UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences
Département d'Informatique

Use of simulators for side-channel analysis

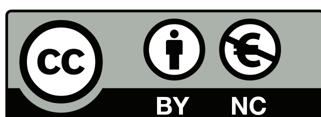
Leakage detection and analysis of cryptographic systems in early stages of development

Nikita Veshchikov

Brussels, 2017

ISBN: 978-949231243-3

© Nikita Veshchikov, 2017



This work is licensed under a Creative Commons Attribution-NonCommercial License.

UNIVERSITÉ LIBRE DE BRUXELLES ULB

Supervisors

Prof. Dr. Olivier Markowitch	Université libre de Bruxelles, Belgium
Prof. Dr. Frédéric Robert	Université libre de Bruxelles, Belgium

Doctoral Thesis Committee

Prof. Dr. Lejla Batina	Radboud Universiteit, Nijmegen, Netherlands
Prof. Dr. Gilles Geeraerts	Université libre de Bruxelles, Belgium
Prof. Dr. Sylvain Guilley	Télécom ParisTech and Secure-IC, France
Prof. Dr. Yves Roggeman	Université libre de Bruxelles, Belgium
Dr. Liran Lerman	Université libre de Bruxelles, Belgium

Acknowledgements

This thesis is the result of 6 years of work at Université libre de Bruxelles (ULB) where I was doing the research in cryptography while also spending a considerable amount of time teaching computer sciences to the new students as an assistant during exercise sessions. All the achievements would not have been possible without the help and support of many people who I would like to thank.

First of all, I would like to say thank you to my family, especially my mom and dad, who supported my decision to pursue this adventure from the beginning and through all the years while I was working on it. Спасибо! Next, I would like to thank my PhD supervisor Olivier Markowitch for giving me a lot of liberty to pursue different research directions, and my second supervisor Frédéric Robert for always asking me difficult questions and making me look at the topic from a new angle. Thank you for guiding me through this journey and for all the feedback on this document. Merci!

While working as a researcher I learned a lot through various teamwork projects that resulted in publications for which I am very grateful to all my collaborators. I want to thank my colleague Liran Lerman for being very serious about research and for all the notes and feedback on the draft of this document. Liran is also the only person that I know who can be in a quantum state of being in and out of his office at the same time. תודה! I would also like to thank another colleague of mine Stephane Fernandes Medeiros for not being so serious. Obrigado! Special acknowledgement goes to Liran and Stephane as my first co-authors for introducing me to the magical world of "writing and submitting scientific papers". I am also very grateful to Jorge Nakahara Jr, who was as a post-doc in our team while I was working on my thesis. He gave me some of the best advices on how to structure my thesis and other documents. Obrigado! Another person who I would like to thank is Sylvain Guilley, his valuable inputs helped me to break the encryption scheme of the DPA Contest, which gave rise to lots of interesting results. Merci! Next, I would like to thank Stjepan Picek for showing me a very special approach to the word "deadline" and also for all the crazy awesome time in Šibenik. Hvala ti! I also want to thank Kostas Papagiannopoulos for the huge amount of experimental work that he did during our collaboration while I was writing the code and integrating all of his findings in the software. Ευχαριστώ! I would also like to thank one of my newest colleagues (and co-authors) François

Gérard for the combination of fun and serious work that he brought to the team while we were working on a relatively unusual paper. Merci! Another person that I would like to acknowledge is Antonio Paolillo who was my classmate throughout the university and then also started his PhD in a different domain of computer sciences. He invited me to work on one of his projects, even though it was not related to the IT security it was incredibly interesting and challenging enough, so I gladly worked on it and learned a lot. Grazie!

The research that I did at the university was done at the QualSec group, side by side with some of the most incredible people I have ever met. I did not get a chance to write a paper with every single one of them. Nevertheless, I have learned a lot about cryptography and security from them. I have learned a lot about network security from Naïm Qachri and I want to thank him, especially for the etc et tout. Frédéric Lafitte is another person who taught me a lot about the automated approach to cryptanalysis using SAT solvers, but most importantly about some of the politics in the domain of cryptographic research. Merci! Another colleague from whom I learned a lot, especially about voting systems, is the anonymous known as "J". Spoiler alert: it is Jérôme Dossogne. Merci! I also learned a great deal about random numbers and about quantum cryptography from Helena Bruyninckx. Dank je wel!

During the 6 years of research activities I had a great opportunity to go to numerous summer schools in Bulgaria, Greece and Croatia (almost every year!) where I learned a great deal about cryptography and computer security from some of the best minds in the domain. Those were definitely among the best places to deepen my knowledge about cryptography and to meet other researchers and future collaborators. To be honest, it was also a great place to relax after a long year in Brussels (but only in the evening after the seminars!). Thus, I would like to thank people from ECRYPT II who managed the summer school on tools for cryptographers at Mykonos in 2012; all the organisers and speakers from the summer school on the Design and security of cryptographic functions, algorithms and devices at Albena in 2013; and all people who repeatedly organised, managed and made wonderful lectures for the summer schools on real world cryptography in Šibenik from 2014 to 2016. Thank you, it was really interesting and helpful!

Another place that will always stay in my memory and where I also learned a lot of things about computer security, met new people and started a fruitful collaboration is the Digital Security Group at Radboud university at Nijmegen in Netherlands. I spent there an entire month during a scientific mission which resulted in a publication. I want to thank Lejla Batina for welcoming me at the group. Hvala! I would also like to thank Joan Daemen for finding some time to answer my questions and commenting on some of my ideas while I was there. Dank u! Same goes for Peter Schwabe who also found a little bit of his time to give me a couple of useful advices at several occasions. Danke! I also want to thank all the guys and girls from Nijmegen who welcomed me and made me feel like at home, especially Bariş Ege, Antonio de la

Piedra, Ko Stoffelen, Joost Renes, Joost Rijneveld, Veelasha Moonsamy, Pol Van Aubel, Pedro Maat Massolino, Louiza Papachristodoulou and Freek Verbeek. Bedankt! I also would like to thank Irma Haerkens who works as a secretary in the Digital Security Group and who made my life there easier by taking care of all the administrative arrangements. A special thank you is for Anna Krasnova, who actually gave me the idea for this short term scientific mission at Radboud. Спасибо!

During these year I repeatedly met some great security researchers in different cities all over the Europe. We had numerous conversations on a wide variety of topics which gave me new ideas and insights. So, I would like to thank Lukasz Chmielewski for all the interesting conversations that we had about side-channel analysis. Dziękuję Ci! I am also grateful to Ricardo Chaves for all the discussions that we had every time that we met at a workshop and even at the airport. Obrigado!

During my research I sent a tremendous number of e-mails to the authors of most of the papers that I read. Each time when something was unclear or when I needed an extra piece of information I was sending new e-mails, sometimes I would even chase people at conferences to get an answer (at some point I even got a skype interview to get more details about one of the simulators). This is exactly how I got some of the information that was not in the original papers. I would like to thank all the researchers who answered to my numerous e-mails, it was really helpful! The full list is very long, but I want to highlight some of them: Colin O'Flynn from NewAE for all the extra information about the CHES CTF Challenge (also thank you for comments on my design of the acquisition board); guys from Riscure, especially for their workshops on side-channel attacks; the team of researchers who work on side-channel attacks in Bristol (I have already lost count how many questions I have asked in my e-mails and during life conversations). Thank you for the answers!

While working at the university I had to juggle between teaching and research activities. I would like to acknowledge some people who made my life as an assistant easier and who taught me a lot (which gave me more time for research). I want to thank Markus Lindström as well as already mentioned Jérôme and Naïm for their help and advices that I got when I started to work at the university. Tack and Merci! I also want to express my gratitude to professors Yves Roggeman and Gilles Geeraerts for their valuables lessons on how to better ~~handle~~ teach students. Merci!

I am also very grateful to the people who accepted my request to be in the jury at the defence of my thesis: Lejla Batina, Gilles Geeraerts, Sylvain Guilley, Yves Roggeman and Liran Lerman. Thank you for the time you spent reading the text and also for finding the time for the defence itself. Most importantly, thank you for all the questions and remarks that allowed me to improve this document.

I want to applaud all the people who read and commented on various preliminary versions of this text or some parts of it: both of my supervisors, all the jury members, but also my mother (who read everything), my uncle (who helped me with the abstract) and Stjepan (who double-checked everything about evolutionary algorithms).

Your inputs were very valuable and allowed me to greatly improve the final result. Thank you very much!

A very special word of appreciation goes for the help with the illustrations that I used in this document. I would like to thank a great photographer Nina Ricci-Fedotova for helping me to create beautiful photos of the hardware at the lab. Grazie mille, спасибо! Another very big thank you is for a very creative Masha Kashlyaeva for the magnificent artwork for the cover of this book. Спасибо!

I also want to acknowledge several organisations that helped me a lot by financing my numerous (work related!) trips to Bulgaria, Croatia, France, Greece, Netherlands, Taiwan and Unites States. I want to thank Faculté des Sciences of ULB for the yearly budget that allowed to go to conferences where I presented my work. I also want to thank Fonds de la Recherche Scientifique de la Belgique for funding my trips to various workshops where I met other researchers. Another organisation that helped me to learn more about cryptography also by funding one of my trips to the summer school is ECRYPT, thank you guys! I also want to express my gratitude to all the people who manage the ICT COST Action IC1204, the funding form this action allowed me to work during a whole month in Nijmegen. Thank you!

All the trips to workshops and conferences would not have been possible without some extra help. I want to thank secretaries of our department Pascaline Browaeys and Véronique Bastin for their help with the management of my trips and for making sure I get all the refunds at the end of it! I also want to thank our secretary Maryka Peetroons for her help with the massive amount of administrative paperwork and arrangements that I had to bear during all these years. Merci!

Teaching computer sciences to students was a huge part of my activities at ULB. I want to thank all the students who made my life more challenging and much more fun. Special thanks is for all the students who decided to do their master theses and other security-related projects under my supervision, reading your works allowed me to understand how to present my own results better, I also learned a couple of things thanks to your works. A special acknowledgement goes to guys from UrLab (the hackerspace), who invited me at some point to give a talk on “the stuff that I do that is related to security”, it made me reflect on how to better explain what exactly my research topic is all about. Another special word of gratitude is for old and new members of Cercle Informatique (the student fraternity), thank you for the interest in my work and for your questions about it (that mostly happen over a beer). I would also like to thank UrLab and Cerlce Informatique for their help in the organisation of cryptography related event that I somehow managed to organize!

Finally, I want to thank all my friends who supported me by regularly asking *“How is your PhD?”* and *“How many pages do you have to write?”* every time we met. Thank you for showing your interest in my work by asking me questions about security and cryptography such as *“Could you break into a bank?”* and also for listening my (probably too exhaustive) explanations. Thank you very much!

Use of simulators for side-channel analysis

Nikita Veshchikov

Abstract

Cryptography is the foundation of modern IT security, it provides algorithms and protocols that can be used for secure communications. Cryptographic algorithms ensure properties such as confidentiality and data integrity. Confidentiality can be ensured using encryption algorithms. Encryption algorithms require a secret information called a key. These algorithms are implemented in cryptographic devices. There exist many types of attacks against such cryptosystems, the main goal of these attacks is the extraction of the secret key. Side-channel attacks are among the strongest types of attacks against cryptosystems.

Side-channel attacks focus on the attacked device, they measure its physical properties in order to extract the secret key. Thus, these attacks target weaknesses in an implementation of an algorithm rather than the abstract algorithm itself. Power analysis is a type of side-channel attacks that can be used to extract a secret key from a cryptosystem through the analysis of its power consumption while the target device executes an encryption algorithm. We can say that the secret information is leaking from the device through its power consumption. One of the biggest challenges in the domain of side-channel analysis is the evaluation of a device from the perspective of side-channel attacks or in other words the detection of information leakage. A device can be subject to several sources of information leakage and it is actually relatively easy to find just one side-channel attack that works (by exploiting just one source of leakage), however it is very difficult to find all sources of information leakage or to show that there is no information leakage in the given implementation of an encryption algorithm. Evaluators use various statistical tests during the analysis of a cryptographic device to check that it does not leak the secret key. However, in order to perform such tests the evaluation lab needs the device to acquire the measurements and analyse them. Unfortunately, the development process of cryptographic systems is rather long and has to go through several stages. Thus, an information leakage that can lead to a side-channel attack can be discovered by an evaluation lab at the very last stage using the final product. In such case, the whole process has to be restarted in order to fix the issue, this can lead to significant time and budget overheads. The rationale is that developers of cryptographic systems would like to be able to detect issues related to side-channel analysis during the development of the system, prefer-

ably on the early stages of its development. However, it is far from being a trivial task because the end product is not yet available and the nature of side-channel attacks is such that it exploits the properties of the final version of the cryptographic device that is actually available to the end user.

The goal of this work is to show how simulators can be used for the detection of issues related to side-channel analysis during the development of cryptosystems. This work lists the advantages of simulators compared to physical experiments and suggests a classification of simulators for side-channel analysis. This work presents existing simulators that were created for side-channel analysis, more specifically we show that there is a lack of available simulation tools and that therefore simulators are rarely used in the domain. We present three new open-source simulators called Silk, Ascold and Savrasca. These simulators are working at different levels of abstraction, they can be used by developers to perform side-channel analysis of the device during different stages of development of a cryptosystem. We show how Silk can be used during the preliminary analysis and development of cryptographic algorithms using simulations based on high level of abstraction source code. We used it to compare S-boxes as well as to compare shuffling countermeasures against side-channel analysis. Then, we present the tool called Ascold that can be used to find side-channel leakage in implementations with masking countermeasure using the analysis of assembly code of the encryption. Finally, we demonstrate how our simulator called Savrasca can be used to find side-channel leakage using simulations based on compiled executable binaries. We use Savrasca to analyse masked implementation of a well-known contest on side-channel analysis (the 4th edition of DPA Contest), as a result we demonstrate that the analysed implementation contains a previously undiscovered information leakage. Through this work we also compared results of our simulated experiments with real experiments coming from implementations on microcontrollers and showed that issues found using our simulators are also present in the final product. Overall, this work emphasises that simulators are very useful for the detection of side-channel leakages in early stages of development of cryptographic systems.

Utilisation des simulateurs pour l'analyse des attaques par cannaux auxiliaires

Nikita Veshchikov

Résumé

La cryptographie est la fondation de la sécurité de données, elle nous fournit algorithmes et protocoles que l'on peut utiliser pour sécuriser les transmissions de données. Les algorithmes cryptographiques permettent d'assurer les propriétés telles que la confidentialité et l'intégrité de données. La confidentialité peut être assurée à l'aide des algorithmes de chiffrement, la sécurité des algorithmes de chiffrement se repose sur une valeur secrète qu'on appelle une clé. Ces algorithmes sont implémentés dans des dispositifs cryptographiques. Il existe beaucoup d'attaques contre de tels systèmes, le but principal de ces attaques est d'extraire la clé secrète. Les attaques par canaux auxiliaires sont parmi les attaques les plus puissantes contre les systèmes cryptographiques.

Les attaques par canaux auxiliaires se focalisent sur le dispositif cryptographique, elles analysent les propriétés physiques du dispositif attaqué pour en extraire la clé secrète. Donc, ce type d'attaque exploite les faiblesses de l'implémentation de l'algorithme plutôt que d'attaquer l'abstraction qui est l'algorithme de chiffrement en lui même. L'analyse de la consommation d'énergie est un des types d'attaque par canaux auxiliaires, on peut l'utiliser pour extraire la clé secrète d'un système cryptographique en analysant sa consommation d'énergie pendant que le dispositif exécute l'algorithme de chiffrement. On dit qu'il y a une fuite d'information à travers sa consommation d'énergie. Un des plus grands défis dans le domaine des attaques par canaux auxiliaires est l'évaluation des dispositifs cryptographiques cet-à-dire la détection de fuite d'information à travers les canaux auxiliaires. Un système cryptographique peut être sujet à plusieurs fuites d'information, il est donc plus facile de trouver une attaque qui permet d'extraire la clé secrète (en utilisant une source de fuite d'information) que de trouver toutes les attaques ou de montrer qu'il n'y a pas de fuite d'information dans l'implémentation donnée d'un algorithme de chiffrement. Les évaluateurs utilisent des tests statistiques afin de détecter les fuites d'information venant d'une implémentation. Un laboratoire a besoin du dispositif pour acquérir les données et les analyser. Malheureusement, le processus du développement d'un dispositif cryptographique est assez long et nécessite beaucoup d'étapes. Donc, si une fuite d'information qui mène à une attaque est détectée par un laboratoire d'évaluation c'est fait à la dernière étape du développement en utilisant le produit final. Dans

ce cas il faut revenir au début du processus du développement pour éliminer le problème qui mène à une fuite, cela peut engendrer des frais supplémentaires importants. En résumé, les développeurs des systèmes cryptographiques aimeraient pouvoir détecter les problèmes liés aux attaques par canaux auxiliaires le plus tôt possible dans le processus du développment du dispositif. Néanmoins, ce n'est pas une tâche facile, car le produit final n'est pas encore disponible et la nature des attaques par canaux auxiliaires est telle qu'elle exploite les propriétés du produit final.

Le but de ce travail est de montrer comment les simulateurs peuvent être utilisés pour détecter des problèmes liés aux fuites d'information et aux attaques par canaux auxiliaires pendant le développement d'un système cryptographique. Ce travail présente les avantages des simulateurs comparés à des expériences physiques et propose une classification des simulateurs existants qui sont utilisés dans le domaine des attaques par canaux auxiliaires. Dans ce travail on présente les simulateurs existants et on souligne le fait qu'il y a un manque de simulateurs disponibles dans le domaine des attaques par canaux auxiliaires, ce qui est une des raison pour un taux faible d'utilisation des simulateurs dans ce domaine. On présente trois nouveaux simulateurs, développés en tant que logiciels libres, qu'on appelle Silk, Ascold et Savrasca. Ces trois simulateurs opèrent à des différents niveaux d'abstraction, ils peuvent être utilisés par des développeurs pour analyser le produit en développement du point de vue des attaques par canaux auxiliaires à différentes phases du développement du système. On montre comment on peut utiliser Silk pendant les tests et analyses préliminaires et durant le développement des nouveaux algorithmes cryptographiques en se basant sur le code source de haut niveaux d'abstraction. Nous avons utilisé Silk pour comparer des S-boxes ainsi que pour comparer des techniques de mélange qui sont utilisées en tant que contre-mesures contre les attaques par canaux auxiliaires. Ensuite, nous présentons l'outil Ascold qui peut être utilisé pour détecter des fuites d'information dans les implémentations qui utilisent la contre-mesure appelée masquage, l'outil en question se base sur l'analyse du code assembleur. Enfin, nous montrons comment le simulateur Savrasca peut être utilisé pour trouver des fuites d'information à travers les canaux auxiliaires en se basant sur une simulation faite à partir du code binaire exécutable compilé. Nous avons utilisé Savrasca pour analyser l'implémentation masquée venant d'un concours sur les attaques par canaux auxiliaires (la 4ème édition de DPA Contest), nous avons pu montrer que l'implémentation analysée contient des fuites d'information qui n'étaient pas découvertes auparavant. Durant ce travail nous avons aussi comparé des résultats basés sur les simulations avec les résultats qui utilisent des vrais mesures prises sur un micro-contrôleur, nous avons montré que les problèmes que l'on peut trouver à l'aide de nos simulateurs sont en effet présents dans l'implémentation finale. En conclusion, ce travail souligne que les simulateurs sont très utiles pour la détection de fuite d'information et pour l'analyse des attaques par canaux auxiliaires tôt dans le processus du développement des systèmes cryptographiques.

Использование симуляторов для анализа атак по сторонним каналам

Никита Сергеевич Вещиков

Аннотация

Криптография является основой и фундаментом современной информационной безопасности — эта наука предоставляет алгоритмы и протоколы, которые можно использовать для безопасной передачи данных. Криптографические алгоритмы помогают обеспечивать такие свойства, как, например, конфиденциальность и целостность данных. Конфиденциальность может быть обеспечена с помощью алгоритмов шифрования. Для выполнения задачи в алгоритмах шифрования используют секретное число, которое специалисты по шифрованию называют ключом. Реализацию алгоритма шифрования обычно называют шифровальным устройством или, в более общем случае, криптографической системой. Существует огромное множество разнообразных видов атак на криптографические системы, главной целью этих атак является извлечение секретного ключа шифрования. Атаки по сторонним каналам — это один из самых мощных и эффективных видов атак на шифровальные устройства и другие криптографические системы.

При атаках по сторонним каналам, атакующие концентрируют свои усилия на само́м шифровальном устройстве — они используют его физические свойства и характеристики для того, чтобы извлечь ключ шифрования. Таким образом, подобные атаки нацеливают свои усилия на уязвимости и недостатки в реализации алгоритма, а не на сам алгоритм. Атаки по энергопотреблению являются одной из разновидностей атак по сторонним каналам, они могут быть использованы для извлечения секретного ключа из криптографического устройства путём анализа энергопотребления устройства во время выполнения алгоритма шифрования. Можно сказать, что происходит утечка информации из шифровального устройства благодаря данным, полученным через анализ его энергопотребления. Одной из самых сложных задач в области атак по сторонним каналам является оценка криптографических устройств с точки зрения этих атак. Другими словами, обнаружение утечек информации через сторонние каналы — одна из самых сложных проблем в этой области. Шифровальное устройство может иметь несколько недостатков, приводящих к разным источникам утечки информации. Найти одну работающую атаку по сторонним каналам (использующую один источник утечки информации) доста-

точно легко, а обнаружить все источники утечек информации или доказать, что в конкретном данном устройстве их нет, очень сложно. Лаборатории, занимающиеся оценкой утечек информации, часто прибегают к помощи статистических тестов для анализа шифровальных устройств, чтобы обнаружить утечку информации по сторонним каналам. Однако, для проведения подобных тестов необходимо само криптографическое устройство для проведения опытов и сбора данных для их последующего анализа. К сожалению, процесс разработки криптографических систем трудоёмок и занимает много времени. Следовательно, утечку информации, которую можно использовать для атаки по сторонним каналам, можно обнаружить только на са́мом последнем этапе разработки системы во время анализа готового устройства. В случае обнаружения утечки информации необходимо вернуться к разработке системы для того, чтобы исправить ошибку, а такой подход может привести к большим затратам времени и накладным расходам. Отсюда следует, что разработчики криптографических устройств хотели бы иметь возможность обнаружения проблем, связанных с атаками по сторонним каналам, предпочтительно на самых ранних стадиях разработки криптографических систем. Тем не менее, это далеко не самая простая задача, так как в процессе разработки само устройство еще не доступно (не готово), в то время как природа атак по сторонним каналам такова, что они используют свойства конечного продукта, то есть, устройства, которое попадет в руки конечным пользователям.

Цель этой работы — показать, как симуляторы могут быть использованы для обнаружения утечек информации и атак по сторонним каналам во время разработки криптографических систем. В данной работе перечисляются преимущества симуляторов над обычными экспериментальными способами обнаружения атак по сторонним каналам, и здесь представлена классификация симуляторов для анализа атак по сторонним каналам. В работе рассмотрены существующие симуляторы, разработанные для анализа утечки информации через сторонние каналы; более того: нам удалось показать что в области атак по сторонним каналам не хватает легкодоступных симуляторов, что является одной из главных причин, по которым симуляторы редко используются в этой области. Здесь представлены три новых симулятора с открытым кодом: Silk [силк], Ascold [аскольд] и Savrasca [савраска]. Эти симуляторы оперируют данными на разных уровнях абстракции, они могут быть использованы на разных стадиях процесса разработки шифровальных устройств для обнаружения утечек информации по сторонним каналам. В этой работе показано, как симулятор Silk может быть использован для предварительного анализа и для разработки новых алгоритмов; этот инструмент основан на анализе программ, написанных на языке высокого уровня абстракции. Симулятор Silk был успешно использован для сравнения S-блоков, а также для сравнения алгоритмов перемешивания, которые используются в качестве защиты от атак по сто-

ронним каналам. Затем, в данной работе представлен симулятор Ascold, он может быть использован для обнаружения утечек информации в реализациях алгоритмов, которые используют маскировку в качестве защиты от атак по сторонним каналам. Этот инструмент основан на анализе программы на языке ассемблера. И наконец, эта работа демонстрирует, как симулятор Savrasca может быть использован для обнаружения утечек информации на основе анализа скомпилированного исполняемого файла. Этот симулятор был успешно использован для анализа реализации алгоритма шифрования с использованием маскировки в качестве контрмеры против атак по сторонним каналам, данный алгоритм был представлен на конкурсе атак по сторонним каналам (4-я версия конкурса DPA Contest [дипиэй контест]). В результате анализа с помощью нашего симулятора в данной реализации были обнаружены ранее неизвестные уязвимости, которые могут быть использованы для атак по сторонним каналам. В ходе этой работы результаты атак, основанных на симуляторах, были сравнены с результатами атак на реализациях алгоритмов шифрования в микроконтроллерах. Было успешно показано, что проблемы, обнаруженные с помощью симуляторов, также присущи шифровальным устройствам. В общем, эта работа подчеркивает, что симуляторы являются очень ценным инструментом для обнаружения утечек информации и для анализа атак по сторонним каналам на ранних стадиях в процессе разработки криптографических систем.

Contents

List of Notations **v**
Cryptography . . . v
Side-channel analysis . . . v
Statistics . . . vi
Sets and elements . . . vi
List of Abbreviaitons . . . vii

List of Figures **ix**

List of Tables **xi**

List of Listings **xiii**

1 Introduction **1**

I Preliminary notions **5**

2 Cryptography **7**
2.1 Ciphers . . . 8
2.1.1 Block ciphers . . . 10
2.1.2 Attacks on block ciphers . . . 17
2.2 Summary . . . 20

3 Side-channel analysis **21**
3.1 Types of side-channel attacks . . . 22
3.1.1 Information channel . . . 22
3.1.2 Invasiveness . . . 24
3.1.3 Interference . . . 24
3.1.4 Profiled and unprofiled attacks . . . 26
3.1.5 Simple and differential analysis . . . 27
3.1.6 Summary of types of side-channel attacks . . . 28

3.2 Power analysis 28
3.2.1 Acquisition setup 29
3.2.2 Target operation 35
3.2.3 Leakage model 39
3.2.4 Distinguishers 41
3.2.5 Key enumeration 51
3.3 Analysis of side-channel attacks 52
3.3.1 Performance of an attack 53
3.4 Countermeasures 55
3.4.1 Masking 55
3.4.2 Hiding 60
3.4.3 Other countermeasures 61
3.4.4 Summary on countermeasures 63
3.5 Summary 64

4 The problem of leakage detection 65
4.1 Leakage detection 66
4.2 Analysis during early stages 68
4.3 Goals 71

II Contributions 73

5 Simulation tools for side-channel analysis 75
5.1 Motivation 78
5.2 Levels of abstraction 83
5.3 Survey of existing simulators 86
5.3.1 Other works related to simulations 94
5.4 Summary 95

6 SILK 97
6.1 Description of the tool 98
6.1.1 Parameters 99
6.1.2 Discussion 102
6.2 Evaluation of S-boxes 104
6.2.1 Results based on theoretical metrics 105
6.2.2 Experimental results on simulations 106
6.2.3 Experimental results on a real device 112
6.3 Improvement of S-boxes 114
6.3.1 Genetic algorithms and search strategy 114
6.3.2 Results for Correlation Power Analysis 117
6.3.3 Results for Template Attacks 121

6.3.4 Discussion 122
6.4 Scalable shuffling schemes 127
6.4.1 Extensions of random start index 128
6.4.2 Reverse shuffle 131
6.4.3 Sweep swap shuffle 132
6.5 Analysis of shuffling schemes 135
6.5.1 Randomization 135
6.5.2 Number of shuffles 137
6.5.3 Resources 139
6.5.4 Resistance against side-channel attacks 142
6.5.5 Applications & modifications 147
6.5.6 Discussion 147
6.6 Summary 150

7 ASCOLD 153
7.1 Acquisition setup and evaluation 154
7.2 ILA-Breaching Effects 154
7.2.1 Overwrite effect 155
7.2.2 Memory remnant effect 156
7.2.3 Neighbour leakage effect 158
7.3 Description of the tool 161
7.4 1st order masked S-box for Rectangle cipher 163
7.5 Summary 168

8 SAVRASCA 171
8.1 Description of the tool 172
8.2 Analysis of the DPA Contest 4 174
8.3 Analysis of AES-RSM used in DPA Contest 4 178
8.3.1 Mask bias 178
8.3.2 Experimental results 182
8.3.3 Balanced values for masks 189
8.4 Note on DPA Contest 4.2 194
8.5 Summary 196

9 Conclusions 199

A SILK example 207

B Success rate of S-boxes using simulations 209

C S-boxes generated using evolutionary computations 211
C.1 Success rate of attacks on the S-boxes 213

D Heatmaps of shuffling schemes **215**

E Success rates of attacks on shuffling schemes **221**

F Ascold **example** **225**

G List of microcontrollers supported by SAVRASCA **227**

Bibliography **229**

List of Notations

Cryptography

D Decryption algorithm

E Encryption algorithm

G Key generation algorithm

δ_F Differential uniformity of a function F

$\mathcal{N}_F$ Nonlinearity of a function F

S S-box (Substitution function)

$\mathcal{C}$ A set of ciphertexts

$\mathcal{Z}$ A set of intermediate values used in an encryption algorithm

$\mathcal{P}$ A set of plaintexts

c A ciphertext

z An intermediate value manipulated by an algorithm

k A cryptographic key

p A plaintexts

Lower index $_i$, ith part (byte) of a key (plaintext, ciphertext or intermediate value)

Side-channel analysis

$\mathcal{T}$ A set of power traces

T A power trace

h Hypothesis about the value of an encryption key

L^* True leakage function

L Leakage model (function that models the leakage)

Statistics

ρ Correlation coefficient

μ Mean value

P Probability of an event

σ^2 Variance

Sets and elements

$|x|$ absolute value of x

$\{0,1\}^*$ Set of all binary strings

$\{0,1\}^n$ Set of all binary strings of length n

$\|\mathcal{X}\|$ cardinality of set $\mathcal{X}$

$\mathcal{X}[i]$ The ith element of the set or list $\mathcal{X}$

$\mathcal{X}[\cdot]$ All elements of the set $\mathcal{X}$

$x \oplus y$ exclusive-or between values x and y

$\mathcal{X} \oplus y$ exclusive-or between the set $\mathcal{X} = x_1, x_2, \cdots x_N$ and value y, the result is the set $\mathcal{R} = r_1, r_2, \cdots r_N$ of the same size N where each $r_i = x_i \oplus y$. This operation is comutative i.e., $y \oplus \mathcal{X}$ gives the same result

$\mathbb{F}_2^n$ A vector space that contains all n-bit binary vectors, each element e of $\mathbb{F}_2^n$ is a binary string of size n ($e \in \{0,1\}^n$)

Abbreviations

ALU Arithmetic Logic Unit 23, 173, 196

ASIC Application Specific Integrated Circuit 29, 40, 81, 88, 205

AVR Alf And Vegard's RISC (Reduced Instruction Set Computer) Processor 30, 59, 153–155, 161–164, 168, 171, 172, 202, 204

CC Confusion Coefficient 55, 62, 104–110, 117, 118

CPA Correlation Power Analysis 46, 47, 55, 106, 108–113, 117–120, 122, 125, 142–145, 149, 154, 155, 157, 158, 178, 210, 212, 213, 221–223

CPU Central Processing Unit 25, 172, 176

DoM Difference Of Means 44–46, 182, 183, 185–187, 189, 196

DPA Differential Power Analysis 44, 75, 98, 142, 151, 177

EMA Electro-Magnetic Analysis 23, 24, 28, 177, 194

FPGA Field Programmable Gate Array 29, 30, 40, 88, 160, 205

HD Hamming Distance 41, 43, 58, 70, 89, 92, 93, 99, 151, 155, 173, 175

HDL Hardware Description Language 87, 92, 93

HW Hamming Weight 41, 42, 46, 47, 54, 70, 89, 91–93, 99, 105, 106, 112, 117, 127, 142, 151, 155, 157, 158, 173, 175

I/O Input And Output Operations 80, 155, 172

ILA Independant Leakage Assumption 56, 58, 153–159, 161–165, 167–169, 173, 202, 203, 205

IT Information Technology 1, 2, 7, 68

LEMS Low-Entropy Masking Scheme 57, 186, 193, 196, 197, 200, 205

LSB Least Significant Bit 93, 102, 129

MIA Mutual Information Analysis 51

MSB Most Significant Bit 44, 45, 102, 129, 176–178, 180, 185, 186, 189, 194

MTL Memory Transitions Leak 40, 41, 58, 59, 99, 151, 153, 173

ODL Only Manipulated Data Leak 40, 41, 56, 58, 59, 99, 151, 153, 173

RAM Random Access Memory 25, 80, 81, 114, 149

RNG Random Number Generator 26, 57, 70, 81, 128–130, 132, 133, 141, 146, 165, 167, 168

RP Random Permutation 60, 128, 135, 136, 138, 139, 146, 194

RS Reverse Shuffle 131, 132, 136–141, 147

RSI Random Starting Index 60, 128–131, 136–141, 143, 146, 147

RSM Rotating S-box Masking 57, 175, 178–180, 182, 190, 194, 196, 197

RTL Register Transfer Level 86, 91

SA Stochastic Attack 50, 51, 76

SCARE Side-Channel Analysis Reverse Engineering 75

SNR Signal-To-Noise Ratio 63, 78, 118, 121

SPA Simple Power Analysis 40, 75, 98, 141

SRAM Static RAM 154–157, 160, 162, 173

SSS Sweep Swap Shuffle 132–134, 136–141, 146–148

TA Template Attack 46, 48–50, 53, 76, 117, 118, 121–124, 126, 144, 145, 178, 182, 211, 214, 221, 224

TO Transparency Order 54, 55, 62, 104–110, 117, 118

List of Figures

2.1 Types of ciphers 10
2.2 Counter mode 16
2.3 Cipher Block Chaining mode 16
2.4 Output Feedback mode 17

3.1 Scope of this work 28
3.2 Acquisition setup (scheme) 29
3.3 Acquisition setup (photo) 31
3.4 Acquisition board 32
3.5 Trigger and the power trace 35
3.6 Hamming weight leakage in ATmega328P traces 42
3.7 Hamming distance leakage in ATmega328P traces 43
3.8 Example of DoM on MSB 45
3.9 Example of CPA using HW 47
3.10 Trace points for profiling 49
3.11 Probabilities during the attack phase of TA 49

5.1 Problems with acquisitions for DPA Contest 82

6.1 Correlation for two consecutive intermediate states 101
6.2 Examples of traces produced by Silk 103
6.3 Silk workflow scheme 103
6.4 Success rate of CPA on 5×5 S-boxes using simulations 108
6.5 Success rate of CPA on 8×8 S-boxes using simulations 109
6.6 Success rate of CPA on 4×4 S-boxes using simulations 111
6.7 Success rate of CPA on S-boxes (real traces) 113
6.8 Success rate of CPA on 4×4 S-boxes and thier inverses 119
6.9 Success rate of CPA on 5×5 S-boxes and thier inverses 120
6.10 Success rate of TA on 4×4 S-boxes and their inverses 123
6.11 Success rate of TA on 5×5 S-boxes and inverses 124
6.12 Max of success rate on first and last rounds with CPA 125

6.13 Max of success rate on first and last rounds with TA 126
6.14 Structure of an index for V-RSI 129
6.15 V-RSI with 2 and 3 random bits 130
6.16 M-RS 4×4 with 4 random bits 132
6.17 SSS 4×4 visual scheme 133
6.18 Scheme of the P2-SSS 2×4 technique 133
6.19 AES-128 state representations 135
6.20 Heatmaps of permutations generated by shuffling schemes 137
6.21 Heatmap of RP shuffling scheme 138
6.22 Results of CPA against shuffling 143
6.23 Results of CPA with integration against shuffling 144
6.24 Results of TA against shuffling 145

7.1 Register and memory overwrite effects 157
7.2 Memory-based remnant effect 158
7.3 Neighbour-based leakage effect 159
7.4 Ascold workflow scheme 161
7.5 Hardened and naive S-box t-test 166

8.1 Savrasca workflow scheme 174
8.2 DoM on each bit of intermediate state 186
8.3 DoM leakage in DPA Contest 187
8.4 DoM leakage, highest points per bit 187
8.5 Distinguishers on points in `gf256mul` 188
8.6 Different DoM on MSB in DPA Contest 189

B.1 Zoom on the success rate of CPA on 8×8 S-boxes 209
B.2 Success rate of CPA on 6×4 S-boxes using simulations 210

C.1 Success rates of CPA on S-boxes. 213
C.2 Success rates of TA on S-boxes. 214

D.1 SSS and RP Heatmaps 215
D.2 V-RSI Heatmaps 216
D.3 M-RSI 4×4 Heatmaps (1 - 5 bits) 217
D.4 M-RSI 4×4 Heatmaps (6 - 10 bits) 218
D.5 M-RS 4×4 Heatmaps 219

E.1 Success rate of CPA on shuffling 222
E.2 Success rate of CPA with integration on shuffling 223
E.3 Success rate of TA on shuffling 224

List of Tables

2.1 Block ciphers key length and block length . . . 11
2.2 Properties of S-boxes . . . 13

3.1 Classification of SCA . . . 26

5.1 Sizes of datasets of DPA Contests . . . 81
5.2 Simulators for side-channel analysis . . . 92

6.1 TO and CC metrics on different S-boxes . . . 107
6.2 Properties of S-boxes (CPA, simulated HW leakage model) . . . 117
6.3 Properties of S-boxes (TA, extracted leakage model) . . . 118
6.4 M-RSI 4×4 on AES-128 . . . 131
6.5 M-RS 4×4 use on AES-128 . . . 132
6.6 P-SSS use on AES-128 . . . 134
6.7 MD-SSS use on AES-128 . . . 134
6.8 Properties of shuffling schemes . . . 140
6.9 Execution time of shuffling schemes . . . 141

7.1 Properties of implementations of Rectangle . . . 165
7.2 RNG costs for Rectangle implementations . . . 167

8.1 Binary representation of DPA Contest 4 masks . . . 179
8.2 Bias of bits in mask of DPA Contest . . . 181
8.3 Bias of bits in mask of DPA Contest (all offsets) . . . 181
8.4 Bias in combination of 4 masks . . . 182
8.5 Balanced masks (2 and 4 bytes) . . . 191
8.6 Binary view of balanced masks (2 and 4 bytes) . . . 192
8.7 Balanced masks (2, 3 and 4 bytes) . . . 193

9.1 Our new simulators for side-channel analysis . . . 204

C.1 Evolved S-boxes (TA, extracted leakage model) . . . 211

C.2 Evolved S-boxes (CPA, simulated HW leakage model) 212

F.1 Neighbours-registers in ATmega163 microcontroller 226

List of Listings

6.1 Silk example: generation of 50 simulated traces. 98
6.2 Silk example: generation of simulated noisy traces. 99
7.1 Register overwrite 156
7.2 Memory overwrite 156
7.3 Memory remnant 156
7.4 Clearing remnant 156
7.5 Neighbour leakage for registers 159
8.1 Computation of MixColumns 176
8.2 Code of `gf256mul` 176
8.3 Savrasca execution trace of `gf256mul` when MSB is 1 176
8.4 Savrasca execution trace of `gf256mul` when MSB is 0 176
A.1 Silk example: setting up the parameters. 207
F.1 XOR example with neighbour leakage 225
F.2 XOR example without neighbour leakage 225

Chapter 1

Introduction

Security plays a significant role in our society, as it became an important subject in many domains of our everyday life. Entire industries are created around this idea, they build, provide and test various security systems. Today you can rent a secure deposit box in a bank, buy a safe to store valuable items at home, install cameras to monitor your house, build a panic room and even hire a bodyguard.

Another element of modern society that developed and expanded to almost all activities of our life is the domain of computer science and Information Technology (IT). Nowadays, almost all businesses use some form of computers in their activities e.g., management of a shipments, optimisation of schedules, payment systems or even control of machines in a factory. Moreover, some of the biggest and most profitable companies are build entirely around the IT industry. We would like to avoid naming specific brands; there exist many companies that provide tools for information processing (you can buy software and even processing power i.e., rent hardware). There are many companies that help us communicate through social networks and messengers as well as by building hardware that supports the internet (routers and servers). It is even possible to buy data-storage e.g., for backups or to extend the capacity of your mobile devices (cloud storage).

In this work we are going to dive into the subject of *IT security* which is also referred to as *information security* or *computer security*. More specifically, we will focus our attention on the security of small devices that have some computing capabilities. Security of small (often portable) devices that can communicate become more and more important nowadays due to the growth of the domain called the Internet of Things (IoT). IoT is a network of small devices often connected to the Internet. Moreover, these small devices can often act on the environment e.g., turn on and off the lights, activate irrigation systems and even control water distribution and electrical grids. In addition to that, small devices get more and more computing power: a modern smartphone is more powerful that the entire computing power that was used to launch rockets to space in the second part of the 20th century. Moreover, small de-

vices that did not use to have any computing capabilities and were not connected to the network are now getting these powers: personal heart rate monitors can help to exercise and follow your progress while storing all your data to the cloud, pacemakers can be updated using wireless protocols, even cars can be automatically updated over the network. The rationale is that small pervasive computing devices are getting more widespread every day and in addition to simply processing data, they can communicate over the network and even act on the surrounding physical environment. Thus, the IT security becomes important for huge amount of systems that we use and depend on in our day-to-day life.

There are many different aspects related to the notion of IT security. Here are some examples: data or service *availability* — the idea that a service or information are available, data *authenticity* — the notion that copes with detection of data corruption (modification of an original message), *privacy* — the idea of handling and storing information in such ways that it cannot be used to reveal the identity (or other personal information) of people who generate the data, etc. In our work we will be mostly focusing on the notion of *confidentiality* — the idea of storing and transmitting information in such way that only the intended recipient can read it. Here, we use the word read in the meaning of "being able to make sense of the message". Indeed, anyone can get and record and even modify stored data, the confidentiality is usually provided through a transformation of a message in such way that it is practically impossible to get the original message without a secret information that was used during the transformation. A special type of algorithm is used within a secret value to transform an original message into its scrambled version, the algorithm itself is known by everyone, thus only the entities that have access to the secret value can get the original message.

There exist many different types of attacks against algorithms that provide confidentiality. Usually the goal of an attacker is to find out the secret value through the analysis of available information. Once the attacker gets this secret value they[1] can read all messages, create new ones and send them over the network. Some time ago, the attacker was merely able to observe messages e.g., while they transited over the network. However, with the spread of small computing devices to which attacker have physical access, it opens a door to a new type of attacks that we will discuss throughout this thesis. This type of physical attacks uses the fact that the attacker has access to the computing device and can do virtually anything that they want with the device under attack. In this work we will take a closer look on this type of attacks.

In our case, the attacker wants to extract a secret that is stored in a small portable

[1] Yes, "they", we are going to use a neutral personal pronoun (instead of he or she) throughout this work as in:
"There's not a man I meet but doth salute me
As if I were their well-acquainted friend.."
Act 4, Scene 3, *The Comedy of Errors* by W. Shakespeare.

device. Devices such as smart cards (e.g., used to access buildings or storing metro and bus tickets), car keys, etc. store and process secret values which are required to provide confidentiality. There exist many scenarios when the attacker has unrestricted physical access to the device. For example, there is a secret value embedded in a bank card[2], the bank does not want its customers to get this secret value since it will allow them to copy their bank card (which in its turn gives access to new fraud scenarios). Our ultimate goal is to prevent these attacks from happening. A very similar example is the SIM card in your phone.

Usually, before the device is released on the marked it is tested to check that it behaves as intended. This is also the way that devices are evaluated with respect to their security. In other words, security oriented devices (that are supposed to provide e.g., confidentiality) are evaluated with respect to their robustness against different types of attacks, including the physical attacks that we are interested in. Unfortunately, evaluation methods that work well (and thus that are used nowadays) require the final product in order to test it. However, finding out that a device has a security flaw at the final stage of development is far from ideal: it means that the developer has to redesign some parts of the system (which also creates additional costs). Thus, developers want to be able to use some form of continuous evaluation of their products against physical attacks.

Summing up, in this work we will take a look on small hardware that has to store and process secret information while focusing on the special type of attacks that take advantage of the physical access to the attacked device. As a concrete example of physical properties that can be used to break the security, we will use techniques based on the analysis of power consumption. Our main goal is to provide tools that can help designers of secure devices in the process of hunting security flaws during the process of development. The idea consists in detecting security related issues at different stages of development in order to reduce the risk of creating an insecure product and finding out that it has flaws at the very last stage. The main principle that we advocate consists in the use of automated tools that can simulate the behaviour of the device given the information available during the development (such as the source code of a program). Thus, in this work we will present tools that can simulate the execution of cryptographic algorithms and produce the simulated data about the power consumption of the device under evaluation. This information is generally available only when the device is already built, thus having a tool that can produce a simulated version while the device is still under construction is beneficial since it allows to perform same type of security evaluation before having the final version of the product. As a result, simulations allow to detect security flaws in the code and correct the implementation on an earlier stage of development. The rationale is that the use of the presented tools should decrease the production costs of secure devices

[2]We are not referring to the pin code. The pin code is used to "activate" the card which in its turn uses the secret value.

and improve their overall security against physical attacks based on the analysis of power consumption.

This document is structured as follows. The first part of the text gives an introduction to the subject of this work. Chapter 2 presents the main ideas of cryptography and how it can provide data confidentiality using encryption. Chapter 3 gives an overview of physical attacks that we focus on and explains how this type of attack works, it also introduces the idea of power analysis — a specific subtype of physical attacks that we consider in this work. Chapter 4 presents the problem that security evaluators face while testing the final product, it also presents the motivation for continuous security evaluation during development. The second part of this document presents our contributions. Chapter 5 motivates the use of simulators in the type of analysis we are interested in, it also introduces related works by presenting an overview of existing simulators and listing their strengths and weaknesses. Chapter 6 presents our first high-level of abstraction simulator of dynamic data-dependant part of the power consumption, shows how to use it on several different use-cases typical for this domain and concludes by highlighting the usefulness of simulators due to their high speed compared to physical experiments. Chapter 7 introduces several problems related to the implementation of countermeasures against physical attacks. It presents an automated tool that can help in detecting the discussed implementation issues during the development of the code. The presented tool can pinpoint the line of code that causes the information leakage with an explanation of what causes the leakage, this idea highlights that the analysed implementation issues can be avoided. Chapter 8 introduces another tool that can be used at one of the final stages of development of secure systems. This tool allows to perform simulations of data-dependant part of the power consumption for subsequent leakage detection. The presented tool is shown to be useful in tracking implementation flaws that lead to unintentional leakage of secret information through power consumption. This chapter emphasises the fact that a lot of security related issues can be detected before actually performing real experiments on the physical product. Finally, Chapter 9 sums up the whole work, provides some conclusions and presents a list of open problems and directions for future works.

Part I

Preliminary notions

Chapter 2

Cryptography

Cryptography is the foundation of modern IT security. The word cryptography comes from Greek κρυπτος [kryptós] which means *hidden* or *secret* and γραφειν [gráphein] that translates as *to write*. At its birth, cryptography was used in order to hide secret messages from enemies and opponents, in other words the main goal of cryptography was to provide confidentiality.

Modern cryptography encompasses more concepts, in addition to confidentiality it can provide properties such as message integrity, non-repudiation of authorship and authentication. Nowadays cryptography could be defined as a study of techniques that could be used for secure communications in presence of an adversary.

The goal of cryptography is to create basic building blocks that are called *cryptographic primitives* or *cryptographic algorithms*. A cryptographic primitive is an algorithm that can be used in order to provide one or several basic security properties. For example, encryption algorithms provide message confidentiality, while digital signature algorithms provide authentication, integrity and non-repudiation. Cryptographic primitives are used together to build cryptographic protocols.

Cryptography has a fellow traveller called *cryptanalysis*, together they form cryptology. While cryptography is the art of building cryptographic algorithms and protocols, cryptanalysis is the art of analysing and breaking them. Trying to break a cryptographic algorithm might seem as a counter-intuitive idea. However, every result that comes from cryptanalysis can be used to improve existing schemes (algorithms and protocols) as well as to build better cryptographic schemes in future. Thus, people who build and create cryptographic schemes are also the ones who break them. These people are often called cryptographers.

Up until the middle of the twenties century cryptographers mostly based their developments on the idea of *security through obscurity* i.e., all cryptographic schemes were designed in secret and detail about their construction were kept hidden from public. Nowadays, most of cryptographers rely on the idea called *Kerckhoffs's principle*. Kerckhoffs's principle states that a cryptographic system (or *cryptosystem*) has to

be secure even if every single detail about its construction, structure and functioning, except the *secret key*, is public knowledge. Secret key is the name given to a number that is used as one of the parameters of a cryptographic algorithm. This number is a piece of valuable information that determines the output of a cryptographic algorithm. The entire security of a cryptosystem relies on the secrecy of a key. Thus, it is the only piece of information that has to be kept secret. This principle is very useful in practice and security experts tend to apply it[1], since keys are generally smaller that an algorithm and thus are easier to protect (keep secret) and also easier to change in case when a key is compromised. The area of cryptography that studies how to generate, exchange and keep secret keys is called key management.

Some cryptographic primitives do not use a secret key, for example, this is the case of *hash functions*. Nevertheless, most of cryptographic algorithms such as *digital signatures*, *hash-based message authentication codes* and *ciphers* rely on a secret key in order to provide security. The results of this work could be applied to cryptosystems that use a secret key as one of its parameters. However, for the sake of simplicity in order to better illustrate this work we will focus on ciphers.

2.1 Ciphers

A cipher is a scheme that provides data confidentiality. Thus, ciphers are mainly used in order to send secret messages. In order to provide confidentiality ciphers rely on a *secret key*.

A cipher consists of three algorithms: a *key generation algorithm*, *an encryption algorithm* and a *decryption algorithm*. A key generation algorithm uses a random number and produces a key that could be used with the cipher. An encryption algorithm takes a key and a message as its inputs and produces an *encrypted message* called a *ciphertext*. The input message of an encryption algorithm is called *plaintext* or *cleartext*. The decryption is the inverse of the encryption, a decryption algorithm uses a key and a ciphertext in order to produce the original plaintext.

The ciphertext is usually the message that transits through a network or that is stored on a disk. Thus, a ciphertext is a publicly known information or in other words ciphertexts are considered to be in a hostile environment i.e., they are being observed and handled by a potential *attacker*. In case of ciphers, an attacker is an entity (a person or an organisation) that tries to find the secret key which is used in a cipher.

More formally we will specify a cipher using the notation $\langle \mathsf{G}, \mathsf{E}, \mathsf{D} \rangle$, where G, E and D denote respectively the key generation, the encryption and the decryption

[1] Nevertheless engineers do not always follow the Kerckhoffs's principle. Secret algorithms and protocols developed by companies are often based on bad design. Keeloq [IKD+08] and Mifare smart card [GdKGM+08, dKGHG08] are among relatively recent examples that were broken shortly after the discovery of the algorithm.

algorithms:

$$\begin{cases} \mathsf{G} : \mathcal{R} \to \mathcal{K} \\ \mathsf{E} : \mathcal{P} \times \mathcal{K} \to \mathcal{C} \\ \mathsf{D} : \mathcal{C} \times \mathcal{K} \to \mathcal{P} \end{cases} \tag{2.1}$$

Where $\mathcal{R} \subseteq \{0,1\}^*$ is a set of random numbers, $\mathcal{K} \subseteq \{0,1\}^*$ is a set of keys, $\mathcal{P} \subseteq \{0,1\}^*$ is a set of plaintexts and $\mathcal{C} \subseteq \{0,1\}^*$ is a set of ciphertexts. In other words, random numbers, keys, plaintexts and ciphertexts are all binary strings.

We will use the notation $c = \mathsf{E}_k(p)$ to denote that the message $p \in \mathcal{P}$ is encrypted using the key $k \in \mathcal{K}$, which produces the ciphertext $c \in \mathcal{C}$. The decryption of a ciphertext c using the key k is noted using the notation $p = \mathsf{D}_k(c)$. Note that $p = \mathsf{D}_k(\mathsf{E}_k(p))$.

There are two big families of ciphers, these families are referred as *symmetric encryption* and *asymmetric encryption* (which is usually called *public key encryption*). The main difference between these two families of ciphers is the following. A symmetric encryption scheme uses one secret key and both communicating parties have to know it in order to exchange encrypted messages (and be able to read them), in other words, the same secret key is used for the encryption and for the decryption. A public key encryption scheme uses two linked keys: a secret key (called a *private key* in public key cryptography) and a *public key*. The public key is known by everyone and anyone can use it, while the private key is meant to be kept secret by its owner. The public key is used in order to encrypt messages and the private key is used to decrypt them. The public and the private keys are generated together by the key generation algorithm. A public key encryption algorithm is usually based on a hard mathematical problem such as e.g., *integer factorisation* [RSA78] or *discrete logarithm* [DH76]. In this case, by hard problem we mean a problem such that today an efficient algorithm that can solve it (in a general case) is unknown and best known solutions use exhaustive search.

Due to the way the two families of algorithms behave and due to their properties, generally public key ciphers require longer keys and they are often slower than symmetric ciphers for the same offered level of security. However, due to the fact that they have two separate keys (the public and the private ones), they offer some advantages (a person can distribute the public key without revealing their secret key) that cannot be achieved by using only one secret key[2].

This work focuses on symmetric encryption. There exist two types of symmetric ciphers: *block ciphers* and *stream ciphers*. A stream cipher uses the secret key to

[2]One of the latest trends in cryptography, called *"white-box cryptography"* [CEJvO02], tries to create a system that allows to distribute an implementation of a symmetric algorithm with an embedded secret key (encoded in such way that it is difficult to extract). This promising approach is in development nowadays [BIT16] and it also pushes cryptographers to invent new types of analysis [BHMT16].

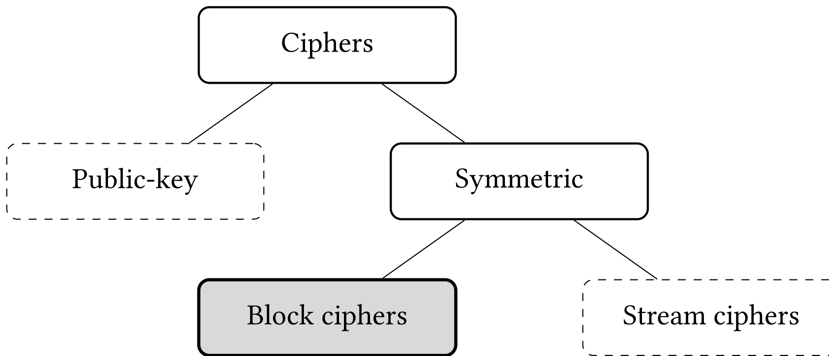


Figure 2.1 – Family tree of types of ciphers and the position of block ciphers in it.

generate a long *pseudorandom* sequence of bits called *keystream*. The keystream is combined with the plaintext bit-by-bit using a binary exclusive-or (xor) operation, the result of this combination is the ciphertext. Stream ciphers are very interesting as a research subject, however the main target of this work are block ciphers.

2.1.1 Block ciphers

A block cipher is an encryption scheme from the family of symmetric cryptographic algorithms, see Figure 2.1. One of the main properties of block ciphers is that a block cipher deals with short fixed length messages called *blocks* i.e., a block cipher encrypts a fixed length plaintext and produces a fixed length ciphertext (same idea also applies to the decryption). Block ciphers also use fixed length secret keys. Thus, for a block cipher, we have:

$$\mathsf{E} : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^m. \tag{2.2}$$

Where m is the size of the block and n is the size of the key. See Table 2.1 for typical block length and key length of block ciphers. When a block cipher can be used with different key sizes and one wants to refer to a specific version of the cipher the key size is noted after the name of the cipher with a dash in-between e.g., AES-128 or Present-80.

Generally, block ciphers can use any fixed length value as the secret key. However, some implementations of DES (Data Encryption Standard) block cipher [DES77] use several key bits for parity check and several block ciphers such as IDEA [LM90] are known to have *weak keys* [DGV93b]. A weak key is a key value that decreases the security of a cipher when such key is used during encryption i.e., it is easier to break a cipher (for example, to find the encryption key or to find the cleartext) that was used with a weak key compared to a non-weak key.

Table 2.1 – Typical block ciphers key length and block length (in bits). Notation a/b means that the algorithm works with values a and b.

Cipher	Year	Key length	Block length	Reference
DES*	1977	56	64	[DES77]
IDEA	1990	128	64	[LM90]
RC6	1998	128/192/256	128	[RRSY98]
Serpent	1998	128/192/256	128	[ABK98]
Twofish	1998	128/192/256	128	[SKW$^+$98]
AES	2001	128/192/256	128	[AES01]
PRESENT	2007	80/128	64	[BKL$^+$07]
Joltik	2014	64/80/96/128	64	[JNP15]
SCREAM	2014	128	128	[GLS$^+$15]

* Additional 8 bits are used for parity checks, resulting in a total of 64 bits.

Modern block ciphers are composed of three types of basic operations: *permutations*, *substitutions* and *key additions*. Permutations change the order of bits, they spread information inside of the block. Substitutions replace one value by another one, these operations are often implemented using *S-boxes* (or *Substitution-boxes*). Key additions mix the secret key and the plaintext; the most common operation that is used as a key addition by the majority of block ciphers is an exclusive-or (also called xor, noted $\oplus$) but some ciphers such as Kalyna [OGK$^+$15, OGDR10] also use modular addition ($\boxplus$) or even modular multiplications (used by IDEA [LM90], noted $\odot$).

We will use the notation S to denote an S-box. Let $\mathbb{F}_2^n$ be the vector space that contains all the n-bit binary vectors, then an S-box is defined as:

$$S : \mathbb{F}_2^m \to \mathbb{F}_2^n \tag{2.3}$$

where m and n are the input and the output sizes of the function S, in this case the S-box is said to be an $m \times n$ S-box (usually pronounced "an m by n S-box") also called (m, n)-function. An S-box can be seen as a vector of n boolean functions $[S_1, S_2, ..., S_n]$ where each function S_i is a mapping:

$$S_i : \mathbb{F}_2^m \to \mathbb{F}_2. \tag{2.4}$$

In other words, each function S_i represents a function that gives the ith bit of the output of the S-box S. For a given function F, we are going to use the term *coordinate function* to denote each boolean function F_i which specifies the ith bit of the output of the function F.

Typical input and output sizes of an S-box are on the order of several bits. Most of the candidates to an ongoing (at the time of writing) cryptographic competition

for authenticated encryption schemes[3] use S-boxes of size 4×4, 5×5 and 8×8; the Data Encryption Standard (DES) block cipher uses 6×4 S-boxes and Advanced Encryption Standard (AES) uses 8×8 S-boxes.

Permutations and substitutions are often called *linear* and *nonlinear* parts (or layers) of the cipher. This terminology refers the linear and nonlinear functions. The *nonlinearity* $\mathcal{N}_F$ of an (m, n)-function F is equal to the minimum nonlinearity of all non-zero linear combinations $v \cdot F$, with $v \neq 0$, of its coordinate functions F_i, i.e.:

$$\mathcal{N}_F = 2^{n-1} - \frac{1}{2} \max_{\substack{a \in \mathbb{F}_2^m \\ v \in \mathbb{F}_2^{n*}}} |WH_F(a, v)|, \tag{2.5}$$

where $|\cdot|$ is the absolute value and $WH_F(a, v)$ represents the Walsh-Hadamard transform of F that is equal to:

$$WH_F(a, v) = \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) + a \cdot x}. \tag{2.6}$$

Intuitively, the nonlinearity of a function is equal to 2^{n-1} minus its highest linearity (or its best linear approximation) of any of its coordinate functions. High nonlinearity of a function F represents the fact that any linear combination of F and its input x is more often different (than equal) from the output of F. For example, when F equals 0 more linear combinations (of all possible combinations) of F and x are equal to 1, which creates a bias. In other words, a highly nonlinear function will not match any linear combination of its inputs.

Another important property that is often used in cryptography in order to describe S-boxes is called *differential uniformity*. It describes the behaviour of a function when a specific difference is added to its input, in other words it says what output difference we can observe if we know the difference between two inputs. More formally, we define the differential uniformity δ_F of a function F as follows:

$$\delta_F = \max_{a \neq 0, b} \|\mathsf{D}(a, b)\| \tag{2.7}$$

where $\|\cdot\|$ is the cardinality of a set, F is a function that maps $\mathbb{F}_2^n$ into $\mathbb{F}_2^n$ and $a, b \in \mathbb{F}_2^n$. The set $\mathsf{D}(a, b)$ is defined by the equation:

$$\mathsf{D}(a, b) = \{x \in \mathbb{F}_2^n : F(x + a) + F(x) = b\}. \tag{2.8}$$

Table 2.2 gives values of nonlinearity and differential uniformity for several S-boxes that we use in this work.

Permutations, substitutions and key additions are applied on the *state* of the block cipher. The state of the block cipher, also called *internal state*, is the intermediate

[3]Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR), `http://competitions.cr.yp.to/caesar.html`

Table 2.2 – Nonlinearity and differential uniformity of some S-boxes.

Size	Name	$\mathcal{N}_F$	δ_F	Reference
8×8	AES	112	4	[AES01]
	AES_{CC}	112	4	[PPE+14]
	SCREAM*	96	8	[GLS+15]
	STRIBOB	100	8	[SB15]
6×4	DES_1	18	16	[DES77]
	DES_2	18	16	
	DES_3	18	16	
	DES_4	22	16	
	DES_5	18	16	
	DES_6	20	16	
	DES_7	14	16	
	DES_8	20	16	
5×5	Ascon	8	8	[DEMS15]
	ICEPOLE	8	8	[MGH+15]
	Keccak	8	8	[BDPA11]
	PRIMATE	12	2	[ABB+14]
	SC2000	12	2	[SYY+01]
4×4	$\text{Evolved}_{\text{TO}}$	4	4	[PMMB15]
	$\text{Evolved}_{\text{CC}}$	4	4	[PPE+14]
	Joltik	4	4	[JNP15]
	Klein	4	4	[GNL11]
	Minalpher	4	4	[STA+15]
	Present	4	4	[BKL+07]
	PRINCE	4	4	[BCG+12]
	Prøst	4	4	[KLL+14]
	Rectangle	4	4	[ZBL+15]

* The algorithm SCREAM was updates several times, one of the modifications concerned its S-box, we are only dealing with the 3rd (latest) version of the algorithm.

value of the encryption algorithm. In case of block ciphers the state usually starts as the plaintext and the final state at the end of the encryption is the ciphertext. The described operations are applied repeatedly on the state of the cipher during several *rounds*. A round is the name that is used by cryptographers to denote one cycle of a transformation that is applied to the state of the algorithm.

Just one basic operation (a permutation, a substitution or a key addition) does not provide any security by itself. However, combined together they provide *diffusion* and *confusion* properties [Sha45]. Diffusion refers to the idea that every bit of the ciphertext should depend on the value of every bit of the key and of the plaintext; it is often stated that if any single bit of the plaintext is flipped then every bit of the ciphertext flips with probability 0.5, in other words statistical properties of the plaintext are dissipated in the ciphertext. Confusion property refers to the idea that the relationship between the secret key and the ciphertext should be as complex as possible. It is often said, that permutations help to provide diffusion, while substitutions provide confusion. Three common ways of combining permutations, substitutions and key additions are often used in order to construct a round of a block cipher. They are called *Substitution-Permutation Network* (SPN), *Feistel network* [Fei73] and *Lai-Massey* [LM90]. We invite the interested reader to consult the book *"Introduction to modern cryptography"* for more information on these schemes [KL08].

Another important part of a block cipher is a *key schedule* also called a *key scheduling algorithm*. A key schedule uses the secret key as its input and produces several sub-keys, each of the sub-keys is called a *round-key*. A round-key is a value that is used as the key during the key addition operation at each round of a block cipher. The key scheduling algorithm mixes the bits of the secret key in order to generate different round-keys using substitutions and permutations. The designers often reuse substitution and permutation operations in the key schedule in order to create an algorithm that could be implemented very efficiently in hardware as well as in software. One of the main goals of adding many round-keys is to make any cryptanalysis more difficult. Most of modern block ciphers start and end by a key addition operation i.e., the first operation of the majority of modern block ciphers mixes the first round-key and the plaintext and the last operation of most of block ciphers mixes the intermediate state of the cipher with the last round-key which gives the ciphertext. This key addition is often called a *key whitening* and it plays an important role against classical cryptanalysis [Sch07]. The idea behind it is relatively simple, if the first operation of a block cipher does not involve a secret value then an attacker can execute a part of the algorithm (till the first use of the secret) without knowing the secret key, same reasoning applies to the last key addition.

In this document, we will often use the Advanced Encryption Standard (AES, also known as Rijndael [AES01, DR02]) block cipher as an example and a study-case. AES is an SPN cipher, it handles blocks of 128 bits, its internal state is represented as a 4×4 matrix of 16 bytes (4 rows and 4 columns). We will specify more details about

the internals of AES in the text at the moment when we will use them. We will refer to some of its internal operations that are named as follows:

- AddRoundKey — the key addition that uses an exclusive-or,
- SubBytes — the application of an 8×8 S-box on every byte of the state (the nonlinear layer of AES),
- ShiftRows — permutation of bytes in each row,
- MixColumns — combination of 4 bytes of a column (creates new values for all the 4 bytes of the column based on its previous values).

Even though a block cipher can encrypt only fixed length messages, there is a way to deal with plaintexts that are shorter or longer than the block size. If the plaintext is shorter than one block then a *padding* is added to it [MVOV96]. If the plaintext is longer than one block then it is divided into blocks and each block is encrypted separately (potentially with a padding in the last block). There are several ways of handling multiple blocks of long messages, a specific way of dealing with longer messages is called a *mode of operation*.

There exist many different modes of operation on block cipher, but we would like to focus our attention to three examples in order to highlight the difference in the way inputs and outputs are handled and what is actually sent through an insecure channel. These differences will become important later in Section 3.2.2. Let's take a look at *Counter* mode (CTR) [DH79, LWR00], *Cipher Block Chaining* mode (CBC) [EMST78] and *Output Feedback* mode (OFB) [FIP80].

In CTR mode the sender chooses a *nonce* which is concatenated with a counter in order to be used as an input in a block cipher. A nonce is a *number used once*, it is transmitted with the encrypted message, however an attacker should not be able to predict this number. The result of the encryption is combined with the secret message using a bitwise xor. Before encrypting the next block, the counter is incremented and the whole procedure repeats, see Figure 2.2.

During the CBC mode each block of the secret message is combined with the ciphertext given by the previous block using a bitwise xor, the result is the input of the encryption algorithm. In order to bootstrap the encryption for the first block, the first part of the secret message is combined with an *Initialisation Vector* (IV). An initialisation vector is a fixed size random value that an attacker can observe (but should not be able to predict). The scheme for CBC encryption is shown in Figure 2.3.

The OFB mode also requires an IV in order to start the encryption process with the first block. However, in the OFB mode the output of the block cipher is combined with a block of a secret message and it is also used as an input of a block cipher for the next encryption, see Figure 2.4.

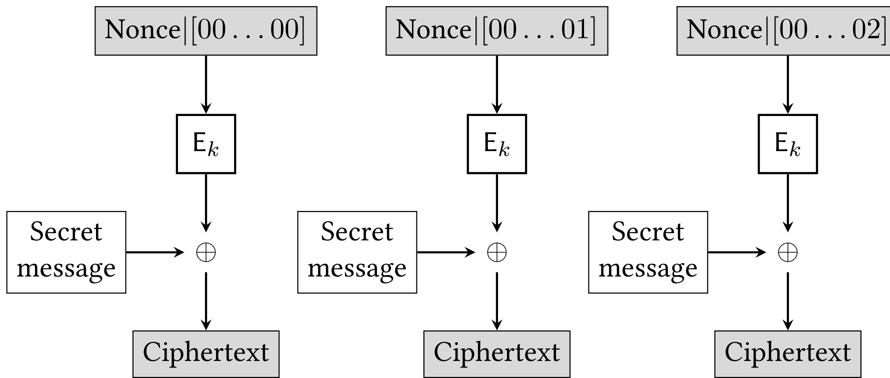


Figure 2.2 – Block cipher used with the Counter mode (CTR).

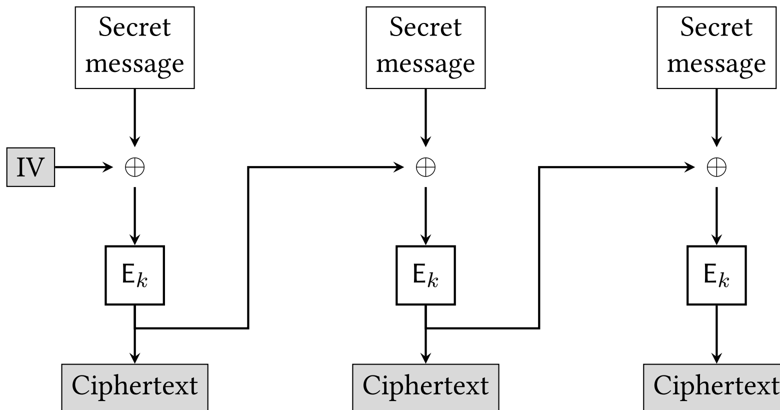


Figure 2.3 – Block cipher used with the Cipher Block Chaining mode (CBC).

In Figures 2.2, 2.3 and 2.4 the information that is observed by an attacker is highlighted in grey and the secret message that a sender wants to transmit is shown in white. By looking at these schemes we can notice that in CBC mode the attacker has access to the *output* of the block cipher algorithm. In CTR mode the attacker knows the *input* of the block cipher (but does not know the output). Finally, in case of OFB mode, the attacker *does not know the input nor the output* of the block cipher unless the attacker discovers the plaintext that correspond to the observed ciphertext (the only exception is the IV i.e., the input of the first encryption which is always known).

Most of the time we are going to focus on the analysis of one block at a time, thus we will be interested in modes only from the point of view of the knowledge of an adversary (known input or known output). For the sake of simplicity, through the rest of the document we are going to use the term *plaintext* to refer to the *input* of a block cipher and *ciphertext* to denote the *output* of a block cipher.

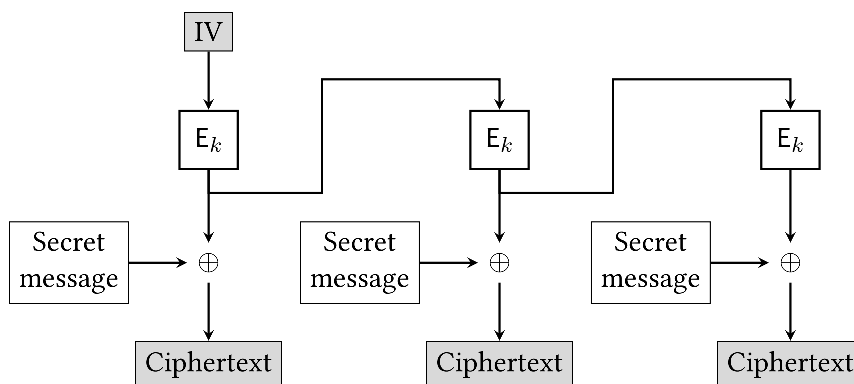


Figure 2.4 – Block cipher used with the Output Feedback mode (OFB).

2.1.2 Attacks on block ciphers

Attacks on block ciphers (as well as on other cryptographic primitives) help cryptographers to study block ciphers, it ultimately helps to develop new, better block ciphers. In cryptography an *attacker* is often called an *adversary*. The goal of an adversary is to find out the secret that communicating parties do not want to reveal. The adversary tries to achieve this goal by analysing publicly available information. In case of block ciphers, algorithms and all the ciphertexts are publicly available, while plaintexts and keys are secret. An attacker, might be only interested in finding a plaintext; however, their ultimate (and most desired) goal is to find the secret key, because a secret key gives an attacker two abilities: (1) decrypting ciphertexts (i.e., getting plaintexts) and (2) creating new ciphertexts. Thus, most interesting and powerful attacks are targeting the secret key.

There exist many different types of attacks on block ciphers. These attacks are often grouped by scenario, which also refers to the capabilities that an adversary can use to their advantage:

- *ciphertext only* — the adversary can observe one or several ciphertexts [MS01],
- *known plaintext* — the adversary can observe a set of pairs: plaintexts *and* corresponding ciphertexts (this type of attacks was used against Enigma during World War II [Sin00]),
- *chosen plaintext (ciphertext)* — the adversary can choose the values of plaintexts (or ciphertexts) and get their encrypted (or decrypted) versions [BK98],
- *adaptive chosen-plaintext (ciphertext)* — the idea resembles the previous one, but the attacker can adapt the values of plaintexts (or ciphertexts) during the attack based on previous results [Ble98],

- *related-key* — the attacker knows that the device uses several different keys unknown to the attacker, however the attacker knows a relationship between those keys e.g., they have a common (identical) part [BE01].

The idea of a known plaintext (and its corresponding ciphertext) might seem a little bit counter-intuitive, but there are a lot of real-life scenarios where it might be the case. For example, a document might be secret while it is being prepared and discussed, but it is supposed to become public knowledge when it is ready. Also, some secret “classified” documents are declassified (their content becomes publicly known) after some time, either deliberately (some governments reveal secret documents decades after their creation [Fri07]) or unintentionally (an agent that handles confidential information might decide to reveal it or a server containing secret information can be compromised [Sch15]).

There is another way of grouping the attacks on block ciphers, this time groups are based on a particular technique (a type of cryptanalysis) that is used in order to analyse the algorithm and the outputs that it produces. We would like to mention tree main types of cryptanalysis[4]:

- *meet in the middle* — the idea is to follow the algorithm from the beginning to the end and from the end towards the beginning at the same time while making hypotheses on the key and checking if the two parts correspond or “meet” in the middle of the algorithm (at its intermediate state) [DH77],
- *differential cryptanalysis* — the goal is to analyse differences (and their frequencies) in ciphertexts depending on differences between plaintexts (the difference often refers to a bitwise exclusive-or operation) [BS91, BS90],
- *linear cryptanalysis* — the idea is to try to find a linear part within a nonlinear part of the cipher and use it in order to “linearise” the cipher (finding linear function that approximates it) [MY92].

