Formal sofwtare methods for cryptosystems implementation security

Pablo Rauzy

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2015-ENST-0039

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Informatique »

présentée et soutenue publiquement par

Pablo Rauzy

le 13 juillet 2015

Méthodes logicielles formelles pour la sécurité des implémentations cryptographiques

Directeur de thèse: **Sylvain Guilley**

Jury

M. François DUPRESSOIR, Chercheur, IMDEA Software Institute	Examinateur
M. Pierre-Alain FOUQUE, Professeur, Université Rennes 1	Rapporteur
Mme. Karine HEYDEMANN, Maître de Conférence, UPMC	Examinatrice
M. David NACCACHE, Professeur, Université Panthéon-Assas	Président
Mme. Marie-Laure POTET, Professeure, Ensimag	Rapporteuse
M. Medhi TIBOUCHI, Chercheur, NTT	Examinateur
M. David VIGILANT, Chercheur, Gemalto	Invité

TELECOM ParisTech
école de l'Institut Mines-Télécom - membre de ParisTech
46 rue Barrault 75013 Paris - (+33) 1 45 81 77 77 - www.telecom-paristech.fr

T H È S E

Refaire un peu le monde en posant des questions,
Essayer d'y répondre avec de la technique,
Créer l'étonnement, être pédagogique.
Haïssant les erreurs, aimant les suggestions,
Ecrire des papiers et des présentations.
Riez de cette fable en vérité tragique
Cachant tant bien que mal la formule magique :
"H-index labidex !". Comptez vos citations
Et multipliez par, un jour de Lune noire,
Le nombre de majeurs se levant à leur gloire,
Inflexiblement droits, des mains des éditeurs.
Bien ! Vous venez de faire un petit peu de science
Résolument valide... aux yeux de la finance.
Et si nous disions "zut" à ces inquisiteurs ?

$\sim$ p4bl0

0x0

Avant-propos

0.1 Contexte

Ma thèse s'est déroulée à Télécom ParisTech, dans les locaux situés rue Dareau dans le 14ème arrondissement de Paris, au sein de l'équipe SEN (Sécurité Électronique Numérique) du département COMELEC (Communication et Électronique) du LTCI (Laboratoire Traitement et Communication de l'Information, UMR 5141 commune avec le CNRS).

Les activités de l'équipe se focalisent autour de trois thèmes de recherche :

- les contraintes de sécurité sont traitées dans la thématique *Matériel pour l'informatique sécurisée et de confiance* (Trusted Computing Hardware) ;
- les contraintes de fiabilité sont traitées dans la thématique *Analyse et conception de processeurs fiables basés sur des technologies non fiables* (Analysis and Design of Reliable Processeurs Based on Unreliable Technologies) ;
- les contraintes de complexité et de consommation sont traitées dans la thématique *Architectures optimales pour l'implémentation d'algorithmes complexes* (Optimal Architectures for Complex Algorithms Implementations).

Ma thèse se positionne dans le premier de ces trois thèmes, avec la particularité de s'attaquer au problème de la confiance en le matériel au niveau logiciel, puisque je suis informaticien d'origine. Je me suis ainsi positionné à l'interface entre deux mondes, celui de l'informatique théorique, d'où je venais après mes trois années passées à l'étudier à l'ENS, et celui du matériel, plus particulièrement de l'électronique embarquée. Au niveau de cette interface, c'est à la cryptologie et aux méthodes formelles que je me suis intéressé.

Cette thèse a été encadrée par Sylvain Guilley et a été financée par une bourse ministérielle obtenue via l'école doctorale ÉDITE (ED130). J'ai aussi bénéficié d'une mission doctorale d'enseignement, que j'ai effectuée à Polytech'UPMC.

Les deux premières années, j'ai encadré au premier semestre les séances de TP du cours de "Programmation en C" des étudiant·e·s de 3ème année de la formation EI2I (Électronique et Informatique parcours Informatique Industrielle), et au second semestre les séances de TP du cours de "Développement Web" des étudiant·e·s de 3ème année de la formation AGRAL (Agroalimentaire).

La troisième année, j'ai été seul en charge du cours de "Programmation orienté objet et langage C++" des étudiant·e·s de 4ème année de la formation EISE (Électronique et informatique parcours Systèmes Embarqués). J'ai pu concevoir, donner, et évaluer les cours, les TP, les examens sur papier et sur machine, ainsi que les projets de ce module. Cela m'a permis d'expérimenter l'intégralité des aspects de l'enseignement d'un cours.

0.2 Remerciements

Je vais commencer ces remerciements par les personnes qui m'ont permis de réaliser ma thèse en m'acceptant dans leur équipe malgré mon arrivée de dernière minute, en catastrophe : Jean-Luc Danger et Sylvain Guilley.

Je tiens à remercier Sylvain tout particulièrement car c'est lui qui en pratique a encadré ma thèse, et je ne pense pas que quiconque aurait pu être un meilleur encadrant pour moi. Sylvain a su me laisser énormément d'autonomie et de liberté, tout en étant très présent lorsque je le sollicitais. Sur le plan scientifique, il m'a laissé d'immenses marges de manœuvre dans nos articles et dans mes présentations, en m'offrant des conseils précieux sans jamais rien m'imposer. Sur le plan personnel, il m'a laissé — sans m'en tenir rigueur — prendre du temps pour des activités annexes, commme travailler sur des projets avec le club Inutile[1] (même à l'approche d'une échéance !), ou militer pour le libre accès à la recherche[2]. Son optimisme naturel et sa bonne humeur ont su me remotiver chaque fois que j'en ai eu besoin.

Même si nous avons beaucoup moins travaillé ensemble, nombre de mes discussions avec Jean-Luc m'ont été utiles, et surtout, je voudrais ici rendre un hommage à sa pratique du jeu-de-mot-naze, que j'apprécie le plus sincèrement du monde, comme mes amis proches[3] pourraient en témoigner.

Ensuite, je veux remercier les membres de mon jury. Je suis extrêmement honoré que François Dupressoir, Pierre-Alain Fouque, Karine Heydemann, David Naccache, Marie-Laure Potet, Mehdi Tibouchi, et David Vigilant aient accepté de compter parmi les membres de mon jury de thèse. J'adresse naturellement un remerciement spécial à Marie-Laure et Pierre-Alain qui jouent pour ma thèse l'important rôle de rapporteu·se/r et qui m'ont permis grâce à leurs remarques d'améliorer sensiblement la qualité de mon tapuscrit.

Un remerciement singulier va à David Naccache, qui a été mon professeur à l'École normale supérieure : je garde un génial souvenir de son cours d'*Informatique scientifique par la pratique* qui m'a permis de voir pour la première fois ce que c'était que la recherche de l'intérieur, notamment la rédaction collaborative d'un article.

En restant pour l'instant dans le monde de la recherche, je veux maintenant remercier tous les membres de l'équipe SEN. Des remerciements particuliers vont à Zakaria Najm, ingénieur de recherche au labo dit "*ZakGyver*", pour son efficacité et sa disponibilité impressionnantes, et pour tout le travail que l'on a fait ensemble, notamment les soirées pré-échéances que l'on a passées à travailler dans la bonne humeur. Il en va de même pour Taoufik Chouta, avec qui j'ai partagé mon bureau et de nombreuses rigolades pendant les deux premières années de ma thèse. Je tiens aussi à remercier Martin Moreau, qui est venu faire son stage de M2 avec Sylvain et moi. On s'est immédiatement très bien entendu, et j'espère sincèrement qu'il pourra continuer notre travail pendant sa thèse à partir de l'an prochain.

Je remercie également les nombreuses autres personnes de la communauté scientifique avec qui j'ai pu échanger lors de conférences, de séminaires, ou simplement par courriel. En particulier je voudrais chaleureusement remercier Gilles Barthe, Pierre-Yves Strub et François Dupressoir pour m'avoir invité à l'IMDEA Software Institute à Madrid, où j'ai passé un super mois.

[1] http://inutile.club/index.php?n=Activites.ProjectSK

[2] http://pablo.rauzy.name/openaccess.html

[3] Par exemple ceux qui l'appellent "*Dr. Danger*", parce que c'est super classe.

J'en viens maintenant à remercier ma famille. Ma sœur Nina, qui est actuellement très occupée par sa noble mission consistant à parcourir le monde pour y répandre sa bonne humeur et la façon de Marseille. Mes petits frères, Raphaël et Manuel, qui sont une source inépuisable de joie de vivre. Mes parents, Magali Marcel et Antoine Rauzy, qui sont je pense les plus géniaux du monde, il n'y a pas vraiment besoin d'en dire plus. Ma belle-mère Anna aussi, qui fait partie de ma famille depuis plus de 10 ans maintenant. Enfin, tous les autres membres de ma famille moins proche : mes grands-parents, oncles, cousin, grands-cousin·e·s, et petits-neveux (et tous les {n,m}-parents qui existeraient mais n'entreraient pas dans les cases citées).

Au tour des ami·e·s. C'est avec eux que j'ai passé l'essentiel de mon temps ces dernières années. Et c'est en grande partie grâce à eux aussi que ma vie est si géniale. Par souci de simplicité, je vais faire ça dans l'ordre chronologique.

Je commence donc par remercier tou·te·s mes ami·e·s de Marseille, en particulier Alexandre Bovier-Lapierre et Yoann Corbasson.

Je continue avec mes ami·e·s rencontré·e·s à l'ENS. D'abord tou·te·s les membres du MCube, qui se reconnaîtront, particulièrement Antoine Amarilli, Floriane Dardard, et Nina Soleymani. Je remercie aussi les Ami·e·s d'Ulminfo, composés majoritairement de mes camarades des promo Info 2008 et 2009, et spécialement mes deux colocataires, Guillaume Claret et Pierre Gaillard. Je remercie également les membres du GdT Logique, ainsi que les membres les plus éminents du club Inutile, qui se reconnaîtront aisément. Je tiens aussi à remercier mes camarades militant·e·s de SUD, du NPA, et des autres organisations politiques proches, avec une mention spéciale à Jordi Brahamcha-Marin.

Et maintenant, les **grotas**. Grotaaaaas! Les grotas et grosses-tasses, c'est le groupe d'ami·e·s dans lequel je gravite principalement depuis près de 4 ans maintenant. Ce sont des gens avec qui je me suis immédiatement senti à l'aise, à un point qu'il est difficile de décrire, et il semble que ce sentiment fut réciproque, puisqu'illes m'ont appelé Grotablo dès les premiers jours. Je ne les ai pas quittés depuis que je les ai rencontrés. Je passe donc (en moyenne) 24 heures par jour avec elleux, que ce soit en chair et en os ou sur un canal de discussion IRC, que ce soit sur des toits parisiens ou dans les tréfonds des catacombes, que ce soit pendant un repas ou un autre repas. Je gromercie (dans l'ordre alphabétique, puisque notre système d'écriture linéaire nous impose d'en suivre un) Gronas, Grossetas-selier, Grotachiet, Grotachin, Grotamiel, Grotarc, Grotarine, Grotarion, Grotarrel, Grotarée, Grotaxime, Grotéa, Grotenedicte, Groterebel, Grot'ha, Grotillon, Grotimon, et Ted[4]. Je remercie également les grotadjacent·e·s (i.e., les membres de la grotadhérence) qui sont trop nombreu·x/ses pour être nommément cité·e·s mais qui se reconnaîtront également.

Pour finir, *last but* pas du tout *least*, je tiens absolument à remercier Léa Lejeune pour être à mes côtés, pour me supporter, pour me tirer sans cesse vers le haut, pour me faire rire, pour me sourire, et de manière générale pour me rendre heureux depuis près de 2 ans.

[4]En fait il s'appelle Flavien.

0.3 Motivation

J'ai commencé à m'intéresser à l'informatique assez jeune, vers le début du collège. C'est à la même période qu'est née ma conscience politique. Très rapidement, ces deux intérêts se sont rejoints avec la découverte du logiciel libre, et donc des licences libres, puis de la culture libre. C'est donc assez naturellement que je décide d'aller faire de l'informatique à l'université tout de suite après le lycée. À la fac, je découvre "pour de vrai" l'aspect scientifique de l'informatique, en plus de ses facettes d'outils et de technologies que je manipulais déjà. L'intérêt que présente pour moi l'informatique en tant que science, ainsi que la grande liberté dont je dispose à l'université, rendent pour moi le monde de la recherche (que j'ai déjà pu entr'apercevoir via des membres de ma famille) très attractif. Après deux ans à la Faculté des Sciences de Luminy, je décide donc de continuer mes études à l'École normale supérieure.

À l'ENS, je découvre pour de vrai l'activité de chercheur au travers de mes stages en laboratoires et de certains cours. Par exemple le cours d'*Informatique scientifique par la pratique*, de David Naccache, me permet de participer à la rédaction de mon premier article tout en découvrant le domaine de la sécurité de l'information, avec lequel j'ai une affinité toute particulière due à mon esprit *hacker*.

Les révélations de Wikileaks ainsi que le travail d'investigation de Reflets.info révèlent l'importance de la protection de la vie privée. Les révélations d'Edward Snowden confirment cette nécessité à l'heure du tout numérique. Dans le cadre de la cyber-communication généralisée, la complexité des données et des réseaux sur lesquelles elles circulent oblige à se poser sérieusement et scientifiquement la question de la protection de la vie privée. L'étude de la sécurité de l'information est alors un élément majeur du combat pour le maintien de la vie privée, de la liberté, et donc in fine de la démocratie. Je me suis donc orienté vers une thèse en sécurité de l'information.

En tant que militant pour la culture libre, mon orientation vers la recherche m'a amené à me pencher sur la problématique du *libre accès*. Le code de la recherche donne pour mission aux chercheurs le partage et la diffusion des connaissances scientifiques[5]. Actuellement, de gros soucis se posent en travers de cette mission, et j'ai donc choisi de passer une partie de mon temps à m'attaquer comme je le pouvais à ce problème. J'ai pour cela commencé à faire de l'information en rédigeant des articles, en organisant et donnant des conférences. J'estime que mon investissement dans ce combat représente environ deux mois de travail sur les trois ans que j'ai consacré à ma thèse. Vous pourrez retrouver en annexe F page 159 un dossier d'introduction au libre accès (déjà publié sur ma page web personnelle et sur le Club Mediapart).

[5] http://www.legifrance.gouv.fr/affichCode.do?idSectionTA=LEGISCTA000006151273&cidTexte=LEGITEXT000006071190&dateTexte=20150218

0x1 Contents

0x2

Abstracts

2.1 Résumé en français

Les implémentations cryptographiques sont vulnérables aux attaques physiques, et ont donc besoin d'en être protégées. Bien sûr, des protections défectueuses sont inutiles. L'utilisation des méthodes formelles permet de développer des systèmes tout en garantissant leur conformité à des spécifications données. Le premier objectif de ma thèse, et son aspect novateur, est de montrer que les méthodes formelles peuvent être utilisées pour prouver non seulement les principes des contre-mesures dans le cadre d'un modèle, mais aussi leurs implémentations, étant donné que c'est là que les vulnérabilités physiques sont exploitées. Mon second objectif est la preuve et l'automatisation des techniques de protection elles-mêmes, car l'écriture manuelle de code est sujette à de nombreuses erreurs, particulièrement lorsqu'il s'agit de code de sécurité.

Les attaques physiques peuvent être classifiées en deux catégories distinctes. Les attaques passives, où l'attaquant peut seulement lire l'information qui fuit par *canaux auxiliaires* (comme la consommation de courant ou les émanations électromagnétiques). Et les attaques actives, où l'attaquant perturbe le système pour faire en sorte de lui faire révéler des secrets via sa sortie standard. Par conséquent, j'ai poursuivi mes objectifs dans ces deux cadres : sur une contre-mesure qui diminue les fuites par canaux auxiliaires, et sur des contre-mesures contre les attaques par *injection de faute*.

Comme il existe déjà des propriétés rigoureuses de sécurité pour les protections contre les fuites par canaux auxiliaires, mes contributions se concentrent sur les méthodes formelles pour la conception et la vérification d'implémentations d'algorithmes protégés. J'ai développé une méthode de protection qui, étant donné une implémentation, en génère une version améliorée qui a un rapport signal à bruit nul sur ses canaux auxiliaires, grâce au fait que la fuite a été rendue constante (en particulier, la fuite ne dépend pas des données sensibles). Dans l'intérêt de la démonstration, j'ai aussi entrepris d'écrire un outil qui automatise l'application de cette méthode sur un code non sécurisé écrit en langage assembleur. Indépendemment, l'outil permet de prouver que la propriété de fuite constante se vérifie pour une implémentation donnée, ce qui permet de vérifier systématiquement le résultat de la méthode de protection. À ma connaissance, paioli est le premier outil permettant de protéger automatiquement une implémentation contre les fuites par canaux auxiliaires en équilibrant sa fuite de manière prouvable.

À l'inverse, la définition même des objectifs de sécurité n'était pas clairement établie pour les attaques par injection de faute lorsque j'ai commencé ma thèse.

Les propriétés de sécurité à prouver n'ayant même pas été formellement énoncées, beaucoup de contre-mesures ont été publiées sans preuve. C'est seulement lors de ma thèse que les "conditions de succès d'attaque" ont été introduites.

En conséquence, la première question a été d'évaluer les contre-mesures existantes par rapport à ces conditions de succès d'attaque. À cette fin, j'ai développé un cadre théorique de travail et un outil qui l'implémente. L'utilisation de finja m'a permis (ainsi qu'à d'autres chercheur·e·s) de vérifier certaines contre-mesures, de retrouver des attaques connues et aussi d'en découvrir de nouvelles.

La seconde question portait sur la minimalité des contre-mesures. J'ai étudié en profondeur l'une des contre-mesures de l'état de l'art. Le résultat de cette étude formelle a été la simplification drastique de la contre-mesure, aussi bien au niveau de la longueur du code que de la nécessité de nombres aléatoires (qui sont coûteux à générer), et ce sans affecter ses propriétés de sécurité. Ce travail a montré la valeur ajoutée des méthodes formelles par rapport à l'ingénierie par essais-erreurs qui a été jusqu'à présent la méthode principale de développement de contre-mesures.

Les contre-mesures existantes revendiquent de protéger contre une ou parfois deux fautes. Cependant, des attaques utilisant plus de deux fautes ont vu le jour, aussi bien en pratique qu'en théorie. La troisième question était alors de concevoir une nouvelle contre-mesure d'ordre supérieur, capable de résister à un nombre arbitraire de fautes. À son tour, la conception d'une nouvelle contre-mesure soulève la question de ce qui fait réellement fonctionner une contre-mesure. Pour tenter de répondre à cette question, j'ai classifié les contre-mesures existantes en essayant d'extraire les principes de protection des techniques employées. Ce travail de catégorisation m'a permis de comprendre l'essence d'une contre-mesure, et, en me basant dessus, de proposer une recette de conception de contre-mesure pouvant résister à un nombre arbitraire (mais fixé) de fautes.

J'ai aussi remarqué que toutes les contre-mesures que j'ai étudiées sont des variantes d'optimisation d'une même technique de base qui consiste à vérifier l'intégrité du calcul en utilisant une forme de redondance. Cette technique est indépendante de l'algorithme auquel elle s'applique, et aussi de la condition de succès d'attaque, puisqu'elle repose entièrement sur des propriétés des structures mathématiques dans lesquelles se trouve les données sensibles, c'est à dire l'arithmétique modulaire. J'ai donc proposé une propriété de résistance face aux attaques par injection de faute qui dépasse la notion de condition de succès d'attaque. La quatrième question a été d'appliquer cette technique de protection à tous les calculs de cryptographie asymétrique, puisqu'ils travaillent tous sur des données mathématiques similairement structurées. Dans cette optique, j'ai développé une abstraction des calculs de cryptographie asymétrique qui permet d'appliquer simplement la méthode de protection par redondance. J'ai formellement défini cette méthode en définissant une transformation de code par réécriture que j'ai prouvée correcte. J'ai écrit un compilateur qui automatise cette transformation et a permis d'obtenir des implémentations protégées d'algorithmes pour lesquels aucune contre-mesure n'a été publiées mais qui sont déjà victimes de nombreuses attaques en faute. Un avantage additionnel du compilateur enredo est de permettre d'étudier le compromis entre sécurité et temps de calcul.

