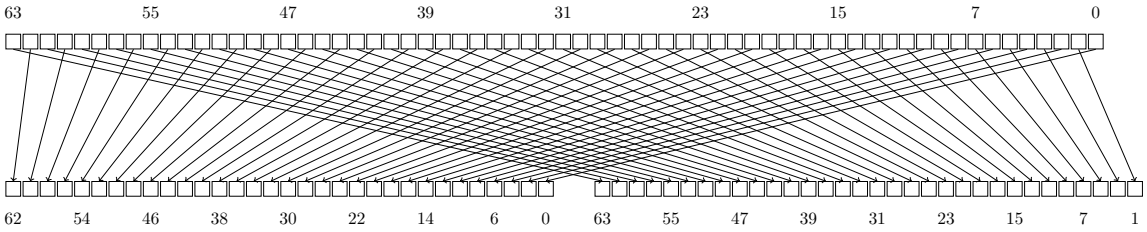
In this section, we present the factors that can lead to a failed CPA when using a multi-bit selection function. Being aware of these factors can help avoid potential pitfalls and improve the chances of successful key recovery. The first factor relates to an implementation technique for Ascon-AEAD on smaller-size register architectures. The second arises from partial correlations between the distributions of key candidates.

4.3.1. Attacks on Implementations with Bit Interleaving

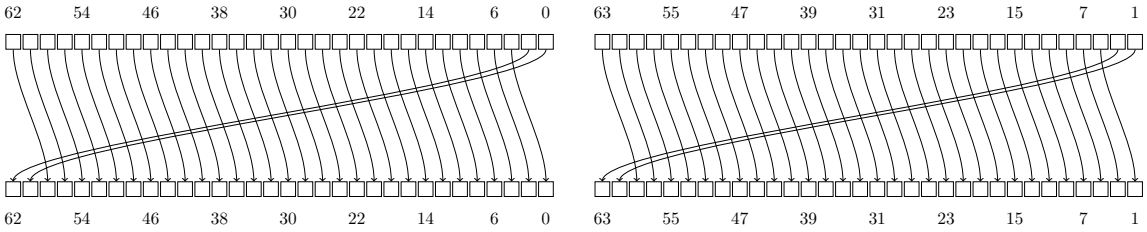
In Ascon-AEAD's design, the state consists of five 64-bit words, as depicted in Figure 2.5. When implemented on 8-bit and 32-bit architectures, a naive solution is to split each word into eight 8-bit registers or two 32-bit registers with consecutive bits. This approach poses no problem for the bitsliced S-box implementation, but

does create an issue for the linear diffusion layer. With this solution, a rotation requires additional instructions and temporary registers.

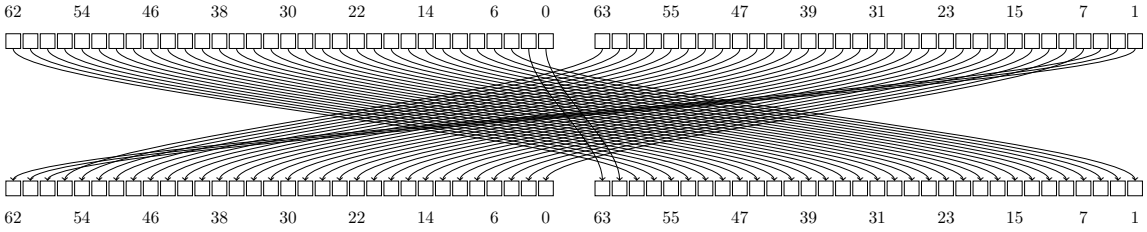
To address this, Bertoni *et al.* [BDP+11] proposed the use of a technique called *bit interleaving*. In this technique, consecutive bits of a word are "interleaved" across the smaller registers, enabling rotations to be carried out without additional instructions or temporary registers. Figure 4.3 illustrates the arrangements of bits of a word in two 32-bit registers with interleaving technique. The bits at odd indexes are placed in one register, while the bits at even indexes are placed in the other. A rotation on the original 64-bit word can be implemented by two rotations on the 32-bit registers. A rotation by an even offset $2r$ is done by rotating both registers by r, as shown in Figure 4.3b. A rotation by an odd offset $2r + 1$ is done by rotating the register containing even indexes by r and the register containing odd indexes by $r+1$, following by swapping the two registers, as shown in Figure 4.3c.



a) Splitting a 64-bit word into two interleaved 32-bit registers.



b) Example of a right rotation by 4. Both registers are rotated right by 2.



c) Example of a right rotation by 5. The left register is rotated right by 2. The right register is rotated right by 3.

Figure 4.3.: Visualization of bit interleaving.

In Section 4.2, we extend the modeling of hypothetical leakages from one bit to multiple bits of an intermediate value. Recall that the actual power consumption is assumed to depend strongly on the register data, primarily through register activity such as load and store operations. Therefore, to correctly model the power consumption, the multiple bits must reside in the same register.

Our extension method presented in Section 4.2 focuses only on d consecutive bits of a word. Specifically, the intermediate value $z_0^{j..j+d}$ is the concatenation of d bits:

$z_0^j, \ldots, z_0^{j+d-1}$. For example, when $d = 2$ and $j = 0$, $z_0^{0..2}$ is the concatenation of z_0^0 and z_0^1. This matches the implementation in our experiments, which uses the naive solution of splitting the 64-bit words. The tuples of key-bit indexes in Table 4.4 for key recovery also correspond to this bit arrangement.

In practice, if the target device uses bit interleaving for its Ascon-AEAD implementation, d consecutive bits in a register are not the same as d consecutive bits in the original word. In the previous example with $d = 2$ and $j = 0$, the bit z_0^0 and z_0^1 now reside in two different registers. Therefore, modeling the power consumption on these two bits is not correct. Instead, we need to model it on z_0^0 and z_0^2, which are the two consecutive bits in the 32-bit register containing even indexes. In summary, caution is needed when selecting multiple bits for modeling in attacks on implementations using the bit interleaving technique.

In contrast, CPA using a 1-bit selection function ($d = 1$) is not affected by implementations with the bit interleaving technique. This is not surprising, as the hypothetical power consumption is modeled using only one-bit intermediate value.

4.3.2. Partial Correlations between Key Candidates

In Section 3.2, we show that the distributions associated with different key candidates are uncorrelated (correlation coefficient of 0) for $d = 1$. We observe that our extension of the selection function to multiple bits ($d \geq 2$), presented in Section 4.2, can be intuitively seen as the concatenation of d 1-bit selection functions. This concatenation is suspected to introduce correlations between the distributions of different key candidates, which may in turn affect the success of the CPA.

To investigate this issue, we examine the correlations between distributions of all possible key pairs for $d = 2$. Figure 4.4 visualizes the results. As we can see, the distribution of a key candidate is partially correlated that of 15 other key candidates, with a correlation coefficient of 0.5 (represented by light blue cells). However, they are not fully correlated with an absolute coefficient of 1. Hence, the correct candidate is still expected to produce hypothetical leakages that are most strongly correlated with the measured power traces. This is confirmed by the CPA's success when given enough traces, as shown in Figure 4.2.

Figure 4.5 shows an example of partial correlation in our key recovery. The clearly distinguished correct key is followed by a group of keys with relatively high correlations, represented in dark gray. This group corresponds to 15 candidates at the top of the ranking table in Table 4.6. One can verify that most of candidates in this group actually correspond to the light blue cells in row 23 in Figure 4.4 (*i.e.*, the row of the correct key candidate). This means that the wrong key candidates whose distributions are partial correlated with that of the correct key tend to have higher correlations than other wrong key candidates.

Note that as the value of d increases, the highest partial correlation between the distributions of key pairs also increases, as shown in Table 4.7. This may explain the observation in Figure 4.2, that the differences in success rates for $d = 2$, $d = 3$, and $d = 4$ are not significant. In practice, depending on the noise level in the measured power traces, this may cause wrong key candidates to have concurrently

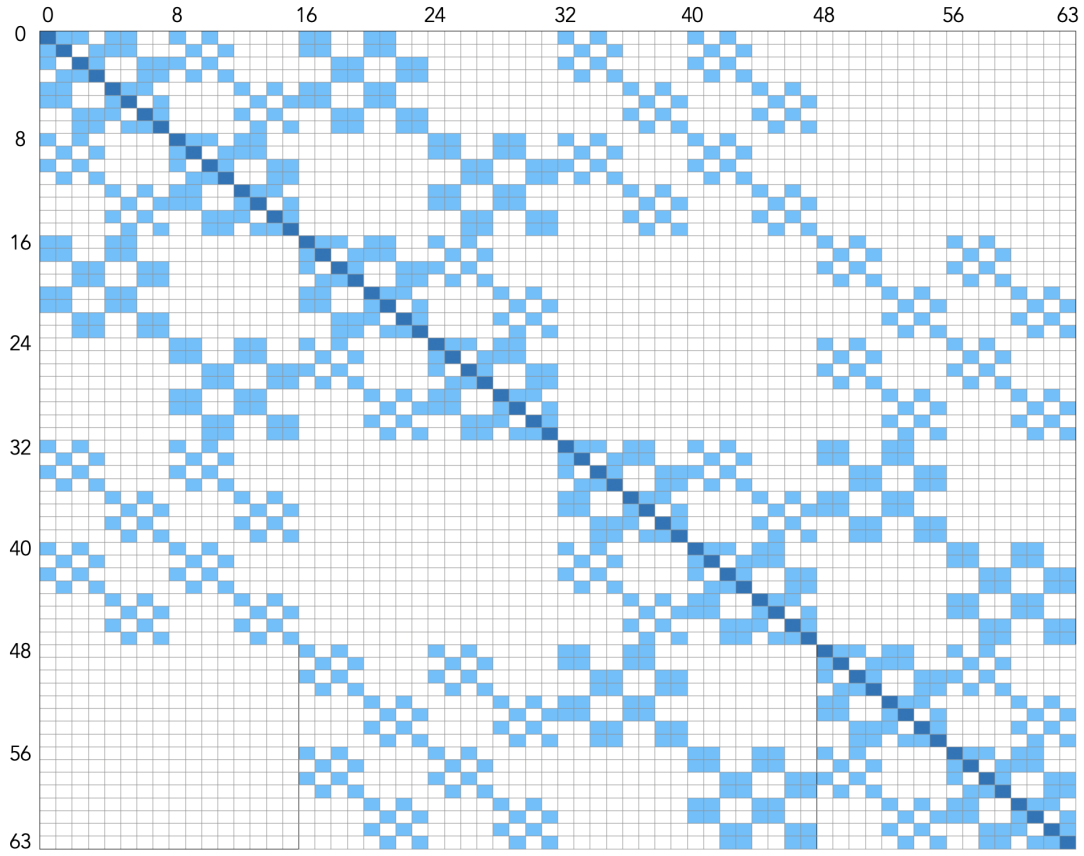


Figure 4.4.: Correlations between distributions of all possible key pairs when $d = 2$. Dark blue, light blue and white cells correspond to correlation coefficient of 1, 0.5 and 0, respectively.

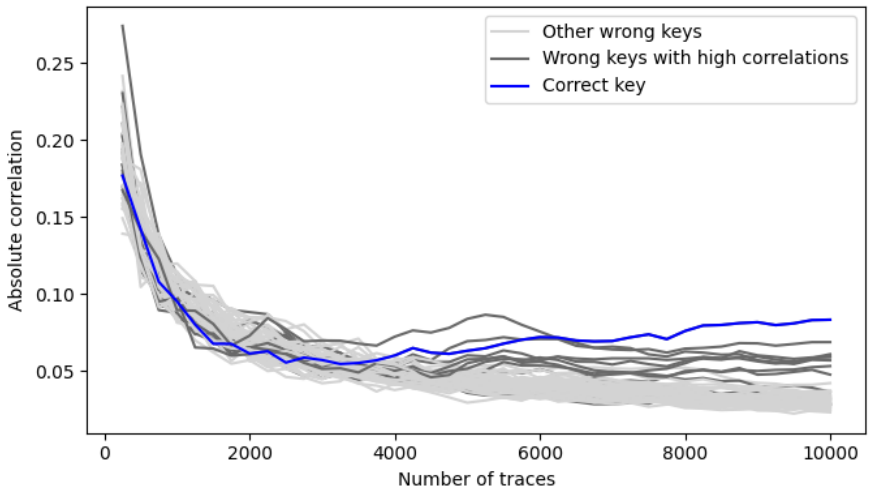


Figure 4.5.: Correlations over increasing number of traces for $d = 2$. The dark gray lines correspond to the first 15 key candidates in Table 4.6.

high correlations. In contrast, for $d = 1$, the distributions of all possible key pairs are uncorrelated with each other.

Rank	Key	Corr.	Rank	Key	Corr.	Rank	Key	Corr.	Rank	Key	Corr.
1	63	0.076	17	40	0.045	33	25	0.040	49	6	0.037
2	23	0.072	18	46	0.045	34	35	0.040	50	7	0.037
3	21	0.071	19	13	0.044	35	57	0.039	51	60	0.037
4	61	0.059	20	33	0.044	36	5	0.039	52	44	0.037
5	31	0.058	21	8	0.044	37	59	0.039	53	18	0.036
6	55	0.057	22	0	0.043	38	45	0.039	54	11	0.035
7	53	0.056	23	48	0.043	39	27	0.039	55	30	0.035
8	29	0.054	24	17	0.042	40	42	0.039	56	20	0.035
9	3	0.051	25	9	0.042	41	41	0.039	57	24	0.035
10	56	0.051	26	28	0.041	42	49	0.039	58	50	0.035
11	1	0.050	27	47	0.041	43	52	0.038	59	4	0.034
12	58	0.046	28	2	0.041	44	19	0.038	60	22	0.034
13	12	0.045	29	43	0.041	45	37	0.038	61	62	0.034
14	14	0.045	30	32	0.041	46	51	0.038	62	54	0.034
15	38	0.045	31	39	0.041	47	34	0.038	63	36	0.033
16	16	0.045	32	10	0.040	48	15	0.037	64	26	0.032

Table 4.6.: Key candidates ranked by correlations at 6000 traces for a recovery with $d = 2$. The correct candidate is highlighted in blue. The first 15 candidates correspond to the dark gray lines in Figure 4.5, and most of these candidates correspond to the light blue cells in row 23 of Figure 4.4.

Number of bits for $z_0^{j..j+d}$	$d = 1$	$d = 2$	$d = 3$	$d = 4$
Highest partial correlation	0	0.5	0.625	0.75

Table 4.7.: Highest partial correlations between distributions of key pairs for different values of d.

4.4. Conclusion

In this chapter, we present the extension of the 1-bit selection function to work on multiple bits. Our results show that CPA using a multi-bit selection function requires fewer power traces to achieve the same success rates as when using 1-bit selection function for full key recovery. We also provide a comparison of time complexity for different number of bits. Finally, we discuss factors that could lead to a failed CPA when using a multi-bit selection function.

This chapter also concludes the first part of our contributions to power analysis attacks on Ascon-AEAD. In the next part, we explore the second category of physical attacks, which involves injecting faults into cryptographic executions and exploiting their effects through statistical analysis.

Part II.

Fault Attacks

5

Analysis of SIFA on Nonce-based Authenticated Encryption

Contents

In this chapter, we present the contribution from our study on the threat of Fault Attacks (FA) against nonce-based authenticated encryption. We focus on a prominent type of FA, namely Statistical Ineffective Fault Analysis (SIFA), and investigate it from a different and noteworthy perspective: the factors that can lead to attack failure. This highlights aspects that have been largely overlooked in previous studies.

5.1. Context and Motivation

In the literature, Differential Fault Analysis (DFA)-like attacks [BS97] have been shown to be ineffective against authenticated encryption schemes due to the use of a unique nonce in each encryption [SC16]. To address this limitation, Dobraunig *et al.* (SAC 2018) [DMM+19] proposed the use of Statistical Ineffective Fault Analysis (SIFA) [DEK+18; DEG+18]. The authors demonstrated that their attack strategy applies to a wide range of authenticated encryption schemes where the nonce is mixed with the key during the initialization phase. The core idea of the attack is to cause a biased distribution in a bit before the S-box computation of the second initialization round by *ineffective faults*, and then recover the key through a statistical

analysis of this bias. The authors validated their approach by mounting attacks on 8-bit implementations of two authenticated encryption schemes based on the Keccak-f permutation [BDP+11], namely, Keyak [BDP+16b] and Ketje [BDP+16a]. To induce faults, they employed a common method, instruction skip, realized by clock glitching. The authors conjectured that their attack strategy can be adopted to other authenticated encryption schemes such as Ascon-AEAD.

In [DMM+19] and many other demonstrations of SIFA [DEK+18; DEG+18], instruction skip is a common used method to cause a bias in the intermediate value under ineffective faults. This skip typically targets an instruction involved in the S-box computation. However, it is often unclear which specific instruction was skipped in those demonstrations. This may lead one to think that blindly skipping any instruction in the S-box computation is sufficient to introduce a bias in the intermediate value under ineffective faults. However, we will show that this is not the case for certain nonce-based authenticated encryption implementations.

In this chapter, we provide a more in-depth analysis of the attack strategy introduced by Dobraunig *et al.* [DMM+19], with a focus on instruction skip as the fault method. Specifically, we present the following results:

- First, we model common instruction skip scenarios in practice and formalize the probability of a fault being ineffective. Our analysis shows that when an attacker skips an instruction, the probability of causing an ineffective fault depends on both the type of instruction and the device architecture. In particular, we show that skipping an `XOR` instruction on 32-bit or 64-bit systems is very unlikely to result in an ineffective fault, making SIFA practically inefficient.
- Second, we show that in certain authenticated encryption implementations, the intermediate value targeted by the attack strategy in [DMM+19] remains unbiased under ineffective faults, making SIFA inapplicable in these cases. We model such implementations and provide a formal proof for the uniformity of the targeted intermediate value. To support our findings, we present a case study on an 8-bit Ascon-AEAD implementation, where the intermediate value remains uniformly distributed under ineffective faults.

The results presented in this chapter was published and presented in the international workshop *Fault Detection and Tolerance in Cryptography (FDTC 2025)* [NGC25d].

5.2. Statistical Ineffective Fault Analysis

In this section, we provide the background on Statistical Ineffective Fault Analysis (SIFA). We then recall the attack strategy proposed by Dobraunig *et al.* [DMM+19] for applying SIFA to nonce-based authenticated encryption schemes and describe its application to Ascon-AEAD.

5.2.1. Attack Principle

Proposed by Dobraunig *et al.* [DEK+18], SIFA combines the ideas of Ineffective Fault Analysis (IFA) [Cla07] and Statistical Fault Analysis (SFA) [FJL+13b]. It exploits

the distribution caused by ineffective faults, *i.e.*, those that do not affect the outcome of a computation. As the fault effect depends on the *intermediate value* at the chosen *attack point*, the distribution of this value is often biased. Relying on solely ineffective faults makes SIFA applicable even in the presence of popular detection/infection countermeasures [DEK+18]. In the case of detection, the countermeasures can be seen as a *filter* for the ineffective faults.