Most of cryptanalysis starts as an analysis of one round of a block cipher or even as an analysis of one of its operations (e.g., an S-box). After this kind of bootstrap the attack is extended to more rounds in order to reach the full cipher. If a successful attack is discovered then the cipher is considered to be *broken*. More precisely, an attack is considered to be successful if it can be done with less resources than a *generic attack*. For a block cipher, a generic attack is an *exhaustive key search* which is also called a *brute-force attack*. The idea behind an exhaustive key search is very simple: an attacker has to test every single value of the secret key and execute the algorithm

[4]These families of cryptanalysis are subdivided into more specific techniques, sometimes attacks also use combinations of ideas, so the frontier between these types of cryptanalysis can sometimes be fuzzy.

while checking its output. The only way of protecting a block cipher against such attack is to use a large key i.e., the number of different values of the secret key should be big enough so it is practically impossible to test every one of them. A potential attacker could still try to enumerate all the keys, but the idea is to make this task extremely impractical and to discourage the attacker.

In order to measure the strength of an attack cryptographers use the *amount of resources* that the attack requires, this idea is also referred as the *complexity of an attack*:

- *data* — refers to the number of queries that an attacker has to make i.e., it is the number of messages (ciphertexts or plaintext-ciphertext pairs) that an adversary has to observe in order to perform the attack,
- *memory* — is the amount of storage that an adversary needs in order to perform the attack,
- *computational power* — refers to the amount of time that an attacker needs in order to mount the attack, in practice an attacker can trade physical space for time (use many parallel units in order to perform the attack faster).

In some cases an attacker can trade one part for another (e.g., use less memory but more computational power), but generally the total complexity does not change dramatically for a given specific attack. Sometimes, we may also talk about *online* and *offline* complexity of an attack i.e., an attack might allow (or require) to compute some values before even observing a single encrypted message.

It is interesting to note, that even a small improvement on the exhaustive key search is considered to break a cipher from the theoretical point of view. However, it does not mean that any theoretical attack is feasible in practice, it might still require too much resources.

A lot of general ideas on the attacks on block ciphers can be applied to any block cipher. However, each new attack on a specific block cipher can use cipher-specific approach and the ingenuity of an attack is often limited only by the imagination of an attacker. New people and new ideas are the main sources for new attacks and new types of cryptanalysis on block ciphers. During this work, we will be interested in a special type of attacks, called side-channel attacks.

2.2 Summary

Cryptography serves as a foundation of modern security. Basic building blocks of cryptography provide security properties such as confidentiality and authenticity. Ciphers are the building blocks that provide confidentiality, their security is based on a secret value called key. Block cipher is a type of cipher from the family of symmetric ciphers, it deals with messages of fixed size called blocks. Such cipher encrypts one block of the message in order to produce a block of the ciphertext.

A block cipher is composed of several basic operations (permutations, substitutions and key additions) that do not provide the security on their own, but can provide security if they are applied in conjunction. These operations are applied on the plaintext during several rounds in order to produce the ciphertext. Cryptographers study these operations one-by-one as well as in conjunction using statistics and properties such as linearity and nonlinearity.

Cryptanalysis is the art of attacking or breaking block ciphers as well as other cryptographic primitives. There exist many different attacks on block ciphers. The main goal of an attacker is to find out the secret key. Cryptographers build block cipher and also try to break them in order to verify if the cipher is secure and also to understand how to build better block ciphers in future. A block cipher is considered to be broken if there is an attack that can break it using less resources than an exhaustive key search. There exist many types of attacks on block ciphers, they are grouped by the capabilities of the attackers (ciphertext-based, known plaintext and some others) as well as by the type of analysis (meet in the middle, differential and linear).

For more information about different types of attacks, about ciphers as well as about other cryptographic algorithms and protocols we refer interested readers to the book called *"Introduction to modern cryptography"* [KL08] and also to *"Applied cryptography"* [Sch07].

Chapter 3

Side-channel analysis

A side-channel attack is a type of attack on cryptographic systems. A side-channel attack takes into account the *physical properties* of a cryptographic device. Side-channel analysis is relatively new compared to classical cryptanalysis. First references to frequency analysis (that gave birth to classical cryptanalysis) of Caesar cipher by Abu al-Kindi date back to the 9th century [Sin00], even more modern cryptanalysis of Enigma machine was made in 1930s by Marian Rejewski and later his colleagues from the Polish Cipher Bureau [Koz84]. On the other hand, side-channel analysis was discovered in Bell Labs in 1943 and was kept secret for a very long time [Fri07] (NSA secret document published in 1972 was declassified only in 2007), first appearance of side-channel analysis in scientific papers dates back to 1996 [Koc96].

One of the main hypothesis in classical cryptanalysis states that an attacker can observe (and sometimes interact with) the input and the output of the algorithm i.e., the plaintext and the ciphertext in case of a cipher. Classical cryptanalysis of good block ciphers is hard because cryptographers take it into account while designing a block cipher. One of the key components that makes it difficult is a repetition of a round function. While it may be easy to break one round of a block cipher, it is increasingly difficult to break every additional round of the block cipher. Even the knowledge about the value of a *single bit* of *any* intermediate state of a block cipher does lead to an attack that can extract the secret key from the device more efficiently than classical cryptanalysis [BI15]. Thus, if an adversary were to learn an intermediate value, lets say the result of the computation of one round, they will be able to break the encryption and extract the secret key.

In classical cryptanalysis, the goal of the attacker is to find the secret key by analysing the plaintext, the ciphertext and the algorithm. In this kind of setting the analysed algorithm is treated as a function that maps a plaintext to a ciphertext. In other words, a block cipher is modelled as an abstract mathematical object. However, when a block cipher is implemented in software or in hardware the final product is more than just a function, it is not an abstraction any more. A physical cryptographic

device consumes energy, it takes time to compute the result, it dissipates heat and it has many other physical characteristics that can all be measured by an attacker. These properties are not present in an algorithm modelled "as a function" and these characteristics can unintentionally leak information about the internal state of the algorithm and ultimately about the secret key. Side-channel attacks try to extract information about an intermediate state of an algorithm by measuring physical characteristics of a cryptographic device. In other words, a side-channel attack is an attack on the *device* and on the *implementation* of a cryptographic algorithm rather than on the abstract algorithm itself. During a side-channel attack the secret key is extracted from the device that uses it.

Side-channel attacks are also referred as *physical attacks*, since the adversary measures physical properties of a device under attack and often has the physical access to the device itself. At the same time, side-channel attacks are often put in the category of a *grey-box* scenario of attack. This classification comes from the idea of a *black-box* analysis on one hand, where the attacker can only observe and analyse the inputs and outputs of a function (i.e., classical cryptanalysis) and the *white-box* cryptography on the other hand, where the code of a cryptographic algorithm is executed on a device that is fully controlled by an attacker (the adversary can run, stop and modify the code as well as record the state of the system at any moment in time).

3.1 Types of side-channel attacks

Side-channel attacks come in different types and flavours and they can be grouped differently by looking at them from several points of view.

3.1.1 Information channel

During a side-channel attack, the adversary can exploit all kinds of leakages that they can acquire by measuring various physical characteristics of cryptographic devices. These characteristics are measured during the execution of the cryptographic algorithm. From this perspective, side-channel attacks differ by the *channel* or the *source of information* that allow the attacker to learn information about the secret key. Side-channel attacks are often named after the source of information that is measured during the attack:

- *Timing attack* — the execution time of an algorithm is measured in order to infer the information about the secret key, this type of side-channel attacks was one of the first presented to the scientific community in 1996 [Koc96], execution time of an algorithm may vary due to unbalanced branches in the control flow of the algorithm or even due to the increase of the speed in data access thanks to the cache memory, timing attacks are the main reason why cryptographic algorithms and protocols should always execute in constant time;

- *Power analysis* — the instantaneous power consumption is measured during the execution of the algorithm (using an oscilloscope) [KJJ99], instantaneous power consumption changes depending on how many (and which) logic gates are being switched on and off at each moment in time, among other things it depends on the instruction that is executed as well as on the data that is processed;
- *Electro-Magnetic Analysis (EMA)* — electro-magnetic waves are measured using a special probe (which acts as an antenna) while the device executes the cryptographic algorithm [AARR02], this attack is similar to the power analysis but it allows to take measures on a specific part of the chip e.g., memory, bus, Arithmetic Logic Unit (ALU);
- *Photonic emission* — the device is photographed in a dark room while it executes a cryptographic algorithm, the idea is based on the fact that there is a non-negligible chance of a photon being emitted when a transistor changes its state [SNK+12], the device is usually depackaged in order to reveal the silicon layer inside of the chip;
- *Acoustic cryptanalysis* — the sound produced by the device (e.g., its cooling fans or its spinning disks of the hard-drive) is recorded using a microphone, depending on the switching activity (intensity of computations) and memory accesses (their location and frequency) the resulting sound will have different patterns [GST14], these patterns can be analysed in order to reveal information about data being processed by the device.

Those are not the only sources that were used to extract the information about the processed data form the device. More exotic channels could be used e.g., the orientation of a smartphone and its subtle changes in order to find out what text is being entered by the user [CC11]. Another slightly more exotic power analysis technique was tested on smartphones in order to find out their location [MSV+15], the idea is based on the fact that a smartphone will drain a different amount of energy from its battery depending on the strength of the cellular signal.

Making measures on different channels of information require special equipment. Thus, these attacks vary in the setup complexity and in the budget depending on the type of equipment and its cost. Timing attacks require a computer to measure the time accurately e.g., between a request and the corresponding response. Power analysis are also among cheapest attacks, while photonic emission analysis requires a more expensive equipment: one of the cheapest hardware sets that can be used for power analysis costs several hundred dollars[1], the cheapest setup for simple photonic emission analysis can be acquired for *"approximately the price of a mid-range oscilloscope"* [SNK+13] (the equipment that this team has in the lab costs about 20 000 € and

[1]See ChipWhisperer by NewAE Technology Inc. `https://newae.com/tools/chipwhisperer/`

the cheapest working demo setup, that they were able to build, costs about 5 000 €). This work focuses on power analysis, which is one of the cheapest, powerful and wide-spread type of side-channel attacks.

It is interesting to note, that power analysis, electro-magnetic[2] analysis and acoustic cryptanalysis are very similar in terms of the type of data that is produced by the measurements. All three types of attacks analyse *temporal series* i.e., ordered lists of values. Therefore, an attack or analysis technique based on measurements of one type could be often applied as is (or with very small modifications) to the two remaining ones.

3.1.2 Invasiveness

Side-channel attacks could be grouped by their level of *invasiveness*, attacks could be *non-invasive*, *invasive* or *semi-invasive*. During non-invasive attacks (such as one of the first side-channel analysis described in scientific literature [Koc96]) device is used and observed as it is i.e., the physical properties are measured on the device and it is left intact afterwards. Generally, after a non-invasive attack there is no evidence of the attack being performed (measures being taken). During an invasive attack the device is extracted out of its box and often depackaged using chemical or physical treatment e.g., the plastic package of a microcontroller might be destroyed using an acid in order to get to the silicon chip [KK99]. The device is often modified during an invasive attack, some parts of it can be intentionally broken or altered. Moreover, additional components might be added to the circuit in order to change its behaviour. Since invasive attacks modify the package of the device, they leave evidence of the attack. However, sometimes an invasive part of the attack can be performed on one device in order to gain knowledge about its internals and then the final attack is performed on a different device of the same model. Semi-invasive attacks lie between the invasive and non-invasive ones. Most of the time the device could be restored after the modification that was performed during the attack. There is no clear definition of the semi-invasive attacks in literature, they are in the grey area between invasive and non-invasive attacks. If an attacker has to unscrew several screws and disassemble the box of a device in order to get to the microcontroller the attack might be considered as non-invasive by some people, while others will already call it an invasive attack.

3.1.3 Interference

Another way of looking at side-channel attacks is the interference of the attacker with the device. From this point of view there is a separation between *active* and *passive* side-channel attacks. During a passive attack, physical properties of the device

[2] Here we are referring to the system-level EMA as opposed to locally-based EMA which requires to measure the signal in a close proximity of a specific region of the device.

are measured and the attacker is not acting on the device in an abnormal manner i.e., the attacker interacts with the cryptographic system as a normal user while the measurements through a side-channel are being collected. Active attacks imply that an attacker acts on the device in an abnormal manner i.e., in a way that is not expected by the device and that is not supported by the software and hardware. Often, these attacks are referred as *fault attacks* or *fault injections*. For example, an attacker can modify the input voltage as well as the clock frequency [AK96] during a carefully chosen clock cycle while the device is performing cryptographic operations. An attacker can also induce faults by using a laser beam [Hab65] or by simply illuminating a specific part of the device e.g., with a strong flashgun [SA02].

Note however, that some attacks can be situated in a grey area in between active and passive attacks (like the semi-invasive attacks in terms of invasiveness). This is the case of some *cache-based timing attacks* (also called *cache attacks* or *cache timing attacks*) [OST06]. The idea of a cache timing attack is based around the differences in the execution time (of an algorithm). If these differences in time are related to the secret key that is used during the encryption then an attacker can exploit them to recover the encryption key. In case of cache timing attacks the difference in time comes from the mechanism called caching, it allows to accelerate memory accesses when the same memory address is accessed several times. Cache memory is a small memory situated close to the Central Processing Unit (CPU). Accessing this cache memory is faster than accessing the Random Access Memory (RAM), thus to speed up memory accesses it makes sense to place values (within their addresses) that are accessed often to the cache memory. In case of a block cipher it means that e.g., if the same input value of an S-box is used more often during an execution then it will remain in cache and the total execution time will be slightly shorter compared to the scenario when the same S-box case is not accessed many times. Knowing this mechanism an attacker can try to change the contents of the cache memory and compare whether there is a time difference between two executions depending on when the attacker does or does not interfere with the device. The scenario when an attacker can actually change the contents of cache memory can happen if the software of the attacker is running on the same platform with the software of its target e.g., in case of several applications running on the same smartphone or in case when several virtual machines are sharing the same hardware [ZJRR12]. Nevertheless, not all cache timing attacks require to interfere with the cache memory as demonstrated on an OpenSSL implementation of AES [Ber05].

Table 3.1 refers to different types of side-channel attacks from the point of view of invasiveness as well as from the perspective of the interference. The main scope of this work is focused on non-invasive passive attacks.

Table 3.1 – Classification of some examples of side channel attacks and analysis on the level of invasiveness and interference.

	Active	Passive
Non-invasive	Power spikes injection	Power trace acquisition
Semi-invasive	Optical fault injection	Optical inspection of a circuit
Invasive	Circuit modification	Probing

3.1.4 Profiled and unprofiled attacks

Side-channel attacks could also be of two types from the point of view of the preparations done by the attacker. Side-channel attacks could be *unprofiled* and *profiled*. During an unprofiled side-channel attack the attacker measures physical properties of a device and then immediately analyses them in order to extract the secret key from the device. Profiled attacks require an additional step before attacking the target device. During a profiled attack the attacker starts by analysing a device in order to build its model, to be more precise the idea is to build a leakage model of a device in order to attack a similar device afterwards (often it would be the same model of a hardware device that is produced by the same manufacturer).

The attack phase of profiled attacks generally requires less measurements than unprofiled attacks. Profiled attacks are usually more powerful than unprofiled attacks thanks to the *profiling* which is done during a *learning phase*. This learning phase allows the attacker to find out more information about internals of the device which ultimately allows to build a good model that describes how the device leaks the information about the secret key. We will define *profiling* as the process of extracting a device-specific leakage model.

Profiled attacks require the attacker to have a control over the device. Often it means that the attacker should be able to set the secret key to a known value in order to perform measurements for the model, sometimes the attacker is also required to control the Random Number Generator (RNG) that is used by the device. In order to build a model the attacker should be able to control the device over a longer period of time (compared to the time that is necessary for the attack phase). Some scenarios of attacks do not offer a full control to the attacker, thus it is not always possible to perform a profiled attack (the attacker cannot build the model). So, an unprofiled attack could be the only choice in some circumstances and therefore they are also very interesting in practice.

3.1.5 Simple and differential analysis

There is yet another way of looking at side-channel attacks, they can differ from the point of view of the number of measurements and the kind of analysis techniques that are used by the attacker, those types of attacks are *simple analysis* and *differential analysis*.

During a simple side-channel analysis usually, the attacker measures physical characteristics of a cryptographic device while it is handling one input e.g., the attacker measures the power consumption of a device while it encrypts a single plaintext [KJJ99]. In these types of attacks the attacker can often infer all the necessary information about the secret key from the one measurement. These attacks present an interesting advantage of requiring a very small amount of measurements which implies a short data acquisition phase of the attack i.e., the attacker has enough time to get all the necessary data if the device is available for a very short period of time.

A differential side-channel analysis requires to collect measurements corresponding to different inputs (e.g., several plaintexts or several ciphertexts) while the device is performing cryptographic operations. In this type of scenario the attacker then uses statistical tools in order to extract the information from the set of measurements.

In other words, on one hand simple analysis mostly focuses on the sequence of instructions i.e., studies how the control flow of the program differ depending on the manipulated value. On the other hand, attackers use differential analysis to study how variations in the manipulated values influence physical properties of the device during the executions of the same instruction (when the control flow does not change).

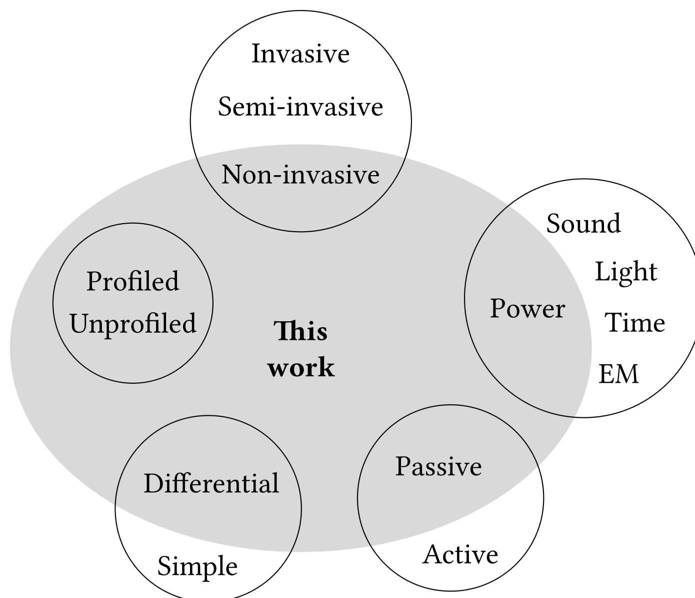


Figure 3.1 – Types of side-channel attacks and the main scope of this work.

3.1.6 Summary of types of side-channel attacks

Side-channel attacks can be categorised in several different ways. All these categories are orthogonal i.e., they do not interfere with each other and do not depend one on the other. However, some types of attacks are used more often in practice (mostly due to their efficiency, complexity and the cost of the equipment). A representation of the types of side-channel attacks and the main focus of this work is shown in Figure 3.1. In order to narrow the scope of this work, we mostly focus on passive non-invasive differential profiled and unprofiled power analysis.

Nevertheless, most of the results of this work can be applied to other types of attacks. Our results can be easily transferred and applied onto two other sources of information, and thus can be useful for EMA and acoustic cryptanalysis (which also handle time series). Moreover, they can also be used in case of simple (as opposed to differential) analysis. Several of such applications are going to be highlighted in the text.

3.2 Power analysis

Power analysis side-channel attacks are among the most commonly used attacks due to their high efficiency compared to the cost of equipment and computational com-

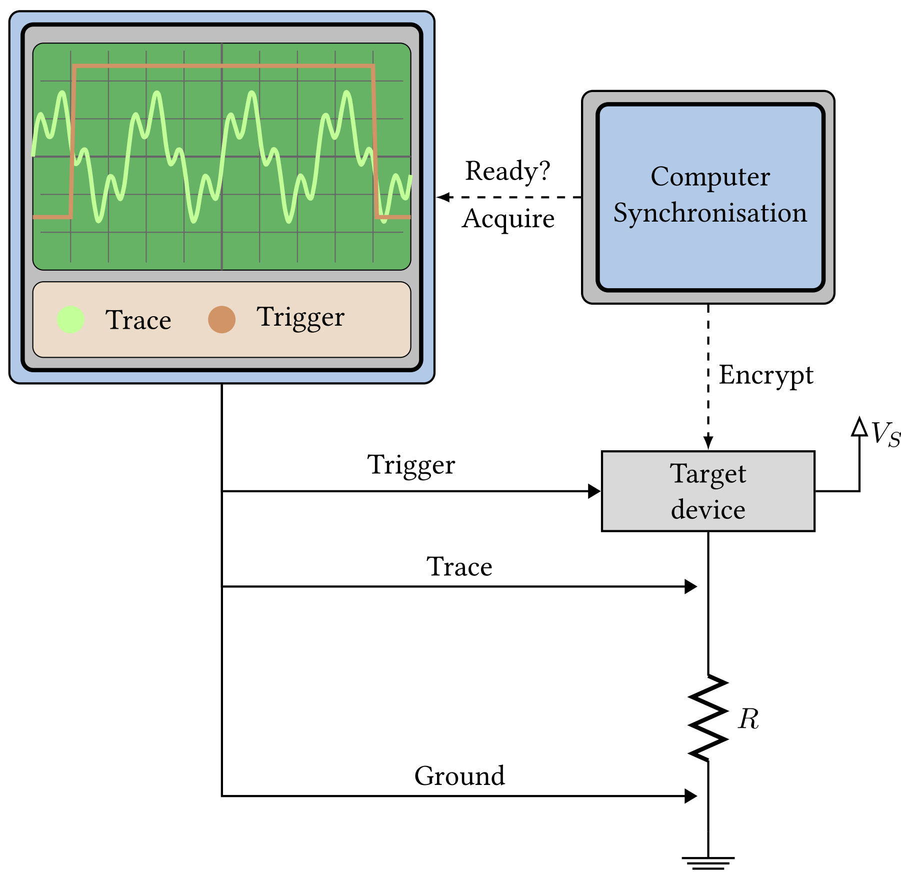


Figure 3.2 – Scheme of the acquisition setup for power analysis.

plexity. A side channel attack has two major phases: *data acquisition* and *data analysis*. We will take a closer look at these stages through the following sections.

3.2.1 Acquisition setup

In case of power analysis we are interested in the instantaneous power consumption of the device (while it executes a cryptographic algorithm). A typical acquisition setup that is used during a power analysis is shown on Figure 3.2, it consists of the device under attack, the measurement circuit, an oscilloscope and a computer which synchronises the acquisitions and often stores the acquired data.

This work focuses on microcontrollers as target devices, however, side-channel attacks can also be applied to other types of hardware such as a Field Programmable Gate Array (FPGA) or an Application Specific Integrated Circuit (ASIC). An oscilloscope is an instrument that allows to measure the change of voltage across a part

of a circuit over a period of time. The measurement circuit could be as simple as resistor connected in series between the power supply and the power input of the target device, as shown in Figure 3.2. However, it can also contain filters and amplifiers that allow to reduce the noise and amplify the measured signal. There are several ways of mounting the whole system together in order to acquire a dataset of power traces [Mor15, Chapter 2], our setup is shown in Figure 3.3. A more detailed view of the acquisition board (created during this work) which hosts the analysed microcontroller is shown in Figure 3.4. This acquisition board hosts a popular AVR microcontroller ATmega328 i.e., the microcontroller under analysis. This board also contains a resistor that we use to measure the voltage, a clock used by the microcontroller, several capacitors for filtering and connectors to make the setup process easier. In our setup, the oscilloscope has an operating system and it was used to make measurements as well as to synchronise the system and to store the results, in other words the oscilloscope played its own role and the role of the computer which synchronises the acquisitions.

We have build our own acquisition board and acquisition setup (shown on Figures 3.4a and Figures 3.3), which we used for our experiments. However, we would like to point out that there exist a ready-to-use acquisition setups and acquisition boards. Among the most known examples in the side-channel analysis domain we can name the ChipWhisperer hardware by NewAE[3] which can be used with several target devices including microcontrollers and FPGAs. This hardware was used for the CHES 2016 Capture The Flag (CTF) challenges[4]. Another notable platform is the lineage of SASEBO acquisition boards[5]. SASEBO boards were used for the DPA Contests[6]: SASEBO-GII with an FPGA for the 2nd edition of DPA Contest and SASEBO-W which allows to analyse smart cards was used for the 4th edition of the contest.

In the domain of side-channel attacks based on power analysis, one measurement acquired by an oscilloscope is called a *power trace*[7]. A power trace is a vector of values that we are going to denote T, we will also use the notation $T[i]$ to denote the value at the position i in the trace T. We will use the notation $\mathcal{T}$ to denote a set of power traces and $\mathcal{T}[i]$ to refer to the ith power trace in the set. In addition to the power traces the acquisition setup usually saves the values (usually plaintexts or ciphertexts or even both) that were used during the acquisition. We will use the notation $\mathcal{P}$ to denote a set of plaintexts and $\mathcal{P}[i]$ to denote the ith plaintext of the set (the notation $\mathcal{C}$ will be used in the same way for ciphertexts).

Note, that generally in domains of mathematics and computer science a *set* is an unordered collection of elements while a *vector* is an ordered collection (with a

[3]`http://store.newae.com/`

[4]`https://wiki.newae.com/CHES2016_CTF`

[5]`https://www.risec.aist.go.jp/project/sasebo/`

[6]`http://www.dpacontest.org`

[7]Some papers also use the term *leakage* to refer to power traces.

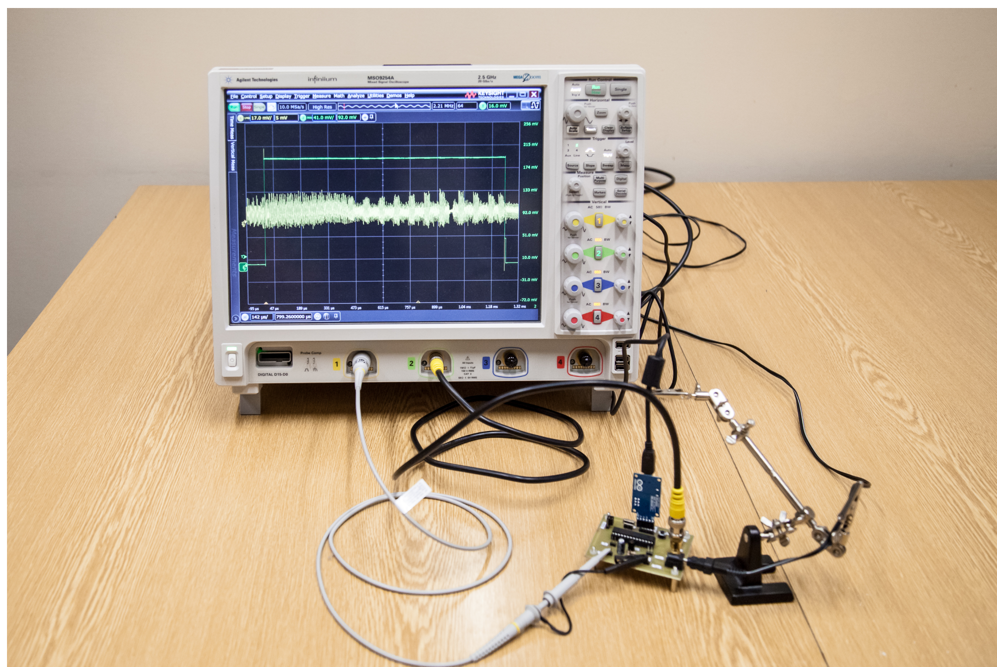
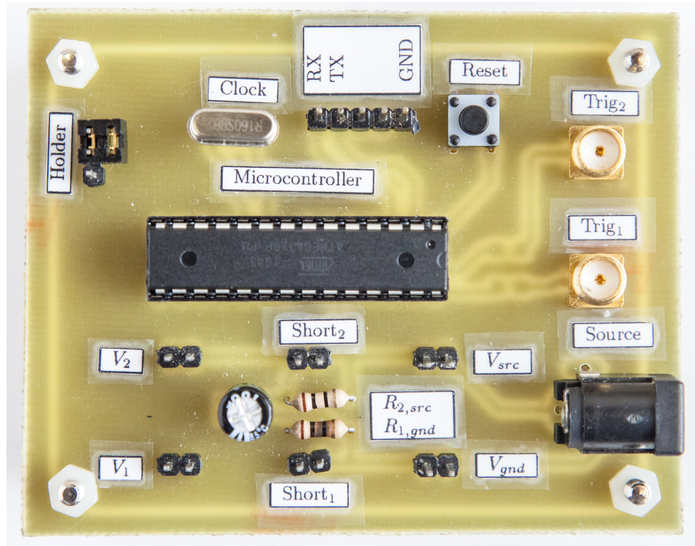


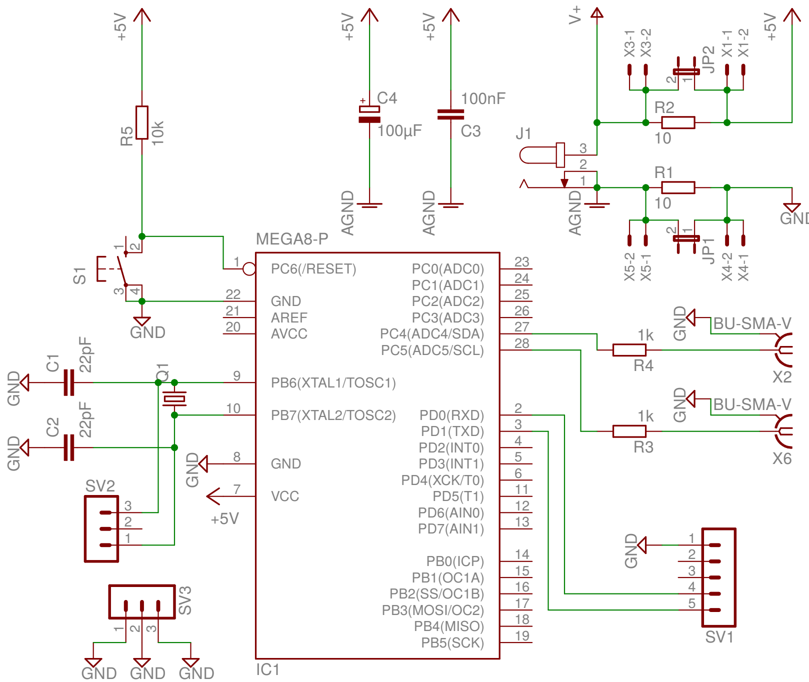
Figure 3.3 – Acquisition setup at DPA Lab at ULB.

notion of an index of an element). The word set is usually used in the domain of side-channel analysis to refer to a collection of traces or plaintexts, thus we will also use the word set. However, we will use an index to denote a specific element of a set to explain how every element of the set is handled during an attack. Side-channel attacks based on power analysis do not need to use traces in a specific order[8] same idea is applicable to the plaintexts and ciphertexts. The only thing that is required is the relationship between a plaintext, a ciphertext and a trace: during side-channel analysis an attacker or an evaluator usually needs to know which plaintext-ciphertext pair is related to the acquired trace. So, in this document we will refer to *sets* of traces, plaintexts and ciphertexts, where the order of elements in the set is not important, but the relationship between them is. Thus, we will suppose that the recordings of plaintexts and traces are done at the same time and that they are synchronised i.e., the ith power trace T is recorded while the ith plaintext p was used, in this case we will say that a trace T and the plaintext p are *associated* to each other. We will use the notation $T^{(p)}$ to specify the trace T which is associated with the plaintext p (and the notation $p^{(T)}$ to specify the plaintext p associated to the power trace T).

[8] Exceptions to this rule are rare and come up e.g., in case of t-tests where we have to acquire (but not analyse!) power traces while caring about the order of plaintexts that are used during the acquisition [SM15].



(a) A photo of the acquisition board.



(b) The scheme of the acquisition board.

Figure 3.4 – The acquisition board which contains the target microcontroller.

It is important to know that an oscilloscope does not measure the instantaneous power consumption, but the voltage across the resistor[9]. In order to get the instantaneous power consumption we need to use the Ohm's law:

$$V = I \times R \tag{3.1}$$

and the equation:

$$P = V \times I \tag{3.2}$$

where V is the voltage, I is the current, R is the resistance and P is the power consumption. The total voltage across the target device (V_{dev}) and the resistor (V_R) is the voltage V_S given by the power supply:

$$V_S = V_R + V_{dev} \tag{3.3}$$

In our case, V_{dev} changes due to the switching activity of logic gates, the oscilloscope measures changes in V_R, while V_S stays constant[10]. Putting together Equations 3.1, 3.2 and 3.3 allows us to transform the measurements of the oscilloscope V_R (using already known values R and V_S) to the power consumption of the target device:

$$P = V_{dev} \times I = (V_S - V_R) \times \frac{V_R}{R} \tag{3.4}$$

which gives us the instantaneous power consumption at a chosen moment in time (corresponding to a point of a trace recorded by the oscilloscope), this computation can be repeated to get a vector of instantaneous power consumptions from the trace given by the oscilloscope.

The resistor R has a constant value, the supply voltage V_S of the circuit also does not change in our setting. Thus, most of the side-channel attacks that use power consumption do not actually analyse the power consumption, but a physical property directly related to it (in our case, V_R). Nevertheless, a transformation from measured values to the actual power consumption can be used as a preprocessing technique before the actual data analysis. For the sake of simplicity, we are going to use terms *power traces* and *power consumption of a device* during the whole document, even though in practice an attacker does not necessarily measures and uses the real power consumption of the analysed device.

The acquisition setup that is used to acquire power traces has many different parameters. We would like to highlight several of those parameters that are important in order to get a good setup and in order to allow other researchers to repeat a

[9]Some attackers directly measure the current using a different type of setup.

[10]Note, that V_S does fluctuate on a small scale, these fluctuations mostly depend on the quality of the power supply (better power supplies can produce a more stable voltage signal). These changes in V_S can be seen as a part of the noise component and can be disregarded, moreover averaging of several measurements can be used to reduce these effects. During our experiments we used a cheap standard off-the-shelf power supply.

described experiment. In addition to noting the model of the target device and the attacked algorithm as well as the clock frequency and the voltage of the power supply one has to choose and record the following parameters:

- the value of the resistor — different values are used in the experiments, its value generally varies in between several Ohms [LMV+13] and several hundred Ohms [DMO16], if this value is too high then the voltage drop between the power supply and the target device would be too big and the device will not be operational, however if its value is too low then the voltage drop across it will be too small for the oscilloscope to measure;
- bandwidth of the oscilloscope — it defines which frequencies will be present in the signal, a bandwidth of 1 GHz is typically enough for side-channel attacks based on power analysis [MOP07, §3.4.3];
- the sampling rate (the acquisition frequency) — the number of single measurements that is recorded by the oscilloscope every second, according to Shannon [Sha49] this rate has to be at least two times bigger than the frequency of the signal that we want to measure, usually one would record several measurements per clock cycle, many analysis techniques use several points of a power traces in order to improve the accuracy of the attack (however if this value is too high, one will have to deal with huge amount of data during the analysis phase), nevertheless, some setups can lead to a successful attack even with one measurement per clock cycle[11];
- the resolution — is the parameter that defines how many different values are represented by the oscilloscope after the analog-to-digital conversion (digitalisation), most oscilloscopes use 8-bit values but some higher end oscilloscopes can use up to 12 bits for each value, higher number means that an attacker would be able to differentiate between very small variations of power consumption and thus will be able to extract secret from the device easier;

During a differential power analysis an attacker analyses a chosen offset in power traces, this offset corresponds to a specific point of power traces which leaks information about the internal state at a carefully chosen point in the algorithm. This type of analysis requires to align power traces i.e., a point $T[i]$ in any power trace of the dataset should correspond to the same operation (moment in time) of the executed algorithm. A *trigger* is used to acquire aligned power traces. Usually, the trigger signal changes its state just before the beginning of the encryption, see Figure 3.2. In this way, the trigger signals the oscilloscope that an encryption has started and in its turn the oscilloscope starts to record the measurements. Figure 3.5 shows a trigger

[11] `https://wiki.newae.com/CHES2016_CTF`

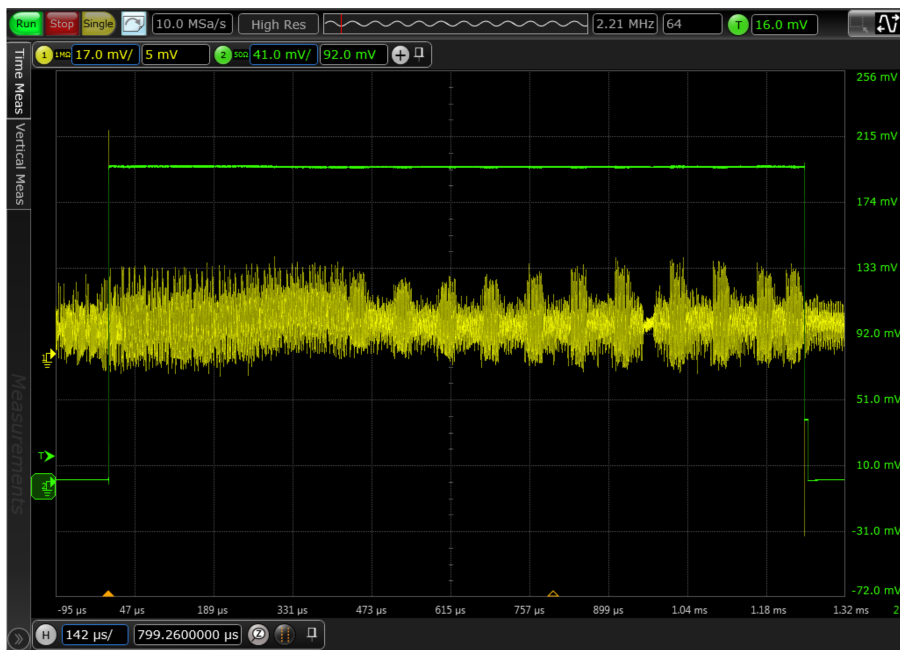


Figure 3.5 – Trigger signal (green) and the power trace (yellow) on the screen of an oscilloscope.

signal (green) that changes its state from low to high at the beginning of an encryption and switches back to low state at the end of the encryption. In this figure we can see the recorded signal (yellow) with a distinctive repetitive pattern, it corresponds to the rounds of AES, a different pattern at the beginning of the trace corresponds to the key schedule (computation of round keys).

3.2.2 Target operation

The idea behind a side-channel attack is to extract the key from an implementation by starting by extracting information about an intermediate state of the algorithm. During power analysis, attackers can choose an intermediate state that they want to target. Many choices are indeed available in case if power traces contain the information about the entire encryption algorithm (all its rounds), most of the time it is indeed the case since the attacker is the person who also performs the acquisition of these power traces. In order to mount an efficient attack it is important to carefully choose a target intermediate state that will "help" the attack.

Proximity to the known value Sometimes the attacker knows the output of the block cipher and in sometimes the input is the known part (recall modes of operation

in Section 2.1.1). An encryption algorithm can be executed in its forward or backward direction which respectively correspond to the encryption and to the decryption. If the target intermediate is “close” to the known value then the attacker will have to execute and analyse less steps of the algorithm between the known value and the target intermediate state. In case of a known input the attacker often targets an intermediate state which occurs during the first round of the encryption algorithm, such an attack is often called an *attack on the first round*. Similarly, *attacks on the last round* target an intermediate state that occurs in between operations of the last round, this method is usually used when the output of the encryption algorithm is known.

The size of the targeted part of the state Guessing or brute-forcing the entire key or the entire state of a block cipher is impractical due to their size. Thus, most of side-channel attacks use the “divide and conquer” approach. The idea is to extract a small part of the state that helps to recover a part of the key and then repeat this procedure several times until the entire key is extracted. The state undergoes several operations during each round and each of them operates on a part of the state at a time due to the hardware limitations (e.g., having only 16-bit registers) as well as designer’s choices (e.g., use of functions that operate on 32 bits of the state). Depending on these choices and limitations, the attacker would often target a part of an intermediate state which corresponds to the output of a specific operation of a block cipher. When an attacker targets an intermediate state that corresponds to the output of an operation A it is often referred as an *attack on (the operation) A* and we say the attacker *targets A*.

The type of the target operation One of the main parts of a side-channel attack on a small part of the key consists in guessing the value of this small part of the key and using it (e.g., with the known input) to execute a part of the algorithm (e.g., a part of the first round) until the desired intermediate state. Attackers can choose any intermediate state of the encryption algorithm for the attack. Any intermediate state is a result of a simple operation (key addition, permutation or substitution) performed on a previous state by the algorithm. In other words, the attacker can focus on any type of operation of the encryption algorithm. Key additions and permutations are mostly linear operations, while a substitution (an S-box) is often the only nonlinear part of the algorithm. For the purpose of choice of the target operation for an attack we are interested in the result of linear and nonlinear operations in case of a wrong input which corresponds to an incorrect hypothesis on the value of the secret key. In case of linear operations such as a permutation of bits in a register or a boolean operation like an exclusive-or a small input error (difference such as e.g., a single bit modification) will result in a small output error. At the same time, in case of nonlinear operations a single bit modification of the input will affect many bits of the output. Given this property of linear and nonlinear operations and the fact that power traces

are noisy (i.e., do not immediately give the attacker the value of an intermediate state, but rather some noisy information on it), the attacker will often choose to attack a nonlinear operation of the encryption algorithm. This choice can be influenced by the type of statistical tools (and their capabilities to represent complex models) that are chosen for the analysis of power traces, see Section 3.2.4.

To sum up, one of the best targets of a side-channel attack is the part of intermediate state that is obtained by applying a single S-box during the first round of encryption in case if the input of the block cipher is known (e.g., the plaintext is known or the encryption scheme uses the counter mode). Same idea applies in case of an attack on the last round of the encryption (when the output of the block cipher is available to the attacker), the attacker will often target the state that is used as an input of the S-box while going backwards in the algorithm. In both cases the target intermediate state is situated "behind" the S-box from the point of view of the known value. An S-box is also a good target since it promotes the divide and conquer approach, typically S-boxes are quite small from the point of view of exhaustive search and an attacker can easily test every single input. Even though most of the papers focus their attention on the S-boxes, these are not the only targets that are explored in literature, other examples include operations such as MixColumns operation of AES [WO15].

The attack on the last round has a couple of small differences compared to the attack on the first round. However, as we will see, these differences are mostly small technical tweaks that do not change the complexity of the attack, its general shape nor the main difficulties that an attacker has to face. The first difference between these two scenarios is that the attacker will have to go backwards in the key scheduling algorithm in order to extract the secret key from the round-key of the last round. This task is relatively straight forward, since a lot of the key schedules of modern block ciphers are easily reversible i.e., it is easy to compute a previous round-key from a given round-key (or to compute the main secret key from a round-key). For example, this is the case of AES-128, given any round-key it is possible to compute a previous round-key and the first round-key is the main secret key. A lot of the modern block ciphers use the secret key (or a part of it) as it is for the first round-key [AES01, BKL+07]. Exceptions to these rules are relatively rare (because cryptographers try to make efficient key scheduling algorithms by making them small and fast), however, several AES candidates [CDN99] use key-scheduling algorithms that are built in such way that their round keys do not reveal information on the main secret key. Kalyna [OGK+15, MGV+16] is among more recent examples that also has a key scheduling algorithm that cannot be easily reversed. The second difference between attacks on the first and on the last round lies in the actual operation that is being attacked. While an attack on the first round targets an S-box, the attack on the last round often has to target the inverse of the S-box. An inverse of an S-box is just another S-box of the same size, this inverse is still a nonlinear operation and it is still

a good target for a side-channel attack. Overall, both scenarios are very similar from the point of view of an attacker and an attack on the first round gives a good idea about the complexity of the attack on the last round. As a result, the vast majority of researchers focus on attacks on the first round of block ciphers. However, it is important to note that in practice one of the two attacks might be impossible because the attacker knows only the inputs (or only the outputs) of the block cipher. At the same time, even if the attacker knows both values (inputs and outputs of the block cipher), one of the attacks might be easier than the other due to many factors such as the details of the key scheduling algorithm, properties of the S-box as well as the details of the hardware architecture.

In order to simplify the explanations and for sake of clarity, we will focus on the attacks on the first round and we will assume that the attacker knows the plaintext while targeting the intermediate state given by the following equation:

$$z = f_k(p) \tag{3.5}$$

where $f_k(\cdot)$ is the function that is applied to the input plaintext p and gives an intermediate result which is computed by the algorithm after application of S-boxes.

While using different S-boxes from various algorithms most of the time we will target the result of the following computation:

$$z_i = S(k_i \oplus p_i) \tag{3.6}$$

where k_i is a part of the key, p_i is a part of the plaintext, z_i is the intermediate targeted value and S is the application of an S-box. We can use such simplification because almost all modern block ciphers start by performing the key addition operation[12] just before applying the S-box e.g., it is the case of AES [AES01] and PRESENT [BKL+07]. This idea still holds in case of an attack on the last round, since almost all modern block ciphers have a key addition as the last operation (key whitening, see Section 2.1.1).

An encryption algorithm can be constructed in such way that it performs linear operations such as permutations (on the key, on the plaintext or on their combination) before applying the S-box. However, we can disregard all linear operations since they do not affect the general complexity of the attack. These linear operations do not affect the ability of an attacker to distinguish two intermediate states (by observing side-channel information). Nevertheless, in practice an attacker will have to perform all the operations in order to mount a successful attack. During an attack on the DES block cipher an attacker will have to perform the linear operation called *initial permutation* in order to get the attack going, but this permutation just affects the

[12]Most ciphers use exclusive-or, other key addition operations such as modular addition or modular multiplications are more rare and this modification does not affect the general algorithm of a side-channel attack.

order in which the attacker will extract secret bits of the key rather than the general algorithm of the attack. As a simple example, let's say that a permutation used in an imaginary encryption algorithm is a swap between the first and the seventh bytes of the state and it is applied after the key addition, in this example an attack on the *first* byte of the state (after the S-box) will give the attacker the *seventh* byte of the key instead of the first one and vice versa. Same reasoning can be applied when single bits of the key (or plaintext) are used in a permutation.

It is important to note, that sometimes an attacker has to attack more than one round of an algorithm. The main reason lies in the key scheduling algorithm and the size of the key. A lot of block ciphers use only a part of a secret key in order to create a round-key e.g., AES-256 or PRESENT. It means that even if an attacker extracts the entire target intermediate state of the first round, it will be impossible to get the entire secret key. In this case the attacker has to execute the entire first round (and a part of the key schedule) of the algorithm using the already found parts of the secret key and start the attack once again on the second round of the algorithm in order to extract the missing parts of the key. The algorithm of the attack does not change at all since the input to the second round is already known an we can treat the second round as if it is the first one. Since the attack does not change at its core in a multi-round scenario and extraction of every additional part of the key requires to execute the same piece of code (perform the same attack) researchers mostly focus on the first round of an algorithm and on a single part of the key during the analysis. In case of an attack on an S-box, the size of a part of the key that is being targeted is equal to the size of an input of the S-box.

3.2.3 Leakage model

Power analysis attacks allow the attacker to extract information about the intermediate state of an algorithm during its execution and ultimately extract the secret key. It is possible because the instantaneous power consumption of the device at a given moment in time depends on the value that is being manipulated at that moment. The relationship between the value of a point of a power trace and the value that is manipulated by the device is called a *leakage function*. The leakage function, noted L^*, describes what information is leaked by the device while it handles a value. In practice the attacker does not know the leakage function of the device. It is practically impossible to compute (or extract) the leakage function from a given device because it depends on too many factors and parameters; basically it depends on the position and size of every single component (such as transistors and buses) in the device. However, in practice it is possible to mount a successful attack by using an approximation of L^*. An approximation of a leakage function that is used during a side-channel attack is called a *leakage model*, we will use L to denote a leakage model.

Generally, during power analysis the attacker considers that at each moment in

time during the execution of an algorithm the instantaneous power consumption P could be decomposed in the following way:

$$P = P_{op} + P_{val} + \varepsilon \tag{3.7}$$

where P_{op} is the part related to the operation that is being computed by the device (the executed instruction), P_{val} is the part related to the value that is handled by the device and ε is the noise. In case of Simple Power Analysis (SPA) [KJJ99] the attacker is mostly interested in P_{op} which is considered to be a constant for a given instruction[13]. In differential power analysis attackers are interested in modelling the part P_{val} which allows to recover an intermediate state of a block cipher. The noise is usually considered to be an independent random variable following a Gaussian distribution with zero mean:

$$\varepsilon \sim \mathcal{N}(0, \sigma^2) \tag{3.8}$$

where σ^2 is the noise variance which depends on the acquisition setup (that includes the ambient noise present in the laboratory).

There are two families of leakage models that are often used in power analysis side-channel attacks: Only manipulated Data Leak (ODL) and Memory Transitions Leak (MTL) [CGP$^+$12]. The ODL model considers that only the manipulated value influences the instantaneous power consumption of the device, this model is often used during attacks on microcontrollers (it works well in case when buses are set to a default value before being set to the manipulated value). The MTL model considers that two values (the previous one and the new one) of a memory unit (e.g., a register) influence the power consumption and that the device leaks some combination of the two consecutively manipulated values, this model is often used during attacks on hardware implementations such as FPGAs and ASICs. However, it also makes sense to use MTL during attacks on microcontrollers (see more in Chapter 7). The ODL and MTL models are often referred as value-based and distance-based models respectively, the leakage is also referred as being value-based or distance-based.

In case of ODL model the function $L(\cdot)$ takes one argument that is the target intermediate state. If the MTL model is used then the argument of $L(\cdot)$ is the result of a *combination* of two intermediate states such as the new and the old value of a register, the most common choice for the combination is an exclusive-or but it can be another function e.g., concatenation. Sometimes for the sake of simplicity in MTL it makes more sense to say that the function describing the leakage model uses two arguments which are the two intermediate states, for example in case of an implementation of an attack (a function in the code will actually have two arguments and the combination happens inside of it). It can also help in case when the attacker does not know *how exactly* two intermediate states are involved in the combination, the

[13] This value changes *during* the execution of one instruction, but for the *same time offset* from the beginning of the clock cycle the value is the same for a given instruction.

attacker can be using a machine learning technique to extract this function without knowing it in advance. Thus, from this point of view the model uses two parameters. For the sake of clarity and simplicity we will suppose that the leakage model $L(\cdot)$ uses only one argument through this document.

Depending on the type of attack, the attacker will use different approaches while choosing the leakage model. During profiled power analysis the attacker has the ability to extract a very good leakage model from the target device by feeding it with different plaintexts and keys while measuring device's power consumption. During the learning phase of a power analysis side-channel attack an attacker has two major goals: finding *points of interest* and extracting a *profile* from power traces. Points of interest are the points of power traces that are influenced by the target intermediate value and a profile is a leakage model of a device under attack. In other words, during the learning phase of power analysis the attacker has to find *how* the device leaks the information and *where* the leakage happens. In literature on side-channel analysis, *profiling* usually refers to the method and action of creating an accurate leakage model of the target device. We are going to use the notation $\mathcal{T}_l$ and $\mathcal{T}_a$ to denote sets of power traces that are used for the learning phase and for the attack respectively, these sets are called the *learning (or profiling) set* and the *attack set*. In a similar way we will use symbols T_l and T_a to denote one trace from the profiling and from the attack set.

During an unprofiled attack the attacker does not have the possibility of extracting an accurate leakage model, thus most of the time the leakage model is chosen using an educated guess that is based on previous experiments, the knowledge about the target device and tests (the attacker might try to attack the same device using several leakage models). There are two simple leakage models that are commonly used during analysis of attacks and countermeasures. The most common and simple examples of the ODL and MTL leakage models are the *Hamming weight (HW)* and the *Hamming distance (HD)* respectively. Hamming weight of a binary word is the number of bits equal to 1 in this word and the Hamming distance between two binary words is the number of single bit differences between the two words. The Hamming distance between two values a and b can also be seen as Hamming weight of the exclusive-or between them (the Hamming weight of $a \oplus b$). Figures 3.6 and 3.7 show the HW and HD leakage of an ATmega328P microcontroller.

3.2.4 Distinguishers

After the acquisition of power traces and the choice of the target intermediate state the attacker can start the analysis. For the sake of simplicity and in order to have a concrete example we will assume that the attacker targets one byte of the intermediate state that is given by doing the key addition followed by the application of an 8×8 S-box (as in the Equation 3.6), however same techniques and ideas could be

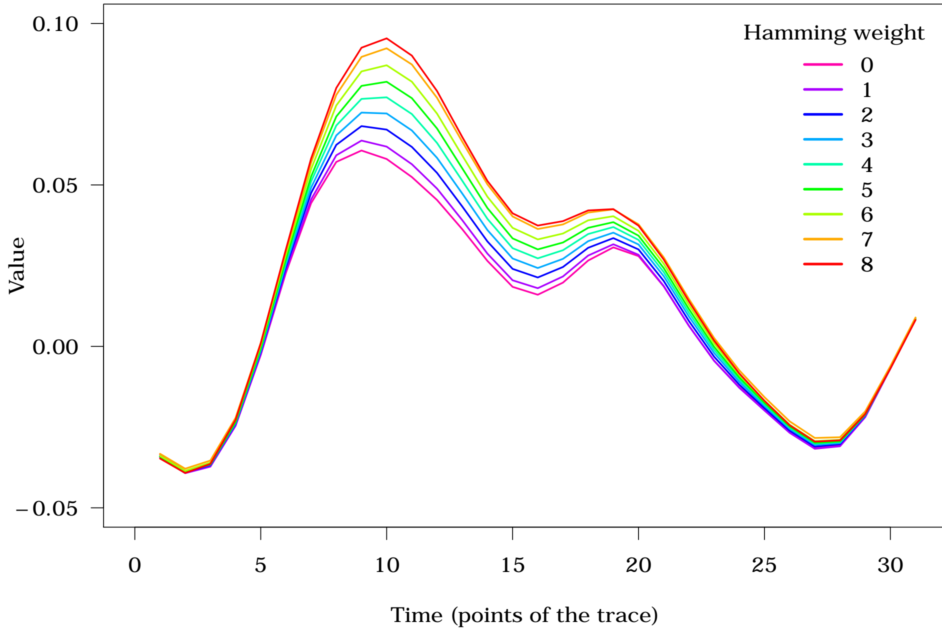


Figure 3.6 – Zoom on the average traces from ATmega328P. Values of different Hamming weights are manipulated by the same instruction. Each average trace corresponds to all values of the same Hamming weight.

applied to any number of bits.

During the analysis phase, the attacker has to go through all possible values of the part of the key that is targeted and run the partial algorithm until the target intermediate state. In other words, for each key hypothesis h the attacker computes the target intermediate state and applies the leakage model in order to get the hypothetical leakage l:

$$l = L(S(h \oplus p_b)) \tag{3.9}$$

where p_b is the byte of the plaintext. Finally, in order to find which of the key hypothesis is the correct one, the attacker can apply a *distinguisher*. A distinguisher is a statistical tool or an algorithm that allows to associate a *weight* to each hypothesis in order to sort them by likelihood of being the correct hypothesis (from the most probable to the least probable). A weight is assigned to each hypothesis by comparing real measurements and the value inferred from a model that was used with the analysed hypothesis. In other words, the distinguisher is a tool that allows to compare real measurements (from power traces) and the hypothetical values computed using a key hypothesis and the leakage model.

Once all the hypotheses are sorted by their weight, each hypothesis gets a *rank*.

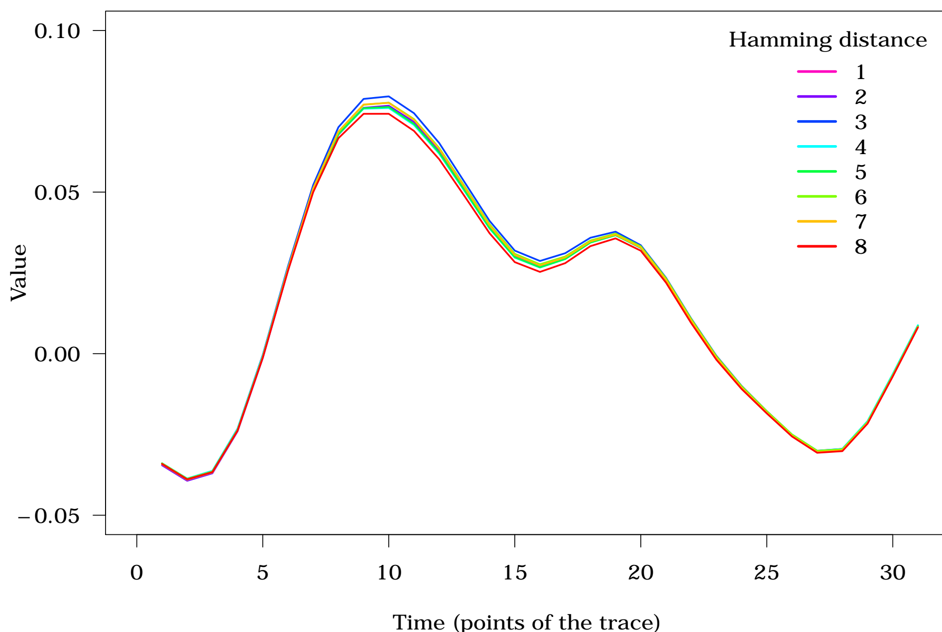


Figure 3.7 – Zoom on the average traces from ATmega328P. Input and output of different Hamming distances resulting from the execution of the same instruction. Each average trace corresponds to all values of the same Hamming distance.

The rank of a hypothesis is its position in the list of sorted hypotheses. We will say that a hypothesis has a high rank (or that it is highly ranked) if it is at the top of the list of hypotheses, in other words it means that the hypothesis is likely to be the correct one. We will use the term low rank to refer to the hypotheses that are at the bottom of the sorted list of hypotheses. We will use the notation $\hat{k}$ to specify the key hypothesis that has the highest rank (which is equal to the secret key in case of a successful attack). Note, that a distinguisher can potentially assign exactly the same weight to more than one hypothesis, and thus an attacker will have to test which one is correct. However, this scenario does not happen very often in practice, mostly because each point of a power trace is a floating point number and traces are noisy, in addition most of distinguishers also assign floating point weights to hypotheses. Thus, we are going to assume that all weights assigned by a distinguisher to different hypotheses are different and that there is only one $\hat{k}$.

Many different distinguishers were discussed in literature over the years, through the history of side-channel analysis newer distinguishers usually use more advanced statistical tools. We would like to focus on four of the commonly used distinguishers. Two of them are associated with unprofiled attacks and the two others are usually

used during profiled side-channel analysis.

Difference of means

Difference of Means (DoM) [KJJ99] is one of the first distinguishers that was suggested in the literature[14]. The idea behind this unprofiled approach is relatively straight forward: the attacker has to divide all traces into two *groups* or *clusters* g_0 and g_1, compute the average value of traces in each group and finally compute the difference between the two averages. If the hypothesis on the value of the key is correct then this difference should be the highest.

Since the attacker has to separate traces into two clusters, the leakage model is often chosen so that it can only output two different values. One of the most common choices is one of the bits of the output, e.g., the Most Significant Bit (MSB):

$$l = MSB(S(h \oplus p_i)) \tag{3.10}$$

in other words, the attacker supposes that the power consumption of the cryptographic device depends on the MSB of the result. In case when l is equal to 0, the power trace $T^{(p)}$ will be put into the group g_0 (it will be added to the group g_1 otherwise):

$$\forall T \in \mathcal{T} : T \in g_i \rightarrow \left(i = MSB(S(h \oplus p_b^{(T)}))\right) \tag{3.11}$$

The value $\hat{k}$ returned by this distinguisher is the hypothesis h such that:

$$\hat{k} = \arg\max_h \left(\max_t(|\mu_0[t] - \mu_1[t]|)\right) \tag{3.12}$$

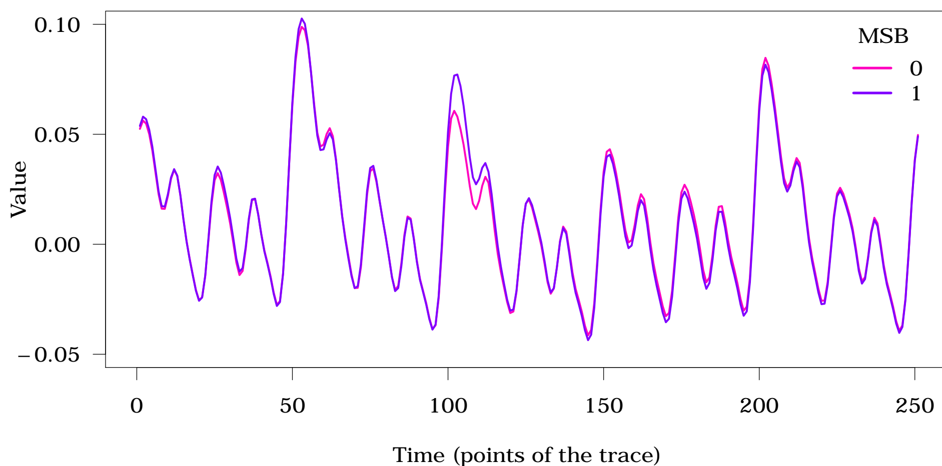
where μ_i is the mean vector of the traces from the group g_i calculated as follows:

$$\mu_i[t] = \frac{1}{N_i} \sum_{T \in g_i} T[t] \tag{3.13}$$

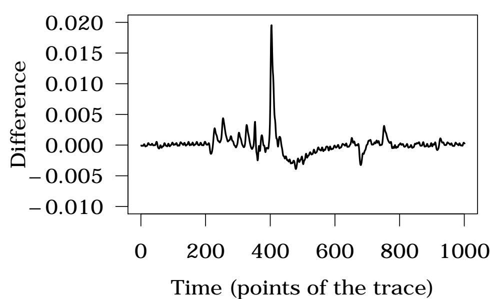
where t denotes the index of the tth point of the power trace (and of μ_i) and N_i is the size of g_i. Informally, μ_i is an average power trace where each point t is the average of the same points in all trace from the same group.

Note, that in case of an unprofiled scenario the attacker does not know which points of the acquired power traces are related to the chosen target intermediate value. Thus, the attacker chooses the maximum difference for each hypothesis (the second max in the Equation 3.12). Also, the attacker does not necessarily know if the power consumption associated to the group g_0 is higher or lower then the one associated to the group g_1. This is the reason of using the absolute value ($|\cdot|$) in the

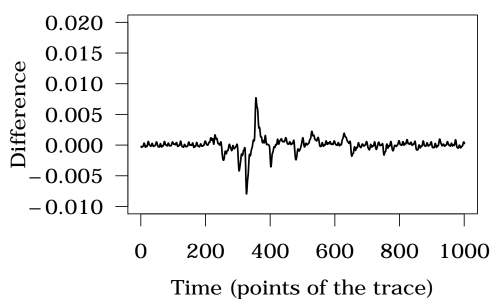
[14] The Difference of Means attack is often referred as Differential Power Analysis (DPA) attack, but we will reserve the term Differential Power Analysis (DPA) to the general idea of "differential" analysis as opposed to "simple" power analysis.



(a) Zoom on average traces with different MSB.



(b) DoM with the correct hypothesis.



(c) DoM with a wrong hypothesis.

Figure 3.8 – Example of DoM using MSB of the output of an S-box.

Equation 3.12. However, an attacker can learn this information about the device (e.g., by doing preliminary analysis) and use it in order to improve the accuracy and the efficiency of the attack i.e., the attacker will not need to compute the absolute value of the difference. This type of preliminary analysis of the target cryptosystem could be seen as a basic form of profiling.

Figure 3.8a shows a zoom on two average traces with different values of the MSB when the hypothesis is correct (equals the value of the key). Figure 3.8b shows the difference of means for the correct hypothesis and Figure 3.8c the same difference with an incorrect key hypothesis, you can see that the difference is higher for the correct one.

Correlation power analysis

Correlation Power Analysis (CPA) [CNK04, BCO04] is a kind of distinguisher that is often used in an unprofiled scenario. This technique relies on the correlation coefficient such as simple Pearson correlation coefficient that associates two sets (of size N) $\mathcal{X} = \{x_1, x_2, \dots x_N\}$ and $\mathcal{Y} = \{y_1, y_2, \dots y_N\}$ to the following value:

$$\rho = cor(\mathcal{X}, \mathcal{Y}) = \frac{\sum_{i=1}^{N}(x_i - \mu_{\mathcal{X}}) \times (y_i - \mu_{\mathcal{Y}})}{\sqrt{\sum_{i=1}^{N}(x_i - \mu_{\mathcal{X}})^2} \times \sqrt{\sum_{i=1}^{N}(y_i - \mu_{\mathcal{Y}})^2}} \tag{3.14}$$

where $\mu_{\mathcal{X}}$ and $\mu_{\mathcal{Y}}$ are the mean values of $\mathcal{X}$ and $\mathcal{Y}$.

The analysis phase of the attack when the attacker applies the distinguisher resembles the one of the DoM. The attacker does not necessarily know which points of each power trace are associated to the target intermediate value. Thus, the attacker computes the correlation coefficient between the hypothetical power consumption and the real one at each point t of power traces for each hypothesis h. The key given by this distinguisher is the following:

$$\hat{k} = \arg\max_{h} \big(\max_{t}(r_t(h)) \big) \tag{3.15}$$

where each coefficient $r_t(h)$ is the absolute value of the correlation coefficient between hypothetical power consumption and real power consumption at the point t of power traces (of size N_t):

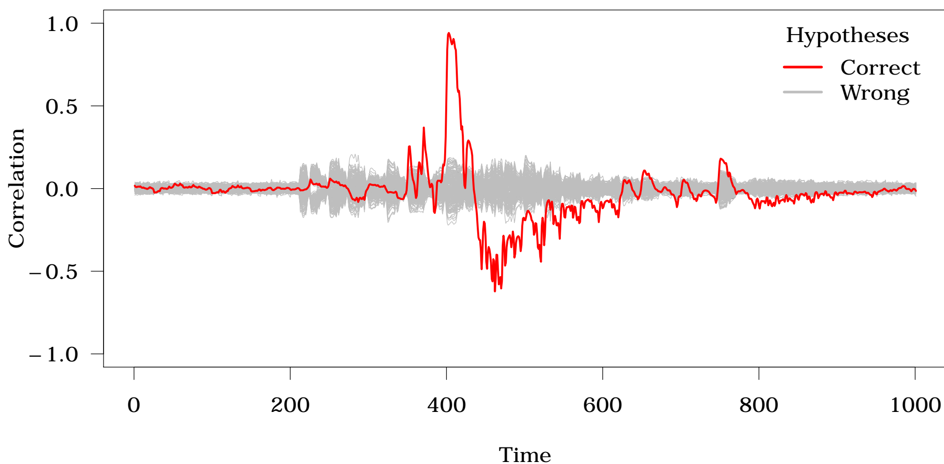
$$r_t(h) = \Big| cor\big(L(S(h \oplus \mathcal{P}[\cdot]_b)), \mathcal{T}[\cdot][t]\big) \Big| \tag{3.16}$$

where b denotes the byte that is being analysed and the notation $[\cdot]$ represents the fact that the same part (byte or point) is selected in all objects (plaintexts or power traces) of a set.

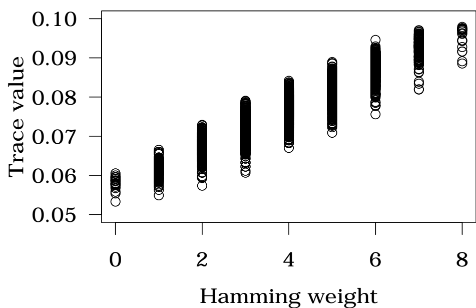
Figure 3.9 illustrates a CPA, Figure 3.9a shows all correlation traces with different hypotheses (the correct one is shown in red). Figures 3.9b and 3.9c show traces points from different traces (at the best offset with the highest correlation for the correct guess) with the corresponding HWs for the wrong and the correct guesses.

Template attack

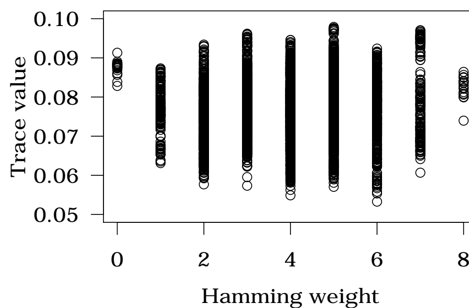
Template Attack (TA) [CRR02, CK13] is a profiled attack that has a learning phase and an attack phase. We denote $\mathcal{T}^z$ a set of traces associated to the intermediate value z (all traces that were recorded while the target intermediate value is equal to $z = f_k(p)$). Lets assume that N_l is the number of traces in each profiling set $\mathcal{T}_l^z$ and that each trace has t points. For the sake of simplicity, we are going to assume that traces contain only the points of interest for the attack. To create traces that contain



(a) All correlation traces for all points and all 256 hypotheses (for one byte).



(b) Values of the trace at the best point. Distribution with the correct hypothesis.



(c) Values of the trace at the best point. Distribution with a wrong hypothesis.

Figure 3.9 – Example of CPA using HW model.

only points of interest the attacker can simply discard all other points in each power trace. Here, when we say that the attacker is looking for the points of interest, we are referring to the idea of *feature selection* i.e., choosing the informative variables (useful for us) among the set of all available variables (points of power traces, in our case). However, in addition to selecting interesting points from power traces an attacker can also combine them before applying a distinguisher, some people may include this combination step into the whole concept of finding the points of interest.

Finding points of interest for profiled attacks can be done using methods e.g., based on correlation or mutual information; selecting good points of interest is actually a very interesting and separate research topic [RO04].

During the profiling step of a Template Attack for each possible value z the attacker estimates a mean vector μ_z:

$$\mu_z = \frac{1}{N_l} \sum_{i=1}^{N_l} \mathcal{T}_l^z[i] \tag{3.17}$$

and the covariance matrix Σ_z:

$$\Sigma_z = \frac{1}{N_l - 1} \sum_{i=1}^{N_l} (\mathcal{T}_l^z[i] - \mu_z)^{\mathsf{T}} (\mathcal{T}_l^z[i] - \mu_z) \tag{3.18}$$

where T is the operator that transposes the matrix.

The idea is to model the dependency between the power consumption and the value z as a multivariate normal conditional dependency:

$$\mathsf{P}(T^z | z; \mu_z, \Sigma_z) = \frac{1}{\sqrt{(2\pi)^t \times det(\Sigma_z)}} e^{-\frac{1}{2}(T^z - \mu_z)^{\mathsf{T}} \Sigma_z^{-1} (T^z - \mu_z)} \tag{3.19}$$

which can be used with the extracted profile in the attack step that gives the key which maximises the likelihood:

$$\hat{k} = \arg\max_h \mathsf{P}(\mathcal{T}_a | z) = \arg\max_h \prod_{T_a \in \mathcal{T}_a} \mathsf{P}(T_a | z; \mu_z, \Sigma_z) \times \mathsf{P}(z) \tag{3.20}$$

where $z = f_h(p)$. Intuitively, we choose the hypothesis which gives the highest probability of being the correct one according to the probability density function given the observed value of points in a power trace.

Figure 3.10 illustrates the profiling phase of a TA, each point represents one recording of a power trace at the moment in time when the S-box is processed. The attack phase is illustrated in Figure 3.11, it illustrates how the probability of each hypothesis being the correct one evolves. Before observing measurements this probability is the same for every hypothesis ($\mathsf{P} = \frac{1}{256}$), it evolves after observing more power traces depending on the observed value.

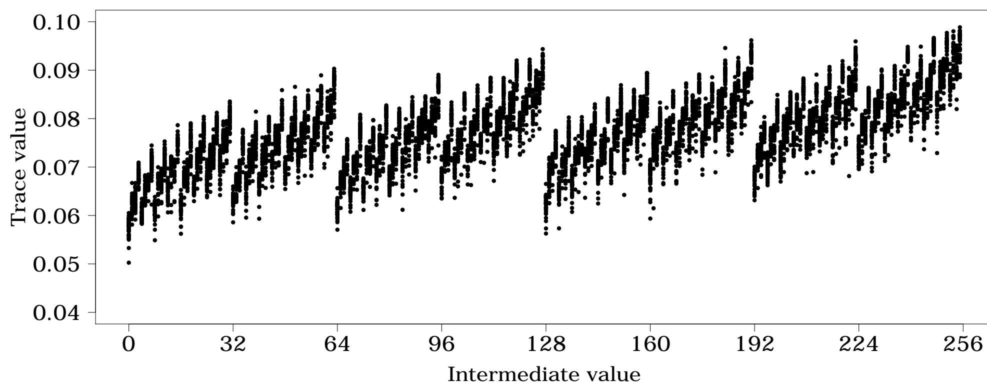
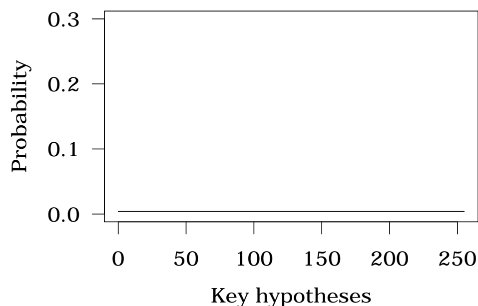
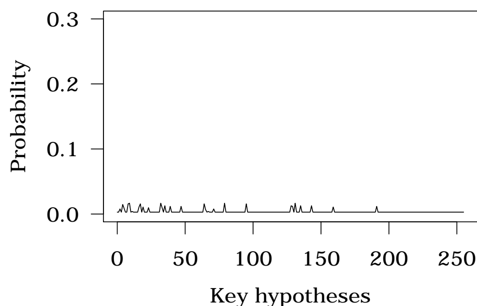


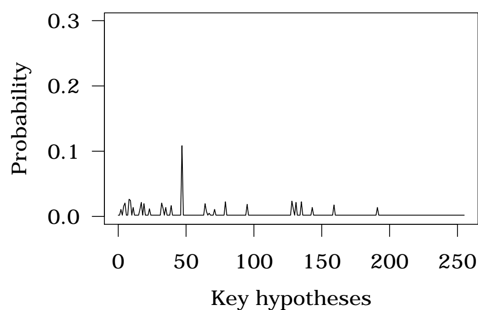
Figure 3.10 – Values from the point of interest of power traces corresponding to the computation of an S-box.