2.2 Abstract

Implementations of cryptosystems are vulnerable to physical attacks, and thus need to be protected against them. Of course, malfunctioning protections are useless. Formal methods help to develop systems while assessing their conformity to a rigorous specification. The first goal of my thesis, and its innovative aspect, is to show that formal methods can be used to prove not only the principle of the countermeasures according to a model, but also their implementations, as it is where the physical vulnerabilities are exploited. My second goal is the proof and the automation of the protection techniques themselves, because handwritten security code is error-prone.

Physical attacks can be classified into two distinct categories. Passive attacks, where the attacker only reads information that leaks through *side channels* (such as power consumption or electromagnetic emanations). And active attacks, where the attacker tampers with the system to have it reveal secrets through its "normal" output channel. Therefore, I have pursued my goals in both settings: on a countermeasure that diminishes side-channel leakage, and on countermeasures which detect *fault injection* attacks.

As there already exist rigorous security properties for protections against side-channel leakage, my contributions concentrate on formal methods for design and verification of protected implementations of algorithms. I have developed a methodology to protect an implementation by generating an improved version of it which has a null side-channel signal-to-noise ratio, as its leakage is made constant (in particular, it does not depend on the sensitive values). For the sake of demonstration, I have also undertaken to write a tool which automates the application of the methodology on an insecure input code written in assembly language. Independently, the tool is able to prove that this constant leakage property holds for a given implementation, which can be used to verify the security of the output of the protection mechanism. To my best knowledge, paioli is the first tool which can automatically protect an implementation against side-channel attacks by provably balancing its leakage.

In contrast, the definition of the security objectives was not clearly established for fault injection attacks before I started my PhD. Many countermeasures had been published without any proofs, as the necessary properties to prove had not been formally expressed. It is only during my PhD that so-called "attack success conditions" were introduced.

Hence, the first question was to evaluate existing countermeasures with respect to these attack success conditions. To this end, I have developed a theoretical framework and a tool implementing it. The use of finja allowed me and others to verify some countermeasures, but also to find known attacks and discover new ones.

A second question was the minimality of the countermeasures. I have studied in depth one of the state-of-the-art's countermeasure. I have been able to drastically simplify it in terms of code length and randomness requirement, without affecting its security properties. This work has shown the added value of formal methods, relative to engineering by trial-and-error like most countermeasures have been developed as of today.

Existing countermeasures claim to protect against one, or sometimes two, fault injections. However, attacks using more than two faults have been shown practical.

The third question was then to create a new high-order countermeasure able to resist any number of faults. In turn, the design of a new countermeasure raises the question of the accurate definition of the rational of countermeasures. In a first step to find an answer, I have classified existing countermeasures in order to extract protection principles from the techniques they rely on. This categorization work allowed me to understand the essence of a countermeasure. Based on it, I have been able to propose a method for designing a countermeasure that can resists an arbitrary (but fixed) number of faults.

I have also noticed that all the countermeasures I have studied are variations of optimization for the same base technique consisting in verifying the integrity of the computation using a form of redundancy. This technique is independent of the algorithm it is applied to and of the attack success conditions, as it only depends on the mathematical structure of the sensitive data, namely modular arithmetic. I have thus proposed a property of resistance against fault injection attacks which goes beyond attack success conditions. The fourth question was then to apply this technique to all asymmetric cryptography algorithms, which all use similarly structured data. In order to achieve this goal, I have developed an abstraction of asymmetric cryptographic computations. By design, this representation permits to apply the redundancy protection scheme. I have formally defined this protection scheme by introducing a code transformation that I have proved correct. I have written a compiler which automates this transformation and allowed to obtain protected implementations of cryptographic algorithms for which no countermeasures were known until now, but which were already victims of fault injection attacks. An additional advantage of the enredo compiler is that it allows to study the trade-off between security and complexity.

0x3

Introduction

3.1 Information Security

The Wikipedia page[1] for *information security* starts by defining it as such:

> "*Information security, sometimes shortened to InfoSec, is the practice of defending information from unauthorized access, use, disclosure, disruption, modification, perusal, inspection, recording or destruction. It is a general term that can be used regardless of the form the data may take (e.g., electronic, physical).*"

In the present era, personal data are everywhere. From our phones to our credit cards, from surveillance cameras watching us in the street to our social networks profiles. Protecting these data is essential to our right to privacy, which is fundamental to our freedom. This has somehow recently been rendered clear and public, since Snowden's and subsequent revelations on the espionage practices of the 14-Eyes.

Information security is thus a very large field, which has interfaces with many others, going from electrical engineering to cognitive sciences. Indeed, it is possible for one to work on information security as long as one is interested in topics ranging from cyber-security to social-engineering, including cryptography, malware analysis, embedded systems, networks, web security, programming languages, psychology, etc.

All these subjects interest me, and I would enjoy participating to do research on them to contribute to the advancement of the state of the art, but also to teach them to students who will in turn contribute by building on top of what I could pass down to them. However, I had to start somewhere, and it is in the domain of cryptology that my PhD has taken place.

3.2 Cryptology

The term *cryptology* could etymologically be defined as "the science of secrets", which happens to be quite a good definition indeed. The field of cryptology is concerned with the study of techniques for secure communications in the presence of untrusted third parties. Knowing the vulnerabilities of a system is one of the best ways to learn how to protect it. The idea is to define what an attacker can and cannot do (e.g., request as many computation as needed, but not the value of the secret key). For this reason, cryptology has two aspects: cryptography ("secret writing"),

[1] https://en.wikipedia.org/wiki/Information_security, accessed 2015-02-23.

which mission is to provide encryption and decryption algorithms[2], and cryptanalysis ("secret untying"), which mission is to analyze and breach cryptographic security systems in order to gain access to the content of encrypted messages without a knowledge of the cryptographic key.

3.2.1 Cryptography

The history of *cryptography* is long and interesting. I invite the reader to take a look at the dedicated Wikipedia page[3] for a detailed overview of this fascinating history, as here I am only going to develop on a part of modern cryptography. Nowadays, cryptography is used everywhere in our daily life: in our credit cards, in our phones, on the Internet for the web and emails, etc.

Cryptography is the practice and study of encryption and decryption algorithms, whose goal is to allow private communications in the presence of untrusted third parties. In this setting, three properties are essential: confidentiality, authenticity, and integrity.

Informally, *confidentiality* means that the content of the communication must be accessible only to the communicants. *Authenticity* means that the communicants want to be sure of who they are communicating with. *Integrity* means that the recipients want to be sure that the message they received is exactly what the sender intended them to receive.

Encryption consists in converting a *plaintext* (or *cleartext*) message into unintelligible *ciphertext*. *Decryption* is the reverse operation.

A *cipher* is a pair of algorithms, usually assumed to be public, which perform corresponding encryption and decryption operations. Each instance is controlled by the algorithms and by a *key* which is supposed to be a secret shared only by the communicants.

In *symmetric cryptography*, the communicants all use the same key. Examples of symmetric cryptography ciphers include DES, AES, and PRESENT. The problem with symmetric cryptography is that the participants need to communicate the key in the first place, which raises a chicken-and-egg problem.

This problem was solved by *asymmetric cryptography*, which has been publicly[4] introduced in 1976. In asymmetric cryptography, two different but mathematically related keys are used: a public key and a private key. With the assumption, supported by mathematical properties, that it is practically not feasible to compute the private key given the public key. The public key, which is used for encryption, can be freely distributed, whereas the private key, which is used for decryption, must stay secret. Examples of asymmetric cryptography ciphers include RSA, DSA, and Elliptic-Curve Cryptography (ECC).

3.2.2 Cryptanalysis

As I explained while defining cryptology, *cryptanalysis* goes hand in hand with cryptography, with which it coevolved. Cryptanalysis is the study of the robustness

[2] An *algorithm* is a self-contained step-by-step set of operations to be performed.

[3] https://en.wikipedia.org/wiki/History_of_cryptography

[4] Declassified documents revealed in 1997 that at least GCHQ had invented asymmetric cryptography as well as the main algorithms and key exchange protocols a few years before the academic community published it independently.

of cryptographic algorithms. In general, the algorithms are known and the goal is to be able to decrypt any given ciphertext, i.e., to find the secret keys used by the communicants.

The goal of an attack is to find the keys faster than using *brute force*, which is the strategy consisting in trying all the possible keys one by one.

Cryptanalysis commonly categorizes attacks according to the amount of information that the attacker has access to, as in the following few examples. In *ciphertext-only attacks*, the attacker only has a set of ciphertexts. In *known-plaintext attacks*, the attacker has a set of ciphertexts for which the corresponding plaintext is known. In *chosen-plaintext attacks*, the attacker can obtain the ciphertexts corresponding to a set of chosen plaintexts. In *chosen-ciphertext attacks*, the attacker can obtain the plaintexts corresponding to a set of chosen ciphertexts. In *related-key attacks*, the attack can do the same as in a chosen-plaintext attack, except multiple different unknown but mathematically related (e.g., differing on a single bit) encryption keys can be used.

In addition to these categories, there are a few common attack methods, such as the ones in the following examples. In *differential cryptanalysis*, the attacker attempts to exploit randomness biases by studying how differences in plaintexts affects the ciphertexts. In *linear cryptanalysis*, the attacker tries to find affine approximations to the actions of the algorithms. In *man-in-the-middle attacks*, the attacker is actively participating in the communication, for instance relaying altered messages between the communicants.

As we can see, cryptanalysis has mainly been concerned with breaking algorithms at the mathematical level. Indeed, all the examples of attack category and attack method I listed are independent of any actual implementation.

3.2.3 Implementations

In the real world, cryptographic algorithms need to be implemented to be used, and they run *in fine* on physical devices.

Landauer's principle pertain to the lower theoretical limit of energy consumption of computation, says Wikipedia[5]. It is explained by Charles H. Bennett [Ben03] in the following term:

> "*Any logically irreversible manipulation of information, such as the erasure of a bit or the merging of two computation paths, must be accompanied by a corresponding entropy increase in non-information bearing degrees of freedom of the information processing apparatus or its environment.*"

In practice, a system performing a computation leaks information at every step. This behavior has opened new attack vectors exploiting these channels which are not classically modeled in cryptanalysis.

3.3 Side Channels

When a physical device is performing a computation, information about this computation leaks through what are called *side channels*. Side channels are physical

[5] https://en.wikipedia.org/wiki/Landauer's_principle, accessed 2015-02-24.

quantities such as time, power consumption, electromagnetic emanations, sound, or even light.

It is possible to exploit side channels to mount powerful attacks against cryptographic systems. Indeed, even though some side-channel attacks require knowledge of the internals of the implementation being attacked, many others such as Differential Power Analysis are effective as black-box attacks.

Side-channel attacks have been pioneered by Kocher et al. when they presented their timing attack paper at CRYPTO 1996 [Koc96]. Since then, it has become an international scientific field of both research and engineering.

More recently, side-channel attacks have spread as an attack vector onto other types of implementation than cryptographic embedded systems. For instance, web applications and the blooming of software-as-a-service has significantly raised the possibility of side-channel attacks on the web. Indeed, sensitive information may leak through non-functional quantities such as packet sizes. Another example is due to the growth of virtual servers for cloud hosting, where multiple logical machines run on a same physical server, giving the opportunity for a attacker to spy on cryptographic computations happening in a colocated virtual machine using cache timing attacks.

3.3.1 Passive Attacks

Side-channel attacks are called *passive* because the attacker does not act on the target system, but only reads leakage information.

A simple example of attack is the Simple Power Analysis (SPA). In SPA, the attacker traces the power consumption of the system while the algorithm runs. If the algorithm is not protected, there are good chances that its operations depends on the secret key. For instance, in the *square-and-multiply* exponentiation algorithm, a loop is iterated for each bit of the exponent, and each iteration consists of either one square operation if the bit is 0, or one square and one multiply operations if it is 1. Hence, even if all instructions consume the same amount of energy (which is unlikely in practice, but could be the case on specialized secure hardware), the difference can be seen instantly on an oscilloscope tracing the power consumption of the device running the algorithm. In that instance the SPA is similar to a specific case of timing attack.

These attacks can be made a lot more powerful by using multiple traces and then statistical methods to exploit the set of traces, as with Differential Power Analysis (DPA) [KJJ99], Correlation Power Analysis (CPA), or Template Attacks.

3.3.2 Countermeasures

Countermeasures against side-channel attacks exist at different levels. Since it is a physical problem, there are of course hardware-level countermeasures. As a computer scientist, I am more interested in software-level countermeasures. In most cases, the best protection can be achieved by using both, e.g., using secure hardware to support the software countermeasure's hypotheses.

In general, countermeasures can be classified under two categories: *palliative* and *curative*. Palliative countermeasures use randomness to blur the information the attacker has access too in the leakage, but without any strong theoretical foun-

dation. On the other hand, curative countermeasures aim at providing a leak-free implementation based on a security rationale. There are two strategies to achieve this: the first one is to make the leakage as decorrelated as possible from the sensitive data (*masking* [MOP06, Chp. 9]), and the second one is to make the leakage constant, irrespective of the sensitive data (*balancing* [MOP06, Chp. 7]).

3.4 Fault Injection

Another way of attacking cryptographic implementation is to put the system in an unexpected state during the computation in an attempt to have it reveal sensitive information. This is the strategy employed by *fault injection* attacks. In contrast with side channels, the idea is not for the attacker to (even indirectly) read intermediate values of the computation that should not be accessible, but rather to modify those values. Ideally, the induced disturbance cause the target algorithm to output a faulted value that can be exploited by the attacker.

These kinds of attacks on cryptographic algorithms have been introduced by Boneh et al. in 1997 [BDL97], while working at the *Bell Communication Research labs* (BellCoRe). They have shown that it is possible to discover the secret key of an RSA[6] computation using a single and easy fault injection.

However, it is interesting to note that many well-known cyber attacks could be classified as fault injection attacks. For example, the goal of a *buffer overflow* attack often consists in replacing data or code instructions from the target programs with data or instructions (e.g., a shellcode[7]) chosen by the attacker and which will help to break the system further. Thus, older cyber attacks are indeed very similar in concept to physical fault injection attacks.

3.4.1 Active Attacks

By opposition to side-channel attacks, fault injection attacks are called *active* because the attacker attempts to have an effect on the target system.

As with side channels, there are many ways of tampering with a device to inject a fault during its computation. Practically, injecting faults on an embedded system can be done using a laser or electromagnetic radiations, or the fault can be induced by tampering with the device's temperature, its clock, or its power supply. In the case of laser and electromagnetic radiations, the added energy can induce current in some wires of the circuits, and eventually affect the value stored in a register or transiting on a bus. In the case of temperature, clock, or power supply tampering, the principle is to put the devices in an environment in which it was not intended to work. For example, the voltage supply value has an effect on the speed at which data travel in the combinatory logic between memories. Thus, modifying the voltage supply for a short period of time (*power glitch*) can for instance prevent the result of a specific instruction to be stored in memory.

Depending on the setting, it is possible for the attacker to inject multiple faults that can be very accurate in timing and location, making it very difficult to protect embedded cryptographic systems. It is also feasible for the attacker to use the results from multiple faulted runs to perform Differential Fault Analysis (DFA).

[6] RSA (Rivest–Shamir–Adleman) is a widely used cipher, including in banking.
[7] https://en.wikipedia.org/wiki/Shellcode

As an example of fault attack, we can go back to the square-and-multiply example from the previous section. A protection method against the attack in the previous example would be to always perform both the squaring and the multiply operations, except making the square operation dummy (its result is not used) on loop iterations corresponding to an exponent bit at 0. This approach would as expected protect against the mentioned SPA attack, but it opens a fault injection vulnerability known as Safe-Error (SE) attacks. Indeed, when the attacker faults a squaring operation, if the algorithm still return the same result as when there is no fault injection, then it means that the faulted squaring operation was dummy and thus that this bit of the secret exponent is 0. Whereas if the result is different, the secret bit is 1.

3.4.2 Countermeasures

The goal of a countermeasure against fault injection attacks is to prevent the system to output a value exploitable by an attacker.

Here too there exist hardware- and software-level countermeasures. Hardware-level countermeasures attempt to make the circuit components more resistant to tampering, or use sensors to try to detect anomaly in the environment (sudden change in light, temperature or voltage supply are examples).

Again, I am more interested in countermeasures applicable at the software level. These countermeasures can be based on error-correcting code or on redundancy to detect breach in the integrity of the computation. There are two ways of detecting an integrity violation: either test for invariant and return an "error" value when a test fails, or have the computation infect its result with potential faults to make it unexploitable.

3.5 Formal Methods

The development of software and hardware is a difficult task. When it comes to critical software or hardware, as in the case of cryptographic systems, the difficulty is much increased. Indeed, the developed systems do not only have to be functionally correct, meaning an absence of bugs and a behavior corresponding to that of their specification. They also need to be correctly secured, meaning an absence of vulnerabilities that an attacker could exploit to break the security of the system.

Formal methods are mathematically based tools which can help software and hardware designers to ensure the correctness of their systems. Formal methods can act at different (and possibly all) stages of design process: specification, development, and verification. The principle is to use tools from mathematics and theoretical computer science to prove that the developed systems respect some functional and security properties.

Proofs can be carried out by hand, but when they become long and difficult, they are as much error-prone as the development of the system they are trying to prove, which means that the problem of trust has only been displaced. However, the idea of formal methods is that these proofs can also be mechanized. For instance, a proof technique can be modeled and programmed. Then, only a small and hopefully understandable program needs to be trusted and this program carries out the proof automatically on instances of the developed systems. In addition, the mechanized

approach also has the advantage of simplifying the process of proving another system, or even a new version of a proven system, which enables optimizations. Indeed, optimizations for speed are very dangerous in critical systems as they might compromise security measures without breaking the functional behavior of the system. This is why speed is often traded for security and why secured systems are most of the time slower. Being able to easily and quickly run a mechanized proof allows to use it as a non-regression test and thus enable the research for optimization.

For these reasons, I believe that formal methods are necessary to build trustable systems.

3.5.1 Implementation Security

The use of formal methods is not a widespread practice in the cryptologic research community. During my PhD I attended a school dedicated to the study of three tools implementing formal methods for cryptology. The Joint EasyCrypt–F*–CryptoVerif School 2014[8] was a success: it "brought together over 80 participants from 12 countries, with backgrounds spanning security, cryptography, programming languages, and formal methods", as a published report says[9]. What the report fails to mention is that almost all of the attendees had a formal methods background, whereas only a handful of people raised their hand when one of the lecturer asked "who here is a cryptologist?".

Sadly, this fact is especially true when it comes to implementation security against physical attacks. Indeed, as this field is very practical, the state-of-the-art research was, until recently at least, largely carried out by industrial research & development engineers. While the solutions they developed work in many cases, there is seldom an explanation of how or why they work. As a matter of fact many publications of countermeasures for instance reveal the fact that the design process was led by trial-and-error, patching vulnerabilities and trying to attack again, until a seemingly secure fixed-point was reached.

The fact that the physical behavior of a system is much more difficult to model than its functional properties does not help either, as formal methods tend to prove properties on models rather than directly on implementations. Similarly, formal methods used for the specification and development phases work on models with few exceptions, even if sometimes a real implementation can be extracted.

3.5.2 Formal Models

As formal methods work on models, applying them to implementation security requires to model parts of the physical behavior of the devices the cryptographic algorithms run on.