Table 5.1 provides an example of the distribution table for a stuck-at-0 fault on 2-bit values. The probability that any value x becomes $x' = 00$ by a stuck-at-0 fault is 1 (first column). The probability that any value x becomes $x' \neq 00$ by a stuck-at-0 fault is 0 (other columns). The diagonal of red values represents the non-uniform distribution of ineffective faults ($x = x'$).

		x'			
		00	01	10	11
x	00	1	0	0	0
	01	1	0	0	0
	10	1	0	0	0
	11	1	0	0	0

Table 5.1.: Fault distribution table for 2-bit stuck-at-0 fault model [DEK+18].

The principle of SIFA is as follows. First, the attacker chooses an intermediate value as the attack point such that its calculation depends on parts of the key. Second, he forces the values at the attack point to follow a non-uniform distribution by fault inductions. In practice, these faults can be achieved with an instruction skip using clock glitches [DEK+18; DEG+18] or using laser [DEK+16]. Third, he collects a set of data (*e.g.*, plaintexts, ciphertexts, nonces) corresponding to the ineffective faults. Finally, key recovery is performed using this set. For each key guess, the attacker computes the intermediate value using the collected data. This value should follow an (unknown) non-uniform distribution when the key is correctly guessed. In contrast, the distribution of the calculated values is expected to be close to uniform for any wrong key guesses. Hence, the key guess corresponding to the calculated distribution that is "furthest" from uniformity can be identified as the correct key.

5.2.2. Application to Nonce-based Authenticated Encryption

As this chapter investigates more thoroughly the extension of SIFA to nonce-based authenticated encryption schemes by Dobraunig *et al.* [DMM+19], we now recap the principle of their attack strategy. The authors focus on the initialization phase of the authenticated decryption, where the nonce N and the key K are processed. The verification of the tag T in the decryption is considered as the filter for ineffective faults. The state before the application of the S-box in the second initialization round is chosen as the attack point. The distribution of one or multiple bits (intermediate value) at this state is assumed to be non-uniform for the filtered computations. A sufficient number of nonces corresponding to ineffective faults is collected for key recovery.

In the demonstration on Keyak and Ketje, the authors recovered the key by exploiting the biased distribution of a single bit before the application of the S-box in the second

round. They also detailed the analysis of the involved key bits in the calculation of the targeted bit. Their experiment was conducted on a 8-bit Xmega 128D4 microprocessor. An instruction in the S-box computation of the first round was skipped by clock glitches. This leads to the bias in the distribution of the targeted bit.

Application to Ascon-AEAD. In [DMM+19], the authors conjectured that their attack strategy can also be adopted to other schemes and Ascon-AEAD is one of them. We now detail the application of this attack strategy to Ascon-AEAD before presenting our analysis in the next sections.

Following [DMM+19], a bit at the input of the S-box in the second round of the initialization is chosen as the attack point. Note that, in Ascon-AEAD, this is the same as choosing the first round output as the attack point because the step of constant addition in the second round does not change the distribution of the targeted bit. Thus, the attacker can choose one of the following as the intermediate value: $z_0^j, z_1^j, z_2^j, z_3^j, z_4^j$ (see Figure 5.1), where the superscript $j \in [0, 63]$ denotes the j-th bit in a word (the rightmost bit is at index 0). The computation of each of these bits involves three S-box applications in the first round as the effect of the linear diffusion layer.

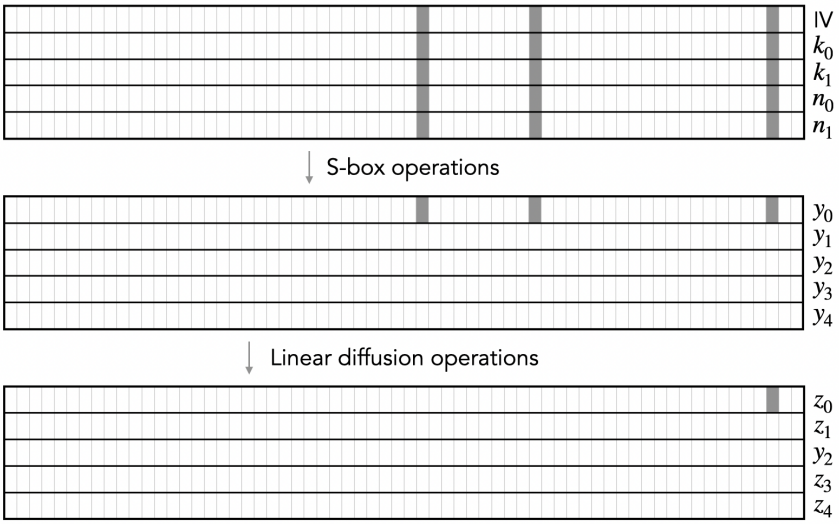


Figure 5.1.: Involved bits in the computation of z_0^j.

We provide hereafter a detailed analysis for a bit of the first word, z_0^j. A similar analysis is applicable for a bit of other words, $z_1^j, z_2^j, z_3^j, z_4^j$. Figure 5.1 depicts the involved bits in the computation $z_0^j = y_0^j \oplus y_0^{j+19} \oplus y_0^{j+28}$, where y_0^j is computed as the following:

$$y_0^j = n_1^j k_0^j \oplus n_0^j \oplus k_1^j k_0^j \oplus k_1^j \oplus k_0^j \mathtt{IV}^j \oplus k_0^j \oplus \mathtt{IV}^j.$$

Generally, the computation of z_0^j involves the following bits:

- 6 bits of the key, including 3 bits from the first key half $(k_0^j, k_0^{j+19}, k_0^{j+28})$ and 3 bits from the second key half $(k_1^j, k_1^{j+19}, k_1^{j+28})$,
- 6 bits of the nonce, including 3 bits from the first nonce half $(n_0^j, n_0^{j+19}, n_0^{j+28})$ and 3 bits from second nonce half $(n_1^j, n_1^{j+19}, n_1^{j+28})$,

- 3 constant bits of the initialization vector `IV`.

We observe that the key bits only have linear influence (XOR operations) on the computations of z_2^j and z_3^j (see Equation 2.3 and Equation 2.2). As a consequence, the intermediate value (z_2^j or z_3^j) will have the same distribution for all the key candidates in the key recovery (not considering its sign). For this reason, the correct key guess cannot be distinguished if one of these bits is chosen as the attack point.

Meanwhile, the key bits influence the computations of z_0^j, z_1^j and z_4^j in a non-linear manner (there are AND operations between the nonce and the key). Therefore, the correct key and the wrong key guesses can be distinguished if the attacker chooses one of these bits as the attack point. We will thus focus on the computations of z_0^j, z_1^j and z_4^j in our analyzes in the next sections. Note that this is similar to the remarks in our analysis of the selection function for CPA in Chapter 3.

5.3. Ineffective Faults with Instruction Skip

In the context of SIFA, a commonly used method to induce a fault in practice is to skip an instruction. This skipped instruction is usually involved in the S-box computation, as demonstrated in prior works [DEK+18; DEG+18; DMM+19].

In this section, we analyze the feasibility of obtaining an ineffective fault via an instruction skip in a w-bit architecture, where $w \in \{8, 32, 64\}$. More specifically, we consider the skipped instruction to be one of the following operations: `XOR`, `AND` and `NOT`, which are core components of authenticated encryption schemes such as Ascon-AEAD and those based on Keccak permutation. We then validate our analysis with a case study on Ascon-AEAD.

5.3.1. Main Idea

In the following, we analyze the effect of skipping each of `XOR`, `AND` and `NOT` instructions. For each case, we assess the practicality of SIFA by computing the probability that a fault is ineffective.

Skipping an `XOR`

We first generalize the skipped instruction into two scenarios: one where the source registers are different from the destination register, and another where one of the source registers is also the destination register. These scenarios commonly appear in ARM architectures. Additionally, the latter scenario captures the 2-operand instruction format found in architectures such as AVR. The skipped instruction is highlighted in red as follows:

$\mathtt{OP}_0$ $\mathtt{R}_2$ – –	$\mathtt{OP}_0$ $\mathtt{R}_2$ – –
…	…
$\mathtt{XOR}$ $\mathtt{R}_2$ $\mathtt{R}_1$ $\mathtt{R}_0$	$\mathtt{XOR}$ $\mathtt{R}_2$ $\mathtt{R}_2$ $\mathtt{R}_0$
…	…
$\mathtt{OP}_2$ – $\mathtt{R}_2$ –	$\mathtt{OP}_2$ – $\mathtt{R}_2$ –
Scenario 1 : $\mathtt{R}_2 = \mathtt{R}_1 \oplus \mathtt{R}_0$	Scenario 2 : $\mathtt{R}_2 = \mathtt{R}_2 \oplus \mathtt{R}_0$

In the above illustration, $\texttt{OP}_0$ and $\texttt{OP}_2$ represent operations, $\texttt{R}_1$ and $\texttt{R}_2$ denote w-bit registers, and $\texttt{R}_0$ represents either a w-bit register or an immediate value. Registers or immediates that are not relevant to our analysis are denoted by "–" for improved readability.

We focus on the value of the destination register $\texttt{R}_2$ in $\texttt{OP}_0$ and XOR. Let v_2 denote the value of $\texttt{R}_2$ after the execution of $\texttt{OP}_0$. Under normal conditions (i.e., without fault injection), this register is subsequently updated by the execution of XOR, resulting in a new value v_2'. However, if XOR is skipped due a to a fault, the value of $\texttt{R}_2$ remains unchanged at v_2. Consequently, the instruction $\texttt{OP}_2$ operates on a faulty value of $\texttt{R}_2$.

Let v_0 and v_1 denote the value of $\texttt{R}_0$ and $\texttt{R}_1$ before the execution of XOR. We have

$$v_2' = v_1 \oplus v_0$$

in the first scenario and

$$v_2' = v_2 \oplus v_0$$

in the second scenario. We make the following assumptions:

1. v_0, v_1 and v_2 are uniformly distributed and independent. This typically occurs when these values originate from, or are derived from, random inputs (*e.g.*, plaintexts, ciphertexts, nonces), which is usually the case in practical SIFA scenarios. As a result, v_2' is uniformly distributed and independent of v_2.
2. When a fault occurs (*i.e.*, the XOR instruction is skipped), it is considered *ineffective* if $v_2 = v_2'$. Otherwise, the fault is referred to as *effective*.

As v_2 and v_2' are independent and uniformly distributed, the probability of an ineffective fault is:

$$\Pr[v_2 = v_2'] = 2^{-w},$$

where $w \in \{8, 16, 32\}$ denotes register bit-width of the hardware architecture. For 32-bit and 64-bit architectures, $\Pr[v_2 = v_2'] \approx 0$, meaning that it is practically inefficient to obtain an ineffective fault. In contrast, $\Pr[v_2 = v_2'] = 2^{-8}$ for 8-bit architecture, implying that an ineffective fault can be expected once every 2^8 executions with random inputs. This shows that performing SIFA by skipping an XOR instruction is highly inefficient for 32-bit and 64-bit architectures, as the probability of achieving an ineffective fault is too low to be practical.

Skipping an AND

Similar to before, we first generalize the skipped instruction into two scenarios. In the case of an XOR instruction, the analysis is the same for both scenarios. However, for the AND instruction considered here, the analysis differs between the two.

$\texttt{OP}_0$ $\texttt{R}_2$ – –	$\texttt{OP}_0$ $\texttt{R}_2$ – –
...	...
AND $\texttt{R}_2$ $\texttt{R}_1$ $\texttt{R}_0$	AND $\texttt{R}_2$ $\texttt{R}_2$ $\texttt{R}_0$
...	...
$\texttt{OP}_2$ – $\texttt{R}_2$ –	$\texttt{OP}_2$ – $\texttt{R}_2$ –
Scenario 1 : $\texttt{R}_2 = \texttt{R}_1 \wedge \texttt{R}_0$	Scenario 2 : $\texttt{R}_2 = \texttt{R}_2 \wedge \texttt{R}_0$

Using the same notations as before, let v_0, v_1 and v_2 denote the values of registers $\texttt{R}_0$, $\texttt{R}_1$, and $\texttt{R}_2$, respectively, prior to the execution of `AND`. Let v_2' represent the value of $\texttt{R}_2$ after the execution of `AND` under normal conditions (*i.e.*, without fault injection). If the `AND` instruction is skipped, the value of $\texttt{R}_2$ remains unchanged at v_2. We make the following assumptions:

1. v_0, v_1 and v_2 are independent and uniformly distributed.
2. When a fault occurs (*i.e.*, the `AND` instruction is skipped), it is considered *ineffective* if $v_2 = v_2'$. Otherwise, the fault is referred to as *effective*.

Scenario 1. We have $v_2' = v_1 \wedge v_0$. For simplicity, we consider 1-bit values for v_0, v_1, v_2 and v_2'. Table 5.2 presents the truth table of the `AND` operation. Since v_2 is uniformly distributed, it takes the value 0 with probability 0.5 and the value 1 with probability 0.5. When $v_2 = 0$, the probability that the fault is ineffective equals the probability that $v_2' = 0$, which is 0.75. Conversely, when $v_2 = 1$, the probability that the fault is ineffective equals the probability that $v_2' = 1$, which is 0.25. Therefore, the total probability of an ineffective fault is

$$\Pr[v_2 = v_2'] = 0.5 \times 0.75 + 0.5 \times 0.25 = 0.5.$$

When considering a w-bit architecture, the probability of an ineffective fault becomes

$$\Pr[v_2 = v_2'] = 2^{-w}.$$

This result is analogous to skipping an `XOR` instruction. For 32-bit and 64-bit architectures, the probability of achieving an ineffective fault is too low to be practical.

v_0	v_1	v_2'
0	0	0
0	1	0
1	0	0
1	1	1

Table 5.2.: Truth table of `AND` operation for $v_2' = v_1 \wedge v_0$ in scenario 1.

Scenario 2. We have $v_2' = v_2 \wedge v_0$. We consider the 1-bit truth table in Table 5.3 for this equation. It can be seen that the probability of a fault being ineffective is $\Pr[v_2 = v_2'] = 0.75$. When considering a w-bit architecture, this probability becomes

$$\Pr[v_2 = v_2'] = 0.75^w.$$

For $w = 32$ and $w = 64$, the values of $\Pr[v_2 = v_2']$ are approximately 10^{-4} and 10^{-8}, respectively. This means that it is more practical for a fault to be ineffective. Therefore, skipping an instruction of this type makes SIFA more applicable.

Skipping a `NOT`

As before, we first generalize the skipped instruction into two scenarios. We use the same notation for register values as above. We make the following assumptions:

v_0	v_2	v_2'	$v_2 = v_2'$
0	0	0	✓
0	1	0	
1	0	0	✓
1	1	1	✓

Table 5.3.: Truth table of AND operation for $v_2' = v_2 \wedge v_0$ in scenario 2.

1. v_0 and v_2 are independent and uniformly distributed.
2. When a fault occurs (*i.e.*, the NOT instruction is skipped), it is considered *ineffective* if $v_2 = v_2'$. Otherwise, the fault is referred to as *effective*.

```
OP0  R2  –  –          OP0  R2  –  –
     ...                    ...
NOT  R2  R0            NOT  R2  R2
     ...                    ...
OP2  –   R2 –          OP2  –   R2 –
Scenario 1 : R2 = ¬R0  Scenario 2 : R2 = ¬R2
```

Scenario 1. We have $v_2' = \neg v_0$. Since v_0 is uniformly distributed, v_2' is also uniformly distributed. Therefore, the probability of a fault being ineffective is

$$\Pr[v_2 = v_2'] = 2^{-w}.$$

This probability is similar to that of skipping an XOR instruction, meaning that it is highly inefficient to obtain an ineffective fault on 32-bit and 64-bit architectures.

Scenario 2. We have $v_2' = \neg v_2$. In this case, an ineffective fault never occurs,

$$\Pr[v_2 = v_2'] = 0.$$

Therefore, if a NOT instruction corresponding to this scenario is skipped, the application of SIFA becomes infeasible.

5.3.2. Application to Ascon-AEAD

To validate the above analysis, we simulate an instruction skip on the 8-bit Ascon-AEAD implementation.[1] We use two instances of the encryption implementation, one as the reference and the other modified to skip an instruction. Specifically, an 8-bit instruction involved in the first round computation during the initialization is skipped. This instruction corresponds to one of the scenarios discussed above. Both instances are executed with the same random nonce (while the associated data and plaintext are set to empty strings, as they are not relevant to the attack). If the outputs produced by both instances are identical, the fault is considered ineffective.

Table 5.4 presents the results of our simulation. As expected, the empirical probabilities are very close to the theoretical ones. This confirms the soundness of our analysis.

[1] The 8-bit implementation can be found at: https://github.com/ascon/ascon-c/tree/main/crypto_aead/asconaead128/bi8

Skipped Instruction	# Ineffective Faults	Empirical Probability	Theoretical Probability
`XOR` S1	81	0.0040	0.0039
`XOR` S2	79	0.0040	0.0039
`AND` S1	80	0.0040	0.0039
`AND` S2	2011	0.1006	0.1001
`NOT` S1	74	0.0037	0.0039
`NOT` S2	0	0	0

Table 5.4.: Empirical results from the instruction skip simulation. The encryption instances were executed 20000 times with random nonces.

5.3.3. Discussion

The key point from our analysis is that, although SIFA is a powerful attack, its applicability depends on both how the fault is introduced and on the architecture of the target device. If an attacker blindly skips an instruction in the hope of causing an ineffective fault, the chances of success can be varied. Especially when skipping an `XOR` instruction in 32-bit and 64-bit architectures, it becomes highly inefficient to obtain an ineffective fault. In such cases, other fault injection techniques may be more efficient. For example, an attacker could use laser fault injection to force one or several bits in a register to be stuck at 0. In this case, only a few bits are affected, rather than the entire 32-bit or 64-bit register, increasing the probability of a fault being ineffective.

In our analysis, we model instruction skip scenarios in authenticated encryption schemes like Ascon-AEAD. However, those scenarios may not cover all practical cases. For example, in the second scenario involving a skipped `NOT` instruction, where $v_2' = \neg v_2$, we argue (under stated assumptions) that an ineffective fault never occurs. Now, suppose that v_2' is used in a subsequent operation, such as $u_c = v_2' \wedge v_0 = \neg v_2 \wedge v_0$. If the `NOT` is skipped, $\mathtt{R}_2$ retains the value v_2, and the operation becomes $u_f = v_2 \wedge v_0$. In this case, an ineffective fault ($u_c = u_f$) occurs with the probability of 0.5 for 1-bit values, as shown in Table 5.5.

v_0	v_2	u_c	u_f	$u_c = u_f$
0	0	0	0	✓
0	1	0	0	✓
1	0	1	0	
1	1	0	1	

Table 5.5.: Truth table of u_c and u_f.