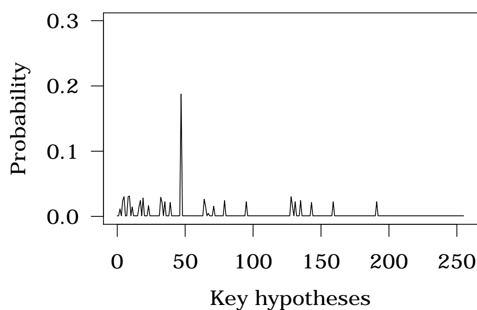
(a) Before observing power traces.



(b) After observing one trace.



(c) After observing two trace.



(d) After observing three traces.

Figure 3.11 – Attack phase of the TA. Probability of each hypothesis being the correct one. The correct value of the key in this example is 48 (the one that actually stands out).

It is important to note, that this approach makes the assumption that the distribution of values in traces associated to each target value z follows a parametric Gaussian distribution with $(t^2 + 3t)/2$ parameters per target value (the values in the mean vector μ_z and in the covariance $t \times t$ matrix Σ_z)[15]. Also, while the equation 3.19 refers to the *real* mean vector and covariance matrix, the attacker only *estimates* their values using the profiling set. Thus, this method works well only if the attacker has enough power traces in order to estimate all the parameters. In the case when the attacker knows the exact values of the mean vector and covariance matrix, as well as when there is no assumption error then TA provides the optimal distinguisher. Another interesting property of template attacks is the fact that these attacks require some noise in power traces in order to work properly, otherwise it will be impossible to invert the covariance matrix Σ_z (Equation 3.19). If the amount of noise present in power traces is very small, the attack can fail due to computational errors in Σ_z^{-1} (rounding errors because of the limitations related to the representations of real numbers in computers). If traces do not have any noise the attacker can consider only the mean values μ_z (to test the attack set for perfect matches against the model), in this case probabilities (Equations 3.19 and 3.20) will be either 0 or 1.

Stochastic attack

Stochastic Attack (SA) [SLP05, CK14] is based on the idea of *polynomial regression*[16]. It is used to model the relationship between the power consumption and the value that is manipulated at a given moment in time. Stochastic attacks model the power consumption P (at a point t of a power trace T) depending on the intermediate value z as a sum of W functions $\theta_i(\cdot)$:

$$P = \alpha_0 + \sum_{i=1}^{W} (\alpha_i \times \theta_i(z)) + \epsilon \tag{3.21}$$

where α_0 is a constant, ϵ represents noise and α_i are coefficients that are related to the importance of each $\theta_i(\cdot)$ (the higher the absolute value of α_i, the higher the impact of $\theta_i(\cdot)$ on the power consumption). Each function $\theta_i(\cdot)$ is a monomial given by the following equation:

$$\theta_i(z) = \prod_{\beta \in \mathcal{G}} B_\beta(z) \tag{3.22}$$

[15] Efficient template attack [CK13] needs to estimate less parameters, it does so by supposing that the covariance matrix is the same for all target values. Thus it needs $256 \times t$ parameters for mean values (in case of one byte) and $(t^2 + t)/2$ for the single covariance matrix.

[16] The term *linear regression* is often used in the literature, however it can only be used to describe a subset of stochastic attacks i.e., stochastic attacks of the first degree, see more in further paragraphs.

where B_β is the function that outputs the bit β of the word z, which could be written as follows:

$$B_\beta(z) = \left\lfloor \frac{z \pmod{2^{\beta+1}}}{2^\beta} \right\rfloor \tag{3.23}$$

and $\mathcal{G}$ is a non-empty set $\mathcal{G} \subseteq \mathbb{Z}_8$ in case of an attack on one byte. Note, that W equals $2^{\lceil \log_2(z) \rceil - 1}$ i.e., sum of $\binom{n}{k}$ for all k except 0, where n is the number of bits in z; the coefficient that corresponds to 0 is the constant α_0 (corresponds to the monomial where no bits are in the product).

One can choose the *degree* of a stochastic attack in order to fine-tune an attack by choosing the complexity of the model. The degree of a stochastic attack is the highest number of variables in a monomial $\theta_i(\cdot)$ with non-zero coefficient α_i. The notation SAi represents a stochastic attack of degree i. For example, SA1 is the stochastic attack of degree 1 which only considers monomials that are composed of a single bit (in other words it uses a simple *linear regression*), while SA2 considers a model where monomials can be composed of single bits and of products of two bits.

The model is created during the profiling phase using the set $\mathcal{T}_l$ through the mechanism of polynomial regression in order to estimate coefficients α_i. The attack phase uses the set $\mathcal{T}_a$ in a way that is similar to the template attack, i.e., the equation 3.20 is used while assuming that $\mathsf{P}(T_a|z)$ follows a Gaussian distribution with the mean given by the model extracted using SA and covariance matrix Σ given by the residual terms ϵ.

The SA that we present here is the "classical" version of the distinguisher. Several modifications and improvements were suggested in the literature e.g., the use of an orthogonal basis [GHM+17] that allows to detect and pinpoint more complex (higher-order) leakages.

Other distinguishers

The distinguishers presented in previous sections constitute a representative group of all distinguishers that we will focus on. Among other commonly used unprofiled distinguishers we would like to highlight Mutual Information Analysis (MIA) [GBTP08, BGP+11]. As for the distinguishers that are used for profiled attacks there are also many techniques based on machine learning e.g., Support Vector Machine (SVM) and Random Forest (RF) [LBM11, LMBM13] as well as Neural Networks (NN) [MPP16] and methods based on clustering algorithms [LMV+13]. New distinguishers or improvements (e.g., efficient algorithms) on existing ones are suggested in literature relatively frequently.

3.2.5 Key enumeration

After running a complete attack on a part of the key the attacker gets a list of hypotheses sorted by the decreasing likelihood of being the correct hypothesis. In the

best case scenario for the attacker, once the attack is performed on every part of the key, the hypothesis that was ranked first (for every part of the key) is the correct one. Thus, combining (usually simply concatenating) all hypotheses of highest rank immediately gives the secret key. The attacker can usually check if this is the case by running the encryption algorithm using a known plaintext-ciphertext pair.

Sometimes the correct hypothesis is not ranked as the first one e.g., due to the high amount of noise in power traces or because the leakage model was not good enough. Nevertheless, it does not necessarily mean that the attack failed, the attacker can still use a *key enumeration algorithm*. The key enumeration algorithm is similar to the brute-force exhaustive key search and in the worst case scenario both algorithms are equivalent. However, in case of a last step of a side-channel attack a key enumeration algorithm uses the information obtained during previous steps i.e., the ranks of hypotheses given by the distinguisher. Since hypotheses are sorted by the decreasing order of likelihood the key enumeration starts by considering highly ranked hypotheses of each part of the key. There exist an optimal key enumeration strategy (with respect to the order of the generated full key-hypotheses) that can be applied after any distinguisher that assigns probabilities to hypotheses [VGRS12]. This algorithm was modified to greatly improve its memory efficiency by rendering it suboptimal but still very efficient [BKM+15]. Different strategies can be applied in case of a distinguisher that does not output probabilities (such as correlation). One strategy consists in converting the outputs of the distinguisher to probabilities and then applying the optimal key enumeration algorithm. Another strategy consists in taking into account weights "as they are" (instead of probabilities) that were assigned to different hypothesis, however it does not necessarily result in an optimal strategy.

The idea of final key enumeration in side-channel attacks is strongly related to the concept of the *key ranking* [PSG16]. While the goal of the key enumeration is to find the correct secret key, the goal of the key ranking is to find the rank of the entire secret key knowing its value i.e., an algorithm of key ranking gives the position at which the correct key is considered during the key enumeration process. The key enumeration is used during an attack, while the key ranking is usually used during the evaluation of a cryptographic product in a laboratory. The goal of an evaluation lab (that might be involved in a certification process) is to find how difficult it is to find the secret key. Thus, an evaluation lab just wants to know how many values an attacker will have to consider before finding the correct key after getting the results from a distinguisher.

3.3 Analysis of side-channel attacks

Analysis of cryptographic systems from the point of view of side-channel attacks allow us to describe how cryptographic devices respond to them. In order to analyse an attack (or an implementation of a cryptographic algorithm) we need to define a

way of measuring its performance. It allows us to measure the performance of an attack while varying its parameters e.g., different measurement setups or leakage models. Which finally gives a way of comparing two attacks against an implementation (or two different implementations challenged with the same attack). Ultimately, this analysis can help in finding weak and strong parts of an implementation which gives researchers a chance on improving existing implementations and create new, better ones.

3.3.1 Performance of an attack

Several ways of measuring the performance of a side-channel attack were developed by the community. The ideas behind them moved towards using more advanced statistical tools and towards encompassing more information about the results of a side-channel attack.

The most straight forward way of measuring the performance of a side-channel attack is to report the number of power traces (also referred as the *number of queries* in this context) that is used during the experiment that lead to the successful key extraction. This method was the first one that was used in the domain [Koc96, CCD00]. Generally researchers report the average number of traces necessary for the key extraction. This method is still used by researchers [Gag13], mostly in order to show the order of magnitude of the amount of data necessary for a successful attack. The number of traces is also used by the evaluators of the DPA Contest as one of the measures of the strength of a given side-channel attack[17]. This metric is basically a measure of average. It does not give us all the information about the behaviour of an attack i.e., in practice the attack can actually work with fewer traces and it can also fail with more traces.

The *success probability* used in the paper on the Template Attack [CRR02] later called *success rate* [SGV08, SMY09] of an attack is a metric that estimates the probability of a successful key extraction in a given setup (fixed number of traces, fixed noise, etc.). We define the success rate Sr of an attack A that outputs the key $\hat{k}$ as the probability $\mathsf{P}(\hat{k} = k)$. Sometimes researchers use a metric that is equivalent to the success rate which is called the *error rate*, it is defined as $Er = 1 - Sr$. The success rate (compared to the average number of queries) is a measure of a variability of the attack, it shows us how often an attack will succeed in a given scenario rather than the fact that "it is possible to extract the key in the given scenario". The idea of the success rate of an attack could be extended to encompass more information about the results of the attack, we can define the success rate of order o, noted Sr^o. The success rate of order o is the probability that the correct key k is ranked among the o highly ranked hypotheses [SMY09], thus the success rate Sr is actually the success rate of order 1.

[17] `http://www.dpacontest.org/v4/rsm_hall_of_fame.php`

A relatively low success rate of order o does not mean that the attack has a low success rate of higher order $o+1$, which means that an attack can be very powerful while still having a relatively low success rate of the first order. Thus, sometimes it is more convenient to use another metric that characterises the attack, this metric is called the *guessing entropy* [KB07] which is also referred as the *average rank*. In the context of side-channel attacks guessing entropy is the average rank of the correct key in the ranking returned by the analysed attack[18].

The success rate metric has an interesting property compared to the guessing entropy. Given the success rate of an attack on a part of the key we can infer the success rate of extracting the entire key in the same scenario (because of independent probabilities), while the guessing entropy of an attack on a part of the key does not necessarily give us the rank of the entire key. Thus both metrics are often used by the researchers nowadays.

Theoretical metrics

Researchers as well as designers want to predict the behaviour of cryptographic systems (from the perspective of side-channel analysis) in early stages of development and even during the design of the algorithms. This desire pushes researchers to create theoretical tools that allow to estimate the resistance of an algorithm or an implementation without collecting real power traces. A theoretical metric generally tries to evaluate the robustness of a given function (e.g. an S-box) against a side-channel attack under several assumptions such as the leakage model.

Transparency Order (TO) was suggested as a theoretical metric that can be used to evaluate the resiliency of an S-boxes from the point of view of side-channel attacks [Pro05]. Later on, this metric was improved by Chakraborty *et al.* [CSM+17]. The transparency order of an $m \times n$ S-box S is defined as follows:

$$TO = \max_{\psi \in \mathbb{F}_2^n} \left(n - \frac{1}{2^{2m} - 2^m} \sum_{w \in \mathbb{F}_2^{m*}} \sum_{j=1}^{n} \left\| \sum_{i=1}^{n} (-1)^{\psi_i \oplus \psi_j} \mathrm{C}_{S_i,S_j}(w) \right\| \right) \tag{3.24}$$

where ψ_i represents the value of the ith bit of ψ, and

$$\mathrm{C}_{S_i,S_j}(w) = \sum_{x \in \mathbb{F}_2^m} (-1)^{S_i(x) \oplus S_j(x \oplus w)} \tag{3.25}$$

in both formulae variables with lower indices i and j represent the corresponding ith and jth bit of the value.

The resiliency of S against side-channel attacks should be higher for lower values of the transparency order. This metric is based on the Hamming weight leakage and ψ

[18] Sometimes there can be a difference of 1 between the two metrics since some researchers like to start counting ranks from 0 and others from 1. However, it does not change the idea of this metric.

represents an initial value of a register that leaks the information, to be more precise, TO supposes that the device leaks $HW(\psi \oplus S(a))$. The use of max in the Equation 3.24 allows to dissociate the metric from a specific device. Transparency order equation represent the best case scenario for the attacker (worst for the developer).

Another theoretical metric, called Confusion Coefficient (CC), that tries to give an evaluation of the resiliency of an S-box was proposed by Fei *et al.* [FLD12, FDLZ15]. The confusion coefficient is defined as follows:

$$CC = \sigma^2 \left[\kappa\left(k_i, k_j\right) \mid \forall i < j\right] \tag{3.26}$$

where each term κ is given by the following formula:

$$\kappa(k_i, k_j) = \mathsf{E}_p \left[\left(L\left(S(k_i \oplus p)\right) - L\left(S(k_j \oplus p)\right)\right)^2\right] \tag{3.27}$$

where k_i and k_j are possible values of the secret key, p is a value of a plaintext and L is the leakage model. Thus, the confusion coefficient is a measure of variance of κ. In its turn, $\kappa(k_i, k_j)$ gives the measure of similarity between two keys (over all plaintexts), given an assumption on the leakage function.

The original paper [FDLZ15] on CC states: *"..for CPA on AES, the deviation of confusion coefficients is about 25% while that for CPA on DES is about 40%. This means the key candidates behave more similarly and randomly in AES than in DES, and therefore AES is harder to attack"*. Which means that lower CC should result in higher resiliency against side-channel attacks. Similar statement is given in the following work by the same authors [FLD12]: *"..the one* [S-box] *with larger confusion coefficients leaks more information, leading to higher success rates.."*. However, later works [PPE$^+$14, Sto15] show that larger CC leads to lower success rate i.e., lower confusion coefficient is better for the attacker, which is contrary to the original work.

Both theoretical metrics, the improved TO as well as the CC were used in order to create new S-boxes using evolutionary algorithms [PMMB15, PPE$^+$14].

3.4 Countermeasures

Each time a new attack appears on the scene, cryptographers suggest ways that can be used to protect cryptosystems against them, side-channel analysis is not an exception. Several types of countermeasures were suggested as a remedy against side-channel attacks. Most of these countermeasures fall into one of the two main groups called *masking* and *hiding*, however some of them stand apart and cannot be easily classified into one of these groups.

3.4.1 Masking

Masking is one of the first countermeasures that was suggested in the scientific literature [CJRR99, GP99, RPD09]. The main principle consists in randomizing the value

that is being manipulated by the device. The idea behind this randomization is the following, if the power consumption of a device depends on the manipulated value (which creates a leakage) the designer could try to use a different random value during each execution. As a result the information leakage will vary at each execution and thus the attacker should not be able to extract the secret value manipulated by the device. In other words, masking technique tries to remove information leakage by acting on the *vertical* axis of power traces i.e., the value at the point of interest on a power trace will be different at each new execution.

Generally, the same key is used for the encryption of multiple blocks (key agreement protocols are relatively costly compared to the encryption itself). Thus, instead of changing the key for every single encryption the key is combined with a random value. This random value is called a *mask* and the combination is done is such a way that it does not affect the result of the encryption i.e., encrypting a given plaintext with the same key gives the same result with and without masking. In order to achieve this effect, some parts of the encryption algorithm have to be altered to take masks into account.

There are two types of masking: *arithmetic* masking and *boolean* masking. The arithmetic masking schemes are build on operations such as modular additions and modular multiplications, this type of masking is generally used with public key cryptography[19]. Symmetric key algorithms, such as block ciphers generally use boolean masking schemes which are based on boolean operations, usually the exclusive-or (xor). This idea is based on secret sharing [Sha79, Bla79], where a secret value is split into several shares and each individual share does not provide information about the whole secret.

There exist several boolean masking schemes that describe how to apply them, however the main principle remains the same. First, a random value is generated in order to create a mask (most of the time the random value *is* the mask, but it is not always the case [NSGD12]), next the mask is mixed with the internal state of the algorithm, then the algorithm is performed and finally the state of the algorithm is *unmasked* (i.e., the effect of masking is removed) to get the ciphertext. Thus, the internal state is masked during the execution of the algorithm, but the output is not affected by the masking. Secret-sharing provides theoretical security under ODL model. In other words, masking secures implementations against an attacker who can extract information about only one value that is processed at a given time and values are manipulated separately. This assumption is usually referred as Independant Leakage Assumption (ILA) [RSV$^+$11].

When the mask is mixed with the intermediate state of the algorithm using an exclusive-or, linear operations (e.g., permutations) are affected differently than non-linear operations such as applications of an S-box. In case of linear operations, it is possible to unmask the state by applying the same linear operation on the mask.

[19]The term *blinding* is often used in public key cryptography to referrer to this type of randomization.

However, a nonlinear operation requires a special treatment (change in one bit of the input will affect multiple output bits). For example, an S-box S has to be replaced by a masked S-box S_m in a following way:

$$S_m(z \oplus m_{in}) = S(z) \oplus m_{out} \tag{3.28}$$

where z is the intermediate value that is masked and m_{in} and m_{out} are the input and the output masks that are used. The input mask is the mask that is initially mixed with the internal state of the algorithm. The output mask is necessary in order to keep the internal state masked after the application of the S-box. If we do not use an output mask then the masked S-box (using a masked input) will produce a result which is the same as produced by the original S-box with an unmasked input. In this case the intermediate value after the application of the S-box will not be masked, thus it will not be randomized. Therefore, attackers can target it in order to extract the key as if they were targeting the original unmasked S-box.

Creation of a masked S-box S_m has to be done for every possible value of the mask, which requires additional resources: ether all masked S-boxes are precomputed (increase in memory use) or masked S-boxes are computed on the fly (each execution requires more time)[20]. Also, it is important to know that masking requires randomness, which is a valuable resource in cryptographic systems. Due to these requirements, designers of masking schemes try to minimise the amount of necessary resources while still providing an increase in security against side-channel attacks. It is done ether by reusing some masks more than one time [MOP07, Example for Masked AES, page 228] or by limiting the total number of masks that can be used [NSGD12]. In addition to these measures, implementations usually apply masking only on the first and the last round of the block cipher, it is done because attackers usually target intermediate values in the first or the last round.

Masking schemes can sometimes be considered to be too expensive to implement in practice due to the constraints imposed on the final product (execution time, available memory and costs related to the RNG). Therefore, several *lightweight* (i.e., low-cost or low-footprint) schemes have been developed. One way of making a masking scheme more lightweight is to reduce the entropy of the masks, in other words it means that we try to reduce the costs related to the generation of random numbers. Examples are masking schemes with two masks [LR11, BDGN13], or in general with reduced mask space [NGD11]. Rotating S-box masking (RSM) [NSGD12] is one of such Low-Entropy Masking Scheme (LEMS), it uses only 16 out of 256 possible masks, according to the condition that those masks form a linear code of high dual distance. Some variants can be derived from the basic RSM scheme for example, Ya-

[20]Note, that we cannot compute just the necessary values in order to speed up the process, since the attacker will be able to target the pre-computation. Thus, we have to compute the entire masked S-box even if we do not need all values.

mashita *et al.* [YMOT14]) also uses 16 masks, but with different ways to generate them.

Researchers have tried to create masking schemes that are provably secure, however, generally these schemes are secure under certain assumptions on the leakage function which are not necessarily true in different scenarios. For example, several schemes [PGA06, SP06, RP10, BFGV12] that were presented with security proofs under some leakage model, were later found to be insecure under another leakage model [CGPR08, CPR07, CPRR13, PRR14] (e.g., ODL vs. MTL models [CGP$^+$12]) or due to some assumptions that are not necessarily true in real implementations (e.g., related to new or unknown type of leakage). In other words, the ILA does not always hold in practice. The exact values that are being manipulated are not always "visible" at the level of abstraction that the developer has to work with e.g., at the high-level source code or even at the assembly code. In addition to the fact that devices often leak distance-based values (MTL model such as Hamming distance), *coupling effects* [RSV$^+$11] also break the ILA. The coupling is an interaction between two values that are used or stored seemingly independently of each other, in such scenario the power consumption of the device (at some point in time) will depend on the combination of the two values. Moreover, *glitches* [MS06] can also unmask a masked value and diminish the security of the scheme. In case of side-channel analysis a glitch is a short unintentional modification of a value (e.g., on a wire) that can be caused by the differences of arrival times of several signals to a logic gate which causes the logic gate to change its output value multiple times.

Masking can be applied in software as well as in hardware [MPO05], including lower levels of abstraction such as masked logic gates e.g., Masked Dual-Rail Precharge Logic [PM05].

Masking does not give a perfect protection against side-channel attacks. Common attacks against masking include *high-order attacks* and *higher dimensional attacks.* Higher-order side-channel attack is an attack that analyses and exploits statistical moments of higher order such as variance, skewness, kurtosis, etc. Higher dimensional attacks exploit several points of interest in each power trace, a common technique consists in using several points related to different secret shares and combining them in order to analyse the result and to extract the secret key. In addition to these relatively generic techniques, attackers can also try to find vulnerabilities in the masking scheme, in its implementations (instances), or even take advantage of glitches that may unmask some parts of the internal state of the device. It is also possible to attack the random number generator [LMBM13, LBM15b] in order to extract the values that were used to generate masks and effectively remove the masking countermeasure.

In order to protect an implementation against these counter-attacks it is possible to use higher-order masking. In these high-order schemes a value is split into more than 2 secret shares. In this case, a masking scheme is called a masking of order d

and the secret is split into $d+1$ secret shares. For example, first ideas about *threshold implementations* [NRR06] suggest to use 3 shares as a way of protecting against attacks that exploit glitches.

Since we have to deal with two types of leakage models: value-based as well as distance-based (ODL and MTL) and some schemes secure under ODL are not secure under MTL, researchers came up with the notion of order-reduction. It states that a dth-order secure masking scheme under value-based leakages is $\lfloor \frac{d}{2} \rfloor$th-order secure under distance-based leakages. Its applicability has been verified experimentally by Balasch *et al.* [BGG$^+$14] for orders 1 and 2 in AVR-based and 8051-based devices. De Groot *et al.* [dGPdlP$^+$16] have also verified the order-reduction experimentally for orders 1 and 2 in the ARM Cortex-M4.

While talking about leakage in masked implementations researchers often use the term ith order leakage to describe *how* a particular leakage can be exploited. This term is related to the difficulty of exploiting the leakage (to extract the key) in terms of the algorithm of the attack and not in terms of the number of traces or known inputs. An unprotected implementation produces a *first order leakage*, it means that there exists at least one point of a power trace which contains all the necessary information about the intermediate state and that this information can be exploited by analysing the first statistical moment (average value). In other words, this type of leakage can be exploited using a first order attack i.e., it can be exploited directly using any of distinguishers presented in Section 3.2.4. Note, that even if an attack can be carried out using one point of a trace (when an implementation results in a first order leakage) attackers would usually choose to use several points for their attack (use higher dimensional attack), it allows to reduce the effects of noise and thus cut down the number of traces required for a successful attack. An implementation protected using a first order masking scheme should only result in a *second order leakage*, if the masking scheme is secure and it is implemented correctly. A second order leakage cannot be exploited using first order attacks and an attacker cannot obtain any information about the secret key by analysing only one point of a power trace. Thus, implementations that produce second order leakage force the attack to use second order statistical moments (variance) or to combine at least two points of a power trace to mount a successful attack. As a result, if someone finds a first order leakage in a masked implementation then we can say that the masking failed (either because the scheme is insecure or because it was not properly implemented). We can say that ith order leakage can be exploited only using ith statistical moment or through a combination of i points of a power trace. Thus, exploiting higher order leakage is more difficult.

3.4.2 Hiding

Hiding countermeasures against side-channel attacks act in the *horizontal* (temporal) domain of power traces. Main idea behind hiding countermeasures consists in changing the location of the point (in the power trace) that leaks the information. In other words, when a hiding countermeasure is applied the moment in time (from the beginning of the execution) when the leakage occurs is different for several different executions. Thus, the power traces are not *aligned* which means that a chosen offset in all traces will no longer correspond to the same operation and same intermediate value. Even if some traces are aligned (at the point of interest), it will likely not be the case for all traces of the dataset, as a result the misaligned traces will contribute to the noise, increasing it and making the attacker's life more difficult.

One of the most common hiding techniques (in software) is the rearrangement of independent operations, this technique is called *shuffling*. For example, the application of an S-box on the state is usually done in this way. Since an S-box is typically smaller than the internal state of a block cipher, the S-box is applied on every part of the state separately. Usually these parts stay independent (during this operation), thus the order in which the S-box is applied to different parts of the state does not influence the result. Therefore, shuffling can be easily applied to these operations.

Several shuffling techniques were presented in literature, the main differences between them consist in two features: (1) which operations are shuffled and (2) how a new arrangement is generated. These two features of the shuffling schemes give rise to their properties such as the required number of random bits and the number of possible permutations which influence the increase in the resistance against side-channel attacks.

Random Permutation (RP) [VMKS12] rearranges the execution of one operation (e.g., the application of an S-box). It consists in generating a random permutation of numbers between 0 and $N_q - 1$, where N_q is the number of operations to be rearranged. The generated permutation is used as the new order of operations. This technique requires additional memory (same size as the state) and up to several bytes of randomness, which can be an expensive resource in some lightweight devices. A technique called Random Starting Index (RSI) was introduced as a lightweight solution to the problem i.e., it requires less randomness and it has less memory overhead. The idea of RSI consists in generating a random index between 0 and $N_q - 1$ and executing the shuffled operation starting from this random index (going through all numbers $\pmod{N_q}$ to loop around from $N_q - 1$ to 0). RSI was applied on AES block cipher by Herbst *et al.* [HOM06] and by Tillich *et al.* [THM07]. RP as well as RSI propose solutions that rearrange the *same operations* applied to *different parts of the state*. Sometimes it is possible to rearrange *different operations*, a technique called scheduling (SchedAES [Med12], applied on AES) was suggested as a way of making it possible. The main difficulty of this technique consists in the fact that some

permutations are no longer possible (because some of the rearranged operations *are dependant*). The idea of SchedAES consists in creating a dependency graph that tells which operations are dependant and cannot be rearranged (this part can be done offline). A list of possible following (next) operations is kept up to date after executing every operation during the online execution of the algorithm. This technique allows to create more different permutations, but requires much more randomness and is much slower than other shuffling schemes.

A designer of a cryptographic system can use other techniques which will result in misalignment of power traces. In addition to shuffling, it is possible to use introduction of random delays and dummy operations to misalign traces[21]. However, these techniques were found to be less effective as a countermeasure against side-channel attacks, since counter-attacks can relatively easily (compared to other countermeasures) deal with random delays and with dummy operations because they only move the informative points across the time domain while keeping their order, one possible counter-attack is based on elastic alignment (dynamic time warping) [vWWB11] or special profiling techniques that are less sensitive to such problems [LMM17, WO15]. It is also possible to use hardware based hiding techniques e.g., dynamically modifying the clock frequency during cryptographic operations or using irregular clocks will also result in misalignment of power traces [MOP07, §7.2.2, page 173].

Even though hiding techniques prevent some side-channel attacks, attackers have found several ways of counter-attacking cryptographic systems protected with hiding. One way of attacking these implementations consists in going to the frequency domain [SDB+10] where the alignment is no longer required, it is also possible to try to align misaligned power traces using pattern matching or to use preprocessing techniques called integration [CCD00] (several points of the power trace are summed together). Another, way of attacking a hiding technique that uses randomness consists in attacking the random number generator ether by influencing it or by retrieving the randomness that was used during hiding in order to effectively remove its effects (as it was already demonstrated against masking countermeasures [LMBM13, LBM15b]).

3.4.3 Other countermeasures

There are several types of countermeasures against side-channel attacks that cannot be classified as nether masking nor hiding or in some cases they can be considered as grey area.

Leakage resilience is the idea that was suggested in order to protect cryptographic devices against side-channel attacks [SPY+10]. Leakage resilience is based on the idea that usually a power trace does not contain all information about the internal state of the device. Thus, the main assumption of leakage resilience states that it is impossible to extract the secret key using a very limited number of measurements. This tech-

[21] See implementations used for CHES Challenge in 2016. `https://ctf.newae.com/flags/`

nique consists in measuring and characterising the leakage of the device and computing how many encryptions (with corresponding power traces) are necessary in order to break the system. The advice given by the advocates of this technique consists in using *rekeying* before a given number of encryptions (which allows to break the system) was performed. Rekeying is a generation of a new secret key that is used for encryption (which can be e.g., based on some main secret). If an attacker cannot extract the secret key using a limited number of traces and the key is updated even more often then the attacker should not be able to extract the key at all. One of the ideas of generating a new key consists in using an update procedure which can be easily (compared to an entire block cipher) protected using e.g., masking. However, implementing leakage resilience appears to be difficult in practice, among other things the main hypothesis is challenged by some results of the attacks on the DPA Contest: several profiled attacks against an implementation *protected* against side-channel attacks with masking and shuffling can *reliably* break it using only one trace[22].

Several countermeasures against side-channel attacks can be labelled as "resistant by design". This category includes the design of S-boxes that are harder to attack (e.g., designed using CC [PPE+14] and TO [PMMB15]), creation of algorithms that were thought in such way that they could be easily protected using common techniques such as masking [GLS+15]. It can also include the design of key scheduling algorithms that are constructed in such way that it is practically impossible to compute the master key from a round key or to compute one round key from another round key (next or previous). A classification of ciphers based on whether the knowledge of a round key gives out information on the master key or on other round keys was introduced as a cryptographic property of ciphers (for the purpose of analysing the immunity of ciphers against some attacks) without the side-channel analysis component in mind [CDN98]. This kind of key schedules was not specifically designed to be a countermeasure against side-channel attacks. Nevertheless, such constructions force the attacker to analyse all rounds of a block cipher in order to get all round keys [MGV+16]. Several AES candidates including RC6 [RRSY98], Serpent [ABK98, ABKT98] and Twofish [SKW+98] use key scheduling algorithms that have this property (their round keys do not reveal information on other round keys nor the master key) [CDN99]. It is interesting to note, that some modern block ciphers often use a trick that allows to optimise their implementations by using the secret key or a part of it as the first round key (which means that a side-channel attack on one round will reveal bits of the master key) e.g., in AES-128 (Rijnael) [AES01, DR02] the first round-key is the secret key, while in PRESENT-80 [BKL+07] the first round-key is equal to the first 64 bits of the entire secret key.

Some hardware (non-algorithmic) countermeasures can try to limit the leakage itself, it can be achieved by using special logic styles such as Sense Amplifier Based Logic [TAV02] and Wave Dynamic Differential Logic [TV04]. Other hardware coun-

[22] See Hall Of Fame at `http://www.dpacontest.org/v4/42_hall_of_fame.php`

termeasures can limit the ability of an attacker to measure the power consumption or to study the device e.g., by using light sensors that can detect when the device is depackaged (which is required by some attacks [FH08]), when the light is detected then the sensor can activate a component which will erase the key before the attacker can extract it. Another hardware countermeasure consists in using an on-chip noise generator or in increasing the level of noise by performing several operations in parallel [MOP07, §7.1.2, page 170][23]. Hardware countermeasures can also include very low-tech ways of making the attacker's life more difficult e.g., by using screws with special heads that make it harder to get to the valuable (for the attacker) electronic components inside of the product.

3.4.4 Summary on countermeasures

There are many different kinds of countermeasures that try to act on different parts of the side-channel analysis issue. Some of them try to prevent the attacker from acquiring traces (screws, packaging, shielding, sensors and detectors), others try to limit the Signal-To-Noise Ratio (SNR) (noise generators, special logic styles) or to quantify the leakage (leakage resilience), moreover, countermeasures try to obscure the leakage by acting on the vertical (masking) or the horizontal (hiding) domains of the power traces (which ultimately also decreases the SNR) and finally algorithmic countermeasures (resilient S-boxes, specific key schedules) can also be applied at a design stage of ciphers.

All these countermeasures make side-channel attacks more difficult, in most cases an attacker can overcome these difficulties by acquiring more power traces, using more sophisticated attacks and learning more about the target device before actually attacking it. Nevertheless, each additional countermeasure cuts out another fraction of all the potential attackers who cannot overcome it.

Unfortunately a perfect countermeasure that can prevent side-channel analysis and stop all attackers does not exist. Therefore, applying several different countermeasures can at least greatly reduce the number of people who are potentially capable of overcoming all of them and perform a successful attack. Thus, usually designers use a solution that involves several countermeasures. For example, a combination of masking and shuffling is often suggested as a good countermeasure [THM07] since masking works better in presence of noise [SVO+10] and shuffling helps to increase the amount of noise. This combination was used in the DPA Contest 4.2 [BBD+14][24].

Overall, creation of new countermeasures that prevent side-channel attacks as well as development of ways that can overcome these countermeasures is a very typical example of an arms race.

[23] Some researchers put noise generators in the domain of hiding countermeasures, but we prefer to reserve hiding to the idea of spreading the leakage in the temporal domain.

[24] `http://www.dpacontest.org/v4/42_doc.php`

3.5 Summary

Classical cryptanalysis looks at cryptographic primitives from the point of view of statistics and mathematics, while side-channel analysis takes into account the implementation of the analysed algorithm. Side-channel attacks are the attacks on the cryptographic hardware, they take advantage of the physical characteristics of a system in order to attack it, it is possible because a cryptographic system can leak the information about its internal state through the physical properties such as power consumption, execution time, sound and many others.

There are many different ways of grouping side-channel attacks: based on the source of information, invasiveness (method of acquiring the measurements), interference (way of interacting with the device), leakage model (profiled or unprofiled) as well as the type of analysis (simple or differential). All these types are independent between each other. Every type of attack comes with its benefits and drawbacks that are related to their costs (equipment, complexity and computational resources).

This work focuses on passive non-invasive differential power analysis of both profiled and unprofiled types. Any side-channel attack based on power analysis has five major points that has to be addressed: (1) the acquisition of power traces, (2) selecting a target intermediate value (of the algorithm) for the attack, (3) choosing (unprofiled) or extracting (profiled) a leakage model, (4) choosing and applying a distinguisher or in other words using the statistical tools for the data analysis (which is also strongly related to the choice of the leakage model) and finally (5) performing the final key enumeration in order to get the full secret key.

Since there is a large variety of different side-channel attacks and each attack has many different phases and steps, evaluation of side-channel attacks is not a trivial task. Many different performance evaluation techniques were developed over the years including properties that can be computed experimentally (average number of queries, success rate and guessing entropy) as well as theoretical properties that do not require experiments (transparency order and confusion coefficient).

Numerous countermeasures were suggested as a remedy against side-channel analysis. Many of the countermeasures try to prevent the attacker from acquiring the data (measuring the physical properties), other countermeasures try to make the analysis difficult by spreading the leakage over the temporal domain, by randomizing it in the vertical domain and by increasing the amount of noise in the system. Countermeasures are applied in software and in hardware as well as on several layers of abstraction. Each new countermeasure makes the attack more difficult, but since a perfect countermeasure does not exist several different techniques are generally applied to protect the cryptosystem and to discourage more potential attackers.

For more information about side-channel attacks based on power analysis as well as countermeasures against such attacks we suggest the book called *"Power analysis attacks — revealing the secrets of smart cards"* [MOP07].

Chapter 4

The problem of leakage detection

Performing a full side-channel attack against a given device goes through many steps: mounting the physical setup, power trace acquisition, preliminary analysis of the device and the statistical analysis of the dataset. This process requires to choose a lot of different parameters: target intermediate values, leakage model, type of attack, distinguishers, etc. Therefore, a full side-channel analysis of a given implementation often requires (and can greatly benefit) from expertise in physics and electrical engineering, cryptography and cryptanalysis, applied statistics and mathematics as well as from software and hardware engineering. Thus, side-channel attacks require a lot of knowledge in different domains due to the multidisciplinary nature of the tasks associated with the analysis. This idea is applicable both in terms of designing a new implementation and evaluating its security (trying to break it), since the knowledge about attacks *does* help to design stronger countermeasures.

Unfortunately, the complexity associated with side-channel analysis results in the fact that it is actually very difficult to evaluate the security of a given device from the perspective of side-channel attacks. Moreover, when a cryptographic implementation is evaluated, ideally we do not want to find a flaw or one attack, but to *either* find all flaws (to fix them) *or* to show that a given implementation does not have any information leakages exploitable by an adversary of a given strength (related to the amount of resources that they have). In addition to that, usually evaluation labs have to perform the analysis in a very short period of time often measured in weeks. What makes the whole campaign of evaluation of side-channel leakage even more challenging is the fact that it is much easier to find *one* working side-channel attack against an implementation than to find all of them and it is often impossible to prove that there are no attacks against the implementation (i.e., the device does not leak exploitable information). After all, the security of modern block ciphers (and other cryptographic algorithms) is not based on a proof that a given algorithm cannot be attacked, but on the fact that, as it is often stated: a lot of really smart people tried

very hard to break the algorithm for a very long time and nobody was successful[1].

4.1 Leakage detection

Real world devices need security evaluation before going on the market. Even if attackers come up with new ideas all the time, a security evaluation will at least ensure that there are no obvious flaws that anyone can exploit. Evaluators face a limited attack scope (computational power and time available for the evaluation) and thus cannot "simply" run all known side-channel attacks. Evaluation in general is an open problem, to address this problem researchers developed methods of *leakage detection* [CDG$^+$13]. In leakage detection an evaluator prioritizes the detection over exploitation which allows to speed up certain evaluation aspects. Even though leakage detection does not necessarily tell us *why* and *where* the leakage occurs and *how* to exploit this leakage, it is often the preferred method due to its rapidity and simplicity. In other words, these methods do not reveal the information about the difficulty of an attack, nor which intermediate values should be targeted, nor about the appropriate leakage model; they just try to detect the presence of leakage.

One of the most commonly used methods is based on the Welch's t-tests, the goal of using such tests is to relax the dependency between the evaluation and the nature of the device (leakage functions, architecture). The main idea of a t-test is to compare the averages of two sets $\mathcal{S}_0$ and $\mathcal{S}_1$ and conclude if they are different (distinguishable) or not. The null hypothesis is that samples (values) were drawn from the same population or in other words that the two sets are indistinguishable:

$$\begin{aligned} H_{null}: &\quad \mu_0 = \mu_1 \\ H_{alt}: &\quad \mu_0 \neq \mu_1 \end{aligned} \tag{4.1}$$

where μ_0 and μ_1 are the mean values of sets $\mathcal{S}_0$ and $\mathcal{S}_1$ respectively. The t-test value t is computed using the formula:

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{\sigma_0^2}{N_0} + \frac{\sigma_1^2}{N_1}}} \tag{4.2}$$

and the degree of freedom v is computed using:

$$v = \frac{\left(\frac{\sigma_0^2}{N_0} + \frac{\sigma_1^2}{N_1}\right)^2}{\frac{\sigma_0^4}{N_0^2(N_0-1)} + \frac{\sigma_1^4}{N_1^2(N_1-1)}} \tag{4.3}$$

where N_0 and N_1 denote the number of elements in sets $\mathcal{S}_0$ and $\mathcal{S}_1$, while σ_0^2 and σ_1^2 denote the variances of the sets with corresponding indexes. Then, normally the

[1] At least no one who succeeded announced it publicly.

degree of freedom υ is used in the Students's t distribution which in its turn can be used to estimate the probability of rejecting the null hypothesis within a chosen confidence interval. However, due to the nature of the data under analysis these steps are usually skipped. Indeed, the evaluator can choose the number of samples (i.e., the number of traces in each set) and since we are comparing points of power traces with same points (offsets from the beginning of the trace) related to traces acquired with different inputs we know that $\sigma_0^2 \approx \sigma_1^2$ (both variances actually depend on the acquisition setup and on the hardware that performs cryptographic operations). These simplifications allow evaluators to use the t value directly in their tests [SM15]. Usually this value is compared to the constant 4.5, more specifically $|t| > 4.5$ allows to reject the null hypothesis. The absolute value of t being greater than 4.5 corresponds to the confidence of 0.99999 to reject the null hypothesis for $\upsilon > 1000$. By looking at the Equation 4.3 we can see that the value of υ grows when the number of samples N_i increases (while the variance σ_i^2 stays fixed). Once again, since the evaluator can choose the number of samples, they can acquire enough power traces to get $\upsilon > 1000$; for example: with $\sigma^2 = 0.1$ and 502 traces in each set already give $\upsilon > 1000$. In the evaluation context, rejecting the null hypothesis implies potential evidence of side-channel leakage.

In case of power analysis, the same t-test is applied on all points of power traces i.e., points from the same offset from the beginning of the trace are used in one t-test which is repeated for all offsets. More specifically, evaluators usually perform a *non-specific* t-test which does not rely on a chosen leakage model (*because* we do not know the best leakage model). Most commonly, the *fixed vs. random* non-specific t-tests are performed. In such tests, the evaluator fixes the secret key and then acquires two sets (of the same size) of traces related to either a chosen fixed plaintext or to a randomly generated plaintext. The t-test is then performed on these two sets of traces.

Note, that the same t-test is repeated with many sets which correspond to different points of power traces. It means that the confidence of rejecting the null hypothesis for the full dataset (i.e., having a potential leakage *somewhere* in a trace) is not equal to the confidence of one t-test (potential leakage at *one point*). Since we are essentially repeating the experiment (t-test) many times, the probability of getting at least one false positive increases. Thus, in order to get a chosen confidence of e.g., 0.999999 we need to either adapt the t value or to change (increase) the number of traces in both sets [ZDD$^+$17].

It is important to understand, that results produced by a t-test are limited. A t-test does not give us a final proof that an attack on the analysed implementation does (or does not) exist. The t-test tells us about the possibility of distinguishing two sets, which can lead to an attack. However, if a t-test rejects the null hypothesis ($|t| > 4.5$) it does not mean that the implementation is easy to attack because we still do not know which intermediate value should be targeted and what leakage model should be used; the found leakage can be very hard to exploit in practice because it

can potentially require to guess a very large part of the key during the attack (leading to an exhaustive search in a large space). Thus a rejection of a null hypothesis should be ideally backed up with an attack to demonstrate the exploitability. Moreover, if a t-test does not reject the null hypothesis ($|t| < 4.5$) then we cannot imply that the implementation is completely secure, it is mostly due to the fact that the t-test analyses one point (offset from the beginning of power traces) at a time. A higher order attack (which takes into account higher order statistical moments) or a higher dimensional attack (that combines several points of a trace) can lead to a successful key extraction on a device that passed the t-test. A t-test can also fail due to the lack of traces (the dataset is too small), however the evaluator is in control of this particular parameter, thus it is a smaller issue. Higher order t-tests try to deal with this issue, but they are less attractive in practice, since they are relatively slow (e.g., require to combine points which lead to a combinatorial explosion), even though there were some works that try to limit the number of tests that have to be performed in such settings [MO17]. Nevertheless, a t-test that does not reject the null hypothesis at least gives the evaluator the confidence that an attacker should not be able to mount a successful attack using only one point per trace, we can say that it does not prove that there is no attacks on the analysed implementation but that it excludes a small subset of (rather simple) attacks. Thus, results of t-tests have to be taken with a grain of salt in both cases whether the null hypothesis was rejected or not.

Other techniques that try to solve the problem of evaluation of cryptographic implementations are generic side-channel distinguishers and extensive profiling techniques [BGP+11, WOM11, SMY09].

4.2 Analysis during early stages

Regardless of the exact evaluation technique they all require a set of power traces, in other words the evaluator actually needs the product in order to analyse it. This very down-to-earth idea has one undesirable property: the developer has to finish the product before getting its security evaluation. As a result, if an issue is discovered at the final stage of development (i.e., on the final product) than the development of some parts of the product has to be restarted; it actually concerns the part at fault (that caused a security flaw, a side-channel leakage) as well as, potentially, parts that depended on it. The rationale is that if a flaw in a product is discovered at its last stage of development, the development has to roll back and restart from some previous stage. This issue results in additional costs (related to engineering and to the new security evaluation) and ultimately slows the whole process of development. Therefore, developers (and especially businesses) are very interested in ways that allow to detect any security-related issues, including side-channel analysis, in early stages of development. This issue relates to the idea of testing and bug hunting in the IT industry: a continuous unit-testing of software under development does help to reduce

the number of errors in the final program.

The main point is that continuous security-evaluation in terms of side-channel analysis in all stages of development can be beneficial and can reduce costs related to the development of secure products. However, normally in case of side-channel analysis, the evaluator does not have an object that they can evaluate while the product is not ready. Moreover, the product changes all the time during its development and it is important to understand that a new line of code in software or a couple of new transistors in hardware can potentially break the security of a previously sound system. Overall, it makes the problem of side-channel analysis of an unfinished product even more challenging.

The same principles and ideas can be applied both in hardware and software implementations of cryptographic algorithms, however we would like to focus our work on the software implementations. A typical process of the implementation of a cryptographic algorithm (in a broad sense) goes through many phases that are not necessarily performed by the same person (or team). We would like to focus our attention on the following stages:

- design of algorithms, which includes both the block ciphers as well as algorithmic countermeasures;
- choice of hardware (microcontroller) and of the implementation details for the block cipher and algorithmic countermeasures (*which* algorithms to implement and *how* to implement them);
- writing the source code i.e., implementing the block cipher with all the chosen countermeasures;
- building the binary or in other words compiling the code for the chosen hardware;

the final step consists in loading the binary executable into the microcontroller, at this stage we consider that the product is finished and a final evaluation can be done using real power traces. In practice some of these steps are done in parallel and independently of each other e.g., different cryptographers can develop generic algorithmic countermeasures while another team develops a block cipher. Even though the evaluation of an algorithm (a cipher or a countermeasure) at the design stage might seem to be far fetched, researches are already trying to design encryption schemes that are easier to protect with masking [GLS+15] and even S-boxes that are more resistant against side-channel analysis by design [PMMB15, PPE+14].

Different devices have different leakage functions and the real leakage evaluation can be done by actually measuring its power consumption. Nevertheless, evaluation during the very early stages (design of an algorithm) could still make sense because a lot of devices are similar (same structure and architecture, same technology and

manufacturer) and therefore they leak in similar ways (value-based leakage or distance based leakage), this is the reason why Hamming weight and Hamming distance leakage models work well (are good enough) to break some implementations using non-profiled attacks.

If we look closer on the implementation stages that we are dealing with, we can notice that the evaluator (or designer) does not deal with the same type of issue during the whole process. On one hand, during the design of an algorithm and during the choice of implementation details we deal with the *choice problem* i.e., we want to compare several designs or implementation options e.g., we want to know which countermeasure to implement given its costs and benefits (time and memory overhead, RNG requirements and increase in the resistance against side-channel attacks compared to a non-protected implementation). On the other hand, during the actual implementation we are dealing with the *flaw detection problem*. This separation of steps actually helps us: at the beginning we do not know how the device leaks (because we do not know which device will be chosen) thus we can only perform comparative analyses but this is exactly what the designer wants (to choose the better option among several ones); while at the end of the process the implementation is already tight to a device and the evaluator can already use their knowledge of the e.g., microcontroller specifications to try to detect side-channel leakages in the implementation.

Since during the design and implementation process power traces are not available yet, the evaluator can only work with the information on their hands — the output of each stage. At the end of each of the considered steps we get a partial specification of the final product: either a high level source code (e.g., for the prototype or a reference implementation of the algorithm), low-level assembly code (usual choice for the actual implementation of cryptographic algorithms) or the compiled binary. Note that the program gets more precise and device specific at each step: while the high level source code is very generic, the assembly implementation is already tight to a specific architecture (instruction set) and the compiled code is device specific (chosen model of a microcontroller). In other words, we are dealing with the code on different levels of abstractions during the whole process.

Intermediate stages of design and development of cryptographic systems provide us with design specifications (in form of the source code in case of software) means that any security evaluation can be done through static or dynamic analysis of the code. A very interesting leakage detection procedure can be based on simulations. A simulation can produce simulated power traces that we can normally record using an oscilloscope (in case of a real experiment). These simulations can be based on the code and specifications available at the current stage of development. Finally, any of already existing leakage detection techniques, analysis and attacks can be applied to these simulated power traces. Moreover, any new leakage detection techniques and new attacks that use power traces as their source of information that are developed

in future can also be applied on the simulated data.

Some ideas going in this general direction were suggested in the work on the Side-Channel Resistant Design Flow (SCARD) [AMM+06]. However, most of the results related to simulations for side-channel analysis is not available to scientists who work on the subject of side-channel analysis, as it will be explained in Section 5.3.

4.3 Goals

The aim of our work is to provide simulation tools for the analysis of implementations of cryptographic algorithms in early stages of their development. To be more specific the first goal is to provide a series of tools that can:

- help designers to compare cryptographic algorithms from the point of view of side-channel attacks,
- allow to compare the impact of a counter-measure applied on an algorithm,
- guide developers in the process of implementing countermeasures and
- assist evaluators in detecting side-channel leakage before deploying the code in the physical device.

Providing a way of comparing designs and implementations should also lead to design improvements, since a comparison technique usually leads to the way of improving or optimising the design criteria. Creating a series of automated tools should also increase the evaluation speed and thus reduce costs related to the security evaluation and development.

Our second goal is to show how these simulation tools can be used during the analysis and evaluation of cryptographic algorithms. This will allow us to evaluate the usefulness of our tools as well as show their strength and limitations.

Overall, we would like to answer the following question: is it possible to evaluate the security of a device from the point of view of side-channel attacks while the device is still under development? If it is possible then how can we do it?

Part II

Contributions

Chapter 5

Simulation tools for side-channel analysis

Parts of this chapter are based on the paper *"Use of simulators for side-channel analysis"* presented in Paris (France) at the Workshop on Security for Embedded and Mobile Systems (SEMS) in 2017 [VG17b].

A simulator is a tool that provides test conditions which approximate the reality, such tool reproduces the behaviour, on some specific aspects, of another object and provides operational conditions under which the tested (or analysed) subject is normally used or will be used once its development is finished.

In case of side-channel attacks based on power analysis we are interested in simulating the instantaneous power consumption of a device over some period of time. As we explained in Section 3.2.3, we assume that the total power consumption is composed of three parts: the instruction (operation) dependent part P_{op}, the data dependent part P_{val} and the noise ϵ, see Equation 3.7. We can simulate one, two or all the three parts of the total power consumption depending on our interests. For example, if we are interested in distinguishing different instructions for the purpose of SPA [KJJ99, May00] or for Side-Channel Analysis Reverse Engineering (SCARE)[1] [Nov03, Cla04, CIMW15] we need to simulate the operation dependent part of the power consumption, while in case of DPA [KJJ99] we are mostly interested in the data dependent part (small variations in energy consumption that depend on the value that is being handled, see Section 3.2.3). The noise part of the data could be interesting for any type of analysis, thus it is often used in all kinds of simulations.

[1]The main goal of this type of investigations through side-channel analysis is the discovery of how a system works and how it was designed. One of the main techniques that is used in SCARE is the discovery of the code that is executed by the analysed device through the analysis of its power consumption. It is possible because different instructions result in different patterns in power traces (same idea is used in SPA).

Any simulator uses a model to represent the underlying reality of the simulated phenomenon. This model is a set of rules that simplifies the reality to a more abstract idea. Depending on the complexity of the model (the number and the type of its parameters as well as their relations and interactions) that is used by a simulator we could refer to the model as being more or less abstract. Thus, simulators could be classified based on their *level of abstraction*. Generally, simulators of a higher level of abstraction tend to require less parameters (less information about the simulated target) and less computational power to process these parameters, they are more generic and they are often less accurate and less precise than their lower level of abstraction counterparts. There are many different levels of abstraction that go gradually from the most to the least abstract. For example, when we talk about simulating the power consumption of a microcontroller, a very high level of abstraction simulator might use the general architecture of the device. In order to go deeper we can use more information and include details on the microarchitecture (microinstructions), description of circuits at the level of logic gates, positions of transistors and connections between them in the device to get to one of the lowest levels of abstraction. Using a lower level of abstraction will generally yield better, more accurate results, but since they require more knowledge about the target device we cannot always use them, manufacturers of microcontrollers and processors tent to hide most of the information about their device (such as e.g., the full logic gate circuits that constitute their product). The rationale is that depending on the available information about the target device we can turn to the appropriate simulator. It is interesting to note that we can still build a very accurate simulator without huge amount of knowledge about the internals of a target device, in order to do so one can use profiling. The profiling can be done in the same way as the profiling step of a profiled side-channel attack (such as TA or SA, recall Sections 3.2.4, 3.2.4 and 3.1.4). In case of profiled simulators we do not need to perform the attack step. Only the learning step is required to extract a profile, such profile can be created for each instruction from the instruction set of e.g., a microcontroller. In other words, the idea is to extract a leakage model from the target device in order to build simulated traces by e.g., concatenating simulated traces of the instructions that constitute a program (while executing it in order to know which values have to be used with the model) [DMO16]. A resulting profiled simulator will be very accurate, but in order to be able to simulate several devices one would need to profile every single one of them separately.

In case of side-channel attacks based on power analysis we will distinguish two types of simulators: *data generators* and *verifiers* (or *checkers*).

Data generators produce *simulated traces* that are in some sense equivalent to real power traces measured using an oscilloscope. A data generator requires two types of inputs: a *description of the cryptosystem* and a *leakage model* (or a *power model*) that allows to produce a simulated trace. The leakage model parameter is the same type of function that an attacker can use with distinguishers during side-channel at-

tacks (thus we use the term leakage model for both of them), see Section 3.2.3. The description of a cryptosystem is either a *program* or a *circuit* depending if the system is implemented in software or hardware. In case of a program it could be a source code (even pseudo code) or a compiled executable file; in case of a circuit it could be e.g., a gate-level netlist, hardware specifications ranging from the architecture-level down to the transistor level, including masks used by (semiconductor) foundries or even a piece of program written in a hardware description language (such as `VHDL` or `Verilog`). In other words, the description of a cryptosystem in our case is any formal description that allows you to create the final product i.e., something that a designer of the system uses at any stage of the development process.

Checkers, the other type of simulators, do not produce simulated traces, but rather verify the implementation for a specific property related to side-channel analysis e.g., the absence of execution time differences that depend on the processed data (which could lead to a timing attack). A checker would also use a description of a cryptosystem as an input and it would output a (possibly empty) list of issues related to the property that it is checking. Thus, a checker can often point out the existence of a specific issue, while a data generator produces a dataset that has to be analysed with any tools chosen by an evaluator (for example, one can simply perform an attack on the simulated traces).

Some simulating tools include statistical analysis functionalities (e.g., t-tests, correlation analysis) and can output a "yes or no" type of answer with respect to whether a leakage was detected. Nevertheless, we will put these tools in the category of generators rather than checkers since they start their analysis by producing a set of simulated traces.

It is important to note that both types of simulators are useful in practice. Even though a verifier can immediately point out the existence of a problem without generating simulated traces, some issues cannot be detected using a verifier without using a set of traces. For example, it is easy to check whether a piece of code contains conditional branches that depend on the (secret) intermediate state, but it is not always possible to say if such conditional branch generates a leakage that can actually be exploited by an attacker.

In the scope of this work we are interested in data generators as well as in checkers of relatively high level of abstraction. In terms of data generator simulations we will focus on the data dependent part of the power consumption (P_{val}). Note, that some properties of a program can be verified using static code analysis that does not require to simulate the code. We will take a closer look at the tools that use dynamic analysis of the code either by executing it several times with different inputs (in case of data generators) or by executing it while tracking how values are being transferred, combined and modified during an execution without using a concrete specific value for the execution itself (in case of checkers). It is worth mentioning that checkers can sometimes use algorithms that cannot be easily put into a category of static or dy-

namic analysis and the frontier between the two can change a little bit depending on their definitions. We will say that a checker is a simulator if it has to go through the code in the order of instructions (from the beginning to the end) while keeping track of values that are being manipulated by the instructions. Thus, we will put tools that can perform code analysis in an order that does not necessarily correspond to the order of instructions (e.g., that "jumps" around the code or goes through several cycles of analysis) into the category of static analysis. For example, QMS [EWTS14] uses static analysis of `C` code to estimate the amount of information leakage in masked implementations.

5.1 Motivation

Simulators have several advantages over real "physical" experiments. These advantages are common to most of simulators that are used in different domains but we are going to focus our attention on side-channel analysis.

First of all, a simulator provides a full control over the environment where every single parameter is under control of the experimentalist. This type of fine-grained control is practically impossible in a real experiment. For example, one can set a specific SNR in a simulated environment and change it in fixed steps in order to study how an algorithm behaves under different SNR. In lab conditions, the level of noise could be reduced by using filters, by putting the setup into a Faraday cage[2] (to get rid of electro-magnetic interferences) and even by lowering the temperature in the lab (to reduce the thermal noise), but it is extremely difficult to create a setup that reliably gives a specifically chosen level of physical noise[3]. Moreover, a simulator allows to set the level of noise to zero which is impossible in a real experiment. Thus, it can be used to get a "clean" signal, which allows to test the quality of signal processing algorithms e.g., we can compare a clean signal with a result of a filtering algorithm applied on a noisy signal. Furthermore, a simulator allows to set a specific, *known* leakage function, which is unknown in a real scenario. It allows to test whether an algorithm can successfully and reliably extract a good leakage model from a noisy dataset i.e., since we know the leakage function we can compare it with the extracted leakage model.

The full control over the simulated environment also allows us to ignore some aspects of a real experiment *by choice*. For example, a simulator can output perfectly aligned traces. During a real experiment, traces can be misaligned due to the clock jitter, imperfections in the measurement equipment and even because of some countermeasures (recall Section 3.4.2). It is possible to build a simulator that is in some

[2] An enclosure (a box or a mesh) that is built from a conductive material, it is used to block electromagnetic fields.

[3] The algorithmic noise being fixed by the executed code can be controlled by the developer and thus can be easily reproduced.

sense unaware of timing i.e., it produces the same number of points regardless of the number of clock cycles used by an instruction. Ultimately it allows to detect differences in the control flow of an execution if it somehow depends on the input data. Some popular microcontrollers and processors have instructions that do not execute using the same number of cycles depending on the processed data e.g., Cortex-M4 processor has instructions (e.g., `ADD`) that can be executed in different number of clock cycles depending on the context (such as the content of registers and the state of its pipeline) [ARMb, §3.3.1]. It can also be influenced by the dual-instruction issue restrictions (related to pipelining) such as in Cortex-A8 [ARMa, §16.3]. Another example is the `BRCC` (branch if carry cleared) instruction of the ATmega-16 microcontroller, it can be executed in 1 or 2 clock cycles depending on the value of the carry flag; we explore this particular issue in detail on a concrete example in Section 8.2. The rationale is that a simulator can help to find and highlight problems that are not very easy to detect in real experiments.

Another advantage provided by simulators is their speed. The acquisition of real power traces can be automated and it usually *is* done automatically to some extent. However, the setup is generally done manually[4] and it is a task that can be quite time-consuming. It could take up to several hours for relatively complex setups. For example, the time spent on *simply putting together and testing the setup* (no trace acquisition!) for CHES 2016 "Capture the Flag" challenge[5] required about 60 man-hours according to Colin O'Flynn (who managed the challenge). However, we would like to emphasise that it is indeed a *complex setup* compared to setups of most labs. Once it was finished, the acquisition time for 1 000 traces (without averaging) captured using their setup was relatively short (around 5-10 minutes)[6], mostly due to the fact that their equipment captured only 1 point per clock cycle during 10 000 cycles[7]. For comparison, in our lab we can capture 10 000 traces with averaging over 16 traces and 30 000 points per traces (200 MSamples/s) in about 4.5 hours (it includes the time for mounting the setup, the acquisition and the data transfer to a computer for further analysis). The setup time also includes the time that is needed to check that the setup and the implementation are running correctly, which is not obvious at all since even experienced people make errors, see Figure 5.1. In addition to the short setup time of simulators, if many experiments that require some changes in the setup have to be done, several simulator instances can be launched in parallel to speed up the process; in order to do several real acquisitions in parallel one would have to buy more hardware (oscilloscopes, probes, target devices and measurement circuits). Moreover, time that is spent on transferring the data from the acquisition

[4]All side-channel analysis labs that we have visited do not have any robotic equipment that is able to automate the setup.

[5]`https://ctf.newae.com/`

[6]`https://wiki.newae.com/CHES2016_CTF#Getting_Help`

[7]`https://wiki.newae.com/CHES2016_CTF`

environment to the analysis environment (if it is not done in the same place) also slows down the whole process. When a simulator is used, traces can be generated and analysed inside of the same program i.e., it is not necessary to read traces from a file (which can also slow down the analysis in case of a big dataset). Even though modern computers are fast and data can be transferred quickly, a dataset of traces that is used in a side-channel analysis can still be described as cumbersome, see Table 5.1 (note that each dataset is compressed), thus time that is spent on reading and transferring data is not negligible.

We cannot claim that our setup is very efficient in terms of the acquisition speed. However, for comparison, the DPA Contest 4 setup takes a little bit more than 2 hours and 50 minutes to acquire 10 000 traces[8]. Note that this time does not include the time spent on the physical setup and on the data compression and transfer from the acquisition platform to the analysis platform. This time is also comparable with the time required by the setup used during CHES 2016 Challenge: 5-10 minutes for 1 000 traces which gives approximately between 1 and 2 hours for 10 000 traces (for short traces with only one point per clock cycle). Thus, our acquisition speed seems to be representative of other side-channel analysis labs.

Note that running *one* simulation is slower than performing *one* real encryption (especially if both are performed on the same hardware). However, simulators do offer a speed gain thanks to the following points:

- the setup — a real experiment requires a time-consuming manual hardware setup, which is not the case in a simulated experiment;
- the speed of the execution unit — a microcontroller executing the code in a real experiment is typically running at a frequency of megahertz (e.g., 3.57 MHz in DPA Contest 4 and 16 MHz in our experiments), while a simulation is usually done on a modern computer that uses a several gigahertz clock (resulting in speed increase up to 1 000 times);
- the ease of parallelism — several simulations can be done in parallel using several processing units (e.g., during one of our experiments we were running 64 parallel simulated experiments using a cluster available at the university's computing center), performing several real experiments in parallel requires to copy the hardware setup perfectly which takes additional time, money and is much harder to do than to lunch several copies of software (for a simulation);
- the memory bottleneck — in a real experiment, traces are recorded on the disk (in order to be used later), disk Input and Output operations (I/O) are slower than the same operations in e.g., RAM, an oscilloscope has to record a trace during an encryption and then transfer it to the disk which means that it cannot

[8]Information based on the meta-data from the reference traces from `http://www.dpacontest.org/v4/traces/rsm/DPA_contestv4_rsm_00000.zip`

Table 5.1 – Sizes of compressed datasets of DPA Contests. The third version of the contest had a different format, thus it did not provide any datasets. More information is available on the official website of the contest `http://www.dpacontest.org/`.

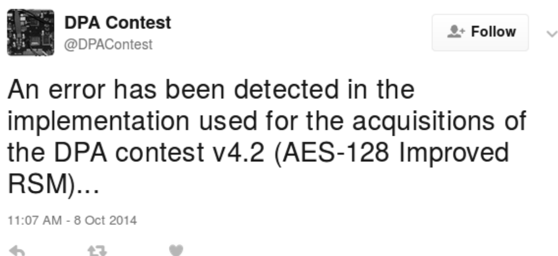
Version	Year Launched	Size Gbytes	Compression Format
1	2008	11.0	`.zip`
2	2010	8.9	`.tar.bz2`
4.1	2013	20.0	`.zip`
4.2	2014	58.5	`.zip`

record a new trace during this time, moreover all the data has to be transferred from the acquisition platform to the analysis platform which also takes time, meanwhile a simulation can be performed immediately on the analysis platform and in the same piece of code thus creating and keeping simulated traces in RAM the whole time.

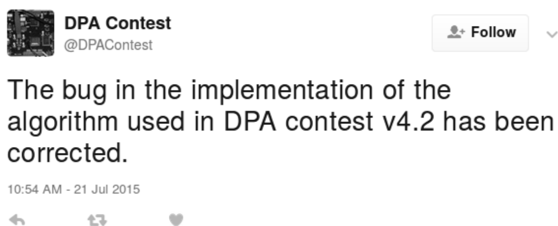
All these factors together contribute to the speed-up in simulated experiments compared to the experiments based on real power traces acquired using an oscilloscope.

Additional advantage provided by simulators is the fact that they could be used to test an unfinished product while it is being developed. Since a simulator uses an abstraction of a real system, the device does not have to be produced or to be fully functional. It allows to test a system that is being developed before actually making the final product. Tests and problem detection in early stages of development can save a huge amount of resources (both time and money), especially in case of an ASIC system (once the hardware chip is manufactured it cannot be updated as a piece of software). Moreover, a more advanced slightly lower level of abstraction simulator (that requires more inputs and can take into account more phenomena) can be used at each new stage of development while product specifications become more and more precise.

Finally, a simulated trace produced by a simulator is more "reliable" than an acquired trace from the point of view of repeatability of the results such as e.g., the success rate of an attack. This property is extremely valuable from the scientific point of view. Compared to a real experiment, a simulator can reliably reproduce exactly the same dataset assuming that same parameters are used (it also includes noisy simulated traces, since one can use a pseudo-RNG with a known seed). In case of a real experiment, if two experimentalists use identical setups (same model of microcontrollers, models of oscilloscopes, etc.) in their labs they will not get same sets of power traces. The environmental noise in their lab will be different and small imperfections in the target device as well as in the setup will also play their role in the final result: acquired traces will be different, which means that the two experimentalists will likely



(a) October 2014. A problem is discovered.



(b) July 2015. Correction is announced.

(c) August 2015. A new problem comes up.

Figure 5.1 – Messages from the official Twitter feed of DPA Contest (`https://twitter.com/DPAContest`) announce problems discovered after the data acquisition.

end up with slightly different results. This difference can be partially overcome by analysing *very large* datasets which takes a lot of time, but will allow to reduce the effect of noise. However, even tests made in the same lab with the same equipment on different target devices (same model from the same manufacturer) give rise to different results [RSV+11]. Moreover, certain physical properties of devices can change over time [RFFT14] which can be reflected in the leakage function of the device and thus modify the outcome of a side-channel attack. Nevertheless, it is important to note that these changes over time and variations among several devices of the same model result in relatively small but noticeable changes in e.g., the success rate of a side-channel attack; they do not give rise to changes of orders of magnitude in terms of side-channel analysis. In addition to repeatability, results produced by simulators take less space (bytes) that a dataset of a real experiment (several megabytes of code and initial parameters vs. several gigabytes of traces, see Table 5.1), this idea can also be used to the advantage of scientific collaborations: parameters that were used for a simulation can be easily sent by e-mail to a colleague and put in a paper within the results of experiments, which is not the case of real power traces[9].

We have mostly focused on the advantages of simulations over real experiments in order to motivate our research and the advocated strategy of side-channel analysis. However, it is important to note, that there are some disadvantages associated with the simulated methods. First of all a simulation can never replace the real experiment, because they are not identical. Therefore, even after performing many simulations for side-channel analysis, a real physical experiment should be done to evaluate the security of the final product. Another disadvantage of simulators is associated with their creation and development. It is actually faster to perform one real experiment than to build a simulator and perform one simulated experiment, mostly because a good simulator is a complex piece of software (which can also contain bugs). However, the development costs (of the simulator) can be quickly amortised if it is used for multiple simulated experiments.

5.2 Levels of abstraction

One of the main reasons for using simulations is their ability to discover issues related to side-channel analysis during the process of development of cryptographic systems. The development process goes through different stages, these stages vary in their level of abstraction. Each next step in the development process brings the implementation to a lower level of abstraction (e.g., from the pseudocode of an algorithm

[9]Public datasets of the DPA Contest (`http://www.dpacontest.org/`) and CHES 2016 "Capture the Flag" (`https://ctf.newae.com/`) partially deal with this problem. Cryptographers can use exactly the same dataset (which should give same results with the same analysis techniques) and can reference the dataset in their papers. However, use of such datasets is limited to the implemented algorithms and devices that were used during the acquisition.

to its source code in C++). Each lower level of abstraction brings in more details and specifies the structure of the system, in other words it gets the representation of the cryptosystem closer towards the concrete final product. Thus, each next step results in a description that contains more information about the final implementation.

A simulator is usually operating at its own level of abstraction while running a simulation. Its level of abstraction is mainly determined by the type of inputs that it receives. Thus, we will be classifying simulators by their level of abstraction based on the level of abstraction of their inputs. Note however, that a simulator can use only a part of information that is available on the level of abstraction which it is operating on (for example it can be useful to run a simulation faster). Lower levels of abstraction representations of the final product contain more information. From the perspective of side-channel analysis it means that we will be able to detect more implementation specific issues further down the line during the development process.

To be able to classify the existing simulators we would like to suggest the following classification of the intermediate states of an implementation. This work mainly focuses on microcontrollers and on the software implementation of cryptographic algorithms, thus we will mostly detail the stages of software development. Here are the intermediate stages of development of a software implementation that we will use to specify the level of abstraction of a simulator:

- A0 *Algorithm* — a mathematical description of the functionalities,
- S1 *High-level source code* — all variables are assigned, datastructures are described and the global order of operations is given[10],
- S2 *Intermediate representation* — the intermediate representation used by compilers, at this stage all registers are *described* as a Static Single Assignment (SSA),
- S2 *Assembly* — all registers are *allocated* and the exact order of instruction is known,
- S3 *Compiled binary* — the code is distributed into sections, alignment is performed, the file is ready to be executed by the chosen model of a device.

Note, that these levels of abstraction are related to the implementation of an encryption algorithm as well as to algorithms of countermeasures such as masking and shuffling. Each intermediate state allows us to detect more issues related to the side-channel analysis. Use of automated tools at each step allows to detect problems early in the development process. Thus we can use a methodology that consists in producing the next representation of a lower level of abstraction and proceed by running

[10]By the global order of operations we mean the following on the example of AES: we may choose to apply all AddRoundKey operations on the state followed by all SubBytes operations *or* we may choose to apply an AddRoundKey immediately followed by SubBytes on the same byte before processing the next one ($\forall i : state[i] = p[i] \oplus k[i]; \forall i : state[i] = S[state[i]]$ vs. $\forall i : state[i] = S[p[i] \oplus k[i]]$).

a simulation at this level, if a problem is detected at this stage the developer can go back *one* step and make changes in the design. Note, that if we can somehow guarantee that some issues related to side-channel leakage can be detected at some point in this process and cannot arise again on a lower level of abstraction then we do not have to deal with it later in the process (i.e., we do not have to try to detect it). It means that developers and evaluators can save time by reducing the number of tests that they run on the system. However, it is very hard to give such guarantees e.g., proofs on masking schemes were later broken by by advances in the research (recall Section 3.4.1).

It is possible to refine our classification by adding more stages and by making it more complex e.g., by including a stage where the type (architecture) of a microcontroller is chosen, as well as a stage of the choice of a specific model of a microcontroller. Moreover, if the final binary can be executed by several models of a microcontroller we can add another stage — the specification of microcontroller, it can be done e.g., through its architecture, or even using a gate-level netlist if this information is available. However, for the sake of simplicity we will suppose that the exact model of microcontroller is known in advance at the moment when the developer starts working on a project. Also, note that depending on the choice of the programming language the stages S1 and 2 can be skipped e.g., because the software is developed in assembly language. Another interesting point that we want to highlight is that a designer of a cryptographic algorithm who creates the A0 representation can often implement the algorithm or a part of it to test it and tune its parameters e.g., for the choice of an S-box (see examples in Section 6.3).

The algorithm level (A0) allows to build proofs e.g., for masking schemes under a chosen leakage model, thus it allows to detect if a scheme works on a theoretical level (we can test if masking is biased due to bad design). The high-level source code stage S1 allows to specify which parts of the state will produce the leakage together (bit-sliced vs. classical implementation) and allows us to note the global order in which values will leak information. At this stage we can also detect if a countermeasure was not applied correctly e.g., some variables are not masked or some parts of the state are not shuffled due to an implementation error. The intermediate representation level of abstraction S2 allows us to detect whether the processing order of variables is incorrect which may cause a problem in masking schemes (several masks and random values have to be applied in a specific order to make sure that the intermediate value stays masked). The assembly stage S3 already allows to detect distance leakage when a value in a register is overwritten by another one, this action can also unmask a masked value (see more details and examples in Section 7.2.1). In other words, at this stage in the development process it is possible to detect insecure register allocation. The final stage S4 where we get the compiled binary file allows the evaluator to test the implementation for timing attacks that can be caused by the misalignment of the code (the S-box table stored in two sections causing time dif-

ferences depending on the location that was accessed). Moreover, if the model of a microcontroller is specified the evaluator can test the code for effects caused by the pipelining and caching mechanisms. Note, that here we presented some examples of side-channel attacks that can be discovered at each stage and this list is not an exhaustive one.

For the sake of completeness we would also like to suggest a hardware classifications. Moreover, some of the existing simulators can deal with both hardware and software implementation. However, since we mainly focus on software implementations, we will not give details over the type of side-channel related issues that could be detected at each stage. So, here is the list of levels of abstractions for a hardware perspective:

A0 Algorithm — same as in the software counterpart,

H1 Register Transfer Level (RTL) — general design and connections between different blocks are specified including the level of parallelism and pipelining,

H2 Gate-level netlist — the full representation in terms of logic gates is specified,

H3 Layout — size and geometry (physical location and proximity) are specified.

The rationale is that developers of cryptosystems should use a toolchain of simulators (rather than only one), each of them being specific to the level of abstraction of the current stage of development of the product. Such approach should allow to detect issues early on and thus reduce the overall time spent on the evaluation and development of secure cryptosystems.