For instance, to formally prove the efficiency of a power analysis side-channel countermeasure it is necessary to model the power consumption leakage. Power consumption leakage in an integrated circuit is closely tied to the number of bit set to 1, as toggling the value of a bit and maintaining it to 1 consume power. Thus,

[8] https://wiki.inria.fr/prosecco/The_Joint_EasyCrypt-F*-CryptoVerif_School_2014

[9] http://prosecco.gforge.inria.fr/personal/hritcu/publications/school2014-siglog.pdf

the most common *leakage model* for power consumption leakage is the Hamming weight[10] of values, or the Hamming distance[11] of value updates.

Similarly, the high-level effect of a hardware fault has to be modeled to study fault injection attack countermeasures. For example these *fault models* include randomizing of intermediate variable, zeroing of intermediate variable, instruction skip or replacement.

3.6 Goals of this Thesis

Implementations of cryptosystems are vulnerable to physical attacks, and thus need to be protected against them. Of course, malfunctioning protections are useless. Formal methods help to develop systems while assessing their conformity to a rigorous specification. The first goal of my thesis, and its innovative aspect, is to show that formal methods can be used to prove not only the principle of countermeasures according to a model but also their implementations, as it is where the physical vulnerabilities are exploited. My second goal is the proof and the automation of the protection techniques themselves, because handwritten security code is error-prone.

Physical attacks can be classified into two distinct categories. Passive attacks, where the attacker only reads information that leaks through *side channels* (such as power consumption or electromagnetic emanations). And active attacks, where the attacker tampers with the system to have it reveal secrets through its "normal" output channel. Therefore, I have pursued my goals in both settings: on a countermeasure that diminishes side-channel leakage, and on countermeasures which detect *fault injection* attacks.

3.6.1 Contributions

As there already exist rigorous security properties for protections against side-channel leakage [MOP06, Chp. 7 & 9], my contributions concentrate on formal methods for design and verification of protected implementations of algorithms. In chapter 4, one will find:

- A methodology to protect an implementation against side-channel attacks by generating an improved version of it which has a null side-channel signal-to-noise ratio, as its leakage is made constant (in particular, it does not depend on the sensitive values) by using a *software* Dual-rail with Precharge Logic (DPL) balancing protocol.
- A proof that the induced code transformation is correct (semantic preserving).
- A compiler named paioli that takes assembly code and protect it by applying the aforementioned code transformation.
- A tool, integrated with the compiler, able to independently prove that this constant leakage property holds for a given implementation, which can be used to assess the security of the output of the compiler.

To my best knowledge, paioli is the first tool which can automatically protect an implementation against side-channel attacks by provably balancing its leakage.

[10] The Hamming weight of a value is the number of 1s in its binary representation.

[11] The Hamming distance of a value update is the number of bit flips necessary to go from the binary representation of its old value to the one of its new value.

In contrast, the definition of the security objectives was not clearly established for fault injection attacks before I started my PhD. Many countermeasures have been published without any proofs, as the necessary properties to prove had not been formally expressed. It is only during my PhD that so-called "attack success conditions" were introduced. In chapters 5, 6, and 7, one will find:

- A theoretical framework based on arithmetic to evaluate existing countermeasure with respect to these attack success conditions.
- A tool named finja which does symbolic evaluation by rewriting according to arithmetic rules, which allowed to study, prove, and simplify existing countermeasures.
  Example: the original Vigilant's CRT-RSA[12] countermeasure was broken, used 9 tests, and needed 5 random numbers; our final version can be proved, works with less code, uses 3 tests, and needs only 1 random number.
- A classification of a family of existing countermeasures which allowed to extract protection principles from the employed techniques.
- A method for designing a countermeasure that can resist an arbitrary (but fixed) number of faults.

The method I have developed and the finja tool I have implemented have also been used by other researchers to study another family of countermeasures and to obtain new results [Kis14].

I have also noticed that all the countermeasures I have studied are variations of optimization for the same base technique consisting in verifying the integrity of the computation using a form of redundancy. This technique is independent of the algorithm it is applied to and of the attack success conditions, as it only depends on the mathematical structure of the sensitive data, namely modular arithmetic. I have thus proposed a property of resistance against fault injection attacks which goes beyond attack success conditions. In chapter 8, one will find:

- An abstraction of the principle of the CRT-RSA countermeasures against Bell-CoRe attacks into a protection scheme that we call *entanglement*, and that is shown to be applicable to all of asymmetric cryptographic computations.
- A provably correct compiler named enredo which automates this transformation and allowed to obtain protected implementations of cryptographic algorithms for which no countermeasures were known until now, but which were already victims of fault injection attacks.

An additional advantage of the enredo compiler is that it allows to study the trade-off between security and complexity.

I have disseminated my contributions in 5 articles accepted at international conferences and journals[13], and 12 talks at conferences and seminars. I have two other articles in preparation, as well as a book chapter that will appear in *Handbook of Pairing Based Cryptography*, Nadia El Mrabet and Marc Joye, editors.

[12] "CRT" stands for "Chinese Remainder Theorem", which is a common optimization technique for RSA.

[13] My publications are listed in Appendix C, page 153.

0x4

Formally Proved Security of Assembly Code Against Power Analysis

This chapter presents a joint work with Sylvain Guilley as my adviser, and Zakaria Najm, the research engineer who helped me with the lab work for the case study. A first version of this work was accepted at the PROOFS 2014 workshop, and an extended version will appear in the *Journal of Cryptographic Engineering*.

Abstract. In his keynote speech at CHES 2004, Kocher advocated that side-channel attacks were an illustration that formal cryptography was not as secure as it was believed because some assumptions (e.g., no auxiliary information is available during the computation) were not modeled. This failure is caused by formal methods' focus on models rather than implementations. In this chapter we present formal methods and tools for designing protected code and proving its security against power analysis. These formal methods avoid the discrepancy between the model and the implementation by working on the latter rather than on a high-level model. Indeed, our methods allow us (a) to automatically insert a power balancing countermeasure directly at the assembly level, and to prove the correctness of the induced code transformation; and (b) to prove that the obtained code is balanced with regard to a reasonable leakage model. We also show how to characterize the hardware to use the resources which maximize the relevancy of the model. The tools implementing our methods are then demonstrated in a case study on an 8-bit AVR smartcard for which we generate a provably protected PRESENT implementation that reveals to be at least 250 times more resistant to CPA attacks.

Contents

4.1 Introduction

The need to trust code is a clear and proved fact, but the code itself needs to be proved before it can be trusted. In applications such as cryptography or real-time systems, formal methods are used to prove functional properties on the critical parts of the code. Specifically in cryptography, some non-functional properties are also important, but are not typically certified by formal proofs yet. One example of such a property is the resistance to side-channel attacks. Side-channel attacks are a real world threat to cryptosystems; they exploit auxiliary information gathered from implementations through physical channels such as power consumption, electromagnetic radiations, or time, in order to extract sensitive information (e.g., secret keys). The amount of leaked information depends on the implementation and as such appears difficult to model. As a matter of fact, physical leakages are usually not modeled when it comes to prove the security properties of a cryptographic algorithm. By applying formal methods directly on implementations we can avoid the discrepancy between the model and the implementation. Formally proving non-functional security properties then becomes a matter of modeling the leakage itself. In this chapter we make a first step towards formally trustable cryptosystems, including for non-functional properties, by showing that modeling leakage and applying formal methods to implementations is feasible.

Many existing countermeasures against side-channel attacks are implemented at the hardware level, especially for smartcards. However, software level countermeasures are also very important, not only in embedded systems where the hardware cannot always be modified or updated, but also in the purely software world. For example, Zhang et al. [ZJRR12] recently extracted private keys using side-channel attacks against a target virtual machine running on the same physical server as their virtual machine. Side channels in software can also be found each time there are some non-logic behaviors (in the sense that it does not appear in the equations / control-flow modeling the program) such as timing or power consumption (refer to [KJJ99]), but also some software-specific information such as packet size for instance (refer to [MO12]).

In many cases where the cryptographic code is executed on secure elements (smartcards, TPM, tokens, etc.) side-channel and fault analyses are the most natural attack paths. A combination of signal processing and statistical techniques on the data obtained by side-channel analysis allows to build key hypotheses distinguishers. The protection against those attacks is necessary to ensure that secrets do not leak, and most secure elements are thus evaluated against those attacks. Usual certifications are the common criteria (ISO/IEC 15408), the FIPS 140-2 (ISO/IEC 19790), or proprietary schemes (EMVCo, CAST, etc.).

Power analysis. It is a form of side-channel attack in which the attacker measures the power consumption of a cryptographic device. *Simple Power Analysis* (SPA) consists in directly interpreting the electrical activity of the cryptosystem. On unprotected implementations it can for instance reveal the path taken by the code at branches even when timing attacks [Koc96] cannot. *Differential Power Analysis* (DPA) [KJJ99] is more advanced: the attacker can compute the intermediate values within cryptographic computations by statistically analyzing data collected from multiple cryptographic operations. It is powerful in the sense that it does not require a precise model of the leakage, and thus works blind, i.e., even if the

implementation is blackbox. As suggested in the original DPA paper by Kocher et al. [KJJ99], power consumption is often modeled by Hamming weight of values or Hamming distance of values' updates as those are very correlated with actual measures. Also, when the leakage is little noisy and the implementation is software, *Algebraic Side-Channel Attack* (ASCA) [RS09] are possible; they consist in modelling the leakage by a set of Boolean equations, where the key bits are the only unknown variables [CFGR12].

Thwarting side-channel analysis is a complicated task, since an unprotected implementation leaks at every step. Simple and powerful attacks manage to exploit any bias. In practice, there are two ways to protect cryptosystems: "palliative" versus "curative" countermeasures. Palliative countermeasures attempt to make the attack more difficult, however without a theoretical foundation. They include variable clock, operations shuffling, and dummy encryptions among others (see also [GM11]). The lack of theoretical foundation make these countermeasures hard to formalize and thus not suitable for a safe certification process. Curative countermeasures aim at providing a leak-free implementation based on a security rationale. The two defense strategies are (a) make the leakage as decorrelated from the manipulated data as possible (*masking* [MOP06, Chp. 9]), or (b) make the leakage constant, irrespective of the manipulated data (hiding or *balancing* [MOP06, Chp. 7]).

Masking. Masking mixes the computation with random numbers, to make the leakage (at least in average) independent of the sensitive data. Advantages of masking are (*a priori*) the independence with respect to the leakage behavior of the hardware, and the existence of provably secure masking schemes [RP10]. There are two main drawbacks to masking. First of all, there is the possibility of high-order attacks (that examine the variance or the joint leakage); when the noise is low, ASCAs can be carried out on one single trace [RSVC09], despite the presence of the masks, that are just seen as more unknown variables, in addition to the key. Second, masking demands a greedy requirement for randomness (that is very costly to generate). Another concern with masking is the overhead it incurs in the computation time. For instance, a provable masking of AES-128 is reported in [RP10] to be 43 (resp. 90) times slower than the non-masked implementation with a 1st (resp. 2nd) order masking scheme. Further, recent studies have shown that masking cannot be analyzed independently from the execution platform: for example *glitches* are transient leakages that are likely to depend on more than one sensitive data, hence being high-order [MS06]. Indeed, a glitch occurs when there is a race between two signals, i.e., when it involves more than one sensitive variable. Additionally, the implementation must be carefully scrutinized to check for the absence of *demasking* caused by overwriting a masked sensitive variable with its mask.

Balancing. Balancing requires a close collaboration between the hardware and the software: two indistinguishable resources, from a side-channel point of view, shall exist and be used according to a dual-rail protocol. *Dual-rail with Precharge Logic* (DPL) consists in precharging both resources, so that they are in a common state, and then setting one of the resources. Which resource has been set is unknown to the attacker, because both leak in indistinguishable ways (by hypothesis). This property is used by the DPL protocol to ensure that computations can be carried out without exploitable leakage [TV06].

Contributions. Dual-rail with Precharge Logic (DPL) is a simple protocol that may look easy to implement correctly; however, in the current context of awareness about cyber-threats, it becomes evident that (independent) formal tools that are able to *generate* and *verify* a "trusted" implementation have a strong value.

- We describe a design method for developing balanced assembly code by making it obey the DPL protocol. This method consists in automatically inserting the countermeasure and formally proving that the induced code transformation is correct (i.e., semantic preserving).
- We present a formal method (using symbolic execution) to statically prove the absence of power consumption leakage in assembly code provided that the hardware it runs on satisfies a finite and limited set of requirements corresponding to our leakage model.
- We show how to characterize the hardware to run the DPL protocol on resources which maximize the relevancy of the leakage model.
- We provide a tool called paioli[1] which implements the automatic insertion of the DPL countermeasure in assembly code, and, independently, is able to statically prove the power balancing of a given assembly code.
- Finally, we demonstrate our methods and tool in a case study on a software implementation of the PRESENT [BKL+07] cipher running on an 8-bit AVR micro-controller. Our practical results are very encouraging: the provably balanced DPL protected implementation is *at least* 250 times more resistant to power analysis attacks than the unprotected version while being only 3 times slower. The Signal-to-Noise Ratio (SNR) of the leakage is divided by approximately 16.

Related work. The use of formal methods is not widespread in the domain of implementations security. In cases where they exist, security proofs are usually done on mathematical models rather than implementations. An emblematic example is the Common Criteria [Con13], that bases its "formal" assurance evaluation levels on "Security Policy Model(s)" (class SPM) and not on implementation-level proofs. This means that it is the role of the implementers to ensure that their implementations fit the model, which is usually done by hand and is thus error-prone. For instance, some masking implementations have been proved; automatic tools for the insertion of masked code have even been prototyped [MOPT12]. However, masking relies a lot on randomness, which is a rare resource and is hard to formally capture. Thus, many aspects of the security are actually displaced in the randomness requirement rather that soundly proved. Moreover, in the field of masking, most proofs are still literate (i.e., verified manually, not by a computer program). This has led to a recent security breach in what was supposed to be a proved [RP10] masking implementation [CGP+12]. Previous similar examples exist, e.g., the purported high-order masking scheme [SP06], defeated one year after in [CPR07].

Timing and cache attacks are an exception as they benefit from the work of Köpf et al. [KB07, KD09]. Their tool, CacheAudit [DFK+13], implements formal methods that directly work on x86 binaries.

Since we started our work on DPL, others have worked on similar approaches. Independently, it has been shown that SNR reduction is possible with other encodings that are less costly, such as "dual-nibble" (Chen et al. [CESY14]) or "m-out-of-n"

[1] http://pablo.rauzy.name/sensi/paioli.html

(Servant et al. [SDMB14]). However, it becomes admittedly much more difficult to balance the resources aimed at hiding one each other. Thus, there is a trade-off between performance (in terms of execution speed and code size) and security. In this chapter we propose a proof-of-concept of maximal security.

In this light it is easy to conclude that the use of formal methods to prove the security of implementations against power analysis is a need, and a technological enabler: it would guarantee that the instantiations of security principles are as strong as the security principles themselves.

Organization of the chapter. The DPL countermeasure is studied in Sec. 4.2. Sec. 4.3 details our method to balance assembly code and prove that the proposed transformation is correct. Sec. 4.4 explains the formal methods used to compute a proof of the absence of power consumption leakage. Sec. 4.5 is a practical case study using the PRESENT algorithm on an AVR micro-controller. Conclusions and perspectives are drawn in Sec. 4.6.

4.2 Dual-rail with Precharge Logic

Balancing (or hiding) countermeasures have been employed against side channels since early 2004, with dual-rail with precharge logic. The DPL countermeasure consists in computing on a redundant representation: each bit y is implemented as a pair $(y_{\text{False}}, y_{\text{True}})$. The bit pair is then used in a protocol made up of two phases:

1. a *precharge* phase, during which all the bit pairs are zeroized $(y_{\text{False}}, y_{\text{True}}) = (0, 0)$, such that the computation starts from a known reference state;
2. an *evaluation* phase, during which the $(y_{\text{False}}, y_{\text{True}})$ pair is equal to $(1, 0)$ if it carries the logical value 0, or $(0, 1)$ if it carries the logical value 1.

The value $(y_{\text{False}}, y_{\text{True}}) = (1, 1)$ is unused. As suggested in [MAM+03], it can serve as a *canary* to detect a fault. Besides, if a fault turns a $(1, 0)$ or $(0, 1)$ value into either $(0, 0)$ or $(1, 1)$, then the previous functional value has been forgiven. It is a type of infection, already mentioned in [IPSW06, SBG+09]. Unlike other infective countermeasure, DPL is not scary [BG13], in that it consists in an erasure. Indeed, the mutual information between the erased and the initial data is zero (provided only one bit out of a dual pair is modified).

4.2.1 State of the Art

Various DPL styles for electronic circuits have been proposed. Some of them, implementing the same logical *and* functionality, are represented in Fig. 4.1; many more variants exist, but these four are enough to illustrate our point. The reason for the multiplicity of styles is that the indistinguishability hypothesis on the two resources holding y_{False} and y_{True} values happens to be violated for various reasons, which leads to the development of dedicated hardware. A first asymmetry comes from the gates driving y_{False} and y_{True}. In *Wave Dynamic Differential Logic* (WDDL) [TV04a], these two gates are different: logical *or* versus logical *and*. Other logic styles are balanced with this respect. Then, the load of the gate shall also be similar. This can be achieved by careful place-and-route constraints [TV04b, GHMP05], that take care of having lines of the same length, and that furthermore do not interfere one with the other (phenomenon called “crosstalk”). As those are complex to implement exactly

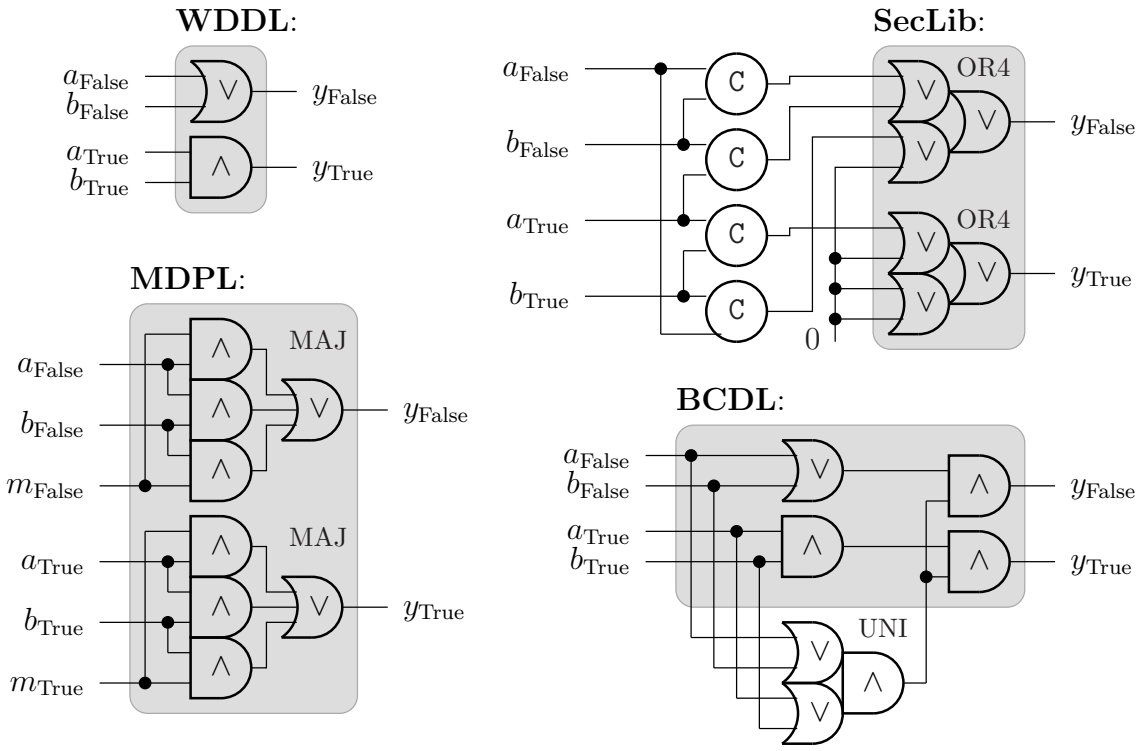


Figure 4.1: Four dual-rail with precharge logic styles.

for all secure gates, the *Masked Dual-rail with Precharge Logic* (MDPL) [PM05] style has been proposed: instead of balancing exactly the lines carrying y_{False} and y_{True}, those are randomly swapped, according to a random bit, represented as a pair $(m_{\text{False}}, m_{\text{True}})$ to avoid it from leaking. Therefore, in this case, not only the computing gates are the same (*viz.* a majority), but the routing is balanced thanks to the extra mask. However, it appeared that another asymmetry could be fatal to WDDL and MDPL: the gates pair could evaluate at different dates, depending on their input. It is important to mention that side-channel acquisitions are very accurate in timing (off-the-shelf oscilloscopes can sample at more than 1 Gsample/s, i.e., at a higher rate than the clock period), but very inaccurate in space (i.e., it is difficult to capture the leakage of an area smaller than about 1 mm^2 without also recording the leakage from the surrounding logic). Therefore, two bits can hardly be measured separately. To avoid this issue, every gate has to include some synchronization logic. In Fig. 4.1, the "computation part" of the gates is represented in a grey box. The rest is synchronization logic. In SecLib [GCS+08], the synchronization can be achieved by Muller C-elements (represented with a symbol C [SEE98]), and act as a decoding of the inputs configuration. Another implementation, *Balanced Cell-based Differential Logic* (BCDL) [NBD+10], parallelize the synchronization with the computation.