5.4. Uniformity of Intermediate Value

In authenticated encryption schemes such as Ascon-AEAD and those based on Keccak permutation, each round typically consists of two types of operations: one for diffusion and another for non-linearity (S-box). In the attack strategy proposed by

Dobraunig *et al.* [DMM+19], a single bit of the state just before the S-box application in the second initialization round is chosen as the intermediate value. This bit generally results from a diffusion process at the end of the first round or at the beginning of the second round. In other words, the intermediate value is the XOR of several S-box output bits from the first round.

Our key observation is as follows: if these S-box output bits are uniformly distributed and independently computed, and if at least one of the S-box computations is unaffected by the fault (*e.g.*, instruction skip), then the distribution of the intermediate value remains uniform. Consequently, SIFA is not applicable in this scenario.

In this section, we first formalize this property and provide a general proof. Then, we demonstrate its application in the context of Ascon-AEAD.

5.4.1. Main Idea

Let v be the intermediate value, and let $y^0, y^1, \ldots, y^{\ell-1}$ denote ℓ output bits from ℓ S-box computations involved in the XOR computation of v. We write:

$$v = y^0 \oplus y^1 \oplus \ldots \oplus y^{\ell-1}. \tag{5.1}$$

We denote $y^j = f(x_0^j, x_1^j, \ldots, x_{m-1}^j)$ the computation of the S-box output bit y^j, where $j \in [0, \ell-1]$ and $x_0^j, x_1^j, \ldots, x_{m-1}^j$ are m S-box input bits. We make the following assumptions:

1. The S-box computations producing $y^0, y^1, \ldots, y^{\ell-1}$ are independent, which also implies that their input tuples $(x_0^0, x_1^0, \ldots, x_{m-1}^0)$, $(x_0^1, x_1^1, \ldots, x_{m-1}^1)$, …, $(x_0^{\ell-1}, x_1^{\ell-1}, \ldots, x_{m-1}^{\ell-1})$ are independent. This independence is usually the case when the S-box is implemented in bitsliced form, as in Ascon-AEAD.
2. Each y^j, for $0 \leq j \leq \ell-1$, is uniformly distributed.
3. At least one of the S-box computations is unaffected by the fault (*e.g.*, instruction skip). Without loss of generality, we assume that the computation of y^0 is unaffected.

Theorem 5.1. *In the case of ineffective faults, the filtered values of v follow a uniform distribution.*

As a consequence of Theorem 5.1, SIFA is not applicable because the distribution of the intermediate value v remains unbiased. We now proceed to prove this theorem. If there is no filter for ineffective faults, the proof becomes trivial since the uniformity of y^0 (by assumption) directly implies the uniformity of the v. However, SIFA relies exclusively on ineffective faults, meaning that only (filtered) correct values of v are considered. This filtering makes the proof more complicated.

Here is the proof sketch. First, we define a Boolean variable ξ that indicates the effectiveness of the fault: $\xi = 1$ if the fault is effective, and $\xi = 0$ otherwise. Our goal is to show that the distribution of v remains uniform given that the fault is ineffective. Formally, we want to prove the following conditional probability:

$$\Pr[v = \mathsf{a} | \xi = 0] = 0.5, \tag{5.2}$$

where $\mathtt{a} \in \{0, 1\}$. Next, we demonstrate that ξ is independent of y^0. Since y^0 is uniformly distributed by assumption, this independence implies that its uniformity is preserved even when conditioning on $\xi = 0$. Finally, the uniformity of y^0 leads directly to the uniformity of v.

Lemma 5.1. *The variable ξ can be expressed as a Boolean function of the input bits to the S-box producing $y^1, \ldots, y^{\ell-1}$. These input bits are denoted by x_i^j, for $0 \leq i \leq m-1$ and $1 \leq j \leq \ell - 1$.*

Proof of Lemma 5.1. We construct a truth table for $y^1, \ldots, y^{\ell-1}$, considering all possible values of the inputs x_i^j, for $0 \leq i \leq m-1$ and $1 \leq j \leq \ell - 1$. We use two copies of this table: the first serves as a reference, while in the second, we re-evaluate the values of $y^1, \ldots, y^{\ell-1}$, taking the fault effect into account. By comparing the two tables, we identify input values where the fault is ineffective. Example 5.1 provides a demonstration of this process.

We then construct the truth table for ξ, considering all possible values of x_i^j, for $0 \leq i \leq m-1$ and $1 \leq j \leq \ell - 1$. If an input results in an ineffective fault, we set $\xi = 0$; otherwise, $\xi = 1$. From this truth table, the Boolean function representing ξ can be derived. □

Example 5.1. *We consider the function $y^1 = x_0^1 \oplus x_1^1 x_2^1$ ($\ell = 2$, $m = 3$). Let registers $\mathtt{R_0}$, $\mathtt{R_1}$, and $\mathtt{R_2}$ store the values x_0^1, x_1^1, and x_2^1, respectively. The value y^1 is computed using the following instructions:*

```
AND  R1  R1  R2
XOR  R0  R0  R1
```

If the `AND` *instruction is skipped, the computation becomes $\tilde{y}^1 = x_0^1 \oplus x_1^1$. The fault is consider ineffective if $y^1 = \tilde{y}^1$, as shown in Table 5.6.*

x_0^1	x_1^1	x_2^1	y^1	$\tilde{y}^1$	$y^1 = \tilde{y}^1$	ξ
0	0	0	0	0	✓	0
0	0	1	0	0	✓	0
0	1	0	0	1		1
0	1	1	1	1	✓	0
1	0	0	1	1	✓	0
1	0	1	1	1	✓	0
1	1	0	1	0		1
1	1	1	0	0	✓	0

Table 5.6.: Example of truth table construction for ξ.

Corollary 5.1. *The variable ξ is independent of the S-box output bit y^0.*

It is evident that ξ is independent of y^0, since the Boolean function representing ξ does not depend on the input bits of y^0, and the computation of y^0 is independent of those of $y^1, \ldots, y^{\ell-1}$.

Proof of Theorem 5.1. Let $\mathtt{a} \in \{0,1\}$. By assumption, $\Pr[y^0 = \mathtt{a}] = 1/2$. Since ξ and y^0 are independent (Corollary 5.1), it follows that

$$\Pr[y^0 = \mathtt{a} | \xi = 0] = \Pr[y^0 = \mathtt{a}] = 0.5.$$

This means that y^0 remains uniformly distributed even when conditioned on the fault being ineffective. Consequently, the uniformity of v directly follows from the uniformity of y^0. □

5.4.2. Application to Ascon-AEAD

In the context of SIFA on Ascon-AEAD, following the attack strategy proposed by Dobraunig *et al.* [DMM+19], a bit in the first round output of the initialization is chosen as the intermediate value (also corresponding to a bit in the state before the application of the S-box in the second round). As discussed in Section 5.2, we focus only on the bits z_0^j, z_1^j and z_4^j, as the key influences these bits in a non-linear manner. We now show that the intermediate value (z_0^j, z_1^j or z_4^j) has uniform distribution even when conditioned on ineffective faults. As a consequence, SIFA is not applicable since the distribution of the intermediate value is unbiased. Our analysis considers an instruction skip as the fault in an 8-bit implementation.

To prove the uniformity of the intermediate value, we show that the 8-bit implementation of Ascon-AEAD satisfies the three assumptions for Theorem 5.1. We present here a detailed analysis for z_0^j, a similar analysis can be straightforwardly applied to z_1^j and z_4^j. Recall that the output bits y_0^j, y_0^{j+19} and y_0^{j+28}, each resulting from separate S-box computations, are involved in the XOR computation of z_0^j, as presented in Section 5.2. Thus, Equation 5.1 becomes

$$z_0^j = y_0^j \oplus y_0^{j+19} \oplus y_0^{j+28}.$$

First, it is straightforward to observe that the S-box computations producing y_0^j, y_0^{j+19} and y_0^{j+28} are independent, due to the bitsliced design of Ascon-AEAD. Hence, the first assumption is satisfied.

Second, we show that each of y_0^j, y_0^{j+19} and y_0^{j+28} is uniformly distributed under fault-free conditions. As the S-box computation is identical for these three bits, it suffices to examine the distribution of one of them, say y_0^j. Recall that the computation of y_0^j is given by:

$$y_0^j = n_1^j k_0^j \oplus n_0^j \oplus k_1^j k_0^j \oplus k_1^j \oplus k_0^j \mathtt{IV}^j \oplus k_0^j \oplus \mathtt{IV}^j$$

Note that in a SIFA attack, the key is fixed on the device. Therefore, we must verify that, for each fixed key (k_0^j, k_1^j) and for each constant bit of the initialization vector ($\mathtt{IV}^j$), the distribution of y_0^j over all possible values of the nonce bits (n_0^j, n_1^j) is uniform. When $\mathtt{IV}^j = 0$, the evaluation of y_0^j simplifies to:

$$y_0^j = \begin{cases} n_0^j & \text{when } (k_0^j, k_1^j) = (0,0), \\ n_0^j \oplus 1 & \text{when } (k_0^j, k_1^j) = (0,1), \\ n_1^j \oplus n_0^j \oplus 1 & \text{when } (k_0^j, k_1^j) = (1,0), \\ n_1^j \oplus n_0^j \oplus 1 & \text{when } (k_0^j, k_1^j) = (1,1), \end{cases}$$

and when $\texttt{IV}^j = 1$, it becomes:

$$y_0^j = \begin{cases} n_0^j \oplus 1 & \text{when } (k_0^j, k_1^j) = (0,0), \\ n_0^j & \text{when } (k_0^j, k_1^j) = (0,1), \\ n_1^j \oplus n_0^j & \text{when } (k_0^j, k_1^j) = (1,0), \\ n_1^j \oplus n_0^j \oplus 1 & \text{when } (k_0^j, k_1^j) = (1,1), \end{cases}$$

Since n_0^j and n_1^j are uniformly distributed by assumption, it follows that y_0^j is also uniformly distributed. Hence, the second assumption is satisfied.

Third, we show that at least one of the computations of y_0^j, y_0^{j+19} and y_0^{j+28} is unaffected by an instruction skip (*i.e.*, the fault). Recall that we exclusively focus on the 8-bit implementation. To establish this, we demonstrate that the bitsliced S-box computation never processes the data involved in the computations of y_0^j, y_0^{j+19} and y_0^{j+28} within a single instruction. In other words, the bits at the three indices $(j, j+19, j+28)$ of a word never appear in the same 8-bit block (*i.e.*, byte or register) in the 8-bit implementation. We verify this for two practical bit arrangements used in Ascon-AEAD, with and without the interleaving technique [BDP+11]. These arrangements are as shown in Table 5.7 and Table 5.8. Figure 5.2 illustrates the 8-bit blocks in Ascon-AEAD's state when the interleaving technique is not applied. One can verify that, for every $j \in [0, 63]$, the three indices never fall within the same 8-bit block. As a result, skipping an instruction does not simultaneously affect all three computations of y_0^j, y_0^{j+19} and y_0^{j+28}. Thus, the third assumption is satisfied.

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

Table 5.7.: Arrangement of 64-bit word into bytes. The positions corresponding to indices 0, 19 and 28 are highlighted in red.

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

Table 5.8.: Arrangement of 64-bit word into bytes using the interleaving technique. The positions corresponding to indices 0, 19 and 28 are highlighted in red.

To empirically validate our analysis, we compute the probabilities of the intermediate value z_0^j taking values 0 and 1, based on the ineffective faults simulated in the

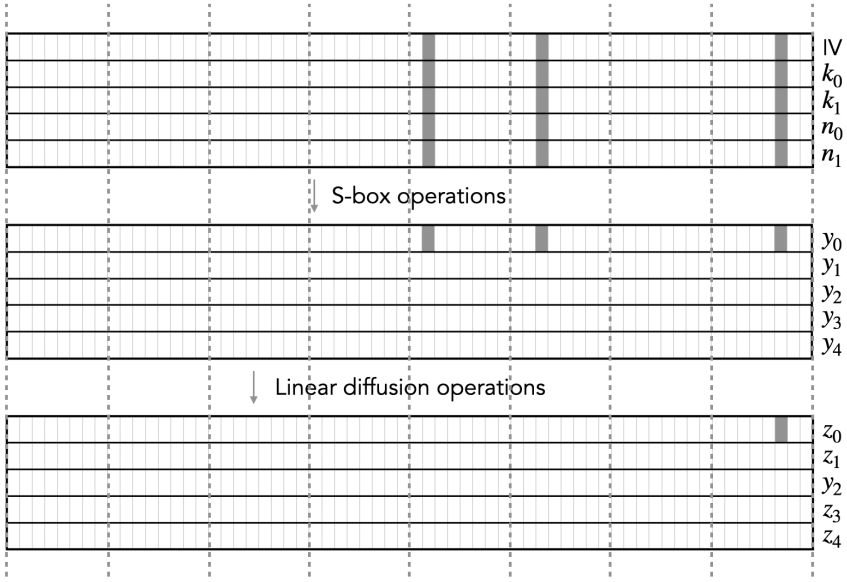


Figure 5.2.: 8-bit blocks in the ASCON-AEAD's states (without using the interleaving technique). Bits involved in the computations of y_0^j, y_0^{j+19}, y_0^{j+28} and z_0^j are highlighted in gray.

previous section. The results are shown in Figure 5.3. As observed, the empirical probabilities gradually converge to a uniform value of 0.5 as the number of ineffective faults increases.

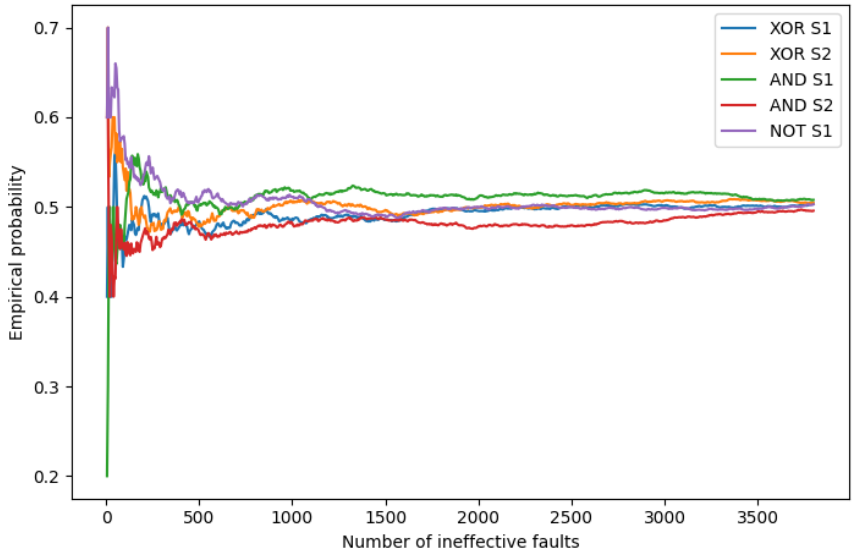


Figure 5.3.: $\Pr[z_0^j = 1 | \xi = 0]$ with increasing number of ineffective faults.

5.5. Conclusion

In this chapter, we provide a more in-depth analysis of the SIFA-based attack strategy on authenticated encryption schemes proposed by Dobraunig *et al.* [DMM+19], with a particular focus on instruction skip as the fault method. We show that the probability of achieving an ineffective fault depends on the type of instruction and the device architecture. Notably, skipping an `XOR` instruction in a 32-bit and 64-bit architectures is unlikely to result in an ineffective fault. Furthermore, we formally show that the intermediate value at the attack point can unexpectedly remain unbiased under ineffective faults. Both findings highlight the cases where SIFA becomes inapplicable.

One limitation of our approach is that the modeling of instruction skip scenarios in Section 5.3 may not be exhaustive. In practice, other instructions, such as `OR` or `BIC` on ARM architectures, could also be used. Another limitation is that we focus only on SIFA failures with the attack strategy proposed by [DMM+19]. A modification of this strategy might lead to a successful attack. For example, Figure 5.2 suggests that targeting an intermediate value at the S-box output could be a promising alternative. We leave the exploration of such cases for future work.

In summary, this chapter presents a novel perspective on SIFA when applied to nonce-based authenticated encryption schemes. In the next chapter, we investigate another prominent type of FA, with a focus on AES.

6

Persistent Fault Attacks on AES with Instruction Skip

Contents

In this chapter, we present the contribution from our study on the threat of Fault Attacks (FA) against the Advanced Encryption Standard (AES). We focus on another prominent type of FA, namely Persistent Fault Analysis (PFA). Our study is restricted to the 128-bit key variant; hereafter, when we refer to AES, we mean this variant.

6.1. Context and Motivation

In the literature, faults are typically categorized according to the duration of their effect, namely into *transient faults*, *persistent faults*, and *permanent faults*. A transient fault affects the execution in a very short period, typically during a single execution.

This means that a transient fault causes errors in only one execution and does not persist in subsequent executions. A permanent fault, on the other hand, has a lasting effect on the target and cannot be erased. Persistent faults, on which we focus in this chapter, fall between the other two categories. A fault of this type persists across different executions but is erased once the device is reset.

The concept of persistent faults was introduced by Schmidt *et al.* [SHP09] with an application to attacking AES. Recently, Persistent Fault Analysis (PFA) has received significant attention since the work of Zhang *et al.* [ZLZ+18]. In their work, the authors developed a dedicated model for the persistent fault setting and proposed a technique for key recovery in block ciphers, with an application to AES. The model assumes that the ciphers are implemented with a lookup table for the S-box and that the faults affect one or multiple S-box elements stored in memory (*e.g.*, ROM). Specifically, the faults result in a *biased faulty S-box*, where one or several S-box elements appear more than once while one or several others disappear. These faults persist across different encryptions until the device is reset. Building on Zhang *et al.*'s model, many follow-up works have either aimed to reduce the number of required ciphertexts by more advanced analyses [CGR20; XZY+21; ZZJ+20; SBH+22; ZFL+22; ZHF+23] or to apply this model to attack different (protected) ciphers [PZR+19; GPT19; TL22; ZFL+22]. Among the previous works, only a few focus on fault injection experiments [ZLZ+18; SHP09; ZZJ+20; SBH+22; GTB+24], while the others primarily focus on analysis based on assumptions about the faults.

We make the first key observation: existing PFA are all based on the assumption that the S-box is implemented as a lookup table stored in memory, with one or more elements being faulted (as modeled by Zhang *et al.* [ZLZ+18]). In practice, this table is typically stored statically in FLASH/ROM and then copied to RAM in runtime. Some works target inducing faults directly in FLASH [GTB+24] or ROM [SHP09], while others focus on inducing faults in SRAM [ZZJ+20], DRAM [ZLZ+18], or during the table transfer from FLASH to RAM [SBH+22]. However, a static S-box table stored in FLASH is not always the case in practice. In embedded systems, there is often a question about using FLASH and RAM for table storage. Storing lookup tables in FLASH consumes a certain amount of permanent memory space. To reduce this, tables can be generated on-the-fly. Table generation helps save FLASH space, while the RAM usage remains unchanged, whether the table is copied from FLASH or generated at runtime. For that reason, many embedded systems use the table generation approach. In reality, some embedded cryptographic libraries, such as MbedTLS[1] and cryptlib,[2] offer this strategy for their AES implementations.