5.3 Survey of existing simulators

There exist many simulators that can be used for code debugging e.g., Valgrind[11] and SimulAVR[12]. Unfortunately, these simulators do not include information about the power consumption of devices. Thus they cannot be useful for studding security issues related to power analysis of implementations of cryptographic algorithms. At the same time, there are a lot of simulators that were built to estimate power consumption of devices. These simulators can help manufacturers during the process of chip development as well as during software development. A big portion of such simulators are built with the intention of helping engineers for e.g., choosing the best power supply and analysing average or peak power consumption of a device. Among such simulators we can find ChipPower [TAV+04] and Nano-Sim [SPW07]. However, such tools are not suitable for purposes of early analysis from the perspective of side-channel attacks based on power analysis. The main reason is that these simulators do

[11] `http://valgrind.org/`

[12] `http://www.nongnu.org/simulavr/`

not output information useful for side-channel analysis. In order to do side-channel analysis using simulators we need simulators that can output information related to the power consumption of a device over a period of time, this information should take into account the data that is being processed by the device. A simulator such as SPICE[13] can be used for side-channel analysis, however this simulator requires precise knowledge about the internal structure of the analysed device, often researchers do not have this information since manufacturers of microchips tend to keep it secret. Moreover, modern microcontrollers are composed of tenth of thousands of logic gates which results in a rather huge circuit, it means that SPICE will not be able to compute the full scheme in a reasonable amount of time. Here we present simulators that were created for the purpose of studying side-channel attacks based on power analysis.

Pinpas (Program Inferred Power Analysis Simulator) [dHVdV+03] tool that was created in 2003 is the first simulator (for side-channel analysis) that was presented to the scientific community. The Pinpas tool is written in `Java`, it could be used to test smart cards during the design and implementation of cryptosystems. Pinpas is composed of two parts: a simulator and analyser. Developers can choose a type of microcontroller (smart card) that they want to simulate with the written code [HBdH04]. Pinpas provides a virtual environment that can simulate the execution of a program under analysis. Nowadays this tool is not available to the general public since Pinpas was adapted by industry in order to build analysis tools based on it [dHdV04].

SCARD (Side Channel Analysis Resistant Design flow) project resulted in a proposal of another simulator for side-channel analysis in 2006 [AMM+06], we will refer to it as SCARD. SCARD is a tool which aims to simulate side channel effects on several abstraction levels. This simulator was created using `SystemC` and tested on ARM7 using cryptographic algorithms implemented both in software and in hardware (using a crypto-peripheral). Thus, this simulator can work during the development of hardware using Hardware Description Language (HDL). SCARD uses SPICE models with information gained from real measurements or obtained through accurate modelling of physical structures. SCARD was used to create a robust implementation based on 8051 IP open Core (by Oregano Systems). This idea can greatly improve the process of development of integrated circuits. However, it is more difficult to use with existing microcontrollers and processors since manufacturers do not provide enough information (details of chip design) to create accurate models of their products.

A simulator based on Cadence NCSim was proposed in 2007 by Kirschbaum *et al.* [KP07]. This simulator uses gate-level netlist and cell descriptions in `Verilog` (as well as information on propagation delays) in order to create simulated traces. The simulator uses transition counting model (number of single-bit modifications) in order to create simulated traces. It was used to analyse the chip that was designed during SCARD project (based on 8051). This simulator is shown to create accurate

[13]`http://bwrcs.eecs.berkeley.edu/Classes/IcBook/SPICE/`

leakage traces in the original paper, however it requires a lot of information about the details of hardware implementation of the device (which are not always available to all evaluators in case of real devices).

Thuillet *et al.* [TAL09] presented another side-channel analysis simulator in 2009, it uses some ideas from PINPAS and is designed for the analysis of smart cards. More details on this simulator are presented in the PhD thesis by P. Andouard [And09, section 4.5], where it is referred as OSCAR. OSCAR is capable of focusing the simulation of side-channel measurements on some parts of the circuit. This simulator is written in `OCaml` programming language, one of the reasons that guided this choice is the idea of developing a version that can be used in `Coq`[14] proof assistant in order to get formal proofs (regarding side-channel analysis) on the implementations. It can simulate several 8-bit Atmel microcontrollers and it requires compiled binary files as an input in order to do so. This simulator focuses on fault injections in addition to power analysis. The main drawback of OSCAR is that the binary file has to be recompiled in order to get simulated traces for different plaintexts (or different keys). This fact slows down the whole process of data acquisition (of simulated traces), since one generally needs a set of traces with different plaintexts and possibly with different keys for the analysis and the recompilation will inevitably consume a lot of time compared to the actual simulation. In this scenario each binary is executed only once to create a simulated trace, which is not very efficient. Unfortunately the source code of this tool is not available because it was developed in collaboration with private enterprises.

A way of creating a simulator based on profiling was presented by Debande *et al.* in 2012 [DBBL12]. Their simulator can be used to generate simulated traces of any specific, previously profiled, device. The profiling uses stochastic models (linear regression, recall Section 3.2.4) and the successive values of the internal state (registers) of the device. Thus, it can be used with microcontrollers as well as FPGAs and ASICs. This type of simulators allows to construct simulated traces that are close to the real measurements. This approach allows to test implementations during the development phase without acquiring real measurements, which significantly speeds up the whole process. The main advantages of this simulator are precision and the fact that the user does not need to have a lot of information about the simulated device i.e., one does not need to know the specifics of the internal structure of the device (that are usually unknown for most of commercial products). The biggest inconvenience related to the use of this simulator (and other profiled simulators) is the fact that one has to do the profiling step for every new device (such as a new model of a microcontroller).

In 2013, Gagnerot presented another simulator in his PhD thesis [Gag13, chapter 10]. This simulator uses compiled binary files in order to create simulated traces, it can be used to simulate power traces as well as to simulate fault injections. The

[14] `https://coq.inria.fr/`

simulations include information from several peripherals (UART, as well as crypto co-processors). The simulator uses HW and HD models and works with a specific 16-bit RISC microprocessor[15]. However, later some additional architectures were added to the design. The simulator was created during a collaboration with a private enterprise, thus its code is unavailable.

Another simulator was proposed in 2015 by Barthe *et al.* [BBD+15]. This tool uses formal methods in order to find weaknesses in masking schemes. Contrary to some other tools (that can just output a simulated trace) it outputs the results of the verification on the analysed scheme. The most attractive property of this simulator is the fact that it uses formal verification, which can be used as a proof that a scheme is secure under the chosen assumptions (such as the leakage model). This method also allows to test an attack against a flaw that was found by the tool. This tool requires to write the algorithm in a specific language called `EasyCrypt` [BDG+13]), which might be problematic due to errors that might occur while translating from one programming language to another. In addition, even if the code is translated from the original (e.g., `Assembly` or `C`) language, there is no guaranty that the new `EasyCrypt` version of the program will be equivalent to the original in all its aspects. To the best of our knowledge automated tools that can transform a program into `EasyCrypt` do not exist nowadays. Moreover, `EasyCrypt` is a relatively high level of abstraction language and, a compiler can (and often does) make changes to the code by rearranging independent instructions during the optimization phase of compiling. Such changes can influence the robustness of the device against side-channel analysis, thus there is no guarantee that such analysis remains valid for the final compiled code. Nevertheless, it allows to validate the general idea of the masking scheme.

Reparaz [Rep16a] presented a simulator in 2016 (first appeared on e-print archive in 2015). It works using the high level description of an implementation (e.g., `C++` code) in order to detect flaws in masking schemes. It can detect flaws in first and higher order masking schemes. The main idea is based on leakage simulation (using the analysis of intermediate values) followed by leakage detection tests (t-tests). This simulator can pinpoint the variables that are causing the leakage. The presented tool could detect design flaws in masking schemes, however it cannot detect issues related to the implementation of these schemes which is stated by the author, since e.g., *"an unfortunate choice of register allocation may cause distance leakage between* [mask] *shares"*.

ELMO (Emulator of Power Leakages for Cortex-M0) [DMO16] simulator was presented in 2016 (paper updated in 2017 on e-print). ELMO is a profiled simulator that was tailored for ARM Cortex-M0. ELMO could be used to analyse implementations from the point of view of side-channel attacks. The analysis done during the creation of ELMO showed how to improve the profiling of a device for creation of such simulators. Among others, authors could speed up profiling by clustering different

[15]This is the only information available in the text due to the non-disclosure agreement.

instructions into 5 groups and do the actual profiling for only one instruction from each group. ELMO also uses profiles that depend on triplets of instructions i.e., information on previous and subsequent instructions are included into the profile which allows to improve the accuracy of simulated traces e.g., it allows to take into account effects produced by pipelining. The fact that ELMO is a profiled tool makes it very precise, however, its accuracy can be improved using more profiling (e.g., profiling all instructions). The authors also showed that their method can be used for simulations of Cortex-M4 processor (see the updated version from 2017). This simulator was later upgraded with leakage detection capabilities using t-tests [MO17], however this update does not change the underlying simulation models. The main difference between ELMO and the tool by Reparaz is that ELMO is built using profiling and it operates on the assembly code, while the tool by Reparaz is using a higher-level of abstraction language as an input and it uses predefined rather than extracted (profiled) models.

In 2016 Bos *et al.* [BHMT16] proposed a tool that can be successfully used in order to break white-box implementations using Differential Computational Analysis (DCA)[16]. Their idea is based on applying side-channel analysis on simulated traces that were obtained by running a simulator on the target white-box implementation. Use of a simulator that generates traces is one of the best options in such scenario since the code is available to the attacker. At the same time the attacker can choose to generate traces without noise and can also use any leakage model. The tools proposed in this paper are based on existing simulators Pin and Valgrind. All the tools are actually extensions (plugins) for these simulators. Thus, these tools allow to simulate x86, x86-64 as well as ARM devices. Moreover, these tools allow to trace and analyse the execution from different perspectives: intermediate values, values written on the stack as well as their address ranges. All the tools that were used during this study are opensource[17]. These tools can probably be used for analysis of cryptographic implementations other than white-box from the perspective of side-channel attacks, however such approach has not been investigated yet.

Allibert *et al.* presented a simulation framework DBI [AFG$^+$15] that can be used to analyse binary files from embedded devices (authors refer to Android platform on ARM processor as an example, but their tool-chain also supports other architectures such as x86 and SPARC). This tool is based on the simulator presented by Gagnerot [Gag13, chapter 10] (who is also one of the co-authors with Allibert). The ideas presented in this paper are also close to the idea of DCA [BHMT16]. However, the approach that Allibert *et al.* suggest is based on the analysis of registers (while Bos *et al.* mostly focus on address ranges), their tool also uses different leakage models. Moreover, this tool can also simulate fault injections during the execution. Unfortunately most of details about this study are not revealed publicly (the source code

[16] First appearance on the e-print archive of the International Association for Cryptographic Research (IACR) in 2015.

[17] `https://github.com/SideChannelMarvels`

is not available, the names of cryptographic libraries that implement AES are not mentioned).

A tool called Inspector-SCA that was developed by Riscure[18] can also be used in order to run simulations and use them for side-channel analysis. The simulator feature was integrated in the Riscure's software suite between 2007 and 2008 (according to an employee) and the last update (at the time of writing) appeared in 2016. This commercially available tool allows to simulate traces and immediately analyse them in the same environment. The user has to supply the source code and to instrument it by specifying interesting intermediate variables that are used during the execution in order to create simulated traces. This tool has some commonly used leakage models such as HW. However, it is possible to create new leakage models, this option also allows to create several leakage points with different leakage models related to the same intermediate value. The Inspector software is not opensource, however, it can be extended via user-defined modules and plugins.

Virtualyzr is another commercially available simulation tool by Secure-IC[19]. Virtualyzr can be used for passive side-channel analysis (such as based on power consumption) as well as for active attacks (fault injections), in this regard it is similar to the simulator described by Gagnerot. Virtualyzr can produce simulated traces based on the source code describing the RTL, while performing analysis on several levels of abstraction: the netlist (created during the synthesis) and the floorplanned circuit (created during the place-and-route procedure). Thus it can be used to detect the issues caused by optimization performed by synthesis tools such as removal of checking procedures used to counter fault injections as well as information leakages created by glitches [DGN+17]. Thus, Virtualyzr mostly focuses on the early stages during the development of hardware implementations. However, its simulation can be coupled with the software counterpart by using a comportmental model of the simulated processor using `Verilog`, `VHDL` and `SystemC` source code as an input.

Table 5.2 provides a summary of existing simulators that were built for side-channel analysis. We can see, that these simulators vary greatly in terms of the types of inputs that they require (source code, assembly, binary files) as well as in the purposes that were intended while these tools were built. These simulators present such a diversity, that it is very difficult to compare them. Moreover, the main issue that we can notice lies in the fact that the vast majority of these tools are not available for free or even commercially. This fact challenges the possibility of reproducing the results that were obtained using most of those tools.

[18]https://www.riscure.com/security-tools/inspector-sca

[19]http://www.secure-ic.com/solutions/virtualyzr/

Table 5.2 – Simulators developed for side-channel analysis.

Name	Year	Input	Type	Main purpose	Leakage models	Stage	Availability
PINPAS	2003	Assembly code, microcontroller type	G	Analysis during development of software	HW, HD-like, device profiling	S2	✗
SCARD	2006	HDL code	G	Leakage detection on early stages of development of IC	HW, HD-like, SPICE on several abstraction layers	H1	✗
Cadence-NCSim	2007	Gate-level netlist, cell description, propagation delays	G	Evaluation of DPA-resistance	Transition counting	H2	✗
OSCAR	2009	Compiled binary	G	Fault injection and power analysis	HW, HD	S3	✗
Debande *et al.*	2012	Successive values of registers	G	Fast evaluation during development	Real traces (profiling)	S2, S3	✗
Gagnerot	2013	Compiled binary	G	Fault injection and power analysis	HW, HD, several points with different models	S3	✗

Continued on next page

Table 5.2 – *Continued from previous page*

Name	Year	Input	Type	Main purpose	Leakage models	Stage	Availability
Barthe *et al.*	2015	`EasyCrypt` code	V	Formal proofs on masking schemes	t-threshold probing model	A0, S1	✗
Bos *et al.*	2015-16	Compiled binary	G	Analysis of white-box crypto	Values and addresses, single bits	S3	✓
DBI	2015	Compiled binary	G	Analysis of white-box crypto and fault injections	HW, HD	S3	✗
ELMO	2016-17	Assembly code	G	Evaluation of DPA-resistance	Real traces (profiling)	S2	✗*
Reparaz	2015-16	High level source code	G	Detection of flaws in masking schemes	HW, LSB, zero-value, identity	S1, H1	✗*
Inspector-SCA	2008-16	Source code	G	Evaluation of DPA-resistance	User-defined	S1	✓€
Virtualyzr	2017	HDL code	G	Fault injection and power analysis	Value & distance leakage, local & global modeling	H1, 2, 3	✓€

G — generator, V — verifier (checker).
* Authors are planing on releasing their code, but at the time of writing it is still not in the public domain.
€ Available commercially.

5.3.1 Other works related to simulations

Many analysis that were done in the field of side-channel attacks use simulations to study how attacks and countermeasures behave depending on noise [SVO+10] or other parameters [LPB+15, SKS09]. There exist studies that show how simulations of different levels of abstraction (in hardware) influence the results of the attacks [TV05] i.e., how taking into account more information (for a simulation) changes the amount of traces required to break the system, which ultimately tells us about the accuracy of simulated traces compared to real power traces. Some researchers use simulations along with real traces to compare different distinguishers [SBG+12].

Due to the absence of publicly available simulators which can be explained by results from Table 5.2, most authors build their simulations from scratch every time. Some papers do not give detailed descriptions of how *exactly* their simulations were created [SVO+10] others provide more details on this matter [LBM15a] (but still do not provide their source code). To the best of our knowledge every team uses some flavour of "home brewed" piece of code that generates a set of leakage points (that represent power traces) in order to analyse them; generally researchers would only simulate points related to the operation that is analysed (e.g., an S-box), however sometimes non-informative points are also added to the set due to the nature of the study [LPB+15]. Another way of creating simulated traces that is used by some researchers consists in taking real power traces and modifying them e.g., by adding random numbers to each point to simulate a noisy setup or by shifting them and removing some points to simulate clock jitter and misalignment [WO15].

We mostly focus our attention on power analysis but there exist simulators that were built for analysis of other types of physical attacks. For example, `ctgrind`[20] tool for timing analysis can check if a function executes in constant time (using Valgrind). The work by Rothbart *et al.* [RNS+05] focus on fault injections and show how a security analysis on high-level of abstraction can detect issues related to these attacks using the tool [NRS+04] that they created for energy estimation based on hierarchical bus models in smart cards.

[20] Available on `https://github.com/agl/ctgrind`

5.4 Summary

There are two types of simulators that can be useful for side-channel analysis: data generators and checkers. The first type allows to generate data that is equivalent to power traces while the second type can verify properties related to the robustness of an implementation against side-channel attacks. We saw that there are 4 main motivations for use of simulators in side-channel analysis: (1) they are faster than real experiments, (2) they provide full control over the environment (i.e., allow us to choose and set all parameters related to the experiment), (3) simulators also allow to publish and communicate the data that was used for the simulation easier (compared to real power traces), and finally (4) simulators can be used during the development of implementations as opposed to real experiments (that require the finished product). Thus, simulators provide a set of features that are simply not available in datasets from real experiments.

In many research papers scientists use simulations to go through a lot of experiments (e.g., vary a parameter through a range of values) which shows us that the demand for available simulators exists in this domain. Moreover, several works showed that simulators can be useful for detection of security issues related to side-channel information leakages. However, most of the simulators that were presented to the scientific community are not actually available (even commercially) with only a couple of exceptions. Moreover, we can find a huge variety of types of simulators even among the unavailable ones, they come in many flavours in term of the models they use, way they are implemented, their capabilities and the type of inputs that they require. Ultimately it makes them very different and thus almost impossible to compare and therefore, it is hard to judge of the quality of these tools (basically we need compare an object to another one in order to judge of its quality). The rationale is that there is a need for *available* simulation tools for side-channel analysis and many different types of simulators can be used for such analysis, but only a few of them are actually available which ultimately makes this domain relatively poorly-studied. Therefore, we designate that one of the main goals of this work is to provide software simulation tools and show how they can be used in case studies.

Chapter 6

Silk: high level of abstraction simulations

The simulator presented in this section is based on the paper *"Silk : high level of abstraction leakage simulator for side channel analysis"* presented in New Orleans (United States of America) at the 4th Program Protection and Reverse Engineering Workshop (PPREW-4) in 2014 [Ves14]. The analysis of shuffling schemes is based on the paper *"Variety of scalable shuffling countermeasures against side channel attacks"* published in the Journal of Cyber Security and Mobility [VML17]. All studies related to S-boxes are based on two papers, first one *"Comparing S-boxes of Ciphers from the Perspective of Side-Channel Attacks"* presented in Yilan (Taiwan) at IEEE Asian Hardware Oriented Security and Trust Symposium (AsianHOST) in 2016 [LMV16] and the second one *"On the Construction of Side-Channel Attack Resilient S-boxes"* presented in Paris (France) at the 8th International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE) in 2017 [LVPM17].

Scientists use simple simulations to run tests of their attacks and countermeasures. Having an automated tool that can generate simulated power traces instead of building a "one-shot" piece of software for one test every single time can greatly improve the efficiency and the reliability of such tests. Moreover, if everyone can have access to the same tool the comparison abilities between different analysis methods as well as their repeatability will improve.

The main goals of the studies described in this chapter are (1) building a high level of abstraction simulator that can generate simulated traces from the source code with several parameters that describe the leakage and (2) showing how such simulator can be used for the analysis of side-channel attacks as well as countermeasures against them.

We start by explaining how our simulator works. Next, we show how it can be used to compare algorithms or their parts on the example of S-boxes, we then use our comparison technique to build S-boxes that are optimised with respect to the desired properties. Finally, we show how our tool can be used to compare countermeasures on the example of shuffling schemes.

6.1 Description of the tool

SILK stands for Simple Leakage Simulator. The main idea behind this simulator is having a very simple tool that can produce simulated traces based on a piece of the source code and several parameters that describe the leakage function. Here below we describe general principles and ideas used in our power trace simulator. Our simulator could be used in order to simulate traces of such devices as microprocessors or microcontrollers (e.g., embedded in a smart card or another portable device). This simulator is written in `C++` and its code is available in our `git` repository[1].

SILK creates simulated traces, thus it is a generator tool according to our terminology. This tool simulates the data dependant part P_{val} and can also add noise ε to the result of the simulation, but it does not simulate the operation dependant part P_{op} (from the Equation 3.7, Section 3.2.3). Thus, SILK can be used for DPA, but not for SPA. Nevertheless, information on the operation dependant part can be included in the simulator through some additional engineering efforts.

The main idea behind the implementation of this simulator is the overloading of operators (`=`, `+=`, `/=`, `&=`, etc.). Each operator returns the value that it would return normally (without the overloading) and writes a leakage value to the trace. In other words, the simulator creates traces on the fly while executing the code. Points of the simulated trace are created based on values that are manipulated each time when a result of a computation is assigned to a variable.

In order to use SILK one needs to do following modifications to their code: first of all, they have to import our library and change basic types such as `int`, `char` and `unsigned` to types defined in our simulator (e.g., `U8`, `U16` or `S8`, `S16`)[2]. Before using SILK one should also setup its parameters (see example in Listing 6.1, more details are available in Section 6.1.1).

```
1  //setup SILK parameters
2  U8::setLeakageFunction(hammingWeightOut);
3  Tracer::setLeakagePointsNbr(10);
4  Tracer::setLeakageOverlap(2);
5  //create traces
6  for(i=0; i<50; ++i){
7    Tracer::clearTrace();
8    setRandom(plaintext);
9    encrypt(key, plaintext, ciphertext);
10   Tracer::traceToFile( traceFileName+to_string(i) );
11 }
```

Listing 6.1 – SILK example: generation of 50 simulated traces.

[1] `https://github.com/nikita-veshchikov/silk`

[2] we used the same naming conventions for types as Cryptosat [LJH14]. It uses SAT solvers for the analysis of cryptographic algorithms [SNC09]., so algorithms that were once modified could be used with both cryptographic tools.

Once the execution of a cryptographic algorithm is complete, Silk allows to fetch the resulting simulated trace. The simulated power trace might be retrieved as a data structure (`std::vector<double>`) for immediate use in the code, thus the same program can actually generate simulated traces and perform side-channel analysis on them. Another option is to save the simulated trace in a file. A user can also add noise to the simulated trace as many times as needed (i.e., generate many noisy traces using the same data) without reiterating the code execution, see Listing 6.2. Code examples are also included with Silk in its `git` repository.

```
setRandom(plaintext);
encrypt(key, plaintext, ciphertext);
// generating 10 noisy traces after a single execution
Tracer::setNoiseVariance(0.03);
for(i=0; i<10; ++i){
  Tracer::noisyTraceToFile( traceFileName+to_string(i) );
}
```

Listing 6.2 – Silk example: generation of simulated noisy traces.

6.1.1 Parameters

Our simulator has four main parameters. Two of these parameters are mandatory and must be defined in order to obtain a simulated trace. The remaining optional parameters might be set independently of one another. Parameters are explained here below, Figure 6.2 shows a simulation that uses the same data and same code with different combinations of parameters.

Leakage function

The first mandatory parameter is the leakage function $L^*(v_{old}, v_{new})$, its parameters v_{old} and v_{new} are the old and the new values of a variable (of the program under simulation). In a simulated environment we know the actual leakage function that we want to simulate, thus it makes more sense to talk about a leakage function rather than a leakage model (the difference between the two terms is discussed in Section 3.2.3). We need to use both the new and the old values in a leakage function in order to be able to simulate leakages that correspond to the ODL and MTL models.

The leakage function L^* might be easily set to a function widely used in power analysis (e.g., Hamming weight, Hamming distance) or any other function that matches the following definition:

$$L^* : \mathbb{Z} \times \mathbb{Z} \to \mathbb{R} \tag{6.1}$$

We will use l_i to denote the value leaked by the ith instruction of an algorithm, $0 \leq i < N_{op}$, where N_{op} is the number of instructions (operations) that we want to simulate:

$$l_i = L^*(val_{old,i}, val_{new,i}) \tag{6.2}$$

where $val_{old,i}$ and $val_{new,i}$ are the old and the new values manipulated during the ith instruction.

Leakage points

The second mandatory parameter is the number N_P, which denotes the number of points that is produced by one operation, $N_P \in \mathbb{N}$. This number might be seen as a representation of the sampling rate of an oscilloscope.

With both mandatory parameters, values of a simulated trace T might be written as follows:

$$T[t] = l_i \quad \forall t \in [N_P \times i; N_P \times (i+1) - 1] \tag{6.3}$$

where t is an offset in a trace and i is the current instruction that is being simulated (as in the Equation 6.2). An example of the resulting simulated trace is shown in Figure 6.2a.

Overlapping

The first optional parameter is a number O. See example of a trace with overlapping in Figure 6.2b. The number $O \in \mathbb{N}$, denotes the overlap in leakages between two consecutive operations. It might be used to simulate the fact that at some point in time several operations leak simultaneously. For example, we might use overlapping to simulate the leakage of a microcontroller that uses a pipeline e.g., while the result of one operation is written into the memory the next one is executed. McCann *et al.* [DMO16] showed that including information from several consecutive instructions results in better profiles, their experiments were performed on a Cortex-M0 that has a 3-stage pipeline. This effect can also be observed on simpler devices such as a ATmega328P microcontroller that only has a 2-stage pipeline i.e., while one instruction is executed the next one is fetched which means that data from two instructions is *not* processed in parallel, see Figure 6.1. However, we can see that some points are related to the values that are manipulated independently by consecutive instructions (see more details in Chapter 7). Sometimes we may also observe that one point of a power trace is related to two consecutive values due to imperfections of the measuring equipment e.g., a probe of an oscilloscope is not fully discharged before the next measurement is recorded.

If the overlapping parameter is set, last O leakage points of one operation are added with first O leakage points of the next one, more formally:

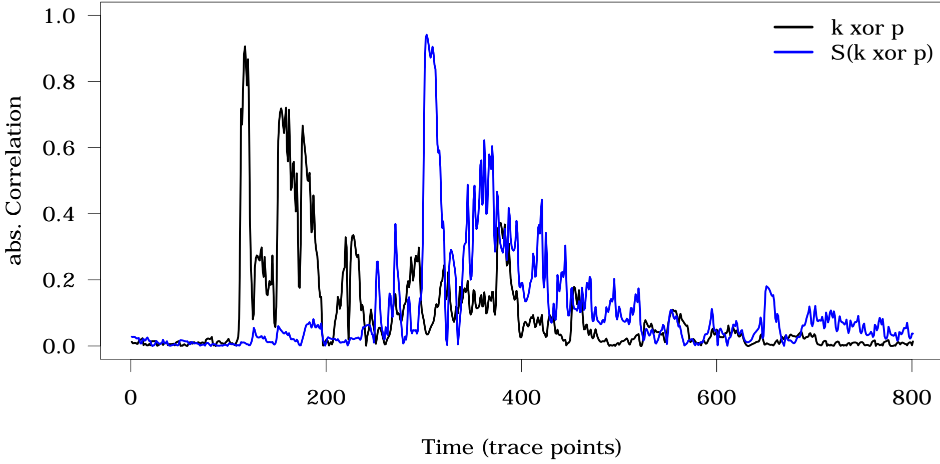


Figure 6.1 – Correlation for two consecutive intermediate states of a byte. States correspond to the input and output of an S-box.

$$T[t] = \begin{cases} l_0 & t \leq N_P - O \\ l_{N_{op}-1} & t \geq (N_P - O)n \\ l_i & (N_P - O)i + O \leq t \leq (N_P - O)(i+1) - 1 \\ l_i + l_{i+1} & \text{otherwise} \end{cases} \quad (6.4)$$

where n is the number of simulated instructions and other notations are the same as in previous sections on parameters.

Note, that in the overlapping section of a simulated trace, where leakages from two consecutive values are combined, the combination is a simple addition. This addition can be replaced by a parameter (combination function) in order to make this simulator more flexible (but it would require to set more parameters at its initialisation). However, using a simple addition of leakages already provides a lot of agility and expressiveness.

Leakage distribution function

Second optional parameter is a couple (D, I), where D is a leakage distribution function:

$$D : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

and I is a number that is used to define the interval $[0, I]$ (on which D is used in the simulation). An example that uses this parameter is shown in Figure 6.2c. This optional parameter can be used to represent the fact that the modelled device does not

leak the same amount of information (or not the same information) during the execution of one operation. For example, a device might leak some information during the computation and a different amount of information when the result of a computation is written back into the memory. Another example when this parameter would be useful consists in the following idea: during an addition, a device might leak information on the Least Significant Bit (LSB) at the beginning and information on the Most Significant Bit (MSB) at the end of the operation.

Parameters of the function D are the leakage l and a moment in time t. If the couple (D, I) is defined, the simulated trace is computed in the following way:

$$T[t] = D\Big(l_i, \frac{((t \mod N_P) + 1) \times I}{N_P}\Big) \tag{6.5}$$

in other words, the function D spreads the leakage l over N_P points equally spaced in the interval $[0, I]$ while potentially modifying it at every single point of the interval that corresponds to the simulated instruction.

We can also define both optional parameters, in this case we will apply the function D on the leakage l_i and then use overlapping, see example in Figure 6.2d.

6.1.2 Discussion

The simulator SILK is the first opensource simulator for side-channel analysis that was presented to the scientific community. It is not attached to a specific hardware, which makes it very generic. These facts together with all the parameters that SILK can use make it very flexible and malleable. However, since it uses a very high-level of abstraction models it cannot be used to detect hardware specific issues or to find issues related to low-level instructions (mostly because a compiler can rearrange operations for optimisation purposes). Nevertheless, SILK allows to generate traces based on a software implementation of an algorithm in automated manner, without writing a new "disposable" piece of code for each new simulation. Moreover, the resulting traces contain points related to all main intermediate states (except of the temporary values used during computations in registers) that are present in the algorithm. The main advantage of SILK is its flexibility (through use of parameters). Since SILK cannot be used to detect specific hardware issues, it should only be used for preliminary analysis and comparisons between different side-channel attacks and countermeasures against them.

Figure 6.3 presents a diagram that shows the general scheme of SILK with its parameters and a typical workflow that uses this simulator.

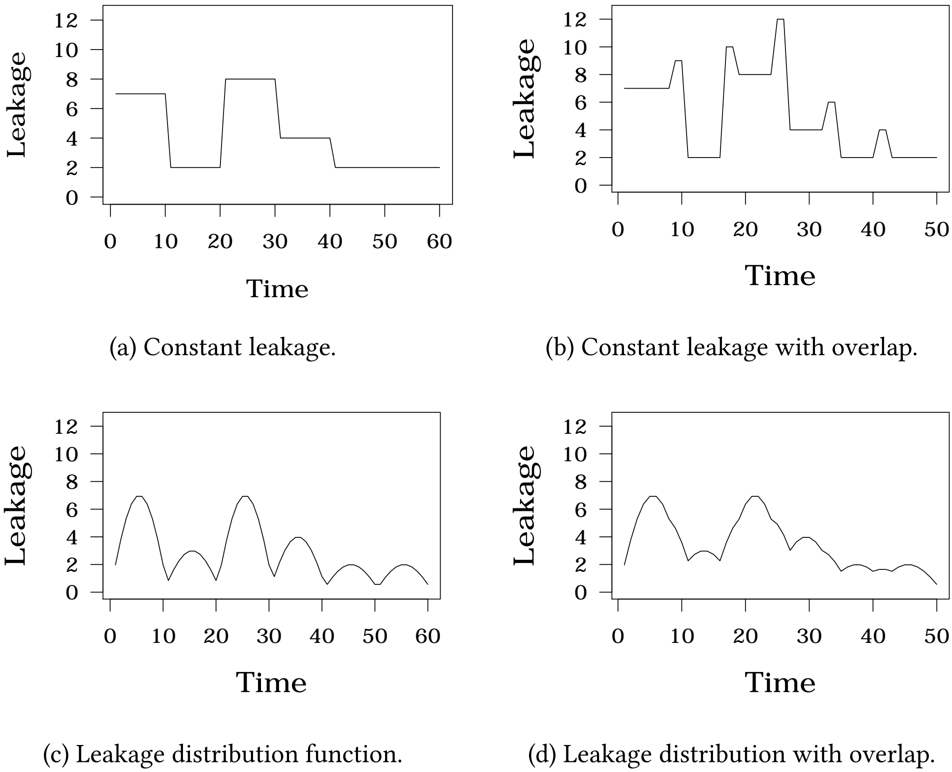


Figure 6.2 – Examples of simulated traces with and without optional parameters. Values used: $L(val_{old}, val_{new}) = HW(val_{new})$, $N_P = 10$, $O = 2$, $I = \pi$, $D(l,t) = l \times \sin(t)$. The code is availabe in the appendix A.

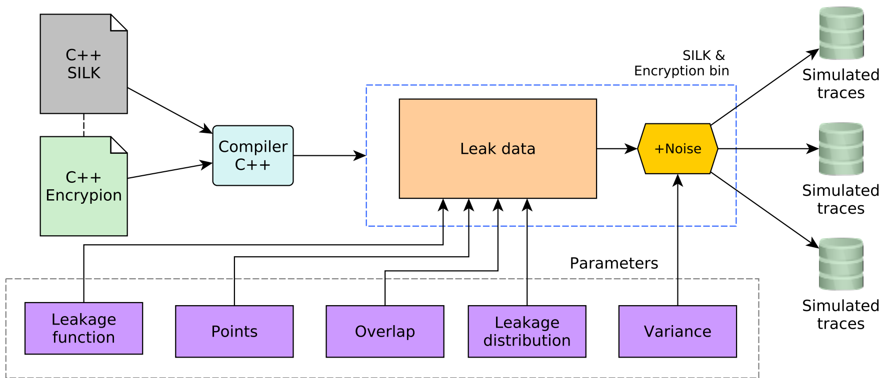


Figure 6.3 – Scheme of the workflow using Silk simulator.

6.2 Evaluation of S-boxes

Simulators can help in analysis of cryptographic algorithms or in the analysis of some parts of block ciphers. Since one of the most targeted part of a block cipher is an S-box, we think that it is a suitable candidate for the demonstration of how simulators can be useful in such analysis. In 2013, the open cryptographic "Competition for Authenticated Encryption: Security, Applicability, and Robustness" (CAESAR) was launched in order to find a suitable portfolio of primitives for authenticated encryption with associated data[3]. Our first goal is to evaluate S-boxes used by CAESAR candidates as well as several additional well-known S-boxes. Our second goal is to compare the approach based on simulations and results given by theoretical metrics (TO and CC). More precisely, we focus on side-channel analysis of the following 23 S-boxes:

- 4×4 S-boxes of Joltik, Prøst, Minalpher, PRESENT, $\text{Evolved}_{\text{CC}}$ and $\text{Evolved}_{\text{TO}}$;
- 5×5 S-boxes of ASCON, ICEPOLE, KECCAK (KETJE, KEYAK), PRIMATE and SC2000[4];
- 6×4 S-boxes of DES (labelled DES_i, $i \in \mathbb{N}$ and $1 \leq i \leq 8$);
- 8×8 S-boxes of SCREAM[5], STRIBOB, AES and AES_{CC}.

Picek *et al.* obtained the S-box $\text{Evolved}_{\text{CC}}$ by using genetic algorithms optimising the CC [PPE+14], they also created AES_{CC} by applying an affine transformation on the S-box of AES and choosing the one that gave the best CC. Later, Picek *et al.* built the S-box $\text{Evolved}_{\text{TO}}$ using genetic algorithms optimising the (improved) TO [PMMB15].

In addition to the previously listed S-boxes, we also used a special implementation trick applied on 4×4 S-boxes to create 8×8 meta-S-boxes. More specifically, we take into account the fact that an engineer can apply two 4×4 S-boxes of the same primitive (which is equivalent to an 8×8 S-box) when the device applies the same 4×4 S-box on two 4-bit words of the same byte at once in order to substitute a byte. To be more precise, these are not new S-boxes, but rather a different way of implementing a 4×4 S-box. In a software implementation this idea allows to trade memory (by encoding a bigger S-box) in order to gain some speed. Since almost all modern integrated circuits use byte-oriented architecture, engineers who implement ciphers (and also attackers) take the architecture into account. For example, an attacker would often target one byte of the secret key at a time. Thus, we also add 3

[3] `https://competitions.cr.yp.to/caesar.html`

[4] The SC2000 also uses 6×6 and 4×4 S-boxes while we analysed the 5×5 S-box.

[5] SCREAM algorithm was modified during the competition, one of the modifications concerned its S-box, we are focusing on the 3rd (latest at the time of analysis) version of the algorithm.

meta-S-boxes in our analysis. We call them $\text{Prøst}_{\times 2}$, $\text{Minalpher}_{\times 2}$ and $\text{PRESENT}_{\times 2}$. For the sake of simplicity we will refer to these meta-S-boxes just as S-boxes.

We focus our analysis on microcontrollers, thus we defined a special version of the Transparency Order that fits our scenario better. The TO metric assumes that the leakage depends on $HW(\psi \oplus S(a))$ where HW is the Hamming weight and ψ denotes the initial content of the register before updating it with $S(a)$. Equation 3.24 iterates over all values of $\psi \in \mathbb{F}_2^n$ in order to dissociate the transparency order metric from a specific device. The value of ψ maximising Equation 3.24 represents the worst case context for the designer and best case for the attacker when implementing the S-box (independently of the considered device). However, in practice, the strategy of the adversary depends on the target device. As a result, in our experiment, we also calculate the TO with ψ equal to zero. It corresponds better to our context in which the (analysed) microcontroller leaks the Hamming weight of the manipulated value. In the following, we denote TO_{max} when we go through all values of ψ, and TO_0 when we fix ψ to 0.

6.2.1 Results based on theoretical metrics

Table 6.1 reports the theoretical metrics CC and TO for each of the S-boxes[6]. The first observation on the TO metric is that big S-boxes lead to high coefficient and, as a result, should lead to higher success probability of side-channel attacks. This result is expected since (1) larger S-boxes have higher nonlinearity, and (2) higher nonlinearity results in higher success rate as shown by Prouff [Pro05]. Higher nonlinearity does help to distinguish key hypothesis using linear distinguisher since a small input error will be amplified more by a highly nonlinear function compared to a linear one. Note, that Heuser *et al.* showed that the robustness of a function against side-channel attacks is not simply related to its nonlinearity, but to its resistance against differential cryptanalysis [HRG14]. High resistance against differential attacks makes an S-box weaker against side-channel analysis and vice versa. During a side-channel attack we start by guessing the key and applying the same exclusive-or operation to it with all the known plaintexts before passing them through the S-box, thus creating a hypothetical output that will be more likely different from the real output (with the correct key) because of the *difference* between the tested hypothesis and the correct key *and* the high resistance of an S-box against differential cryptanalysis. Interestingly, CC (as defined by its authors) metric does not show exactly the same result, leading to a first contradiction between the two metrics.

Another observation is related to the order provided by the two metrics when sorting the S-boxes of the same size from the most resistive S-boxes to the least resistive one. For example, in case of DES S-boxes, we have the following order according

[6]Numbers for CC differ from the numbers by Stoffelen [Sto15] since he assumed a fixed correct key while we computed CC for the entire range using the same algorithm as Picek *et al.* [PPE+14].

to (1) CC sorted according to Fei *et al.* [FLD12] (denoted CC_{Fei}), (2) CC sorted according to Picek *et al.* [PPE+14] and Stoffelen [Sto15] (denoted CC_{Picek}), (3) TO_0, and (4) TO_{max}:

- CC_{Fei}: DES_7, DES_2, DES_8, DES_4, DES_3, DES_5, DES_1, DES_6;
- CC_{Picek}: DES_6, DES_1, DES_5, DES_3, DES_4, DES_8, DES_2, DES_7;
- TO_0: DES_6, DES_1, DES_3, DES_8, DES_5, DES_2, DES_4, DES_7;
- TO_{max}: DES_3, DES_2, DES_4, DES_7, DES_6, DES_5, DES_1, DES_8.

We can notice that all metrics provide different ordering of S-boxes based on their resistance against side-channel attacks, meaning that the metrics are not equivalent. We can notice that results given by TO_{max} and by CC_{Fei}differ from the results from TO_0 and CC_{Picek}. If we compare TO_0 and CC_{Picek}closer we can notice that their results are similar when the difference between values of a metric is high. For example, the results are similar when we compare 4×4 meta-S-boxes with 8×8 S-boxes and all metrics also show that the two evolved 4×4 S-boxes are harder to attack than other 4×4 S-boxes.

Since all metrics disagree with each other on the order of S-boxes according to their resistance against side-channel attacks, and there should be a definitive order between them under a given leakage function (which we fixed to be the same in this case), we can conclude that most of these metrics do not reflect the reality of a real attack (even if assuming that at least one of them is correct in its prediction).

6.2.2 Experimental results on simulations

In order to have a fair comparison of all S-boxes, we implement them in the same way by using look-up tables[7]. Thus, the only thing that changes between different S-boxes of the same size is the order of values in the look-up table. As a result, each simulated trace contains points that relate to the following operation:

$$z = S(k \oplus p). \tag{6.6}$$

We calculate the success rate of CPA by repeating the attack $50\,000$ times with different simulated traces and different random plaintexts. Note that this huge number of repetitions is necessary to get smooth curves of the success rate.

We use SILK tool in order to generate simulated traces. In our experiments we use the Hamming weight as the leakage function and we set the simulator to produce 1 point per instruction. Thus we have 4 points per simulated trace: one related to p, one

[7]We implemented each of the three 8×8 meta-S-boxes (that were build from 4×4 S-boxes) with a single look-up table of 256 values.

Table 6.1 – Modified Transparency Order (TO) and Confusion Coefficient (CC) metrics applied on S-boxes.

Size	S-box	TO		CC	
		TO_{max}	TO_0	$\sigma^2[\bar{\kappa}]$	$\sigma[\bar{\kappa}]$
8×8	AES	6.916	6.869	0.111	0.334
	AES_{CC}	6.916	6.828	0.149	0.386
	SCREAM	6.854	6.792	0.122	0.349
	STRIBOB	6.877	6.815	0.098	0.313
	$Minalpher_{\times 2}$	4.329	3.827	1.710	1.308
	$PRESENT_{\times 2}$	4.643	3.765	1.710	1.308
	$Prøst_{\times 2}$	4.643	4.580	1.051	1.025
6×4	DES_1	3.097	2.853	0.247	0.497
	DES_2	2.960	2.960	0.136	0.369
	DES_3	2.919	2.867	0.209	0.457
	DES_4	2.984	2.984	0.172	0.414
	DES_5	3.018	2.938	0.234	0.484
	DES_6	3.004	2.665	0.363	0.602
	DES_7	2.986	2.986	0.123	0.350
	DES_8	3.115	2.927	0.164	0.405
5×5	ASCON	2.839	2.839	0.502	0.709
	ICEPOLE	3.548	3.548	0.190	0.436
	KECCAK	3.871	3.871	0.115	0.338
	PRIMATE	3.613	3.581	0.308	0.555
	SC2000	3.548	3.363	0.260	0.510
4×4	$Evolved_{CC}$	2.500	1.533	1.388	1.178
	$Evolved_{TO}$	1.900	1.700	1.262	1.124
	Joltik	2.567	2.567	0.158	0.397
	Minalpher	2.300	2.033	0.660	0.812
	PRESENT	2.467	2.000	0.660	0.812
	Prøst	2.467	2.433	0.309	0.555

related to k, one related to their combination $k \oplus p$ and one related to the application of the S-box on this combination (giving z). We vary the noise variance from 0.5 to 20, but for the sake of space we report here only the case when the noise variance equals to 3. All other scenarios result in the same type of outcome with faster (for lower noise variance) or slower (when we have more noise) rise of the success rate.

We can notice a strong discordance between theoretical metrics and simulations

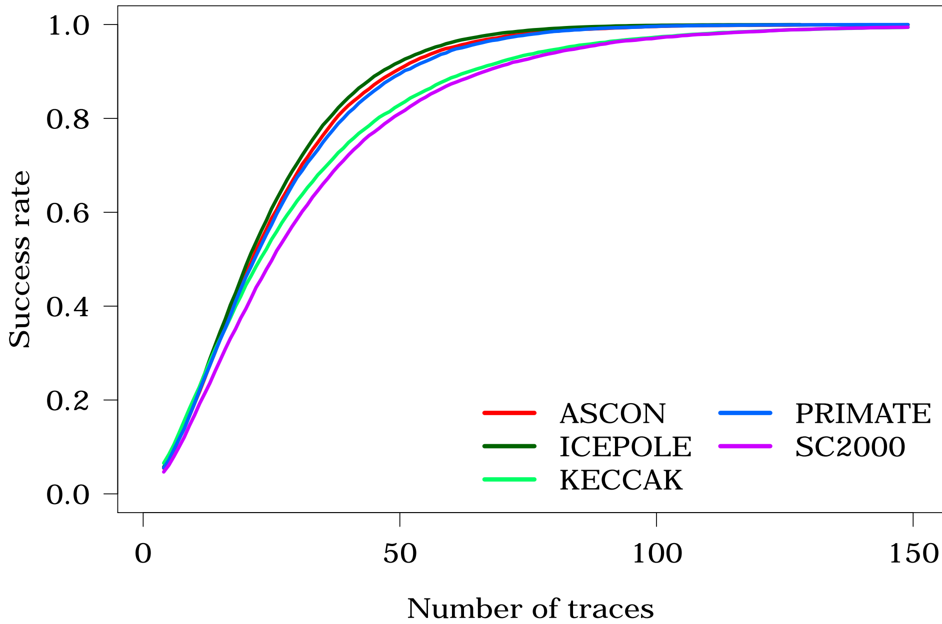


Figure 6.4 – Success rate of CPA on 5×5 S-boxes using simulations.

for other S-boxes. For example, here is the ranking for 5×5 S-boxes (see Figure 6.4):

- CC_{Picek}: Ascon, PRIMATE, SC2000, ICEPOLE, Keccak;
- TO_0: Ascon, SC2000, ICEPOLE, PRIMATE, Keccak;
- TO_{max}: Ascon, ICEPOLE & SC2000, PRIMATE, Keccak;
- SR of CPA: SC2000, Keccak, PRIMATE, Ascon, ICEPOLE.

All theoretical metrics highlight the S-box of Ascon as the best against side-channel attacks, and the S-box of Keccak as the worst from the same perspective. However, this order differs from the results reported by CPA in which SC2000 is the most resistant S-box while ICEPOLE provides the worst result.

Figure 6.5 shows the success rate of the CPA on simulated traces against 8×8 S-boxes (Figure B.1 in the Appendix B provides a closer look on the success rate). According to this results, all 8×8 meta-S-boxes are more difficult to attack than other 8×8 S-boxes as expected since these meta-S-boxes created from 4×4 S-boxes are more linear. This particular result agrees with the outcome of the theoretical metrics.

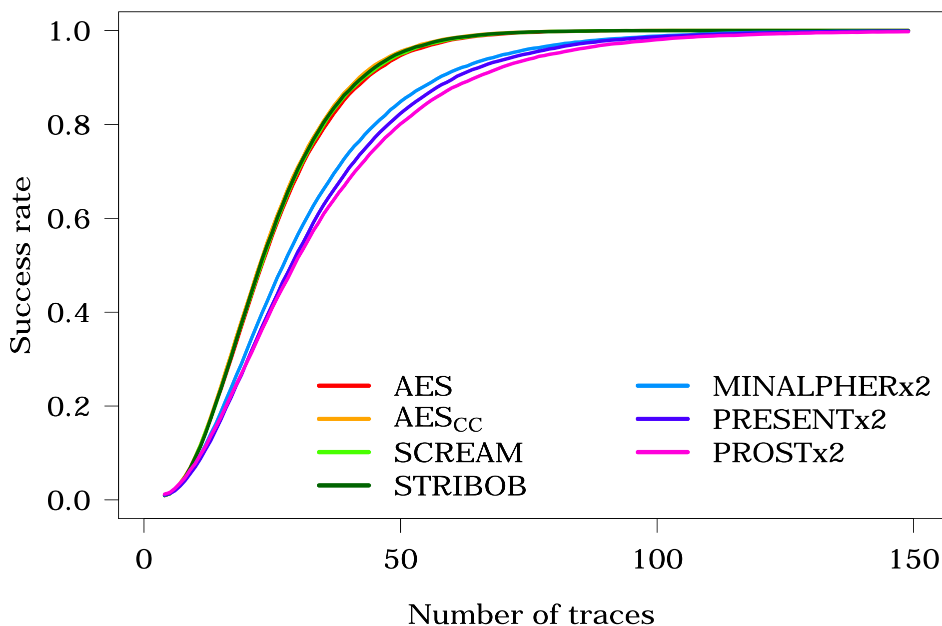


Figure 6.5 – Success rate of CPA on 8×8 S-boxes using simulations.

Figure 6.5 also reports that the difficulty of attacking an S-box differs from the outcome of the theoretical metrics. For example, the classification of 8×8 S-boxes from the most difficult to attack to the least difficult are the following:

- $\mathrm{CC}_{\mathsf{Picek}}$: $\mathrm{AES}_{\mathrm{CC}}$, SCREAM, AES, STRIBOB;
- TO_0: SCREAM, STRIBOB, $\mathrm{AES}_{\mathrm{CC}}$, AES;
- TO_{max}: AES & $\mathrm{AES}_{\mathrm{CC}}$, STRIBOB, SCREAM;
- SR of CPA: AES, SCREAM, STRIBOB, $\mathrm{AES}_{\mathrm{CC}}$.

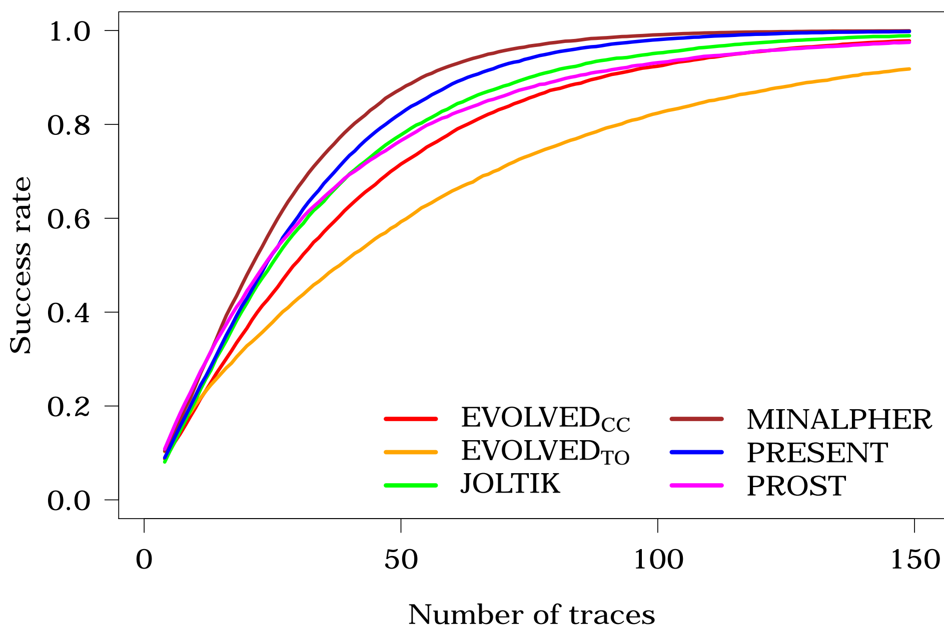
Note that the S-box $\mathrm{AES}_{\mathrm{CC}}$, which takes the CC metric into account in order to improve its resistance against side-channel attacks, leads to the worst resistance against CPA, which is surprising.

In addition to the discordances described here, Figure 6.6 shows that the curves of success rate of some 4×4 S-boxes *cross* each other. More precisely, the success rate curve related to Prøst crosses the success rate curves of Minalpher, PRESENT and Joltik. Furthermore, the curve of $\mathrm{Evolved}_{\mathrm{CC}}$ crosses the curve of $\mathrm{Evolved}_{\mathrm{TO}}$. In other words, for example, Joltik is harder to attack than Prøst with a small set of traces while the results are inverted with a larger number of traces. It is worth to note that

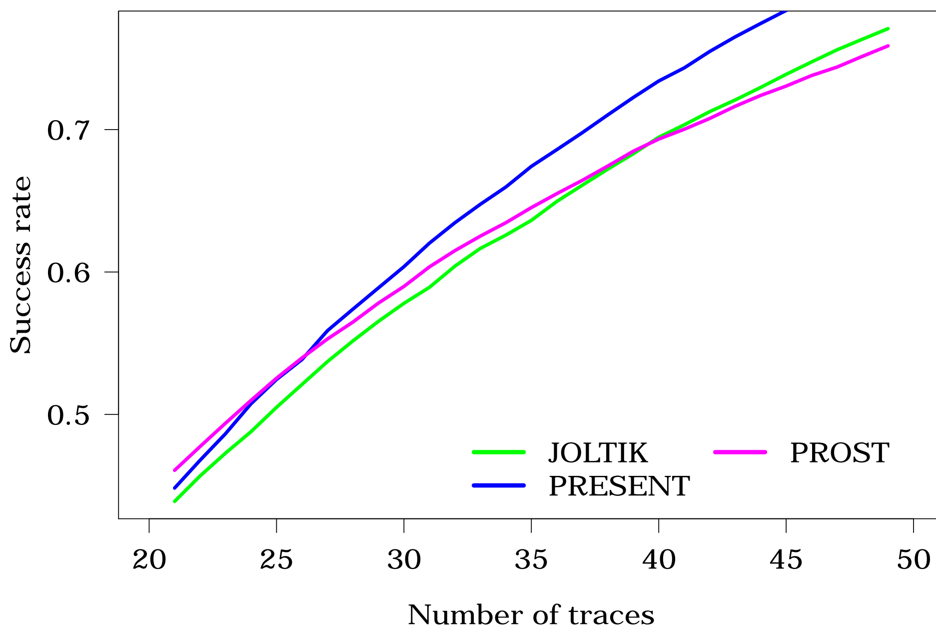
these results cannot be represented using any theoretical metric based on a single scalar value. In brief, all these results highlight that theoretical metrics (such as TO and CC) do not match actual attacks when the leakage model matches the leakage function (representing the worst case scenario for the developer) i.e., when the adversary knows how the device leaks information and does not make any assumption errors on the choice of a leakage model.

Results of our simulations (Figure 6.5) and results from theoretical metrics (Table 6.1 8×8 S-boxes) suggest the CC and TO metrics can provide useful information only when the difference between the S-boxes is large: notice the success rate (in case of simulations) and the values provided by metrics (in the table) for 8×8 S-boxes vs. 8×8 meta-S-boxes constructed from 4×4 S-boxes.

The success rates of the CPA on the 6×4 S-boxes of DES are available in the Appendix B.



(a) Success rate of all 4×4 S-boxes.



(b) Zoom on intersections.

Figure 6.6 – Success rate of CPA on 4×4 S-boxes using simulations.

6.2.3 Experimental results on a real device

In order to confirm our simulated results, we acquired real power traces on a popular 8-bit microcontroller ATmega328P. The acquisition was done using a digital oscilloscope that acquires 250×10^6 samples per second. The measurements were performed on a small 10 Ω resistor that was inserted between the ground pin of the microcontroller and the power supply of 5 V (the acquisition setup described in Section 3.2.1). We already know that ATmega328P leakage function is strongly related to the HW of the manipulated value (recall Figure 3.6). We implemented and attacked the following S-boxes:

- 8×8 S-boxes of AES, SCREAM and STRIBOB;
- 4×4 S-boxes of Minalpher, Present and Prøst.

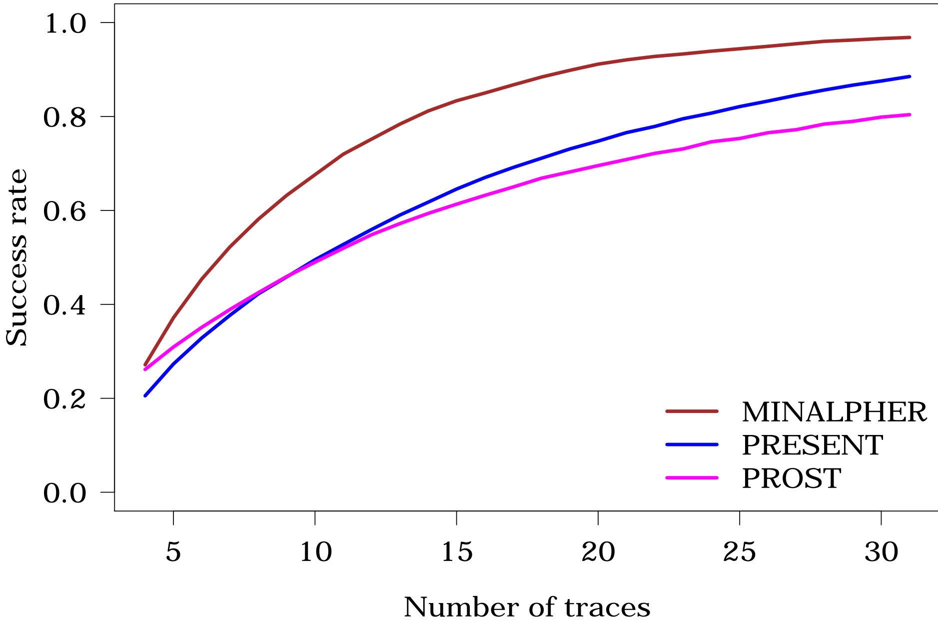
For these real experiments, we used the same code of the implemented S-boxes that were used by Silk (analysed in the previous section). Furthermore, we applied the same physical attack (CPA). We estimate the success rate by repeating the physical attack 10 000 times with different sets of simulated traces.

Figures 6.7a and 6.7b show the success rate of our attack on real implementations. We can note that experimental results that use simulations fit well the results that use real measurements (mostly because ATmega328P leakage function is very close to the HW). Figure 6.7b shows that S-boxes of AES, SCREAM and STRIBOB are indeed similar, as shown by results that use simulations (see Figure 6.5). Figures 6.7a and 6.6 report the same order between 4×4 S-boxes of Minalpher, Present and Prøst and also show that curve of the success rate of Prøst indeed crosses the curve related to Present. However, the experimental results on real traces differ from the outcome of the theoretical metrics (as already reported in the previous section using simulated leakage scenarios).

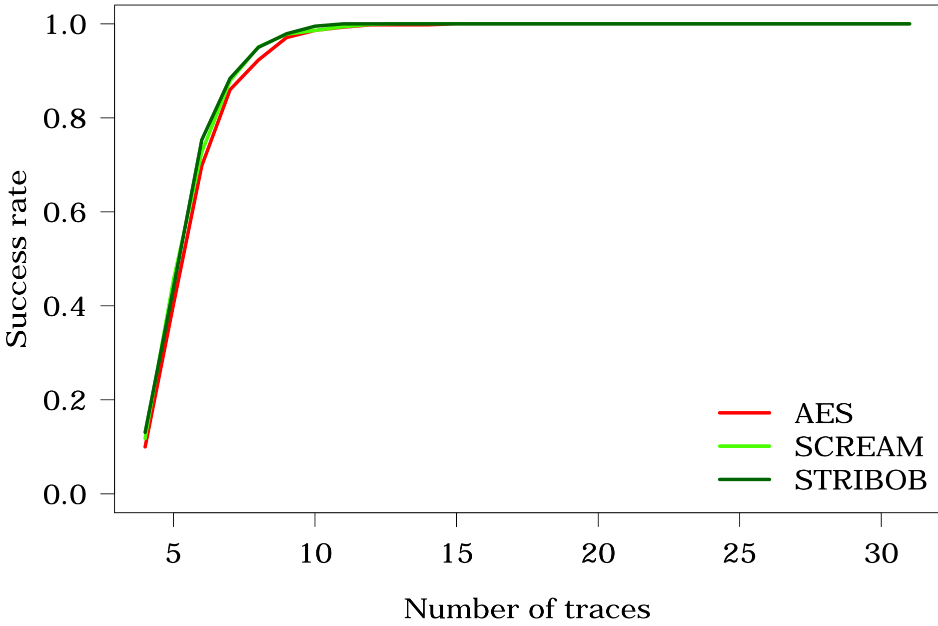
Note that due to time constraints that we had during this work, we were not able to build setups and acquire power traces on all the 26 S-boxes (which was very easy and fast in a simulated environment). However, our results suggest that even simple and high level of abstraction simulation gives a good approximation of what happens in real experiments. Overall, if we were to make all the 26 acquisitions on all the S-boxes we would need much more time, our analyses that used simulated traces were running in parallel (in a computing center), while our lab (and most of other labs that we know of) have only one or two oscilloscopes[8].

Simulated traces are generated on the fly, during the experiment, thus it takes a very small overhead to generate them (several minutes on our hardware). They are

[8] To be more precise we only have one oscilloscope of a specific model, which is also the case of other labs that we visited. Sometimes labs have several oscilloscopes of different qualities (e.g., bandwidth, acquisition speed, precision), but in order to be able to compare datasets one must use same type of equipment (same acquisition setup).



(a) 4×4



(b) 8×8

Figure 6.7 – The success rate of CPA on S-boxes implemented in on a microcontroller.

actually generated by the same executable file that later analyses them, the simulated traces are stored directly in RAM, which also increases the total execution speed (no need to read them from a relatively slower main memory i.e., disk). Acquiring 10 000 traces in our lab takes about 4.5 hours, we used more than 50 000 simulated traces per experiment and performed 156 experiments (26 S-boxes and 6 levels of noise variance). Thus performing real experiments would have taken us more than 3500 hours (almost *five months* of non-stop data acquisition in case if we were to work 24 hours a day). Thus, using a simulator helped us to significantly speed up our experiments. The fact that the simulation tool SILK that we used is already described in an earlier work also helps in at least two ways: (1) we can benefit from the existing code and (2) there is no longer need in describing it in our analysis, we can simply reference it and list the values of all parameters that we used.

6.3 Improvement of S-boxes

Our method based on simulations helps to compare S-boxes from the point of view of side-channel analysis. Thus, it provides us a tool of discriminating "better" and "worse" S-boxes which ultimately gives us a metric that can be taken into account during the design of a new S-box. We used this idea to create new S-boxes using evolutionary computing by tacking into account properties used in classical cryptanalysis (nonlinearity and differential uniformity, see Equations 2.5 and 2.7) and the success rate of side-channel attacks. Use of simulators enables us to perform this type of analysis since evolutionary computations require to evaluate a function multiple times, simulations being much faster than real experiment allow to perform all these evaluations in a reasonable amount of time.

6.3.1 Genetic algorithms and search strategy

Genetic algorithms is a family of strategies that can be used to search an object (in the search space) that has some desired properties. To be more precise, genetic algorithms provide a *heuristic* for searching a maximum (or a minimum) of a function, they are search strategies that can be used to optimise a chosen criteria. A heuristic is an approach of problem solving that is not guaranteed to be optimal (but is often good enough in practice). Thus, genetic algorithms do not provide a guarantee on finding the global maximum (or minimum). However if a problem does not have an optimal theoretical solution that can be computed using a deterministic algorithm (e.g., an algorithm that can compute the global maximum of a function does not exist) a genetic algorithm is a good choice. It is the case for problems that have high dimensionality (a number of parameters) in other words, genetic algorithms are well fitted for multi-criteria optimisation. Genetic algorithms were already used in cryptography [Pic16] and among others they were used to create S-boxes [PPE$^+$14, PMMB15], thus it is

an excellent choice for creating S-boxes that fit our criteria (high resistance against side-channel attacks).

The idea of genetic algorithms comes from biology, more precisely genetic algorithms try to mimic *evolution through natural selection.* Main principles stay the same: a population of individuals evolve by means of reproduction (and inheritance of characteristics) and random mutations under the pressure of natural selection. However, instead of a population organisms we are interested in a set of objects that are often called *solutions* in the domain of genetic algorithms (in our particular case it is a set of S-boxes).

We encode solutions as lists of values between 0 and $2^n - 1$ where n is the size of the S-box. Note that this representation (permutation encoding) is highly efficient since it ensures that S-boxes are bijections (which is a condition we enforce on our S-boxes). The process of evolution in case of a genetic algorithm has 3 main steps: reproduction, mutation and selection (of the fittest). We worked with the 3-tournament selection mechanism which is the option that offers the fastest convergence [ES03]. This mechanism selects three solutions randomly and discards the worst solution. Afterwards, we create an *offspring* (a new S-box) from the remaining two solutions by using a *crossover operator.* Crossover operator is a procedure that specifies a way to create a new solution (an S-box) from two parents by combining them. We used a technique called *order crossover*, which works by first randomly selecting two crossover points and copying everything between those two points from the first parent to the offspring. Then, starting from the second crossover point in the second parent, the unused numbers are copied in the order they appear in that parent [ES03]. To be more specific to the case of S-boxes, a new offspring S-box is created from two parent S-boxes by copying a (randomly chosen) part of the first parent (order of values in the S-box) and filling-in the rest of the S-box by values from the second parent, where the order of values is the same as the order in this second parent. Thus, the new S-box inherits the order of values from its two parents. Mutations are used in order to introduce perturbations and more novelty into the population (set of S-boxes), we used a technique called *toggle mutation* which randomly selects two values and swaps them in the S-box. In order to bootstrap the whole process we use an initial population of 100 randomly generated S-boxes.

In order to compare individuals in the population (S-boxes in the set), genetic algorithms define a fitness function (that we want to maximise). In our experiments, we maximise the nonlinearity while minimising the differential uniformity as well as the Success Rate (SR), hence the subtraction from 2^n value and 1, respectively:

$$\mathsf{fitness} = \mathcal{N}_F + (2^n - \delta_F) + (1 - \mathsf{SR}). \tag{6.7}$$

We stop the whole process when 100 generations go by without improvement of the fitness function. We give equal weights to $\mathcal{N}_F$ and δ_F since our experiments show

there is no statistically significant difference in those two cases[9].

We would like to point out that although we work with genetic algorithms, our methodology is not exclusive for these techniques, it could work with any other heuristics that support the permutation encoding. Naturally, it is to be expected that in such case (a different heuristic method) one will probably have to change the fitness function and the termination criterion. For further details about genetic algorithms we would like to refer interested readers to the work of Eiben and Smith [ES03].

In Section 3.2.2 we already talked about the fact that sometimes an attacker will choose to focus on the first or the last round of the encryption and thus attack the S-box or the inverse of the S-box (which is basically "just another S-box"). We have also showed that different S-boxes of the same size can behave differently under side-channel attacks (they have different success rate, see Section 6.2). Thus, since adversaries often have a choice between the first and the last round (i.e., attacking an S-box or its inverse), *attackers can choose to focus on the easiest of the two targets*! Therefore, we actually used 3 slightly different fitness functions in our experiments, we refer to them as different evaluation strategies. The core of the fitness function stays the same in all the 3 strategies, but the success rate part of the fitness function was used in the 3 following ways:

- Forward strategy (F) — the success rate of the attack on the S-box is computed (i.e., adversary targets the S-box),
- Forward & Inverse strategy ($F + I$) — we compute the average between the success rate of the attack on the S-box and the success rate of the same attack on the inverse of the same S-box (i.e., adversary can choose to target the first of the last round),
- Kleptographic[10] strategy (K) — instead of minimizing the success rate we *maximize* it by using SR instead of $(1 - \text{SR})$ in Equation 6.7 (the adversary *is* the creator of the block cipher).

While F and $F + I$ strategies allow us to create new strong S-boxes that can be more resistant against the side-channel attacks, the K strategy is meant to emphasize the fact that is it possible to create a seemingly strong cryptographic primitive with a *backdoor*. In security, a backdoor is a secret weakness that is introduced in a scheme or its implementation during the design phase, these weaknesses can then be

[9]Note that in general case of a fitness function it is possible to sacrifice one parameter in order to boost another. However, in our case it is impossible to sacrifice the nonlinearity $\mathcal{N}_F$ in order to improve the success rate due to the fact that $\mathcal{N}_F \in \mathbb{N}$ and $\text{SR} \in \mathbb{R}$ and $0 < \text{SR} < 1$. In other words the minimal step in values of $\mathcal{N}_F$ is 1, while 1 is the maximum increase that the SR can get, thus, the whole fitness will decrease if $\mathcal{N}_F$ decreases while boosting the SR.

[10]Kleptography is the concept of creating seemingly strong cryptographic algorithms and protocols with hidden backdoors (weaknesses). These backdoors allow their creators to break cryptographic algorithms easily.

Table 6.2 – Properties of evolved S-boxes when considering CPA.

Size	Name	$\mathcal{N}_F$	δ_F	Strategy
4×4	Evolved$_{SR1}$	4	4	F
	Evolved$_{SR2}$	4	4	$F + I$
	Evolved$_{K}$	4	4	K
5×5	Evolved$_{SR1}$	8	6	F
	Evolved$_{SR2}$	8	6	$F + I$
	Evolved$_{SR3}$	10	6	$F + I$
	Evolved$_{SR4}$	10	4	$F + I$
	Evolved$_{SR5}$	8	6	F
	Evolved$_{SR6}$	8	4	F
	Evolved$_{SR7}$	10	4	F
	Evolved$_{SR8}$	12	2	F
	Evolved$_{SR9}$	12	2	$F + I$
	Evolved$_{K}$	8	4	K

exploited by the designer once the product is on the market. In our case the strong primitive is an S-box with good cryptographic properties (nonlinearity and differential uniformity) and the backdoor is its weakness against side-channel attacks.

In order to compare our generated S-boxes we used the following existing S-boxes that we name according to the algorithm which uses them:

- 4×4 S-boxes: Evolved$_{\text{CC}}$, Evolved$_{\text{TO}}$, Klein, Present and PRINCE;
- 5×5 S-boxes: Ascon, Keccak (Ketje, Keyak) and PRIMATE.

Where the S-boxes Evolved$_{\text{CC}}$ and Evolved$_{\text{TO}}$ were also generated using genetic algorithms while taking into account theoretical metrics (CC and TO) in order to estimate their resistance against side-channel attacks.

To test our method, we use two distinguishers: Correlation Power Analysis (CPA) and Template Attack (TA). Table 6.2 and Table 6.3 display all the generated 4×4 and 5×5 S-boxes taking into account respectively the CPA and TA. S-boxes are provided in the Appendix C. Note that our new 5×5 S-boxes have better nonlinearity and differential uniformity values than Keccak or Ascon, but we can easily adapt our strategy to output S-boxes with any combinations of values.

6.3.2 Results for Correlation Power Analysis

We generated simulated traces by considering that the leakage function is the Hamming weight and the leakage model of the adversary is the same (i.e., the adversary

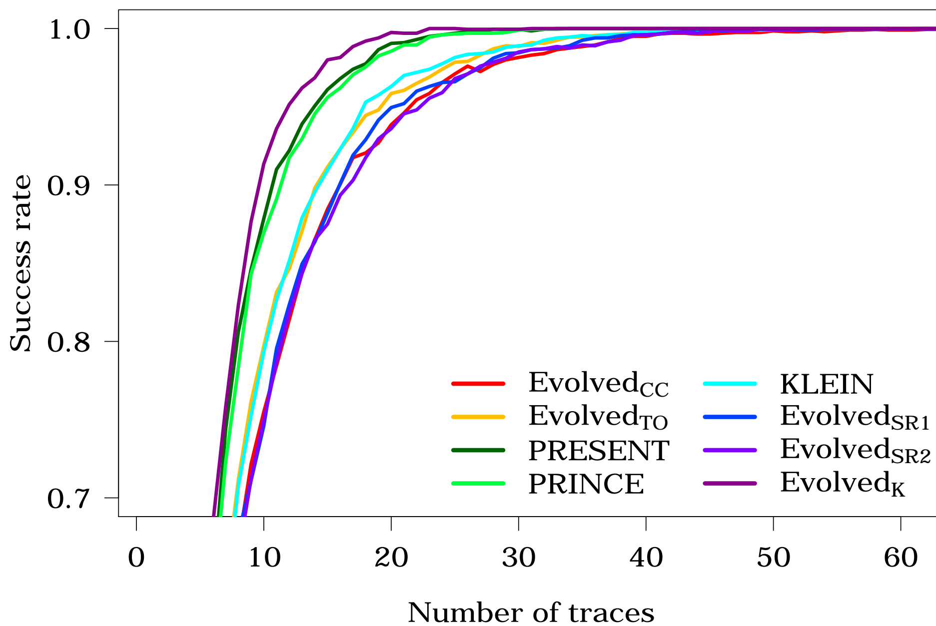
Table 6.3 – Properties of S-boxes Evolved for ATmega328P microcontroller using TA.

Size	Name	$\mathcal{N}_F$	δ_F	Strategy
4×4	EvolvedTA$_{SR1}$	4	4	F
	EvolvedTA$_{SR2}$	4	4	$F+I$
	EvolvedTA$_{SR3}$	4	4	F
	EvolvedTA$_{SR4}$	4	4	$F+I$
5×5	EvolvedTA$_{SR1}$	8	6	F
	EvolvedTA$_{SR2}$	10	6	$F+I$
	EvolvedTA$_{SR3}$	12	2	F
	EvolvedTA$_{SR4}$	12	2	$F+I$
	EvolvedTA$_{SR5}$	10	4	F
	EvolvedTA$_{SR6}$	8	4	$F+I$

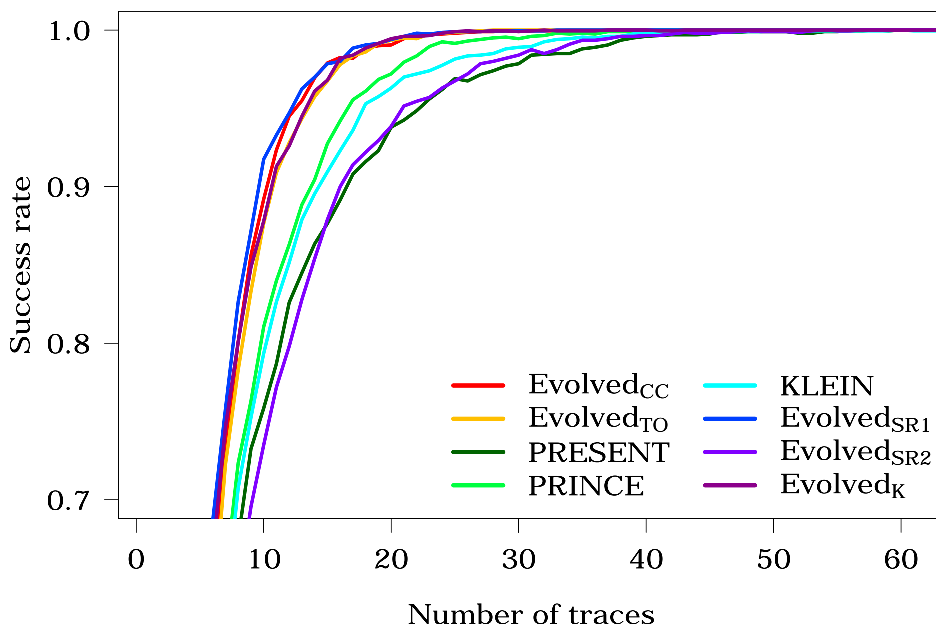
has a perfect knowledge on how the device leaks information). We use the same level of noise of 0.5 variance representing a signal-to-noise ratio (1) of 2.13 when considering 4×4 S-boxes, and (2) of 2.58 when considering 5×5 S-boxes. It is worth to note that the order of the (generated) S-boxes sorted by the resistance against side-channel analysis is not influenced by the SNR.