4.2.2 DPL in Software

In this chapter, we want to run DPL on an off-the-shelf processor. Therefore, we must: (a) identify two similar resources that can hold true and false values in an indiscernible way for a side-channel attacker; (b) play the DPL protocol by ourselves, in software. We will deal with the former in Sec. 4.4.2. The rest of this section deals

with the latter.

The difficulty of balancing the gates in hardware implementations is simplified in software. Indeed in software there are less resources than the thousands of gates that can be found in hardware (aimed at computing fast, with parallelism). Also, there is no such problem as early evaluation, since the processor executes one instruction after the other; therefore there are no unbalanced paths in timing. However, as noted by Hoogvorst et al. [HDD11], standard micro-processors cannot be used *as is* for our purpose: instructions may clobber the destination operand without precharge; arithmetic and logic instructions generate numbers of 1 and 0 which depend on the data.

To reproduce the DPL protocol in software requires (a) to work at the bit level, and (b) to duplicate (in positive and negative logic) the bit values. Every algorithm can be transformed so that all the manipulated values are bits (by the theorem of equivalence of universal Turing machines), so (*a*) is not a problem. Regarding (*b*), the idea is to use two bits in each register / memory cell to represent the logical value it holds. For instance using the two least significant bits, the logical value 1 could be encoded as 1 (`01`) and the logical value 0 as 2 (`10`). Then, any function on those bit values can be computed by a look-up table indexed by the concatenation of its operands. Each sensitive instruction can be replaced by a *DPL macro* which does the necessary precharge and fetch the result from the corresponding look-up table.

$$
\begin{array}{rcl}
r_1 & \leftarrow & r_0 \\
r_1 & \leftarrow & a \\
r_1 & \leftarrow & r_1 \wedge 3 \\
r_1 & \leftarrow & r_1 \ll 1 \\
r_1 & \leftarrow & r_1 \ll 1 \\
r_2 & \leftarrow & r_0 \\
r_2 & \leftarrow & b \\
r_2 & \leftarrow & r_2 \wedge 3 \\
r_1 & \leftarrow & r_1 \vee r_2 \\
r_3 & \leftarrow & r_0 \\
r_3 & \leftarrow & op[r_1] \\
d & \leftarrow & r_0 \\
d & \leftarrow & r_3
\end{array}
$$

Figure 4.2: DPL macro for $d = a$ op b.

Fig. 4.2 shows a DPL macro for the computation of $d = a$ op b, using the two least significant bits for the DPL encoding. The register r_0 is an always-zero register, a and b hold one DPL encoded bit, and *op* is the address in memory of the look-up table for the op operation.

This DPL macro assumes that before it starts the state of the program is a valid DPL state (i.e., that a and b are of the form `/.+(01|10)/`[2]) and leaves it in a valid DPL state to make the macros chainable.

The precharge instructions (like $r_1 \leftarrow r_0$) erase the content of their destination register or memory cell before use. If the erased datum is sensitive it is DPL encoded, thus the number of bit flips (i.e., the Hamming distance of the update) is independent of the sensitive value. If the erased value is not sensitive (for example the round counter of a block cipher) then the number of bit flips is irrelevant. In both cases the power consumption provides no sensitive information.

The activity of the shift instructions (like $r_1 \leftarrow r_1 \ll 1$) is twice the number of DPL encoded bits in r_1 (and thus does not depend on the value when it is DPL encoded). The two most significant bits are shifted out and must be 0, i.e., they cannot encode a DPL bit. The logical *or* instruction (as in $r_1 \leftarrow r_1 \vee r_2$) has a constant activity of one bit flip due to the alignment of its operands. The logical *and* instructions (like $r_1 \leftarrow r_1 \wedge 3$) flips as many bits as there are 1s after the two least significant bits (it's normally all zeros).

Accesses from/to the RAM (as in $r_3 \leftarrow op[r_1]$) cause as many bit flips as there are 1s in the transferred data, which is constant when DPL encoded. Of course, the

[2] As a convenience, we use regular expressions notation.

position of the look-up table in the memory is also important. In order not to leak information during the addition of the offset ($op + r_1$ in our example), op must be a multiple of 16 so that its four least significant bits are 0 and the addition only flips the number of bits at 1 in r_1, which is constant since at this moment r_1 contains the concatenation of two DPL encoded bit values.

We could use other bits to store the DPL encoded value, for example the least and the third least significant bits. In this case a and b have to be of the form `/.+(0.1|1.0)/`, only one shift instruction would have been necessary, and the `and` instructions' mask would be 5 instead on 3.

4.3 Generation of DPL Protected Assembly Code

Here we present a generic method to protect assembly code against power analysis. To achieve that we implemented a tool (See App. B.1) which transforms assembly code to make it compliant with the DPL protocol described in Sec. 4.2.2. To be as universal as possible the tool works with a generic assembly language presented in Sec. 4.3.1. The details of the code transformation are given in Sec. 4.3.2. Finally, a proof of the correctness of this transformation is presented in Sec. 4.3.3.

We implemented paioli[1] using the OCaml[3] programming language, which type safety helps to prevent many bugs. On our PRESENT case-study, it runs in negligible time ($\ll 1$ second), both for DPL transformation and simulation, including balance verification. The unprotected (resp. DPL) bitslice AVR assembly file consists of 641 (resp. 1456) lines of code. We use nibble-wise jumps in each PRESENT operation, and an external loop over all rounds.

4.3.1 Generic Assembly Language

Our assembly language is generic in that it uses a restricted set of instructions that can be mapped to and from virtually any actual assembly language. It has the classical features of assembly languages: logical and arithmetical instructions, branching, labels, direct and indirect addressing. Fig. 4.3 gives the Backus–Naur Form (BNF) of the language while Fig. 4.4 gives the equivalent code of Fig. 4.2 as an example of its usage.

The semantics of the instructions are intuitive. For `Opcode2` and `Opcode3` the first operand is the destination and the other are the arguments. The `mov` instruction is used to copy registers, load a value from memory, or store a value to memory depending on the form of its arguments. We remark that the instructions use the "`instr dest op1 op2`" format, which allows to map similar instructions from 32-bit processors directly, as well as instructions from 8-bit processors which only have two operands, by using the same register for `dest` and `op1` for instance.

4.3.2 Code Transformation

Bitsliced code. As seen in Sec. 4.2, DPL works at the bit level. Transforming code to make it DPL compliant thus requires this level of granularity. Bitslicing

[3] http://ocaml.org/

```
Prog    ::= ( Label? Inst? ( ';' <comment> )? '\n' )*
Label   ::= <label-name> ':'
Inst    ::= Opcode0
          | Branch1 Addr
          | Opcode2 Lval Val
          | Opcode3 Lval Val Val
          | Branch3 Val Val Addr
Opcode0 ::= 'nop'
Branch1 ::= 'jmp'
Opcode2 ::= 'not' | 'mov'
Opcode3 ::= 'and' | 'orr' | 'xor' | 'lsl' | 'lsr'
          | 'add' | 'mul'
Branch3 ::= 'beq' | 'bne'
Val     ::= Lval | '#' <immediate-value>
Lval    ::= 'r' <register-number>
          | '@' <memory-address>
          | '!' Val ( ',' <offset> )?
Addr    ::= '#' <absolute-code-address>
          | <label-name>
```

Figure 4.3: Generic assembly syntax (BNF).

is possible on any algorithm[4], but we found that bitslicing an algorithm is hard to do automatically. In practice, every bitslice implementations we found were hand-crafted. However, since Biham presented his bitslice paper [Bih97], many block ciphers have been implemented in bitslice for performance reasons, which mitigate this concern. So, for the sake of simplicity, we assume that the input code is already bitsliced.

DPL macros expansion. This is the main point of the transformation of the code.

Definition 4.1 (Sensitive value)**.** A *value* is said *sensitive* if it depends on sensitive data. A sensitive data depends on the secret key or the plaintext[5].

Definition 4.2 (Sensitive instruction)**.** We say that an *instruction* is *sensitive* if it may modify the Hamming weight of a sensitive value.

All the sensitive instructions must be expanded to a DPL macro. Thus, all the sensitive data must be transformed too. Each literal ("immediate" values in assembly terms), memory cells that contain initialized constant data (look-up tables, etc.), and registers values need to be DPL encoded. For instance, using the two least significant bits, the 1s stay 1s (`01`) and the 0s become 2s (`10`).

[4]Intuitively, the proof invokes the Universal Turing Machines equivalence (those that work with only $\{0, 1\}$ as alphabet are as powerful as the others).

[5]Other works consider that a sensitive data must depend on both the secret key and the plaintext (as it is usually admitted in the "*only computation leaks*" paradigm; see for instance [RP10, §4.1]). Our definition is broader, in particular it also encompasses the random probing model [ISW03].

Table 4.1: Look-up tables for `and`, `or`, and `xor`.

idx	0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111
and	00 , 00 , 00 , 00 , 00 , 01 , 10 , 00 , 00 , 10 , 10 , 00 , 00 , 00 , 00 , 00
or	00 , 00 , 00 , 00 , 00 , 01 , 01 , 00 , 00 , 01 , 10 , 00 , 00 , 00 , 00 , 00
xor	00 , 00 , 00 , 00 , 00 , 10 , 01 , 00 , 00 , 01 , 10 , 00 , 00 , 00 , 00 , 00

Since the implementation is bitsliced, only the logical (bit level) operators are used in sensitive instructions (`and`, `or`, `xor`, `lsl`, `lsr`, and `not`). To respect the DPL protocol, `not` instructions are replaced by `xor` which inverse the positive logic and the negative logic bits of DPL encoded values. For instance if using the two least significant bits for the DPL encoding, `not` *a b* is replaced by `xor` *a b* `#3`. Bitsliced code never needs to use shift instructions since all bits are directly accessible.

Moreover, we currently run this code transformation only on block ciphers. Given that the code is supposed to be bitsliced, this means that the branching and arithmetic instructions are either not used or are used only in a deterministic way (e.g., looping on the round counter) that does not depend on sensitive information.

```
mov r1 r0
mov r1 a
and r1 r1 #3
lsl r1 r1 #1
lsl r1 r1 #1
mov r2 r0
mov r2 b
and r2 r2 #3
orr r1 r1 r2
mov r3 r0
mov r3 !r1,op
mov d r0
mov d r3
```

Figure 4.4: DPL macro of Fig. 4.2 in assembly.

Thus, only `and`, `or`, and `xor` instructions need to be expanded to DPL macros such as the one shown in Fig. 4.4. This macro has the advantage that it actually uses two operands instructions only (when there are three operands in our generic assembly language, the destination is the same as one of the two others), which makes its instructions mappable one-to-one even with 8-bit assembly languages.

Look-up tables. As they appear in the DPL macro, the addresses of look-up tables are sensitive too. As seen in Sec. 4.2.2, the look-up tables must be located at an address which is a multiple of 16 so that the last four bits are available when adding the offset (in the case where we use the last four bits to place the two DPL encoded operands). Tab. 4.1 present the 16 values present in the look-up tables for `and`, `or`, and `xor`.

Values in the look-up tables which are not at DPL valid addresses, i.e., addresses which are not a concatenation of `01` or `10` with `01` or `10`, are preferentially DPL invalid, i.e., `00` or `11`. Like this if an error occurs during the execution (such as a fault injection for instance) it poisons the result and all the subsequent computations will be faulted too (infective computation).

4.3.3 Correctness Proof of the Transformation

Formally proving the correctness of the transformation requires to define what we intend by "correct". Intuitively, it means that the transformed code does the "same thing" as the original one.

Definition 4.3 (Correct DPL transformation)**.** Let S be a valid state of the system (values in registers and memory). Let c be a sequence of instructions of the system.

Table 4.2: `anddab`.

	a, b, d			
Before	0, 0, ?	0, 1, ?	1, 0, ?	1, 1, ?
After	0, 0, 0	0, 1, 0	1, 0, 0	1, 1, 1

Table 4.3: DPL `anddab`.

	a, b, d			
Before	10, 10, ?	10, 01, ?	01, 10, ?	01, 01, ?
After	10, 10, 10	10, 01, 10	01, 10, 10	01, 01, 01

Let $\widehat{S}$ be the state of the system after the execution of c with state S, we denote that by $S \xrightarrow{c} \widehat{S}$. We write $dpl(S)$ for the DPL state (with DPL encoded values of the 1s and 0s in memory and registers) equivalent to the state S.
We say that c' is a *correct DPL transformation* of c if $S \xrightarrow{c} \widehat{S} \implies dpl(S) \xrightarrow{c'} dpl(\widehat{S})$.

Proposition 4.1 (Correctness of our code transformation)**.** *The expansion of the sensitive instructions into DPL macros such as presented in Sec. 4.2.2 is a correct DPL transformation.*

Proof. Let a and b be instructions. Let c be the code $a; b$ (instruction a followed by instruction b). Let X, Y, and Z be states of the program. If we have $X \xrightarrow{a} Y$ and $Y \xrightarrow{b} Z$, then we know that $X \xrightarrow{c} Z$ (by *transitivity*).

Let a' and b' be the DPL macro expansions of instructions a and b. Let c' be the DPL transformation of code c. Since the expansion into macros is done separately for each sensitive instruction, without any other dependencies, we know that c' is $a'; b'$.
If we have $dpl(X) \xrightarrow{a'} dpl(Y)$ and $dpl(Y) \xrightarrow{b'} dpl(Z)$, then we know that $dpl(X) \xrightarrow{c'} dpl(Z)$.

This means that a chain of correct transformations is a correct transformation. Thus, we only have to show that the DPL macro expansion is a correct transformation.

Let us start with the `and` operation. Since the code is bitsliced, there are only four possibilities. Tab. 4.2 shows these possibilities for the `and d a b` instruction.

Tab. 4.4 shows the evolution of the values of a, b, and d during the execution of the macro which `and d a b` expands to. We assume the look-up table for `and` is located at address *and*. Tab. 4.3 sums up the Tab. 4.4 in the same format as Tab. 4.2.

This proves that the DPL transformation of the `and` instructions are correct. The demonstration is similar for `or` and `xor` operations. □

The automatic DPL transformation of arbitrary assembly code has been implemented in our tool described in App. B.1.

4.4 Formally Proving the Absence of Leakage

Now that we know the DPL transformation is correct, we need to prove its efficiency security-wise. We prove the absence of leakage on the software, while obviously the leakage heavily depends on the hardware. Our proof thus makes an hypothesis on the hardware: we suppose that the bits we use for the positive and negative logic in the DPL protocol leak the same amount. This may seem like an unreasonable hypothesis, since it is not true in general. However, the protection can be implemented

Table 4.4: Execution of the DPL macro expanded from `and d a b`.

a, b	10, 10				10, 01				01, 10				01, 01			
	d	r1	r2	r3	d	r1	r2	r3	d	r1	r2	r3	d	r1	r2	r3
`mov r1 r0`	?	0	?	?	?	0	?	?	?	0	?	?	?	0	?	?
`mov r1` a	?	10	?	?	?	10	?	?	?	01	?	?	?	01	?	?
`and r1 r1 #3`	?	10	?	?	?	10	?	?	?	01	?	?	?	01	?	?
`shl r1 r1 #1`	?	100	?	?	?	100	?	?	?	010	?	?	?	010	?	?
`shl r1 r1 #1`	?	1000	?	?	?	1000	?	?	?	0100	?	?	?	0100	?	?
`mov r2 r0`	?	1000	0	?	?	1000	0	?	?	0100	0	?	?	0100	0	?
`mov r2` b	?	1000	10	?	?	1000	01	?	?	0100	10	?	?	0100	01	?
`and r2 r2 #3`	?	1000	10	?	?	1000	01	?	?	0100	10	?	?	0100	01	?
`orr r1 r1 r2`	?	1010	10	?	?	1001	01	?	?	0110	10	?	?	0101	01	?
`mov r3 r0`	?	1010	10	0	?	1001	01	0	?	0110	10	0	?	0101	01	0
`mov r3 !r1,` and[6]	?	1010	10	10	?	1001	01	10	?	0110	10	10	?	0101	01	01
`mov` d `r0`	0	1010	10	10	0	1001	01	10	0	0110	10	10	0	0101	01	01
`mov` d `r3`	10	1010	10	10	10	1001	01	10	10	0110	10	10	01	0101	01	01

in a soft CPU core (LatticeMicro32, OpenRISC, LEON2, etc.), that would be laid out in a Field-Programmable Gate Array (FPGA) or in an Application-Specific Integrated Circuit (ASIC) with special balancing constraints at place-and-route. The methodology follows the guidelines given by Chen et al. in [CSS13]. Moreover, we will show in Sec. 4.4.2 how it is possible, using stochastic profiling, to find bits which leakages are similar enough for the DPL countermeasure to be sufficiently efficient even on non-specialized hardware. That said, it is important to note that the difference in leakage between two bits of the same register should not be large enough for the attacker to break the DPL protection using SPA or ASCA.

Formally proving the balance of DPL code requires to properly define the notions we are using.

Definition 4.4 (Leakage model)**.** The attacker is able to measure the power consumption of parts of the cryptosystem. We model power consumption by the Hamming distance of values updates, i.e., the number of bit flips. It is a commonly accepted model for power analysis, for instance with DPA [KJJ99] or Correlation Power Analysis (CPA) [BCO04]. We write $H(a, b)$ the Hamming distance between the values a and b.

Definition 4.5 (Constant activity)**.** The activity of a cryptosystem is said to be constant if its power consumption does not depend on the sensitive data and is thus always the same.
Formally, let $P(s)$ be a program which has s as parameter (e.g., the key and the plaintext). According to our leakage model, a program $P(s)$ is of *constant activity* if:

- for every values s_1 and s_2 of the parameter s, for each cycle i, for every sensitive value v, v is updated at cycle i in the run of $P(s_1)$ if and only if it is updated also at cycle i in the run of $P(s_2)$;
- whenever an instruction modifies a sensitive value from v to v', then the value of $H(v, v')$ does not depend on s.

Remark 4.1. The first condition of Def. 4.5 mostly concerns leakage in the horizontal / time dimension, while the second condition mostly concerns leakage in the vertical / amplitude dimension.

Remark 4.2. The first condition of Def. 4.5 implies that the runs of the program $P(s)$ are constant in time for every s. This implies that a program of constant activity is not vulnerable to timing attacks, which is not so surprising given the similarity between SPA and timing attacks.