Our second key observation is that persistent faults in the literature are primarily induced by bit flips in memory, often achieved through laser-based techniques [ZZJ+20; SHP09; GTB+24]. Laser-based fault injection is highly complex and requires specialized, expensive equipment, costing hundreds of thousands of dollars [BH22]. It also demands high-precision laser pulses with proper intensity and focus, as well as chip decapsulation without causing damage. Another approach for inducing persistent faults is electromagnetic fault injection (EMFI), as demonstrated in [SBH+22]. The experimental setup in [SBH+22] is also quite complex. It requires precise con-

[1] https://github.com/Mbed-TLS/mbedtls, version 3.6.1
[2] https://www.cs.auckland.ac.nz/~pgut001/cryptlib/, version 3.4.8

Reference	Fault target	Fault injection technique
[ZLZ+18]	T-table elements stored in DRAM	Flipping bits by rowhammer
[ZZJ+20]	S-box table stored in SRAM	Flipping bits by laser
[SHP09]	S-box table stored in ROM	Flipping bits by laser
[GTB+24]	S-box table stored in FLASH memory	Flipping bits by laser
[SBH+22]	Transfer of S-box table from FLASH to RAM	Faults during transfer by electromagnetic
This work Section 6.3	S-box table generation	Skipping an instruction by clock glitch
This work Section 6.4	Round constant table generation	Skipping an instruction by clock glitch

Table 6.1.: Comparison between our work with previous PFA on AES.

trol over high-voltage pulses (up to 200V) with very low rise time (less than 4ns) to avoid damaging the device, expertise in EM probe customization, and accurate probe positioning for effective fault injection.

Meanwhile, instruction skip, a simple technique easily achievable with almost any fault injection techniques (*e.g.*, voltage/clock glitch, laser, EM), has not yet been explored for inducing persistent faults in the literature. In addition to being easy to implement, instruction skip can also be achieved with very low-cost equipment, for example, using clock glitch for about 130 dollars or EM for about 10 dollars (see [BH22]). This leads us to the following research question: *Can we perform a PFA attack with an instruction skip (that is easy to insert)?* Targeting faults in S-box elements stored in memory, as done in previous works, may seem infeasible with this approach. However, considering our first key observation about implementations that generate table at runtime, we could instead focus on skipping an instruction during the table generation phase.

Our third key observation is that existing PFA all target to fault the S-box elements. Their analysis phases focus on exploiting the biased distribution of the ciphertexts as the consequence of the biased faulty S-box. This leads us to the second research question: *Are the S-box elements the only constants that the attacker can target to fault in the PFA context?*

In this chapter, we provide affirmative answers to the two research questions above. In particular, we present the following results:

- First, we show that a PFA attack on AES can be performed using an instruction

skip. This is particularly effective against implementations that generate the S-box table at runtime, during the initialization phase, before executing the first AES operation. We showcase an attack on the AES implementation in MbedTLS by using a clock glitch to skip an instruction, resulting in a faulty S-box that can be exploited to recover the key.

- Second, we show that the S-box is not the only target for fault injection in PFA attacks. We propose, to our knowledge, the first PFA that exploits a fault in a constant other than S-box elements. Specifically, we consider a persistent fault induced on a round constant involved in the AES key schedule. We demonstrate that the key can be effectively recovered using a differential fault analysis.

Table 6.1 presents a comparison between this work and the existing PFA in the literature that include practical fault injection experiments. Other studies, such as [CGR20; XZY+21; ZFL+22; ZHF+23], which focus on the analysis phase based on assumptions about the faults, are excluded from this comparison.

For reproducibility, we publish the source code for our experiments and simulations. The experimental code is intended for those with access to the required hardware, while the simulation code can be used by those without the hardware. The code is available at

`https://github.com/nvietsang/pfa-inskip`.

The results presented in this chapter was published in the international journal *IACR Communications in Cryptology (CiC 2025)* [NGC25b].

6.2. Related Works

In this section, we briefly review related works in the literature. We categorize these works into two groups: those addressing the fault analysis phase and those addressing the fault injection phase.

6.2.1. Fault Analysis Phase

At CHES 2018, Zhang *et al.* [ZLZ+18] introduced a model dedicated to persistent faults and a novel analysis known as Persistent Fault Analysis (PFA). In this model, the S-box is assumed to be implemented as a lookup table and stored in memory. A single fault on an S-box element v alters this value to the faulty value $v' \neq v$. Since S-box is a permutation, v no longer appears in the S-box, while v' appears twice as often. Consequently, one value will never be observed in each ciphertext byte, and another value will be observed twice as often. This results in a non-uniform probability distribution for each ciphertext byte as shown in Figure 6.1. If an attacker collects a sufficiently large number of ciphertexts, he can recover the last round key through a statistical analysis. Both the fault value (*i.e.*, $v \oplus v'$) and the fault location (*i.e.*, the location of v) are assumed to be known in this model.

Many follow-up works have been proposed based on the same model introduced by Zhang *et al.* [ZLZ+18]. Carré *et al.* [CGR20] reduced the number of ciphertexts

needed for the analysis of AES by applying maximum likelihood estimation. Pan *et al.* [PZR+19] showed that PFA can break higher-order masking schemes with a single persistent fault and showcased on the masked implementations of the AES and PRESENT ciphers. Note that to apply PFA, they assumed that (part of) the masked S-box computation is realized as a lookup table. Gruber *et al.* [GPT19] applied PFA to the authenticated encryption schemes. Xu *et al.* [XZY+21] enhanced PFA by extending the analysis to deeper middle rounds. Caforio and Banik [CB19] constructed PFA on generic Feistel schemes, with an additional requirement for the model that an attacker can collect both correct and faulty ciphertexts.

At CHES 2020, Zhang *et al.* [ZZJ+20] relaxed the assumption of knowing the fault value and the fault location for the case of a single fault with applications to the AES and PRESENT ciphers. For multiple faults, both [ZLZ+18] and [ZZJ+20] (double faults) presented analysis methods, however, the values and locations of the faults need to be known in both works. This assumption for the case of multiple faults was then relaxed by Engels *et al.* [ESP20] and Soleimany *et al.* [SBH+22] with applications to the AES and LED ciphers. Zheng *et al.* [ZLZ+21] and Zhang *et al.* [ZHF+23] proposed a collision analysis and chosen-plaintext analysis, respectively, which operate under a relatively relaxed model that does not require any information about the fault value, the fault location, or the number of faults.

6.2.2. Fault Injection Phase

Schmidt *et al.* [SHP09] reported that irradiating ultraviolet (UV) for a few minutes can flip bits from 0 to 1 in various types of non-volatile memory (EPROM, EEPROM, FLASH). They also demonstrated a real attack on AES by faulting an S-box element. In 2014, Kim *et al.* [KDK+14] exposed persistent bit flips on DRAM using the rowhammer technique, successfully inducing errors in most DRAM modules from major manufacturers. Using this technique, Zhang *et al.* [ZLZ+18] faulted the AES' T-tables stored in DRAM in their PFA attack. In [ZZJ+20], Zhang *et al.*'s experiment on the SRAM of an ATmega163L microcontroller showed that a single laser pulse can flip two adjacent bits. Soleimany *et al.* [SBH+22] experimented with electromagnetic fault injection (EMFI) on an STM32F407VG microcontroller. The EM pulse more likely affects multiple S-box elements (3-5 elements for the LED S-box and 4-6 elements for the AES S-box). Grandamme *et al.* [GTB+24] demonstrated that it is feasible to inject faults into S-box elements stored in FLASH memory, even when the device is powered off. Selmke *et al.* [SBH+15] demonstrated their experiments of flipping bits at precise locations into 90 nm and 45 nm SRAM cells.

6.3. Attack with a Fault Injected in S-box Generation

In practice, some public embedded cryptographic libraries, such as MbedTLS and cryptlib, support the S-box table generation at runtime. This feature is particularly useful when users aim to save ROM/FLASH usage for statically storing the full table and benefit from faster RAM access. The generation occurs during the initialization phase before the first AES operation. Once generated, the S-box table is stored in RAM and reused across all subsequent AES operations as long as the device is

not reset. This persistent reuse of the S-box introduces a potential vulnerability. If the S-box generation process is faulted, the resulting fault affects multiple AES operations. By collecting sufficient faulty outputs, it becomes feasible to perform a PFA to recover the key.

Since the analysis phase has been well investigated in the literature, *e.g.* [ZLZ+18; ZZJ+20], we refer to these works for the analysis of the key recovery. In this section, we focus on the injection of a fault in the S-box generation. We begin by describing the fault model. Next, we detail the S-box generation process with a concrete C implementation and point out where the fault can be injected. We then present the experimental results of the fault injection via clock glitching.

6.3.1. Fault Model

We consider AES implementations that generate the S-box table at runtime during the initialization phase, before the execution of the first AES encryption. The fault model is summarized as follows:

- The attacker can fault an S-box element by an instruction skip in the S-box table generation during the initialization phase.
- The fault does not need to be precise in either value or location. Knowing that an S-box value has changed is sufficient for the attack (we will later show how to verify that the fault injection is successful).
- The injected fault is persistent, meaning that the affected S-box element remains faulty across encryptions until the device is reset.
- The attacker can collect multiple ciphertexts.

We note that the attack presented in this section is a *ciphertext-only attack*. Therefore, the attacker does not need access to the plaintexts.

6.3.2. S-box Generation

We refer to [DR02] for the mathematical aspect of the S-box. Here, we only focus on the implementation aspect. We use the AES implementation provided by MbedTLS library[3] for our demonstration. Listing 6.1 shows the extracted C code for generating the forward S-box table (`FSb`). Two additional tables (`pow` and `log`) are involved in this process. The i-th S-box element, `FSb`$[i]$ for $1 \leq i \leq 255$, is computed based the two values `log`$[i]$ and `pow`$[255 - $ `log`$[i]]$. The generation of the `pow` and `log` tables is performed with a loop of 256 iterations in line 18. The computation of the `FSb` is then performed with a loop of 255 iterations in line 26, except for the first element `FSb`$[0]$, which is directly assigned in line 25. To obtain a faulty S-box element, one can cause an instruction skip during an iteration in the generation of the `FSb` table,

[3]The source code can be found at https://github.com/Mbed-TLS/mbedtls/blob/71c569d44bf3a8bd53d874c81ee8ac644dd6e9e3/library/aes.c#L375. To use the table generation feature, we need to deactivate the default macros `MBEDTLS_AES_ROM_TABLES`, `MBEDTLS_HAVE_ASM` and `MBEDTLS_AESNI_C` in the configuration file https://github.com/Mbed-TLS/mbedtls/blob/v3.6.1/include/mbedtls/mbedtls_config.h.

or `log` table, or `pow` table.

In addition, Listing 6.1 illustrates the generation of four T-tables (`FT0, FT1, FT2, FT3`) derived from the S-box table `FSb` (line 37). These T-tables are instrumental in accelerating computation. Each 8-bit S-box element corresponds to four 32-bit elements distributed across the T-tables. Consequently, to have an equivalent effect as faulting an S-box element, we need to fault all four corresponding elements in the T-tables. This approach is inefficient because it requires targeting multiple elements. However, the generation of the last three T-tables is based on applying a cyclic shift to the first T-table (lines 46-48). Therefore, by faulting the generation process of the first T-table (lines 38-45), it is possible to achieve an equivalent effect, making the fault injection more efficient.

```
1  #define ROTL8(x) (((x) << 8) & 0xFFFFFFFF) | ((x) >> 24)
2  #define XTIME(x) (((x) << 1) ^ (((x) & 0x80) ? 0x1B : 0x00))
3
4  // Forward S-box & tables
5  static uint8_t  FSb[256];
6  static uint32_t FT0[256];
7  static uint32_t FT1[256];
8  static uint32_t FT2[256];
9  static uint32_t FT3[256];
10
11 static void aes_gen_tables(void){
12   int i;
13   uint8_t x, y, z;
14   uint8_t pow[256];
15   uint8_t log[256];
16
17   // Compute pow and log tables over GF(2^8)
18   for (i = 0, x = 1; i < 256; i++) {
19     pow[i] = x;
20     log[x] = (uint8_t) i;
21     x ^= XTIME(x);
22   }
23
24   // Generate the forward S-box
25   FSb[0x00] = 0x63;
26   for (i = 1; i < 256; i++) {
27     x = pow[255 - log[i]];
28     y  = x; y = (y << 1) | (y >> 7);
29     x ^= y; y = (y << 1) | (y >> 7);
30     x ^= y; y = (y << 1) | (y >> 7);
31     x ^= y; y = (y << 1) | (y >> 7);
32     x ^= y ^ 0x63;
33     FSb[i] = x;
34   }
35
36   // Generate the forward T-tables
37   for (i = 0; i < 256; i++) {
38     x = FSb[i];
39     y = XTIME(x);
40     z = y ^ x;
41
```

```
42      FT0[i] = ((uint32_t) y) ^
43               ((uint32_t) x <<  8) ^
44               ((uint32_t) x << 16) ^
45               ((uint32_t) z << 24);
46      FT1[i] = ROTL8(FT0[i]);
47      FT2[i] = ROTL8(FT1[i]);
48      FT3[i] = ROTL8(FT2[i]);
49    }
50  }
```

Listing 6.1: Implementation of S-box generation in C

6.3.3. Experiment

We use the ChipWhisperer to perform the fault injection via a clock glitch. An additional fast clock cycle is inserted between two ordinary clock cycles during the execution. The width of the induced clock is chosen such that it is short enough to disrupt the correct execution of the current instruction but still recognized by the microprocessor. When the next clock edge arrives, the microprocessor starts executing the next instruction, effectively skipping the current one. Our fault targets the generation of the `pow`, `log` and `FSb` tables, as shown in Listing 6.1. Notably, we do not require a precise fault on a specific instruction or a specific value during the table generation. Any fault that causes an error in an S-box element is sufficient for the attack.

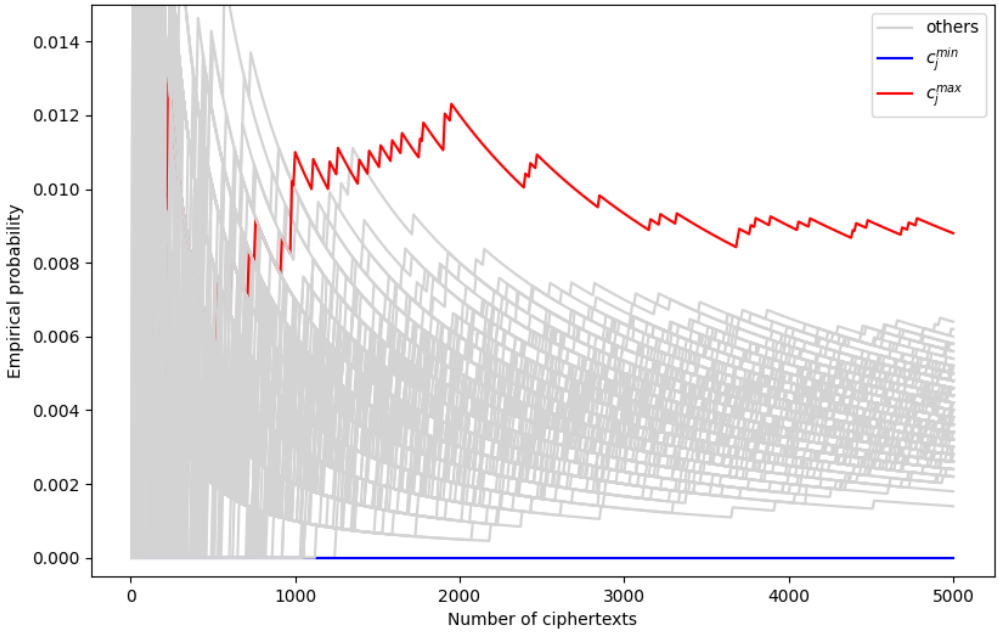


Figure 6.1.: Empirical probability for 256 values of a ciphertext byte.

In this experiment, we use the AES encryption with Electronic Code Book (ECB) mode from the MbedTLS library. The results should be analogous for other modes, as the attack exploits the biased distribution of the ciphertext bytes. After performing the clock glitch, we collect a number of ciphertexts. We now discuss how to verify, using these ciphertexts, whether the fault has been successfully injected. Let us

consider what can be observed when an S-box element is faulted as intended. We compute the empirical probabilities for all 256 possible values of each ciphertext byte. Figure 6.1 shows this for a specific ciphertext byte, denoted as c_j, where $0 \leq j \leq 15$. If an S-box element is erroneous, the distribution of c_j reveals that one value (denoted by $c_j^{\min}$) never appears, while another value (denoted by $c_j^{\max}$) appears twice as often. In practice, if this pattern is not observed across all 16 ciphertext bytes $(c_0, \ldots, c_{15})$ after analyzing a sufficiently large number of ciphertexts, the fault injection was likely unsuccessful. In such cases, the fault injection process should be repeated with different parameters of glitch.

In summary, the attack is automated with the following steps:

1. Perform the clock glitch with chosen values for the parameters (delay, offset and width).
2. Request the device to do encryption N times and collect N ciphertexts.
3. Check if we can observe $(c_j^{\min}, c_j^{\max})$ as described above:
   - If yes, proceed with the key recovery.
   - If no, reset the device, go back to step 1 and try with different parameter values for the clock glitch.

In our experiment using the ChipWhisperer, it takes around 30 minutes identify a glitch configuration that causes the fault as desired. This duration is primarily dominated by the calculation of probabilities to check whether an S-box element is erroneous (step 3). However, we note that this duration depends on the initial parameter values of the glitch. If the initial configuration is already close to the successful ones, the time required will be shorter, otherwise, it may take longer.

We now briefly recall the process of recovering the last round key. For further details and optimizations regarding the required number of ciphertexts, we refer to several related works, such as [ZZJ+20; CGR20; XZY+21].

Let $S[i]$ and $S'[i]$ denote the original S-box element that is faulted and its altered value, respectively. The index i represents the location of the element in the table. The fault value can be expressed as $f = S[i] \oplus S'[i]$. Since $S[i]$ no longer appears in the S-box table and $c_j^{\min}$ is absent in the distribution of c_j, we deduce that $c_j^{\min} = S[i] \oplus k_j$, where k_j $(0 \leq j \leq 15)$ represents the j-th byte of the last round key. Thus, k_j can be recovered using the following equation:

$$k_j = c_j^{\min} \oplus S[i].$$

Note that the fault location i is unknown to the attacker. However, this can be resolved through brute force by testing all 256 possible locations in the S-box ($i \in [0, 255]$), resulting in 256 candidates for the last round key. We can derive the 256 corresponding master key candidates, as the AES key schedule relies solely on the forward (faulty) S-box table for both key expansion and key reversal. If a correct plaintext-ciphertext pair is available, identifying the correct key becomes straightforward. For ciphertext-only attacks, where such a pair is not accessible, obtaining the unique correct key candidate is a challenge. Zhang *et al.* [ZHF+23] addressed this issue, but in the context of a chosen-plaintext attack (not ciphertext-only).