Figures 6.8 and 6.9 provide the success rate of CPA on the newly generated S-boxes as well as their inverses compared to the existing S-boxes[11]. The first observation is that the nonlinearity of an S-box and its differential uniformity are not the only metrics impacting side-channel attacks (e.g., all the 4×4 S-boxes have the same nonlinearity and differential uniformity but differ from the point of view of side-channel analysis). Furthermore, the S-boxes generated using F strategy (attack on the first round, forward direction) as well as the already known S-boxes are weak against side-channel attacks when considering an adversary who targets the last round of the cipher (i.e., attacking the inverse of the S-boxes). The generated S-boxes taking into account such adversary ($F+I$ strategy) provide good side-channel resistance in forward and in inverse direction. The new 4×4 S-box Evolved$_{\mathrm{SR2}}$ happens to be the best generated S-box among all of the considered S-boxes. From the point of view of kleptography, the generated 4×4 Evolved$_{\mathrm{K}}$ turns out to be the best: it has good cryptographic properties and it is the easiest S-box to attack using side-channel information. Note that the S-boxes Evolved$_{\mathrm{CC}}$ and Evolved$_{\mathrm{TO}}$ differ from a side-channel point of view. The rationale is that the Confusion Coefficient and the Transparency Order are not equivalent, as we already reported in the previous section. Moreover, they both turn out to be very bad (with respect to side-channel analysis) in the inverse

[11]Please, note the vertical scales of these figures, they are not identical for S-boxes of different sizes and do not start at 0 for a better visual representation. Same scale figures can be found in the Appendix C.1

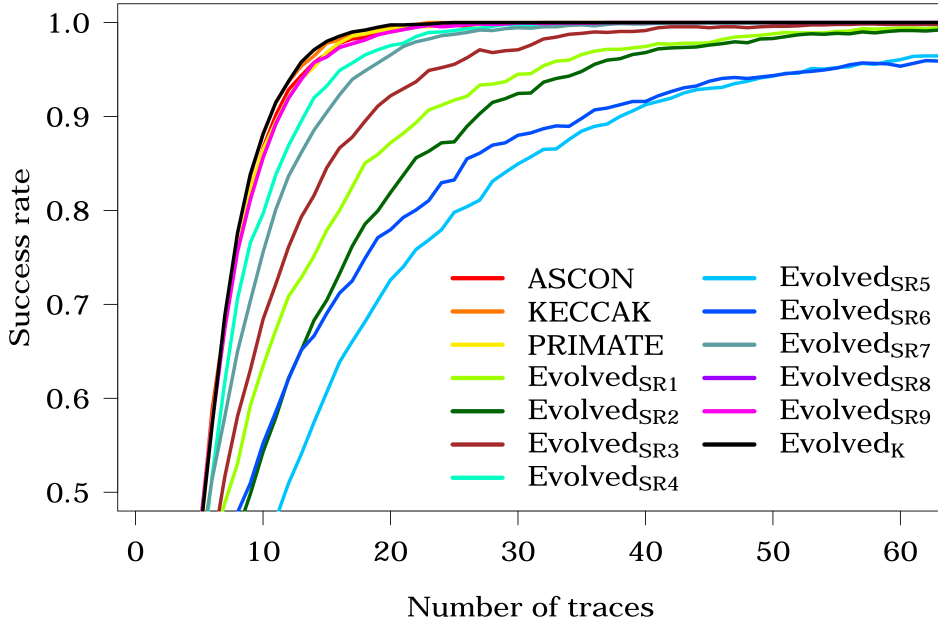


(a) 4×4 S-boxes

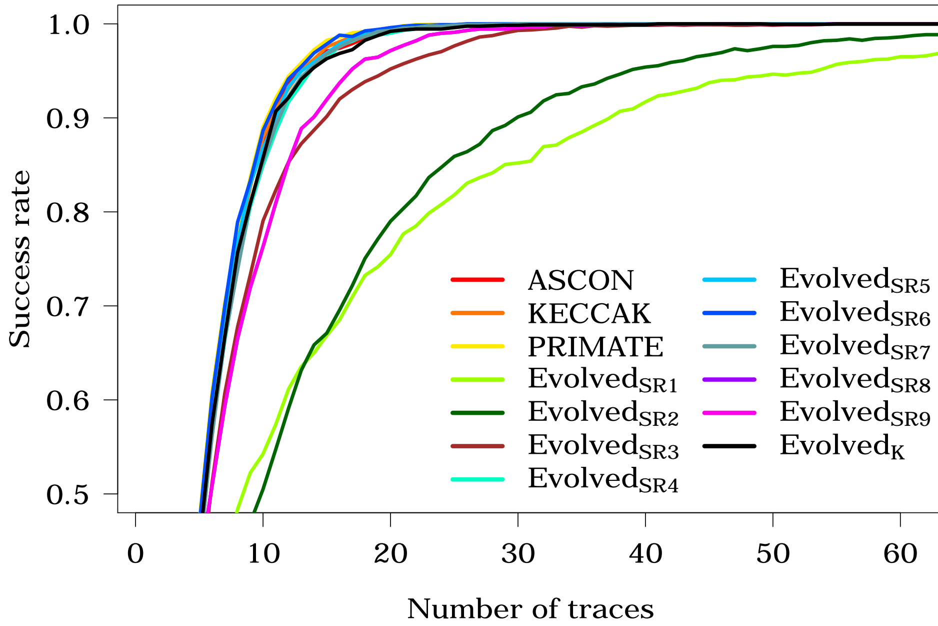


(b) Inverses of 4×4 S-boxes

Figure 6.8 – Success rate of CPA on 4×4 S-boxes.



(a) 5×5 S-boxes



(b) Inverses of 5×5 S-boxes

Figure 6.9 – Success rate of CPA on 5×5 S-boxes.

direction.

Regarding the 5×5 S-boxes, we generated several S-boxes having different cryptographic properties (by varying the value of the differential uniformity and the nonlinearity metrics). This palette of S-boxes gives rise to 9 S-boxes having different levels of resistance against side-channel attacks. All the generated S-boxes provide a higher resistance compared to the existing (considered) S-boxes while having good cryptographic properties. This allows the designer to choose S-boxes among several options with cryptographic properties that fit their requirements.

6.3.3 Results for Template Attacks

We repeated the same experiments using a different leakage model and another distinguisher. This time we extracted a leakage model from a real device (a microcontroller) and then used the extracted model as a leakage function during our simulations.

In order to extract a leakage model from the target device we used the profiling step of the TA and the following setup for the data acquisition. A set of 80 000 power traces was collected on an 8-bit Atmel (ATmega328P) microcontroller at a 16 MHz clock frequency. The power consumption of the device was measured using an Agillent Infiniium 9000 Series oscilloscope that was set up to acquire 200 MSamples/s. Otherwise the hardware setup was the one that we described in Section 3.2.1 (Figure 3.4). In order to reduce noise in traces we used averaging, thus each power trace represents an average of 64 single acquisitions. Our target device was using a random data (processing plaintexts with AES S-box to get different target values for profiling). We target the first round of the cipher and focus on the first byte of the key. We extracted the leakage model L of the device by averaging all traces associated to the same target value and by selecting the 8 instants that are the most (linearly) correlated with the target value. During the attack phase of our experiments we used one trace in the attack set. We used the extracted leakage model during our simulations as a leakage function in the simulator with a small additional Gaussian noise[12] having a standard deviation of 5×10^{-6}. This leads to a SNR of 0.40 and 0.37 for the best point when considering respectively an 4×4 S-box and an 5×5 S-box. It is worth to note that we do not claim that this profiled attack represents the optimal physical attack against the analysed implementation. Other profiled attacks could provide higher success rates [LPB$^+$15]. In other words, our purpose here is to provide S-boxes resilient against a *chosen* profiled attack as a proof of concept.

Figures 6.10 and 6.11 show the success rate of TA on the considered S-boxes. We can notice that Figure 6.10a and Figure 6.10b show results similar to the results that we obtain in the previous section: when we consider well-known S-boxes or newly generated S-boxes (while considering only the forward strategy) the corre-

[12] A small amount of noise is necessary in order to avoid numerical issues during TA.

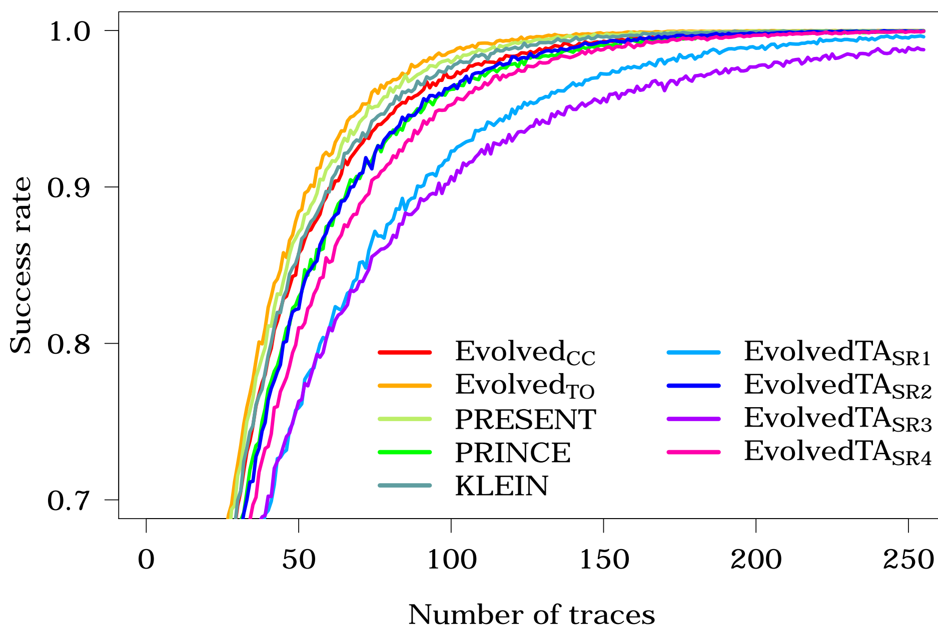
sponding inverse S-boxes show them weaker against side-channel attacks. The 4×4 EvolvedTA$_{SR2}$ S-box that was generated by taking into account the S-box and its inverse gives the best result: it is as good as Present S-box in terms of its inverse and it is one of the best among well known 4×4 S-boxes (in the forward direction) with the exception of 4×4 EvolvedTA$_{SR1}$ that was designed to be good in the forward direction (but not as an inverse). In terms of 5×5 S-boxes, 5×5 EvolvedTA$_{SR5}$ provides the best result in forward direction. In the inverse direction, EvolvedTA$_{SR6}$ outperforms all the known S-boxes. Note that it is still difficult to create resilient S-boxes while having good cryptographic properties and being better than existing S-boxes in both forward and inverse directions. However, we deem that we can still create a more resilient 5×5 S-box against TA since 5×5 S-boxes provide a large set of possible solutions.

6.3.4 Discussion

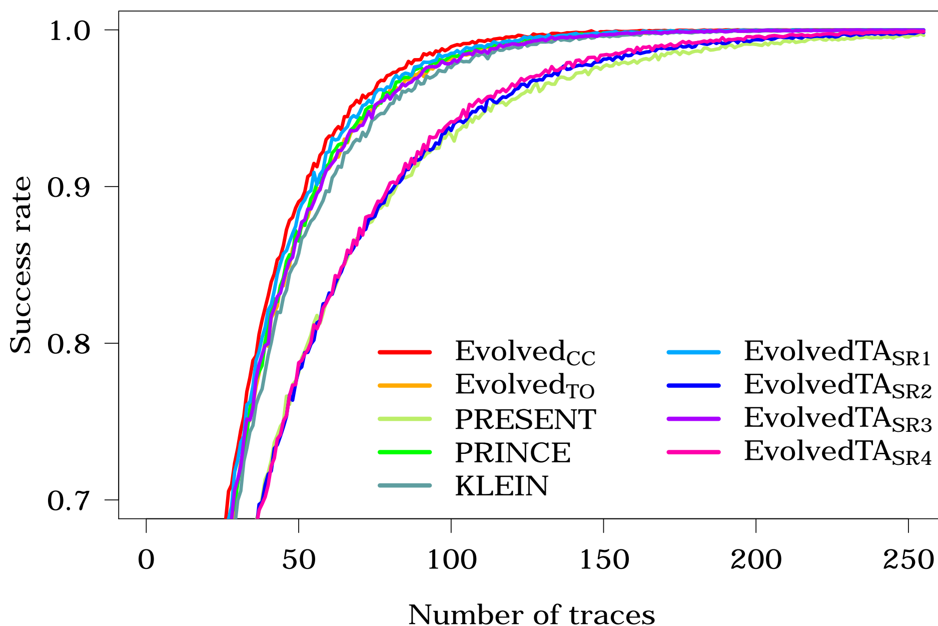
The previous sections report the improvement of the success probability of physical attacks on 4×4 and 5×5 S-boxes. Our results highlight that the improvement is more significant for the 5×5 S-boxes than for the 4×4 S-boxes. The reason relies on the fact that 5×5 S-boxes have a wider range of obtainable values (more different permutations) when compared with 4×4 S-boxes.

Figures 6.12 and 6.12 provide the success rate on each S-box targeted by an adversary exploiting the plaintext (by attacking the forward S-box used in the first round of the cryptographic primitive) and the ciphertext (by attacking the inverse S-box used in the the last round of the primitive). Plots on these figure correspond to the maximum of the two attacks. These results highlight the usefulness of our approach by providing new S-boxes outperforming well known S-boxes in several contexts. More precisely, the 4×4 Evolved$_{SR2}$ and the 5×5 Evolved$_{SR2}$ S-boxes provide the best results against CPA while the 4×4 EvolvedTA$_{SR4}$ and the 5×5 EvolvedTA$_{SR6}$ S-box provide the best results against TA.

Note also that all our results report the success rate of adversaries targeting *one* part of the key. It is worth to note that, in practice, adversaries extract the *full* secret key. As a result, a small-scale decrease of the first order success rate of an attack on one part of the key leads to a significant reduction of the success probability of the attack on the full key. Therefore, designers of cryptographic primitives should consider optimisation methods minimising the success rate of physical attacks against S-boxes. As an example, let us take two 4×4 S-boxes with similarly close success rates: Evolved$_K$ and the S-box of Present. During a CPA using 15 attack traces, Evolved$_K$ results in success rate of 0.9820 while the S-box of Present gives the success rate of 0.9605 (difference of about 0.02). However, it is important to note that this success rate corresponds to an attack on one 4-bit nibble. During an attack on a full cipher with 80-bit key, the adversary repeats the attack on each nibble (i.e., 20

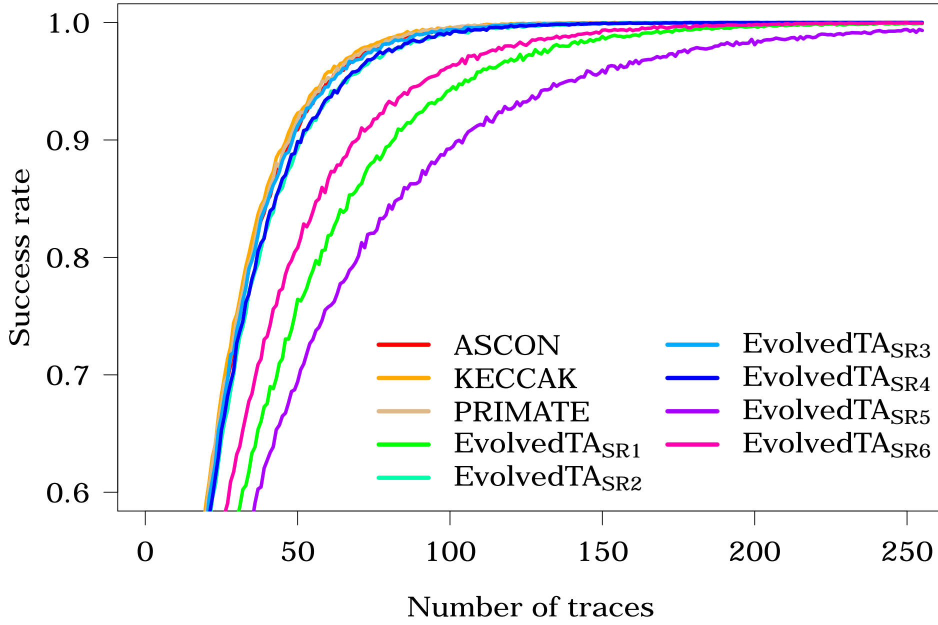


(a) 4×4 S-boxes

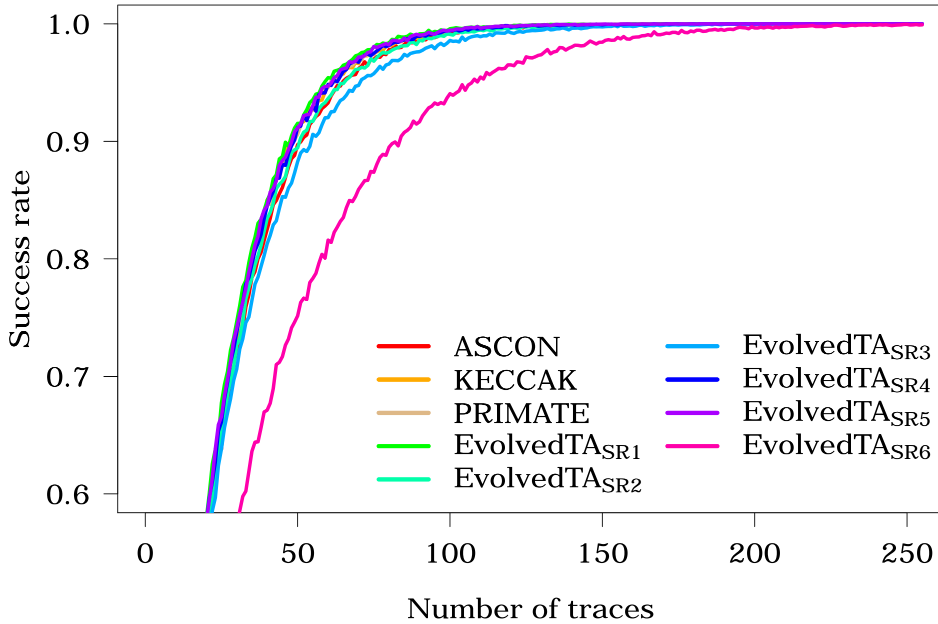


(b) Inverses of 4×4 S-boxes

Figure 6.10 – Success rate of TA on 4×4 S-boxes.

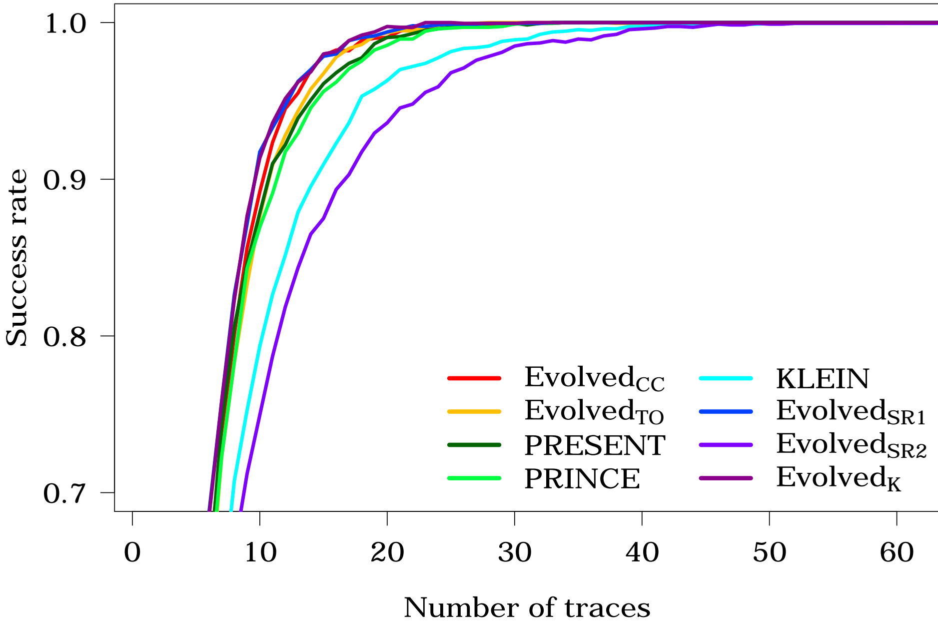


(a) 5×5 S-boxes

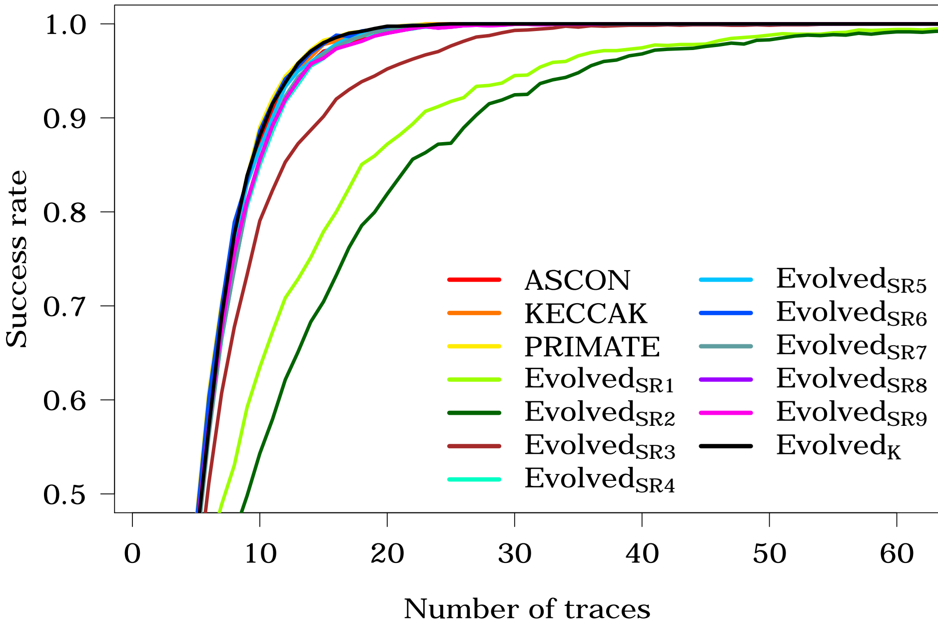


(b) Inverses of 5×5 S-boxes

Figure 6.11 – Success rate of TA on 5×5 S-boxes.

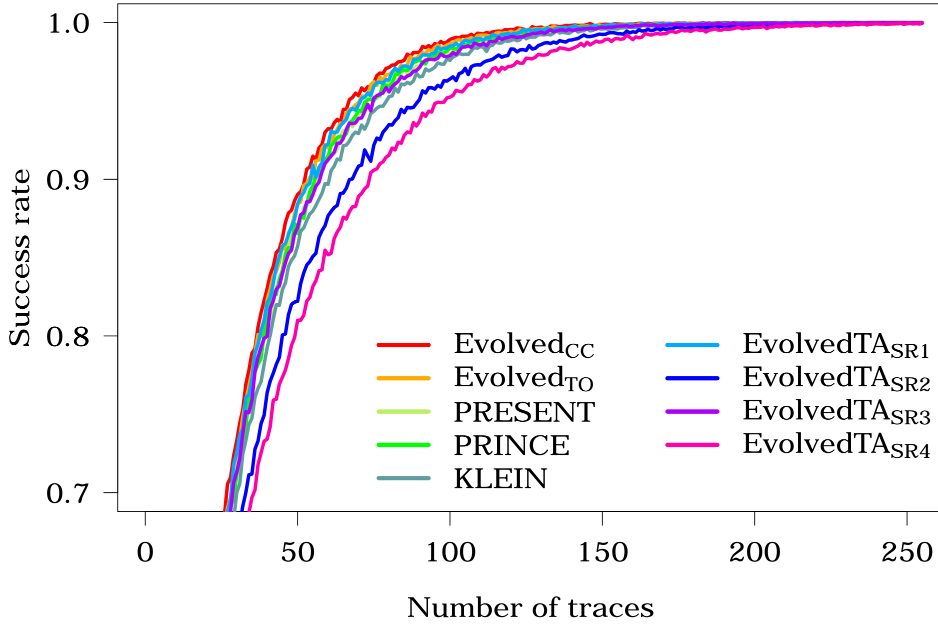


(a) Correlation power analysis on 4×4 S-boxes

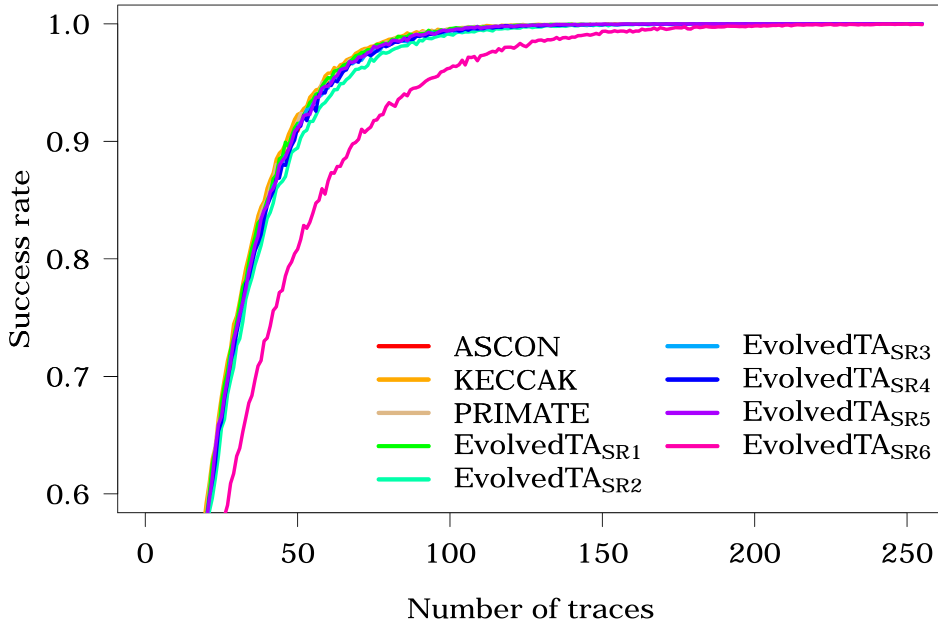


(b) Correlation power analysis on 5×5 S-boxes

Figure 6.12 – Maximum success rate between CPA attacks on the first round (S-box) and last round (inverse of the S-box) of an algorithm.



(a) Template attacks on 4×4 S-boxes



(b) Template attacks on 5×5 S-boxes

Figure 6.13 – Maximum success rate between TA on the first round (S-box) and last round (inverse of the S-box) of an algorithm.

times). Thus, the success rate of a complete attack results in the success rate of 0.4466 on the PRESENT S-box and 0.6954 in case of Evolved$_K$ which is a significant increase even though the success rates of attacks on one nibble are very close. Nevertheless, it is still not the "full story", indeed here we only consider the success rate of order 1 i.e., we are only looking at the top candidate after the sorting process performed by a distinguisher. Thus, it is possible that the correct candidate (real secret key) is still easily attainable using a final key enumeration (recall Section 3.2.5). An interesting direction for the future research could be using genetic algorithms with more criteria and with the success rate of a higher order or using the guessing entropy in the fitness function (which would allow to effectively block the key enumeration).

We would like to emphasise that the method we are using is a passive approach towards the resistance of implementations against side-channel attacks. It does not actively change the behaviour of the device through algorithmic means such as masking or shuffling. Nevertheless, this idea comes with no costs for the final product (no randomness or extra memory requirements and no time overheads), the costs are only related to the development of the cipher. Cautionary note consists in the fact that creation of such S-boxes (or other operations involved in a block cipher) does not give any guarantees on the leakage reduction in the final product: first of all, a different device probably have a different leakage function (and thus the HW results would not apply) and secondly, the attacker can also target a different operation. Moreover, the success rate of a specific attack (with a chosen leakage model) is only a *metric* — it does not give a guarantee that the final product will not be attackable. A very similar idea can be applied to the design of block ciphers, properties such as nonlinearity and differential uniformity are also only metrics; if an S-box has good properties does not imply that a cipher that uses it will be "unbeatable", the problem of composability is actually a big issue in the whole area of cryptography. Combining several secure elements does not necessarily result a secure product. The rationale is that the final implementation of a cipher has to be tested in all its aspects even if each single step relies on good design.

6.4 Scalable shuffling schemes

The goal of this section is to show how SILK can be used in order to compare a large number of different countermeasures using the example of shuffling schemes. One of the difficulties that evaluators can encounter if they try to compare many different shuffling schemes in practice (using physical experiments) is the amount of time that they have to spend on the data acquisition. We were able to perform a lot of experiments in parallel because we used a simulated environment.

A shuffling algorithm is the countermeasure against side-channel analysis, the idea of this countermeasure is based around changing the order of independent operations that manipulate the internal state at every execution (recall Section 3.4.2).

Only several different shuffling schemes were presented in literature: Random Permutation (RP), Random Starting Index (RSI) and scheduling (recall Section 3.4.2). The later one cannot be easily and fairly compared with the first two from all points of view, since they do not shuffle same type of operations. In order to have more different shuffling schemes for the comparison we invented new shuffling techniques some of which are extensions of the RSI scheme. Here we are going to describe 3 families of scalable shuffling techniques. We start by describing the shuffling algorithms and then proceed with their comparisons from different points of view including side-channel analysis.

For the sake of simplicity all examples presented in this section are given for the SubBytes operation (application of the S-box) on the state of 128-bit version of the AES block cipher. This section presents shuffling techniques on the example of the first round of AES, but same shuffling techniques can be applied to any number of rounds depending on system's requirements and amount of available resources (time, memory, amount of random bits, etc.). All presented shuffling algorithms can be easily adapted for other operations of AES as well as for other algorithms.

The constraints on the RNG of the cryptographic system can be specified in different ways, most importantly we can say that we have a fixed amount of random bits available per unit of time, the unit of time can be one clock cycle, one round of a cipher or even a full encryption of one block. For the sake of simplicity we will be always be referring to generic units of time instead of a more specific choice such as e.g., a clock cycle.

Most of the shuffling techniques suggested in this section are based on the fact that the internal state might be seen as a vector or as a matrix. Indeed, in the memory of a computer, a vector of size 16, a 4×4 matrix or even a 2×8 matrix are all just arrays of 16 memory units (in our case bytes).

6.4.1 Extensions of random start index

Basic version of Random Starting Index (RSI) shuffling for AES-128 represents the AES state as a vector of 16 elements. S-box is applied on all 16 bytes one by one starting from a randomly chosen index (between 0 and 15). This shuffling technique requires 4 random bits and it gives us 16 possible starting indexes (and 16 different shuffles in total).

Two different variations of the basic RSI technique might be implemented, those techniques generalize RSI and might be applied with less or more random bits (between 1 and 11 bits in our AES-128 examples).

Vector-RSI

Vector RSI (V-RSI) extension, uses the same representation of the AES-128 state as the basic RSI, the state is used as a vector and a random starting index might be chosen

with less than 4 bits of randomness. It might be done by giving a fixed value to all missing bits, by reusing some of the available random bits (eventually by combining them) or even by combining these two approaches, see Figure 6.14.

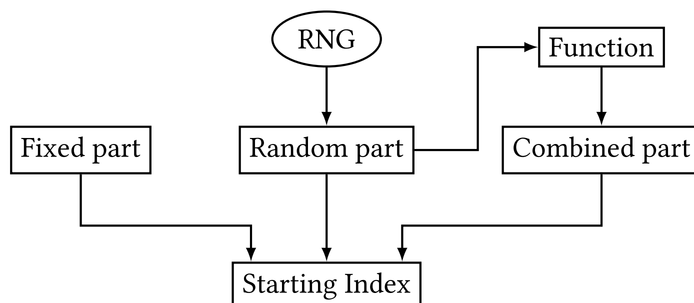


Figure 6.14 – Structure of V-RSI index generation. The order of bits coming from different parts might be chosen arbitrarily.

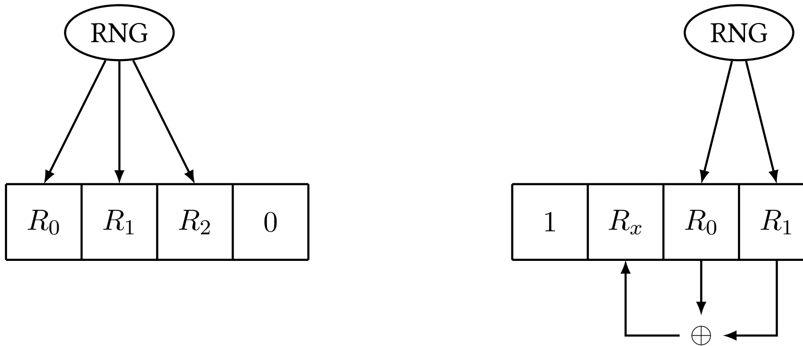
For example, if we have only 3 available random bits for the V-RSI (due to the constraints on the RNG), we can fix the position of the missing one as the LSB of the starting index and always assign its value to 0. In this case we will have 8 possible shuffles with only even numbers as starting indexes, see Figure 6.15a.

Here is another example, lets say we have only 2 random bits per unit of time and we would like to use V-RSI shuffling. We can fix those random bits as two LSBs of the index, fix the value of the MSB of the index to 1 and assign the value of the second MSB to the exclusive-or (xor) of the two available random bits. If we use this algorithm to generate the starting index we will be able to generate 4 different shuffles that might start with indexes 8, 11, 13 or 14, see Figure 6.15b. This last example is not practical, especially in software implementations since it requires additional computations. Nevertheless, this example is meant to illustrate that we can actually *choose* a set of any starting indexes for an implementation by choosing how to assign values to missing bits that are needed in order to have a random index.

An idea of using a 3-part computation (fixed part, random part and combined part) in order to generate a random index might be used in a scenario when the shuffling scheme used by the device is the same for all devices, but each *instance* is different thanks to different combination functions and different fixed parts. In such scenario each device will use a different shuffling, thus an attacker will have less chances of being able to profile one device in order to attack another one.

Matrix-RSI

The second extension, that we call Matrix RSI (M-RSI), of the RSI technique handles the internal state of AES as a matrix and treats it row by row. Since the state is handled row by row, we can just apply the V-RSI on each row. Since all rows are



(a) Using 3 random bits and 1 fixed bit, giving all even numbers.

(b) Using 2 random bits, 1 fixed bit and 1 combined. Giving 8, 11, 13 and 14.

Figure 6.15 – Examples of V-RSI use with 2 and 3 available random bits.

handled separately, we can also start with any row i.e., we can reuse V-RSI technique in order to choose the starting row. This technique allows us to shuffle SubBytes operation with 1 to 10 random bits and can give us from 2 to 1024 possible shuffles in case we handle the state in a classical 4×4 configuration. Same approach can go up to 11 random bits while generating 2048 possible shuffles if we handle the state for the sake of shuffling as a 8×2 matrix (8 rows and 2 columns).

Table 6.4 shows how M-RSI might be applied on AES-128 state (handled as a 4×4 matrix) depending on the number of available random bits per unit of time. This table is structured as follows, *All rows* part shows how we can go through all rows and *Cells in a row* part shows how we may handle all cells in one row. Following notations are used: *Fixed* — normal, non-random algorithm is used (e.g., 0, 1, 2, 3); *Rand(n)* — starting index is chosen using n random bits; *S* — same random numbers are used to get the starting index in each row, *D* — different random bits are used to generate the starting index in different rows. For example, if we have 6 available random bits and we want to use M-RSI, according to the table we might use 2 bits in order to choose a random row to start with and we can also use 1 bit per row in order to choose a random starting index in each row (using V-RSI with 1 bit on a vector of 4 bytes).

Notice that this table only gives some examples of how to use M-RSI per number of available bits, multiple combinations might be implemented for some numbers e.g., for 4 bits we might also use 2 bits in order to choose a starting row and then use 2 random bits in order to choose a random start cell (same in each row). Some of these choices might be more efficient and/or more secure than others. This dependency is also influenced by the hardware (i.e., available instructions).

Unfortunately, we were not able to find a “nice” combination that could be implemented efficiently (that does not need special cases, when implemented) for 7 available random bits.

Table 6.4 – Examples of M-RSI use on 4×4 AES-128 state using different number on random bits.

Available	All rows		Cells in a row		Number of
Random bits	Bits	Handling	Bits	Handling	Shuffles
1	1	Rand(1)	0	Fixed	2
2	2	Rand(2)	0	Fixed	4
3	2	Rand(2)	1	Rand(1), S	8
4	0	Fixed	4	Rand(1), D	16
4*	2	Rand(2)	2	Rand(2), S	16
5	1	Rand(1)	4	Rand(1), D	32
6	2	Rand(2)	4	Rand(1), D	64
8	0	Fixed	8	Rand(2), D	256
9	1	Rand(1)	8	Rand(2), D	512
10	2	Rand(2)	8	Rand(2), D	1024

* The second version with 4 bits offers more security, see Section 6.5 and Table 6.8.

6.4.2 Reverse shuffle

The idea behind the simplest version of Reverse Shuffle (RS) technique is the following: AES-128 state is used as a vector of 16 bytes, S-box is applied to all bytes of the state following forward or reversed order (depending on the value of 1 random bit). For example, if the value of the random bit is 0 we may go through the state from byte 0 to byte 15 and if the value of the random bit is 1 we can go through bytes in the reversed order (from 15 to 0).

Matrix-RS

RS might be extended by using the state of AES-128 as a $m \times n$ matrix instead of a vector (where $m \times n$ is the size of the original vector, 16 in our case), we are going to call this extension Matrix-RS (M-RS). We will specify the exact M-RS version by using the notation M-RS $m \times n$. Note that M-RS 1×16 gives us the original simple RS.

The idea behind M-RS 4×4 is the following: we can use RS on each row (of 4 bytes) as well as for all rows (start from row 0 or row 3 in the matrix). It allows us to use from 1 up to 5 random bits for shuffling. For example, if we have 4 random bits we can go through all rows in forward order (no randomness required), we can also go through all cells in each row in forward or reversed order (different order for all rows, 4 bits of randomness), see example in Figure 6.16, also see Table 6.5.

Table 6.5 shows how M-RS 4×4 might be applied on AES-128 depending on the

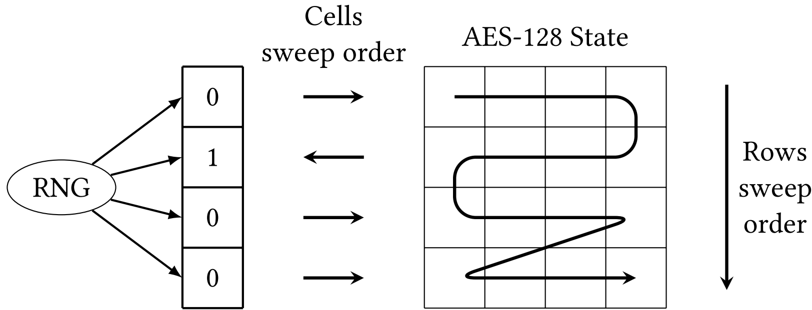


Figure 6.16 – Example of M-RS 4×4 with 4 available random bits.

Table 6.5 – Examples of M-RS use on 4×4 AES-128 state using different number on random bits.

Available	All rows		Cells in a row		Number of
Random bits	Bits	Handling	Bits	Handling	Shuffles
1	1	Rand	0	Fixed	2
2	1	Rand	1	Rand, S	4
3	1	Rand	2	Rand, 2S	8
4	0	Fixed	4	Rand, D	16
5	1	Rand	4	Rand, D	32

number of available random bits. This table uses following notations: *Rand* means that indexes are handled in forward or reversed order randomly, *Fixed* means that same fixed order is used to go through cells in a row (or rows in the matrix); *S* means that same random bits are used on several rows[13], *D* means that different random bits are used for all rows.

Since a 16 byte AES-128 state might be represented as a matrix in several different ways (matrix of different size), we may use it to our advantage while using more or less random bits for shuffling. If we want to use more than 5 random bits and generate more shuffles we can use M-RS 8×2 shuffle, it will allow us to use up to 9 random bits (1 bit per row and 1 bit for all rows) and generate 512 shuffles.

6.4.3 Sweep swap shuffle

The idea of Sweep Swap Shuffle (SSS) is based on the fact that the state of AES-128 might be represented as a $m \times n$ matrix (e.g., a 4×4 or a 2×8 matrix). A matrix

[13] 2S in line 3 means that same bits are reused 2 times on 2 different rows and then different random bits are used on 2 other rows

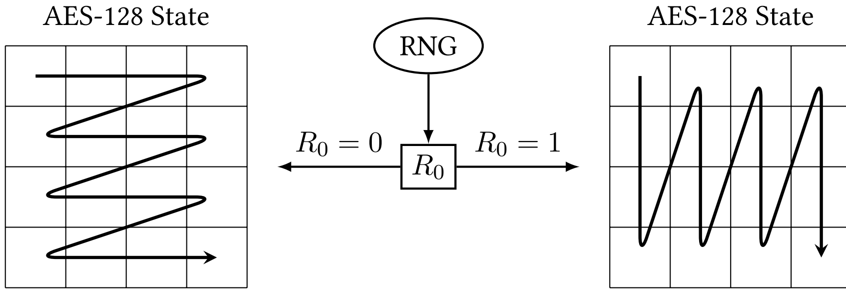


Figure 6.17 – Going through bytes of AES-128 state matrix with SSS 4×4.

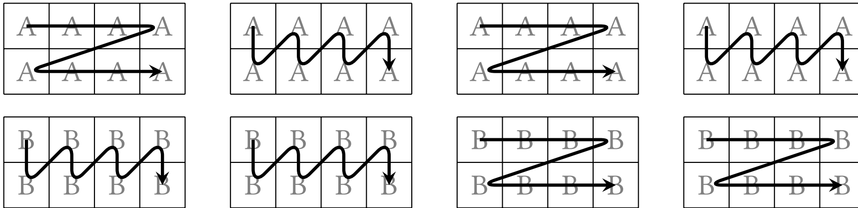


Figure 6.18 – Four possible shuffles of AES-128 state with P2-SSS 2×4 technique using 2 random bits.

might be handled row-by-row or column-by-column. SSS might also be implemented e.g., by swapping pieces of code that go through row and column indexes. In order to specify how a vector is represented as a matrix we will use the notation SSS $m \times n$. Figure 6.17 shows two possible orders of SSS 4×4.

Part SSS

The idea behind Part-SSS (P-SSS) extension of SSS technique is the following: a state of AES-128 might be broken into several equal parts e.g., 2 parts of 8 bytes. An SSS technique could be then applied to each part separately, it would allow us to create more different shuffles (by using more than 1 random bit). We will use the notation P∂-SSS $m \times n$ in order to specify the number ∂ of identical parts that we want to use. Note that P1-SSS 4×4 gives us the original SSS 4×4. See example of P2-SSS 2×4 in Figure 6.18

By using P-SSS on AES-128 we can generate up to 16 shuffles by using 1 to 4 random bits, see Table 6.6.

Table 6.6 – Examples of P-SSS use on AES-128 state using different number on random bits.

Random bits (and ∂)	Technique	Shuffles
1	P1-SSS 4×4	2
2	P2-SSS 2×4	4
4	P4-SSS 2×2	16

Table 6.7 – Examples of MD-SSS use on AES-128 state with different number on random bits.

N_Δ	Random bits	State representation	Shuffles
2	1	2×8	2
3	3	$2 \times 4 \times 2$	6
4	5	$2 \times 2 \times 2 \times 2$	24

Multidimensional SSS

The idea behind Multidimensional SSS (MD-SSS) extension of SSS technique is based on the fact that a vector might be seen as a multidimensional matrix[14]. For example the state of AES-128 might be seen as $2\times4\times2$ matrix, also see examples on Figure 6.19. It allows us to go through all dimensions in any order, e.g., in 2 dimensions the state might be handled row by row or column by column (go through the first dimension then through the second one or the other way around).

To specify a version of SSS we are going to use the notation MD-SSS $\Delta_1 \times \Delta_2 \times \cdots \times \Delta_{N_\Delta}$, where N_Δ is the number of dimensions and Δ_i is the size of the state in the dimension i. The number of shuffles that can be generated with MD-SSS depends on the number of dimensions that is used to represent the state for the shuffling. Since we can choose any ordering of dimensions to handle the state, the number of different shuffles that might be generated is given by $N_\Delta!$ and thus the number of necessary random bits is given by $\lceil \log_2 N_\Delta! \rceil$. Table 6.7 gives several examples of MD-SSS used with AES-128 state using different number of available random bits.

[14]It is important to note, that we can think about the state as if it was a three dimensional matrix for the purpose of shuffling, but it does not mean that the state has to be represented and manipulated as such during the entire algorithm.

AES-128 state as a vector

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

2 dimensions

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

3 dimensions

8	9	A	B
C	D	E	F

1	2	3	4
5	6	7	8

Figure 6.19 – Examples of representations of AES-128 state as multidimensional matrices.

6.5 Analysis of shuffling schemes

In order to study our shuffling algorithms as well as to compare them to the existing schemes from the theoretical point of view, we introduce a couple of new terms and definitions. For the efficiency analysis we implement some of them in an ATmega382P microcontroller and also perform a range of side-channel analysis based on simulations provided by our Silk tool.

6.5.1 Randomization

A *Randomization range* or *randomization interval* of a shuffling technique is the biggest interval where the shuffling algorithm operates and where the shuffled operations might be reordered.

A randomization range of a shuffling technique might be one operation (e.g., AddRoundKey), one round (or several operations of one round), several rounds or the entire algorithm. If the same shuffling technique is applied on SubBytes operation of all rounds of AES, then the randomization interval of this technique is still one operation (SubBytes) since instructions in between different SubBytes are not reordered among them.

The randomization range of all our shuffling techniques is one operation (SubBytes, as presented at the beginning of Section 6.4). RP also has a randomization range of one operation. SchedAES has a very wide randomization range and it allows to generate many different shuffles but requires a huge amount of randomness (up to 3 000 bits per encryption) and memory.

A *fully randomized* instruction (or operation) is an instruction that might be re-

ordered and executed at any instant in time by a given shuffling technique inside of its randomization range without changing the final result of the algorithm that is being shuffled. A *partially randomized* instruction is an instruction that is not fully randomized, but that might be reordered and executed during at least 2 different instants in time by a given shuffling technique inside of its randomization range. An *unrandomized instruction* is an instruction that is always executed at the same moment in time inside of the randomization range using a shuffling technique.

We will say that shuffling algorithm is fully randomized if all instructions inside of its randomization range are fully randomized. If at least one instruction is unrandomized or only partially randomized, then the shuffling technique is partially randomized. Note that there cannot be only unrandomized instructions in a shuffling algorithm, since in this case there would be no shuffling at all.

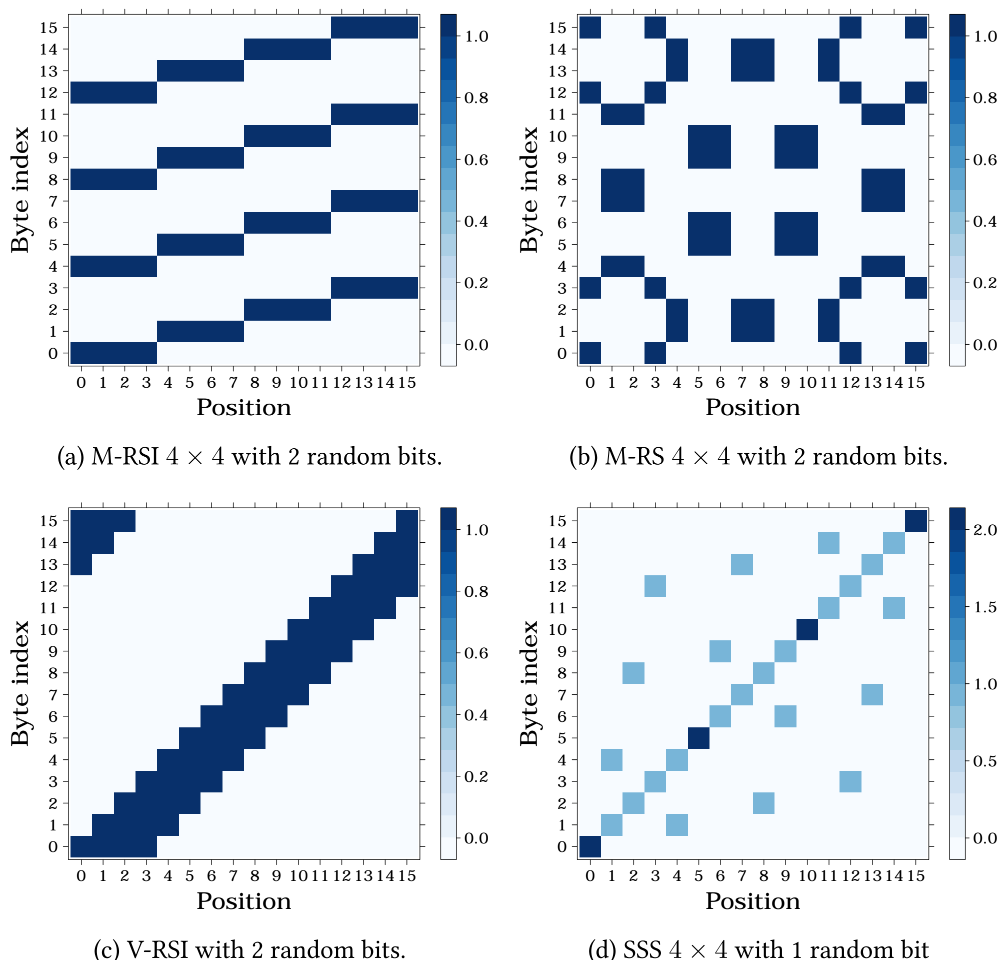
RSI applied to SubBytes is fully randomized in its basic version, but if we use less random bits (as in V-RSI) it becomes only partially randomized. Different versions of M-RSI might be fully randomized or partially randomized depending on choices made during the implementation (different number of random bits used to choose the start index for rows and inside of each row). RS and its extensions are always partially randomized and it does not have unrandomized instructions if used with AES. RP is also fully randomized.

Unfortunately, SSS is partially randomized and have unrandomized instructions since some bytes are always used at the same moment in time, indeed the S-box is always applied on the first byte at the beginning and on the last byte at the end of the SubBytes operation. Moreover, if we use e.g., SSS 4×4 on AES-128 S-box on bytes 0, 5, 10 and 15 are unrandomized since these bytes are situated on the diagonal of the 4×4 matrix. In general, handling a square matrix row by row or column by column does not change the moment in time when the elements on the diagonal are used. Thus, SSS $n \times n$ will have n unrandomized instructions.

In order to analyse all of the proposed shuffling schemes, we executed each one of them through the entire range of possible random inputs that each algorithm could receive as a parameter. For every algorithm we generated a heatmap of all possible positions (moments in time) where a SubBytes operation can take place on every single byte id. See examples of such plots in Figure 6.20, other figures are available in the Appendix D. We can see that, for each scheme, available positions for each byte are uniformly randomly distributed, with the exception of 4 bytes of SSS (the bytes that are situated on the diagonal of the matrix). The exact patterns that we can observe on the heatmaps depend on the way the scheme was implemented (i.e. which bits were chosen to be fixed and which are random, recall Figures 6.14 and 6.15).

We also generated same type of heatmap for the RP shuffling scheme, see Figure 6.21. We used the implementation of RP shuffling scheme from DPA Contest 4.2[15] [BBD$^+$14]. It is currently impossible to enumerate all possible inputs (ran-

[15] http://www.dpacontest.org/v4/data/v4_2/smart_v42_2.zip



(a) M-RSI 4×4 with 2 random bits.

(b) M-RS 4×4 with 2 random bits.

(c) V-RSI with 2 random bits.

(d) SSS 4×4 with 1 random bit

Figure 6.20 – Examples of heatmaps of positions when the SubBytes operation takes place for every byte.

domness) required by this shuffling scheme in a reasonable amount of time, so the heatmap from Figure 6.21 is generated using 2^{35} randomly chosen permutations. Using this approximation we can see that the ratio between the highest number of occurrences of a byte at a given position to the lowest number is equal to 1.000116, which is less than 0.01% of difference which approximately equals to 2^{-13}.

6.5.2 Number of shuffles

We will say that a shuffling algorithm is *optimal* if it is able to generate 2^n different shuffles using n random bits. If we have n random bits of information we will be

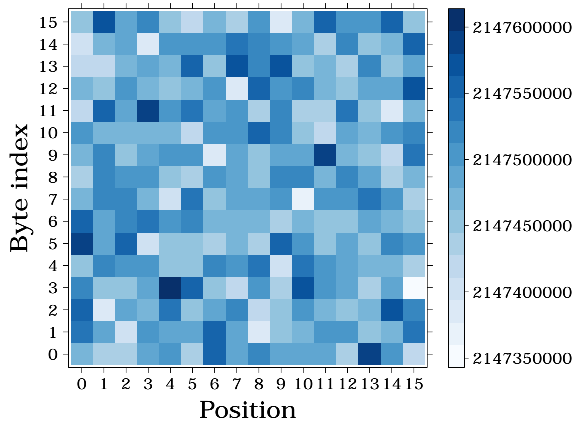


Figure 6.21 – Heatmap of RP shuffling scheme. Implementation from DPA Contest 4.2 [BBD+14]

able to generate at most 2^n different values. If a shuffling algorithm uses n bits of randomness and generates less than 2^n different shuffles, then it is not an optimal shuffling algorithm (from the point of view of information theory).

RS, RSI and all of their extensions use n bits in order to generate 2^n shuffles, see Tables 6.4 and 6.5, thus these schemes are optimal, however it is not always the case of SSS. The simple version of SSS is optimal as well as P-SSS, but not the MD-SSS since for a MD-SSS we can obtain $(N_\Delta)!$ shuffles and $\forall\{a, b\} \in \mathbb{N}^2, (a > 2) \implies (a! \neq 2^b)$.

RP is able to generate all $N_b!$ possible permutations of the state (N_b is the number of bytes that have to be shuffled), but it is not optimal since it requires more than $log_2((N_b)!)$ random bits, the implementation proposed by Veyrat-Charvillon et al. [VMKS12] requires 64 bits of randomness.

Number of possible shuffles that can be generated by a scheme influences the security that it offers against side-channel analysis. Doubling the number of instants when an operation could be executed increases the amount of traces required for a successful attack roughly by a factor of four [MOP07]. Thus, in a perfect scenario, a shuffling algorithm that generates more different permutations offers more security (because there should be more possibilities of different operations being performed at a given moment in time), however it is not always true. It is very important to notice, that some particular cases of RSI and RS extensions do not always improve the strength of the countermeasure when more random bits are used. For example, in the simplest version of RS it can generate only two permutations (forward and reversed), thus we know that we have only two possible indexes at each moment in time. If one would use 4×4 M-RS with 4 random bits as suggested in Table 6.5 when rows are always handled in forward order while each row might be handled in

the forward or the reversed order, we still have only two possible indexes that might be used at each moment in time. Same reasoning applies in some other cases, thus not all versions of each scheme give a security increase when more random bits are used, for more details see Table 6.8. Nevertheless, when we increase the number of random bits in a scheme we always increase the number of different shuffles that could be generated. Thus, from this perspective, the security of a scheme increases i.e., when an attacker learns the position of one byte it gives him less information about positions of all other bytes. The rationale is that there are two slightly different ways of looking at a difficulty of an attack on a shuffling scheme: (1) attacking one byte or (2) attacking the entire state (all the bytes). The number of shuffles influences the second type of difficulty, but not necessarily the first one.

6.5.3 Resources

In addition to randomness, shuffling algorithms also require a certain additional memory and time. In order to support RP one needs to use an additional data structure (of the same size as the internal state of the algorithm, so its memory overhead is $O(N_b)$, where N_b is the size of the state in bytes). Depending on the algorithm and exact details of the implementation, RP might also require additional time overhead of $O(N_b)$ up to $O((N_b)^3)$ [BBD+14]. Scheduling (SchedAES) requires an additional datastructure that allows to track which operations are already performed on each part of the internal state, this datastructure effectively doubles the size of the AES implementation, the countermeasure slows down AES by the factor of 7 in case of AES-128 [Med12]. Our extensions of RS, RSI and SSS do not require as much memory, their memory overhead is limited to a couple of variables (generally to recompute and hold new indexes), in other words their memory overhead is $O(1)$. The only exception might be MD-SSS, where we need to store a small table of the size equal to the number of dimensions, which is always smaller than the size of the original internal state; in this case the memory overhead is $O(N_\Delta)$.

Shuffling countermeasure also results in a time overhead compared to a normal unprotected implementation. The exact time overhead might vary depending on the implementation and on the available hardware. We did several experiments on a ATmega328P 8-bit microcontroller, all our code was written in `C++`. The microcontroller used an external 16 MHz clock. We implemented some of the variations of shuffling techniques that were described in Section 6.4. We applied several techniques on the SubBytes operations of the first and the last round of AES-128. The only detail that changed between different implementations were the two calls to functions that implemented different versions of SubBytes. In order to measure the time we encrypted 10 000 random plaintexts with different random bits as inputs to our shuffling techniques. Table 6.9 presents our results including and excluding the time needed for the generation of random bits (for shuffling). The first and the last rounds used same

Table 6.8 – Min and max number of different SubBytes operations that might occur at a fixed moment in time i.e. the number of different bytes of the state that might be handled at a given moment in time during shuffling.

Technique	Random Bits	Operations		Total
		Min	Max	Shuffles
RP [VMKS12]	64	16	16	16!
RP●	45	16	16	16!
RSI	4	16	16	16
V-RSI	1	2	2	2
	2	4	4	4
	3	8	8	8
	4	16	16	16
M-RSI 4×4	1	2	2	2
	2	4	4	4
	3	8	8	8
	4	2	2	16
	4*	16	16	16
	5	4	4	32
	6	8	8	64
	8	4	4	256
	9	8	8	512
	10	16	16	1024
RS	1	2	2	2
M-RS 4×4	1	2	2	2
	2	4	4	4
	3	4	4	8
	4	2	2	16
	5	4	4	32
P1-SSS 4×4	1	1	2	2
P2-SSS 2×4	2	1	2	4
P4-SSS 2×2	4	1	2	16
MD-SSS	1	1	2	2
	3	1	3	6
	5	1	4	24

* Second version, recall Table 6.4.

● Theoretical lower bound on the number of necessary random bits, $\lceil \log_2 16! \rceil = 45$.

Table 6.9 – Execution time of 10 000 AES-128 encryptions with different shuffling techniques applied to the SubBytes operation of the first and the last rounds. Columns "Including RNG" and "Excluding RNG" give information including and excluding time for the generation of random numbers required for shuffling.

Algorithm	Random bits	Including RNG		Excluding RNG	
		Time (ms)	Overhead (%)	Time (ms)	Overhead (%)
No shuffling	0	13197	0.0	13197	0.0
RS	1	14500	9.9	13547	2.7
M-RS 4×4	1	14201	7.6	13246	0.3
	2	14438	9.4	13486	2.1
	3	14480	9.7	13528	2.5
	4	14362	8.8	13412	1.6
	5	14465	9.6	13514	2.4
SSS 4×4	1	14394	9.0	13441	1.8
V-RSI	1	14426	9.3	13473	2.1
	2	14426	9.3	13473	2.1
	3	14424	9.3	13473	2.1
	4	14423	9.3	13472	2.1
M-RSI 4×4	1	14186	7.5	13232	0.3
	2	14185	7.5	13232	0.3
	3	14322	8.5	13369	1.3
	4	14323	8.5	13372	1.3
	5	14425	9.3	13474	1.3
	6	14423	9.3	13475	1.3
	8	14319	8.5	13373	1.3
	9	14397	9.1	13452	1.9
	10	14398	9.1	13452	1.9

random bits for shuffling. We can see that most of the overhead comes from the RNG. All calculations (of indexes for memory accesses during shuffling) did not use conditional branching in any way dependent on the random bits used for the shuffling in order to prevent possible SPA and timing attacks (recall Sections 3.1.1 and 3.1.5).

We can see that the overhead is relatively small and reasonable, but not negligible. Different techniques give slightly different time overheads which give the ability to choose depending on timing constraints that are imposed on a system. These results might probably be improved by implementing other variants of our shuffling tech-

niques or by implementing them in the assembly code (while taking into account the architecture of the microcontroller).

6.5.4 Resistance against side-channel attacks

To analyse how the proposed countermeasures can resist against side-channel attacks compared to an unprotected device we implemented 17 variants of the proposed shuffling schemes and also an unprotected version of the algorithm. We analyse 3 different scenarios which represent 3 attackers of different strength in order to show the differences between presented shuffling schemes.

Unprofiled attack

Correlation Power Analysis (CPA) [CNK04, BCO04] is one of the most popular non-profiled DPA attacks. We have tested several versions of our shuffling schemes against CPA attacks. The CPA was conducted using the Hamming weight (HW) leakage model on simulated power traces (the attacker knows how the device leaks). We have implemented the following algorithm:

$$\forall i \in [0, 15], z_i = S(k_i \oplus p_i) \tag{6.8}$$

where lower index i gives one byte of a variable while z is the resulting 4×4 intermediate state, k is a 16 bytes fixed key and p is a 16 bytes plaintext. The application of the AES S-box S was shuffled using different shuffling schemes. In order to simulate power traces we used Silk tool with the following parameters: Hamming weight as the leakage function and the noise variance was set to 2. Same parameters were used for all simulations with different shuffling schemes.

Figure 6.22 shows the estimated number of traces that is necessary in order to extract the key with various shuffling techniques used as countermeasures. As expected, the number of traces increases with the number of different bytes that might be handled at a fix moment in time (due to shuffling). The success rate of this attack on each scheme is shown in Figure E.1 (Appendix E).

The same CPA was applied to all shuffling techniques. Among the presented shuffling schemes, there are several techniques that give the same advantage (resistance) against a classic versions of DPA-like attacks, but some of them may generate more different permutations in total (see Table 6.8) and should make implementation specific attacks more difficult (in the sense of attacking the entire state).

Implementation specific unprofiled attack

We used same simulation parameters and same algorithms in a more complex setting. We applied a CPA attack in a scenario when the attacker is aware of the shuffling scheme and knows the details of its implementation. We applied a preprocessing

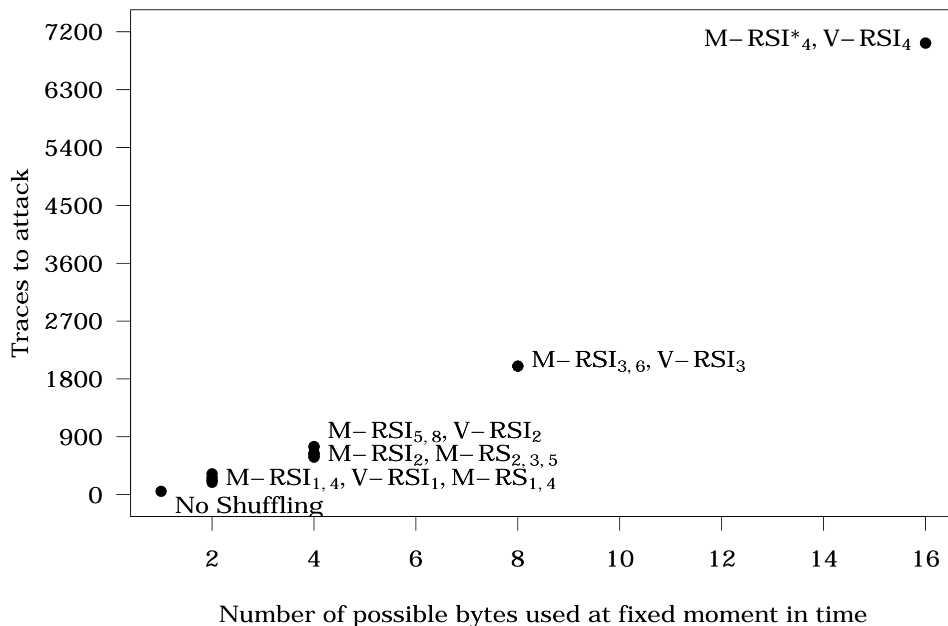


Figure 6.22 – Number of traces needed to extract the key using CPA on different implementations. Note the vertical scale.

technique called integration [MOP07, §8.2.3] to the simulated traces before applying the correlation power attack. In this scenario we sum all points which could be related to the attacked byte according to the shuffling scheme that is used, recall Figure 6.20 and Figures of the Appendix D. Thus, for a scheme where a byte can be handled in v different positions we integrate v points, for example we sum 4 points while attacking M-RSI 4×4 with 2 random bits, those points correspond to positions where byte 0 can potentially be handled (i.e., points that correspond to bytes 0, 1, 2 and 3 in a scenario without shuffling, see Figure 6.20a). To sum up, we suppose that the attacker knows exactly where a given byte can be handled in a power traces and thus the attacker analyses (preprocesses) only these points. In other words, this scenario corresponds to a more powerful attacker with more knowledge about the attacked cryptosystem.

Figure 6.23 shows the number of traces needed to successfully extract the secret (lowest number of traces where the success rate of the attack reaches the value 1). More details on some attacks are shown in Figure E.2 (Appendix E). We can see, that this technique is more powerful than a “simple” CPA (from the previous section) against all shuffling schemes (note that the vertical scale between Figures 6.22 and 6.23 is not the same). Nevertheless, our results show that when more bytes can

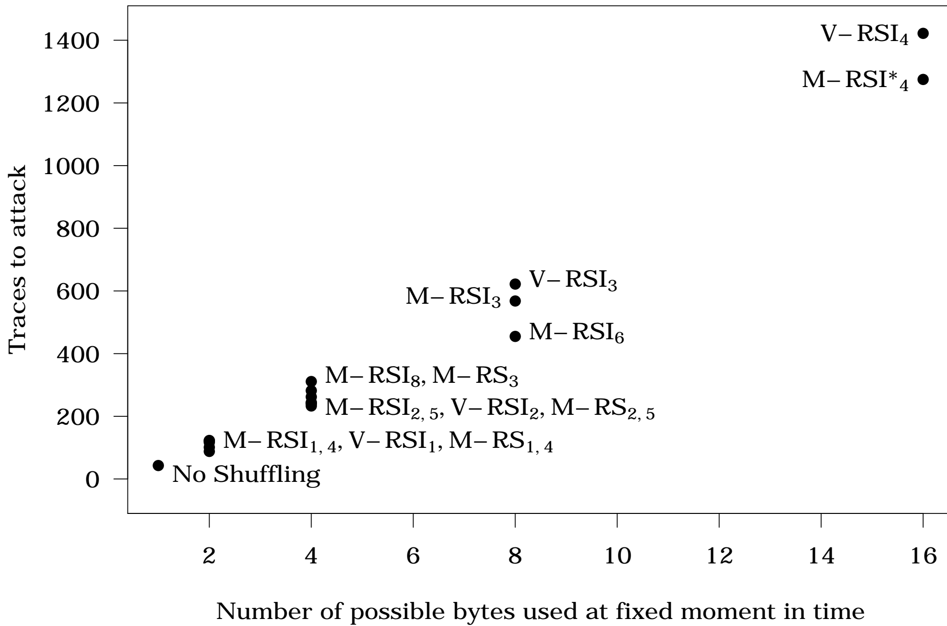


Figure 6.23 – Number of traces needed to extract the key using CPA with integration as a preprocessing technique. Note the vertical scale.

be found in a particular position (fixed moment in time during the execution of an algorithm), then the difficulty of this attack increases.

Profiled attack

We used a Template Attack (TA) [CRR02] in a scenario when an attacker has profiling capabilities and has the knowledge of the shuffling scheme (i.e., the adversary knows all points in time when a byte can be handled in the same way as in the unprofiled CPA with integration). However, in this scenario the attacker does not control the randomness during the profiling, which corresponds to a real case scenario (the randomness for cryptographic operations is generated inside of the device), thus an attacker does not know the full permutation (order of bytes during a single execution)[16].

For each target value (attacked byte) the template corresponds to an average and a covariance matrix. The complexity of the parameters' estimation for each of these templates depends on the number of points that have to be considered during an attack. The number of points in each attack depends on the number of points in time

[16]Even in a case when an attacker knows all random values, he might choose not to use them in order to speed up the profiling phase of the attack by building less templates.

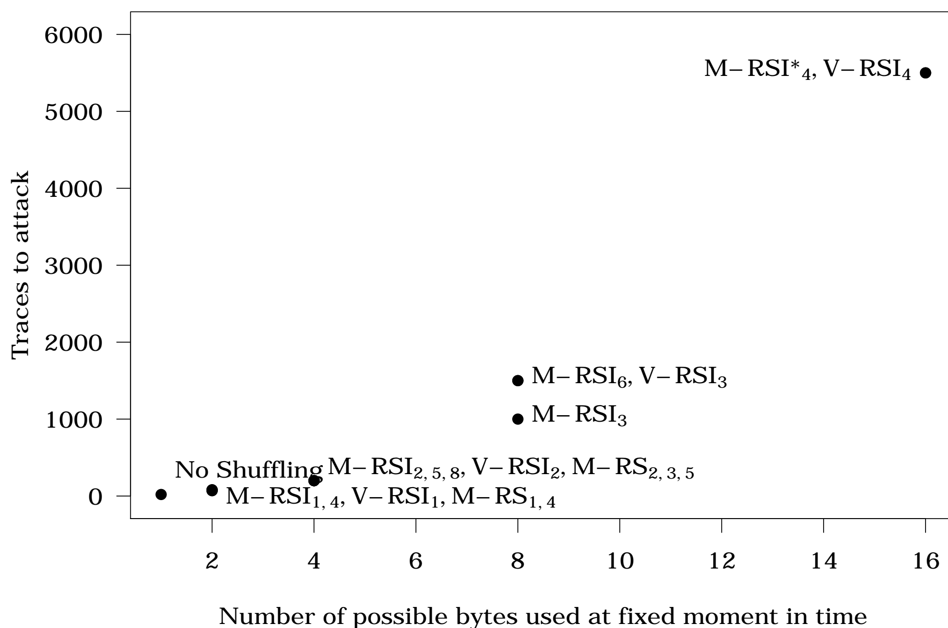


Figure 6.24 – Number of traces needed to extract the key using TA on different implementations. Note the vertical scale.

when a byte might be handled and thus the number of points depends on the shuffling scheme. We used 5 000 traces per target value in order to build all profiles.

The number of traces needed to extract the key with high probability is shown in Figure 6.24 (the success rate of each attack can be found in Figure E.3 in Appendix E). These results confirm that the success of an attack depends on the number of possible bytes that could be handled at the same moment in time (due to shuffling), which is also the case for two other CPA attacks. We can also note that the TA is better than the CPA with integration when the number of points used for the TA is low (i.e., when the number of positions where a given byte can be handled by the shuffling scheme is low or in other words when the number of possible bytes used at a fixed moment in time is low) e.g., see Figures 6.23 and 6.24 for 2 possible positions. However, the TA is less effective than the CPA with integration of points for schemes that result in permutations where a byte could be in many different positions (many points to consider in the TA), compare Figures 6.23 and 6.24 for 16 possible bytes used at a fixed moment. The advantage of the TA compared to the CPA with integration decreases when more points have to be taken into account due to the fact the TA suffers from the estimation error in high dimensionality contexts [LBM15a].

Targeting the RNG

Another implementation specific technique that an attacker might be using in order to attack these schemes could also be implemented. An attacker that knows the exact implementation of the shuffling countermeasure that was used can try to recover random bits used to shuffle the bytes and then extract the key using this knowledge (by finding the positions of shuffled operations using known random numbers). This technique was used to attack a masking scheme of a DPA Contest [LBM15b, LMBM13]. The idea of this attack it to target the random number generator which allows to effectively remove the security mechanism that uses randomness. Thus, all masking and shuffling schemes are vulnerable to attacks that can successfully target the random number generator.

Attacks on other shuffling schemes

Our results with all three attacks suggest that the difficulty of attacking a given shuffling scheme mostly comes from the number of positions where a given byte can be handled during an execution of the cryptographic algorithm. To be more precise, all schemes that can put a given byte at v positions require the same number of traces to extract the key for a given attack, see Figures 6.22, 6.23 and 6.24 where all points of the same column overlap or lie close to each other. Moreover, we can observe that the success rate of each attack on all schemes that result in putting a given byte to the same number of positions also closely follow each other, see Figures E.1, E.2 and E.3 in Appendix E.

Using these observations, we conclude that a given attack on any shuffling algorithm A_s will give the same performance that this same attack on a scheme A'_s that can shuffle a byte into the same number of positions as the scheme A_s. Thus, an attack on the first byte of the SSS is identical to attacking an unprotected implementation, while an attack on the second byte will perform as an attack on a byte of e.g., V-RSI with 1 random bit (see Table 6.8 and Figures 6.20d and D.2a). Same reasoning can be applied to P-SSS, MD-SSS as well as any other shuffling schemes. Thus, the RP scheme is as difficult to attack as M-RSI 4×4 with 10 bits or V-RSI with 4 random bits (see Table 6.8).

Nevertheless, it is important to note, that this reasoning holds if the RNG is not biased and if the implementation under attack does not have additional flaws that an attacker can exploit nor additional sources of information available to the attacker. This result can also vary in case if additional countermeasures are applied with a shuffling scheme. Another important thing to note is that these attacks are attacks on one byte, difficulty of extending an attack on one part of the key to the attack on the full state increases with the number of shuffles, as already discussed in Section 6.5.2.