4.4.1 Computed Proof of Constant Activity

To statically determine if the code is correctly balanced (i.e., that the activity of a given program is constant according to Def. 4.5), our tool relies on symbolic execution. The idea is to run the code of the program independently of the sensitive data. This is achieved by computing on sets of all the possible values instead of values directly. The symbolic execution terminates in our case because we are using the DPL protection on block ciphers, and we avoid combinatorial explosion thanks to bitslicing, as a value can initially be only 1 or 0 (or rather their DPL encoded

[6] See Tab. 4.1.

counterparts). Indeed, bitsliced code only use logical instructions as explained in Sec. 4.3.2, which will always return a result in $\{0, 1\}$ when given two values in $\{0, 1\}$ as arguments[7].

Our tool implements an interpreter for our generic assembly language which work with sets of values. The interpreter is equipped to measure all the possible Hamming distances of each value update, and all the possible Hamming weight of values. It watches updates in registers, in memory, and also in address buses (since the addresses may leak information when reading in look-up tables). If for one of these value updates there are different possible Hamming distances or Hamming weight, then we consider that there is a leak of information: the power consumption activity is not constant according to Def. 4.5.

Example 4.1. Let a be a register which can initially be either 0 or 1. Let b be a register which can initially be only 1. The execution of the instruction `orr a a b` will set the value of a to be all the possible results of $a \vee b$. In this example, the new set of possible values of a will be the singleton $\{1\}$ (since $0 \vee 1$ is 1 and $1 \vee 1$ is 1 too). The execution of this instruction only modified one value, that of a. However, the Hamming distance between the previous value of a and its new value can be either 0 (in case a was originally 1) or 1 (in case a was originally 0). Thus, we consider that there is a leak.

By running our interpreter on assembly code, we can statically determine if there are leakages or if the code is perfectly balanced. For instance for a block cipher, we initially set the key and the plaintext (i.e., the sensitive data) to have all their possible values: all the memory cells containing the bits of the key and of the plaintext have the value $\{0, 1\}$ (which denotes the set of two elements: 0 and 1). Then the interpreter runs the code and outputs all possible leakage; if none are present, it means that the code is well balanced. Otherwise we know which instructions caused the leak, which is helpful for debugging, and also to locate sensitive portions of the code.

For an example in which the code is balanced, we can refer to the execution of the `and` DPL macro shown in Tab. 4.4. There we can see that the Hamming distance of the updates does not depend on the values of a and b. We also note that at the end of the execution (and actually, all along the execution) the Hamming weight of each value does not depend on a and b either. This allows to chain macros safely: each value is precharged with 0 before being written to.

4.4.2 Hardware Characterization

The DPL countermeasure relies on the fact that the pair of bits used to store the DPL encoded values leak the same way, i.e., that their power consumptions are the same. This property is generally not true in non-specialized hardware. However, using the two closest bits (in terms of leakage) for the DPL protocol still helps reaching a better immunity to side-channel attacks, especially ASCAs that operate on a limited number of traces.

The idea is to compute the leakage level of each of the bits during the execution of the algorithm, in order to choose the two closest ones as the pair to use for the DPL protocol and thus ensure an optimal balance of the leakage. This is facilitated by the

[7] Or their DPL encoded counterparts, for instance $\{1, 2\}$ (i.e., $\{\texttt{01}, \texttt{10}\}$).

fact that the algorithm is bitsliced. Indeed, it allows to run the whole computation using only a chosen bit while all the others stay zero. We will see in Sec. 4.5.1 how we characterized our smartcard in practice.

4.5 Case Study: PRESENT on an ATmega163 AVR Micro-Controller

4.5.1 Profiling the ATmega163

We want to limit the size of the look-up tables used by the DPL macros. Thus, DPL macros need to be able to store two DPL encoded bits in the four consecutive bits of a register. This lets 13 possible DPL encoding layouts on 8-bit. Writing `X` for a bit that is used and `x` otherwise, we have:

1. `xxxxxxXX`,
2. `xxxxxXXx`,
3. `xxxxXXxx`,
4. `xxxXXxxx`,
5. `xxXXxxxx`,
6. `xXXxxxxx`,
7. `XXxxxxxx`,
8. `xxxxxXxX`,
9. `xxxxXxXx`,
10. `xxxXxXxx`,
11. `xxXxXxxx`,
12. `xXxXxxxx`,
13. `XxXxxxxx`.

As explained in Sec. 4.4.2, we want to use the pair of bits that have the closest leakage properties, and also which is the closest from the least significant bit, in order to limit the size of the look-up tables.

To profile the AVR chip (we are working with an *Atmel ATmega163 AVR* smartcard, which is *notoriously leaky*), we ran eight versions of an unprotected bitsliced implementation of PRESENT, each of them using only one of the 8 possible bits. We used the Normalized Inter-Class Variance (NICV) [BDGN14a], also called *coefficient of determination*, as a metric to evaluate the leakage level of the variables of each of the 8 versions. Let us denote by L the (noisy and non-injective) leakage associated with the manipulation of the sensitive value V, both seen as random variables; then the NICV is defined as the ratio between the inter-class and the total variance of the leakage, that is: $\text{NICV} = \frac{\mathsf{Var}[\mathbb{E}[L|V]]}{\mathsf{Var}[L]}$. By the Cauchy-Schwarz theorem, we have $0 \leqslant \text{NICV} \leqslant 1$; thus the NICV is an absolute leakage metric. A key advantage of NICV is that it detects leakage using public information like input plaintexts or output ciphertexts only. We used a fixed key and a variable plaintext on which applying NICV gave us the leakage level of all the intermediate variables in bijective relation with the plaintext (which are all the sensible data as seen in Def. 4.1). As we can see on the measures plotted in Fig. 4.5 (which can be found in App. 4.A), the least significant bit leaks very differently from the others, which are roughly

equivalent in terms of leakage[8]. Thus, we chose to use the xxxxxXXx DPL pattern to avoid the least significant bit (our goal here is not to use the optimal pair of bits but rather to demonstrate the added-value of the characterization).

4.5.2 Generating Balanced AVR Assembly

We wrote an AVR bitsliced implementation of PRESENT that uses the S-Box in 14 logic gates from Courtois et al. [CHM11]. This implementation was translated in our generic assembly language (see Sec. 4.3.1). The resulting code was balanced following the method discussed in Sec. 4.3, except that we used the DPL encoding layout adapted to our particular smartcard, as explained in Sec. 4.5.1. App. 4.B presents the code of the adapted DPL macro. The balance of the DPL code was then verified as in Sec. 4.4. Finally, the verified code was mapped back to AVR assembly. All the code transformations and the verification were done automatically using our tool.

4.5.3 Cost of the Countermeasure

The table in Tab. 4.5 compares the performances of the DPL protected implementation of PRESENT with the original bitsliced version from which the protected one has been derived. The DPL countermeasure multiplies by 1.88 the size of the compiled code. This low factor can be explained by the numerous instructions which it is not necessary to transform (the whole permutation layer of the PRESENT algorithm is left as is for instance). The protected version uses 64 more bytes of memory (sparsely, for the DPL macro look-up tables). It is also only 3 times slower[9], or 24 times if we consider that the original bitsliced but unprotected code could operate on 8 blocks at a time.

Table 4.5: DPL cost.

	bitslice	DPL	cost
code (B)	1620	3056	×1.88
RAM (B)	288	352	+64
#cycles	78,403	235,427	×3

Note that these experimental results are only valid for the PRESENT algorithm on the *Atmel ATmega163 AVR* device we used. Further work is necessary to compare these results to those which would be obtained with other algorithms such as Advanced Encryption Standard (AES), and on other platforms such as ARM processors.

4.5.4 Attacks

We attacked three implementations of the PRESENT algorithm: a bitsliced but unprotected one, a DPL one using the two less significant bits, and a DPL one using two bits that are more balanced in term of leakage (as explained in Sec. 4.5.1). On

[8]These differences are due to the internal architecture of the chip, for which we don't have the specifications.

[9]Notice that PRESENT is inherently slow in software (optimized non-bitsliced assembly is reported to run in about 11,000 clock cycles on an *Atmel ATtiny 45* device [EGG+12]) because it is designed for hardware. Typically, the permutation layer is free in hardware, but requires many bit-level manipulations in software. Nonetheless, we emphasize that there are contexts where PRESENT must be supported, but no hardware accelerator is available.

each of these, we computed the success rate of using monobit CPA of the output of the S-Box as a model. The monobit model is relevant because only one bit of sensitive data is manipulated at each cycle since the algorithm is bitsliced, and also because each register is precharged at 0 before a new intermediate value is written to it, as per the DPL protocol prescribe. Note that this means we consider the resistance against first-order attacks only. Actually, we are precisely in the context of [MOS11], where the efficiency of correlation and Bayesian attacks gets close as soon as the number of queries required to perform a successful attack is large enough. This justifies our choice of the CPA for the attack evaluation.

The results are shown in Fig. 4.9 (which can be found in App. 4.C.2). They demonstrate that the first DPL implementation is at least 10 times more resistant to first-order power analysis attacks (requiring almost $1,500$ traces) than the unprotected one. The second DPL implementation, which takes the chip characterization into account, is 34 *times more resistant* (requiring more than $4,800$ traces).

Interpreting these results requires to bear in mind that the *attacks setting was largely to the advantage of the attacker*. In fact, these results are very pessimistic: we used our knowledge of the key to select a narrow part of the traces where we knew that the attack would work, and we used the NICV [BDGN14a] to select the point where the SNR of the CPA attack is the highest (see similar use cases of NICV in [BDGN14b]). We did this so we could show the improvement in security due to the characterization of the hardware. Indeed, without this "cheating attacker" (for the lack of a better term), i.e., when we use a monobit CPA taking into account the maximum of correlation over the full round, as a normal attacker would do, the unprotected implementation breaks using about 400 traces (resp. 138 for the "cheating attacker"), while the poorly balanced one is still not broken using $100,000$ traces (resp. about $1,500$). We do not have more traces than that so we can only say that with an experimental SNR of 15 (which is quite large so far), the security gain is more than $250\times$ and may be much higher with the hardware characterization taken into account as our results with the "cheating attacker" shows.

As a comparison[10], an unprotected AES on the same smartcard breaks in 15 traces, and in 336 traces with a first order masking scheme using less powerful attack setting (see success rates of masking in App. 4.C.1), hence a security gain of $22\times$. Besides, we notice that our software DPL protection thwarts ASCAs. Indeed, ASCAs require a high signal to noise ratio on a single trace. This can happen both on unprotected and on masked implementation. However, our protection aims at theoretically cancelling the leakage, and practically manages to reduce it significantly, even when the chosen DPL bit pair is not optimal. Therefore, coupling software DPL with key-update [MSGR10] allows to both prevent against fast attacks on few traces (ASCAs) and against attacks that would require more traces (regular CPAs).

4.6 Conclusions and Perspectives

Contributions. We present a method to protect any bitsliced assembly code by transforming it to enforce the Dual-rail with Precharge Logic (DPL) protocol, which

[10] We insist that the comparison between two security gains is very platform-dependent. The figures we give are only valid on our specific setup. Of course, for different conditions, e.g., lower signal-to-noise ratio, masking might become more secure than DPL.

is a balancing countermeasure against power analysis. We provide a tool which automates this transformation. We also formally prove that this transformation is correct, i.e., that it preserves the semantic of the program.

Independently, we show how to formally prove that assembly code is well balanced. Our tool is also able to use this technique to statically determine whether some arbitrary assembly code's power consumption activity is constant, i.e., that it does not depend on the sensitive data. In this chapter we used the Hamming weight of values and the Hamming distance of values update as leakage models for power consumption, but our method is not tied to it and could work with any other leakage models that are computable. We present how to characterize the targeted hardware to make use of the resources which maximize the relevancy of our leakage model to run the DPL protocol.

We then applied our methods using our tool using an implementation of the PRESENT cipher on a real smartcard, which ensured that our methods and models are relevant in practice. In our case study, the provably balanced DPL protected implementation is at least 250 times more resistant to power analysis attacks than the unprotected version while being only 3 times slower. These figures could be better. Indeed, they do not take into account hardware characterization which helps the balancing a lot, as we were able to see with the "cheating attacker". Moreover, we have used the hardware characterization data grossly, only to show the added-value of the operation, which as expected is non-negligible. And of course interpreting our figures require to take into account that the *ATmega163*, the model of smartcard that we had at our disposal, is notoriously leaky.

These results show that software balancing countermeasures are realistic: our formally proved countermeasure is an order of magnitude less costly than the state of the art of formally proved masking [RP10].

Future work. The first and foremost future work surely is that our methods and tools need to be further tested in other experimental settings, across more hardware platforms, and using other cryptographic algorithms.

We did not try to optimize our PRESENT implementation (neither for speed nor space). However, automated proofs enable optimization: indeed, the security properties can be checked again after any optimization attempt (using proofs computation as non-regression tests, either for changes in the DPL transformation method, or for handcrafted optimizations of the generated DPL code).

Although the mapping from the internal assembly of our tool to the concrete assembly is straightforward, it would be better to have a formal correctness proof of the mapping.

Our work would also benefit from automated bitslicing, which would allow to automatically protect any assembly code with the DPL countermeasure. However, it is still a challenging issue.

Finally, the DPL countermeasure itself could be improved: the pair of bits used for the DPL protocol could change during the execution, or more simply it could be chosen at random for each execution in order to better balance the leakage among multiple traces. Besides, unused bits could be randomized instead of being zero in order to add noise on top of balancing, and thus reinforce the hypotheses we make on the hardware. An anonymous reviewer of the PROOFS 2014 workshop suggested that randomness could instead be used to mask the intermediate bits.

Indeed, the reviewer thinks that switching bus lines may only increase noise, while masking variables may provide sound resistance, at least at first order. The resulting method would therefore: 1. gain both the 1st-order resistance of masking countermeasures and the significant flexibility of software-defined countermeasures; 2. still benefit from the increase of resistance resorting to the use of the DPL technique, as demonstrated by present chapter. This suggestion is of course only intuitive and lacks argumentation based on precise analysis and calculation.

We believe formal methods have a bright future concerning the certification of side-channel attacks countermeasures (including their implementation in assembly) for trustable cryptosystems.

4.A Characterization of the Atmel ATmega163 AVR Micro-Controller

Fig. 4.5 shows the leakage level computed using NICV [BDGN14a] for each bit of the *Atmel ATmega163 AVR* smartcard that we used for our tests (see Sec. 4.5.1). We can see the first bit leaks very differently from the others. Thus it is not a good candidate to appear in the bit pair used for the DPL protocol.

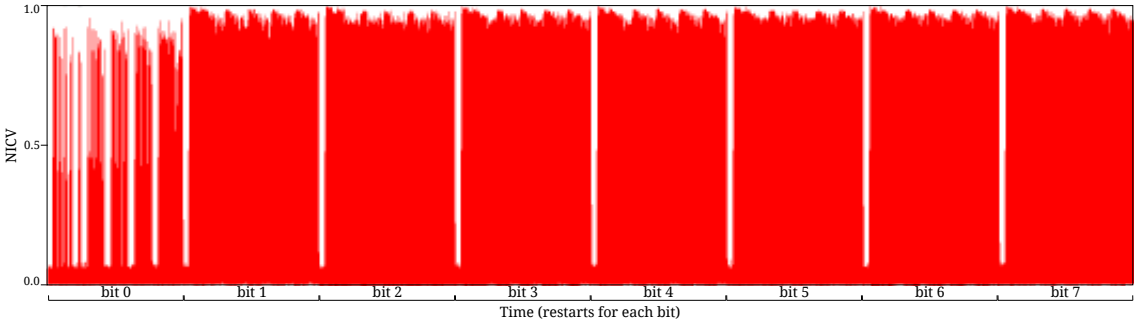


Figure 4.5: Leakage during unprotected encryption for each bit on *ATmega163*.

4.B DPL Macro for the AVR Micro-Controller

Once we profiled our smartcard as described in Sec. 4.5.1, we decided to use the bits 1 and 2 for the DPL protocol (`xxxxxXXx`), that is, the DPL value of 1 becomes 2 and the DPL value of 0 becomes 4. To avoid using the least significant bit (which leaks very differently from the others), we decided to align the two DPL bits for look-up table access starting on the bit 1 rather than 0 (`xxxXXXXx`). With these settings, the DPL macro automatically generated by paioli is presented in Fig. 4.6 (it follows the same conventions as Fig. 4.2). As we can see the only modification is the mask applied in the logical *and* instructions which is now 6 instead of 3 to reflect the new DPL pattern.

Note that the least significant bit is now unused by the DPL protocol and allowed paioli to compact the look-up tables used by the DPL macros. Indeed, their addresses need to be of the form `/.+0000./` leaving the least significant bit free and thus allowing to interleave two look-up tables one on another without overlapping of their actually used cells (see Sec. 4.3.2).

$$
\begin{array}{rcl}
r_1 & \leftarrow & r_0 \\
r_1 & \leftarrow & a \\
r_1 & \leftarrow & r_1 \wedge 6 \\
r_1 & \leftarrow & r_1 \ll 1 \\
r_1 & \leftarrow & r_1 \ll 1 \\
r_2 & \leftarrow & r_0 \\
r_2 & \leftarrow & b \\
r_2 & \leftarrow & r_2 \wedge 6 \\
r_1 & \leftarrow & r_1 \vee r_2 \\
r_3 & \leftarrow & r_0 \\
r_3 & \leftarrow & op[r_1] \\
d & \leftarrow & r_0 \\
d & \leftarrow & r_3
\end{array}
$$

Figure 4.6: DPL macro for $d = a$ op b on the *ATmega163*.

4.C Attacks

4.C.1 Attack results on masking (AES)

For the sake of comparison, we provide attack results on the same smartcard tested with the same setup. Figure 4.7 shows the success rate for the attack on the first

byte of an AES.

We estimate the number of traces for a successful attack as the abscissa where the success rate curve first intersects the 80% horizontal line.

(a) Univariate CPA attack on unprotected AES.

(b) Bi-variate 2O-CPA on first-order protected AES.

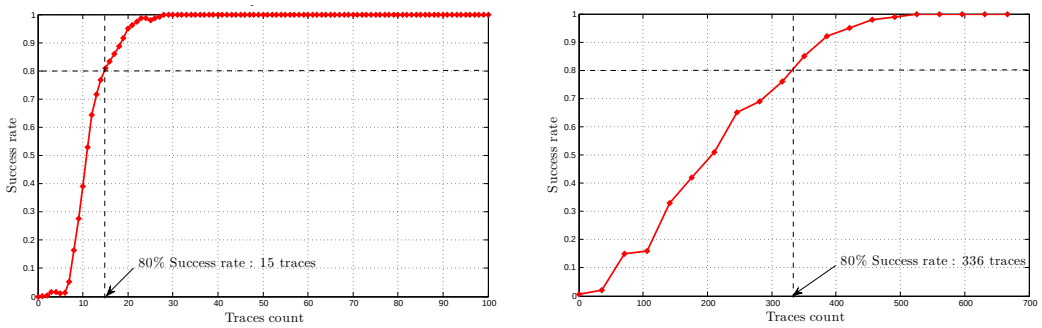


Figure 4.7: Attacking AES on the *ATmega163*: success rates.

4.C.2 Attack results on DPL (PRESENT)

Fig. 4.9 shows the success rates and the correlation curves when attacking our three implementations of PRESENT. The sensitive variable we consider is in line with the choice of Kocher et al. in their CRYPTO'99 paper [KJJ99]: it is the least significant bit of the output of the substitution boxes (that are 4×4 in PRESENT).

In Fig. 4.8, we give, for the unprotected bitslice implementation, the correspondence between the operations of PRESENT and the NICV trace. The zones of largest NICV correspond to operations that access (read or write) sensitive data in RAM. To make the attacks more powerful, they are not done on the maximal correlation point over the full first round of PRESENT[11] ($500,000$ samples), but rather on a smaller interval (of only 140 samples, i.e., one clock period of the device) of high potential leakage revealed by the NICV computations, namely sBoxLayer.