6.4. Attack with a Fault Injected in a Round Constant

In this section, we show that S-box elements are not the only target for PFA attacks. We demonstrate that, by faulting the 8th round constant in AES, an attacker can recover the key using a Differential Fault Analysis (DFA).

Note this fault attack is akin to the work of Kim [Kim12], which presented a DFA with a fault in the AES key schedule. The main difference is that Kim considered a *transient fault* induced into the first column of the 8th round in the key schedule process, whereas we consider a *persistent fault* induced into the 8th round constant. Our analysis is somewhat simpler since the fault value remains the same across executions due to the nature of a persistent fault. Nonetheless, we can directly apply Kim's analysis to recover the last round key. Therefore, we refer to the original analysis of Kim [Kim12] for the key recovery. In this section, we only focus on the effect of a persistent fault on the 8th round constant, which is the main difference from [Kim12].

6.4.1. Fault Model

As in previous PFA with S-box table based on Zhang *et al.*'s model [ZLZ+18], the round constants are assumed to be stored as a lookup table in memory. The attacker can induce a memory fault in an element of the table, for example, by flipping bits using laser, as in [ZZJ+20; GTB+24].

In this work, we consider AES implementations that generate the tables at runtime, including the table of round constants. We still demonstrate a persistent fault using an instruction skip. The fault model is summarized as follows:

- The attacker can fault the 8th round constant by skipping an instruction during the round constant table generation in the initialization phase.
- The fault value does not need to be known. However, the fault location must be precise, meaning the 8th round constant is specifically targeted. (We will later show how to verify the success of the fault injection.)
- The injected fault is persistent, *i.e.*, the affected round constant remains faulty across multiple encryptions until the device is reset.
- The attacker has access to both plaintexts and ciphertexts, as also assumed in previous PFA [ZHF+23; ZLZ+21].
- The attacker can operate the device in two scenarios: with and without the fault, to collect correct and faulty ciphertexts. Note that this assumption is also used in previous PFA [CB19].

The attack presented in this section is a *differential attack*. The attacker needs to collect several pairs of correct-faulty ciphertexts encrypted with the same plaintexts.

6.4.2. Fault Propagation

We now describe how the differential effects of the fault propagate through the last three rounds of the AES encryption. Let a denote the difference in the output of the associated XOR operation caused by the fault. Figure 6.2 illustrates the propagation of a through the 8th, 9th and 10th round keys. Due to the `SubBytes` transformation, this difference a results in two additional differential values, denoted b and c, in the 9th and 10th round keys. Since the fault is persistent, the values a, b, and c remain the same across all correct-faulty ciphertext pairs.

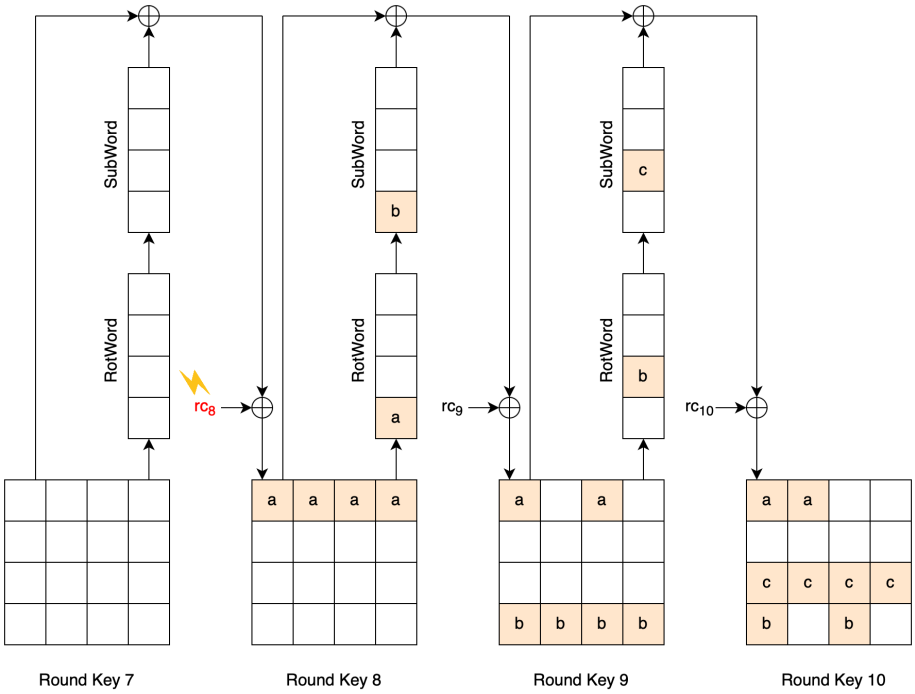


Figure 6.2.: Differential propagation in the key schedule. rc_8 denotes the 8th round constant. This figure omits the XOR operations to recursively generate the 2nd, 3rd and 4th columns of each round key for a succinct illustration.

Figure 6.3 depicts the influence of the differential values a, b and c in the last three rounds of AES encryption. First, a spreads to the first row of the 8th round key, and thus the output state of the `AddRoundKey` in the 8th round (state (5) in Figure 6.3). Next, the `SubBytes` transformation causes the changes in the differential values of the first row from (a, a, a, a) to (e, f, g, h) (state (6) in Figure 6.3). Since `SubBytes` is a non-linear transformation, the differences e, f, g, and h are plaintext-dependent and vary among different correct-faulty ciphertext pairs. Then, the `MixColumns` transformation spreads them to the four cells of its corresponding column. Now, there is a differential relation of the four cells in each column (state (8) in Figure 6.3), *e.g.*, $(2e, e, e, 3e)$ for the first column. The `AddRoundKey` of the 9th round key then adds a to two cells of the first row and b to four cells of the last row (state (9) in Figure 6.3).

Given a pair of correct-faulty ciphertexts, we make hypotheses for bytes of the last round key and for the differential values a, b, and c. Note that we only need to make hypotheses for 2 bytes at a time (see [Kim12] or Appendix B). We then compute

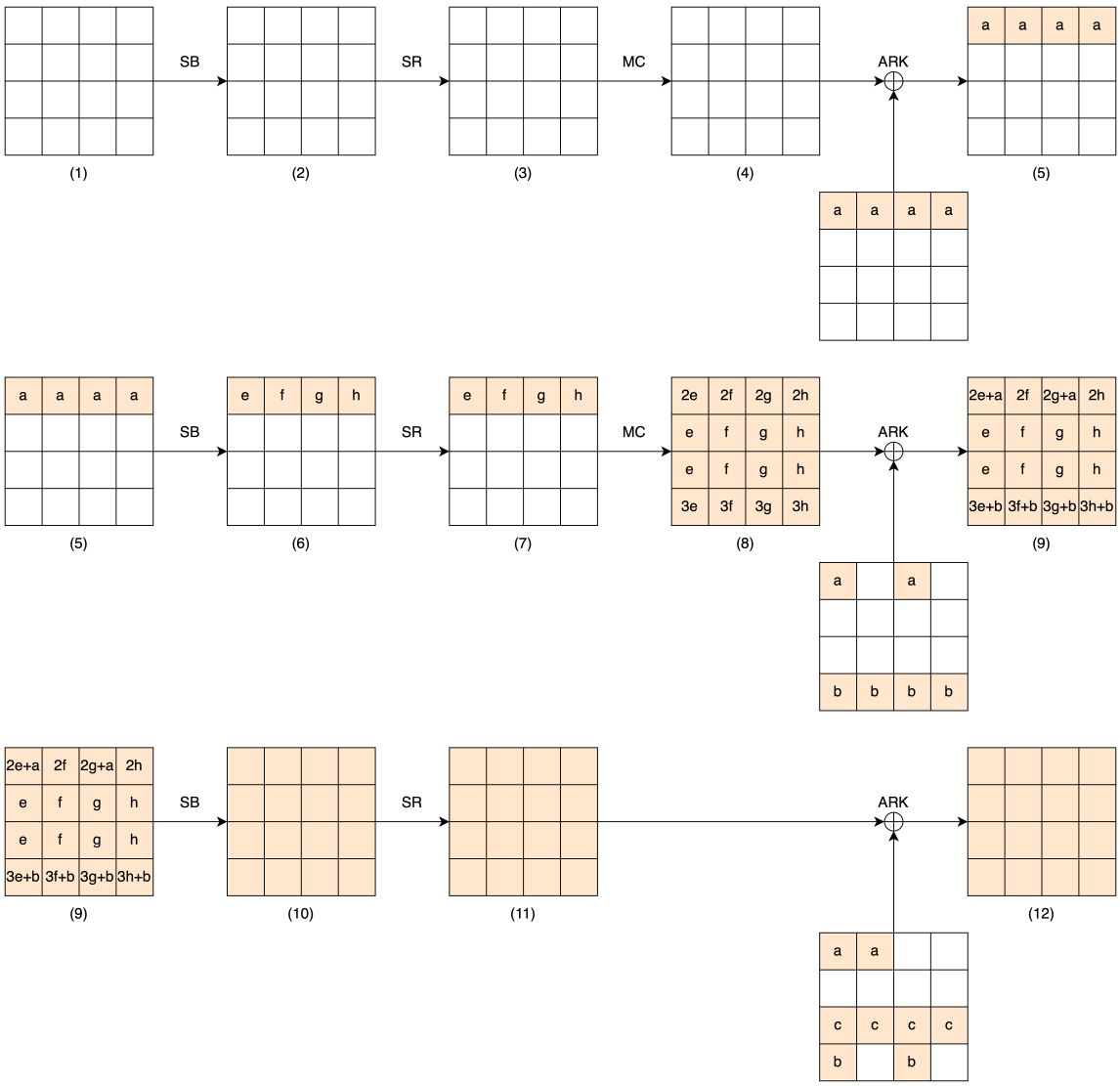


Figure 6.3.: Differential propagation in the last three rounds of AES. SB, SR, MC, and AK denote `SubBytes`, `ShiftRows`, `MixColumns`, and `AddRoundKey` transformations, respectively.

backward to the state at the beginning of the last round (state (9) in Figure 6.3). The correct hypothesis will lead to a match for the differential relation in this state. The analysis algorithm is provided in Appendix B, and a similar algorithm can be found in [Kim12].

According to Kim's analysis [Kim12], using $N = 2$ pairs of correct-faulty ciphertexts can reduce the search space of the key to 2^{24} candidates. However, Kim's approach assumes transient faults where the differential values a, b, and c vary for each ciphertext pair. In contrast, we exploit a persistent fault where the differential values a, b, and c remain consistent for all N pairs. Our simulation shows that with $N = 3$ pairs, we obtain a single candidate, which is the correct key. With $N = 2$ pairs, we obtain around 20 candidates. The correct key is then identified using a correct plaintext-ciphertext from the set of N pairs.

6.4.3. Experiment

As before, we use the AES implementation from MbedTLS for demonstration. Our goal is to skip an instruction in the generation of the round constants table during the initialization phase, using a clock glitch. Listing 6.2 shows the extracted implementation of the round constant generation from this library.[4] Recall that the target of our fault is the 8th round constant ($i = 7$).

```
1 #define XTIME(x) (((x) << 1) ^ (((x) & 0x80) ? 0x1B : 0x00))
2 static uint32_t round_constants[10];
3
4 for (i = 0, x = 1; i < 10; i++) {
5     round_constants[i] = x;
6     x = XTIME(x);
7 }
```

Listing 6.2: Code of round constant generation in C

Before the fault injection, we encrypt three chosen (possibly random) plaintexts and collect their corresponding correct ciphertexts. The device is then reset. A clock glitch injection is performed during the initialization phase by inserting an additional fast clock cycle to skip a proper instruction. We subsequently encrypt the same three plaintexts again and collect their corresponding faulty ciphertexts. The three correct-faulty ciphertext pairs are then analyzed to recover the last round key. If the analysis does not yield a unique key candidate, the fault injection was likely unsuccessful in altering the targeted round constant. This is also used as the verification to know whether the fault is successful injected as desired.

In summary, the attack is automated with the following steps:

1. Encrypt N chosen (possibly random) plaintexts and collect N corresponding correct ciphertexts. We set $N = 3$ so that the key recovery returns a single candidate for the correct key.
2. Reset the device.
3. Perform the clock glitch with chosen values for the parameters (delay, offset and width).
4. Encrypt the same N plaintexts and collect N corresponding ciphertexts.
5. Check if we can obtain a unique key candidate when performing the key recovery on the N pairs of ciphertexts.
    - If yes, perform the inverse key schedule and return the recovered master key.
    - If no, go back to step 2 and try with different parameter values for the clock glitch.

[4] The source code can be found at https://github.com/Mbed-TLS/mbedtls/blob/71c569d44bf3a8bd53d874c81ee8ac644dd6e9e3/library/aes.c#L394

In our experiment using the ChipWhisperer, it typically takes around 3 minutes to find a suitable configuration for the glitch parameters (delay, offset and width) that results in a successful fault injection. Note that this duration depends on the initial configuration's proximity to the successful configuration. If the initial parameters are close to the optimal values, the time to achieve a successful fault injection is shorter. However, if the starting configuration is far from the optimal settings, it may take longer to achieve the desired outcome.

6.4.4. Discussion

This analysis reduces the PFA based on statistical analyses from previous works (*e.g.*, [ZLZ+18; ZZJ+20; SBH+22]), which require hundreds to thousands of ciphertexts, to a DFA needing only 3 pairs of correct-faulty ciphertexts. This approach is much more efficient in terms of data complexity for the key recovery.

A limitation of this attack is that DFA does not work in the ciphertext-only context. Our attack assumes that the attacker can encrypt the same plaintext twice, once with the fault and once without, to collect a pair of correct-faulty ciphertexts. In practice, the attacker can first collect the correct ciphertexts, then reset the device, perform the fault injection and collect the faulty ciphertexts. This approach is feasible if the implementation allows re-initialization, where the tables are freed and then recreated in memory. Some configurations of the MbedTLS library allow this. However, in configurations where the initialization occurs only once, the attacker cannot collect the both correct and faulty ciphertexts. This difficulty in switching between faulty and non-faulty modes might restrict the applicability of DFA.

Another advantage of this DFA attack is its applicability to implementations that do not use a lookup table for the S-box, as it targets a round constant instead. It is worth noting that the assumption of a table-based implementation is common in prior works (*e.g.*, [ZLZ+18; ZZJ+20; SBH+22; ZHF+23]). However, such implementations are known to pose significant risks of side-channel attacks when the CPU relies on caches [Ber05]. This vulnerability has led to the development of modern ciphers, such as Ascon [DEM+21]. Recently selected by NIST as the lightweight cipher standard, Ascon is designed to support efficient bitsliced implementations, avoiding reliance on a lookup table for S-box operations. For AES, there is also bitsliced implementation that does not use S-box lookup tables [RSD06].

6.5. Countermeasures

In this section, we discuss the applicability of common countermeasures against fault attacks in the PFA context. These countermeasures include Error Correction Codes in memory and Dual Modular Redundancy.

6.5.1. Error Correction Codes

For systems that use static tables of constants (S-box elements, round constants), Error Correction Codes (ECC) can be an effective method for detecting and cor-

recting faults. In this method, redundant bits are stored alongside the data in the memory. These bits are typically derived from a linear relationship with the data, enabling the detection and correction of errors. When the data is read, the memory controller verifies this relationship and corrects any detected errors before passing the data to the processor. As a result, a fault persists only until the affected element is accessed in ECC-protected memory. In turn, no faulty ciphertexts are produced, making statistical analyses relying on faulty ciphertexts infeasible.

6.5.2. Dual Modular Redundancy

ECC may not be useful for systems that dynamically generate the tables at runtime, as the fault injection targets the generation process rather than the memory. In such cases, we consider a popular Dual Modular Redundancy (DMR) countermeasure, where a ciphertext C is obtained by encrypting a plaintext P, then decrypted to yield P' for comparison with P. A fault is detected if the decrypted result does not match the original plaintext, *i.e.*, $P \neq P'$.

First, we consider the differential attack with a faulty round constant, as presented in Section 6.4. The round keys for both encryption and decryption are derived from the master key using the same round constant table, which contains the faulty element. Consequently, the verification $P = P'$ always holds, although the ciphertext C is faulty. Therefore, the countermeasure is not effective in this case.

Second, we consider the statistical attack with a faulty S-box element, as presented in Section 6.3. The AES S-box consists of 256 elements, and an encryption accesses the S-box 160 times (in 10-round version). Note that we exclude the S-box accesses during the key schedule, because both encryption and decryption use the same (faulty) round keys derived from the (faulty) forward S-box table. If the faulty S-box element is accessed only during the key schedule and not during encryption, the fault remains undetected.

Suppose a fault is injected into an element of the forward S-box table, while the inverse S-box used for decryption remains unaffected. There is a chance that the faulty element may not be accessed during encryption. This is because an encryption accesses the S-box only 160 times, while the table has 256 elements. As a result, the ciphertext could remain correct, and the fault is undetected by the DMR.

We now investigate the maximum number of correct ciphertexts, denoted by ℓ, that can be produced before the fault is detected. Suppose that with random plaintexts, each S-box element is accessed with uniform probability. In ℓ encryptions, there are $160 \cdot \ell$ S-box accesses in total. The probability that there exists an S-box element that is never accessed in ℓ encryptions is

$$\left(1 - \frac{1}{256}\right)^{160 \cdot \ell}.$$

The probability p that every S-box element is accessed at least once in ℓ encryptions is therefore

$$p = 1 - \left(1 - \frac{1}{256}\right)^{160 \cdot \ell}.$$

Figure 6.4 shows the probability p corresponding to different numbers of encryptions ℓ. We see that $p \approx 0.98$ when $\ell = 6$ and $p \approx 1.00$ when $\ell = 9$. In the worst case, the fault is detected after 9 encryptions.

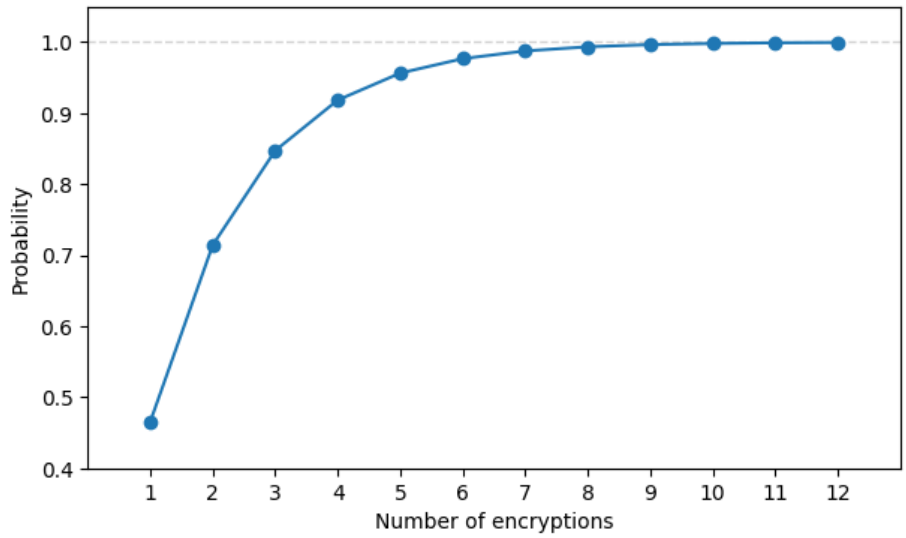


Figure 6.4.: Probability p of different number of encryptions ℓ.