6.5.5 Applications & modifications

It is easy to apply RSI, RS, SSS and their extensions to SubBytes or AddRoundKey operations of AES-128 since each of them operates only on one byte of the state at a time and does not have any precedence requirements inside of the operation. In order to apply these techniques to ShiftRows or MixColumns operations we may simply consider a row or a column as a memory unit (instead of a byte).

Same techniques might be adapted for 192-bit and 256-bit versions as well as for other operations of AES cipher by using more random bits to handle additional rows (for performing the shuffling during the round key derivation). RSI, RS, SSS and their extensions might be also applied to other algorithms. These techniques should be applicable if parts of the state are used independently from each other during some computations e.g., the application of the S-box in ciphers like DES [DES77] or PRESENT [BKL+07].

We can also combine RS, RSI, SSS and their extensions in order to obtain more different shuffles, e.g., RS might be used with 10-bit version of M-RSI on the AES-128 in order to get 2048 different shuffles by using 11 random bits.

Finally, it is worth noting that not all techniques presented here as well as their extensions are equally practical and are equally secure with a given number of random bits. Nevertheless, we considered that all versions with their extensions should be presented for the sake of completeness of this work. For example, SSS is not as practical as RSI extensions for security, optimality as well as penalty reasons; however, we think that SSS is a nice case study from the theoretical point of view.

6.5.6 Discussion

All of the shuffling schemes that we propose and describe are similar i.e., each one suggests a way of going through all indexes of the state in a particular order that could be easily implemented with a small overhead. Most of these techniques could be seen as extensions and generalizations of the RSI shuffling scheme.

Our scalable shuffling schemes can offer different number of shuffles and different number of positions (moments in time) where a particular byte is handled, overall it results in different levels of security. In order to choose which shuffling scheme to implement we advise the designers of a cryptosystem to consider the following criteria in given order:

- Number of available random bits
- Timing constraints
- Number of different operations that could be handled at a given moment in time

- Number of shuffles

The first criterion is related to the basic constraints of the system, so the designer should use as much randomness as possible in order to increase the security. The second criterion changes depending on the given hardware and implementation details, so it has to be tested for each platform individually, however, our results show that all shuffling schemes that we present result in very similar timing overheads. Results of all our attacks suggest that a scheme which generates shuffles such that higher number of different bytes that could potentially be handled at a fixed moment in time (from the beginning of the execution) increases the difficulty of an attack. Thus a designer of a cryptosystem should choose a shuffling algorithm that maximizes this number. Finally, the number of different shuffles that a given shuffling scheme can produce does not influence the number of traces that is needed in order to mount a successful attack. However, mounting some profiled attacks becomes increasingly difficult when the number of shuffles grows since an attacker would have to create a model per shuffle [BGH+16]. This parameter can also help in case if the attacker can find out a position of one byte in order to reduce the remaining uncertainty on the positions of other bytes.

Our results based on side-channel analysis using 3 different types of attackers (of increasing strength and capabilities) show that the number of different operations that might occur at a given moment in time produces the biggest effect on the success rate of an attack. This result hold even for attacks that take into account the implemented scheme. From this perspective, the SSS scheme presents a disadvantage because it does not shuffle all bytes, some of the bytes always remain at a fixed position in a trace. However, SSS could still be interesting in practice, because it could be implemented using only a couple of additional instructions on many hardware platforms (without taking into account the random number generator) e.g., conditional swap instruction (MSWAPF) available on TMS320x2803x or using compare-and-exchange (CMPXCHG) that is available on many Intel and AMD processors. SSS could also be easily combined with other shuffling schemes thus giving a boost to the security of the system.

A specific type of attack could also be mounted against all of the presented shuffling schemes. If an attacker targets the random number generator in order to find out the ordering that is generated by the shuffling scheme, they can effectively remove the protection given by the countermeasure. This type of attack could be mounted on any shuffling or masking countermeasures [LBM15b]. Thus, algorithmic countermeasures that rely on randomness require a secure random number generator that could not be easily targeted through side-channel analysis.

Most importantly, from the point of view of side-channel analysis, we were able to study 18 different implementations (including the unprotected version) using 3 different attacks. In case of a real experiment, we would have to create 18 setups and

perform 18 acquisitions. Using a simulator we were able to run all these experiments in parallel (at the same time, using a cluster of a computing center), which allowed us to substantially reduce the amount of time necessary for these experiments. Moreover, we were able to generate all the data on the fly (stored in RAM), without using disk space for storage and cut all the timing costs related to the data transfer (from the lab to the servers that were running the analysis). Based on previous experiments that we were running in our lab, we know that acquiring 10 000 traces takes about 4.5 hours using our setup (it includes the time needed for the setup, data transfer, verifications of the dataset and the actual acquisition). We used 200 repetitions to compute the success rate of CPA attacks and computed their success rates up to 8 000 traces (while using a different set of traces in each repetition). It means that analysis of one shuffling scheme requires at least 1 600 000 traces. Every acquisition would have to be performed 18 times in case of real experiments, thus we would have to acquire 28 800 000 traces in total. Even if we assume that we need 3 hours for the acquisition of 10 000 traces (recall the DPA Contest setup mentioned in Section 5.1) and neglect the time spent on the physical setup, the data compression and transfer, the acquisitions would still require 8 640 hours (360 days!). In other words, if we were to acquire these traces using a real setup instead of a simulation we would need almost *a year* of non-stop (24 hours a day!) acquisitions just to get the power traces. If we assume that our setup can be improved in terms of speed using better software, though the choice of parameters of the oscilloscope or by using several hardware setups in parallel. Then, even if a team from another lab could manage to do all the acquisitions 10 times faster, they still can save more than a month worth of time by using a simulator. The rationale is that using a simulator does save a lot of time and it is reasonably the only way of performing massive comparative studies.

6.6 Summary

We showed that a generic high-level of abstraction simulator that can generate traces for side-channel analysis can be built by overloading operators in a language like C++. The described tool, called SILK, uses a very simple way of simulating the data dependant part of the power consumption and can also add some noise to the simulation. Since SILK does not simulate the operation dependant part of power consumption it cannot suite for accurate estimation of power consumption of a specific hardware. Moreover, at first stages of development of new algorithms (whether it is a novel idea for an attack, a countermeasure or another algorithm related to side-channel analysis) the hardware specifications are usually unknown. As the result, SILK is not suited for detecting implementation specific issues and flaws in a final product. This tool should only be used for preliminary analysis and comparative studies.

We would like to emphasise, that many studies already used same general methods for their simulations, however they did not use the same tool across all analysis: disposable home-brewed pieces of code for one-time analysis tend to be used by labs. Moreover, some details about simulations cannot be found in the published papers and each lab might implement the actual simulation slightly differently, even though main principles of generating values that represent points of power traces stay the same. Up until SILK was presented all simulation tools for side-channel analysis were not publicly available and free, which is the reason why researchers did not use the same tool for their analysis (thus comparing two studies that use simulations can be tricky since simulations are not done in exactly the same way).

This part of our work merely tries to suggest an opensource automated tool that can be used for simulations that can be done using very high-level of abstraction techniques. From this point of view, SILK is similar to another high-level of abstraction simulator presented by Reparaz [Rep16b]. We showed that it can be used to compare different algorithms and parts of algorithms which ultimately lead us towards a way that can be used to improve S-boxes. We also showed that SILK can be used to compare countermeasures on the example of shuffling schemes. We stress out, that in both cases we were able to compare many algorithms which is very difficult to do using real power traces due to time constraints, simulating the traces allowed us to save several months worth of data acquisition time. On the example of several S-boxes we were able to show that our simulations fit real experiments relatively well. However, we stress out that one simulation performed for one particular algorithm is not perfect, this type of simulations fit the reality in terms of relationships between several simulations i.e., the same attack can require a different number of traces between real and simulated experiment, but several attacks on real and simulated traces will behave similarly with respect to each other in both situations. Thus, the suggested approach works well for comparative studies and allows to estimate an increase in performance of one algorithm compared to another one e.g., algorithm implemented

with and without a countermeasure.

Even with respect to high-level of abstraction simulations SILK has its limitations. SILK cannot easily simulate glitches and some other effects that are present in real devices and measurements setups. Using some additional engineering efforts, SILK can be modified and improved. For example, it can be upgraded by adding effects related to the clock jitter: a clock is not perfectly stable in terms of its frequency, it means that some clock cycles can be slightly longer than others. Thus, an oscilloscope, recording measurements at a much higher frequency than the device operates will record more points corresponding to longer clock cycles. As a result recorded traces will not be aligned at all points which does make the side-channel analysis harder[17]. Another way to improve SILK could be done by adding an equivalent of an analog-to-digital conversion of simulated traces which also happens with real measurements in a digital oscilloscope. The voltage measured by an oscilloscope is a real number, however a digital oscilloscope can only record a measurement in binary using a finite amount of memory, thus it can not record measurement with infinite precision. Therefore, a measurement is rounded to the nearest number that the oscilloscope can represent. Generally, the error (difference between the real voltage and the recorded voltage) is small compared to the value that is recorded. However, in Differential Power Analysis we are specifically interested in small variations that are present in power traces. SILK uses a double precision 64-bit to represent real number (IEEE 754 standard) for each point of a trace, which has a higher precision than representations used by modern oscilloscopes. Moreover, the precision varies in between different scopes. These differences in the precision of measurements can have a small but measurable influence on the outcome of side-channel attacks. Thus, preprocessing simulated results to make them reflect the reality of analog-to-digital conversion with loss of precision can be beneficial for the accuracy of subsequent side-channel analysis.

One of the main questions that remain open for high-level of abstraction simulations is the method of choosing best parameters for a simulation. On one hand, since it is a very abstract model that is not tight to a specific device, use of generic ODL and MTL models (such as HW and HD) could be sufficient. On the other hand, using profiling of real devices might be beneficial to create improved simulations that reflect the reality better. Since SILK is the first opensource simulator of this kind we cannot compare it to another one, thus right now it is impossible to say whether there is a better way of simulating power traces for side-channel analysis.

[17]Irregular clocks are actually used as a countermeasure against side-channel analysis, recall Section 3.4.2

Chapter 7

ASCOLD: masking implementation checker

The simulator and the analysis presented in this section are based on the paper *"Mind the Gap: Towards Secure 1st order Masking in Software"* presented in Paris (France) at the 8th International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE) in 2017 [PV17].

Masking is one of the most commonly used countermeasures against side-channel attacks. However, implementing a masking in a real device is far from being a trivial task even in case of a sound masking schemes. It is mainly due to the violations of the ILA and differences between value-based and distance-based leakage models that are also related to the notion of order-reduction (ODL vs. MTL models, recall Sections 3.2.3 and 3.4.1).

This part of our work has two main goals: (1) bridging the gap between theory and practical use of masking through the analysis of ILA-breaching effects and (2) providing a tool that can detect hazardous instructions that lead to a decrease in security of masking implementations due to ILA-breaching.

Through our analysis we focus on an Alf and Vegard's RISC (Reduced Instruction Set Computer) processor (AVR) microcontroller ATmega163 in order to analyse how and under which conditions an implementation can violate the ILA. Subsequently we establish solutions that mitigate these issues. Then we build an assembly-oriented tool that can detect ILA violations in AVR-based masked implementations. And finally, we use all our findings and analysis to create a 1st order masking implementation which enforces the ILA.

7.1 Acquisition setup and evaluation

For all our experiments we use a smart card equipped with an 8-bit AVR microcontroller ATmega163[1]. The device uses a 4.4 MHz clock, 1024 bytes of Static RAM (SRAM) and 17 Kbytes of Flash memory. The acquisition of power traces is performed using the Riscure PowerTracer[2] and the Picoscope 5203 oscilloscope. The sampling rate of the oscilloscope is set to 31.5 MSamples/s, the only post-processing applied is the alignment of power traces (using the software by Riscure).

We use two statistical analysis techniques for the evaluations of our sets of traces. We use the t-test as a detection tool with respect to the ILA-breaching effects and also to test solutions proposed to enforce the ILA. In addition to t-tests, we also employ the 1st order CPA methods [BCO04] to show the exploitability of the found ILA-breaching effects. In order to reduce the computational cost of the evaluation we use the memoryless formulas suggested by Schneider *et al.* [SM15] for t-tests and the incremental approach for CPA by Botinelli *et al.* [BB15].

7.2 ILA-Breaching Effects

Here below we present three ILA-breaching effects that were identified in the ATmega163 microcontroller all these effects are hazardous to the security of any masking scheme. These findings demonstrate that independent computations *do not* necessarily lead to independent leakages and thus, the ILA does not hold and the order-reduction (recall Section 3.4.1) can become applicable.

We assume that an intermediate 4-bit key-dependant value z is masked using a first order masking scheme and the two shares x_0 and x_1 combined give the value z in the following way:

$$z = x_0 \oplus x_1. \tag{7.1}$$

In our experiments the shares x_0 and x_1 are always manipulated in a theoretically sound manner, adhering to the masking scheme's requirements i.e., we never combine the shares directly using an exclusive-or instruction in the following way: `EOR x0, x1`. Basically, shares x_0 and x_1 are never used together as operands of an instruction in our experiments. Since a good way of ensuring that a microcontroller will not accidentally combine secret shares is to use a lower level programming language we focus on an assembly-based scenario when the developer fully specifies the control flow of a program (compiler is not allowed to rearrange instructions for optimizations).

For all the described theoretically sound scenarios we show experimentally that ILA is not fulfilled by employing 1st order univariate analysis. More specifically, we

[1] `http://www.atmel.com/images/doc1142.pdf`

[2] `https://www.riscure.com/security-tools/hardware/power-tracer`

perform correlation-based analysis [BCO04], computing the correlation coefficient ρ between the Hamming weight of the sensitive value and the experimentally acquired set of traces. Moreover, to maintain a wide attack scope (and not limit ourself to CPA), we also use the leakage detection methodology [CDG+13, SM15] and compute the 1st order random vs. fixed t-test (Equation 4.2)[3]. We conclude every scenario by suggesting possible solutions that enforce ILA and checking that this solution does help.

The ILA-breaching effects can manifest in several data storage units such as registers, SRAM cells, I/O buffers, etc. These effects may be related to different instructions of the AVR ISA[4] leading to a very large number of potential scenarios. In order to maintain a feasible scope, we limit our analysis and discussion to storage units and instructions that are often encountered in the context of cryptographic implementations. More specifically we analyse registers and SRAM memory accesses as well as the use of logical instructions (e.g., exclusive-or).

7.2.1 Overwrite effect

The overwrite effect is observable when a share gets overwritten by a different share of the same secret value. For instance, if share x_0 in a data storage unit (e.g., register or memory cell) is overwritten by share x_1, then the power consumption correlates with the number of flipped bits (i.e., $x_0 \oplus x_1$) or in other words it correlated with the Hamming distance between x_0 and x_1. Even though this effect was observed by Daemen *et al.* [BDPA09] and later revisited by Coron *et al.* [CGP+12] we include it in our experiments for the sake of completeness and in order to show all the effects on the same platform.

Here we address the most common situations in which overwriting happens. We perform two experiments: a register-based overwrite via the register-to-register move instruction `MOV`, and a memory-based overwrite via the memory-to-register store instruction `ST`. The assembly source code that we used in our experiments are given in Listings 7.1 and 7.2[5].

[3] Note that we perform both the t-test and the attack to show that the leakage detected by the t-test can be exploited.

[4] `http://www.atmel.com/images/Atmel-0856-AVR-Instruction-Set-Manual.pdf`

[5] Register `X` is a pointer register that is used for memory addressing, it is composed of registers `R26` (lower part) and `R27` (high part).

```
; x0 in R17
; x1 in R23
MOV R17, R23
;
;
```

Listing 7.1 – Register overwrite experiment.

```
; x0 in SRAM at adr 0x0080
; x1 in R17
LDI R27, 0x00
LDI R26, 0x80
ST X, R17
```

Listing 7.2 – Memory overwrite experiment.

Figure 7.1 shows the results of our correlation analysis and of the t-test for both overwriting experiments. Our experiments confirm that overwriting indeed produces an ILA-breaching effect and that it manifests in registers as well as in SRAM. Looking at our figures we can notice that the exploitability of the effect varies depending on to the data storage unit. More specifically, there is an order of magnitude in between the amount of traces required to exploit the overwrite effect in SRAM compared to the same effect in registers.

Preventing register and memory-based overwrites is relatively straightforward: the corresponding register (or memory cell) needs to be cleared before proceeding with the next value; another way of preventing this effect would be simply not using the same memory unit to handle different secret shares.

7.2.2 Memory remnant effect

The memory remnant effect is a leakage originating from consecutive SRAM accesses to different shares of the same secret value. Assume that shares x_0 and x_1 are stored in SRAM cells and that they are accessed sequentially. Naturally, the first access leaks share x_0 (value-based leakage), yet it also creates a *remnant* of x_0. A remnant is the information about a previously used (transferred, stored or handled) value, it can remain e.g., in values of wires of a bus or in a hidden register which is not documented by a manufacturer. The second access will leak the transition of the share x_1 and the remnant x_0, therefore reducing the security.

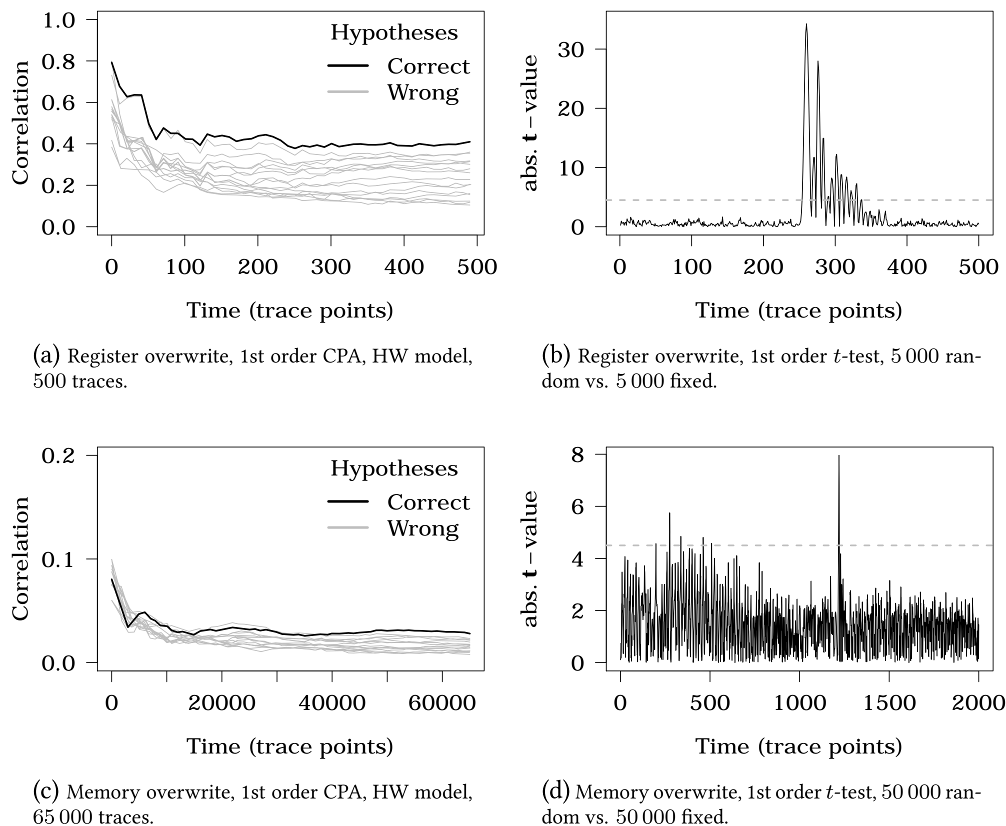
```
;share x0 at adr 0x0080
;share x1 at adr 0x0090
LDI R27, 0x00
LDI R26, 0x80
LD R17, X
LDI R27, 0x00
LDI R26, 0x90
LD R20, X
;
;
```

Listing 7.3 – Memory remnant experiment.

```
;share x0 at adr 0x0080
;share x1 at adr 0x0090
LDI R27, 0x00
LDI R26, 0x80
LD R17, X
LDI R17, 0x00
LDI R26, 0x85
LD R17, X
LDI R26, 0x90
LD R20, X
```

Listing 7.4 – Clearing remnant experiment.

We perform two experiments in order to analyse the memory remnant effect and



(a) Register overwrite, 1st order CPA, HW model, 500 traces.

(b) Register overwrite, 1st order t-test, 5 000 random vs. 5 000 fixed.

(c) Memory overwrite, 1st order CPA, HW model, 65 000 traces.

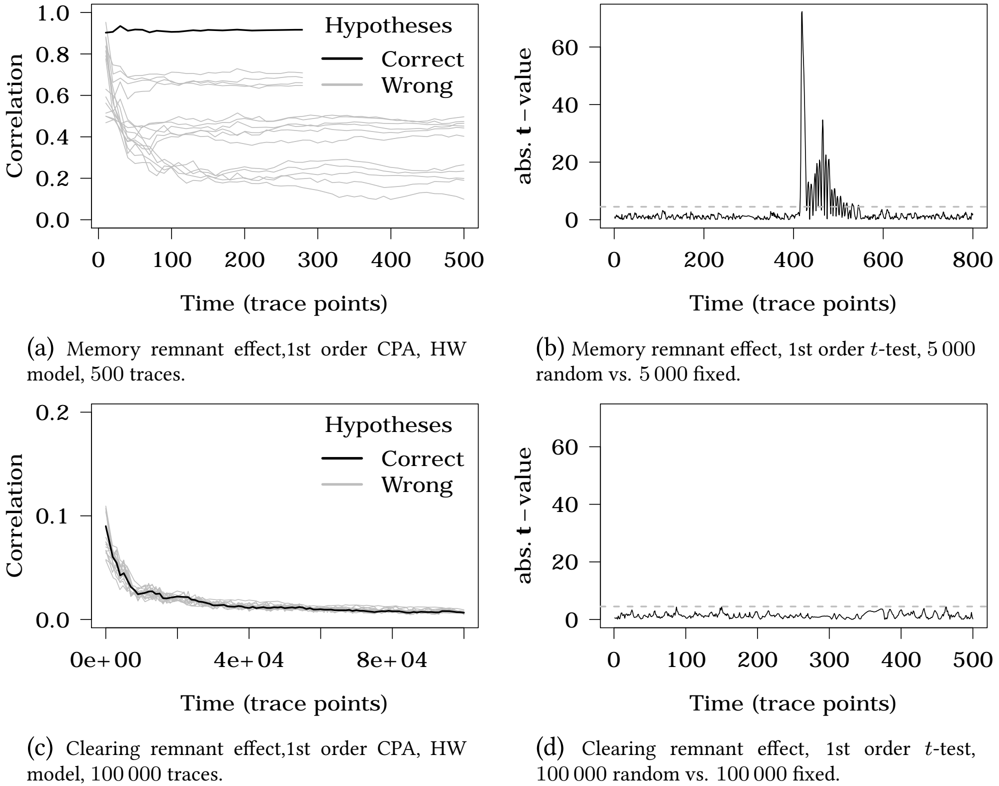
(d) Memory overwrite, 1st order t-test, 50 000 random vs. 50 000 fixed.

Figure 7.1 – Analysis of register and memory-based overwrite effects.

its impact on security. The code of two scenarios from our experiments is shown in Listings 7.3 and 7.4. The first scenario accesses two secret shares one after another (lines 5 and 8 of Listing 7.3). In the second scenario, presented in Listing 7.4 we load an unrelated address of SRAM into the pointer register (line 7) and access it (line 8) in order to clear the register and remove the remnant of the previous secret share.

Figure 7.2 shows the results of our analysis in both scenarios. In Figures 7.2a and 7.2b we can see that consecutive SRAM accesses can potentially lead to ILA violations, in other words consecutive memory accesses leak a combination of values that were accessed. Exploiting the memory remnant effect in a univariate manner in ATmega163 requires less than 500 traces with our setup. Preventing an attacker from exploiting the effect can be achieved by insertion of a dummy SRAM access. Alternatively, the software developer can ensure that different shares of the same secret value are not accessed sequentially. Note that the memory store instruction ST also produces a similar effect (shown here with load instruction LD).

We speculate that the memory remnant effect is probably caused by the structure



(a) Memory remnant effect,1st order CPA, HW model, 500 traces.

(b) Memory remnant effect, 1st order t-test, 5 000 random vs. 5 000 fixed.

(c) Clearing remnant effect,1st order CPA, HW model, 100 000 traces.

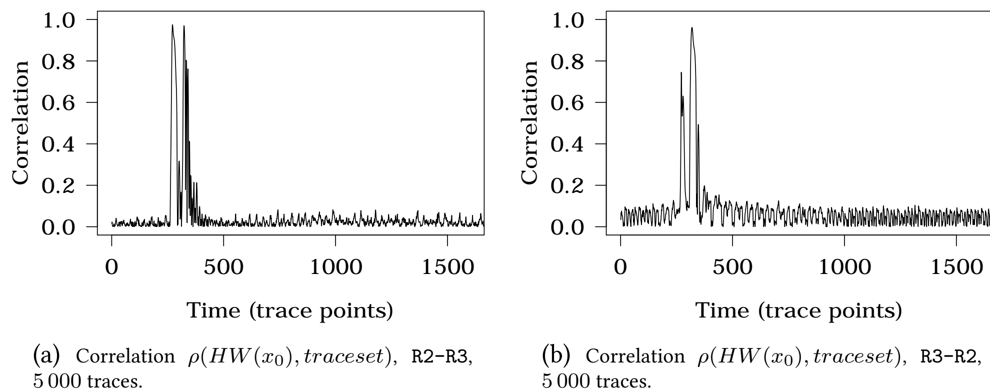
(d) Clearing remnant effect, 1st order t-test, 100 000 random vs. 100 000 fixed.

Figure 7.2 – Analysis of memory-based remnant effect.

of the the memory access mechanism. Potentially it can be caused by the pipelining stages (because several values being processed at the same time). Interestingly, the authors of the ELMO simulator [DMO16] found that profiling triplets of sequential instructions gives better (more accurate) profiles compared to profiles of single instructions. However, we must note that ARM Cortex-M0 that they have profiled uses a pipeline with 3 stages, while ATmega163 used in our experiments has only 2 stages of pipelining. However, since in our experiments the load instructions are 2 instructions apart (Listing 7.3, lines 5 and 8), we can safely assume that pipeline is not what causing ILA violation in our experiments. Therefore, in our case the ILA violations likely happen because of the architecture of the memory access mechanisms.

7.2.3 Neighbour leakage effect

The neighbour leakage effect implies that accessing or processing the contents of a data storage unit will produce a leakage that is also related to another unit. For example, let us assume that share x_0 is stored in register `Ra` and share x_1 is being



(a) Correlation $\rho(HW(x_0), traceset)$, R2-R3, 5 000 traces.

(b) Correlation $\rho(HW(x_0), traceset)$, R3-R2, 5 000 traces.

Figure 7.3 – Analysis of neighbour-based leakage effect.

processed in register Rb. Assume also that the registers Ra and Rb are subject to the neighbour leakage effect. Processing Ra will produce a value-based leakage of x_0. At the same time, the neighbouring leakage effect will cause Rb to leak the value of x_1 producing a combined leakage of both shares which happens at the same moment in time (related to the same point of a power trace). As a result, the ILA does not hold and the attacker will likely be able to recover the sensitive intermediate value z by analysing only one point of power traces.

Listing 7.5 shows the assembly code that we use in our experiments related to the neighbouring effect between the registers. During each experiment all registers were cleared (set to 0) with the exception of one register that was analysed, this studied register contained a sensitive value. We have performed such experiments on the neighbouring effects using all 32 registers but for the sake of space and without loss of generality we present the results related to registers R2 and R3 here below. Results on all register are presented in the table of "neighbours" in the Appendix F (Table F.1).

```
1  ; clear all registers
2  ; put sensitive value is in the selected register
3  MOV R0, R0
4  NOP ; 5 nops in the actual code
5  MOV R1, R1
6  NOP ; 5 times
7  MOV R2, R2
8  NOP ; 5 times
9  MOV R3, R3
10 NOP ; 5 times
11 ...
12 MOV R31, R31
```

Listing 7.5 – Neighbour leakage experiment for registers.

Figure 7.3a shows the results of our experiments with registers R2. Only R2 contains a sensitive value, thus only the line 7 of Listing 7.5 should leak. However we can observe a second peak at the moment in time when R3 is manipulated. Note, that this effect is symmetrical in ATmega163: when the sensitive value is stored in R3 (only line 9 should leak) we can also observe two peaks corresponding to moments when R2 and R3 are used (Figure 7.3b). As a result, we have identified a pair of data storage units (R2 and R3) that exhibit the neighbour leakage effect. During our experiments we observed that this issue mostly affects consecutive registers (with consecutive indexes such as 2 and 3, 4 and 5, etc.). However, we also identified that there are exceptions to this rule e.g., register R0 (see Table F.1).

We also observed that the neighbouring leakage effect is persistent, i.e. the MOV instructions will trigger the same behaviour even if performed in a different order (not necessarily as presented in Listing 7.5). We did not identify a similar effect in the SRAM, but our experiments were limited to a small region of cells (testing all cells would require a huge amount of time since there are much more addresses in SRAM than the number of registers).

Neighbour-like effects have been observed in consecutive registers, but the question whether they are caused by physical proximity or they stem from other effects remains open. Neighbouring leakage effects were observed by De Cnudde *et al.* [CBG+17][6] in case of FPGAs, since they could change the physical configuration of the implementation (i.e., assign functionalities to different cells in the FPGA) they were able to show that the effect comes from the physical proximity in their case. Since we worked on a microcontroller (the structure and configuration is fixed) we cannot perform the same experiment to check this hypothesis. However, given the pairwise manifestation of the neighbouring effect (the whole neighbouring seems to have a pair-based configuration) we speculate that they relate to the structure of the register file and likely involve the storage and multiplexing mechanisms related to the activation of registers. This hypothesis can be tested by looking at the structure of the microcontroller. Unfortunately manufacturers do not make this data available. However, it can be done using optical inspection and other reverse-engineering techniques, nevertheless this approach remains rather expensive (for small labs in academia) nowadays. Note that it is hard to link architectural options at the hardware layer directly to side-channel effects. As a solution to the neighbour effect the developer can opt to avoid storing shares in hazardous registers. Alternatively, they can store all shares in SRAM, except for the ones currently in use.

[6] Their findings were actually presented at the same COSADE workshop where we presented our original paper upon which this chapter is based on.

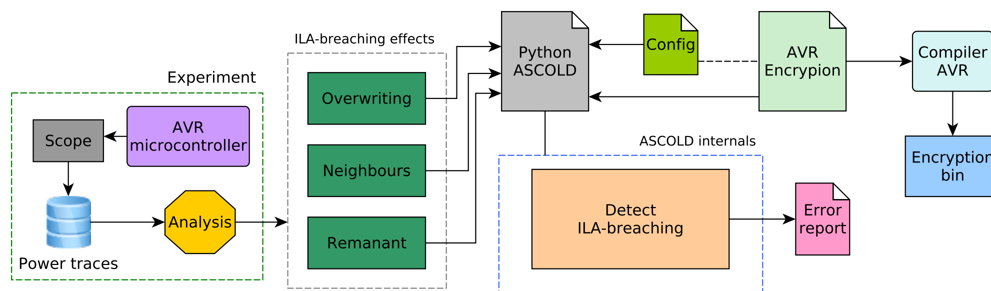


Figure 7.4 – Scheme of the workflow using AscoLD.

7.3 Description of the tool

The ILA-breaching effects that we identify can be noticed by observing the assembly source code, but it can be only done with the knowledge resulting from an experimental evaluation i.e., after the analysis that we have already performed. Thus, in order to assess the security of implementations at the assembly level, we developed a tool called AscoLD, standing for Assembly Code Leakage Detection tool. AscoLD integrates the rules related to undesirable ILA-beaching effects and allows to detect these issues by analysing the assembly source code. The tool is written in `python` and the source code is available in our `git` repository[7]. AscoLD uses assembly code as its input in order to run a simulation while checking for potential issues that can cause side-channel information leakage (due to the architecture dependent ILA-breaching effects). Figure 7.4 shows the general workflow using AscoLD tool. The tool is compatible with assembly code used by AVR, the assembly code can be used directly (if it is used for the development) or extracted from the compiled binary file (using a disassembler). Thus, it is possible to be sure that the executed code will be exactly the same as the code which is analysed by AscoLD. Otherwise, it becomes impossible to provide any guarantees on the quality of the analysis i.e., be sure that no additional issues are introduced during compilation. Note that differences between the analysed code and the final (executed) code is exactly the issue that is present in some tools e.g., the ones created by Barthe *et al.* [BBD+15] (analysis is based on `EasyCrypt` code) and by Reparaz [Rep16a] (uses high level source code that is transformed by a compiler).

In addition to the assembly source code, AscoLD requires a configuration file which is a `YAML`[8] text file. This configuration file basically specifies the nature of

[7] `https://github.com/nikita-veshchikov/ascold`
[8] A human-readable data serialization language often used for configuration files.

the state at the beginning of the program. More specifically, one has to provide the list of variables (and possibly addresses and registers) that correspond to random values, to secret shares (used during masking) and to constants. This type of information carries the "intent" or a purpose of a variable, which is impossible to find reliably in an automated manner, therefore the developer (who actually knows what they are doing) has to provide this information in addition to the developed code. A toy example of code and corresponding configuration file are shown in the Appendix F. The simulation run by the tool does not use an instance of an execution i.e., we do not use specific values such as a fixed key or plaintext in order to run the program. Ascold starts a program in an initial state and propagates all changes such as combinations of values, their modifications and replacements of one value by another. More precisely, it keeps track of which shares or combinations of shares (and possibly random values) are stored in each register or SRAM.

During any arithmetic or logical operation, shares stored in different operands are verified. Specifically we check whether we combine different shares of the same secret value without randomizing beforehand. In the same way, we verify the implementation for the device-specific distance-based leakages for every arithmetic and logical operation as well as for SRAM store and load instructions that are executed. Analytically, we verify whether the previously stored value and the new value cause the overwrite effect (recall Section 7.2.1). Similarly, our tool checks the load and store instructions for remnant effects discussed in Section 7.2.2. Ascold also uses the *matrix of neighbours* (see Table F.1) which allows us to check whether secret shares of the same value are stored in registers that were found to be neighbours. Ultimately it allows Ascold to detect neighbour leakage effect (recall Section 7.2.3).

As the result of the simulation, Ascold prints out a line number and the rule that was violated by the program. The exact instruction and the line number allow the developer to easily spot the problem. It also specifies which secret shares were combined. Note that not all combinations are hazardous, yet we opt for a "safety" approach in order to speed-up the verification process. Therefore, some messages are labelled as *errors* while other messages are labelled as *warnings*.

Ascold works with the AVR family of microcontrollers, it implements the most common memory instructions such as load and store as well as a set of commonly used (in cryptography) instructions such as arithmetic operations (`ADD`, `MUL`, . . .) and logical operations (`AND`, `EOR`, `OR`, . . .). The same core principles can be applied in order to build a similar tool for a different instruction set or to add new AVR instructions supported by newer microcontrollers.

The scope of Ascold tool is limited to the results of our findings. The current version of our tool incorporates our findings which are based on the ATmega163, other models of microcontrollers might have slightly different (even additional) issues that cause unintentional information leakage which result in violations of the ILA. Among other things, the main difference between different models of microcontrollers can

be visible in the structure of the matrix of neighbours (especially if a microcontroller has a different number of registers than ATmega163). Ascold does not take into account the effects of pipelining which might be an issue in case of a microcontroller which can potentially handle two different shares of the same sensitive value (at different stages of the pipeline) during the same clock cycle. This issue does not affect ATmega163 since it has two stages of pipeline: while one instruction is executed the next one is fetched. Thus, two secret shares cannot be manipulated at the same time in the pipeline. Ascold does not implement all AVR instructions, most importantly the current version of Ascold does not support branching instructions (which still allows to check one round of a block cipher or any implementation with unrolled loops). However, we implemented a set of the most commonly used instructions which includes load and store instructions as well as logical and arithmetic instructions. New instructions and rules (checking ILA-breaching effects) can be added with some additional software-engineering efforts which we would like to encourage and simplify by providing the full source code of Ascold in our `git` repository.

7.4 1st order masked S-box for Rectangle cipher

We have discussed the ILA-breaching effects in Section 7.2 and integrated these observations in the Ascold tool. It allowed us to build a masked implementation that takes into account the ILA-breaching effects and avoids leakage related to these issues. As a proof of concept we choose to implement a 1st order masked S-box of Rectangle by using the Ishai-Sahai-Wagner (ISW) [ISW03] scheme. Our aim is to produce an assembly-based, lightweight S-box implementation that is secure against 1st order univariate attacks. It will force the attacker to resort to a 2nd order approach or to multivariate techniques.

We implemented Rectangle in the AVR assembly language to be executed on ATmega163 which we used for all our tests. During the implementation we were already aware of all the ILA-breaching issues, in other words we knew what should be done in order to develop a secure piece of software the enforces the ILA. Nevertheless, the number of different focal points that has to be kept in mind during the implementation is too big to be sure that we are not forgetting about a leakage scenario. Note, that every issue that we described earlier has to be dealt with for *every* instruction. Thus, keeping in mind the entire implementation including assignments of mask shares to registers and all potential leakages is very challenging. Therefore even knowing about ILA-breaching does not prevent developers from errors. Thus, developers should definitely resort to the use of automated tools like Ascold to test their code while it is being developed. The process of using Ascold resembles the use of a compiler: once the code is written the developer can launch our tool which in its turn outputs line numbers with comments on the ILA-breaching if it is detected. Finally, when Ascold does not output any more errors the developer can compile the

code to get the executable file.

In the following text we will say that a masking implementation is *naive* if it does not take precautions against ILA-breaching effects and we will say that the implementation is *hardened* otherwise.

Our implementation uses a *bit-sliced* [DGV93a, Bih97] representation, due to both the bit-sliced structure of RECTANGLE and due to the $GF(2)$-oriented nature of the ISW countermeasure. A bit-sliced implementation is a software implementation technique often used in cryptography. It uses an n-bit processor as a set of n 1-bit execution units operating in Same Instruction Multiple Data (SIMD) mode. We employ a bit-slicing factor of 2 i.e., we exploit the 8-bit AVR architecture in order to process two 4×4 S-boxes in parallel (nibble-slicing). In other words, the S-box is decomposed into $GF(2)$ operations that can be accelerated by via SIMD instructions, which are in our case 8-bit assembly instructions. We use the decomposition suggested by Zhang *et al.* [ZBL$^+$15] which is optimal with respect to $GF(2)$ multiplicative complexity, since Grosso *et al.* [GLSV14] established that the minimum number of nonlinear operations required by 4×4 S-boxes is 4.

In order to create hardened implementations we use the solutions that we suggest in Section 7.2. We create two hardened implementations that we call *efficient* and *conservative*. In the efficient approach, after processing any share we clear the registers using an on-demand technique (only when the register is needed) and insert dummy load instructions (`LD`) to avoid overwrite and remnant effects. We avoid the neighbouring leakage effects by always storing the shares in SRAM i.e., the registers contain only the shares used by the current instruction. In the conservative approach we perform all the mentioned clearing techniques but in addition, we insert dummy store instructions (`ST`) and perform thorough register and memory clearing. Both efficient and conservative approaches are applied to every single instruction of the implementation i.e., the cost (number of cycles) is linear with respect to the number of instructions that manipulate masked shares.

In addition to the two hardened implementations we created three non-hardened implementations. One of them is an unprotected implementation (no masking). The two others are naive implementations that use the same masking technique. The first naive implementation uses a 1st order masking and the second uses a 2nd order masking.

Table 7.1 shows a summary of the costs for all 5 implementations: the time overhead as well as the requirements in the amount of random numbers. We can immediately notice that the resulting computational overhead of hardened implementations is quite significant, they are about 12 (efficient) and 15 (conservative) times slower than the naive 1st order masking implementation. Also, both hardened 1st order masking implementations are between 1.3 (efficient) and 1.7 (conservative) times slower than the 2nd order naive implementation. However, the 2nd order masked implementation requires 3 times more randomness (which is also an expensive and

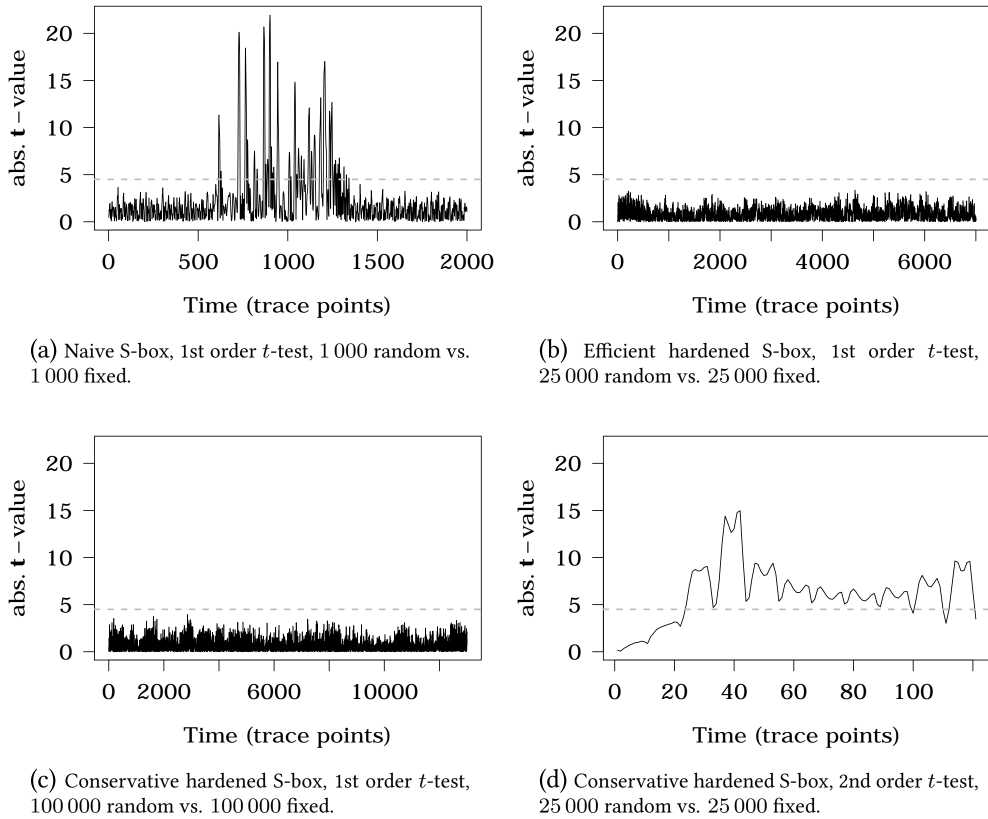
Table 7.1 – Comparison of bit-sliced implementations of the S-box of RECTANGLE in ATmega163.

Masking	Implementation	Latency cycles	Throughput bits/cycle $\times 10^{-3}$	RNG bytes
Unprotected	Naive	32	250	0
1st order	Naive	87	91	4
	Hardened (eff.)	993	8	4
	Hardened (cons.)	1319	6	4
2nd order	Naive	775	10	12

valuable resource in cryptography) than the 1st order masking.

We evaluate our hardened and naive 1st order masked implementations using the random vs. fixed t-test. Figure 7.5 shows the results of our analysis. As expected due to the ILA-breaching effects, the naive 1st order masking implementation quickly rejects the null hypothesis using only $1\,000$ traces, see Figure 7.5a. The efficient hardened implementation does not yield any statistically significant leakage even with $25\,000$ traces (Figure 7.5b). However, we note that a $50\,000$ random vs. $50\,000$ fixed t-test is able to detect leakage. In other words, trying to reduce the cost of enforcing ILA can have a detrimental effect on security. The conservative hardened implementation does not detect any leakage even up to $100\,000$ traces (Figure 7.5c). Note, that a *2nd order* t-test with $25\,000$ traces on a chosen sample window is able to detect the leakage (Figure 7.5d), which is expected since 1st order masking does not protect against 2nd order attacks. Therefore, we conclude that for the given device the informativeness of 1st order attacks is substantially limited and a 2nd order attack is the preferable adversarial strategy. Thus, we can achieve the goal of forcing the attacker to use a 2nd order attack by using our techniques that harden the masking implementation against ILA-breaching effects.

So far, the only way to guarantee the actual security order of a real-world implementation was to increase the scheme's theoretical order d, in order to ensure that the implementation attains an actual order of $\lfloor \frac{d}{2} \rfloor$ [BGG$^+$14, dGPdlP$^+$16] due to the order-reduction. Clearing the ILA-breaching effects results in a significant overhead and it is device-dependent, yet it is the only technique known to us that can enforce 1st order univariate security. However, hardening does not increase the order d of the scheme thus, the costs related to the RNG are not increased. The previous suggestions require a higher order schemes resulting in significant RNG overheads since both the implementation cost (memory and time) and the RNG cost are quadratic with respect to the order d. By comparing our implementations we can observe that hardening a 1st order results in a slower implementation than increasing the order of the mask-



(a) Naive S-box, 1st order t-test, 1 000 random vs. 1 000 fixed.

(b) Efficient hardened S-box, 1st order t-test, 25 000 random vs. 25 000 fixed.

(c) Conservative hardened S-box, 1st order t-test, 100 000 random vs. 100 000 fixed.

(d) Conservative hardened S-box, 2nd order t-test, 25 000 random vs. 25 000 fixed.

Figure 7.5 – Hardened and naive S-box t-test evaluations.

Table 7.2 – RNG costs for 1st and 2nd order masked implementations of RECTANGLE with an AES-based RNG.

Order d	16-bit random numbers per AND	RNG cycles using	
		2-round AES	10-round AES
1st	1	6 000	31 000
2nd	3	18 000	93 000

ing scheme to 2. Still, the hardened solution does not require extra random numbers. We maintain that removing these effects can also be beneficial to higher-order implementations i.e., it is complimentary to masking. The extent to which higher-order implementations can benefit from removing ILA effects remains an open question.

The exact cost and quality of RNG is unclear in many applications. In case of an AES-based pseudo-RNG one can imply the optimistic cost of a 2-round AES execution per 16 random bytes or the pessimistic cost of a 10-round AES execution per 16 random bytes [GSF14]. If we were to use such pseudo-RNG in a masked implementation of RECTANGLE, the cost of the random number generation will prevail on the cost in terms of the number of clock cycles[9]. Indeed, RECTANGLE has 25 rounds and uses 4 16-bit AND instructions per round, see RNG costs in Table 7.2. In the case of a true-RNG, structures that combine physical generators with deterministic random bit generators can face similar performance issues [BK].

[9] http://point-at-infinity.org/avraes/

7.5 Summary

The main points of our findings consist in following statements: (1) independent manipulation of shares in software does not necessarily result in independent leakage (which breaks the ILA), (2) it is possible to enforce ILA in software implementations however, it comes with a relatively significant time overheads but no RNG-related overheads, and finally (3) it is possible to detect ILA-breaching using automated analysis of the assembly code.

Summing up, we highlight the following focal points regarding the ILA-breaching effects and solutions that can be used to avoid their negative effects:

- All identified effects are device-dependent, we observed them only in the ATmega163 microcontroller i.e., there is no hard guarantee that they are observable, reproducible and identical in other AVR-based microcontrollers: it is possible that another model of an AVR microcontroller will only have a subset of these effects or even another ILA-breaching effect that we did not observe (e.g., related to pipelines). In addition to other AVR devices these effects should also be studied in different architectures such as ARM, PIC, etc. Both intra-AVR and inter-architectural observability of the effects as well as whether there are more ILA-breaching effects remains an open question. However, if we were to find out what exactly (i.e., which architectural choices in the design of microcontrollers) causes the ILA-breaching effects then we may predict them in other devices as well as try to avoid them during the construction of new ones.
- The effects are often counter-intuitive when viewed from the assembly-code level of abstraction. They originate from the hardware (structure) and the physical layer (geometrical configuration), thus they can only be detected via experimental evaluation since manufacturers of such devices do not reveal their full specifications down to transistor level. Linking the assembly ILA-breaching effects to a particular hardware component or physical phenomenon is a non-trivial task [RSV+11, Stö12] especially without knowledge of the underlying chip architecture and properties.
- Since the detection of such issues requires experimental evaluation, different instructions, code arrangements and choices of memory units can potentially lead to additional, unidentified ILA-breaching effects. Nevertheless, we claim that it is possible to construct a hardened masked operations in ATmega163 by removing the identified ILA-breaching effects. The question of whether the suggested solutions are computationally optimal or more efficient clearing techniques can be identified remains open.

However, it is very important to note that our findings and analysis are not a definitive proof that the first order leakage is completely removed by the methods that

we use to deal with ILA violations. As already mentioned by Balasch *et al.* [BGG+14], we are always limited by the traces at hand, therefore we cannot rule out the existence of first order leakages. Nevertheless, we establish that their informativeness is limited compared to 2nd order leakages in the target device. Note that extra care is taken in order to assess all effects independently, i.e. we use the suggested solutions so as to isolate the effect under discussion from the rest. The rationale is that by using our suggestions in an implementation we effectively force the attacker to use more complex attacks (even if the complexity of these attacks do not grow exponentially).

The main message of these analyses is that due to the nature of the breaching effects the assembly-level soundness cannot enforce ILA and hence the 1st order security. However, it is possible to acquire sufficient knowledge about effects and solutions for a particular device. These non-intuitive checks discussed above can be subsequently integrated into a code-checking tool which can identify such effects in assembly code. Therefore entire families of leakages resulting from ILA-breaching effects *can* be automatically tracked and effectively removed during the development of a cryptographic device before the code is even loaded into the microcontroller.

Use of tools such as Ascold for detection of ILA violations is significantly faster than tests done using real experiments. Indeed, our method can detect a problem by going through the assembly code once while other leakage detection techniques based on the analysis of power traces do require long acquisition and analysis phases which need many traces (more computations and time than in our case). Moreover, our tool Ascold analyses the code that actually runs in the final product and therefore does not suffer from modifications that can occur during the translation of the program from one language to another (as the tool by Barthe *et al.* [BBD+15]) nor does it suffer from rearrangements and modifications that can be done by a compiler (as the tool by Reparaz [Rep16a]).

Ascold was build specifically for the analysis of the first order masking implementations, but it can be used for the analysis of higher order masking. However, it is important to keep in mind that Ascold will complain in an overprotective manner i.e., it will print error and warning messages as soon as at least two shares of the same secret value are combined, even though it is not necessarily problematic for higher order masking schemes.

The next step and the most interesting direction for future research would be integrating our findings into a compiler. Compiler-assisted masking was already suggested in literature on side-channel analysis [MOPT12], however it does not deal with the effects that we described. Since we can detect the discussed ILA violations using analysis of assembly code, it should be possible to integrate them into a compiler. Another direction for future research that should be investigated is the way of reducing the number of dummy operations that we used in our hardened implementation, right now this implementation is not efficient (in terms of speed) compared to the 2nd order masking or compared to the naive approach due to the size of code and

the amount of dummy read instructions. The possibility of reducing the number of dummy operation through the rearrangement of independent instructions is an interesting project that can greatly improve our results. Finally, it would be interesting to test ASCOLD on the 4th edition of the DPA Contest.

Chapter 8

SAVRASCA: architecture specific simulator

The simulator presented in this chapter is based on the paper *"Use of simulators for side-channel analysis"* presented in Paris (France) at the Workshop on Security for Embedded and Mobile Systems (SEMS) in 2017 [VG17b]. The analysis performed using this simulator is based on the paper *"Flaws in masking scheme of DPA Contest 4"* published in the journal IET Information Security [VG17a].

One of the final states of an implementation of a cryptographic algorithm before the product gets to the consumer is a compiled binary file. This executable file contains all the code that is supposed to be uploaded into the memory of the hardware device such as a microcontroller (for example embedded into a smart card). At the stage of compiled binary file all processes that influence the code are over (i.e., choices of the programmer, modifications introduced by the compiler). The code will not be modified any more unless the designer decides to create a newer version, which means that all software-related issues that can be found during side-channel analysis can be found through the analysis of the compiled file. The main goal of this part of the work is to provide a tool that can be used for side-channel analysis of compiled code. We decided to focus on generic simulations to make them usable with a variety of microcontrollers. As a result we created a simulator that can generate simulated traces based on a compiled binary.

During this work, we explain how to build a generic simulator for generation of simulated traces using generic models on the example of ATmega family of AVR microcontrollers. We then show how this simulator can be used to detect implementation issues related to side-channel analysis. We base our analysis example on the 4th edition of the DPA Contest. Thanks to our simulator we find a novel issue related to side-channel leakage in the implementation that was used in this contest. Finally, we show how to exploit the detected flaw and how to correct the implementation to avoid it.

8.1 Description of the tool

Our simulator is called SAVRASCA which stands for Simulator of AVR Assembly for Side-Channel Analysis. Our simulator is able to output simulated traces based on a compiled binary file. Our code with some examples is available online in our `git` repository[1].

Our simulator is based on an open-source project SimulAVR. SimulAVR is a simulator for the Atmel AVR family of microcontrollers, it is written in C++. SimulAVR (and as the result SAVRASCA) is a cycle accurate simulator i.e., the number of clock cycles reported by the simulator corresponds to the number of cycles used by a real device executing the same code. SimulAVR takes Extensible Linking Format (ELF) compiled binary files as an input. It can be used in order to debug software written for popular microcontrollers such as ATmega128, ATmega328, ATmega16 as well as several other models (the three models listed here are often used in side-channel analysis experiments). Full list of supported microcontrollers (18 models at the time of writing) is available on the SimulAVR official website[2]. It is also available in the Appendix G.

SimulAVR has a tracing feature which is particularly helpful in the debugging process, the simulator can output *execution traces*. An execution trace is the list of instructions (with their parameters) that were executed, see examples in Listings 8.3 and 8.4. An execution trace does not contain instructions that were not executed but that are present in the code (e.g., in case of conditional branches), it contains several occurrences of a block of instructions if it was executed repeatedly (e.g., in case of a loop). Execution traces also contain CPU-wait cycles that are appended by the processor to keep the pipeline flowing smoothly (for instructions that require several clock cycles). This feature turned out to be handy in practice during our experiments in the context of side-channel analysis. It can help to detect conditional instructions that will leak information through side-channels.

In order to use SimulAVR one has to instrument the code and recompile it. Actually, only one minor modification has to be done in the analysed code. It is necessary in order to interact with the interface of the simulator. The modification concerns only the I/O and does not require to redesign the software, it consists of about 15 lines of C++ code and it is well documented on the SimulAVR website. Basically, it allows to route the keyboard input and screen output to a couple virtual pipe registers. Since this modification only touches the I/O operations it should not affect the core of the encryption code from the perspective of side-channel analysis.

Our modification of SimulAVR consists in the following idea: on each access to a memory unit the simulator computes the leakage function in order to generate one leakage point and then writes it into a file. It happens on both types of memory ac-

[1] `https://github.com/nikita-veshchikov/savrasca`

[2] `http://www.nongnu.org/simulavr/usage.html`

cesses i.e., read and write. A memory unit could be a register or a normal memory cell (e.g., SRAM). The leakage function could be any function that could be expressed in `C++`, there are two such functions: one for read memory access and the other one for writing. We are going to call the first one L_r^* and the second one L_w^*. This separation exists because in case of a memory write we can use the new value (that is being written) as well as the old one (already stored in the memory unit). Thus, for example, one might use the Hamming weight of a value for L_r^* and the Hamming distance for L_w^*. In addition, it is also possible to use separate leakage functions for register access and for memory access in order to highlight the difference between these types of memory[3]. Thus, our simulator focuses on the simulations of the data dependent part of the power consumption (P_{val}, recall Section 3.2.3) and it can use ODL as well as MTL leakage models. Note, that one can easily disable one of the two leakage function or even use the leakage function that corresponds only to the register accesses while disabling the one that corresponds to SRAM; it allows to perform a more fine-grained analysis on the code. It is important to highlight that two programs that need the same number of clock cycles to finish their execution will not necessarily result in simulated traces of the same length. The length of a simulated trace will depend on the number of memory and register accesses which depend on the type of instruction. For example, an `ADD R0, R1` instruction will produce 3 leakage points (one per register access for reading them and transferring to the ALU and one for writing the result back to `R0`), while an `INC R0` instruction will produce 2 leakage points (one for accessing `R0` and transferring it to the ALU and one for writing it back to `R0`). This property allows to discover issues resulting from the dependencies between values that are handled by the device and the control flow, as we show in Section 8.2. Figure 8.1 shows the general workflow using Savrasca simulator.

Savrasca is a low level of abstraction simulator compared to Silk and Ascold. However, it uses a lot of simplifications in its model. One of our goals is to build a generic and easy to use simulator. Potentially, it could be fed with much more information (use more complex leakage models) in order to build more accurate and even lower-level simulations. For example, our version does not take into account the address of a memory cell (nor the register id) for the computation of the leakage function. One might also take into account more features related to the architecture of the micro-controller such as different leakage functions for each instruction (to include P_{op} into the simulation) as well effects related to the use of pipelines, bus transfers, etc. However, it would make the simulator less generic and would require more parameters which would ultimately make it more difficult to use. Nevertheless, it will make it more accurate. Our simulator could also be easily modified in order to output several leakage points per instruction, this feature could be used in order to simulate the fact that an instruction leaks different values at the beginning and at

[3]It echoes with our discovery that we made in the analysis of ILA-breaching effects (leakage of information in SRAM vs. registers), recall Section 7.2.1

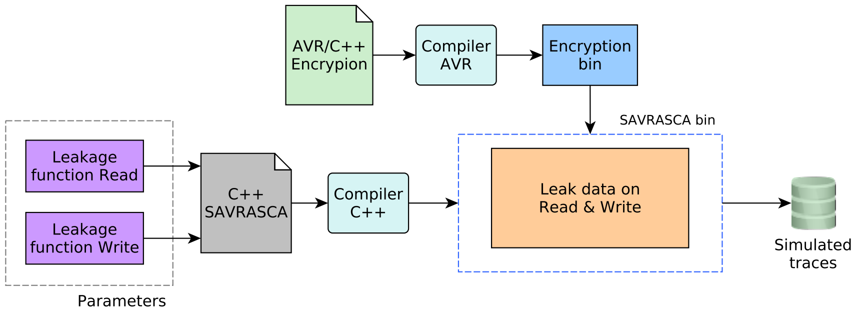


Figure 8.1 – Scheme of the workflow using SAVRASCA simulator.

the end of its execution (just like SILK).

Our tool has several common points with existing simulators. SAVRASCA resembles OSCAR [TAL09] simulator in the way it functions, however it does not require to recompile the code for every new plaintext (or key), which makes it faster than OSCAR. Even if we do not take the re-compilation time into account, SAVRASCA is faster than OSCAR since our modification of SimulAVR had only negligible effect on its performance and Andouard showed that SimulAVR is two times faster than OSCAR [And09, section 4.5.3]. Leakage models used by SAVRASCA can be user-defined (as well as in SILK or Inspector-SCA), i.e., these models can be profiled using techniques applied during the creation of the tool by Debande *et al.* [DBBL12] or ELMO [DMO16] simulator. Unlike SILK or the tool presented by Reparaz [Rep16a], SAVRASCA does not suffer from reordering of instructions (by the compiler), thus it *can* be used to detect issues related to the order of instructions and to other modifications that a compiler makes during the compilation (e.g., for optimizations). Also, unlike other simulators SAVRASCA can output simulated traces *and* execution traces (the unrolled version of code that was actually executed i.e., only the executed instructions from branches and loops) of the same code. Finally, unlike most of existing simulators, SAVRASCA is publicly available.

8.2 Analysis of the DPA Contest 4

We used SAVRASCA in order to analyse the code from the 4th version of the DPA Contest[4]. DPA Contest makes an excellent case study for our simulator, since the code with all implementation details as well as power traces are available on the official website.

[4]http://www.dpacontest.org/v4/rsm_doc.php

It is worth noting that implementation target of the the DPA Contest 4 is a smart card that contains an Atmel's ATmega163 microcontroller. SimulAVR does not support ATmega163, but it does support ATmega16 — a newer version of this microcontroller. There are some minor differences between these two microcontrollers such as the maximum clock frequency (8 MHz for ATmega163 and 16 MHz for ATmega16) and the lower limit for the operating voltage for the highest clock frequency ATmega163 can operate between 4 V and 5.5 V, ATmega16 can operate between 4.5 V and 5.5 V [Atmb, Atmc]. During the acquisitions of power traces for the DPA Contest, the microcontroller used a 3.57 MHz clock and operated at 2.5 V. Another difference between these two microcontrollers is that ATmega16 has one additional instruction: `BREAK`. This instruction is used *"For On-Chip Debug Only"*, see "Instruction Set Summary" section of the ATmega16 datasheet [Atmb]. Using SimulAVR we are looking at the individual instructions and at the instruction flow, which are identical between ATmega163 and ATmega16. Thus, for the purpose of our analysis both microcontrollers could be treated as identical.

During our analysis we used the following settings for the two leakage functions in Savrasca:

$$L_r^* = HW(value)$$
$$L_w^* = HD(value_{old}, value_{new})$$

where HW and HD are the Hamming weight and the Hamming distance functions. We used the same secret key and fed the same inputs that were used during the DPA Contest 4 in our simulator.

DPA Contest 4 implements AES-256 on a smart card and uses a countermeasure against side-channel analysis. More precisely, it uses a lightweight masking countermeasure called Rotating S-box masking (RSM) [NSGD12]. The AES-RSM implementation used for the DPA Contest 4 is executed in constant time regardless of the secret key value, it could also be noticed on power traces that were acquired for the contest. Nevertheless, our simulated traces did not have the same size even though SimulAVR always returned the same number of clock cycles for the execution of the program. This result persisted after extensive code analysis as well as with various compilation options (such as `-O`n with different values of n for optimizations). Same type of differences in simulated trace sizes were noticed with different values of secret keys and inputs. We discovered that the size of simulated traces depended on the value that was manipulated by the microcontroller.

We were able to find that even though the execution always takes the same number of clock cycles, the number of register accesses indeed depends on the manipulated value. This phenomenon occurs due to the way `gf256mul` function is implemented. This function is implemented in assembly language in the file `gf256mul.S` of the DPA Contest implementation (see `aes_enc.c` file, which makes the call to `gf256mul`, Listing 8.1). This function performs a multiplication in a Galois Field

256 (noted $GF(256)$) and it is used to multiply a value by 2 in $GF(256)$ during MixColumns computation (operation `xtime` [DRN03, §2.1.3 page 6]). Notice that the Flash memory is limited on the smart card that is used for the DPA Contest. Thus, the MixColumns operations cannot be saved along with SubBytes (using T-tables [DRN03, §5.2.1 page 18]), but are instead computed. Among others, `gf256mul` has the instructions that cause a side-channel leakage, see Listing 8.2.

```
#define GF256MUL_2(a) (gf256mul(2, (a), 0x1b))
/* ... */
t = tmp[4*i+0] ^ tmp[4*i+1] ^ tmp[4*i+2] ^ tmp[4*i+3];
state->s[4*i+0] = GF256MUL_2(tmp[4*i+0]^tmp[4*i+1])
                  ^ tmp[4*i+0] ^ t;
state->s[4*i+1] = GF256MUL_2(tmp[4*i+1] ^ tmp[4*i+2])
                  ^ tmp[4*i+1] ^ t;
state->s[4*i+2] = GF256MUL_2(tmp[4*i+2] ^ tmp[4*i+3])
                  ^ tmp[4*i+2] ^ t;
state->s[4*i+3] = GF256MUL_2(tmp[4*i+3] ^ tmp[4*i+0])
                  ^ tmp[4*i+3] ^ t;
```

Listing 8.1 – Computation of MixColumns operation on one column using `gf256mul` function in `aes_enc.c`.

```
LSL B       ; Left shift by 1 position
BRCC 3f     ; BRanch if Carry Cleared to label 3f
EOR B, R20  ; B = exclusive-or(B, R20)
```

Listing 8.2 – Part of `gf256mul` from `gf256mul.S`.

```
dpa4.elf 0x0dac: gf256mul+0x6   ADD R22, R22
dpa4.elf 0x0dae: gf256mul+0x7   BRCC ->0x0002
dpa4.elf 0x0db0: gf256mul+0x8   EOR R22, R20
```

Listing 8.3 – Part of the execution trace of `gf256mul` (input MSB=1) produced by Savrasca.

```
dpa4.elf 0x0dac: gf256mul+0x6   ADD R22, R22
dpa4.elf 0x0dae: gf256mul+0x7   BRCC ->0x0002
dpa4.elf 0x0dae: gf256mul+0x7   CPU-waitstate
```

Listing 8.4 – Part of the execution trace of `gf256mul` (input MSB=0) produced by Savrasca.

In Listing 8.2, the register `B` (which is a combination of two bytes of the state) contains the value that is being multiplied by 2 in $GF(256)$ and the register `R20` contains the reducer (always equal to `0x1B` in our case). The instruction `LSL B` (line 1 of Listing 8.2) will set the Carry Flag (CF) to 1 only if MSB of `B` is 1. The carry flag is used in `BRCC` instruction which is executed in 1 cycle if the CF is equal to 1 (does not branch) and it is executed in 2 cycles if CF is equal to 0 (does branch) [Atma]. The

`EOR` instruction is always executed in 1 cycle and the label (`3f`) where `BRCC` branches is situated just after the `EOR` instruction. Thus, if CF is equal to 1, `BRCC` and `EOR` take 1 cycle each for the total of 2 cycles, while if CF is equal to 0, `BRCC` takes 2 cycles and `EOR` is not executed. As the result, the code is always executed in constant time but does not use the registers the same number of times.

In practice it means that depending on the manipulated value we will observe different instructions (either an `EOR` or the second clock cycle of the branch instruction) being executed during the same clock cycle, see Listings 8.3 and 8.4 (these listings are the execution traces obtained with our simulator using different inputs)[5]. An attacker could profile these two instructions and cluster power traces into two sets, thereby recovering the value of the MSB of the intermediate state of the cipher.

Since the difference in power consumption between two different instructions is generally higher than the difference that one might observe while using different values with the same instruction, such clustering could be very telling. An attacker might also use a non-profiled techniques in order to gain information about the internal state of the device by targeting the same vulnerability. The idea consists in using a non-supervised clustering algorithm in order to create two groups of traces, such clustering technique was successfully applied as a part of a side-channel attack in 2013 [LMV+13]. This vulnerability could also be exploited using basic form of DPA (with MSB as the leakage model).

This vulnerability could also be easily targeted using Electro-Magnetic Analysis (EMA) by carefully placing the probes in the region where the accessed registers are situated in order to detect whether a register was or was not used. Such setup will improve the quality of acquired data in terms of distinguishability between cases when the MSB of the intermediate state is 0 or when it is equal to 1. This kind of EMA that is used to find out whether a register is accessed or not effectively leads to a simple (not differential) type of analysis which can distinguish between cases when MSB equals 1 or 0.

It is worth noting that the file `gf256mul.S` contains two implementations of `gf256mul`, one of which has a label `OPTIMIZE_SMALL_A` (actually used in the DPA Contest). Our simulation tests show same type of problem in both implementations.

It is interesting to note that an analogous problem was discovered by Koeune and Quisquater [KQ99] against the similar type of MixColumns implementation in 1999 (not in case of the DPA Contest). Their attack was a timing attack that took advantage of the fact that the two branches in the control flow (the case when a polynomial is reduced vs. the case when it is not) did not take the same amount of time. The 4th version of DPA Contest does however implement MixColumns operation in constant time, but not with a constant amount of register accesses, which is the vulnerability

[5] Note that for an unsigned binary value a left-shift by one (line 1 of Listing 8.2) corresponds to a multiplication by 2, which is also equivalent to adding the value to itself (line 1 in Listings 8.3 and 8.4). This modification is done by the compiler.

that we found. Thus, secure implementations should not only be constant-time, but also constant-flow. It is not easy to keep track of such details in the control flow of a cryptographic algorithm, thus our findings highlight the importance of automated tools such as simulators during the process of development.

As we have just explained, we can relatively easily find the value of the most significant bit of an intermediate value in the first round of the cipher. This intermediate value is a combination between two bytes that go into `gf256mul` function (see e.g., line 5 of Listing 8.1). However, DPA Contest uses a masking scheme, thus each byte is masked and the value of the extracted MSB is also masked. The rationale is that we have to take into account the masking scheme in order to mount a successful attack on the DPA Contest 4. The following sections show how it can be done in case of the 4th edition of DPA Contest.

8.3 Analysis of AES-RSM used in DPA Contest 4

The DPA Contest 4 uses a lightweight masking scheme called RSM. The idea behind RSM countermeasure used in this implementation is to precompute 16 masks $m_0, m_1 \dots m_{15}$ (16 different bytes) such that the input mask m_i corresponds to the output mask $m_{i+1 \bmod 16}$ in the following way:

$$MaskedS_i(B) = S(B \oplus m_i) \oplus m_{(i+1) \bmod 16}$$

where B is the byte of the state. In order to randomize this masking scheme a random offset (between 0 and 15) is generated in order to choose which mask is applied to each byte of the state.

Masks used for DPA Contest 4 are the following 16 values:

```
0x00, 0x0f, 0x36, 0x39, 0x53, 0x5c, 0x65, 0x6a,
0x95, 0x9a, 0xa3, 0xac, 0xc6, 0xc9, 0xf0, 0xff.
```

This AES-RSM implementation was already analysed [BBB+13, BBD+14, MGH14, YE13] and attacked numerous times [KP14] using various techniques including popular distinguishers such as CPA [BDG+14], TA [OWW14] and also Machine Learning [LMBM13, ZGLG14]. These previous analyses mostly focus on ways of retrieving one byte of the key at a time either through combination of several points of power traces or by trying to retrieve the mask offset before going through key hypotheses.

8.3.1 Mask bias

We will say that a randomly chosen bit b is *biased* if the probability $\mathsf{P}(b = 1) \neq 0.5$ (which also gives $\mathsf{P}(b = 0) \neq 0.5$ i.e., the choice is not a "random coin toss"). We will use the symbol $\mathcal{B}$ for *the bias of a bit* to denote the value of the following expression:

$$\mathcal{B} = |\mathsf{P}(b = 1) - \mathsf{P}(b = 0)|$$

Table 8.1 – Binary representation of masks in DPA Contest 4.

Masks	bits							
0x00	0	0	0	0	0	0	0	0
0x0f	0	0	0	0	1	1	1	1
0x36	0	0	1	1	0	1	1	0
0x39	0	0	1	1	1	0	0	1
0x53	0	1	0	1	0	0	1	1
0x5c	0	1	0	1	1	1	0	0
0x65	0	1	1	0	0	1	0	1
0x6a	0	1	1	0	1	0	1	0
0x95	1	0	0	1	0	1	0	1
0x9a	1	0	0	1	1	0	1	0
0xa3	1	0	1	0	0	0	1	1
0xac	1	0	1	0	1	1	0	0
0xc6	1	1	0	0	0	1	1	0
0xc9	1	1	0	0	1	0	0	1
0xf0	1	1	1	1	0	0	0	0
0xff	1	1	1	1	1	1	1	1
$P(b = 0)$	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

it means that if a bit is not biased $\mathcal{B}$ is equal to 0 and $\mathcal{B} \neq 0$ otherwise, while higher value of $\mathcal{B}$ means that the bias is bigger.

Masking helps to randomize the internal state of a cryptographic algorithm during its execution, recall Section 3.4.1. Normally masks should be chosen randomly and the values chosen should be uniformly distributed. Values that were used for the AES-RSM of DPA Contest 4 were selected in such way that if we choose a random mask among these 16 values, the value of each bit is equally likely to be 1 or 0, see Table 8.1. However, these values happen to create a bias when they are combined. This bias is high enough that it allows to mount a first order side-channel attack i.e., the AES-RSM implementation in DPA Contest 4 has a first order leakage[6]. The nature of this bias comes up when several masked bytes are combined together, it is explained in the following paragraphs.

[6]Notice that this leakage is not the same as that identified in a older work [MGH14], and could not have been detected by the method described in this older paper due to the nature of the problem that we explain here.

Let p_i the plaintext bytes, normally organized as follows in AES:

$$\begin{pmatrix} p_0 & p_4 & p_8 & p_{12} \\ p_1 & p_5 & p_9 & p_{13} \\ p_2 & p_6 & p_{10} & p_{14} \\ p_3 & p_7 & p_{11} & p_{15} \end{pmatrix}$$

then the input of the MixColumns $z_i = SB(p_i \oplus k_i) \oplus m_i$ followed by ShiftRows is laid out as follows:

$$\begin{pmatrix} z_0 & z_4 & z_8 & z_{12} \\ z_1 & z_5 & z_9 & z_{13} \\ z_2 & z_6 & z_{10} & z_{14} \\ z_3 & z_7 & z_{11} & z_{15} \end{pmatrix} \xrightarrow{\text{ShiftRows}} \begin{pmatrix} z_0 & z_4 & z_8 & z_{12} \\ z_5 & z_9 & z_{13} & z_1 \\ z_{10} & z_{14} & z_2 & z_6 \\ z_{15} & z_3 & z_7 & z_{11} \end{pmatrix}. \tag{8.1}$$

The MixColumns in DPA Contest uses the trick of Rijmen & Daemen described in the Rijndael AES proposal [DRN03, § 5.1, page 16], see line 3 of Listing 8.1. Some commonly used values are factored, such as the exclusive-or combination of all four bytes of the column:

$$z_0 \oplus z_5 \oplus z_{10} \oplus z_{15} \tag{8.2}$$

in the `C++` code actual indices are $0, 1, 2$ and 3 since a ShiftRows operation was applied to the state. Here, we see that two masked bytes are combined in lines 4, 6, 8 and 10 of Listing 8.1 (argument to `gf256mul` function). Thus, several times we have the combination of masked bytes of the state such as $z_i \oplus z_{(i+5) \bmod 16}$. Table 8.2 gives the bias of every bit of a combination of two masks (bias for all offsets between two masks is presented in Table 8.3). We can notice that all bits of such combinations, except the 5th one are biased (or unbalanced). We can also notice that the 3rd bit is always 1, in other words, the 3rd bit of the resulting byte is always masked using the same value (in this case it is masked with value 1 or in other words it is always inverted). Nevertheless, MSB leaks more: it has more points of interest that leak information and the leakage is higher due to the problem described at the beginning of Section 8.2, see the side-channel leakage profiling described in Figure 8.3.