This makes the attack much more powerful and has to be taken into account when interpreting its results. In fact, the results we present are very pessimistic: we used our knowledge of the key to select a narrow part of the traces where we knew that the attack would work, and we used the NICV [BDGN14a] to select the point where the SNR of the CPA attack is the highest. We did this so we could show the improvement in security due to the characterization of the hardware. Indeed, without this "cheating attacker" (for the lack of a better term), i.e., when we use a monobit CPA taking into account the maximum of correlation over the full round, as a normal attacker would do, the unprotected implementation breaks using about 400 traces (resp. 138 for the "cheating attacker"), while the poorly balanced one is still not broken using $100,000$ traces (resp. about $1,500$). We do not have more traces than that so we can only say that with an experimental SNR of 15 (which is quite large so far), the security gain is more than 250× and may be

[11] Note that using the maximum correlation point to attack the DPL implementations resulted in the success rate remaining always at $\approx 1/16$ (there are 2^4 key guesses in PRESENT when targeting the first round, because the substitution boxes are 4×4) in average (at least on the number of traces we had ($100,000$)) on both on them.

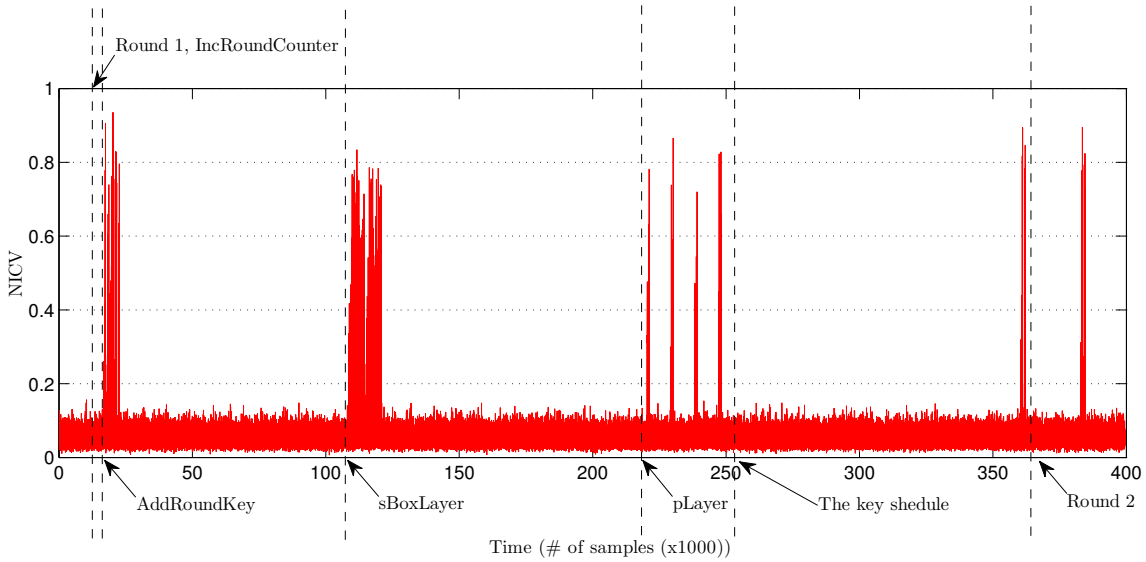
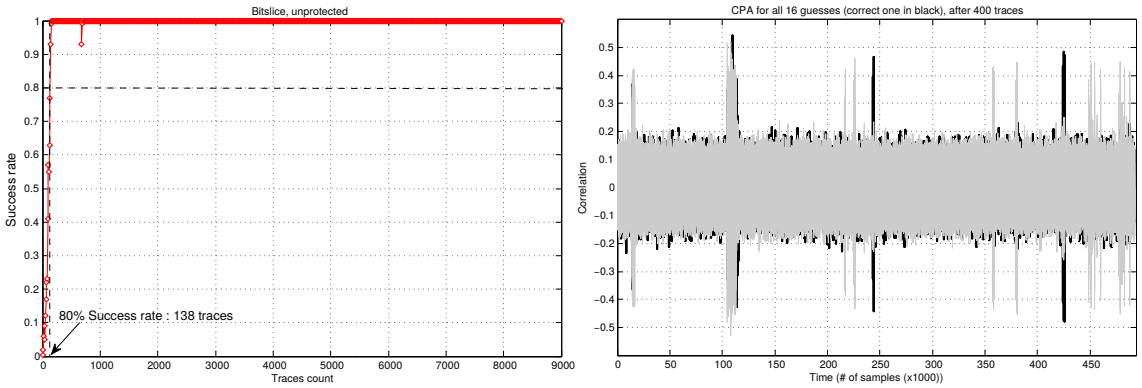


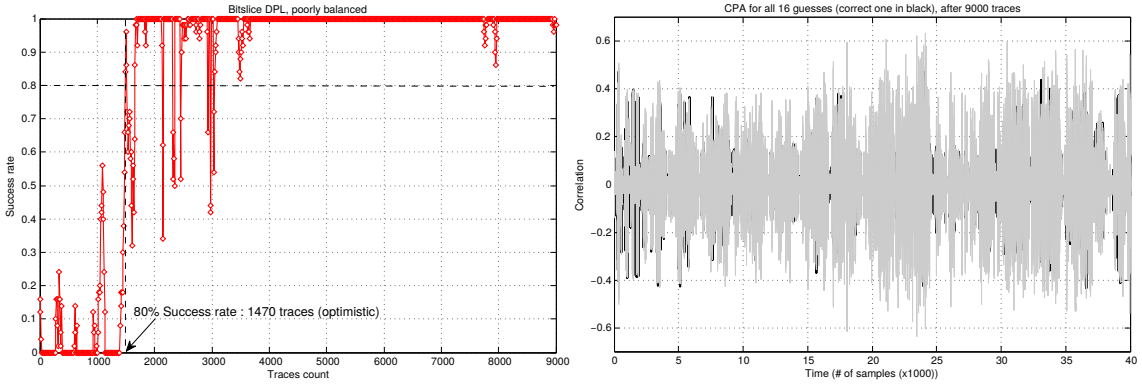
Figure 4.8: Correspondence between NICV and the instructions of PRESENT.

much higher with the hardware characterization taken into account as our results with the "cheating attacker" shows. Another way of understanding the 250-fold data complexity increase for the CPA is to turn this figure into a reduction of the SNR: according to [TPR13, BDGN14b], our DPL countermeasure has attenuated the SNR by a factor of at least $\sqrt{250} \approx 16$.

(a) Monobit CPA attack on unprotected bitslice implementation.



(b) Monobit CPA attack on poorly balanced DPL implementation (bits 0 and 1).



(c) Monobit CPA attack on better balanced DPL implementation (bits 1 and 2).

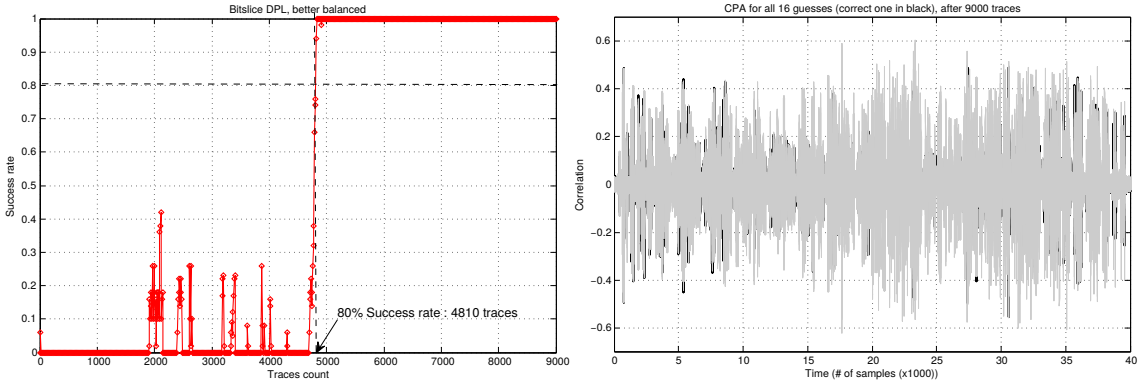


Figure 4.9: Attacks on our three implementations of PRESENT;
Left: success rates (estimated with 100 attacks/step), and
Right: CPA curves (whole first round in (a), and only sBoxLayer for (b) and (c)).

0x5

Formal Proof of CRT-RSA Countermeasures Against Fault Attacks

This chapter presents a joint work with Sylvain Guilley as my adviser. A first version of this work was accepted at the PROOFS 2013 workshop, and an extended version appeared in the *Journal of Cryptographic Engineering*.

Abstract. In this chapter, we describe a methodology that aims at either breaking or proving the security of CRT-RSA implementations against fault injection attacks. In the specific case-study of the BellCoRe attack, our work bridges a gap between formal proofs and implementation-level attacks. We apply our results to three implementations of CRT-RSA, namely the unprotected one, that of Shamir, and that of Aumüller et al. Our findings are that many attacks are possible on both the unprotected and the Shamir implementations, while the implementation of Aumüller et al. is resistant to all single-fault attacks. It is also resistant to double-fault attacks if we consider the less powerful threat-model of its authors.

Contents

5.1 Introduction

It is known since 1997 that injecting faults during the computation of CRT-RSA could yield to malformed signatures that expose the prime factors (p and q) of the public modulus ($N = p \cdot q$). Notwithstanding, computing without the fourfold acceleration conveyed by the *Chinese Remainder Theorem* (CRT) is definitely not an option in practical applications. Therefore, many countermeasures have appeared that consist in step-wise internal checks during the CRT computation. To our best knowledge, none of these countermeasures have been proven formally. Thus without surprise, some of them have been broken, and then patched. The current state-of-the-art in computing CRT-Rivest–Shamir–Adleman (RSA) without exposing p and q relies thus on algorithms that have been carefully scrutinized by cryptographers. Nonetheless, neither the hypotheses of the fault attack nor the security itself have been unambiguously modeled.

This is the purpose of this chapter. The difficulties are *a priori* multiple: in fault injection attacks, the attacker has an extremely high power because he can fault any variable. Traditional approaches thus seem to fall short in handling this problem. Indeed, there are two canonical methodologies: *formal* and *computational* proofs. Formal proofs (e.g., in the so-called Dolev-Yao model) do not capture the requirement for faults to preserve some information about one of the two moduli; indeed, it considers the RSA as a black-box with a key pair. Computational proofs are way too complicated (in terms of computational complexity) since the handled numbers are typically $2,048$ bit long.

The state-of-the-art contains one reference related to the formal proof of a CRT-RSA implementation: it is the work of Christofi, Chetali, Goubin and Vigilant [CCGV13]. For tractability purposes, the proof is conducted on reduced versions of the algorithms parameters. One fault model is chosen authoritatively (the zeroization of a complete intermediate data), which is a strong assumption. In addition, the verification is conducted on a pseudo-code, hence concerns about its portability after its compilation into machine-level code. Another reference related to formal proofs and fault injection attacks is the work of Guo, Mukhopadhyay, and Karri. In [GMK12], they explicit an Advanced Encryption Standard (AES) implementation that is provably protected against differential fault analyses [BS97]. The approach is purely combinational, because the faults propagation in AES concerns 32-bit words called columns; consequently, all fatal faults (and thus all innocuous faults) can be enumerated.

Contributions. Our contribution is to reach a full fault coverage of the CRT-RSA algorithm, thereby keeping the proof valid even if the code is transformed (e.g., compiled or partitioned in software/hardware). To this end we developed a tool called finja[1] based on symbolic computation in the framework of modular arithmetic, which enables formal analysis of CRT-RSA and its countermeasures against fault injection attacks (See App. B.2). We apply our methods on three implementations: the unprotected one, the one protected by Shamir's countermeasure, and the one protected by Aumüller et al.'s countermeasure. We find many possible fault injections that enable a BellCoRe attack on the unprotected implementation of the CRT-RSA computation, as well as on the one protected by Shamir's countermea-

[1] http://pablo.rauzy.name/sensi/finja.html

sure. We formally prove the security of the Aumüller et al.'s countermeasure against the BellCoRe attack, under a fault model that considers *permanent faults* (in memory) and *transient faults* (one-time faults, even on copies of the secret key parts), with or without forcing at zero, and with possibly faults at various locations.

Organization of the chapter. We recall the CRT-RSA cryptosystem and the BellCoRe attack in Sec. 5.2; still from an historical perspective, we explain how the CRT-RSA implementation has been amended to withstand more or less efficiently the BellCoRe attack. Then, in Sec. 5.3, we define our approach. Sec. 5.4, Sec. 5.5, and Sec. 5.6 are case studies using the methods developed in Sec. 5.3 of respectively an unprotected version of the CRT-RSA computation, a version protected by Shamir's countermeasure, and a version protected by Aumüller et al.'s countermeasure. Conclusions and perspectives are in Sec. 5.7.

5.2 CRT-RSA and the BellCoRe Attack

This section summarizes known results about fault attacks on CRT-RSA (see also [Koç94], [TW12, Chap. 3] and [JT11, Chap. 7 & 8]). Its purpose is to settle the notions and the associated notations that will be used in the later sections, to present our novel contributions.

5.2.1 RSA

RSA is both an *encryption* and a *signature* scheme. It relies on the fact that for any message $0 \leq M < N$, $(M^d)^e \equiv M \mod N$, where $d \equiv e^{-1} \mod \varphi(N)$, by Euler's theorem[2]. In this equation, φ is Euler's totient function, equal to $\varphi(N) = (p-1)\cdot(q-1)$ when $N = p \cdot q$ is a composite number, product of two primes p and q. For example, if Alice generates the signature $S = M^d \mod N$, then Bob can verify it by computing $S^e \mod N$, which must be equal to M unless Alice is only pretending to know d. Therefore (N, d) is called the private key, and (N, e) the public key. In this chapter, we are not concerned about the key generation step of RSA, and simply assume that d is an unknown number in $[\![1, \varphi(N) = (p-1)\cdot(q-1)[\![$. Actually, d can also be chosen to be equal to the smallest value $e^{-1} \mod \lambda(N)$, where $\lambda(N) = \frac{(p-1)\cdot(q-1)}{\gcd(p-1,q-1)}$ is the Carmichael function (see PKCS #1 v2.1, §3.1).

5.2.2 CRT-RSA

The computation of $M^d \mod N$ can be speeded-up by a factor of four using the Chinese Remainder Theorem (CRT). Indeed, numbers modulo p and q are twice as short as those modulo N. For example, for $2,048$ bits RSA, p and q are $1,024$ bits long. CRT-RSA consists in computing $S_p = M^d \mod p$ and $S_q = M^d \mod q$, which can be recombined into S with a limited overhead. Due to the little Fermat theorem (the special case of the Euler theorem when the modulus is a prime), $S_p = (M \mod p)^{d \mod (p-1)} \mod p$. This means that in the computation of S_p, the processed data have $1,024$ bits, and the exponent itself has $1,024$ bits (instead

[2] We use the usual convention in all mathematical equations, namely that the "mod" operator has the lowest binding precedence, i.e., $a \times b \mod c \times d$ represents the element $a \times b$ in $\mathbb{Z}_{c\times d}$.

of 2,048 bits). Thus the multiplication is four times faster and the exponentiation eight times faster. However, as there are two such exponentiations (modulo p and q), the overall CRT-RSA is roughly speaking four times faster than RSA computed modulo N.

This acceleration justifies that CRT-RSA is always used if the factorization of N as $p \cdot q$ is known. In CRT-RSA, the private key has a richer structure than simply (N, d): it is actually the 5-tuple (p, q, d_p, d_q, i_q), where:

- $d_p \doteq d \mod (p-1)$,
- $d_q \doteq d \mod (q-1)$, and
- $i_q \doteq q^{-1} \mod p$.

The CRT-RSA algorithm is presented in Alg. 5.1. It is straightforward to check that the signature computed at line 3 belongs to $[\![0, p \cdot q - 1]\!]$. Consequently, no reduction modulo N is necessary before returning S.

Algorithm 5.1: Unprotected CRT-RSA

Input : Message M, key (p, q, d_p, d_q, i_q)
Output: Signature $M^d \mod N$

1 $S_p = M^{d_p} \mod p$ /* Intermediate signature in $\mathbb{Z}_p$ */
2 $S_q = M^{d_q} \mod q$ /* Intermediate signature in $\mathbb{Z}_q$ */
3 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ /* Recombination in $\mathbb{Z}_N$ (Garner's method [Gar65]) */
4 **return** S

5.2.3 The BellCoRe Attack

In 1997, a dreadful remark has been made by Boneh, DeMillo and Lipton [BDL97], three staffs of Bell Communication Research: Alg. 5.1 could reveal the secret primes p and q if the line 1 or 2 of the computation is faulted, even in a very random way. The attack can be expressed as the following proposition.

Proposition 5.1 (BellCoRe attack)**.** *If the intermediate variable S_p (resp. S_q) is returned faulted as $\widehat{S_p}$ (resp. $\widehat{S_q}$)[3], then the attacker gets an erroneous signature $\widehat{S}$, and is able to recover q (resp. p) as* $\gcd(N, S - \widehat{S})$.

Proof. For any integer x, $\gcd(N, x)$ can only take 4 values:

- 1, if N and x are coprime,
- p, if x is a multiple of p,
- q, if x is a multiple of q,
- N, if x is a multiple of both p and q, i.e., of N.

In Alg. 5.1, if S_p is faulted (i.e., replaced by $\widehat{S_p} \neq S_p$), then $S - \widehat{S} = q \cdot ((i_q \cdot (S_p - S_q) \mod p) - (i_q \cdot (\widehat{S_p} - S_q) \mod p))$, and thus $\gcd(N, S - \widehat{S}) = q$. If S_q is faulted (i.e., replaced by $\widehat{S_q} \neq S_q$), then $S - \widehat{S} \equiv (S_q - \widehat{S_q}) - (q \mod p) \cdot i_q \cdot (S_q - \widehat{S_q}) \equiv 0 \mod p$ because $(q \mod p) \cdot i_q \equiv 1 \mod p$, and thus $S - \widehat{S}$ is a multiple of p. Additionally, $S - \widehat{S}$ is not a multiple of q. So, $\gcd(N, S - \widehat{S}) = p$.

[3] In other papers, the faulted variables (such as X) are written either as X^* or $\tilde{X}$; in this chapter, we use a hat which can stretch to cover the adequate portion of the expression, as it allows to make an unambiguous difference between $\widehat{X^e}$ and $\widehat{X}^e$.

In both cases, the greatest common divisor could yield N. However, $(S - \widehat{S})/q$ in the first case (resp. $(S - \widehat{S})/p$ in the second case) is very unlikely to be a multiple of p (resp. q). Indeed, if the random fault is uniformly distributed, the probability that $\gcd(N, S - \widehat{S})$ is equal to p (resp. q) is negligible[4]. □

This version of the BellCoRe attack requires that two identical messages with the same key can be signed; indeed, one signature that yields the genuine S and an other that is perturbed and thus returns $\widehat{S}$ are needed. Little later, the BellCoRe attack has been improved by Joye, Lenstra and Quisquater [JLQ99]. This time, the attacker can recover p or q with one only faulty signature, provided the input m of RSA is known.

Proposition 5.2 (One faulty signature BellCoRe attack)**.** *If the intermediate variable S_p (resp. S_q) is returned faulted as $\widehat{S_p}$ (resp. $\widehat{S_q}$), then the attacker gets an erroneous signature $\widehat{S}$, and is able to recover p (resp. q) as* $\gcd(N, M - \widehat{S}^e)$ *(*with an overwhelming probability*).*

Proof. By proposition 5.1, if a fault occurs during the computation of S_p, then $\gcd(N, S - \widehat{S}) = q$ (*most likely*). This means that:

- $S \not\equiv \widehat{S} \mod p$, and thus $S^e \not\equiv \widehat{S}^e \mod p$ (*indeed, if the congruence was true, we would have $e|p-1$, which is very unlikely*);
- $S \equiv \widehat{S} \mod q$, and thus $S^e \equiv \widehat{S}^e \mod q$;

As $S^e \equiv M \mod N$, this proves the result. A symmetrical reasoning can be done if the fault occurs during the computation of S_q. □

5.2.4 Protection of CRT-RSA Against the BellCoRe Attack

Many protections against the BellCoRe attack have been proposed. A few of them are listed below, and then the most salient features of these countermeasures are described:

- Obvious countermeasures: no CRT optimization, or with signature verification;
- Shamir [Sha99];
- Aumüller et al. [ABF+02];
- Vigilant, original [Vig08a] and with some fixes by Coron et al. [CGM+10];
- Kim et al. [KKHH11].