A key aspect of implementing the DMR countermeasure is defining the device's response upon detecting a fault. Zhang *et al.* [ZLZ+18] demonstrated that returning no ciphertext, a zero-value ciphertext, or a random ciphertext upon fault detection does not prevent a PFA attack. Instead, it only introduces noise into the key recovery analysis, increasing the number of ciphertexts required for a successful attack. Note that this assumes the device continues to return correct ciphertexts in subsequent encryptions if no fault is detected during those encryptions. Therefore, aborting the operation and preventing the device from returning ciphertexts from the moment a fault is detected onwards could be a mitigation. In this case, the attacker can collect at most 8 correct ciphertexts (since with $\ell = 9$ encryptions, the fault is detected with a probability approximately equal to 1), which is insufficient for statistical analysis.

6.6. Conclusion

In this chapter, we first demonstrate that a PFA attack on AES can be carried out using a simple instruction skip. We consider implementations that generate the S-box table at runtime before executing the first AES operation. Through an experiment on the ChipWhisperer platform, using the AES implementation in the MbedTLS library, we show that skipping a single instruction via a clock glitch is enough to induce a persistent fault in an S-box element and enable the key recovery.

Second, we introduce the first PFA attack that does not rely on faulting the S-box table. We demonstrate that inducing a fault in a round constant involved in the AES key schedule can enable key recovery through a differential analysis. This approach significantly reduces the data requirements of previous works based on Zhang *et al.*'s model, which typically analyze hundreds to thousands of ciphertexts. In contrast, our differential analysis needs only three correct-faulty ciphertext pairs. Finally, we

provide a discussion on the applicability of common countermeasures against fault attacks in the PFA context.

This chapter also concludes our contributions on fault attacks, marking the end of the main contributions of this thesis. In the next chapter, we provide the overall conclusion and outline perspectives for future work.

7

Conclusion and Perspectives

Contents

In the last chapter, we summarize our research contributions and conclude the thesis. Additionally, we provide some perspectives for future work that can be continued the directions in this thesis.

7.1. Conclusion

This thesis focuses exclusively on physical attacks against symmetric cryptography standards selected by NIST, including AES and Ascon-AEAD. The presented attacks accompanied by experimental results, which demonstrate the practicality of our work.

In the first part, we present the results of our study on the Correlation Power Analysis (CPA) attacks on Ascon-AEAD. Specifically, we provide an analysis on the selection functions used in the literature to explain when and why a failure or success occurs. We then validate our analysis with a second-order CPA targeting a masked software implementation, which also gives the first clue of the computational cost of a second-order CPA attack on Ascon-AEAD. Moreover, we extend the selection function to operate on multiple bits, offering advantages in CPA attacks in terms of number of traces and success rates.

In the second part, we present the results of our study on two prominent types of fault attacks on Ascon-AEAD and AES, namely Statistical Ineffective Fault Analysis (SIFA) and Persistent Fault Analysis (PFA). For SIFA, we analyze the generic attack strategy described in the literature for nonce-based authenticated encryption schemes. We show that, although SIFA is generally considered as a powerful attack, it can fail in some implementations due to the low probability of an ineffective fault and the uniformity of the intermediate value. Our findings are supported by a demonstration on Ascon-AEAD. For PFA, we show that persistent faults can be induced by an instruction skip, which is simpler than the memory faulting techniques

commonly reported in the literature. Additionally, we introduce the first PFA attack targeting a fault in a round constant rather than in S-box elements.

Overall, the results presented in this thesis show that achieving security in embedded cryptography is highly challenging. Moreover, attacks tend to be improved over time. However, the insights into how these attacks operate can guide the design of effective countermeasures and help prevent severe compromises in the future.

7.2. Perspectives

Despite the contributions presented in this thesis, several open questions remain for future research. These questions relate to exploring new directions and improving the efficiency of the attacks.

7.2.1. Correlation Power Analysis on Ascon-AEAD

In Part I, we focus exclusively on the first-round computation to choose a selection function. An interesting direction for future work would be to investigate whether computations in the second round can also be used as selection functions. In this case, the selection function is of degree four and its computation would be more complex. More key bits would be involved in the computation, which could allow more key bits to be recovered in a single CPA run.

In Chapter 3, we propose the use of a SAT solver to find CPA runs in such a way that the overlap between recovered key bits across runs are minimized. This strategy allows full key recovery with the least number of CPA runs. An interesting research idea would be to explore the opposite approach: performing CPA runs with overlapping recovered key bits and compare those overlaps. If the overlapping key bits are consistent, it is likely that they are correct. This approach could potentially improve the success rates while requiring fewer traces. Though, further study is needed to confirm this.

In Chapter 4, we extend the selection function to operate on multiple bits. Another possible direction is to gradually increase the number of bits used to model the hypothetical leakages across successive CPA runs. Specifically, we could begin by a CPA using a 1-bit selection function to recover 3 key bits and select top 4 key candidates. Next, we perform a CPA using a 2-bit selection function, which overlaps the previous 1-bit selection function. This allows us to recover 6 key bits, of which 3 come from the selected candidates of the previous CPA. The process can then be extended with 4-bit, 6-bit, and larger selection functions. The goal is to exploit higher correlation coefficients when modeling with more bits. This idea is similar to the work of Tunstall *et al.* on AES [THM+07].

7.2.2. Statistical Ineffective Fault Analysis on Ascon-AEAD

In Chapter 5, we show that the probability of achieving an ineffective fault with an instruction-skip fault is too low for practical attacks on 32-bit and 64-bit architectures. Note that one assumption in our analysis is that the register data is uniformly

distributed. If this assumption does not hold, the claim no longer applies. This suggests that a chosen-input SIFA attack could be possible. The input here is the data involved in the computation of the intermediate value that the attacker can control. In the case of Ascon-AEAD, this input is the nonce appearing in the first initialization round. This is an interesting direction for future study. In the literature, a chosen-input SIFA on AES is proposed by [EST+24]. However, their goal is to reduce the search space of the key from 2^{32} to 2^{16}. Our goal is instead to obtain higher probability of an ineffective fault on large-bit architectures.

Another finding in Chapter 5 is that, for some nonce-based authenticated encryption implementations, SIFA attack is not applicable due to the uniformity of the intermediate value. For Ascon-AEAD, this holds when a bit at the output of the linear diffusion layer in the first round is chosen as the intermediate value, following the attack strategy in [DMM+19]. A promising research direction is to choose a different location as the intermediate value. One option is an S-box output bit in the first round. In this case, only two key bits are involved in the computation of the intermediate value. However, it is not clear whether skipping any instruction in the S-box computation would cause this new intermediate biased under ineffective faults. This remains to be investigated.

The latter finding also opens a research question: *Can we leverage insights from the uniformity of the intermediate value to design a generic implementation method that mitigates or resists SIFA?* This question relates to sharing the computation so that at least one share is not affected by faults. It connects to the concept of *threshold implementations* by Nikova *et al.* [NRR06], where the computation is split into several *component functions* that are independent of at least one share. This approach was originally proposed to secure digital circuits in the presence of glitches. In our context, we can instead consider faults that occur in a single component function. In this case, at least one share remain unaffected by faults, and the intermediate value stays uniform. A similar idea has been proposed in the literature by Dhooghe *et al.* [DOT24], but their work considers only a single *gate/register* fault. Intuitively, if the attack targets a single component function, the intermediate value should remain uniform even in the presence of multiple faults. Exploring the limits on the number of faults could be an interesting problem for future research.

7.2.3. Persistent Fault Analysis on AES

In Chapter 6, we present a PFA attack based on a biased S-box table, where the bias is caused by fault injection. To our knowledge, all the statistical analysis methods for persistent faults in the literature are also based on this bias. Several countermeasures for PFA therefore aim to detect such biases in the S-box table [TBG23; CB19]. Take these observations into account, a natural research question arises: *What if two S-box elements are swapped by faults?* In that case, those countermeasures can be bypassed, and the statistical analysis methods from the literature would no longer be applicable. The S-box remains a permutation, but becomes effectively "weakened". This could open a new direction for persistent fault attacks. A possible line of analysis could be to exploit the weakness of a non-biased faulty S-box using, for example, linear cryptanalysis.

However, the main difficulty in this direction is inducing faults that swap two S-box elements, since this requires very precise bit flips. Some works suggest that such precision may be realistic. For example, Selmke *et al.* [SBH+15] demonstrate the ability to flip bits at precise locations in 90 nm and 45 nm SRAM cells. Nonetheless, the specific task of swapping two S-box elements by faults has not been widely studied, perhaps because its practical applications are unclear. We hope this open question will motivate further investigation.

Publications and Communications

Below, I list my peer-reviewed publications in international journals and conference proceedings, as well as the presentations and posters I delivered during my doctoral studies.

International Conference Proceedings

[MNR+24] Darius Mercadier, Viet Sang Nguyen, Matthieu Rivain, and Aleksei Udovenko. "OBSCURE: Versatile Software Obfuscation from a Lightweight Secure Element". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2024.2 (2024), pp. 588–629. DOI: 10.46586/tches.v2024.i2.588-629.

[NGC25a] Viet Sang Nguyen, Vincent Grosso, and Pierre-Louis Cayrel. "Correlation Power Analysis on Ascon with Multi-Bit Selection Function". In: *Proceedings of the 22nd International Conference on Security and Cryptography - SECRYPT*. INSTICC. SciTePress, 2025, pp. 72–83. ISBN: 978-989-758-760-3. DOI: 10.5220/0013460000003979.

[NGC25c] Viet Sang Nguyen, Vincent Grosso, and Pierre-Louis Cayrel. "Practical Second-Order CPA Attack on Ascon with Proper Selection Function". In: *Constructive Approaches for Security Analysis and Design of Embedded Systems (CASCADE)*. Springer Nature Switzerland, 2025, pp. 311–339. ISBN: 978-3-032-01405-4. DOI: 10.1007/978-3-032-01405-4_13.

[NGC25d] Viet Sang Nguyen, Vincent Grosso, and Pierre-Louis Cayrel. "SIFA on Nonce-based Authenticated Encryption: When Does It Fail? Application to Ascon". In: *2025 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. (To appear). 2025.

Journal Articles

[NGC25b] Viet Sang Nguyen, Vincent Grosso, and Pierre-Louis Cayrel. "Practical Persistent Fault Attacks on AES with Instruction Skip". In: *IACR Communications in Cryptology* 2.1 (Apr. 8, 2025). ISSN: 3006-5496. DOI: 10.62056/a60l5wol7.

Presentations

1. "SIFA on Nonce-based Authenticated Encryption: When Does It Fail? Application to Ascon", at the 22th Workshop on Fault Detection and Tolerance in Cryptography (FDTC), on 14 September 2025, in Kuala Lumpur, Malaysia.

2. "Correlation Power Analysis on Ascon with Multi-bit Selection Function", at the 22th International Conference on Security and Cryptography (SECRYPT), on 11 June 2025, in Bilbao, Spain.
3. "Practical Second-Order CPA Attacks on Ascon with Proper Selection Function", at the 1st International Conference Constructive Approaches for Security Analysis and Design of Embedded Systems - CASCADE, on 2 April 2025, in Saint-Étienne, France.
4. "Attacks and Countermeasures in Persistent Fault Model", at the 7th *Journée thématique sur les Attaques par Injection de Fautes* (JAIF), on 1 October 2024, in Rennes, France.
5. "Resistance of Threshold Implementations against Statistical (Ineffective) Fault Attacks", at 11th *Journée de la Recherche de l'École doctorale EDSIS*, on 18 June 2024, in Saint-Étienne, France.
6. "Persistent Fault Model: Generalization, Cryptanalysis and Countermeasures", at the 17th *Journées Codage et Cryptographie* (JC2), on 16 October 2023, in Najac, France.

Posters

1. "Attacks and Countermeasures in Persistent Fault Model", at the 7th *Journée thématique sur les Attaques par Injection de Fautes* (JAIF), on 1 October 2024, in Rennes, France.
2. "Resistance of Threshold Implementations against Statistical (Ineffective) Fault Attacks", at 11th *Journée de la Recherche de l'École doctorale EDSIS*, on 18 June 2024, in Saint-Étienne, France.

Bibliography

[77] *Data Encryption Standard*. National Bureau of Standards, NBS FIPS PUB 46, U.S. Department of Commerce. Jan. 1977.

[ABC+17] Stéphanie Anceau et al. "Nanofocused X-Ray Beam to Reprogram Secure Circuits". In: *CHES 2017*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. LNCS. Springer, Cham, Sept. 2017, pp. 175–188. DOI: 10.1007/978-3-319-66787-4_9.

[ABF+03] Christian Aumüller et al. "Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures". In: *CHES 2002*. Ed. by Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar. Vol. 2523. LNCS. Springer, Berlin, Heidelberg, Aug. 2003, pp. 260–275. DOI: 10.1007/3-540-36400-5_20.

[ADG+21] Md Toufiq Hasan Anik et al. "Detecting Failures and Attacks via Digital Sensors". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.7 (2021), pp. 1315–1326. DOI: 10.1109/TCAD.2020.3020921.

[ADM+10] M. Agoyan et al. "How to flip a bit?" In: *2010 IEEE 16th International On-Line Testing Symposium*. IEEE, 2010, pp. 235–239. DOI: 10.1109/IOLTS.2010.5560194.

[BB17] Paul Bottinelli and Joppe W. Bos. "Computational aspects of correlation power analysis". In: *Journal of Cryptographic Engineering* 7.3 (Sept. 2017), pp. 167–181. DOI: 10.1007/s13389-016-0122-9.

[BBB+21] Anubhab Baksi et al. "DEFAULT: Cipher Level Resistance Against Differential Fault Attack". In: *ASIACRYPT 2021, Part II*. Ed. by Mehdi Tibouchi and Huaxiong Wang. Vol. 13091. LNCS. Springer, Cham, Dec. 2021, pp. 124–156. DOI: 10.1007/978-3-030-92075-3_5.

[BBH17] Jakub Breier, Shivam Bhasin, and Wei He. "An electromagnetic fault injection sensor using Hogge phase-detector". In: *2017 18th International Symposium on Quality Electronic Design (ISQED)*. 2017, pp. 307–312. DOI: 10.1109/ISQED.2017.7918333.

[BCN+06] H. Bar-El et al. "The Sorcerer's Apprentice Guide to Fault Attacks". In: *Proceedings of the IEEE* 94.2 (2006), pp. 370–382. DOI: 10.1109/JPROC.2005.862424.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. "Correlation Power Analysis with a Leakage Model". In: *CHES 2004*. Ed. by Marc Joye and Jean-Jacques Quisquater. Vol. 3156. LNCS. Springer, Berlin, Heidelberg, Aug. 2004, pp. 16–29. DOI: 10.1007/978-3-540-28632-5_2.

[BDD+12] Guido Bertoni et al. "Power analysis of hardware implementations protected with secret sharing". In: *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture Workshops*. 2012, pp. 9–16. DOI: 10.1109/MICROW.2012.12.

[BDG16] Alex Biryukov, Daniel Dinu, and Johann Großschädl. "Correlation Power Analysis of Lightweight Block Ciphers: From Theory to Practice". In: *ACNS 2016*. Ed. by Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider. Vol. 9696. LNCS. Springer, Cham, June 2016, pp. 537–557. DOI: 10.1007/978-3-319-39555-5_29.

[BDL01] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. "On the Importance of Eliminating Errors in Cryptographic Computations". In: *Journal of Cryptology* 14.2 (Mar. 2001), pp. 101–119. DOI: 10.1007/s001450010016.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. "On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract)". In: *EUROCRYPT'97*. Ed. by Walter Fumy. Vol. 1233. LNCS. Springer, Berlin, Heidelberg, May 1997, pp. 37–51. DOI: 10.1007/3-540-69053-0_4.

[BDP+11] Guido Bertoni et al. *The Keccak SHA-3 submission*. https://keccak.team/files/Keccak-submission-3.pdf. 2011.

[BDP+12] Guido Bertoni et al. "Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications". In: *SAC 2011*. Ed. by Ali Miri and Serge Vaudenay. Vol. 7118. LNCS. Springer, Berlin, Heidelberg, Aug. 2012, pp. 320–337. DOI: 10.1007/978-3-642-28496-0_19.

[BDP+16a] Guido Bertoni et al. *CAESAR submission: Ketje v2*. https://keccak.team/files/Ketjev2-doc2.0.pdf. 2016.

[BDP+16b] Guido Bertoni et al. *CAESAR submission: Keyak v2*. https://keccak.team/files/Keyakv2-doc2.2.pdf. 2016.

[Ber05] Daniel J. Bernstein. *Cache-timing attacks on AES*. 2005. URL: https://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[BGP+11] Lejla Batina et al. "Mutual Information Analysis: a Comprehensive Study". In: *Journal of Cryptology* 24.2 (Apr. 2011), pp. 269–291. DOI: 10.1007/s00145-010-9084-8.

[BH22] Jakub Breier and Xiaolu Hou. "How Practical Are Fault Injection Attacks, Really?" In: *IEEE Access* 10 (2022), pp. 113122–113130. DOI: 10.1109/ACCESS.2022.3217212.

[BJC15] Jakub Breier, Dau Jap, and Chong-Ning Chen. "Laser Profiling for the Back-Side Fault Attacks: With a Practical Laser Skip Instruction Attack on AES". In: *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*. New York, NY, USA: ACM, 2015, pp. 99–103. ISBN: 978-1-4503-3448-1. DOI: 10.1145/2732198.2732208.

[BKH+20] J. Breier et al. "A Countermeasure Against Statistical Ineffective Fault Analysis". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 67.12 (2020), pp. 3322–3326. DOI: 10.1109/TCSII.2020.2989184.

[BLM+19] Christof Beierle et al. "CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks". In: *IACR Trans. Symm. Cryptol.* 2019.1 (2019), pp. 5–45. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i1.5-45.

[BMS12] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. "A Differential Fault Attack on the Grain Family of Stream Ciphers". In: *CHES 2012*. Ed. by Emmanuel Prouff and Patrick Schaumont. Vol. 7428. LNCS. Springer, Berlin, Heidelberg, Sept. 2012, pp. 122–139. DOI: 10.1007/978-3-642-33027-8_8.

[BS97] Eli Biham and Adi Shamir. "Differential Fault Analysis of Secret Key Cryptosystems". In: *CRYPTO'97*. Ed. by Burton S. Kaliski Jr. Vol. 1294. LNCS. Springer, Berlin, Heidelberg, Aug. 1997, pp. 513–525. DOI: 10.1007/BFb0052259.

[CB19] Andrea Caforio and Subhadeep Banik. "A Study of Persistent Fault Analysis". In: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by Shivam Bhasin, Avi Mendelson, and Mridul Nandi. Cham: Springer International Publishing, 2019, pp. 13–33.