Biases for all offsets between two masks are presented in Table 8.3, we can see that for these values of masks any combination of two bytes has some bias (for at least 4 out of 8 bits). We can also note that a combination of two masked values which are 8 bytes apart ($m_i \oplus m_{i \bmod 16}$) completely removes the masks i.e., a bit is always "masked" with the same value in such combination. Moreover, the 3rd bit always has the highest possible bias (1.00) for any of the combinations, which means it is not masked at all.

Masks of the analysed AES-RSM implementation also create a bias when we combine 4 bytes of the same column of the state at a time, see Table 8.4. This combination is also used in the implementation of DPA Contest 4 (see line 3 of Listing 8.1). When combining 4 bytes, almost all bits are well balanced (no bias), however two of the bits

Table 8.2 – Bias of every bit of the combinations $m_i \oplus m_{(i+5 \bmod 16)}$. Unbalanced bits are highlighted.

Bit index	7	6	5	4	3	2	1	0
bit = 0 (%)	37.5	25.0	50.0	62.5	0.0	37.5	75.0	62.5
bit = 1 (%)	62.5	75.0	50.0	37.5	100	62.5	25.0	37.5
$\mathcal{B}$	0.25	0.50	0.00	0.25	1.00	0.25	0.50	0.25

Table 8.3 – Bias in combinations of masks of DPA Contest 4. Each column is the bit index, each line is the bias of a combination $m_i \oplus m_{(i+\text{offset}) \bmod 16}$. The highlighted line corresponds to the offset that is used in MixColumns combination of the first round.

Offset	Bit index							
	7	6	5	4	3	2	1	0
0	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1	0.75	0.50	0.00	0.25	1.00	0.25	0.50	0.75
2	0.50	0.00	1.00	0.50	1.00	0.50	0.00	0.50
3	0.25	0.50	0.00	0.25	1.00	0.25	0.50	0.25
4	0.00	1.00	1.00	0.00	1.00	0.00	1.00	0.00
5	0.25	0.50	0.00	0.25	1.00	0.25	0.50	0.25
6	0.50	0.00	1.00	0.50	1.00	0.50	0.00	0.50
7	0.75	0.50	0.00	0.25	1.00	0.25	0.50	0.75
8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
9	0.75	0.50	0.00	0.25	1.00	0.25	0.50	0.75
10	0.50	0.00	1.00	0.50	1.00	0.50	0.00	0.50
11	0.25	0.50	0.00	0.25	1.00	0.25	0.50	0.25
12	0.00	1.00	1.00	0.00	1.00	0.00	1.00	0.00
13	0.25	0.50	0.00	0.25	1.00	0.25	0.50	0.25
14	0.50	0.00	1.00	0.50	1.00	0.50	0.00	0.50
15	0.75	0.50	0.00	0.25	1.00	0.25	0.50	0.75

(3 and 5) are essentially never masked. This property leads to a first order leakage in the DPA Contest 4 implementation. In order to exploit this leakage an attacker would need to attack 4 bytes at a time and use bits 3 and 5 in the distinguisher. Ultimately it means that the attacker will have to go through 2^{32} hypotheses, which is quite unusual for a side-channel attack since it requires more computational power (than for a more common choice of targeting one byte at a time which leads to 2^8 hypotheses). Nevertheless, such an attack is feasible and could be performed in a relatively short

Table 8.4 – Bias in combinations of masks of DPA Contest 4. Bias of every bit of the combinations 4 mask bytes from the same column of the state i.e., $m_i \oplus m_{(i+5) \bmod 16} \oplus m_{(i+10) \bmod 16} \oplus m_{(i+15) \bmod 16}$. Unbalanced bits are highlighted.

Bit index	7	6	5	4	3	2	1	0
bit = 0 (%)	50	50	100	50	100	50	50	50
bit = 1 (%)	50	50	0	50	0	50	50	50
$\mathcal{B}$	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0

time using several standard PCs [MOW14].

When masks are chosen uniformly at random and are not biased (and their combinations are not biased as well), then an attacker does not learn anything by finding out the masked internal state of a cipher. The bias in masks or their combinations helps the attacker in the task of mounting a side-channel attack. If a combination of masks of a bit has some non negligible bias (e.g., it is more likely be equal to 1) then by discovering the value of the masked internal state, the attacker learns that the non-masked version of the internal state is more likely to be equal to a certain value. By acquiring more data (power traces) the attacker can effectively find out the real value of the non-masked internal state of the algorithm.

8.3.2 Experimental results

The choice of unbalanced values for the AES-RSM masking scheme could be exploited. We were able to successfully attack the DPA Contest 4 using the issue that we found using Savrasca combined with our analysis of the masking implementation. We performed a first order attack that we describe here below.

Attack settings and requirements

The vulnerability that we discovered can be exploited using several different attacks with various distinguishers. It includes profiled and unprofiled attacks as well as distinguishers of different complexity (and strength) such as DoM or the one used in the TA. On one hand using an unprofiled distinguisher is possible and it puts less constraints on the attacker (no need to do the profiling step and less complex computations for statistics). On the other hand TA is more powerful and can be used with less traces in the attack step.

For the sake of clarity and without the loss of generality we describe only one simple and fast to evaluate multiple times attack. In our work we focus on a simple distinguisher in order to highlight the fact that the vulnerability which we exploit leads to very powerful attacks even when the adversary uses very basic statistical

tools (difference of means). We hope that it helps to emphasise the importance of such problems, in other words the bias in combination of masks is a big issue for such schemes. However, in order to speed up the computation of the success rates of our attacks we decide to use a profiled version of the difference of means distinguisher. This choice allows to speed up the computations because we need to cluster traces into two groups only once (per attack) and this same clustering is used with all key hypotheses. We are attacking two bytes (2^{16} hypotheses) and repeating our experiments 400 times (per number of traces and per bit index) to get the success rate. Thus, this acceleration trick is very useful to get the result (success rate curves) in a more reasonable amount of time: we have to perform the clustering only 400 times (for a fixed number of traces) instead of 400×2^{16} which actually gives us only one point on a success rate curve. Note that real attackers are not interested in computing the success rate, but in finding the key, therefore they have to perform less computations and thus they can use an unprofiled version of DoM. An unprofiled attack that uses simple difference of means or correlation as a distinguisher have same basic requirements as our attack but does not need the profiling step. A profiled attack such as TA can also benefit from the values of masks during the profiled step.

For our attack we suppose that the attacker can record a small set of traces for profiling. The attacker has to know the plaintexts, the key and masks associated with these traces (for profiling). We also suppose that the attacker can acquire a set of traces with an unknown key, unknown masks (offsets) and known plaintexts for the attack. Our attack also works if the mask spaces are not the same between the profiling and the attack step (i.e., they use two different sets of 16 masks in their implementations), but the two sets of masks must have the same bias. During the profiling step masks are used only to find out whether a bit is equal to 0 or 1 for a single intermediate value and to associate a trace to a set, while the attack step does not use the values of masks at all[7] (only values of the plaintext and the fact that a combination of masks is biased) even the exact value of the bias is not used during the attack, the attacker only has to know whether the combination is more likely to be equal to 0 or to 1. The rationale is that if the attacker knows which particular instruction is targeted (and which points of a power trace from the attack set correspond to this instruction), there is no need of using the same algorithm and knowing masks during the profiling step, one can simply profile the targeted instruction using all possible values. Our experimental results are based on the idea that the attacker knows the implementation that is used for AES, more particularly the attacker knows that the specific combinations of values (that we analyse) are indeed handled at some point during the algorithm. However, notice that DPA Contest 4 uses a set of very common tricks to build an efficient implementation, which is also the case for a lot of other real-world implementations because there are only a handful of possible efficient implementations of AES.

[7] Our `C++` code of the attack does not have values of masks in it.

Finally, it is interesting to note that our attack can work on full entropy masking schemes that reuse some masks on several parts of the state i.e., each mask is selected at random among all possible values but it is used multiple times during the same encryption.

Attack description

Our attack consists of three steps, first the attacker has to profile power traces, then (during the actual attack) the attacker has to group power traces into two clusters and finally he has to go through the hypotheses of the sub-key in order to determine the correct one. More formally, we build two templates Θ_0 and Θ_1 using profiling traces:

$$\Theta_0 = mean(\mathcal{T}[i] \text{ such that } \forall i : bit_{id}(\mathcal{Z}[i]_0 \oplus \mathcal{Z}[i]_1) = 0),$$
$$\Theta_1 = mean(\mathcal{T}[i] \text{ such that } \forall i : bit_{id}(\mathcal{Z}[i]_0 \oplus \mathcal{Z}[i]_1) = 1),$$

where Θ_0 and Θ_1 are vectors of points and $bit_{id}(\cdot)$ is the function that returns the value of the bit index id (targeted by the attack). During the first part of the attack step, we cluster traces into two groups g_0 and g_1 by comparing each trace to the two profiles:

$$\begin{aligned} \mathcal{T}[i] &\in g_0 \qquad \text{if } dist(\mathcal{T}[i], \Theta_0) < dist(\mathcal{T}[i], \Theta_1), \\ \mathcal{T}[i] &\in g_1 \qquad \text{otherwise,} \end{aligned}$$

where the function $dist(\cdot, \cdot)$ is the Euclidean distance.

The final step of the attack searches the sub-key. This sub-key is the hypothesis that maximizes the amount of intermediate states that were clustered using it according to the clustering of power traces:

$$\hat{k} = \arg\max_{k_a, k_b} \left\{ count\Big(\forall i \cdot bit_{id}\big(S(\mathcal{P}[i]_a \oplus k_a) \oplus S(\mathcal{P}[i]_b \oplus k_b)\big) = Cl(\mathcal{T}[i])\Big) \right\} \tag{8.3}$$

where the function $Cl(\cdot)$ returns the id of the cluster that contains the trace:

$$Cl(\mathcal{T}[i]) = \begin{cases} 0 & \text{if } \mathcal{T}[i] \in g_0, \\ 1 & \text{if } \mathcal{T}[i] \in g_1. \end{cases}$$

In other words, during this step we cluster the intermediate states for different plaintexts with each hypothesis and compare whether clusters based on power traces are similar to clusters based on processed values.

The resulting sub-key $\hat{k}$ consists of two parts k_a and k_b, where a and b are such that $a = b + 5 \mod 16$. The difference (of 5) between indexes for the two bytes of the secret key is explained by the fact that we target two bytes of the same column after the ShiftRows operation, see Equation 8.1. Thus, our attack targets 2 bytes of the secret key and the attacker has to go through 2^{16} hypotheses.

Experimental results of the attack

We performed a profiled DoM attack described above on reference traces. For our experiments we use reference traces from the first part of the public dataset of DPA Contest 4 (file `DPA_contestv4_rsm_00000.zip` with 10 000 traces). These traces are available on the official website[8]. We calculate the success rate of an attack based on 400 repetitions of the attack. Each repetition uses a different set of power traces, each set was selected by choosing traces uniformly at random from the entire dataset (without using the same trace multiple times in the same repetition). Each of our attacks uses 8 points per trace. We use 1024 traces in order to build our profiles. We use the DoM distinguisher in order to choose these 8 points (using a known key). For each attack these 8 points were chosen as points that give the largest absolute value of DoM between two clusters during the learning phase. The two clusters are created based on the value of a single bit of the input into the `gf256mul` function ($z_0 \oplus z_5$).

In our experiments we attack two bytes of the key: k_0 and k_5. We conduct 10 attacks in total, all attacks are focused on the time interval during the first round when `gf256mul` function is called and executed. Namely, we conduct 8 identical attacks (one per bit index) and two additional attacks on the MSB (using points from different clock cycles) to estimate the impact of the issue described in Section 8.2 (`EOR` vs. `BRCC` instructions).

The success rates of our attacks on each bit are shown in Figure 8.2. The difference between the success rates of attacks on different bits comes from two separate phenomena: (1) the bias in combinations of masks of each bit which depends on the choice of masks, and (2) the amount of leakage of each bit which depends on the target hardware.

We can see that some bits leak more information (Figures 8.3 and 8.4) and bits that have higher bias in masks (Table 8.2) are easier to exploit. Thus, the bit 3 which is always masked with the same value gives the attack that has the highest success rate and the bit 5 (no bias in masks) does not provide useful information. We can also notice that the success rate of the attack on bit 1 is higher than the success rate of the same attack on bit 6, because, even though combinations of masks of these bits have the same bias, bit 1 leaks more (through the nature of the device). We can notice the same phenomenon with bits that have mask bias 0.25. However, the attack on the MSB has the highest success rate among all bits that have unbalanced masks with the same value of bias.

While attacks on all bits exploit the hardware leakage (related to P_{val}) and the bias in masks, the attack on the MSB also benefits from the fact that two different instructions are executed during a certain clock cycle as explained in Section 8.2 (i.e., it can also exploit differences in P_{op}, recall Equation 3.7). This additional impact is visible in Figure 8.4. In order to find out the impact that the unfortunate choice of the

[8] `http://www.dpacontest.org/v4/rsm_traces.php`

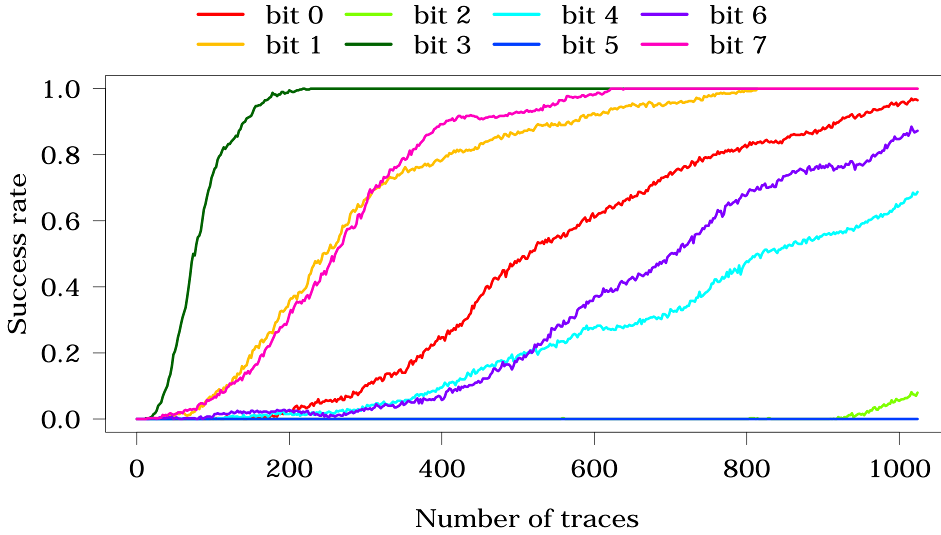


Figure 8.2 – The success rate of the attack on each bit of the intermediate state $z_0 \oplus z_5$. The success rate was calculated using 400 repetitions.

implementation has on the success rate we perform a second experiment. This time we perform the same DoM attack, but we exclude the points from the clock cycle where the difference between BRCC and EOR instructions appears. "Fortunately" the attacker has a lot of choice to target a combination of bytes during this attack, see Figure 8.5. We use the next 8 best points (from a different clock cycle), thus this attack does not benefit from the difference between instructions. We also performed the same attack using 8 points *only* from the cycle where either BRCC or EOR can be executed.

Figure 8.6 shows the success rate of these attacks together with the original attack on the MSB (that simply uses 8 best points from any clock cycle). We can see that the implementation error in gf256mul function actually gives a clear additional advantage to the adversary which results in a boost of the success rate of the attack.

As could be noticed in Figure 8.2, the bias in values that are used for masks allows us to attack this protected implementation using a first order attack using one of the simplest distinguishers. In other terms, this implementation has a first order leakage that was not discovered before. Interestingly, the other first order leakage identified earlier by Moradi *et al.* [MGH14] also targeted biases in values. It required about 500 traces to recover the key while our attack on the 3rd bit that uses a much simpler distinguisher has a stable success rate of 1 after 230 traces. These results highlight the fact that the issue that we identify and exploit represents a big threat to LEMS implementations. We can also notice that the implementation issue in the computa-

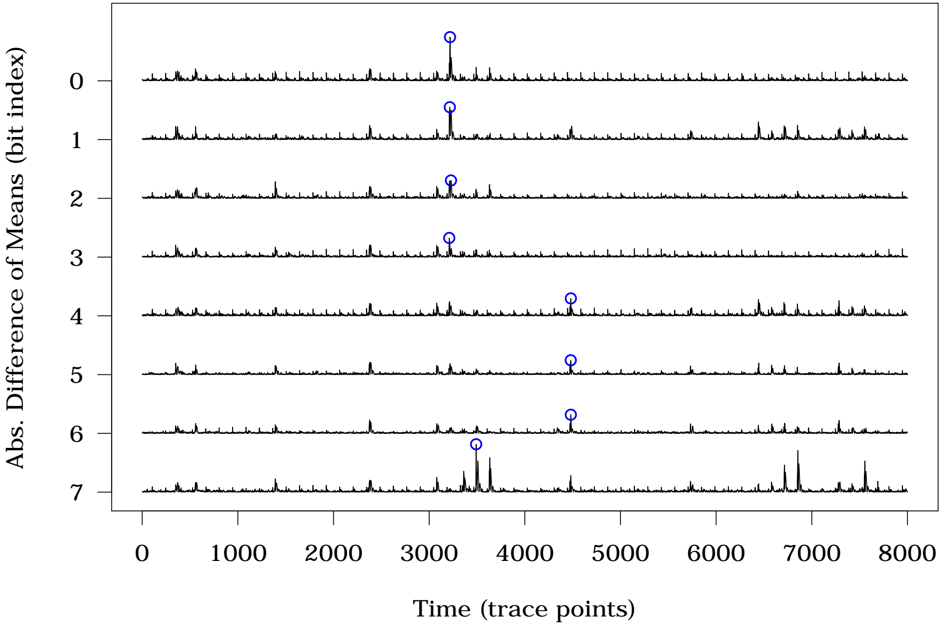


Figure 8.3 – Difference of Means. Best point is highlighted with a circle.

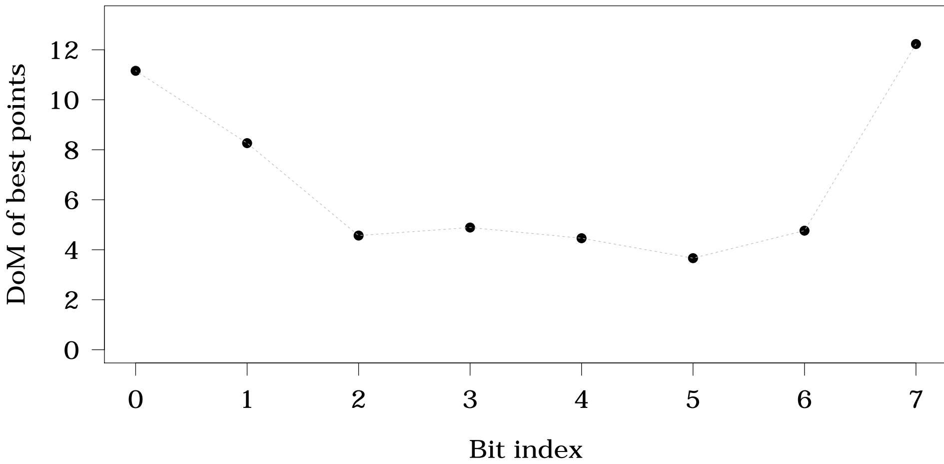


Figure 8.4 – Absolute value of DoM on each bit. Highest points per bit.

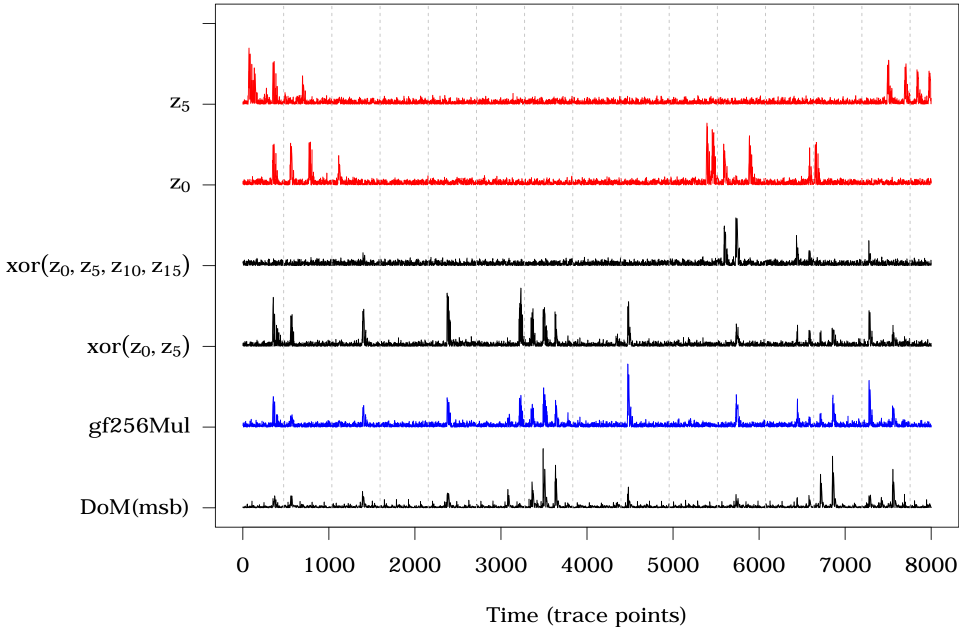


Figure 8.5 – Different distinguishers (correlation and difference of means) applied on the values used during the `gf256mul` function.

tion of MixColumns operation contributes to the leakage that exists due to the bias in masks, ultimately it results in a more powerful attack against this implementation.

Full attack and possible improvements

Our experiments focus on two bytes of the key. This attack could be easily used in order to extract the entire key since `gf256mul` is called 4 times during MixColumns (lines 4, 6, 8, and 10 of Listing 8.1). The same offset between bytes of the state is used (5 bytes) during these calls, moreover each byte of the state is involved in two computations of `gf256mul`. Thus an attacker can use multiple attack strategies in order to perform an attack on all parts of the key.

First simple strategy consists in attacking two bytes that are used in one call of `gf256mul` and then attacking a call to `gf256mul` that uses two other bytes e.g., lines 4 and 8 in Listing 8.1. This strategy would require to enumerate 2^{16} values twice in order to extract 4 bytes of the secret key (2^{17} values to test in total).

Second strategy consists in attacking two bytes used in the first call to `gf256mul` (2^{16} values to test, line 4 of Listing 8.1), supposing that the result is correct and attacking the next call (line 6 of Listing 8.1) while using previously discovered byte thus going through 2^8 values for the remaining byte. Then finally repeat this process

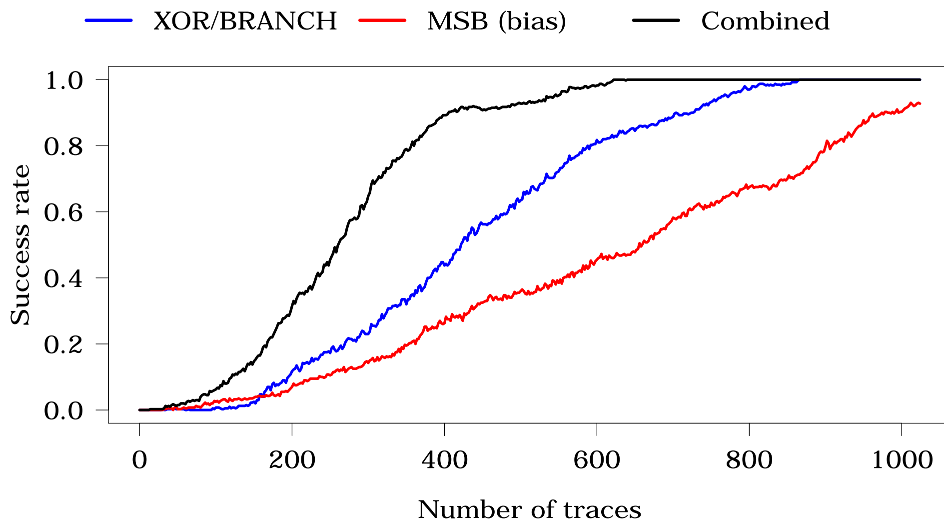


Figure 8.6 – Success rate of 3 DoM attacks. DoM on the instruction `XOR` / `BRANCH`. DoM on the second best clock cycle (MSB) and the combined attack.

one more time to attack the last byte used in the next call to `gf256mul` (line 8 of Listing 8.1). As a result this technique would require to go through $2^{16} + 2^8 + 2^8$ hypotheses in order to recover 4 bytes (which is less than required by the first strategy).

Both attack techniques are interesting in practice. While the second one requires less computational resources than the first one, the first one will not propagate an error if it occurs during the attack on the first part of the key (which might be the case in the second one). Both attacks could use a checking mechanism that is based on the fact that we attack two bytes at a time and that each byte is used twice. In both cases the attacker can target all four calls to `gf256mul` and double-check the results.

Our attack could probably be further improved by attacking several instructions (from different clock cycles) and by using better profiling (more points per trace and more traces in the profiling set). This attack could be also improved by combining attacks on several bits at a time as well as through combination with attacks on operations other than MixColumns.

8.3.3 Balanced values for masks

The rotating S-box masking scheme is very useful in practice when we need a relatively lightweight countermeasure against side-channel attacks. Thus, it is interesting to find a set of 16 masks that do not have any bias whether one-by-one or when combined as described earlier.

Fixing 2 and 4-part combinations

In order to mitigate the problem that we found, designers should be able to generate masks that do not create bias when combined two-by-two or four-by-four. More formally, we want a set of 16 masks m_i where we have $P(b = 1) = 0.5$ and $\mathcal{B} = 0$ for every bit b of the combination of two bytes $m_i \oplus m_{i+5 \pmod{16}}$ as well as for every bit of the combination of four bytes $m_i \oplus m_{i+5 \pmod{16}} \oplus m_{i+10 \pmod{16}} \oplus m_{i+15 \pmod{16}}$. Such set of masks would not leak in the way that we were able to exploit during our attacks and it can be used in the AES-RSM implementation of DPA Contest. We were able to find many sets of masks that fit our criteria. Here we present a simple way of generating them.

In order to generate a set of masks that fits our requirements we start by generating all sequences of 16 bits that have 8 bits equal to 1 (and 8 equal to 0). There are only $\binom{16}{8} = 12870 \approx 2^{13.65}$ such sequences, so nowadays it is possible to enumerate them all on a standard computer. Each of these sequences is a potential candidate for any bit index of the future set of masks (i.e., it corresponds to any column in Table 8.1 or in Table 8.6). Once this set is generated, we can filter it in order to reject all sequences of bits that do not fit the criterion of the bias (combination of two and four bits situated 5 values apart as bytes in MixColumns). After this filtering we obtained 2192 unique sequences of 16 bits, each of them corresponded to our criteria. The final step is to choose 8 of them in order to obtain 16 different values (masks). Table 8.5 presents 8 examples of resulting masks and Table 8.6 shows binary representation of our first example of balanced masks. There exist many more such sets of masks, there are 2192 binary strings and we need to choose 8 of them thus, there are $\binom{2192}{8}$ possibilities to choose from (approximately $2^{73.46}$). Note that not all of them fit our criteria of having 16 different values, we were not able to go through the entire space (due to its size) to get the exact number of such sets. Nevertheless, we were able to generate our examples by picking 8 random binary strings and verify the condition.

The same technique could be used to generate a set of more than 16 masks for a different algorithm or if we want to use more than 16 precomputed values in the AES-RSM scheme. Also, this way of generating the set of masks could be modified in order to encompass new criteria in case if a novel problem related to the choice of these values is discovered.

Other biased combinations

The way of generating the set of masks presented in the previous section could be modified in order to encompass new criteria in case if a novel problem related to the choice of these values is discovered. New criteria should also be included depending on the specific low entropy masking scheme and on the algorithm that it is being applied to (in our case we care about combinations that occur in MixColumns, thus we analyse masks that are situated 5 bytes apart form each other).

Table 8.5 – Examples of balanced masks for combinations of 2 and 4 bytes.

id	Masks
1	0x13, 0x94, 0x25, 0xCB, 0x8E, 0x5F, 0xD9, 0x37 0x77, 0xC6, 0xA8, 0x38, 0x05, 0xEA, 0x70, 0xE8
2	0xAE, 0x1F, 0x43, 0x70, 0x6A, 0x5C, 0xA4, 0xE1 0x5D, 0xAF, 0x11, 0xD6, 0xAB, 0xC2, 0xB0, 0x1D
3	0xA4, 0xE1, 0x08, 0xDC, 0xDF, 0x6D, 0x00, 0x32 0xA0, 0x5F, 0x6F, 0xF6, 0x9B, 0x97, 0x11, 0x6A
4	0x6F, 0x84, 0xB5, 0x0F, 0x9A, 0x65, 0x4B, 0x3D 0x3A, 0xEA, 0xE2, 0x98, 0x55, 0x11, 0xD0, 0xE6
5	0xC0, 0x5A, 0x7D, 0x7F, 0x36, 0x82, 0xED, 0x87 0x43, 0x76, 0xA9, 0xFC, 0x0C, 0x80, 0x9B, 0x31
6	0xF4, 0xC3, 0xE8, 0x22, 0x39, 0x74, 0x1E, 0xF8 0x4F, 0x57, 0x87, 0x01, 0xF3, 0x2C, 0x8C, 0x9B
7	0x56, 0x11, 0xFA, 0x0B, 0x8B, 0xED, 0xFC, 0xB2 0x14, 0xAC, 0xA1, 0x6F, 0xD3, 0x62, 0x1C, 0x45
8	0x27, 0x00, 0x5B, 0xE0, 0x70, 0x47, 0xBE, 0x0E 0xD0, 0x39, 0x8D, 0x81, 0xDF, 0xFF, 0xA8, 0x76

If we take a closer look at the algorithm in the Listing 8.1 (line 3) we can see that 4 masked bytes are combined. Since the system is used in an 8-bit software implementation there are essentially two ways of combining 4 bytes using an exclusive-or. Let z_0, z_1, z_2 and z_3 be four intermediate values (bytes) that we want to combine, the first way of getting the combinations $z_0 \oplus z_1 \oplus z_2 \oplus z_3$ is the following:

$$\left(\left((z_0 \oplus z_1) \oplus z_2\right) \oplus z_3\right) \tag{8.4}$$

and the second one is given by:

$$\left(\left(z_0 \oplus z_1\right) \oplus \left(z_2 \oplus z_3\right)\right) \tag{8.5}$$

where parentheses give the order in which the bytes are combined. In some cases we may also care about the order of z_i in those combinations and in the actual pairwise exclusive-or operations (e.g., z_0 may be combined with z_2 instead of z_1 as in our example). However, for the sake of simplicity we focus on the arrangements of exclusive-or operations with the same fixed order of bytes.

Table 8.6 – Example of a balanced mask with their binary representation.

Masks	bits							
0x13	0	0	0	1	0	0	1	1
0x94	1	0	0	1	0	1	0	0
0x25	0	0	1	0	0	1	0	1
0xcb	1	1	0	0	1	0	1	1
0x8e	1	0	0	0	1	1	1	0
0x5f	0	1	0	1	1	1	1	1
0xd9	1	1	0	1	1	0	0	1
0x37	0	0	1	1	0	1	1	1
0x77	0	1	1	1	0	1	1	1
0xc6	1	1	0	0	0	1	1	0
0xa8	1	0	1	0	1	0	0	0
0x38	0	0	1	1	1	0	0	0
0x05	0	0	0	0	0	1	0	1
0xea	1	1	1	0	1	0	1	0
0x70	0	1	1	1	0	0	0	0
0xe8	1	1	1	0	1	0	0	0

Note, that in most software implementations it is possible to execute only one exclusive-or operation at a time. Thus, the first way of combining would give us (and would leak) information on the combination $z_0 \oplus z_1$, followed by $z_0 \oplus z_1 \oplus z_2$ and finally $z_0 \oplus z_1 \oplus z_2 \oplus z_3$. Thus, the first way of combining would also leak information about the combination of 3 masks!

Upon the analysis of combinations $m_i \oplus m_{i+5 \pmod{16}} \oplus m_{i+10 \pmod{16}}$ in the mask set of the DPA Contest we can note that the bias is always 0 for all bits (i.e., no issues with unintentional leakage). However, if we look at the masks from Table 8.5 we can notice that their combinations (of 3 values) are biased. There are two ways of dealing with this issue. The first one would be to use the second way of getting a combination of 4 bytes (Equation 8.5), however it means that the designer of the masking scheme (and the person who selects the values for the masks) suppose that the software developer will be aware of it and will take care of the potential problem. However, it is not always possible to ensure this assumption in practice (human mistakes and compiler optimisations might be an issue). Thus, the second way of dealing with combinations of 3 values or any other similar problems would be to encompass additional properties in the mask generation process.

Table 8.7 shows another set of masks that we generate using an additional filtering rule (on the combinations of 3) while filtering sequences of 16 bits as in the previous section. In the case if we forbid bias in combinations of two bytes $m_i \oplus m_{i+5 \pmod{16}}$,

Table 8.7 – Examples of sets of balanced masks for combinations of 2, 3 and 4.

id	Masks
1	0x13, 0x94, 0x2D, 0xC3, 0xE6, 0x9E, 0x79, 0x41 0x82, 0xBD, 0x58, 0x62, 0xE7, 0x9C, 0x3D, 0x6A
2	0xAE, 0x5F, 0x03, 0x30, 0x28, 0x96, 0x79, 0xD5 0xEA, 0xCE, 0xA5, 0x25, 0x52, 0xB2, 0x59, 0xCD
3	0xA4, 0xE1, 0x08, 0xDC, 0x5F, 0x7B, 0xC5, 0x8D 0xE2, 0x70, 0x92, 0xB4, 0x6F, 0x3B, 0x02, 0x1F
4	0x6F, 0x84, 0xB5, 0x0F, 0x1A, 0xF0, 0xDF, 0xE8 0xF5, 0x16, 0x11, 0x4A, 0x00, 0xF3, 0x6C, 0xAB
5	0xA3, 0x6A, 0x14, 0xBC, 0x5A, 0xE5, 0xD8, 0xF7 0xA1, 0x03, 0x32, 0x0D, 0x4F, 0xED, 0x9E, 0x50
6	0xC0, 0x5A, 0x7D, 0x7F, 0xA6, 0x37, 0x12, 0x19 0xCE, 0x5B, 0xA1, 0xA8, 0xC7, 0xAC, 0x50, 0xA5
7	0xF4, 0xC3, 0x68, 0xBA, 0xA1, 0x07, 0x56, 0xED 0x74, 0x11, 0x0C, 0x9B, 0xE3, 0x1E, 0x4B, 0xBC
8	0xE9, 0x06, 0xB3, 0x0E, 0x6F, 0xE0, 0x94, 0x71 0x86, 0xDB, 0x3D, 0x9A, 0x18, 0x67, 0xFD, 0x40

three bytes $m_i \oplus m_{i+5 \pmod{16}} \oplus m_{i+10 \pmod{16}}$ and four bytes $m_i \oplus m_{i+5 \pmod{16}} \oplus m_{i+10 \pmod{16}} \oplus m_{i+15 \pmod{16}}$ we end up with 768 binary strings of 16 bits that satisfy our requirements. Once again, here we present 8 examples of sets of well-balanced masks, but there are more of such sets. The total number of sets to choose from is equal to $\binom{768}{8}$ which is approximately $2^{61.33}$ (note that not all of them will produce a set of 16 different values that we impose on the scheme).

Note, that similar issues related to combinations of several masks can appear in other LEMS applied to different algorithms. The techniques of combining values by groups (as in Equation 8.5) instead of "one-by-one" (as in Equations 8.4) might become a necessity if a mask set has so many constraints on combinations that it cannot be satisfied using the required number of different values (16 in our case). In other words, we may need to choose a specific way of combining values (and forbidding the others for security reasons) in order to be able to relax the constraints on the values of masks.

For a number of combined bytes other than 4 we may use more different orderings for combining them. For example, let us take a hypothetical encryption algorithm that uses a combination of 6 bytes on the intermediate state ($z_0, z_1 \dots z_5$). We

will use $\underset{i}{\oplus}$ to note that the given exclusive-or operation ($\oplus$) leaks information on the combination of i values e.g., $\underset{3}{\oplus}$ means that the operation leaks information on the combination of 3 bytes. The 6 values can be combined one-by-one (it will leak information on combinations of $2, 3, 4, 5$ and 6 values):

$$\left(\left(\left(\left((z_0 \underset{2}{\oplus} z_1) \underset{3}{\oplus} z_2\right) \underset{4}{\oplus} z_3\right) \underset{5}{\oplus} z_4\right) \underset{6}{\oplus} z_5\right) \tag{8.6}$$

or by three groups of 2 (leakage on combinations of $2, 4$ and 6 values):

$$\left(\left((z_0 \underset{2}{\oplus} z_1) \underset{4}{\oplus} (z_2 \underset{2}{\oplus} z_3)\right) \underset{6}{\oplus} (z_4 \underset{2}{\oplus} z_5)\right) \tag{8.7}$$

or in two groups of 3 bytes (resulting in leakage of combinations of $2, 3$ and 6 values):

$$\left(\left((z_0 \underset{2}{\oplus} z_1) \underset{3}{\oplus} z_2\right) \underset{6}{\oplus} \left((z_3 \underset{2}{\oplus} z_4) \underset{3}{\oplus} z_5\right)\right) \tag{8.8}$$

in these equations each $\oplus$ produces a combination of values that will leak information on the combination of its operands (left and right parts). In case of a specific algorithm we may use the technique from the Equation 8.7 or 8.8 (while avoiding the other one). Note that here we only focused on the way of combining values while always using them in the same order $(0, 1 \ldots 5)$ but even the order of z_i in each combination may be important for a specific implementation of a given algorithm.

8.4 Note on DPA Contest 4.2

As a response to the attacks on the 4th edition of the DPA Contest its authors created a new updated version (4.2) of the contest with several modifications and improvements [BBD+14].

The version 4.2 is fully written in assembly language, it uses a different set of masks[9] and also uses a shuffling countermeasure called Random Permutation (RP). We applied the same analysis to this new version.

As a result we found that DPA Contest 4.2 fixes the problem related to the implementation of the MixColumns operation, this version is executed in constant time (as the one in the version 4) and it does the same number of register accesses regardless of the input value. This modification effectively solves the problem of the value of MSB that can be extracted using simple EMA (the execution of `BRCC` vs. `EOR`, recall Section 8.2). However, we would like to point out that the authors were unaware of the problem at it was solved unintentionally and by chance.

The updated version also uses the RSM technique, the new masks used in version 4.2 are the following 16 values:

[9] `http://www.dpacontest.org/v4/42_doc.php`

```
0x03, 0x0c, 0x35, 0x3a, 0x50, 0x5f, 0x66, 0x69,
0x96, 0x99, 0xa0, 0xaf, 0xc5, 0xca, 0xf3, 0xfc.
```

We can notice that these values were obtained by applying an exclusive-or between masks used in DPA Contest 4 and the constant `0x03`. As a result all masks from the version 4.2 have the same problem i.e., their combinations are biased as combinations of masks from the version 4 of the contest. Thus, version 4.2 of the DPA Contest could be attacked by exploiting the same bias. However, in case of DPA Contest 4.2 an attacker also have to deal with shuffling.

8.5 Summary

We showed how a debugger for microcontrollers, such as SimulAVR, can be transformed into a simple low-level of abstraction simulator for side-channel analysis. Note, that this is a simulator that deals with a narrow scope of phenomena i.e., we are only dealing and simulating leakage related to memory transfers (read and write accesses to any memory unit). Nevertheless, we were able to show that even such simple simulation can be very useful for side-channel analysis. Our tool Savrasca can be improved by introducing additional models that encompass more details about the simulated microcontroller e.g., simulating leakages resulting from the ALU as well as from data transfers on the internal buses.

With the help of Savrasca we were able to detect an issue in the implementation of AES that is used in DPA Contest. Thus, we showed that detection of issues related to timing differences and differences in the control flow of operations can be done automatically using our method based on simulations. The detection of the issue in this implementation pushed us towards further analysis of the masking countermeasure that was used in the AES-RSM implementation of DPA Contest 4. As a result, we have found a novel problem related to the choice of masks in the RSM scheme. We successfully used it in order to analyse DPA Contest 4 reference traces and found that it could be exploited: an attacker can retrieve secret information using a first order side-channel attack even with very simple distinguishers (DoM). The vulnerability is related to the choice of values for precomputed masks in AES-RSM implementation. Masks in DPA Contest 4 are unbalanced which leads to a bias and first order leakage when masked values are combined in MixColumns.

Our analyses show that a leakage can occur and can be exploited by observing a combination of two or even four bytes of the state. Such combination happen in block ciphers very often, since every single bit of the output should depend on every bit of the plaintext and of the key; thus block ciphers use operations that combine different parts of the internal state. Generally, during an implementation of a masking scheme the programmer has to pay attention and make precautions in order not to combine the mask and the masked value (to avoid accidental leakage). Our results show that even a combination of two seemingly unrelated masks can be dangerous in case of precomputed values such as used in AES-RSM scheme. The rationale is that in LEMS schemes all masks are related through the choice of masks among all possible values.

We specifically chose on of the simplest and weakest forms of side-channel attacks (simple distinguisher, only one bit of the output is considered with few leakage points) to emphasise how important this issue is: even in this scenario we achieve better success rate than some previous works. However, our attack can be definitely improved by focusing on several (biased) bits at a time, by combining it with other attacks on the DPA Contest, by choosing a better distinguisher as well as through use of more focal points (by targeting more operations).

This issue that we analyse was not discovered by previous analysis, since they mostly focus on a single byte of the state while our work focused on combinations of bytes. However, we would like to point out that such issues can potentially be discovered using other analysis techniques. For example, the paper by Moradi *et al.* [MGH14] uses a leakage detection technique that can potentially be used to detect the leakage that we identify. Normally their method is particularly well suited for analysis of single bytes (or binary words of other length). In order to analyse combinations of words or larger parts, their method could be adapted: if we were to use it to detect leakages that occur only after a combination of several masks (as it happens in e.g., MixColumns) it would require to go through all possible combinations. However, such adaptation will quickly result in "combinatorial explosions" i.e., rapid growth of the number of tests (that have to be performed) due to the number of possible combinations. In this particular case, our findings concern combinations of 2 and 4 bytes, detecting these issues using the method described by Moradi *et al.* would require to run their tests $\binom{16}{2}$ and $\binom{16}{4}$ times (respectively 120 and 1820). Moreover, since the experimentalist does not know in advance what combinations (which bytes and how many bytes are concerned)[10] one would need to run 2^n repetitions of their tests in the worst case scenario (for a state of n bytes).

As the first step towards the improvement of LEMS schemes we showed how to correct the implementation of AES-RSM in DPA Contest in order to effectively remove the bias in the combinations of masks that we found and exploited. Moreover, we also gave a set of examples of masks that could be used in order to do this improvement. In this work we provided some hints on how to fix a particular issue that we exploit, but these problems have to be addressed more generally (for AES, including decryption, and for other block ciphers) and in more details in future works. We would like to emphasize the fact that the choice of masks in LEMS countermeasures can lead to non-obvious leakages when masked values are combined. Thus, selection of mask sets is a non-trivial task that has to be studied more in depth. This choice depends on the masking scheme (number of masks and the way they are used) and it also depends on the particular algorithm (different parts of the internal state are combined in different block ciphers). Most interesting and promising directions for future works include two ideas: (1) finding a way of detecting leakages and biases that appear after a combination of several parts of the internal state, and (2) more generally, studying in depth the choice of masks for LEMS.

Overall, our results suggest that use of simulators is beneficial for leakage detection in at least two ways: (1) we were able to identify a problem that was not detected by previous works and moreover, (2) we did it without actually analysing the power traces, only by looking at the code using automated methods. Afterwards, we anal-

[10]We may try to get an idea on how many and which bytes are conserved by analysing the combinations that happen in the algorithm, but those are not the only combinations that happen in an actual implementation, recall Chapter 7.

ysed the actual power traces in order to confirm our findings and to show that the found issues are indeed exploitable.

Chapter 9

Conclusions

Cryptography provides us with encryption — a tool that allows to ensure confidentiality of information and ultimately gives us the method of secure communications. Unfortunately, simply having a secure algorithm is not the end of the story. Even if cryptographers can provide a theoretically sound encryption algorithm, creating a secure final product that uses this algorithm can be very tricky. Indeed, when an encryption algorithm is implemented it goes from abstract mathematical world to the real physical world and the cryptographic device that implements it brings new properties into the algorithm. These properties, such as power consumption — the main focus of this work, can unintentionally leak secret information that is handled by the device. An attacker can take advantage of these information leakages by measuring the physical properties of the target cryptographic system and thus extracting the encryption key from a seemingly secure implementation using side-channel analysis.

Side-channel attacks belong to one of the strongest types of attacks against cryptographic systems. Nowadays they are much more powerful than classical cryptanalysis against modern encryption algorithms. At the same time, side-channel analysis is one of the most complex kinds of issues that developers and security evaluators want to analyse. It is very complex because a side-channel attack involves many steps, choices and parameters which are related to different domains of computer science, mathematics and engineering. Thus, there exist many types and flavours of side-channel attacks and any one of them presents a potential threat against a cryptographic system. Therefore, an evaluator of a cryptographic system (with respect to its security) is faced with a big challenge of verifying that the given system is secure i.e., does not have security flaws and can resist side-channel attacks.

Security evaluation of any system is a non-trivial task since an evaluator has to check and show that the system is well protected against *all* known attacks, while an attacker is satisfied if they can find *one* security flaw. Moreover, attackers can sometimes find and exploit new, previously unknown issues that evaluation lab did not test. Performing security evaluation of cryptographic systems from the perspective

of side-channel analysis presents evaluators and developers with an additional problem: in order to test a cryptographic device for its resistance against side-channel attacks the device has to be complete. Indeed, since we are dealing with attacks that take advantage of physical properties of the device, we need the physical device in order to test it. It is not a huge problem in itself, however if a side-channel related security flaw can be discovered at the very last step of the development process then the manufacturer might have to restart a substantial part of the design in order to deal with discovered issues, which in its turn increases development costs. Therefore, developers and manufacturers of cryptographic systems can benefit from methods that allow them to discover issues related to side-channel attacks on early stages of development of cryptographic implementations.

One way of trying to ensure the absence of security issues during the development process is using provably secure countermeasures against side-channel attacks. Unfortunately, even countermeasures with security proofs such as masking are frequently broken in real implementations, it happens because of assumptions used in the proofs (value-based vs. distance-based leakage), due to unaccounted physical phenomena (e.g., glitches) and because of issues related to the implementation of those masking schemes (such as choosing values for a Low-Entropy Masking Scheme (LEMS)). Therefore, this approach does not always yield the expected result in practice, partially due to human errors related to the amount of different centres of attention that the developer has to keep track of. Thus, the use of good security policies and development methods could also benefit from additional continuous side-channel analysis on all stages of development. In our work we suggest that simulations can be used to achieve this goal.

There already exist methods (attacks and analysis techniques) that can be used to evaluate the security of cryptographic systems using power traces available at the final stage of development. The same analysis methods can be used on simulated traces that we can generate using intermediate representations (versions) of the developed system which are available on any stages of its development. Through this work we suggested a number of simulation techniques on different stages of development that can be used in order to generate simulated traces that model the real power traces. Ultimately we were able to provide 3 automated and opensource tools based on simulations.

First of all, in Chapter 5 we were able to show that simulations are useful for side-channel analysis in general due to their advantages over real experiments (full control over the system, high reliability and speed). We also emphasised that even though several simulators were presented to the scientific community, the majority of them remain unavailable and thus cannot be used or compared among them. Moreover, researchers are already using simulation techniques, but they tend to adopt disposable pieces of code designed to be used only once. Therefore we can conclude that despite the need and benefits that the community can get from use of available simulators,

this domain remains underdeveloped. Therefore, we made first steps towards filling in this gap by providing automated simulation tools for the community and showing how evaluators can use them.

The first tool that we presented in Chapter 6, called SILK, is meant to be used at the earliest stages of analysis of cryptographic algorithms and countermeasures against side-channel attacks. Its goal is to generate simulated traces given a high-level description of the leakage and the `C++` source code that we want to study, it produces traces that contain information on the dynamic power consumption (related to the data dependant part) and the noise. Being a very high-level of abstraction simulator it is very generic and flexible. We showed how it can be useful for comparative analysis of S-boxes, which ultimately results in new approaches that can be used for building S-boxes. We also showed how SILK can be used for comparing countermeasures on the example of shuffling schemes. We were able to show that simulations based on very simple and abstract models are well representative of real experiments in terms of relative strengths of attacks (or countermeasures) i.e., two attacks will have similar success rates when performed on simulated or real power traces. However, this approach is not well suited for detection of implementation flaws in a specific implementation, mostly due to the fact that the vast majority of device specific and implementation specific details are not "visible" or available at this layer of abstraction. Thus, we advertise the use of this tool for preliminary analysis of countermeasures described in higher abstraction layers and for tests and development of new attacks and preprocessing techniques. But most importantly, we showed that SILK can be used to perform massive comparative studies that are very hard to execute using only physical experiments due to time and cost constraints. Thus we conclude that it is possible to gain several orders of magnitude in terms of speed of side-channel analysis while using tools such as SILK for preliminary studies of algorithms.

Several questions related to high-level of abstraction simulations remain open. First of all, we do not know how to choose a good set of parameters (such as the leakage function) for such simulations and whether it is possible to build such high level of abstraction simulators to be device-specific. Other open problem is related to the fact that some issues can never be discovered using analysis based on high-level of abstraction simulators since such abstract models do not encompass all features of the final product and, a security flaw can potentially be related to (i.e., caused by) a very low-level feature. Thus, it would be interesting to know the limits of very high-level of abstraction simulations and to find out which type of issues *cannot* be discovered using only high-level of abstraction simulators. The same question about the limits of side-channel leakage that can be detected on *each level of abstraction* (recall Section 5.2) remains as an open problem. Finally, it is interesting to know how to build better and easier to use high-level of abstraction simulators. Such study requires more different simulators that use the same type of input and provide same kind of output but run the simulation differently (apply different models).

The second tool presented in this work in Chapter 7 is called Ascold. This tool works by processing slightly lower-level of abstraction information (compared to Silk) — assembly code. The goal of this tool is to detect some of the side-channel leakages by analysing the assembly code written by a developer. Ascold outputs all lines of code that can break the Independant Leakage Assumption (ILA), it also prints a message explaining the nature of the detected problem. Thus, it can be used during the development process, before the code gets compiled and loaded into the microcontroller. Ascold has two main distinctive features compared to many other simulators with respect to its structure and functions: (1) Ascold is build based on extensive profiling of a specific model of a microcontroller but it does not output simulated traces, (2) it can directly pin-point the location of hazardous operations related to misuse (poor implementation) of a masking scheme, thus it is a checker. Findings presented in this part of our work showed that creating a correct masking implementation that does not leak is far from being trivial and that it requires to keep track of huge amount of potential issues that can result in breaching the independent leakage assumption (which is required to build a masking scheme). The rationale is that a developer will most likely not be able to implement a masking scheme without forgetting about one of the ILA breaching issues that we discovered, thus an automated tool is required to check the assembly code. Moreover, we were able to provide such tool.

Ascold is a device specific tool since all information that was used to build it was extracted from one model of microcontroller. Therefore, several questions remain open. More tests have to be performed on other devices (including different architectures) to find out whether these results are similar on them. Moreover, we do not know if more similar effects exist and how to test their absence or existence. However, the main question related to this type of tools if the knowledge about the cause (underlying hardware details) of each ILA-breaching effect. Opensource hardware platform can shed a lot of light on this issue and thus will be extremely beneficial to the entire community. Another extremely interesting question that should be addressed is how rules used in the Ascold tool could be integrated in a compiler in order to provide a fully automated ILA-enforcing compiler assisted masking.

The third tool that we created during this work is called Savrasca and it is described in Chapter 8. This tool works on even lower-level type of input — compiled executable files. Savrasca uses a compiled binary as its input and produces power traces similar to the ones produced by Silk, but using a lower level knowledge: fully compiled assembly code and the knowledge of the model of microcontroller with user-defined leakage functions. Thus, its goals is to provide traces that can be used on last steps of the development process in order to evaluate the security of the implementation. Savrasca is device specific, but it can support multiple models of AVR microcontrollers. It allows to work with the lowest-level inputs available during the development process of a cryptographic implementation, just before the code gets

loaded into the microcontroller. Thus, it can already get to the analysis of very specific low-level issues related to side-channel analysis. Using this tool we were able to detect a previously undiscovered issue in an implementation (from DPA Contest) that was analysed by several dozen researchers over more than 3 years. We have discovered this problem in matter of minutes using our automated tool by analysing only the compiled binary file. Moreover, this discovery pushed us to uncover more advanced issue related to the same implementation and confirm all our findings on real power traces. In addition to that, we were able to show how to correct the discovered issue. The rationale is that having automated tools significantly increases chances and abilities of evaluators to detect implementation flaws.

Even though Savrasca uses a very low-level input (compiled binary) we used it with very simple leakage models. A very interesting question arises: what leakage features should be included in such simulators without rendering them too cumbersome to use while improving their accuracy (resemblance) with respect to real power traces. Moreover, there are actually two ways of improving the accuracy of such simulators: first way would be simply profiling each instruction of a device and integrating those profiles into the simulator, the second would be to use low-level models to build a more accurate trace; for example, SPICE simulations can be used on parts of the circuit alongside with other higher-level models (as in SCARD) to improve the overall accuracy of the simulation. While the first approach is simpler it does not provide us with the knowledge on ”what” is causing a leakage. The second approach, on the other hand, does give us more clues on how to solve side-channel leakage issues, but it requires the knowledge on the underlying hardware (that manufacturers do not like to share).

Overall, we were able to provide 3 tools that can be used at different stages of development of cryptographic systems: from the very creation and analysis using the most abstract models (Silk), through the process of checking implementation specific issues (Ascold) to the very last stage before the code gets into the microcontroller (Savrasca). Two of the developed tools can generate simulated traces and the third one can run a simulation for automated leakage detection (based on the knowledge previously extracted from the device). Thus, Silk and Savrasca can actually be used with any of the existing leakage detection techniques that are usually applied on real power traces, while Ascold is useful during the development itself in order to help the developer in writing the code that enforces the ILA. Table 9.1 shows the information on our tools in the format presented in the survey in Table 5.2.

Table 9.1 – Simulators developed for side-channel analysis.

Name	Year	Input	Type	Main purpose	Leakage models	Stage	Availability
SILK	2014	C++ code	G	Comparative analysis	User defined	A0, S1	✓
ASCOLD	2017	AVR assembly & YAML config.	V	Detection of issues in masking implementations	Profiling	S2	✓
SAVRASCA	2017	Compiled binary	G	Evaluation of DPA-resistance	User defined	S3	✓

G — generator, V — verifier (checker).

As the result of our work, we can answer the question we asked ourself at the beginning: the robustness of cryptographic devices against side-channel attacks can and should be evaluated in early stages of development, it can be done using simulations that can either immediately detect a security flaw or output a set of simulated traces resembling real traces, which can in their turn be analysed with any existing leakage detection and security evaluation techniques.

As a bonus contribution to our main goals of providing simulation tools for side-channel analysis we were able to analyse and work on novel topics related to side-channel analysis. We showed a set of novel problems related to masking implementations: the ILA-breaching effects and the problem of choice of values in LEMS. We also developed a range of scalable shuffling countermeasures that can be selected and tuned depending on the requirements of the system. Moreover, we also showed a novel approach in the design of S-boxes. Our analysis of masking schemes and their implementations with ASCOLD and SAVRASCA tools highlighted that even a theoretically sound masking scheme can be broken due to the issues related to the concrete implementation of the masking scheme (ILA-breaching effects or bad choice of masks). Issues in the masking scheme can be found using tools such as the one presented by Barthe *et al.* [BBD+15], however they do not detect all issues that arise at the implementation stage. Thus, it is important to develop tools and methods for the verification of secure *implementations*, such as the Paioli tool [RGN13].

In addition to the unsolved problems mentioned above, we would like to point out some interesting topics for future research. During this work we have focused our attention on software implementations of block ciphers in microcontrollers in case when the evaluator can use the microcontroller as a black box i.e., the evaluator does not have full knowledge of the internals of the hardware. Our work can be extended for other types of hardware such as FPGAs and ASICs: the idea is to build side-channel analysis simulators based on hardware description languages such as `VHDL`. Note, that in this case the developer knows more about the hardware structure of the cryptographic system thus predictive capabilities of such simulators can potentially be greater that in case of microcontrollers. Another way of extending this work would be to work on open hardware platform which should allow us to understand why some types of leakage occur and where (which part of the hardware) they originate from. Both of our trace generators (SILK and SAVRASCA) use relatively generic and abstract user defined models, an interesting approach is to apply profiled models in case of such simulators. From our perspective, a very powerful contribution to the domain of side-channel analysis could be done by implementing a simulator that can be parametrised using profiled models, in this case any researcher can contribute to it by providing a new profile: either for a new model of microcontroller or by creating a better profile for an already profiled device. Through our analysis, we used our trace generators with relatively simple user defined models, this domain can definitely benefit from more complex models that can give more information and allow

to detect more security flaws in early stages of development. Moreover, our tools do not encompass all phenomena that are actually studied in side-channel analysis, creation of new available simulators that also model glitches and fault injections are very promising direction for future works. Finally, we were not able to compare our simulators with other existing tools due to their non-availability. We made all our code available, we would like to emphasise that creating more available simulators will allow other researchers to compare them and to improve existing tools or build new and better ones.

We would like to point out an important issue and offer a cautionary note related to the detection of security flaws related to side-channel attacks in real products. Even if many issues can be detected during the development with the help of simulators and automated tools it is critical to perform the concluding security evaluation on the final version of the device. It is very important because simulators do not offer a hard guarantee on providing results that fit the reality perfectly thus, some security flaws can only be found through the analysis of the final product. The main reason lies in the fact that simulators use models and a model, however good it appears to be, is always different from the reality.

To sum up, we provided 3 simulation-based tools that can be used for side-channel analysis on different stages of development of cryptographic systems. We showed how these tools can be used on multiple examples including protected and unprotected algorithms. Through our work we were able to show the usefulness of such approach by discovering previously unknown issues and by showing how much time can be saved by using simulators for side-channel analysis. We hope researchers as well as developers of cryptographic devices will integrate our results in their processes.

Appendix A

Silk example

```
1  #include <iostream>
2  #include "../silk/silk.hpp"
3  using namespace std;
4
5  const U8 sbox[] = {
6    0x0C, 0x05, 0x06, 0x0B, 0x09, 0x00, 0x0A, 0x0D,
7    0x03, 0x0E, 0x0F, 0x08, 0x04, 0x07, 0x01, 0x02};
8
9  void mix(U8 msg, U8 key){
10   U8 state = msg ^ key;
11   state = ((sbox[state >> 4])<<4) + sbox[sbox[state && 0xF]];
12 }
13
14 int main(){
15
16   //set up the parameters
17   U8::setLeakageFunction(hammingWeightOut);
18   Tracer::setLeakagePointsNbr(10);
19   Tracer::setLeakageOverlap(2);
20   Tracer::setLeakageDistributionFunc(sinMul, M_PI);
21
22   Tracer::clearTrace();                  // clear the trace
23   mix(0x08, 0xB7);                       // call your function
24   Tracer::traceToFile("simulation.csv"); // save the simulated trace
25
26   return 0;
27 }
```

Listing A.1 – Silk example: setting up the parameters.

Appendix B

Success rate of S-boxes using simulations

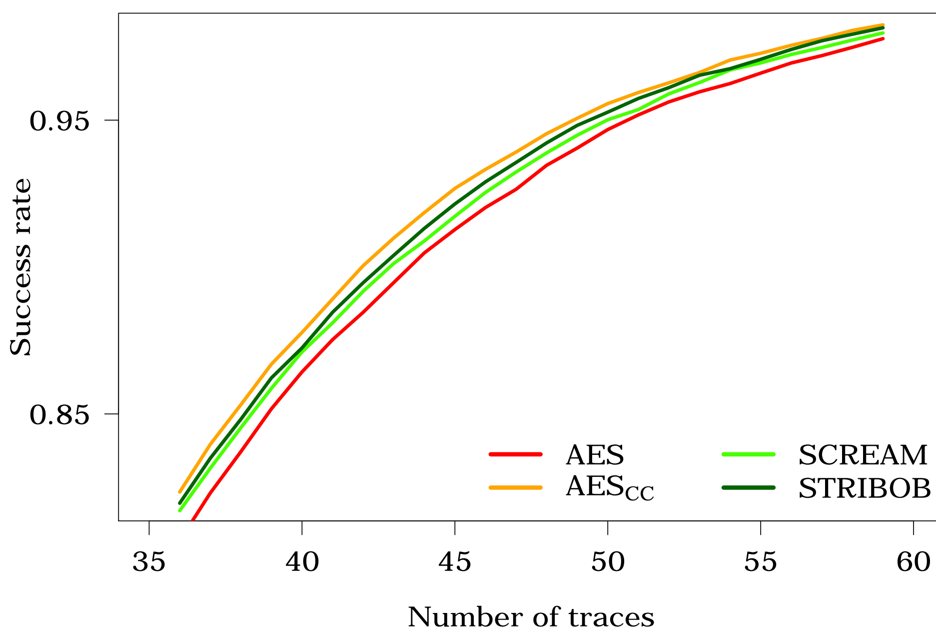
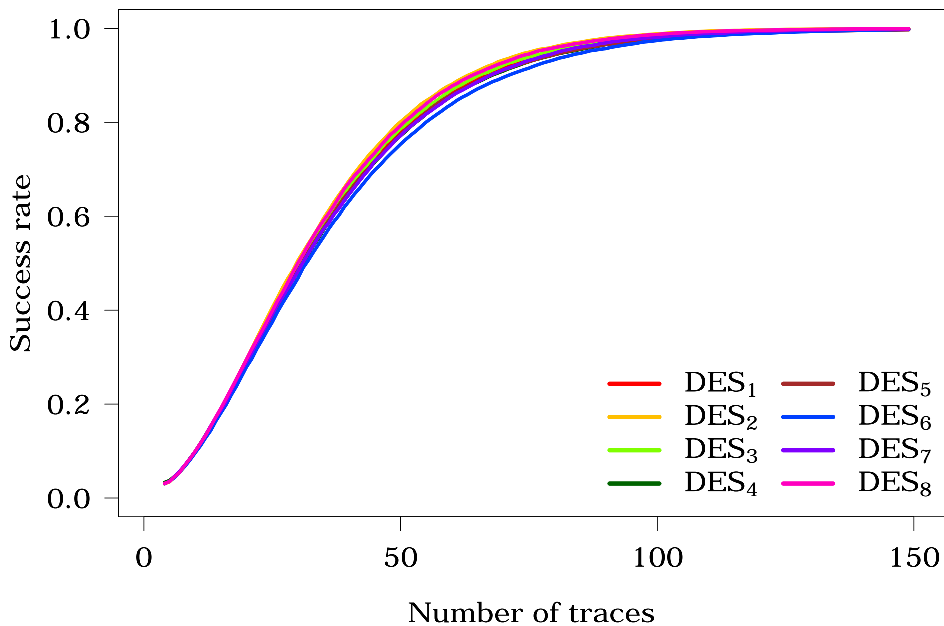
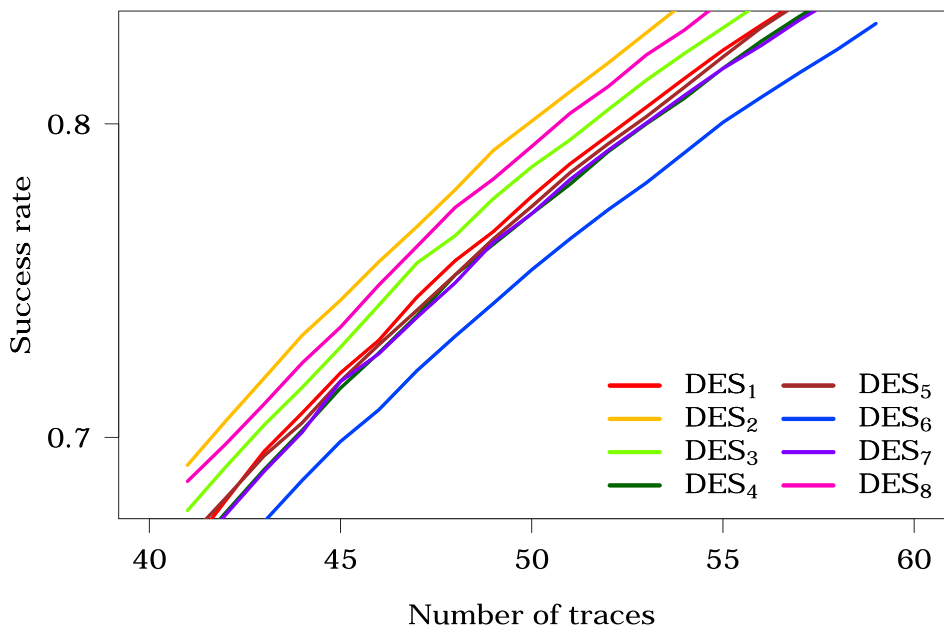


Figure B.1 – Zoom on the success rate of CPA on 8×8 S-boxes.



(a) Success rate.



(b) Zoom.

Figure B.2 – Success rate of CPA on 6×4 (DES) S-boxes using simulations.

Appendix C

S-boxes generated using evolutionary computations

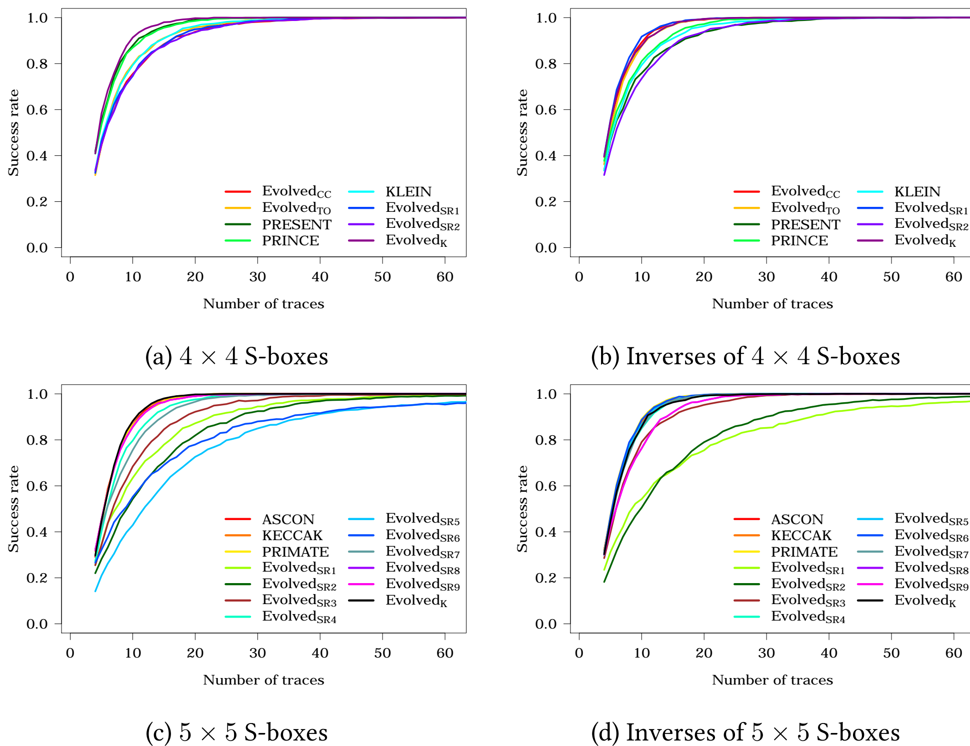
Table C.1 – Evolved S-boxes when considering TA with a model extracted from an ATmega328P microcontroller. Values of S-boxes are given in hexadecimal format.

Size	Name	S-box
4×4	EvolvedTA$_{SR1}$	0,5,7,C,A,6,2,4,9,8,B,F,D,E,1,3.
	EvolvedTA$_{SR2}$	4,2,D,E,B,1,6,5,7,8,3,A,F,0,C,9.
	EvolvedTA$_{SR3}$	9,4,5,D,3,0,1,F,B,2,C,7,E,8,A,6.
	EvolvedTA$_{SR4}$	4,0,6,7,1,2,A,F,5,3,C,E,D,9,B,8.
5×5	EvolvedTA$_{SR1}$	1F,15,01,0C,14,1D,12,00,1A,09,08,17,05,0E,0B,0D, 04,18,1B,0A,13,11,06,1E,10,19,16,02,0F,07,03,1C.
	EvolvedTA$_{SR2}$	07,14,1D,11,12,02,06,13,19,0F,09,0C,1C,15,0A,08, 01,0B,1F,0D,03,17,1E,05,04,1B,0E,00,1A,18,10,16.
	EvolvedTA$_{SR3}$	1F,01,02,1A,04,1B,15,0C,08,1E,17,07,0B,1C,18,09, 10,0D,1D,06,0F,13,0E,14,16,03,19,0A,11,05,12,00.
	EvolvedTA$_{SR4}$	1F,01,02,1E,04,0E,1D,15,08,13,1C,05,1B,14,0B,06, 10,0F,07,1A,19,12,0A,03,17,0D,09,11,16,18,0C,00.
	EvolvedTA$_{SR5}$	01,13,11,16,0F,03,08,10,0B,1A,02,1B,0C,1E,17,12, 19,0D,14,00,05,04,18,07,1F,1D,06,15,0A,1C,0E,09.
	EvolvedTA$_{SR6}$	1F,15,18,05,01,06,08,0B,12,02,17,0D,03,1C,04,16, 0F,1B,09,13,0C,11,1E,1A,00,19,0A,07,1D,14,0E,10.

Table C.2 – Evolved S-boxes when considering CPA. Values of S-boxes are given in hexadecimal format.

Size	Name	S-box
4×4	Evolved$_{SR1}$	2,4,8,0,F,B,7,D,6,5,E,3,1,9,C,A.
	Evolved$_{SR2}$	F,E,0,A,1,8,9,B,7,6,4,C,5,2,3,D.
	Evolved$_{K}$	0,F,1,9,B,5,8,2,E,3,C,6,D,4,A,7.
5×5	Evolved$_{SR1}$	1E,07,15,02,0E,09,19,04,17,12,0B,08,1C,0A,1D,06, 0C,1B,05,0D,00,14,18,1F,10,13,11,1A,01,16,03,0F.
	Evolved$_{SR2}$	15,02,1F,0A,19,11,1B,12,08,0E,0C,07,06,0F,10,16, 13,00,17,09,1D,18,0D,03,04,1A,14,1C,05,1E,01,0B.
	Evolved$_{SR3}$	1D,15,03,02,1C,0A,0C,09,11,10,1F,0D,18,14,19,16, 06,12,0F,17,01,04,13,1B,0B,07,0E,05,1A,1E,00,08.
	Evolved$_{SR4}$	0A,1C,01,13,04,08,12,10,06,05,03,0D,02,18,09,00, 0F,1B,1A,11,14,1D,0B,0E,16,07,15,19,0C,17,1E,1F.
	Evolved$_{SR5}$	04,17,1C,18,07,00,12,19,0E,14,10,15,06,13,1F,08, 1A,11,0C,0B,05,1E,0F,01,02,1D,1B,09,0D,03,0A,16.
	Evolved$_{SR6}$	09,05,1E,1C,0D,16,14,06,07,1D,01,10,03,02,13,1F, 1B,15,08,18,04,00,0F,1A,0A,12,0B,0E,19,17,11,0C.
	Evolved$_{SR7}$	1B,13,17,16,0B,0F,0D,1A,03,06,01,09,02,14,08,11, 10,12,00,0A,1F,18,05,0C,1D,1C,04,07,0E,1E,15,19.
	Evolved$_{SR8}$	00,0E,1C,16,19,01,0D,11,13,08,02,1D,1A,17,03,0A, 07,0B,10,18,04,1E,1B,05,15,0C,0F,12,06,09,14,1F.
	Evolved$_{SR9}$	00,07,0E,0B,1C,10,16,18,19,04,01,1E,0D,1B,11,05, 13,15,08,0C,02,0F,1D,12,1A,06,17,09,03,14,0A,1F.
	Evolved$_{K}$	15,07,06,03,18,0E,04,01,0C,05,0A,16,1F,1D,19,13, 12,0F,11,1B,09,1A,17,10,08,0B,00,14,02,1C,1E,0D.

C.1 Success rate of attacks on the S-boxes



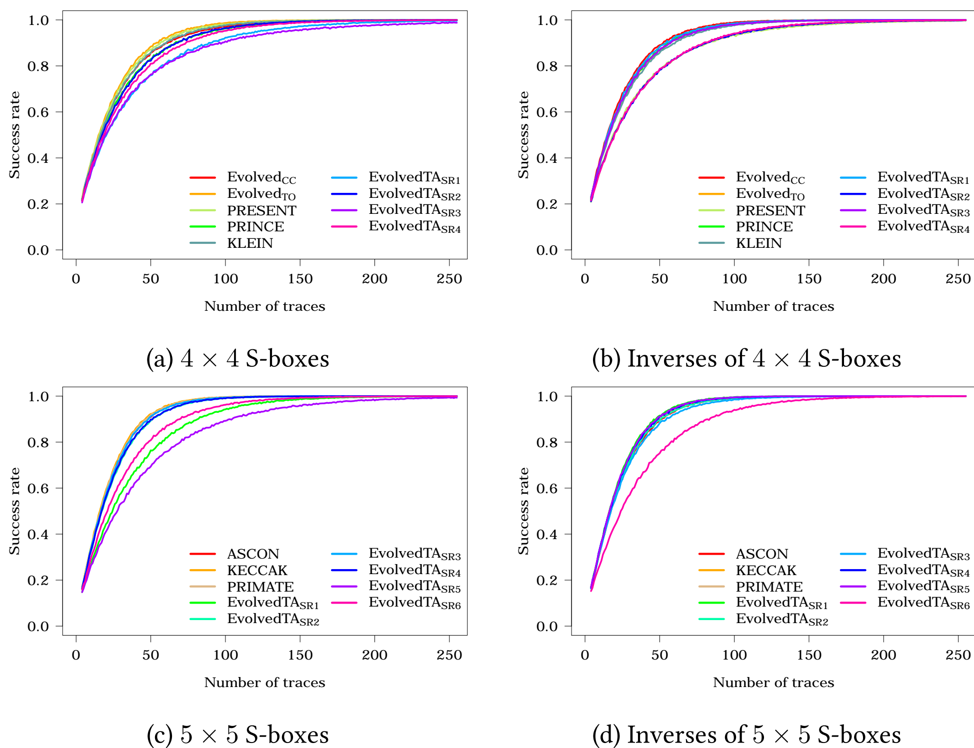
(a) 4×4 S-boxes

(b) Inverses of 4×4 S-boxes

(c) 5×5 S-boxes

(d) Inverses of 5×5 S-boxes

Figure C.1 – Success rates of CPA on S-boxes.