Obvious Countermeasures

Fault attacks on RSA can be thwarted simply by refraining from implementing the CRT.

If this is not affordable, then the signature can be verified before being outputted. If $S = M^d \mod N$ is the signature, this straightforward countermeasure consists in testing $S^e \stackrel{?}{\equiv} M \mod N$. Such protection is efficient in practice, but is criticized for three reasons. First of all, it requires an access to e, which is not always present in the secret key structure, as in the 5-tuple example given in Sec. 5.2.2. Nonetheless, we attract the author's attention on paper [Joy09] for a clever embedding of e

[4] If it nonetheless happens that $\gcd(N, S - \widehat{S}) = N$, then the attacker can simply retry another fault injection, for which the probability that $\gcd(N, S - \widehat{S}) \in \{p, q\}$ increases.

into [the representation of] d. Second, the performances are incurred by the extra exponentiation needed for the verification. In some applications, the public exponent can be chosen small (for instance e can be equal to a number such as 3, 17 or 65537), and then d is computed as $e^{-1} \mod \lambda(N)$ using the extended Euclidean algorithm or better alternatives [JP03]. But in general, e is a number possibly as large as d (both are as large as N), thus the obvious countermeasure doubles the computation time (which is really non-negligible, despite the CRT fourfold acceleration). Third, this protection is not immune to a fault injection that would target the comparison. Overall, this explains why other countermeasures have been devised.

Shamir

The CRT-RSA algorithm of Shamir builds on top of the CRT and introduces, in addition to the two primes p and q, a third factor r. This factor r is random[5] and small (less than 64 bit long), and thus coprime with p and q. The computations are carried out modulo $p' = p \cdot r$ (resp. modulo $q' = q \cdot r$), which allows for a retrieval of the intended results by reducing them modulo p (resp. modulo q), and for a verification by a reduction modulo r. Alg. 5.2 describes one version of Shamir's countermeasure. This algorithm is aware of possible fault injections, and thus can raise an *exception* if an incoherence is detected. In this case, the output is not the (purported faulted) signature, but a specific message "error".

Algorithm 5.2: Shamir CRT-RSA

Input : Message M, key (p, q, d, i_q),
32-bit random prime r
Output: Signature $M^d \mod N$,
or error if some fault injection has been detected.

1 $p' = p \cdot r$
2 $d_p = d \mod (p-1) \cdot (r-1)$
3 $S'_p = M^{d_p} \mod p'$ `// Signature modulo` p'
4 $q' = q \cdot r$
5 $d_q = d \mod (q-1) \cdot (r-1)$
6 $S'_q = M^{d_q} \mod q'$ `// Signature modulo` q'
7 $S_p = S'_p \mod p$
8 $S_q = S'_q \mod q$
`// Same as in line 3 of Alg. 5.1`
9 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$
10 **if** $S'_p \not\equiv S'_q \mod r$ **then**
11 | **return** error
12 **else**
13 | **return** S
14 **end**

[5]The authors notice that in Shamir's countermeasure, r is *a priori* not a secret, hence can be static and safely divulged.

Aumüller

The CRT-RSA algorithm of Aumüller et al. is a variation of that of Shamir, that is primarily intended to fix two shortcomings. First it removes the need for d in the signature process, and second, it also checks the recombination step. The countermeasure, given in Alg. 5.3, introduces, in addition to p and q, a third prime t. The computations are done modulo $p' = p \cdot t$ (resp. modulo $q' = q \cdot t$), which allows for a retrieval of the intended results by reducing them modulo p (resp. modulo q), and for a verification by a reduction modulo t. However, the verification is more subtle than for the case of Shamir. In Shamir's CRT-RSA (Alg. 5.2), the verification is *symmetrical*, in that the computations modulo $p \cdot r$ and $q \cdot r$ operate on the same object, namely m^d. In Aumüller et al.'s CRT-RSA (Alg. 5.3), the verification is *asymmetrical*, since the computations modulo $p \cdot t$ and $q \cdot t$ operate on two different objects, namely $M^{d_p \mod (t-1)}$ and $M^{d_q \mod (t-1)}$. The verification consists in an identity that resembles that of ElGamal for instance: is $(M^{d_p \mod (t-1)})^{d_q \mod (t-1)}$ equivalent to $(M^{d_q \mod (t-1)})^{d_p \mod (t-1)}$ modulo t? Specifically, if we note S'_p the signature modulo p', then $S_p = S \mod p$ is equal to $S'_p \mod p$. Furthermore, let us denote

- $S_{pt} = S'_p \mod t$,
- $S_{qt} = S'_q \mod t$,
- $d_{pt} = d_p \mod (t-1)$, and
- $d_{qt} = d_q \mod (t-1)$.

It can be verified that those figures satisfy the identity: $S_{pt}^{d_{qt}} \equiv S_{qt}^{d_{pt}} \mod t$, because both terms are equal to $M^{d_{pt} \cdot d_{qt}} \mod t$. The prime t is referred to as a security parameter, as the probability to pass the test (at line 23 of Alg. 5.3) is equal to $1/t$ (i.e., about 2^{-32}), assuming a uniform distribution of the faults. Indeed, this is the probability to find a large number that, once reduced modulo t, matches a predefined value.

Alg. 5.3 does some verifications during the computations, and reports an error in case a fault injection can cause a malformed signature susceptible of unveiling p and q. More precisely, an error is returned in either of these seven cases:

1. p' is not a multiple of p (*because this would amount to faulting p in the unprotected algorithm*)
2. $d'_p = d_p + \mathsf{random}_1 \cdot (p-1)$ is not equal to $d_p \mod (p-1)$ (*because this would amount to faulting d_p in the unprotected algorithm*)
3. q' is not a multiple of q (*because this would amount to faulting q in the unprotected algorithm*)
4. $d'_q = d_q + \mathsf{random}_2 \cdot (q-1)$ is not equal to $d_q \mod (q-1)$ (*because this would amount to faulting d_q in the unprotected algorithm*)
5. $S - S'_p \mod p$ is nonzero (*because this would amount to faulting the recombination modulo p in the unprotected algorithm*)
6. $S - S'_q \mod q$ is nonzero (*because this would amount to faulting the recombination modulo q in the unprotected algorithm*)
7. $S_{pt}^{d_q} \mod t$ is not equal to $S_{qt}^{d_p} \mod t$ (*this checks simultaneously for the integrity of S'_p and S'_q*)

Notice that the last verification could not have been done on the unprotected algorithm, it constitutes the added value of Aumüller et al.'s algorithm. These seven cases are *informally* assumed to protect the algorithm against the BellCoRe attack.

Algorithm 5.3: Aumüller CRT-RSA

Input : Message M, key (p, q, d_p, d_q, i_q),
32-bit random prime t
Output: Signature $M^d \mod N$,
or error if some fault injection has been detected.

1 $p' = p \cdot t$
2 $d'_p = d_p + \mathsf{random}_1 \cdot (p-1)$ `// Against SPA, not fault attacks`
3 $S'_p = M^{d'_p} \mod p'$ `// Signature modulo` p'
4 **if** $(p' \mod p \neq 0)$ **or** $(d'_p \not\equiv d_p \mod (p-1))$ **then**
5 | **return** error
6 **end**
7 $q' = q \cdot t$
8 $d'_q = d_q + \mathsf{random}_2 \cdot (q-1)$ `// Against SPA, not fault attacks`
9 $S'_q = M^{d'_q} \mod q'$ `// Signature modulo` q'
10 **if** $(q' \mod q \neq 0)$ **or** $(d'_q \not\equiv d_q \mod (q-1))$ **then**
11 | **return** error
12 **end**
13 $S_p = S'_p \mod p$
14 $S_q = S'_q \mod q$
15 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ `// Same as in line 3 of Alg. 5.1`
16 **if** $(S - S'_p \not\equiv 0 \mod p)$ **or** $(S - S'_q \not\equiv 0 \mod q)$ **then**
17 | **return** error
18 **end**
19 $S_{pt} = S'_p \mod t$
20 $S_{qt} = S'_q \mod t$
21 $d_{pt} = d'_p \mod (t-1)$
22 $d_{qt} = d'_q \mod (t-1)$
23 **if** $S_{pt}^{d_{qt}} \not\equiv S_{qt}^{d_{pt}} \mod t$ **then**
24 | **return** error
25 **else**
26 | **return** S
27 **end**

The criteria for fault detection is not to detect all faults; for instance, a fault on the final return of S (line 26) is not detected. However, of course, such a fault is not exploitable by a BellCoRe attack.

Remark 5.1. Some parts of the Aumüller algorithm are actually not intended to protect against fault injection attacks, but against side-channel analysis, such as the Simple Power Analysis (SPA). This is the case of lines 2 and 8 in Alg. 5.3. These SPA attacks consist in monitoring via a side-channel the activity of the chip, in a view to extract the secret exponent, using *generic* methods described in [KJJ99] or more *accurate* techniques such as wavelet transforms [SED+11, DSE+12]. They can be removed if a minimalist protection against only fault injection attacks is looked for; but as they do not introduce weaknesses (in this very specific case), they are simply kept as such.

Vigilant

The CRT-RSA algorithm of Vigilant [Vig08a] also considers computations in a larger ring than $\mathbb{Z}_p$ (abbreviation for $\mathbb{Z}/p\mathbb{Z}$) and $\mathbb{Z}_q$, to enable verifications. In this case, a small random number r is cast, and computations are carried out in $\mathbb{Z}_{p\cdot r^2}$ and $\mathbb{Z}_{q\cdot r^2}$. In addition, the computations are now conducted not on the plain message M, but on an encoded message M', built using the CRT as the solution of those two requirements:

i: $M' \equiv M \mod N$, and
ii: $M' \equiv 1 + r \mod r^2$.

This system of equations has a single solution modulo $N \times r^2$, because N and r^2 are coprime. Such a representation allows to conduct in parallel the functional CRT-RSA (line *i*) and a verification (line *ii*). The verification is elegant, as it leverages this remarkable equality: $(1+r)^{d_p} = \sum_{i=0}^{d_p} \binom{d_p}{i} \cdot r^i \equiv 1 + d_p \cdot r \mod r^2$. Thus, as opposed to Aumüller et al.'s CRT-RSA, which requires one exponentiation (line 23 of Alg. 5.3), the verification of Vigilant's algorithm adds only one affine computation (namely $1 + d_p \mod r^2$).

The original description of Vigilant's algorithm involves some trivial computations on p and q, such as $p-1$, $q-1$ and $p \times q$. Those can be faulted, in such a way the BellCoRe attack becomes possible despite all the tests. Thus, a patch by Coron et al. has been released in [CGM+10] to avoid the reuse of $\widehat{p-1}$, $\widehat{q-1}$ and $\widehat{p \times q}$ in the algorithm.

Kim

Kim, Kim, Han and Hong propose in [KKHH11] a CRT-RSA algorithm that is based on a collaboration between a customized modular exponentiation and verifications at the recombination level based on Boolean operations. The underlying protection concepts being radically different from the algorithms of Shamir, Aumüller and Vigilant, we choose not to detail this interesting countermeasure.

Other Miscellaneous Fault Injections Attacks

When the attacker has the power to focus its fault injections on *specific bits* of *sensitive resources*, then more challenging security issues arise [BCDG12]. These

threats require a highly qualified expertise level, and are thus considered out of the scope of this chapter.

Besides, for completeness, we mention that other fault injections mitigating techniques have been promoted, such as the *infective computation scheme* (refer to the seminal paper [BOS03]). This family of protections, although interesting, is neither covered by this chapter.

In this chapter, we will focus on three implementations, namely the unprotected one (Sec. 5.4), the one protected by Shamir's countermeasure (Sec. 5.5), and the one protected by Aumüller et al.'s countermeasure (Sec. 5.6).

5.3 Formal Methods

For all the countermeasures presented in the previous section (Sec. 5.2), we can see that no formal proof of resistance against attacks is claimed. Informal arguments are given, that convince that for some attack scenarii, the attack attempts are detected hence harmless. Also, an analysis of the probability that an attack succeeds by chance (with a low probability of $1/t$) is carried out, however, this analysis strongly relies on assumptions on the faults distribution. Last but not least, the algorithms include protections against both passive side-channel attacks (typically SPA) and active side-channel attacks, which makes it difficult to analyze for instance the minimal code to be added for the countermeasure to be correct.

5.3.1 CRT-RSA and Fault Injections

Our goal is to prove that a given countermeasure works, i.e., that it delivers a result which does leak information about neither p nor q (when the implementation is subject to fault injections) exploitable in a BellCoRe attack. In addition, we wish to reach this goal with the two following assumptions:

- our proof applies to a very general attacker model, and
- our proof applies to any implementation that is a (strict) refinement of the abstract algorithm.

First, we must define what computation is done, and what is our threat model.

Definition 5.1 (CRT-RSA)**.** The CRT-RSA computation takes as input a message M, assumed known by the attacker but which we consider to be random, and a secret key (p, q, d_p, d_q, i_q). Then, the implementation is free to instantiate any variable, but must return a result equal to $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$, where:

- $S_p = M^{d_p} \mod p$, and
- $S_q = M^{d_q} \mod q$.

Definition 5.2 (Fault injection)**.** An attacker is able to request RSA computations, as per Def. 5.1. During the computation, the attacker can modify any intermediate value by setting it to either a *random value* or *zero*. At the end of the computation the attacker can read the result.

Of course, the attacker cannot read the intermediate values used during the computation, since the secret key and potentially the modulus factors are used. Such "whitebox" attack would be too powerful; nonetheless, it is very hard in practice

for an attacker to be able to access intermediate variables, due to specific protections (e.g., blinding) and noise in the side-channel leakage (e.g., power consumption, Electromagnetic (EM) emanation). Remark that our model only takes into account fault injection on data; the control flow is supposed not to be mutable.

Remark 5.2. We notice that the fault injection model of Def. 6.2 corresponds to that of Vigilant [Vig08a], with the exception that the conditional tests can also be faulted. To summarize, an attacker can modify a value in the global memory (*permanent fault*), and modify a value in a local register or bus (*transient fault*), but cannot inject a permanent fault in the input data (message and secret key), nor modify the control flow graph.

The independence of the proofs on the algorithm implementation demands that the algorithm is described at a high level. The two properties that characterize the relevant level are as follows:

1. The description should be low level enough for the attack to work if protections are not implemented.
2. Any additional intermediate variable that would appear during refinement could be the target of an attack, but such a fault would propagate to an intermediate variable of the high level description, thereby having the same effect.

From those requirements, we deduce that:

1. The RSA description must exhibit the computation modulo p and q and the CRT recombination; typically, a completely blackbox description, where the computations would be realized in one go without intermediate variables, is not conceivable.
2. However, it can remain abstract, especially for the computational parts. For instance a fault in the implementation of the multiplication (or the exponentiation) is either inoffensive, and we do not need to care about it, or it affects the result of the multiplication (or the exponentiation), and our model takes it into account without going into the details of how the multiplication (or exponentiation) is computed.

In our approach, the protections must thus be considered as an augmentation of the unprotected code, i.e., a derived version of the code where additional variables are used. The possibility of an attack on the unprotected code attests that the algorithm is described at the adequate level, while the impossibility of an attack (to be proven) on the protected code shows that added protections are useful in terms of resistance to attacks.

5.3.2 finja

Several tools are *a priori* suitable for a formal analysis of CRT-RSA. PARI/GP is a specialized computer algebra system, primarily aimed at solving number theory problems. Although PARI/GP can do a fair amount of symbolic manipulation, it remains limited compared to systems like Axiom, Magma, Maple, Mathematica, Maxima, or Reduce. Those last software also fall short to implement automatically number theoretic results like Euler's theorem. This explains why we developed from scratch a system to reason on modular numbers from a formal point of view. Our system is not general, in that it cannot for instance factorize terms in an expression. However, it is able to simplify recursively what is simplifiable from

a set of unambiguous rules. This behavior is suitable to the problem of resistance to fault attacks, because the redundancy that is added in the computation is meant to be simplified at the end (if no faults happened).

Our tool finja works within the framework of modular arithmetic, which is the mathematical framework of CRT-RSA computations (See App. B.2). The general idea is to represent the computation term as a tree which encodes the arithmetic properties of the intermediate variables. Our tool then does *symbolic computation* to simplify the term. This is done by *term rewriting*, using rules from arithmetic and the properties encoded in the tree. Fault injections in the computation term are simulated by changing the properties of a subterm, thus impacting the simplification process. An attack success condition is also given and used on the term resulting from the simplification to check whether the corresponding attack works on it. For each possible fault injection, if the attack success condition (which may reference the original term as well as the faulted one) can be simplified to *true* then the attack is said to work with this fault, otherwise the computation is protected against it. The outputs of finja are in HTML form: easily readable reports are produced, which contain all the information about the possible fault injections and their outcome.

High-Level Overview of finja

The process of verifying a countermeasure with finja can be described as such:

1. the user provides:
   - a description of the protected algorithm,
   - an attack success condition,
   - a fault model;
2. finja reads the algorithm and produce an internal representation of it (a tree), which it is able to symbolically execute (simplify);
3. finja makes a copy of the original tree and simplifies it;
4. for each possible fault(s) injection(s), finja:
   (a) produces a copy of the original tree;
   (b) injects the fault in the copy;
   (c) simplifies the faulted tree;
   (d) verifies if the attack success condition holds for the simplified faulted tree (and the simplified original tree, if applicable):
      - if it holds, then the working attack is reported,
      - if it does not, then the countermeasure is considered secure against this particular attack;
5. finja outputs a report in HTML form which recaps all the working attacks, and considers the countermeasure secure if no working attack were found during the complete coverage of the algorithm for the given fault model.

The important points of this high-level description are studied in more details in the rest of this section.

Computation Term

The computation is expressed in a convenient statement-based input language. This language's Backus–Naur Form (BNF) is given in Fig. 5.1.

A computation term is defined by a list of statements finished by a `return` statement. Each statement can either:

```
1  term     ::= ( stmt )* 'return' mp_expr ';'
2  stmt     ::= ( decl | assign | verif ) ';'
3  decl     ::= 'noprop' mp_var ( ',' mp_var )*
4             | 'prime' mp_var ( ',' mp_var )*
5  assign   ::= var ':=' mp_expr
6  verif    ::= 'if' mp_cond 'abort with' mp_expr
7  mp_expr  ::= '{' expr '}' | expr
8  expr     ::= '(' mp_expr ')'
9              | '0' | '1' | 'Random' | var
10             | '-' mp_expr
11             | mp_expr '+' mp_expr
12             | mp_expr '-' mp_expr
13             | mp_expr '*' mp_expr
14             | mp_expr '^' mp_expr
15             | mp_expr 'mod' mp_expr
16 mp_cond  ::= '{' cond '}' | cond
17 cond     ::= '(' mp_cond ')'
18             | mp_expr '=' mp_expr
19             | mp_expr '!=' mp_expr
20             | mp_expr '=[' mp_expr ']' mp_expr
21             | mp_expr '!=[' mp_expr ']' mp_expr
22             | mp_cond '/\' mp_cond
23             | mp_cond '\/' mp_cond
24 mp_var   ::= '{' var '}' | var
25 var      ::= [a-zA-Z][a-zA-Z0-9_']*
```

Figure 5.1: BNF of finja's input language.