[Cen14] Center for Embedded Systems (CES). *Safer In-Vehicle Control Systems and Medical Devices*. https://faculty.washington.edu/scottcs/NSF/2014/CES.pdf. 2014 Compendium of Industry-Nominated NSF I/UCRC Technological Breakthroughs. 2014.

[CGR20] Sébastien Carré, Sylvain Guilley, and Olivier Rioul. "Persistent Fault Analysis with Few Encryptions". In: *COSADE 2020*. Ed. by Guido Marco Bertoni and Francesco Regazzoni. Vol. 12244. LNCS. Springer, Cham, Apr. 2020, pp. 3–24. DOI: 10.1007/978-3-030-68773-1_1.

[CJR+99] Suresh Chari et al. "Towards Sound Approaches to Counteract Power-Analysis Attacks". In: *CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, Berlin, Heidelberg, Aug. 1999, pp. 398–412. DOI: 10.1007/3-540-48405-1_26.

[CKN01] Jean-Sébastien Coron, Paul C. Kocher, and David Naccache. "Statistics and Secret Leakage". In: *FC 2000*. Ed. by Yair Frankel. Vol. 1962. LNCS. Springer, Berlin, Heidelberg, Feb. 2001, pp. 157–173. DOI: 10.1007/3-540-45472-1_12.

[Cla07] Christophe Clavier. "Secret External Encodings Do Not Prevent Transient Fault Analysis". In: *CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Vol. 4727. LNCS. Springer, Berlin, Heidelberg, Sept. 2007, pp. 181–194. DOI: 10.1007/978-3-540-74735-2_13.

[CPH+21] Ludovic Claudepierre et al. "TRAITOR: A Low-Cost Evaluation Platform for Multifault Injection". In: *Proceedings of the 2021 International Symposium on Advanced Security on Software and Systems (ASSS '21)*. Virtual Event, Hong Kong. 2021, pp. 51–56. DOI: 10.1145/3457340.3458303.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. "Template Attacks". In: *CHES 2002*. Ed. by Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar. Vol. 2523. LNCS. Springer, Berlin, Heidelberg, Aug. 2003, pp. 13–28. DOI: 10.1007/3-540-36400-5_3.

[DEG+18] Christoph Dobraunig et al. “Statistical Ineffective Fault Attacks on Masked AES with Fault Countermeasures”. In: *ASIACRYPT 2018, Part II*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11273. LNCS. Springer, Cham, Dec. 2018, pp. 315–342. DOI: 10.1007/978-3-030-03329-3_11.

[DEK+16] Christoph Dobraunig et al. “Statistical Fault Attacks on Nonce-Based Authenticated Encryption Schemes”. In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Berlin, Heidelberg, Dec. 2016, pp. 369–395. DOI: 10.1007/978-3-662-53887-6_14.

[DEK+18] Christoph Dobraunig et al. “SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography”. In: *IACR TCHES* 2018.3 (2018), pp. 547–572. ISSN: 2569-2925. DOI: 10.13154/tches.v2018.i3.547-572. URL: https://tches.iacr.org/index.php/TCHES/article/view/7286.

[DEM+21] Christoph Dobraunig et al. “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.

[DLV03] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. “Differential Fault Analysis on AES”. In: *ACNS 2003*. Ed. by Jianying Zhou, Moti Yung, and Yongfei Han. Vol. 2846. LNCS. Springer, Berlin, Heidelberg, Oct. 2003, pp. 293–306. DOI: 10.1007/978-3-540-45203-4_23.

[DMM+19] Christoph Dobraunig et al. “Fault Attacks on Nonce-Based Authenticated Encryption: Application to Keyak and Ketje”. In: *SAC 2018*. Ed. by Carlos Cid and Michael J.: Jacobson Jr. Vol. 11349. LNCS. Springer, Cham, Aug. 2019, pp. 257–277. DOI: 10.1007/978-3-030-10970-7_12.

[DO22] Shay Delarea and Yossi Oren. “Practical, Low-Cost Fault Injection Attacks on Personal Smart Devices”. In: *Applied Sciences* 12.1 (2022), p. 417. DOI: 10.3390/app12010417.

[DOT24] Siemen Dhooghe, Artemii Ovchinnikov, and Dilara Toprakhisar. “StaTI: Protecting against Fault Attacks Using Stable Threshold Implementations”. In: *IACR TCHES* 2024.1 (2024), pp. 229–263. DOI: 10.46586/tches.v2024.i1.229-263.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES – The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. ISBN: 978-3-540-42580-9. DOI: 10.1007/978-3-662-04722-4.

[ESP20] Susanne Engels, Falk Schellenberg, and Christof Paar. “SPFA: SFA on Multiple Persistent Faults”. In: *17th Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2020, Milan, Italy, September 13, 2020*. IEEE, 2020, pp. 49–56. DOI: 10.1109/FDTC51366.2020.00014.

[EST+24] Baris Ege et al. “Practical Improvements to Statistical Ineffective Fault Attacks”. In: *COSADE 2024*. Ed. by Romain Wacquez and Naofumi Homma. Vol. 14595. LNCS. Springer, Cham, Apr. 2024, pp. 59–75. DOI: 10.1007/978-3-031-57543-3_4.

[FJL+13a] Thomas Fuhr et al. “Fault Attacks on AES with Faulty Ciphertexts Only”. In: *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. 2013, pp. 108–118. DOI: 10.1109/FDTC.2013.18.

[FJL+13b] Thomas Fuhr et al. “Fault Attacks on AES with Faulty Ciphertexts Only”. In: *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*. Ed. by Wieland Fischer and Jörn-Marc Schmidt. IEEE Computer Society, 2013, pp. 108–118. DOI: 10.1109/FDTC.2013.18.

[Gir05] Christophe Giraud. “DFA on AES”. In: *Advanced Encryption Standard – AES*. Ed. by Hans Dobbertin, Vincent Rijmen, and Aleksandra Sowa. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 27–41. ISBN: 978-3-540-31840-8.

[GJJ+11] Gilbert Goodwill et al. *A testing methodology for side channel resistance validation*. NIST non-invasive attack testing workshop. 2011. URL: https://csrc.nist.rip/news%5C_events/non-invasive-attack-testing-workshop/papers/08%5C_Goodwill.pdf.

[GKK+20] Mael Gay et al. “Error control scheme for malicious and natural faults in cryptographic modules”. In: *Journal of Cryptographic Engineering* 10.4 (Nov. 2020), pp. 321–336. DOI: 10.1007/s13389-020-00234-7.

[GP99] Louis Goubin and Jacques Patarin. “DES and Differential Power Analysis (The “Duplication” Method)”. In: *CHES’99*. Ed. by Çetin Kaya Koç and Christof Paar. Vol. 1717. LNCS. Springer, Berlin, Heidelberg, Aug. 1999, pp. 158–172. DOI: 10.1007/3-540-48059-5_15.

[GPT19] Michael Gruber, Matthias Probst, and Michael Tempelmeier. “Persistent Fault Analysis of OCB, DEOXYS and COLM”. In: *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2019, Atlanta, GA, USA, August 24, 2019*. IEEE, 2019, pp. 17–24. DOI: 10.1109/FDTC.2019.00011.

[GTB+24] Paul Grandamme et al. “Switching Off your Device Does Not Protect Against Fault Attacks”. In: *IACR TCHES* 2024.4 (2024), pp. 425–450. DOI: 10.46586/tches.v2024.i4.425-450.

[GYT+14] Nahid Farhady Ghalaty et al. “Differential Fault Intensity Analysis”. In: *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*. 2014, pp. 49–58. DOI: 10.1109/FDTC.2014.15.

[HBB16] Wei He, Jakub Breier, and Shivam Bhasin. “Cheap and Cheerful: A Low-Cost Digital Sensor for Detecting Laser Fault Injection Attacks”. In: *Security, Privacy, and Applied Cryptography Engineering*. Ed. by Claude Carlet, M. Anwar Hasan, and Vishal Saraswat. Cham: Springer International Publishing, 2016, pp. 27–46. ISBN: 978-3-319-49445-6.

[Hem04] Ludger Hemme. "A Differential Fault Attack Against Early Rounds of (Triple-)DES". In: *CHES 2004*. Ed. by Marc Joye and Jean-Jacques Quisquater. Vol. 3156. LNCS. Springer, Berlin, Heidelberg, Aug. 2004, pp. 254–267. DOI: 10.1007/978-3-540-28632-5_19.

[HR08] Michal Hojsík and Bohuslav Rudolf. "Differential Fault Analysis of Trivium". In: *FSE 2008*. Ed. by Kaisa Nyberg. Vol. 5086. LNCS. Springer, Berlin, Heidelberg, Feb. 2008, pp. 158–172. DOI: 10.1007/978-3-540-71039-4_10.

[JPS05] Marc Joye, Pascal Paillier, and Berry Schoenmakers. "On Second-Order Differential Power Analysis". In: *CHES 2005*. Ed. by Josyula R. Rao and Berk Sunar. Vol. 3659. LNCS. Springer, Berlin, Heidelberg, Aug. 2005, pp. 293–308. DOI: 10.1007/11545262_22.

[JWK04] Nikhil Joshi, Kaijie Wu, and Ramesh Karri. "Concurrent Error Detection Schemes for Involution Ciphers". In: *CHES 2004*. Ed. by Marc Joye and Jean-Jacques Quisquater. Vol. 3156. LNCS. Springer, Berlin, Heidelberg, Aug. 2004, pp. 400–412. DOI: 10.1007/978-3-540-28632-5_29.

[KDK+14] Yoongu Kim et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 2014, pp. 361–372. DOI: 10.1109/ISCA.2014.6853210.

[Kim12] Chong Hee Kim. "Improved Differential Fault Analysis on AES Key Schedule". In: *IEEE Transactions on Information Forensics and Security* 7.1 (2012), pp. 41–50. DOI: 10.1109/TIFS.2011.2161289.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. "Differential Power Analysis". In: *CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, Berlin, Heidelberg, Aug. 1999, pp. 388–397. DOI: 10.1007/3-540-48405-1_25.

[Koc96] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: *CRYPTO'96*. Ed. by Neal Koblitz. Vol. 1109. LNCS. Springer, Berlin, Heidelberg, Aug. 1996, pp. 104–113. DOI: 10.1007/3-540-68697-5_9.

[KT23] Yen-Ting Kuo and Atsushi Takayasu. "A Lattice Attack on CRYSTALS-Kyber with Correlation Power Analysis". In: *ICISC 23, Part I*. Ed. by Hwajeong Seo and Suhri Kim. Vol. 14561. LNCS. Springer, Singapore, Nov. 2023, pp. 202–220. DOI: 10.1007/978-981-97-1235-9_11.

[LSG+10] Yang Li et al. "Fault Sensitivity Analysis". In: *CHES 2010*. Ed. by Stefan Mangard and François-Xavier Standaert. Vol. 6225. LNCS. Springer, Berlin, Heidelberg, Aug. 2010, pp. 320–334. DOI: 10.1007/978-3-642-15031-9_22.

[LWL+22] Sinian Luo et al. "An Efficient Soft Analytical Side-Channel Attack on Ascon". In: Dalian, China: Springer-Verlag, 2022. ISBN: 978-3-031-19207-4. DOI: 10.1007/978-3-031-19208-1_32.

[May00] Rita Mayer-Sommer. "Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards". In: *CHES 2000*. Ed. by Çetin Kaya Koç and Christof Paar. Vol. 1965. LNCS. Springer, Berlin, Heidelberg, Aug. 2000, pp. 78–92. DOI: 10.1007/3-540-44499-8_6.

[MBT+17] Kerry McKay et al. *Report on Lightweight Cryptography*. Tech. rep. NISTIR 8114. Includes NIST's motivations and the initial call for submissions of lightweight cryptographic algorithms. Gaithersburg, MD: National Institute of Standards and Technology, 2017. URL: https://doi.org/10.6028/NIST.IR.8114.

[MDH+13] Nicolas Moro et al. "Electromagnetic Fault Injection: Towards a Fault Model on a 32-bit Microcontroller". In: *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. 2013, pp. 77–88. DOI: 10.1109/FDTC.2013.9.

[Mes00] Thomas S. Messerges. "Using Second-Order Power Analysis to Attack DPA Resistant Software". In: *CHES 2000*. Ed. by Çetin Kaya Koç and Christof Paar. Vol. 1965. LNCS. Springer, Berlin, Heidelberg, Aug. 2000, pp. 238–251. DOI: 10.1007/3-540-44499-8_19.

[MK19] Onur Mutlu and Jeremie S. Kim. "RowHammer: A Retrospective". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2019). Early access publication. DOI: 10.1109/TCAD.2019.2915318.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007. ISBN: 978-0-387-30857-9.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. *Breaking Cryptographic Implementations Using Deep Learning Techniques*. Cryptology ePrint Archive, Report 2016/921. 2016. URL: https://eprint.iacr.org/2016/921.

[MV04] David McGrew and John Viega. *The Galois/Counter Mode of Operation (GCM)*. Tech. rep. Submission to the NIST Modes of Operation Process, 2004.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. "Threshold Implementations Against Side-Channel Attacks and Glitches". In: *ICICS 06*. Ed. by Peng Ning, Sihan Qing, and Ninghui Li. Vol. 4307. LNCS. Springer, Berlin, Heidelberg, Dec. 2006, pp. 529–545. DOI: 10.1007/11935308_38.

[OC14] Colin O'Flynn and Zhizhang (David) Chen. "ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research". In: *COSADE 2014*. Ed. by Emmanuel Prouff. Vol. 8622. LNCS. Springer, Cham, Apr. 2014, pp. 243–260. DOI: 10.1007/978-3-319-10175-0_17.

[PQ03] Gilles Piret and Jean-Jacques Quisquater. "A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD". In: *CHES 2003*. Ed. by Colin D. Walter, Çetin Kaya Koç, and Christof Paar. Vol. 2779. LNCS. Springer, Berlin, Heidelberg, Sept. 2003, pp. 77–88. DOI: 10.1007/978-3-540-45238-6_7.

[PRB09] Emmanuel Prouff, Matthieu Rivain, and Regis Bevan. "Statistical Analysis of Second Order Differential Power Analysis". In: *IEEE Transactions on Computers* 58.6 (2009), pp. 799–811. DOI: 10.1109/TC.2009.15.

[PYG+16] Conor Patrick et al. "Lightweight Fault Attack Resistance in Software Using Intra-instruction Redundancy". In: *SAC 2016*. Ed. by Roberto Avanzi and Howard M. Heys. Vol. 10532. LNCS. Springer, Cham, Aug. 2016, pp. 231–244. DOI: 10.1007/978-3-319-69453-5_13.

[PZR+19] Jingyu Pan et al. "One Fault is All it Needs: Breaking Higher-Order Masking with Persistent Fault Analysis". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, pp. 1–6. DOI: 10.23919/DATE.2019.8715260.

[RAD+20] Keyvan Ramezanpour et al. *Active and Passive Side-Channel Key Recovery Attacks on Ascon*. NIST Lightweight Cryptography Workshop. 2020. URL: https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2020/documents/papers/active-passive-recovery-attacks-ascon-lwc2020.pdf.

[RBW+23] Azade Rezaeezade et al. *One for All, All for Ascon: Ensemble-based Deep Learning Side-channel Analysis*. Cryptology ePrint Archive, Report 2023/1922. 2023. URL: https://eprint.iacr.org/2023/1922.

[Riv09] Matthieu Rivain. "Differential Fault Analysis on DES Middle Rounds". In: *CHES 2009*. Ed. by Christophe Clavier and Kris Gaj. Vol. 5747. LNCS. Springer, Berlin, Heidelberg, Sept. 2009, pp. 457–469. DOI: 10.1007/978-3-642-04138-9_32.

[RPD+23] Nathan Roussel et al. "Security Evaluation of a Hybrid CMOS/MRAM Ascon Hardware Implementation". In: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2023, Antwerp, Belgium, April 17-19, 2023*. IEEE, 2023, pp. 1–6. DOI: 10.23919/DATE56975.2023.10137126.

[RSD06] Chester Rebeiro, A. David Selvakumar, and A. S. L. Devi. "Bitslice Implementation of AES". In: *CANS 06*. Ed. by David Pointcheval, Yi Mu, and Kefei Chen. Vol. 4301. LNCS. Springer, Berlin, Heidelberg, Dec. 2006, pp. 203–212. DOI: 10.1007/11935070_14.

[SBD+20] Thierry Simon et al. "Friet: An Authenticated Encryption Scheme with Built-in Fault Detection". In: *EUROCRYPT 2020, Part I*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. LNCS. Springer, Cham, May 2020, pp. 581–611. DOI: 10.1007/978-3-030-45721-1_21.

[SBH+15] Bodo Selmke et al. "Precise Laser Fault Injections into 90 nm and 45 nm SRAM-cells". In: *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*. Ed. by Naofumi Homma and Marcel Medwed. Vol. 9514. Lecture Notes in Computer Science. Springer, 2015, pp. 193–205. DOI: 10.1007/978-3-319-31271-2_12.

[SBH+22] Hadi Soleimany et al. “Practical Multiple Persistent Faults Analysis”. In: *IACR TCHES* 2022.1 (2022), pp. 367–390. DOI: 10.46586/tches.v2022.i1.367-390.

[SBR+20] Sayandeep Saha et al. “Fault Template Attacks on Block Ciphers Exploiting Fault Propagation”. In: *EUROCRYPT 2020, Part I*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. LNCS. Springer, Cham, May 2020, pp. 612–643. DOI: 10.1007/978-3-030-45721-1_22.

[SC16] Dhiman Saha and Dipanwita Roy Chowdhury. “EnCounter: On Breaking the Nonce Barrier in Differential Fault Analysis with a Case-Study on PAEQ”. In: *CHES 2016*. Ed. by Benedikt Gierlichs and Axel Y. Poschmann. Vol. 9813. LNCS. Springer, Berlin, Heidelberg, Aug. 2016, pp. 581–601. DOI: 10.1007/978-3-662-53140-2_28.

[SD17] Niels Samwel and Joan Daemen. “DPA on hardware implementations of Ascon and Keyak”. In: *Proceedings of the Computing Frontiers Conference*. CF’17. Siena, Italy: Association for Computing Machinery, 2017, pp. 415–424. ISBN: 9781450344876. DOI: 10.1145/3075564.3079067.

[SHP09] Jörn-Marc Schmidt, Michael Hutter, and Thomas Plos. “Optical Fault Attacks on AES: A Threat in Violet”. In: *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2009, pp. 13–22. DOI: 10.1109/FDTC.2009.37.

[Sin05] Carsten Sinz. “Towards an Optimal CNF Encoding of Boolean Cardinality Constraints”. In: *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*. Ed. by Peter van Beek. Vol. 3709. Lecture Notes in Computer Science. Springer, 2005, pp. 827–831. ISBN: 3-540-29238-1. DOI: 10.1007/11564751_73.

[SMG16] Tobias Schneider, Amir Moradi, and Tim Güneysu. “ParTI – Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks”. In: *CRYPTO 2016, Part II*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. LNCS. Springer, Berlin, Heidelberg, Aug. 2016, pp. 302–332. DOI: 10.1007/978-3-662-53008-5_11.