(a) 4×4 S-boxes

(b) Inverses of 4×4 S-boxes

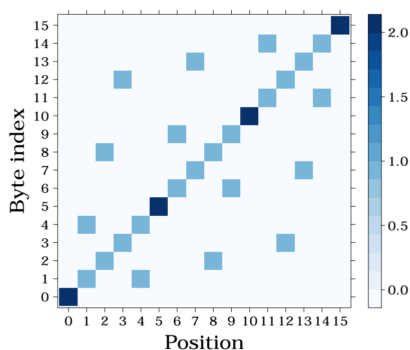
(c) 5×5 S-boxes

(d) Inverses of 5×5 S-boxes

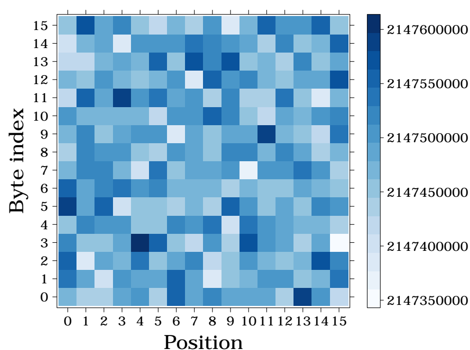
Figure C.2 – Success rates of TA on S-boxes.

Appendix D

Heatmaps of shuffling schemes

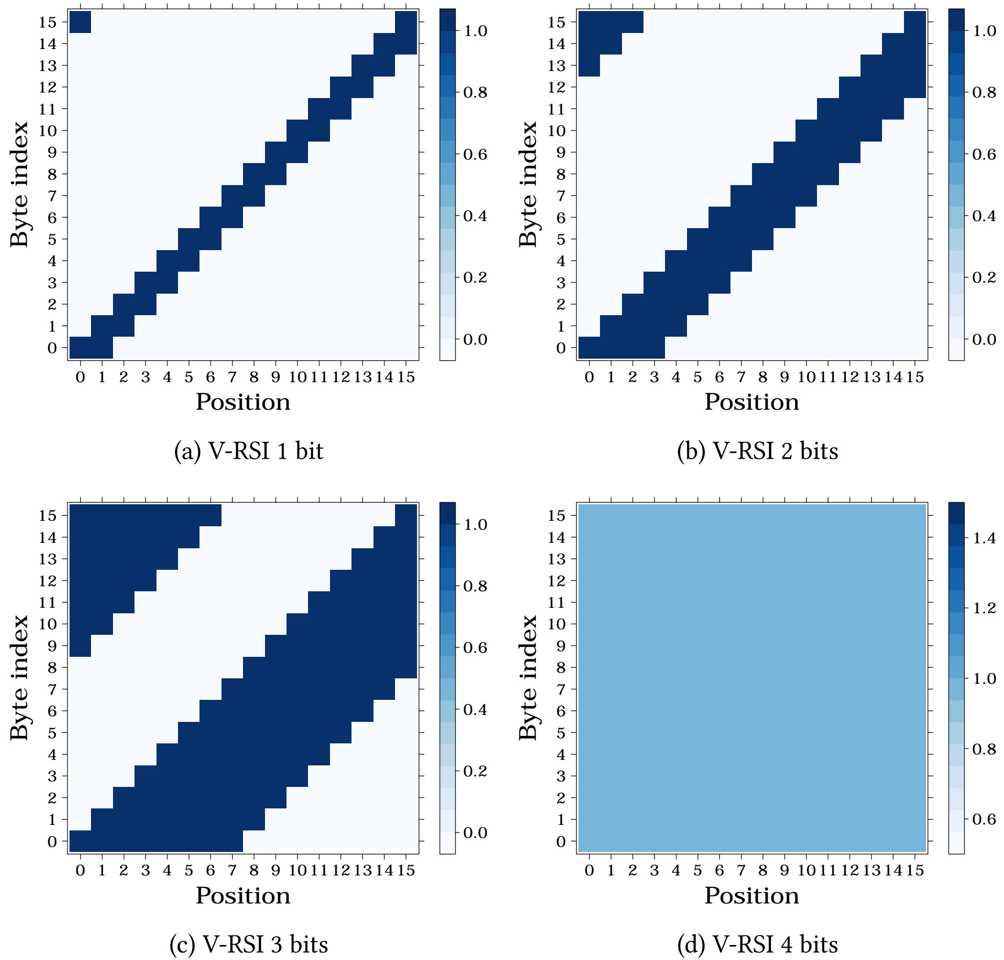


(a) 4×4 SSS with 1 bit



(b) Approximation for RP

Figure D.1 – SSS and RP heatmaps of positions when the SubBytes operation takes place for every byte.



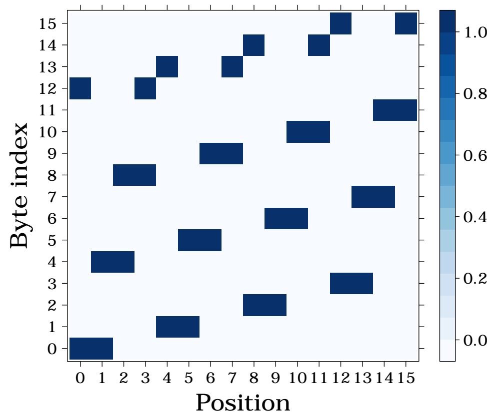
(a) V-RSI 1 bit

(b) V-RSI 2 bits

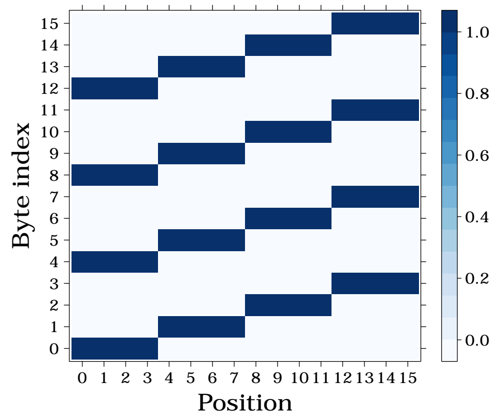
(c) V-RSI 3 bits

(d) V-RSI 4 bits

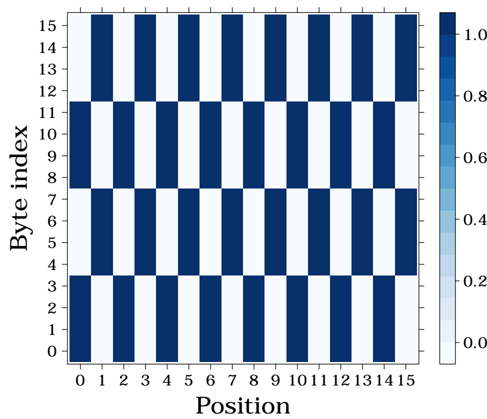
Figure D.2 – V-RSI Heatmap of positions when the SubBytes operation takes place for every byte.



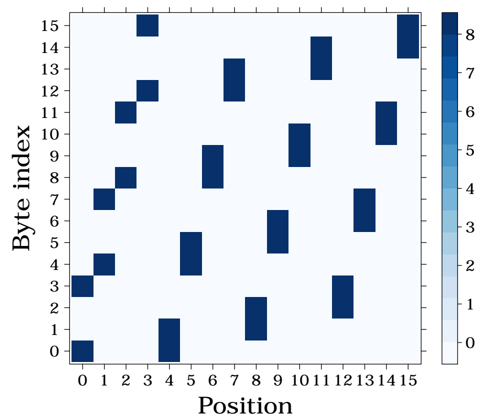
(a) M-RSI 1 bit



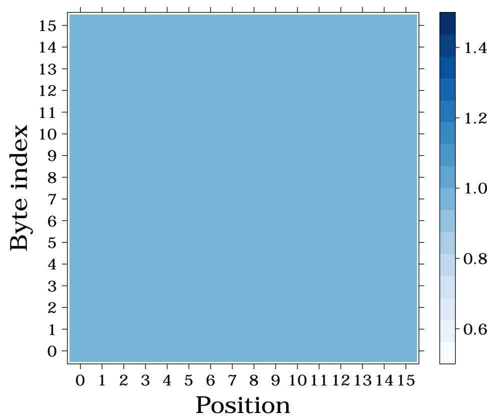
(b) M-RSI 2 bits



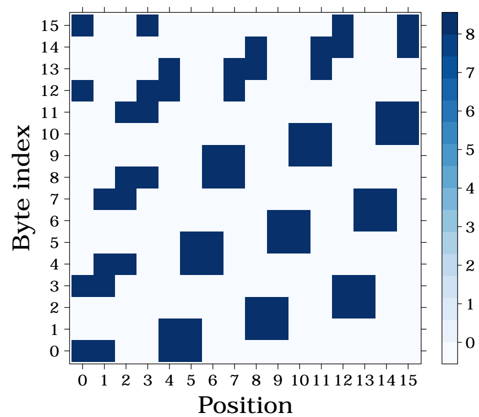
(c) M-RSI 3 bits



(d) M-RSI 4 bits

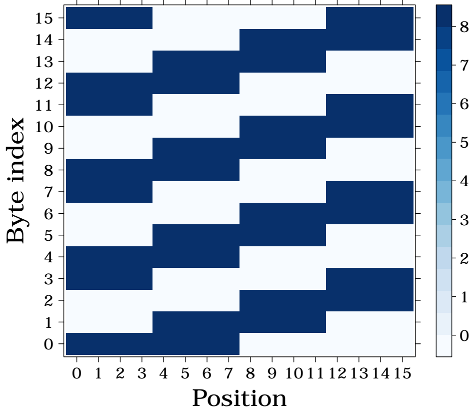


(e) M-RSI* 4 bits

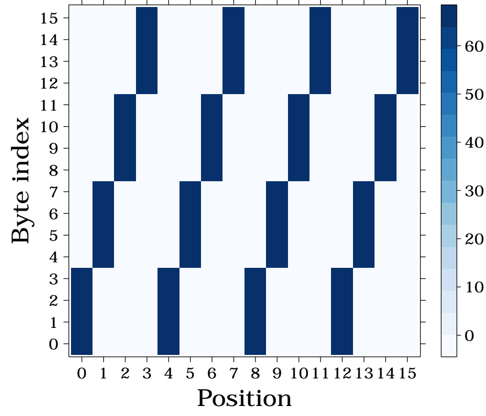


(f) M-RSI 5 bits

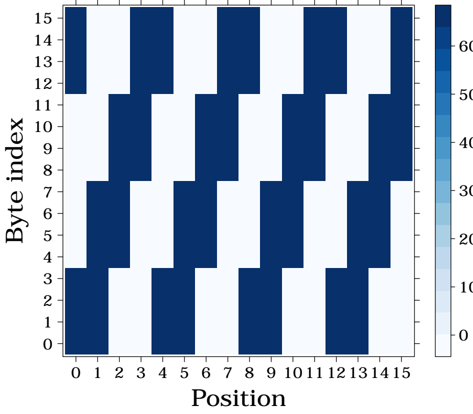
Figure D.3 – M-RSI 4×4 Heatmaps of positions when the SubBytes operation takes place for every byte.



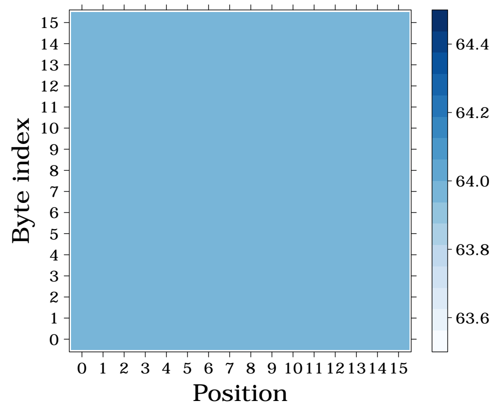
(a) M-RSI 6 bits



(b) M-RSI 8 bits

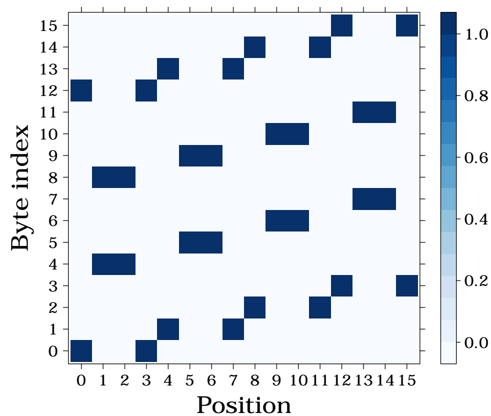


(c) M-RSI 9 bits

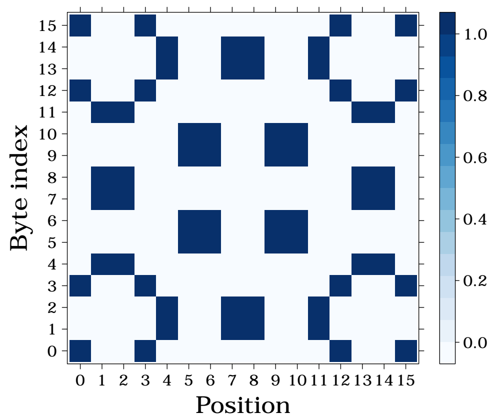


(d) M-RSI 10 bits

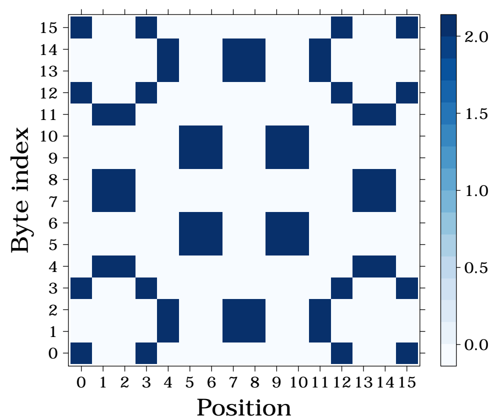
Figure D.4 – M-RSI 4×4 Heatmaps part 2 of positions when the SubBytes operation takes place for every byte.



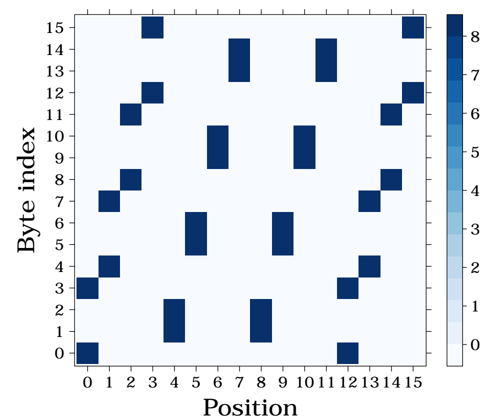
(a) M-RS 1 bit



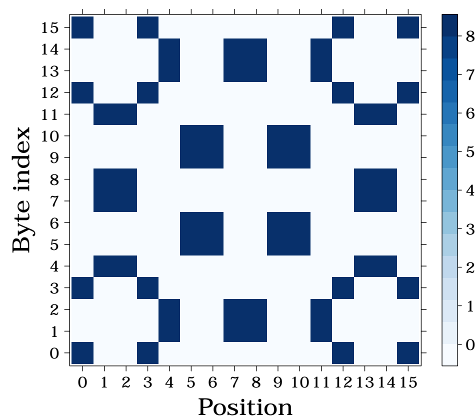
(b) M-RS 2 bits



(c) M-RS 3 bits



(d) M-RS 4 bits



(e) M-RS 5 bits

Figure D.5 – M-RS 4×4 Heatmaps of positions when the SubBytes operation takes place for every byte.

Appendix E

Success rates of attacks on shuffling schemes

Success rates of CPA (with and without preprocessing) and TA on different shuffling schemes as well as the unprotected implementation. Note, that the horizontal axis of each plot uses a logarithmic scale.

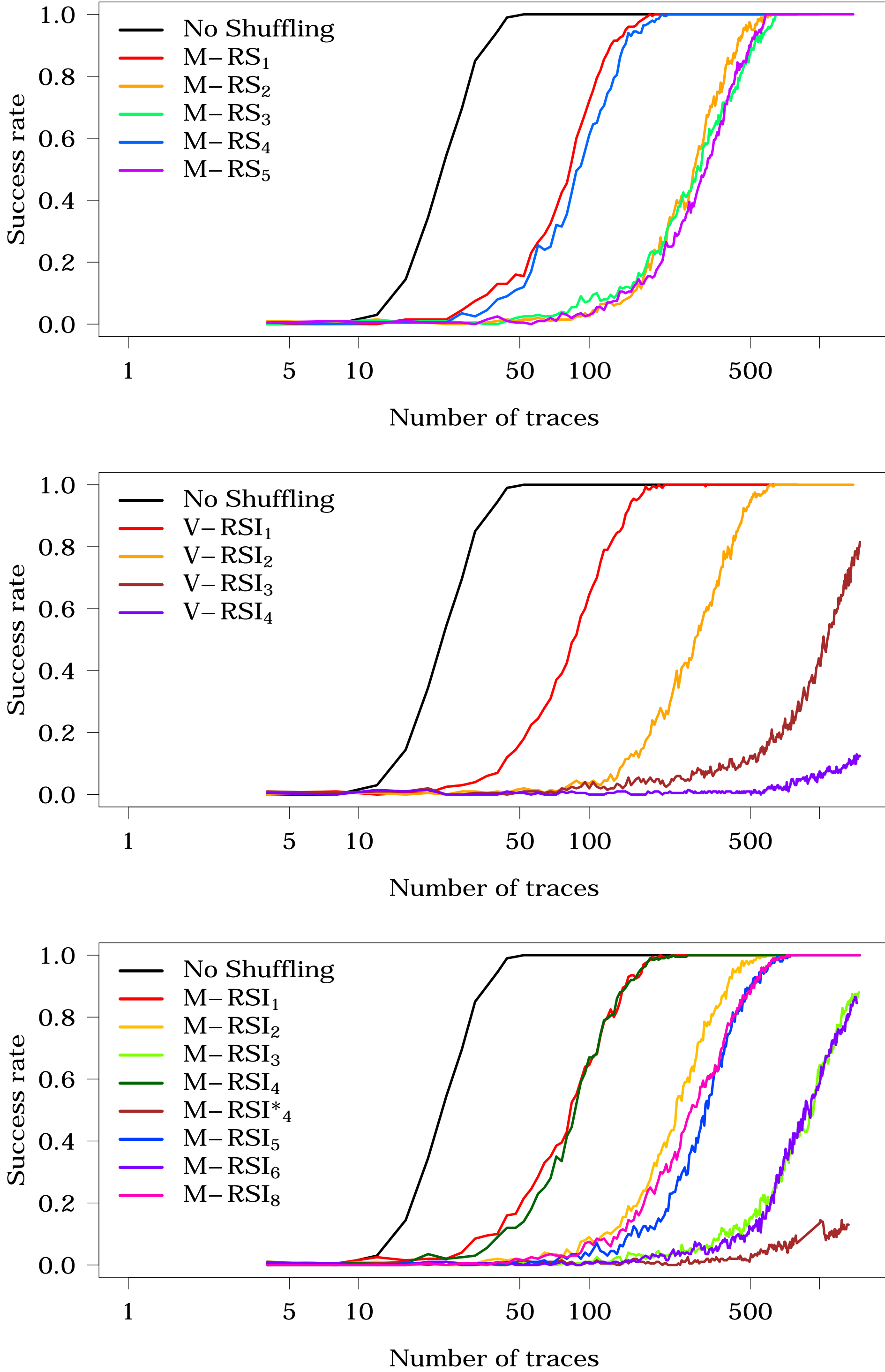


Figure E.1 – The success rate of CPA against different shuffling techniques. Horizontal axis is logarithmic.

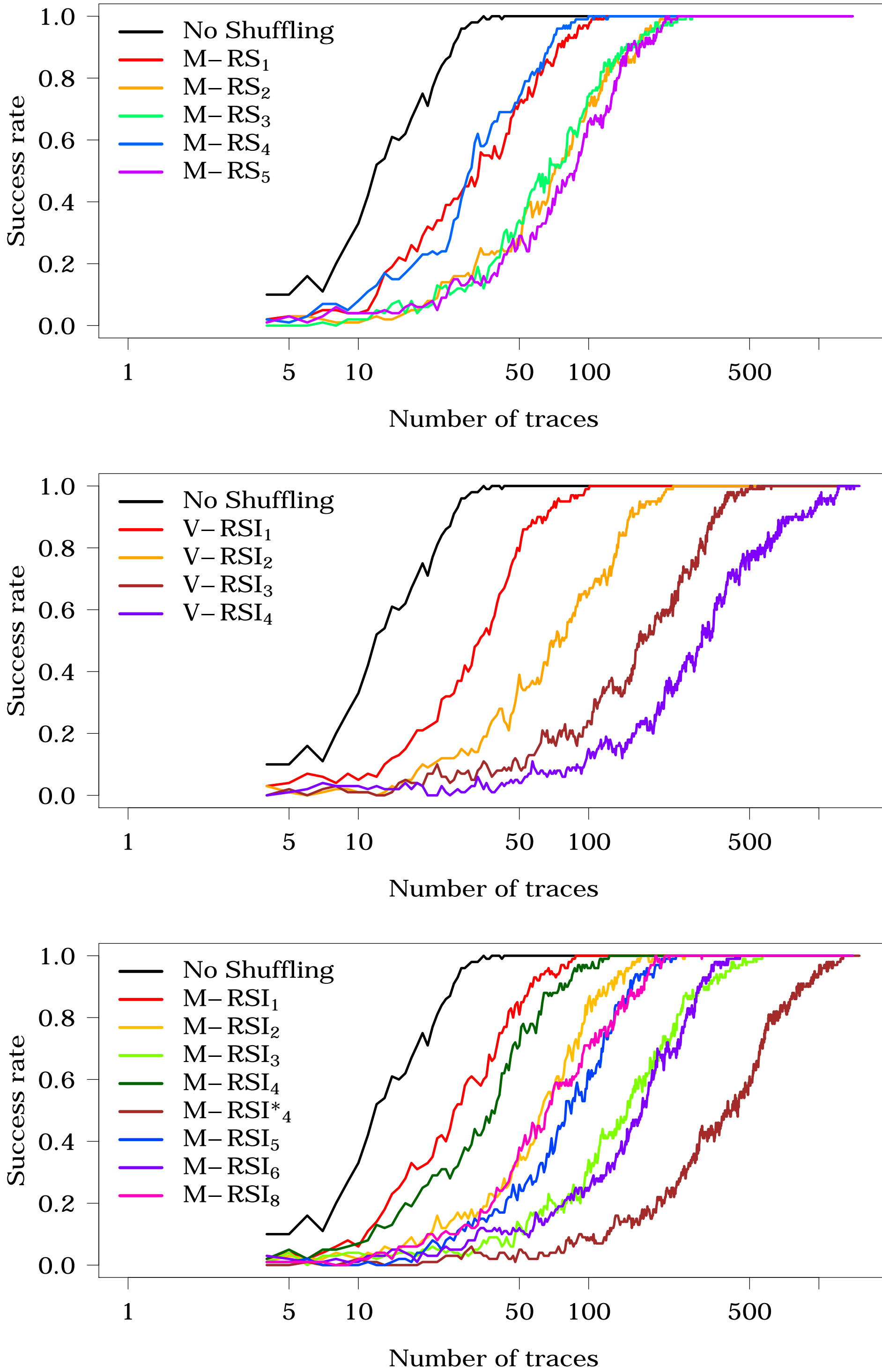


Figure E.2 – The success rate of CPA with integration preprocessing against different shuffling techniques. Horizontal axis is logarithmic.

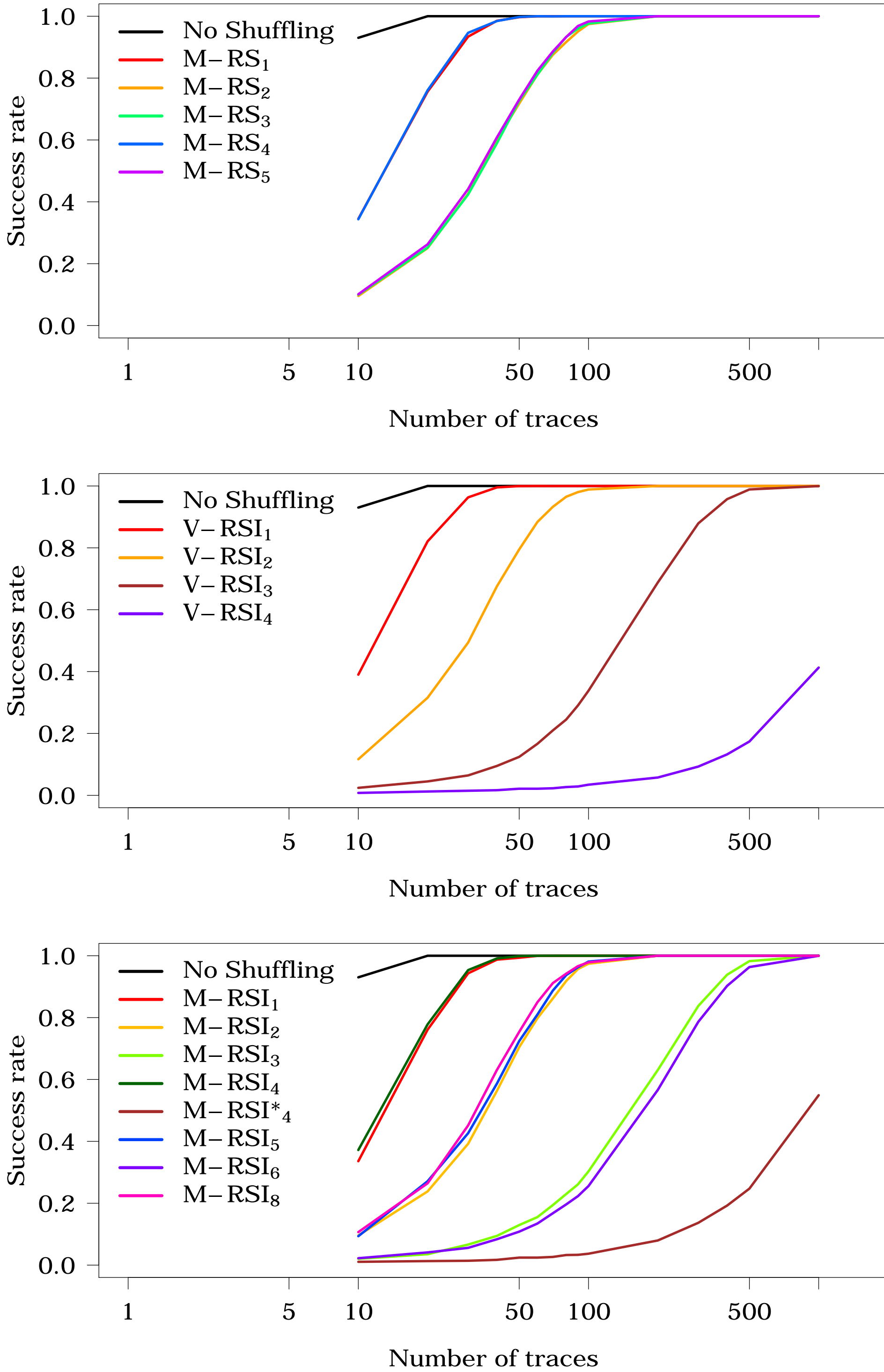


Figure E.3 – The success rate of a TA against different shuffling techniques. Horizontal axis is logarithmic.

Appendix F

ASCOLD example

Here is a small piece of code that loads 2 shares, a random value and then combined them all.

```
; loading shares a0, a1
lds r2, 0xA000
lds r3, 0xC007
; loading the random value
lds r6, 0xF001
eor r2, r3
eor r2, r6
```

Listing F.1 – Simple xor example with neighbour-registers leakage.

```
; loading shares a0, a1
lds r2, 0xA000
lds r10, 0xC007
; loading the random value
lds r6, 0xF001
eor r2, r6
eor r2, r10
```

Listing F.2 – Simple xor example without neighbour-registers leakage.

Here is the configuration file:

```
rand_list_of_addr:
    - 0xF001
mask_list_of_addr:
    - [0xA000, a, 0]
    - [0xC007, a, 1]
```

Table F.1 – Table of neighbours-registers in ATmega163 microcontroller. Value 1 (highlighted in grey) denotes the presence of leakage.

Reg	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
00	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
01	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
02	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
03	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
04	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
05	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
06	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
07	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
08	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
09	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Appendix G

List of microcontrollers supported by SAVRASCA

Here is the list of microcontrollers supported by SAVRASCA (and SimulAVR at the moment of writing). The list of devices suported by SimulAVR is availalbe on its official website: `http://www.nongnu.org/simulavr/usage.html`, the list of devices supported by SAVRASCA can also be printed by executing the tool with `--help` option.

- ATmega128
- ATmega1284a
- ATmega16
- ATmega164a
- ATmega168
- ATmega32
- ATmega324a
- ATmega328
- ATmega48
- ATmega644a
- ATmega8
- ATmega88
- AT90can128
- AT90can32
- AT90can64
- AT90s4433
- AT90s8515
- ATtiny2313

Bibliography

[AARR02] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side-channel(s). In Jr. et al. [JKP03], pages 29–45.

[ABB+14] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, and Qingju Wang an Kan Yasuda. PRIMATEs v1.02, Sept 2014. CAESAR submission.

[ABK98] Ross J. Anderson, Eli Biham, and Lars R. Knudsen. Serpent: A new block cipher proposal. In Serge Vaudenay, editor, *Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings*, volume 1372 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 1998.

[ABKT98] Ross J. Anderson, Eli Biham, Lars R. Knudsen, and Haifa Technion. Serpent: A flexible block cipher with maximum assurance. In *The first AES candidate conference*, pages 589–606, 1998.

[AES01] Specification for the advanced encryption standard (AES). Federal Information Processing Standards Publication 197, 2001.

[AFG+15] Julien Allibert, Benoit Feix, Georges Gagnerot, Ismael Kane, Hugues Thiebeauld, and Tiana Razafindralambo. Chicken or the egg - computational data attacks or physical attacks. Cryptology ePrint Archive, Report 2015/1086, 2015. `http://eprint.iacr.org/2015/1086`.

[AK96] Ross Anderson and Markus Kuhn. Tamper resistance-a cautionary note. In *Proceedings of the second Usenix workshop on electronic commerce*, volume 2, pages 1–11, 1996.

[AMM+06] Manfred Josef Aigner, Stefan Mangard, Francesco Menichelli, Renato Menicocci, Mauro Olivieri, Thomas Popp, Giuseppe Scotti, and Alessandro Trifiletti. Side channel analysis resistant design flow. In *International Symposium on Circuits and Systems (ISCAS), 21-24 May 2006, Island of Kos, Greece*. IEEE, 2006.

[And09] Philippe Andouard. *Outils d'aide à la recherche de vulnérabilités dans l'implantation d'applications embarquées sur carte à puce*. PhD thesis, Bordeaux 1, 2009.

[ARMa] ARM Holdings. Cortex-A8 Technical Reference Manual. `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0344k/DDI0344K_cortex_a8_r3p2_trm.pdf`.

[ARMb] ARM Holdings. Cortex-M4 Technical Reference Manual. `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439b/DDI0439B_cortex_m4_r0p0_trm.pdf`.

[Atma] Atmel Corporation. 8-bit AVR instruction set. `http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instruction_list.html`.

[Atmb] Atmel Corporation. Atmega-16 datasheet. `http://www.atmel.com/images/doc2466.pdf`.

[Atmc] Atmel Corporation. Atmega-163 datasheet. `http://www.atmel.com/images/doc1142.pdf`.

[BB15] Paul Bottinelli and Joppe W. Bos. Computational aspects of correlation power analysis. *IACR Cryptology ePrint Archive*, 2015:260, 2015.

[BBB+13] Pierre Belgarric, Shivam Bhasin, Nicolas Bruneau, Jean-Luc Danger, Nicolas Debande, Sylvain Guilley, Annelie Heuser, Zakaria Najm, and Olivier Rioul. Time-Frequency Analysis for Second-Order Attacks. In Francillon and Rohatgi [FR14], pages 108–122.

[BBD+14] Shivam Bhasin, Nicolas Bruneau, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Analysis and improvements of the DPA contest v4 implementation. In Chakraborty et al. [CMS14], pages 201–218.

[BBD+15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified Proofs of Higher-Order Masking. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S.

Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Wang and Sako [WS12], pages 208–225.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[BDG+13] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. In Alessandro Aldini, Javier Lopez, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, volume 8604 of *Lecture Notes in Computer Science*, pages 146–166. Springer, 2013.

[BDG+14] Nicolas Bruneau, Jean-Luc Danger, Sylvain Guilley, Annelie Heuser, and Yannick Teglia. Boosting Higher-Order Correlation Attacks by Dimensionality Reduction. In Chakraborty et al. [CMS14], pages 183–200.

[BDGN13] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. A low-entropy first-degree secure provable masking scheme for resource-constrained devices. In *Proceedings of the Workshop on Embedded Systems Security, WESS 2013, Montreal, Quebec, Canada, September 29 - October 4, 2013*, pages 7:1–7:10. ACM, 2013.

[BDPA09] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Note on side-channel attacks and their countermeasures, May 2009. `http://keccak.noekeon.org/NoteSideChannelAttacks.pdf`.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The keccak reference, 2011. Submission to NIST (Round 3).

[BE01] Mark Blunden and Adrian Escott. Related key attacks on reduced round KASUMI. In Matsui [Mat02], pages 277–285.

[Ber05] Daniel J Bernstein. Cache-timing attacks on aes, 2005.

[BFGV12] Josep Balasch, Sebastian Faust, Benedikt Gierlichs, and Ingrid Verbauwhede. Theory and practice of a leakage resilient masking scheme. In Wang and Sako [WS12], pages 758–775.

[BGG+14] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.

[BGH+16] Nicolas Bruneau, Sylvain Guilley, Annelie Heuser, Olivier Rioul, François-Xavier Standaert, and Yannick Teglia. Taylor expansion of maximum likelihood attacks for masked and shuffled implementations. In Cheon and Takagi [CT16], pages 573–601.

[BGP+11] Lejla Batina, Benedikt Gierlichs, Emmanuel Prouff, Matthieu Rivain, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Mutual Information Analysis: a Comprehensive Study. *J. Cryptology*, 24(2):269–291, 2011.

[BHMT16] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. In Gierlichs and Poschmann [GP16], pages 215–236.

[BI15] Andrey Bogdanov and Takanori Isobe. How secure is AES under leakage. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 361–385. Springer, 2015.

[Bih97] Eli Biham. A Fast New DES Implementation in Software. In Eli Biham, editor, *FSE*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 1997.

[BIT16] Andrey Bogdanov, Takanori Isobe, and Elmar Tischhauser. Towards practical whitebox cryptography: Optimizing efficiency and space hardness. In Cheon and Takagi [CT16], pages 126–158.

[BK] Elaine Barker and John Kelsey. Recommendation for random number generation using deterministic random bit generators. `http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf`.

[BK98] Alex Biryukov and Eyal Kushilevitz. From differential cryptanalysis to ciphertext-only attacks. In Krawczyk [Kra98], pages 72–88.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In Paillier and Verbauwhede [PV07], pages 450–466.

[BKM+15] Andrey Bogdanov, Ilya Kizhvatov, Kamran Manzoor, Elmar Tischhauser, and Marc Witteman. Fast and memory-efficient key recovery in side-channel attacks. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, volume 9566 of *Lecture Notes in Computer Science*, pages 310–327. Springer, 2015.

[Bla79] George Robert Blakley. Safeguarding cryptographic keys. *Proc. of the National Computer Conference1979*, 48:313–317, 1979.

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Krawczyk [Kra98], pages 1–12.

[BS90] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.

[CBG+17] Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. Does coupling affect the security of masked implementations? In Guilley [Gui17], pages 1–18.

[CC11] Liang Cai and Hao Chen. Touchlogger: Inferring keystrokes on touch screen from smartphone motion. In Patrick D. McDaniel, editor, *6th USENIX Workshop on Hot Topics in Security, HotSec'11, San Francisco, CA, USA, August 9, 2011*. USENIX Association, 2011.

[CCD00] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In Koç and Paar [KP00], pages 252–263.

[CDG+13] Jeremy Cooper, Elke DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, and Pankaj Rohatgi. Test vector

leakage assessment (TVLA) methodology in practice, 2013. http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf.

[CDN98] Gary Carter, Ed Dawson, and Lauren Nielsen. Key schedules of iterative block ciphers. In Colin Boyd and Ed Dawson, editors, *Information Security and Privacy, Third Australasian Conference, ACISP'98, Brisbane, Queensland, Australia, July 1998, Proceedings*, volume 1438 of *Lecture Notes in Computer Science*, pages 80–89. Springer, 1998.

[CDN99] Gary Carter, Ed Dawson, and Lauren Nielsen. Key schedule classification of the AES candidates. In *Proceedings of the end AES Conference, Rome, Italy*, pages 1–14, 1999.

[CEJvO02] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box DES implementation for DRM applications. In Joan Feigenbaum, editor, *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.

[CGP+12] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of security proofs from one leakage model to another: A new issue. In Werner Schindler and Sorin A. Huss, editors, *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings*, volume 7275 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2012.

[CGPR08] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, and Matthieu Rivain. Attack and improvement of a secure s-box calculation based on the fourier transform. In Oswald and Rohatgi [OR08], pages 1–14.

[CIMW15] Christophe Clavier, Quentin Isorez, Damien Marion, and Antoine Wurcker. Complete reverse-engineering of aes-like block ciphers by SCARE and FIRE attacks. *Cryptography and Communications*, 7(1):121–162, 2015.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In Wiener [Wie99], pages 398–412.

[CK13] Omar Choudary and Markus G. Kuhn. Efficient template attacks. In Francillon and Rohatgi [FR14], pages 253–270.

[CK14] Omar Choudary and Markus G. Kuhn. Efficient stochastic methods: Profiled attacks beyond 8 bits. *IACR Cryptology ePrint Archive*, 2014:885, 2014.

[Cla04] Christophe Clavier. Side channel analysis for reverse engineering (scare) - an improved attack against a secret a3/a8 gsm algorithm. Cryptology ePrint Archive, Report 2004/049, 2004. `http://eprint.iacr.org/2004/049`.

[CMS14] Rajat Subhra Chakraborty, Vashek Matyas, and Patrick Schaumont, editors. *Security, Privacy, and Applied Cryptography Engineering - 4th International Conference, SPACE 2014, Pune, India, October 18-22, 2014. Proceedings*, volume 8804 of *Lecture Notes in Computer Science*. Springer, 2014.

[CNK04] Jean-Sébastien Coron, David Naccache, and Paul C. Kocher. Statistics and secret leakage. *ACM Trans. Embedded Comput. Syst.*, 3(3):492–508, 2004.

[CPR07] Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side channel cryptanalysis of a higher order masking scheme. In Paillier and Verbauwhede [PV07], pages 28–44.

[CPRR13] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 2013.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Jr. et al. [JKP03], pages 13–28.

[CSM+17] Kaushik Chakraborty, Sumanta Sarkar, Subhamoy Maitra, Bodhisatwa Mazumdar, Debdeep Mukhopadhyay, and Emmanuel Prouff. Redefining the transparency order. *Des. Codes Cryptography*, 82(1-2):95–115, 2017.

[CT16] Jung Hee Cheon and Tsuyoshi Takagi, editors. *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam,*

December 4-8, 2016, Proceedings, Part I, volume 10031 of *Lecture Notes in Computer Science*, 2016.

[DBBL12] Nicolas Debande, Maël Berthier, Yves Bocktaels, and Thanh-Ha Le. Profiled model based power simulator for side channel evaluation. Cryptology ePrint Archive, Report 2012/703, 2012. `http://eprint.iacr.org/2012/703`.

[DEMS15] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. ASCON v1.1, Aug 2015. CAESAR submission.

[DES77] Specification for the data encryption standard (DES). Federal Information Processing Standards publication 46, 1977.

[DGN+17] Jean-Luc Danger, Sylvain Guilley, Philippe Nguyen, Robert Nguyen, and Youssef Souissi. Analyzing security breaches of countermeasures throughout the refinement process in hardware design flow. In David Atienza and Giorgio Di Natale, editors, *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, pages 1129–1134. IEEE, 2017.

[dGPdlP+16] Wouter de Groot, Kostas Papagiannopoulos, Antonio de la Piedra, Erik Schneider, and Lejla Batina. Bitsliced masking and ARM: friends or foes? In Andrey Bogdanov, editor, *Lightweight Cryptography for Security and Privacy - 5th International Workshop, LightSec 2016, Aksaray, Turkey, September 21-22, 2016, Revised Selected Papers*, volume 10098 of *Lecture Notes in Computer Science*, pages 91–109. Springer, 2016.

[DGV93a] Joan Daemen, René Govaerts, and Joos Vandewalle. A new approach to block cipher design. In Ross J. Anderson, editor, *Fast Software Encryption, Cambridge Security Workshop, Cambridge, UK, December 9-11, 1993, Proceedings*, volume 809 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 1993.

[DGV93b] Joan Daemen, René Govaerts, and Joos Vandewalle. Weak keys for IDEA. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 224–231. Springer, 1993.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.

[DH77] Whitfield Diffie and Martin E. Hellman. Special feature exhaustive cryptanalysis of the NBS data encryption standard. *IEEE Computer*, 10(6):74–84, 1977.

[DH79] Whitfield Diffie and Martin E Hellman. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE*, 67(3):397–427, 1979.

[dHdV04] Jerry den Hartog and Erik P. de Vink. Virtual analysis and reduction of side-channel vulnerabilities of smartcards. In Theodosis Dimitrakos and Fabio Martinelli, editors, *Formal Aspects in Security and Trust: Second IFIP TC1 WG1.7 Workshop on Formal Aspects in Security and Trust (FAST), an event of the 18th IFIP World Computer Congress, August 22-27, 2004, Toulouse, France*, volume 173 of *IFIP*, pages 85–98. Springer, 2004.

[dHVdV+03] Jerry den Hartog, Jan Verschuren, Erik P. de Vink, Jaap de Vos, and W. Wiersma. PINPAS: A tool for power analysis of smartcards. In Dimitris Gritzalis, Sabrina De Capitani di Vimercati, Pierangela Samarati, and Sokratis K. Katsikas, editors, *Security and Privacy in the Age of Uncertainty, IFIP TC11 18th International Conference on Information Security (SEC2003), May 26-28, 2003, Athens, Greece*, volume 250 of *IFIP Conference Proceedings*, pages 453–457. Kluwer, 2003.

[dKGHG08] Gerhard de Koning Gans, Jaap-Henk Hoepman, and Flavio D. Garcia. A practical attack on the MIFARE classic. In Gilles Grimaud and François-Xavier Standaert, editors, *Smart Card Research and Advanced Applications, 8th IFIP WG 8.8/11.2 International Conference, CARDIS 2008, London, UK, September 8-11, 2008. Proceedings*, volume 5189 of *Lecture Notes in Computer Science*, pages 267–282. Springer, 2008.

[DMO16] Carolyn Whitnall David McCann and Elisabeth Oswald. Elmo: Emulating leaks for the arm cortex-m0 without access to a side channel lab. Cryptology ePrint Archive, Report 2016/517, 2016.

[DR02] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard.* Springer-Verlag, 2002.

[DRN03] Joan Daemen, Vincent Rijmen, and NIST. AES Proposal: Rijndael, April 2003. `http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf`.

[EMST78] William F Ehrsam, Carl HW Meyer, John L Smith, and Walter L Tuchman. Message verification and transmission error detection by block chaining, February 14 1978. US Patent 4,074,066.

[ES03] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin Heidelberg New York, USA, 2003.

[EWTS14] Hassan Eldib, Chao Wang, Mostafa M. I. Taha, and Patrick Schaumont. QMS: evaluating the side-channel resistance of masked software from source code. In *The 51st Annual Design Automation Conference 2014, DAC '14, San Francisco, CA, USA, June 1-5, 2014*, pages 209:1–209:6. ACM, 2014.

[FDLZ15] Yunsi Fei, A. Adam Ding, Jian Lao, and Liwei Zhang. A statistics-based success rate model for DPA and CPA. *J. Cryptographic Engineering*, 5(4):227–243, 2015.

[Fei73] Horst Feistel. Cryptography and computer privacy. *Scientific american*, 228:15–23, 1973.

[FH08] Julie Ferrigno and Martin Hlavác. When AES blinks: introducing optical side channel. *IET Information Security*, 2(3):94–98, 2008.

[FIP80] DES modes of operation. Federal Information Processing Standards publication 81, December 2nd, 1980.

[FLD12] Yunsi Fei, Qiasi Luo, and A. Adam Ding. A statistical model for DPA with novel algorithmic confusion analysis. In Prouff and Schaumont [PS12], pages 233–250.

[FN14] Gerhard Fettweis and Wolfgang Nebel, editors. *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*. European Design and Automation Association, 2014.

[FR14] Aurélien Francillon and Pankaj Rohatgi, editors. *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*. Springer, 2014.

[Fri07] Jeffrey Friedman. Tempest: A signal problem. *NSA Cryptologic Spectrum*, page 4, 1972, (partially decalssified in 2007).

[Gag13] Georges Gagnerot. *Étude des attaques et des contre-mesures associées sur composants embarqués*. PhD thesis, Université de Limoges, 2013.

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In Oswald and Rohatgi [OR08], pages 426–442.

[GdKGM+08] Flavio D. Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs. Dismantling MIFARE classic. In Sushil Jajodia and Javier López, editors, *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*, volume 5283 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2008.

[GH15] Tim Güneysu and Helena Handschuh, editors. *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*. Springer, 2015.

[GHM+17] Sylvain Guilley, Annelie Heuser, Tang Ming, Olivier Rioul, SAS Secure-IC, and LTCI Telecom-ParisTech. Stochastic Side-Channel Leakage Analysis via Orthonormal Decomposition. In *Innovative Security Solutions for Information Technology and Communications*, volume 10006 of *Lecture Notes in Computer Science*. Springer, 2017.

[GLS+15] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, Kerem Varici, Anthony Journault, François Durvaux, Lubos Gaspar, and Stéphanie Kerckhof. SCREAM Side-Channel Resistant Authenticated Encryption with Masking, Aug 2015. CAESAR submission.

[GLSV14] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-designs: Bitslice encryption for efficient masked software implementations. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 18–37. Springer, 2014.

[GM06] Louis Goubin and Mitsuru Matsui, editors. *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*. Springer, 2006.

[GNL11] Zheng Gong, Svetla Nikova, and Yee Wei Law. KLEIN: A new family of lightweight block ciphers. In Ari Juels and Christof Paar, editors, *RFID. Security and Privacy - 7th International Workshop, RFIDSec 2011, Amherst, USA, June 26-28, 2011, Revised Selected Papers*, volume 7055 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

[GP99] Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar,

editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.

[GP16] Benedikt Gierlichs and Axel Y. Poschmann, editors. *Cryptographic Hardware and Embedded Systems - CHES - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*. Springer, 2016.

[GSF14] Vincent Grosso, François-Xavier Standaert, and Sebastian Faust. Masking vs. multiparty computation: how large is the gap for aes? *J. Cryptographic Engineering*, 4(1):47–57, 2014.

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 444–461. Springer, 2014.

[Gui17] Sylvain Guilley, editor. *Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers*, volume 10348 of *Lecture Notes in Computer Science*. Springer, 2017.

[Hab65] Donald H Habing. The use of lasers to simulate radiation-induced transients in semiconductor devices and circuits. *IEEE Transactions on Nuclear Science*, 12(5):91–100, 1965.

[HBdH04] Gijs Hollestelle, Wouter Burgers, and JI den Hartog. Power analysis on smartcard algorithms using simulation. Technical report, Eindhoven University of Technology, 2004. `http://eprints.eemcs.utwente.nl/798/01/200422.pdf`.

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *Applied Cryptography and Network Security, 4th International Conference, ACNS 2006, Singapore, June 6-9, 2006, Proceedings*, volume 3989 of *Lecture Notes in Computer Science*, pages 239–252, 2006.

[HRG14] Annelie Heuser, Olivier Rioul, and Sylvain Guilley. A Theoretical Study of Kolmogorov-Smirnov Distinguishers — Side-Channel Analysis vs. Differential Cryptanalysis. In Prouff [Pro14], pages 9–28.

[IEE04] IEEE. *2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004), 16-20 February 2004, Paris, France.* IEEE Computer Society, 2004.

[IKD+08] Sebastiaan Indesteege, Nathan Keller, Orr Dunkelman, Eli Biham, and Bart Preneel. A practical attack on keeloq. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[JKP03] Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*. Springer, 2003.

[JNP15] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Joltik v1.3, Aug 2015. CAESAR submission.

[KB07] Boris Köpf and David A. Basin. An information-theoretic model for adaptive side-channel attacks. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 286–296. ACM, 2007.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Wiener [Wie99], pages 388–397.

[KK99] Oliver Kömmerling and Markus G. Kuhn. Design principles for tamper-resistant smartcard processors. In Scott B. Guthery and Peter Honeyman, editors, *Proceedings of the 1st Workshop on Smartcard Technology, Smartcard 1999, Chicago, Illinois, USA, May 10-11, 1999.* USENIX Association, 1999.

[KL08] Jonathan Katz and Yehuda Lindell. Introduction to modern cryptography: principles and protocols. cryptography and network security, 2008.

[KLL+14] Elif Bilge Kavun, Martin M. Lauridsen, Gregor Leander, Christian Rechberger, Peter Schwabe, and Tolga Yalçin. Prøst v1.1, Jun 2014. CAESAR submission.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO 1996, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[Koz84] Władysław Kozaczuk. *Enigma: how the German machine cipher was broken, and how it was read by the Allies in World War Two*. University Publications of America, 1984.

[KP00] Çetin Kaya Koç and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, volume 1965 of *Lecture Notes in Computer Science*. Springer, 2000.

[KP07] Mario Kirschbaum and Thomas Popp. *Evaluation of power estimation methods based on logic simulations*. Citeseer, 2007.

[KP14] Sebastian Kutzner and Axel Poschmann. On the Security of RSM - Presenting 5 First- and Second-Order Attacks. In Prouff [Pro14], pages 299–312.

[KQ99] François Koeune and Jean-Jacques Quisquater. A timing attack against rijndael. Technical report, UCL Crypto Group, 1999.

[Kra98] Hugo Krawczyk, editor. *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*. Springer, 1998.

[LBM11] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. Side Channel Attack: an Approach Based on Machine Learning. In *Second International Workshop on Constructive SideChannel Analysis and Secure Design*, pages 29–41. Center for Advanced Security Research Darmstadt, 2011.

[LBM15a] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. The bias-variance decomposition in profiled attacks. *J. Cryptographic Engineering*, 5(4):255–267, 2015.

[LBM15b] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. A machine learning approach against a masked AES - reaching the limit of side-channel attacks with a learning model. *J. Cryptographic Engineering*, 5(2):123–139, 2015.

[LJH14] Frédéric Lafitte, Jorge Nakahara Jr., and Dirk Van Heule. Applications of SAT solvers in cryptanalysis: Finding weak keys and preimages. *JSAT*, 9:1–25, 2014.

[LM90] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In Ivan Damgård, editor, *Advances in Cryptology - EUROCRYPT '90, Workshop on the Theory and Application of of Cryptographic Techniques, Aarhus, Denmark, May 21-24, 1990, Proceedings*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 1990.

[LMBM13] Liran Lerman, Stephane Fernandes Medeiros, Gianluca Bontempi, and Olivier Markowitch. A machine learning approach against a masked AES. In Francillon and Rohatgi [FR14], pages 61–75.

[LMM17] Liran Lerman, Zdenek Martinasek, and Olivier Markowitch. Robust profiled attacks: should the adversary trust the dataset? *IET Information Security*, 11:188–194(6), July 2017.

[LMV+13] Liran Lerman, Stephane Fernandes Medeiros, Nikita Veshchikov, Cédric Meuter, Gianluca Bontempi, and Olivier Markowitch. Semi-supervised template attack. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - 4th International Workshop, COSADE 2013, Paris, France, March 6-8, 2013, Revised Selected Papers*, volume 7864 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 2013.

[LMV16] Liran Lerman, Olivier Markowitch, and Nikita Veshchikov. Comparing sboxes of ciphers from the perspective of side-channel attacks. In *2016 IEEE Asian Hardware-Oriented Security and Trust, AsianHOST 2016, Yilan, Taiwan, December 19-20, 2016*, pages 1–6. IEEE Computer Society, 2016.

[LPB+15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine

learning revisited (and the curse of dimensionality in side-channel analysis). In Stefan Mangard and Axel Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, volume 9064 of *Lecture Notes in Computer Science*, pages 20–33. Springer, 2015.

[LR11] Pierre-Yvan Liardet and Fabrice Romain. Secured cryptographic calculation method, in particular against DFA and one-way attacks, and corresponding component, April, 6 2011. Patent: EP2509252A1, and also demand US20120257747 A1 (*under a slightly different title: "Method of secure cryptographic calculation, in particular, against attacks of the DFA and unidirectional type, and corresponding component"*).

[LVPM17] Liran Lerman, Nikita Veshchikov, Stjepan Picek, and Olivier Markowitch. On the Construction of Side-Channel Attack Resilient S-boxes. In Guilley [Gui17], pages 102–119.

[LWR00] Helger Lipmaa, David Wagner, and Phillip Rogaway. Comments to nist concerning aes modes of operation: Ctr-mode encryption, 2000.

[Mat02] Mitsuru Matsui, editor. *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*, volume 2355 of *Lecture Notes in Computer Science*. Springer, 2002.

[May00] Rita Mayer-Sommer. Smartly analyzing the simplicity and the power of simple power analysis on smartcards. In Koç and Paar [KP00], pages 78–92.

[Med12] Stephane Fernandes Medeiros. The schedulability of AES as a countermeasure against side channel attacks. In Andrey Bogdanov and Somitra Kumar Sanadhya, editors, *Security, Privacy, and Applied Cryptography Engineering - Second International Conference, SPACE 2012, Chennai, India, November 3-4, 2012. Proceedings*, volume 7644 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2012.

[MGH14] Amir Moradi, Sylvain Guilley, and Annelie Heuser. Detecting hidden leakages. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security - 12th International Conference, ACNS 2014, Lausanne, Switzerland, June 10-13, 2014. Proceedings*, volume 8479 of *Lecture Notes in Computer Science*, pages 324–342. Springer, 2014.

[MGH+15] Paweł Morawiecki, Kris Gaj, Ekawat Homsirikamol, Krystian Matusiewicz, Josef Pieprzyk, Marcin Rogawski, Marian Srebrny, and Marcin Wójcik. ICEPOLE v2, Aug 2015. CAESAR submission.

[MGV+16] Stephane Fernandes Medeiros, François Gérard, Nikita Veshchikov, Liran Lerman, and Olivier Markowitch. Breaking kalyna 128/128 with power attacks. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, volume 10076 of *Lecture Notes in Computer Science*, pages 402–414. Springer, 2016.

[MO17] David McCann and Elisabeth Oswald. Practical evaluation of masking software countermeasures on an iot processor. Cryptology ePrint Archive, Report 2017/399, 2017. http://eprint.iacr.org/2017/399.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards.* Springer, 2007.

[MOPT12] Andrew Moss, Elisabeth Oswald, Dan Page, and Michael Tunstall. Compiler assisted masking. In Prouff and Schaumont [PS12], pages 58–75.

[Mor15] Amir Moradi. *Advances in side-channel security.* PhD thesis, Habilitation, Ruhr-Universität Bochum, 2015.

[MOW14] Luke Mather, Elisabeth Oswald, and Carolyn Whitnall. Multi-target DPA attacks: Pushing DPA beyond the limits of a desktop computer. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 243–261. Springer, 2014.

[MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In Rao and Sunar [RS05], pages 157–171.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. Cryptology ePrint Archive, Report 2016/921, 2016. http://eprint.iacr.org/2016/921.

[MS01] Itsik Mantin and Adi Shamir. A practical attack on broadcast RC4. In Matsui [Mat02], pages 152–164.

[MS06] Stefan Mangard and Kai Schramm. Pinpointing the side-channel leakage of masked AES hardware implementations. In Goubin and Matsui [GM06], pages 76–90.

[MSV+15] Yan Michalevsky, Aaron Schulman, Gunaa Arumugam Veerapandian, Dan Boneh, and Gabi Nakibly. Powerspy: Location tracking using mobile device power analysis. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 2015, Washington, D.C., USA, August 12-14, 2015.*, pages 785–800. USENIX Association, 2015.

[MVOV96] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[MY92] Mitsuru Matsui and Atsuhiro Yamagishi. A new method for known plaintext attack of FEAL cipher. In Rainer A. Rueppel, editor, *Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings*, volume 658 of *Lecture Notes in Computer Science*, pages 81–91. Springer, 1992.

[NGD11] Maxime Nassar, Sylvain Guilley, and Jean-Luc Danger. Formal analysis of the entropy / security trade-off in first-order masking countermeasures against side-channel attacks. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *Progress in Cryptology - INDOCRYPT 2011 - 12th International Conference on Cryptology in India, Chennai, India, December 11-14, 2011. Proceedings*, volume 7107 of *Lecture Notes in Computer Science*, pages 22–39. Springer, 2011.

[Nov03] Roman Novak. Side-channel attack on substitution blocks. In Jianying Zhou, Moti Yung, and Yongfei Han, editors, *Applied Cryptography and Network Security, First International Conference, ACNS 2003. Kunming, China, October 16-19, 2003, Proceedings*, volume 2846 of *Lecture Notes in Computer Science*, pages 307–318. Springer, 2003.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[NRS+04] Ulrich Neffe, Klaus Rothbart, Christian Steger, Reinhold Weiss, Edgar Rieger, and Andreas Mühlberger. Energy estimation based on hierarchical bus models for power-aware smart cards. In *2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004), 16-20 February 2004, Paris, France* [IEE04], pages 300–305.

[NSGD12] Maxime Nassar, Youssef Souissi, Sylvain Guilley, and Jean-Luc Danger. RSM: A small and fast countermeasure for aes, secure against 1st and 2nd-order zero-offset scas. In Wolfgang Rosenstiel and Lothar Thiele, editors, *2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012*, pages 1173–1178. IEEE, 2012.

[OGDR10] Roman Oliynykov, Ivan Gorbenko, Viktor Dolgov, and Viktor Ruzhentsev. Results of ukrainian national public cryptographic competition. *Tatra Mountains Mathematical Publications*, 47(1):99–113, 2010.

[OGK+15] Roman Oliynykov, Ivan Gorbenko, Oleksandr Kazymyrov, Victor Ruzhentsev, Oleksandr Kuznetsov, Yurii Gorbenko, Oleksandr Dyrda, Viktor Dolgov, Andrii Pushkaryov, Ruslan Mordvinov, and Dmytro Kaidalov. A new encryption standard of ukraine: The kalyna block cipher. Cryptology ePrint Archive, Report 2015/650, 2015. `http://eprint.iacr.org/2015/650`.

[OR08] Elisabeth Oswald and Pankaj Rohatgi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*. Springer, 2008.

[OST06] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of AES. In Pointcheval [Poi06], pages 1–20.

[OWW14] Yossef Oren, Ofir Weisse, and Avishai Wool. A new framework for constraint-based probabilistic template side channel attacks. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 17–34. Springer, 2014.

[PGA06] Emmanuel Prouff, Christophe Giraud, and Sébastien Aumônier. Provably secure s-box implementation based on fourier transform. In Goubin and Matsui [GM06], pages 216–230.

[Pic16] Stjepan Picek. Evolutionary computation and cryptology. In Tobias Friedrich, Frank Neumann, and Andrew M. Sutton, editors, *Genetic and Evolutionary Computation Conference, GECCO 2016, Denver, CO, USA, July 20-24, 2016, Companion Material Proceedings*, pages 883–909. ACM, 2016.

[PM05] Thomas Popp and Stefan Mangard. Masked dual-rail pre-charge logic: Dpa-resistance without routing constraints. In Rao and Sunar [RS05], pages 172–186.

[PMMB15] Stjepan Picek, Bodhisatwa Mazumdar, Debdeep Mukhopadhyay, and Lejla Batina. Modified transparency order property: Solution or just another attempt. In Rajat Subhra Chakraborty, Peter Schwabe, and Jon A. Solworth, editors, *Security, Privacy, and Applied Cryptography Engineering - 5th International Conference, SPACE 2015, Jaipur, India, October 3-7, 2015, Proceedings*, volume 9354 of *Lecture Notes in Computer Science*, pages 210–227. Springer, 2015.

[Poi06] David Pointcheval, editor. *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, volume 3860 of *Lecture Notes in Computer Science*. Springer, 2006.

[PPE+14] Stjepan Picek, Kostas Papagiannopoulos, Baris Ege, Lejla Batina, and Domagoj Jakobovic. Confused by confusion: Systematic evaluation of DPA resistance of various s-boxes. In Willi Meier and Debdeep Mukhopadhyay, editors, *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*, volume 8885 of *Lecture Notes in Computer Science*, pages 374–390. Springer, 2014.

[Pro05] Emmanuel Prouff. DPA attacks and s-boxes. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 424–441. Springer, 2005.

[Pro14] Emmanuel Prouff, editor. *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, volume 8622 of *Lecture Notes in Computer Science*. Springer, 2014.

[PRR14] Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. On the practical security of a leakage resilient masking scheme. In Josh Benaloh,

editor, *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings*, volume 8366 of *Lecture Notes in Computer Science*, pages 169–182. Springer, 2014.

[PS12] Emmanuel Prouff and Patrick Schaumont, editors. *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*. Springer, 2012.

[PSG16] Romain Poussier, François-Xavier Standaert, and Vincent Grosso. Simple key enumeration (and rank estimation) using histograms: An integrated approach. In Gierlichs and Poschmann [GP16], pages 61–81.

[PV07] Pascal Paillier and Ingrid Verbauwhede, editors. *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*. Springer, 2007.

[PV17] Kostas Papagiannopoulos and Nikita Veshchikov. Mind the Gap: Towards Secure 1st-order Masking in Software. In Guilley [Gui17], pages 282–297.

[Rep16a] Oscar Reparaz. Detecting flawed masking schemes with leakage detection tests. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 204–222. Springer, 2016.

[Rep16b] Oscar Reparaz. Detecting flawed masking schemes with leakage detection tests. *IACR Cryptology ePrint Archive*, 2016:282, 2016.

[RFFT14] Md. Tauhidur Rahman, Domenic Forte, Jim Fahrny, and Mohammad Tehranipoor. ARO-PUF: an aging-resistant ring oscillator PUF design. In Fettweis and Nebel [FN14], pages 1–6.

[RGN13] Pablo Rauzy, Sylvain Guilley, and Zakaria Najm. Formally proved security of assembly code against power analysis: A case study on balanced logic. Cryptology ePrint Archive, Report 2013/554, 2013. `http://eprint.iacr.org/2013/554`.

[RNS+05] Klaus Rothbart, Ulrich Neffe, Christian Steger, Reinhold Weiss, Edgar Rieger, and Andreas Mühlberger. Power consumption profile analysis for security attack simulation in smart cards at high abstraction

level. In Wayne H. Wolf, editor, *EMSOFT 2005, September 18-22, 2005, Jersey City, NJ, USA, 5th ACM International Conference On Embedded Software, Proceedings*, pages 214–217. ACM, 2005.

[RO04] Christian Rechberger and Elisabeth Oswald. Practical template attacks. In Chae Hoon Lim and Moti Yung, editors, *Information Security Applications, 5th International Workshop, WISA 2004, Jeju Island, Korea, August 23-25, 2004, Revised Selected Papers*, volume 3325 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2004.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.

[RPD09] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-order masking and shuffling for software implementations of block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2009.

[RRSY98] Ronald L. Rivest, Matthew J.B. Robshaw, Ray Sidney, and Yiqun Lisa Yin. The RC6^{TM} block cipher. In *First Advanced Encryption Standard (AES) Conference*, page 16, 1998.

[RS05] Josyula R. Rao and Berk Sunar, editors. *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*. Springer, 2005.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[RSV$^+$11] Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Dina Kamel, and Denis Flandre. A formal study of power variability issues and side-channel attacks for nanoscale devices. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011.*

Proceedings, volume 6632 of *Lecture Notes in Computer Science*, pages 109–128. Springer, 2011.

[SA02] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In Jr. et al. [JKP03], pages 2–12.

[SB15] Markku-Juhani O. Saarinen and Billy B. Brumley. STRIBOBr2: “WHIRLBOB”, Aug 2015. CAESAR submission.

[SBG+12] Youssef Souissi, Shivam Bhasin, Sylvain Guilley, Maxime Nassar, and Jean-Luc Danger. Towards different flavors of combined side channel attacks. In Orr Dunkelman, editor, *Topics in Cryptology - CT-RSA 2012 - The Cryptographers’ Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings*, volume 7178 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 2012.

[Sch07] Bruce Schneier. *Applied cryptography: protocols, algorithms, and source code in C.* john wiley & sons, 2007.

[Sch15] Bruce Schneier. *Data and Goliath: The hidden battles to collect your data and control your world.* WW Norton & Company, 2015.

[SDB+10] Oliver Schimmel, Paul Duplys, Eberhard Boehl, Jan Hayek, R Bosch, and W Rosenstiel. Correlation power analysis in frequency domain. In *COSADE 2010 First International Workshop on Constructive SideChannel Analysis and Secure Design*, 2010.

[SGV08] François-Xavier Standaert, Benedikt Gierlichs, and Ingrid Verbauwhede. Partition vs. comparison side-channel distinguishers: An empirical evaluation of statistical tests for univariate side-channel attacks against two unprotected CMOS devices. In Pil Joong Lee and Jung Hee Cheon, editors, *Information Security and Cryptology - ICISC 2008, 11th International Conference, Seoul, Korea, December 3-5, 2008, Revised Selected Papers*, volume 5461 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2008.

[Sha45] Claude E Shannon. A mathematical theory of cryptography. *Memorandum MM*, 45:110–02, 1945.

[Sha49] Claude Elwood Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[Sin00] Simon Singh. *The code book: the science of secrecy from ancient Egypt to quantum cryptography.* Anchor, 2000.

[SKS09] François-Xavier Standaert, François Koeune, and Werner Schindler. How to compare profiled side-channel attacks? In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *Applied Cryptography and Network Security, 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009. Proceedings*, volume 5536 of *Lecture Notes in Computer Science*, pages 485–498, 2009.

[SKW+98] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: A 128-bit block cipher. *NIST AES Proposal*, 15, 1998.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Rao and Sunar [RS05], pages 30–46.

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Güneysu and Handschuh [GH15], pages 495–513.

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.

[SNK+12] Alexander Schlösser, Dmitry Nedospasov, Juliane Krämer, Susanna Orlic, and Jean-Pierre Seifert. Simple photonic emission analysis of AES - photonic side channel analysis for the rest of us. In Prouff and Schaumont [PS12], pages 41–57.

[SNK+13] Alexander Schlösser, Dmitry Nedospasov, Juliane Krämer, Susanna Orlic, and Jean-Pierre Seifert. Simple photonic emission analysis of AES. *J. Cryptographic Engineering*, 3(1):3–15, 2013.

[SP06] Kai Schramm and Christof Paar. Higher order masking of the AES. In Pointcheval [Poi06], pages 208–225.

[SPW07] Bharat B. Sukhwani, Uday Padmanabhan, and Janet Meiling Wang. Nano-sim: A step wise equivalent conductance based statistical simulator for nanotechnology circuit design. *CoRR*, abs/0710.4633, 2007.

[SPY+10] François-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung, and Elisabeth Oswald. Leakage resilient cryptography in practice. In Ahmad-Reza Sadeghi and David Naccache, editors, *Towards Hardware-Intrinsic Security - Foundations and Practice*, Information Security and Cryptography, pages 99–134. Springer, 2010.

[STA+15] Yu Sasaki, Yosuke Todo, Kazumaro Aoki, Yusuke Naito, Takeshi Sugawara, Yumiko Murakami, Mitsuru Matsui, and Shoichi Hirose. Minalpher v1.1, Aug 2015. CAESAR submission.

[Stö12] Marc Stöttinger. *Mutating runtime architectures as a countermeasure against power analysis attacks.* PhD thesis, Darmstadt University of Technology, Germany, 2012.

[Sto15] Ko Stoffelen. Intrinsic side-channel analysis resistance and efficient masking, 2015. Master Thesis.

[SVO+10] François-Xavier Standaert, Nicolas Veyrat-Charvillon, Elisabeth Oswald, Benedikt Gierlichs, Marcel Medwed, Markus Kasper, and Stefan Mangard. The world is not enough: Another look on second-order DPA. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 112–129. Springer, 2010.

[SYY+01] Takeshi Shimoyama, Hitoshi Yanami, Kazuhiro Yokoyama, Masahiko Takenaka, Kouichi Itoh, Jun Yajima, Naoya Torii, and Hidema Tanaka. The block cipher SC2000. In Matsui [Mat02], pages 312–327.

[TAL09] Céline Thuillet, Philippe Andouard, and Olivier Ly. A Smart Card Power Analysis Simulator. In *CSE (2)*, pages 847–852. IEEE Computer Society, 2009.

[TAV02] Kris Tiri, Moonmoon Akmal, and Ingrid Verbauwhede. A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards. In *Proceedings of the 28th European Solid-State Circuits Conference - ESSCIRC 2002, Florence, Italy, September 24-26*, pages 403–406. IEEE, 2002.

[TAV+04] Yuh-Fang Tsai, Ananth Hegde Ankadi, Narayanan Vijaykrishnan, Mary Jane Irwin, and Theocharis Theocharides. Chippower: an architecture-level leakage simulator. In *Proceedings IEEE International SOC Conference, September 12-15, 2004, Hilton Santa Clara, CA, USA*, pages 395–398. IEEE, 2004.

[THM07] Stefan Tillich, Christoph Herbst, and Stefan Mangard. Protecting AES software implementations on 32-bit processors against power analysis. In Jonathan Katz and Moti Yung, editors, *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007, Proceedings*, volume 4521 of *Lecture Notes in Computer Science*, pages 141–157. Springer, 2007.

[TV04] Kris Tiri and Ingrid Verbauwhede. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004), 16-20 February 2004, Paris, France* [IEE04], pages 246–251.

[TV05] Kris Tiri and Ingrid Verbauwhede. Simulation models for side-channel information leaks. In William H. Joyner Jr., Grant Martin, and Andrew B. Kahng, editors, *Proceedings of the 42nd Design Automation Conference, DAC 2005, San Diego, CA, USA, June 13-17, 2005*, pages 228–233. ACM, 2005.

[Ves14] Nikita Veshchikov. SILK: high level of abstraction leakage simulator for side channel analysis. In Mila Dalla Preda and Jeffrey Todd McDonald, editors, *Proceedings of the 4th Program Protection and Reverse Engineering Workshop, PPREW@ACSAC 2014, New Orleans, LA, USA, December 9, 2014*, pages 3:1–3:11. ACM, 2014.

[VG17a] Nikita Veshchikov and Sylvain Guilley. Implementation flaws in the masking scheme of DPA Contest v4. *IET Information Security*, 2017.

[VG17b] Nikita Veshchikov and Sylvain Guilley. Use of simulators for side-channel analysis. In *2017 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2017, Paris, France, April 26-28, 2017*, pages 104–112. IEEE, 2017.

[VGRS12] Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 390–406. Springer, 2012.

[VMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In Wang and Sako [WS12], pages 740–757.

[VML17] Nikita Veshchikov, Stephane Fernandes Medeiros, and Liran Lerman. Variety of scalable shuffling countermeasures against side channel attacks. *Journal of Cyber Security and Mobility*, pages 195–232, July 2017.

[vWWB11] Jasper G. J. van Woudenberg, Marc F. Witteman, and Bram Bakker. Improving differential power analysis by elastic alignment. In Aggelos Kiayias, editor, *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 104–119. Springer, 2011.

[Wie99] Michael J. Wiener, editor. *Advances in Cryptology - CRYPTO 1999, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*. Springer, 1999.

[WO15] Carolyn Whitnall and Elisabeth Oswald. Robust profiling for dpa-style attacks. In Güneysu and Handschuh [GH15], pages 3–21.

[WOM11] Carolyn Whitnall, Elisabeth Oswald, and Luke Mather. An Exploration of the Kolmogorov-Smirnov Test as a Competitor to Mutual Information Analysis. In Emmanuel Prouff, editor, *CARDIS*, volume 7079 of *Lecture Notes in Computer Science*, pages 234–251. Springer, 2011.

[WS12] Xiaoyun Wang and Kazue Sako, editors. *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*. Springer, 2012.

[YE13] Xin Ye and Thomas Eisenbarth. On the Vulnerability of Low Entropy Masking Schemes. In Francillon and Rohatgi [FR14], pages 44–60.

[YMOT14] Noritaka Yamashita, Kazuhiko Minematsu, Toshihiko Okamura, and Yukiyasu Tsunoo. A smaller and faster variant of RSM. In Fettweis and Nebel [FN14], pages 1–6.

[ZBL+15] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. *SCIENCE CHINA Information Sciences*, 58(12):1–15, 2015.

[ZDD+17] Liwei Zhang, A. Adam Ding, Francois Durvaux, Francois-Xavier Standaert, and Yunsi Fei. Towards sound and optimal leakage detection procedure. Cryptology ePrint Archive, Report 2017/287, 2017. `http://eprint.iacr.org/2017/287`.

[ZGLG14] Zhong Zeng, Dawu Gu, Junrong Liu, and Zheng Guo. An improved side-channel attack based on support vector machine. In *Tenth International Conference on Computational Intelligence and Security, CIS 2014, Kunming, Yunnan, China, November 15-16, 2014*, pages 676–680. IEEE Computer Society, 2014.

[ZJRR12] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 305–316. ACM, 2012.