[SVO+10] François-Xavier Standaert et al. “The World Is Not Enough: Another Look on Second-Order DPA”. In: *ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Springer, Berlin, Heidelberg, Dec. 2010, pp. 112–129. DOI: 10.1007/978-3-642-17373-8_7.

[TBG23] Pierre-Antoine Tissot, Lilian Bossuet, and Vincent Grosso. “BALoo: First and Efficient Countermeasure Dedicated to Persistent Fault Attacks”. In: *29th International Symposium on On-Line Testing and Robust System Design, IOLTS 2023, Crete, Greece, July 3-5, 2023*. Ed. by Alessandro Savino et al. IEEE, 2023, pp. 1–7. DOI: 10.1109/IOLTS59296.2023.10224871.

[THM+07] Michael Tunstall et al. *Correlation power analysis of large word sizes*. http://www.geocities.ws/mike.tunstall/papers/THMWMM.pdf. 2007. URL: hhttp://www.geocities.ws/mike.tunstall/papers/THMWMM.pdf.

[TL22] Honghui Tang and Qiang Liu. "MPFA: An Efficient Multiple Faults-Based Persistent Fault Analysis Method for Low-Cost FIA". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.9 (2022), pp. 2821–2834. DOI: 10.1109/TCAD.2021.3117512.

[TMK+25] Meltem Sönmez Turan et al. *Ascon-Based Lightweight Cryptography Standards for Constrained Devices: Authenticated Encryption, Hash, and Extendable Output Functions*. Tech. rep. NIST SP 800-232. National Institute of Standards and Technology (NIST), Aug. 2025. DOI: 10.6028/NIST.SP.800-232.ipd.

[TSS17] Adrian Tang, Simha Sethumadhavan, and Salvatore J. Stolfo. "CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC, Canada: USENIX Association, 2017, pp. 1057–1074.

[VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. "Soft Analytical Side-Channel Attacks". In: *ASIACRYPT 2014, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. LNCS. Springer, Berlin, Heidelberg, Dec. 2014, pp. 282–296. DOI: 10.1007/978-3-662-45611-8_15.

[VTM+17] Aurélien Vasselle et al. "Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot". In: *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2017, pp. 41–48. DOI: 10.1109/FDTC.2017.18.

[WGL+24] Huaxin Wang et al. "In-depth Correlation Power Analysis Attacks on a Hardware Implementation of CRYSTALS-Dilithium". In: *Cybersecur.* 7.1 (2024), p. 21. DOI: 10.1186/S42400-024-00209-9.

[WP23] Léo Weissbart and Stjepan Picek. *Lightweight but Not Easy: Side-channel Analysis of the Ascon Authenticated Cipher on a 32-bit Microcontroller*. Cryptology ePrint Archive, Report 2023/1598. 2023. URL: https://eprint.iacr.org/2023/1598.

[WW04] Jason Waddle and David Wagner. "Towards Efficient Second-Order Power Analysis". In: *CHES 2004*. Ed. by Marc Joye and Jean-Jacques Quisquater. Vol. 3156. LNCS. Springer, Berlin, Heidelberg, Aug. 2004, pp. 1–15. DOI: 10.1007/978-3-540-28632-5_1.

[XZY+21] Guorui Xu et al. "Pushing the Limit of PFA: Enhanced Persistent Fault Analysis on Block Ciphers". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.6 (2021), pp. 1102–1116. DOI: 10.1109/TCAD.2020.3048280.

[YKS+23] Shih-Chun You et al. "Low Trace-Count Template Attacks on 32-bit Implementations of ASCON AEAD". In: *IACR TCHES* 2023.4 (2023), pp. 344–366. DOI: 10.46586/tches.v2023.i4.344-366.

[ZFL+22] Fan Zhang et al. "Free Fault Leakages for Deep Exploitation: Algebraic Persistent Fault Analysis on Lightweight Block Ciphers". In: *IACR TCHES* 2022.2 (2022), pp. 289–311. DOI: 10.46586/tches.v2022.i2.289-311.

[ZHF+23] Fan Zhang et al. “Efficient Persistent Fault Analysis with Small Number of Chosen Plaintexts”. In: *IACR TCHES* 2023.2 (2023), pp. 519–542. DOI: 10.46586/tches.v2023.i2.519-542.

[ZLZ+18] Fan Zhang et al. “Persistent Fault Analysis on Block Ciphers”. In: *IACR TCHES* 2018.3 (2018), pp. 150–172. ISSN: 2569-2925. DOI: 10.13154/tches.v2018.i3.150-172. URL: https://tches.iacr.org/index.php/TCHES/article/view/7272.

[ZLZ+21] Shihui Zheng et al. “A Persistent Fault-Based Collision Analysis Against the Advanced Encryption Standard”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.6 (2021), pp. 1117–1129. DOI: 10.1109/TCAD.2021.3049687.

[ZZJ+20] Fan Zhang et al. “Persistent Fault Attack in Practice”. In: *IACR TCHES* 2020.2 (2020), pp. 172–195. ISSN: 2569-2925. DOI: 10.13154/tches.v2020.i2.172-195. URL: https://tches.iacr.org/index.php/TCHES/article/view/8548.

Appendix

Appendix A. Supplementary Analyses

A.1. Derivation of Selection Functions

In Chapter 3, we present the derivation of the selection function of $\tilde{z}_0^j$. This appendix details the derivations of $\tilde{z}_4^j$ and $\tilde{z}_1^j$.

The j-th bit of y_1 and y_4 are computed as:

$$y_1^j = n_0^j(k_1^j \oplus k_0^j \oplus 1) \oplus n_1^j \oplus k_1^j k_0^j \oplus k_1^j \oplus k_0^j \oplus \mathtt{IV}^j,$$
$$y_4^j = n_1^j(k_0^j \oplus 1) \oplus n_0^j \oplus k_0^j \mathtt{IV}^j \oplus k_0^j.$$

In y_1^j, we remove $k_1^j k_0^j \oplus k_1^j \oplus k_0^j \oplus \mathtt{IV}^j$ as they contribute a constant amount to the power consumption. For the same reason, $k_0^j \mathtt{IV}^j \oplus k_0^j$ is removed in y_4^j. The fine-tuned versions of y_1^j and y_4^j, denoted by $\tilde{y}_1^j$ and $\tilde{y}_4^j$, are:

$$\tilde{y}_1^j = n_0^j(k_{01}^j \oplus 1) \oplus n_1^j,$$
$$\tilde{y}_4^j = n_1^j(k_0^j \oplus 1) \oplus n_0^j,$$

where $k_{01}^j = k_0^j \oplus k_1^j$.

Recall the linear operations applied on the y_1 and y_4:

$$z_1 = y_1 \oplus (y_1 \ggg 61) \oplus (y_1 \ggg 39),$$
$$z_4 = y_4 \oplus (y_4 \ggg \ \ 7) \oplus (y_4 \ggg 41).$$

The j-th bit of z_1 and z_4 are thus computed as:

$$z_1^j = y_1^j \oplus y_1^{j+61} \oplus y_1^{j+39},$$
$$z_4^j = y_4^j \oplus y_4^{j+7} \oplus y_4^{j+41}.$$

We then apply the linear operations for $\tilde{y}_1^j$ and $\tilde{y}_4^j$. The fine-tuned versions of z_1^j and z_4^j, denoted by $\tilde{z}_1^j$ and $\tilde{z}_4^j$, are::

$$\begin{aligned}\tilde{z}_1^j = &\left(n_0^j(k_{01}^j \oplus 1) \oplus n_1^j\right) \\ &\oplus \left(n_0^{j+61}(k_{01}^{j+61} \oplus 1) \oplus n_1^{j+61}\right) \\ &\oplus \left(n_0^{j+39}(k_{01}^{j+39} \oplus 1) \oplus n_1^{j+39}\right),\end{aligned}$$

$$\begin{aligned}\tilde{z}_4^j = &\left(n_1^j(k_0^j \oplus 1) \oplus n_0^j\right) \\ &\oplus \left(n_1^{j+7}(k_0^{j+7} \oplus 1) \oplus n_0^{j+7}\right) \\ &\oplus \left(n_1^{j+41}(k_0^{j+41} \oplus 1) \oplus n_0^{j+41}\right).\end{aligned}$$

A.2. Finding Optimal Number of CPA Runs

In Chapter 3, we present a method for finding the minimum number of CPA runs required for full key recovery using a SAT solver. This appendix provides the Python code used to find this number.

```
import pycryptosat as cs
import sys

# Cardinality constraint system proposed by Sinz
# For more details: C. Sinz, Towards an Optimal CNF Encoding of
#                   Boolean Cardinality Constraints.

# sSat      : the CNF system used
# setId     : list of the variable id of the set
# nb_sets   : number of sets
# cc        : cardinality constraint
# startextra: index of additional variable unused so far,
#             it can be a value large enough.
def Cardinality_Constraints(sSat, setId, nb_sets, cc, startextra):
    sSat.add_clause([-(setId[0]), startextra])
    for j in range(2,cc+1):
        sSat.add_clause([-(startextra+j-1)])
    for i in range(2,nb_sets):
        sSat.add_clause([-(setId[i-1]),
                         startextra+cc*(i-1)
                        ])
        sSat.add_clause([-(startextra+cc*(i-2)),
                         startextra+cc*(i-1)
                        ])
        for j in range(2,cc+1):
            sSat.add_clause([-(setId[i-1]),
                             -(startextra+cc*(i-2)+j-2),
                               startextra+cc*(i-1)+j-1
                            ])
            sSat.add_clause([-(startextra+cc*(i-2)+j-1),
                               startextra+cc*(i-1)+j-1
                            ])
        sSat.add_clause([-(setId[i-1]),
                          -(startextra+cc*(i-2)+cc-1)
                        ])
    sSat.add_clause([-(setId[nb_sets-1]),
                      -(startextra+cc*(nb_sets-2)+cc-1)
                     ])
    startextra+=(nb_sets-1)*cc
```

```
40
41 if __name__ == "__main__":
42     # Run ‘python3 64 19 28 23‘ for k0
43     # Run ‘python3 64 61 39 24‘ for k1
44     n  = int(sys.argv[1]) # n = 64 indexes
45     s1 = int(sys.argv[2]) # 1st shift
46     s2 = int(sys.argv[3]) # 2nd shift
47     ub = int(sys.argv[4]) # upper bound for number of subsets
48
49     # Compute all subset
50     list_set=[]
51     for i in range(n):
52         list_set.append([i,(i+s1)%n,(i+s2)%n])
53     # Save the universe
54     universe=list(range(n))
55     # Create the solver object
56     sSat=cs.Solver()
57
58     # Add constraint
59     # For each element of the universe, we should select at least
60     # one subset containing the element
61     for i in universe:
62         # create an empty disjuction
63         clause_presence=[]
64         for j in list_set:
65             # if subset j contains the element i, we add the
66             # variable corresponding to the set j  to the
67             # disjunctions
68             if i in j:
69                 clause_presence += [list_set.index(j)+1]
70
71         # One variable in clause_presence must be set to true
72         # Add disjunction to the conjunction
73         sSat.add_clause(clause_presence)
74
75     # Cardinality constraints
76     Cardinality_Constraints(sSat,
77                 list(range(1,len(list_set)+1)),
78                 len(list_set),
79                 ub,
80                 len(list_set)+1)
81     # Solve it
82     satq,solution=sSat.solve()
83     # SAT or UNSAT
84     print(satq)
85     # if SAT print the solution
86     if(satq):
87         # for all variable corresponding to a subset
88         for i in range(1,len(list_set)+1):
89             #  if True then the subset has been selected
90             if solution[i]:
91                 print(list_set[i-1])
92             # else the subset has not been selected each element of
93             # this subset should appear in at least one other
94             # selected subset
```

Listing A.1: Python script to find the optimal number of CPA runs.

Appendix B.
Key Recovery Algorithm

In Chapter 6, we present a Persistent Fault Analysis (PFA) attack in which a fault is injected into the 8th-round constant of AES. This appendix details the key-recovery algorithm for the attack.

Let $(C_j^i, \tilde{C}_j^i)$ be the i-th pair of correct-faulty ciphertext byte at index j, where $i \in [1, N]$ and $j \in [0, 15]$. Let ΔS_j be the difference of the j-th byte in the state at the beginning of the last round (state (9) in Figure 6.3). Let K_j and $\hat{K}_j$ be the correct value and the hypothesis for byte at index j of the last round key. To recover K, we perform the following algorithm:

1. Recover (K_{12}, K_9): For each candidate $(\hat{K}_{12}, \hat{K}_9)$ of 2^{16} possibilities, we compute

$$\Delta S_{12} = \texttt{SubBytes}^{-1}(C_{12}^i \oplus \hat{K}_{12}) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_{12} \oplus \hat{K}_{12})$$
$$\Delta S_{13} = \texttt{SubBytes}^{-1}(C_9^i \oplus \hat{K}_9) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_9 \oplus \hat{K}_9)$$

   If $\Delta S_{12} = 2 \cdot \Delta S_{13}$ for every $i \in [1, N]$, then $(\hat{K}_{12}, \hat{K}_9)$ is a good candidate.

2. Recover (K_6, c): For each candidate $(\hat{K}_{16}, \hat{c})$ of 2^{16} possibilities, we compute

$$\Delta S_{14} = \texttt{SubBytes}^{-1}(C_6^i \oplus \hat{K}_6) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_6 \oplus \hat{K}_6 \oplus c)$$
$$\Delta S_{13} = \texttt{SubBytes}^{-1}(C_9^i \oplus K_9) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_9 \oplus K_9)$$

   If $\Delta S_{14} = \Delta S_{13}$ for every $i \in [1, N]$, then $(\hat{K}_6, \hat{c})$ is a good candidate.

3. Recover (K_3, b): For each candidate $(\hat{K}_3, \hat{b})$ of 2^{16} possibilities, we compute

$$\Delta S_{15} = \texttt{SubBytes}^{-1}(C_3^i \oplus \hat{K}_3) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_3 \oplus \hat{K}_3 \oplus b)$$
$$\Delta S_{13} = \texttt{SubBytes}^{-1}(C_9^i \oplus K_9) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_9 \oplus K_9)$$

   If $\Delta S_{15} = 3 \cdot \Delta S_{13} \oplus b$ for every $i \in [1, N]$, then $(\hat{K}_3, \hat{b})$ is a good candidate.

4. Recover (K_5, K_{15}): For each candidate $(\hat{K}_5, \hat{K}_{15})$ of 2^{16} possibilities, we compute

$$\Delta S_9 = \texttt{SubBytes}^{-1}(C_5^i \oplus \hat{K}_5) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_5 \oplus \hat{K}_5)$$
$$\Delta S_{11} = \texttt{SubBytes}^{-1}(C_{15}^i \oplus \hat{K}_{15}) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_{15} \oplus \hat{K}_{15})$$

   If $\Delta S_{11} = 3 \cdot \Delta S_9 \oplus b$ for every $i \in [1, N]$, then $(\hat{K}_5, \hat{K}_{15})$ is a good candidate.

5. Recover (K_8, a): For each candidate $(\hat{K}_8, \hat{a})$ of 2^{16} possibilities, we compute

$$\Delta S_9 = \texttt{SubBytes}^{-1}(C_5^i \oplus K_5) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_5 \oplus K_5)$$
$$\Delta S_8 = \texttt{SubBytes}^{-1}(C_8^i \oplus \hat{K}_8) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_8 \oplus \hat{K}_8)$$

   If $\Delta S_8 = 2 \cdot \Delta S_9 \oplus a$ for every $i \in [1, N]$, then $(\hat{K}_8, \hat{a})$ is a good candidate.

6. Recover K_2: For each candidate $\hat{K}_2$ of 2^8 possibilities, we compute

$$\Delta S_9 = \texttt{SubBytes}^{-1}(C_5^i \oplus K_5) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_5 \oplus K_5)$$
$$\Delta S_{10} = \texttt{SubBytes}^{-1}(C_2^i \oplus \hat{K}_2) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_2 \oplus \hat{K}_2 \oplus c)$$

   If $\Delta S_{10} = \Delta S_9 \oplus a$ for every $i \in [1, N]$, then $\hat{K}_2$ is a good candidate.

7. Recover (K_4, K_1): For each candidate $(\hat{K}_4, \hat{K}_1)$ of 2^{16} possibilities, we compute

$$\Delta S_4 = \texttt{SubBytes}^{-1}(C_4^i \oplus \hat{K}_4) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_4 \oplus \hat{K}_4 \oplus a)$$
$$\Delta S_1 = \texttt{SubBytes}^{-1}(C_1^i \oplus \hat{K}_1) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_1 \oplus \hat{K}_1)$$

   If $\Delta S_4 = 2 \cdot \Delta S_1 \oplus b$ for every $i \in [1, N]$, then $(\hat{K}_4, \hat{K}_1)$ is a good candidate.

8. Recover (K_{14}, K_{11}): For each candidate $(\hat{K}_{14}, \hat{K}_{11})$ of 2^{16} possibilities, we compute

$$\Delta S_6 = \texttt{SubBytes}^{-1}(C_{14}^i \oplus \hat{K}_{14}) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_{14} \oplus \hat{K}_{14} \oplus c)$$
$$\Delta S_7 = \texttt{SubBytes}^{-1}(C_{11}^i \oplus \hat{K}_{11}) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_{11} \oplus \hat{K}_{11} \oplus b)$$

   If $\Delta S_7 = 3 \cdot \Delta S_6 \oplus b$ for every $i \in [1, N]$, then $(\hat{K}_{14}, \hat{K}_{11})$ is a good candidate.

9. Recover (K_0, K_{13}): For each candidate $(\hat{K}_0, \hat{K}_{13})$ of 2^{16} possibilities, we compute

$$\Delta S_0 = \texttt{SubBytes}^{-1}(C_0^i \oplus \hat{K}_0) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_0 \oplus \hat{K}_0 \oplus a)$$
$$\Delta S_1 = \texttt{SubBytes}^{-1}(C_{13}^i \oplus \hat{K}_{13}) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_{13} \oplus \hat{K}_{13})$$

   If $\Delta S_0 = 2 \cdot \Delta S_1 \oplus a$ for every $i \in [1, N]$, then $(\hat{K}_0, \hat{K}_{13})$ is a good candidate.

10. Recover (K_{10}, K_7): For each candidate $(\hat{K}_{10}, \hat{K}_7)$ of 2^{16} possibilities, we compute

$$\Delta S_2 = \texttt{SubBytes}^{-1}(C_{10}^i \oplus \hat{K}_{10}) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_{10} \oplus \hat{K}_{10} \oplus c)$$
$$\Delta S_3 = \texttt{SubBytes}^{-1}(C_7^i \oplus \hat{K}_7) \oplus \texttt{SubBytes}^{-1}(\tilde{C}_7 \oplus \hat{K}_7)$$

   If $\Delta S_3 = 3 \cdot \Delta S_2 \oplus b$ for every $i \in [1, N]$, then $(\hat{K}_{10}, \hat{K}_7)$ is a good candidate.