- declare a variable with no properties (line 3);
- declare a variable which is a prime number (line 4);
- declare a variable by assigning it a value (line 5), in this case the properties of the variable are the properties of the assigned expression;
- perform a verification (line 6), which means testing an invariant and immediately abort and return a value (which can be any valid expression).

As can be seen in lines 9 to 15, an expression can be:

- zero, one, a random term[6], or an already declared variable;
- the sum (or difference) of two expressions;
- the product of two expressions;
- the exponentiation of an expression by another;
- the modulus of an expression by another.

The condition in a verification can be (lines 18 to 23):

- the equality or inequality of two expressions;
- the (non-)equivalence of two expressions modulo another (lines 20 and 21);
- the conjunction or disjunction of two conditions.

Optionally, variables (when declared using the `prime` or `noprop` keywords), ex-

[6] `Random` is not a keyword of the input language, but it is used in finja's output report as the representation of internal symbols.

pressions, and conditions can be protected (lines 3, 4, 7 and 16, `mp` stands for "maybe protected") from fault injection by surrounding them with curly braces. This is useful for instance when it is needed to express the properties of a variable which cannot be faulted in the studied attack model. For example, in CRT-RSA, the definitions of variables d_p, d_q, and i_q are protected because they are seen as input of the computation.

Finally, line 25 gives the regular expression that variable names must match (they start with a letter and then can contain letters, numbers, underscore, and simple quote).

After it is read by finja, the computation expressed in this input language is transformed into a tree (just like the abstract syntax tree in a compiler). This tree encodes the arithmetical properties of each of the intermediate variable, and thus its dependencies on previous variables. For instance, being null or being the product of other terms (and thus, being a multiple of each of them), are possible properties. The properties of intermediate variables can be everything that is expressible in the input language.

Example 5.1. The following is a code in finja's input language with the computed properties of the variables in comments:

```
1 prime p ;        -- p is a prime number
2 noprop a ;       -- a has no properties
3 b := a * p ;     -- b is a multiple of a and of p
4 c := 0 * b ;     -- c is null
5 d := b / a ;     -- d is prime, and is equal to p
6 return d;
```

Fault Injection

A fault injection on an intermediate variable is represented by changing the properties of the subterm (a node and its whole subtree in the tree representing the computation term) that represent it. In the case of a fault which forces at zero, then the whole subterm is replaced by a term which only has the property of being null (i.e., by `0`). In the case of a randomizing fault, by a term which has no properties (i.e., by `Random`).

In practice, finja simulates *all the possible fault injections* of the attack model it is launched with. The parameters allow to choose:

- *how many faults* have to be injected (however, the number of tests to be done is impacted by a factorial growth with this parameter, as is the time needed to finish the computation of the proof);
- *the type* of each fault (*randomizing* or *zeroing*);
- if *transient* faults are possible or if only *permanent* faults should be performed.

Example 5.2. If we have the term `t := a + b * c`, it can be faulted in five different ways (using the randomizing fault). In this original term, the variable `t` has the property of being the sum of `a` and a term which is a multiple of `b` and `c`.

1. `t := Random`, the result of the addition is faulted,
   in this first case, `t` has no more properties;

2. `t := Random + b * c`, a is faulted,
   in this second case, `t` has the property of being the sum of an unknown number and a term which is a multiple of `b` and `c`;
3. `t := a + Random`, the result of the multiplication is faulted,
   in this third case, `t` has the property of being the sum of `a` and an unknown number;
4. `t := a + Random * c`, b is faulted,
   in this fourth case, `t` has the property of being the sum of `a` and a term which is a multiple of an unknown number and `c`
5. `t := a + b * Random`, c is faulted,
   in the fifth case, `t` has the property of being the sum of `a` and a term which is a multiple of `b` and an unknown number.

Attack Success Condition

The attack success condition is expressed using the same condition language as presented in Sec. 5.3.2. It can use any variable introduced in the computation term, plus two special variables `_` and `@` which are respectively bound to the expression returned by the computation term as given by the user and to the expression returned by the computation with the fault injections. This success condition is checked for each possible faulted computation term.

Example 5.3. Consider the assignation from Ex. 5.2 and `t` as the returned value. If the properties that interest us is to know whether the result of the computation is congruent with a modulo b, then we can use `@ =[b] a` as the attack success condition. It would always be true if we has written `_ =[b] a` (or the equivalent `t =[b] a`) as it is true for `t` when it is not faulted. However, it will only be true for the fifth case of Ex. 5.2 when the computation is faulted. If we had used the zeroing fault, it would also have been true for the third and fourth cases.

Simplification Process

The simplification is implemented as a recursive traversal of the term tree, based on pattern-matching. It works just like a naive interpreter would, except it does symbolic computation only, and reduces the term based on rules from arithmetic. Simplifications are carried out in $\mathbb{Z}$ ring, and its $\mathbb{Z}_N$ subrings. The tool knows how to deal with most of the $\mathbb{Z}$ ring axioms:

- the neutral elements (0 for sums, 1 for products);
- the absorbing element (0, for products);
- inverses and opposites (only if N is prime);
- associativity and commutativity.

Remark 5.3. Note that finja does not implement distributivity as it is not confluent. Associativity is implemented by flattening as much as possible ("removing" all unnecessary parentheses), and commutativity is implemented by applying a stable sorting algorithm on the terms of products or sums. This behavior ensures the confluence of the rewriting system by making the simplification process entirely deterministic. The completeness of the rewriting system has not been proven yet.

The tool also knows about most of the properties that are particular to $\mathbb{Z}_N$ rings and applies them when simplifying a term modulo N:

```
1 noprop M, e ;
2 prime  {p}, {q} ;
3
4 dp := { e^-1 mod (p-1) } ;
5 dq := { e^-1 mod (q-1) } ;
6 iq := { q^-1 mod p } ;
7
8 Sp := M^dp mod p ;
9 Sq := M^dq mod q ;
10
11 S := Sq + (q * (iq * (Sp - Sq) mod p)) ;
12
13 return S ;
14
15 %%
16
17 _ != @ /\ ( _ =[p] @ \/ _ =[q] @ )
```

Figure 5.2: finja code for the unprotected CRT-RSA computation.

- identity:
  - $(a \mod N) \mod N = a \mod N$,
  - $N^k \mod N = 0$;
- inverse:
  - $(a \mod N) \times (a^{-1} \mod N) \mod N = 1$,
  - $(a \mod N) + (-a \mod N) \mod N = 0$;
- associativity and commutativity:
  - $(b \mod N) + (a \mod N) \mod N = a + b \mod N$,
  - $(a \mod N) \times (b \mod N) \mod N = a \times b \mod N$;
- subrings: $(a \mod N \times m) \mod N = a \mod N$.

In addition to those properties a few theorems are implemented to manage more complicated cases where the properties are not enough when conducting symbolic computations:

- Fermat's little theorem;
- its generalization, Euler's theorem.

Remark 5.4. The algorithm only exhibits evidence of safety. If after a fault injection, the algorithm does not simplify (see Sec. 5.3.2, § Simplification Process on page 65) to an error detection, then it might only reveal that some simplification is missing. However, if it does not claim safety, it produces a *simplified* occurrence of a possible weakness to be investigated further.

5.4 Study of an Unprotected CRT-RSA Computation

The description of the unprotected CRT-RSA computation in finja code is given in Fig. 5.2 (note the similarity of finja's input code with Alg. 5.1).

As we can see, the definitions of d_p, d_q, and i_q are protected so the computation

of the values of these variables cannot be faulted (since they are seen as inputs of the algorithm). After that, S_p and S_q are computed and then recombined in the last expression, as in Def. 5.1.

To test whether the BellCoRe attack works on a faulted version $\widehat{S}$, we perform the following tests (we note $|S|$ for the simplified version of S):

1. is $|S|$ different than $|\widehat{S}|$?
2. is $|S \mod p|$ equal to $|\widehat{S} \mod p|$?
3. is $|S \mod q|$ equal to $|\widehat{S} \mod q|$?

If the first test is true and at least one of the second and third is true, we have a BellCoRe attack, as seen in Sec. 5.2. This is what is described in the attack success condition (after the `%%` line).

Without transient faults enabled, and in a single fault model, there are 12 different fault injections of which 8 enable a BellCoRe attack with a randomizing fault, and 9 with a zeroing fault. As an example, replacing the intermediate variable holding the value of $i_q \cdot (S_p - S_q) \mod p$ in the final expression with zero or a random value makes the first and second tests false, and the last one true, and thus allows a BellCoRe attack.

5.5 Study of Shamir's Countermeasure

The description, using finja's formalism, of the CRT-RSA computation allegedly protected by Shamir's countermeasure is given in Fig. 5.3 (again, note the similarity with Alg. 5.2).

Using the same settings as for the unprotected implementation of CRT-RSA, we find that among the 31 different fault injections, 10 enable a BellCoRe attack with a randomizing fault, and 9 with a zeroing fault. This is not really surprising, as the test which is done on line 18 does not verify if a fault is injected during the computations of S_p or S_q, nor during their recombination in S. For instance zeroing or randomizing the intermediate variable holding the result of $S_p - S_q$ during the computation of S (line 16) results in a BellCoRe attack. To explain why there is this problem in Shamir's countermeasure, some context might be necessary. It can be noted that the fault to inject in the countermeasure must be more accurate in timing (since it targets an intermediate variable obtained by a *subtraction*) than the faults to achieve a BellCoRe attack on the unprotected CRT-RSA (since a fault during an *exponentiation* suffices). However, there is today a consensus to believe that it is very easy to pinpoint in time any single operation of a CRT-RSA algorithm, using a simple power analysis method [KJJ99]. Besides, timely fault injection benches exist. Therefore, the weaknesses in Shamir's countermeasure can indeed be practically exploited.

If the attacker can do *transient faults*, there are a lot more attacks: 66 different possible fault injections of which 24 enable a BellCoRe attack with a randomizing fault and 22 with a zeroing fault. In practice, a transient faults would translate into faulting the variable when it is read (e.g., in a register or on a bus), rather than in (persistent) memory. This behavior could also be the effect of a fault injection in cache, which is later replaced with the good value when it is read from memory again. To the authors knowledge, these are not impossible situations. Nonetheless, growing the power of the attacker to take that into account break some very important assumptions that are classical (sometimes even implicit) in the literature. It does

```
1  noprop error, M, d ;
2  prime {p}, {q}, r ;
3  iq := { q^-1 mod p } ;
4
5  p' := p * r ;
6  dp := d mod ((p-1) * (r-1)) ;
7  Sp' := M^dp mod p' ;
8
9  q' := q * r ;
10 dq := d mod ((q-1) * (r-1)) ;
11 Sq' := M^dq mod q' ;
12
13 Sp := Sp' mod p ;
14 Sq := Sq' mod q ;
15
16 S := Sq + (q * (iq * (Sp - Sq) mod p)) ;
17
18 if Sp' !=[r] Sq' abort with error ;
19
20 return S ;
21
22 %%
23
24 _ != @ /\ ( _ =[p] @ \/ _ =[q] @ )
```

Figure 5.3: finja code for the Shamir CRT-RSA computation.

not matter that the parts of the secret key are stored in a secure "key container" if their values can be a faulted at read time. Indeed, we just saw that allowing this kind of fault enable even more possibilities to carry out a BellCoRe attack successfully on a CRT-RSA computation protected by the Shamir's countermeasure. For instance, if the value of p is randomized for the computation of the value of S_p (line 13), then we have $S \neq \widehat{S}$, but also $S \equiv \widehat{S} \mod q$, which enables a BellCoRe attack, as seen in Sec. 5.2.

It is often asserted that the countermeasure of Shamir is unpractical due to its need for d (as mentioned in [ABF+02] and [Vig08a]), and because there is a possible fault attack on the recombination, i.e., line 16 (as mentioned in [Vig08a]). However, the attack on the recombination can be checked easily, by testing that $S - S_p \not\equiv 0 \mod p$ and $S - S_q \not\equiv 0 \mod q$ before returning the result. Notwithstanding, to our best knowledge, it is difficult to detect all the attacks our tool found, and so the existence of these attacks (new, in the sense they have not all been described previously) is a compelling reason for not implementing Shamir's CRT-RSA.

5.6 Study of Aumüller et al.'s Countermeasure

The description of the CRT-RSA computation protected by Aumüller et al.'s countermeasure is given in Fig. 5.4 (here too, note the similarity with Alg. 5.3)

Using the same method as before, we can prove that on the 52 different possible faults, *none* of which allow a BellCoRe attack, whether the fault is zero or random. This is a proof that the Aumüller et al.'s countermeasure works when there is one fault[7].

Since it allowed more attacks on the Shamir's countermeasure, we also tested the Aumüller et al.'s countermeasure against *transient faults* such as described in Sec. 5.5. There are 120 different possible fault injections when transient faults are activated, and Aumüller et al.'s countermeasure is resistant against such fault injections too.

We also used finja to confirm that the computation of d_p, d_q, and i_q (in terms of p, q, and d) must not be part of the algorithm. The countermeasure effectively needs these three variables to be inputs of the algorithm to work properly. For instance there is a BellCoRe attack if d_q happens to be zeroed. However, even with d_p, d_q, and i_q as inputs, we can still attempt to attack a CRT-RSA implementation protected by the Aumüller et al.'s countermeasure by doing more than one fault.

We then used finja to verify whether Aumüller et al.'s countermeasure would be resistant against *high order* attacks, starting with two faults. We were able to break it if at least one of the two faults was a zeroing fault. We found that this zeroing fault was used to falsify the condition of a verification, which is possible in our threat-model, but which was not in the one of the authors of the countermeasure. If we protect the conditions against fault injection, then the computation is immune two double-fault attacks too. However, even in this less powerful threat-model, a CRT-RSA computation protected by Aumüller et al.'s countermeasure is breakable using 3 faults, two of which must be zeroing the computations of d_{pt} and d_{qt}.

[7]This result is worthwhile some emphasis: the genuine algorithm of Aumüller is thus *proved* resistant against single-fault attacks. At the opposite, the CRT-RSA algorithm of Vigilant is not immune to single fault attacks (refer to [CGM+10]), and the corrections suggested in the same paper by Coron et al. have not been proved yet.

```
1  noprop error, M, e, r1, r2 ;
2  prime {p}, {q}, t ;
3
4  dp := { e^-1 mod (p-1) } ;
5  dq := { e^-1 mod (q-1) } ;
6  iq := { q^-1 mod p } ;
7
8  p' := p * t ;
9  dp' := dp + r1 * (p-1) ;
10 Sp' := M^dp' mod p' ;
11
12 if p' !=[p] 0 \/ dp' !=[p-1] dp abort with error ;
13
14 q' := q * t ;
15 dq' := dq + r2 * (q-1) ;
16 Sq' := M^dq' mod q' ;
17
18 if q' !=[q] 0 \/ dq' !=[q-1] dq abort with error ;
19
20 Sp := Sp' mod p ;
21 Sq := Sq' mod q ;
22
23 S := Sq + (q * (iq * (Sp - Sq) mod p)) ;
24
25 if S !=[p] Sp' \/ S !=[q] Sq' abort with error ;
26
27 Spt := Sp' mod t ;
28 Sqt := Sq' mod t ;
29 dpt := dp' mod (t-1) ;
30 dqt := dq' mod (t-1) ;
31
32 if Spt^dqt !=[t] Sqt^dpt abort with error ;
33
34 return S ;
35
36 %%
37
38 _ != @ /\ ( _ =[p] @ \/ _ =[q] @ )
```

Figure 5.4: finja code for the Aumüller et al. CRT-RSA computation.

5.7 Conclusions and Perspectives

We have formally proven the resistance of the Aumüller et al.'s countermeasure against the BellCoRe attack by fault injection on CRT-RSA. To our knowledge, it is the first time that a formal proof of security is done for a BellCoRe countermeasure.

During our research, we have raised several questions about the assumptions traditionally made by countermeasures. The possibility of fault at read time is, in particular, responsible for many vulnerabilities. The possibility of such fault means that part of the secret key can be faulted (even if only for one computation). It allows an interesting BellCoRe attack on a computation of CRT-RSA protected by Shamir's countermeasure. We also saw that the assumption that the result of conditional expression cannot be faulted, which is widespread in the literature, is a dangerous one as it increased the number of fault necessary to break Aumüller et al.'s countermeasure from 2 to 3.

The first of these two points demonstrates the lack of formal studies of fault injection attack and their countermeasures, while the second one shows the importance of formal methods in the field of implementation security.

As a first perspective, we would like to address the hardening of software codes of CRT-RSA under the threat of a bug attack. This attack has been introduced by Biham, Carmeli and Shamir [BCS08] at CRYPTO 2008. It assumes that a hardware has been trapped in such a way that there exists two integers a and b, for which the multiplication is incorrect. In this situation, Biham, Carmeli and Shamir mount an explicit attack scenario where the knowledge of a and b is leveraged to produce a faulted result, that can lead to a remote BellCoRe attack. For sure, testing for the correct functionality of the multiplication operation is impractical (it would amount to an exhaustive verification of 2^{128} multiplications on 64 bit computer architectures). Thus, it can be imagined to use a countermeasure, like that of Aumüller, to detect a fault (caused logically). Our aim would be to assess in which respect our fault analysis formal framework allows to validate the security of the protection. Indeed, a fundamental difference is that the fault is not necessarily injected at *one* random place, but can potentially show up at *several* places.

As another perspective, we would like to handle the repaired countermeasure of Vigilant [CGM+10] and the countermeasure of Kim [KKHH11]. Regarding Vigilant, the difficulty that our verification framework in OCaml [INR] shall overcome is to decide how to inject the remarkable identity $(1 + r)^{d_p} \equiv 1 + d_p \cdot r \mod r^2$: either it is kept as such such, like an *ad hoc* theorem (but we need to make sure it is called only at relevant places, since it is not confluent), or it is made more general (but we must ascertain that the verification remains tractable). However, this effort is worthwhile[8], because the authors themselves say in the conclusion of their article [CGM+10] that:

> *"Formal proof of the FA-resistance of Vigilant's scheme including our countermeasures is still an open (and challenging) issue."*

Regarding the CRT-RSA algorithm from Kim, the computation is very detailed (it goes down to the multiplication level), and involves Boolean operations (`and`, `xor`,

[8] Some results will appear in the proceedings of the 3rd ACM SIGPLAN Program Protection and Reverse Engineering Workshop (PPREW 2014) [RG14b], collocated with POPL 2014.

etc.). To manage that, more expertise about both arithmetic and logic must be added to our software.

Eventually, we wish to answer a question raised by Vigilant [Vig08a] about the prime t involved in Aumüller et al.'s countermeasure:

> *"Is it fixed or picked at random in a fixed table?"*

The underlying issue is that of *replay* attacks on CRT-RSA, that are more complicated to handle; indeed, they would require a formal system such as ProVerif [Bla], that is able to prove interactive protocols.

Concerning the tools we developed during our research, they currently only allow to study fault injection in the data, and not in the control flow, it would be interesting to enable formal study of fault injections affecting the control flow.

Eventually, we would like to define and then implement an automatic code mutation algorithm that could transform an unprotected CRT-RSA into a protected one. We know that with a few alterations (see that the differences between Alg. 5.1 and Alg. 5.3 are enumerable), this is possible. Such promising approach, if successful, would uncover the *smallest possible* countermeasure of CRT-RSA against fault injection attacks.

Acknowledgements

The authors wish to thank Jean-Pierre Seifert and Wieland Fischer for insightful comments and pieces of advice. We are also grateful to the anonymous reviewers of PROOFS 2013 (UCSB, USA), who helped improve the preliminary version of this chapter. Eventually, we acknowledge precious suggestions contributed by Jean-Luc Danger, Jean Goubault-Larrecq, and Karine Heydemann.

0x6

Formal Analysis of CRT-RSA Vigilant's Countermeasure

This chapter presents a joint work with Sylvain Guilley as my adviser. This work was first published at the PPREW 2014 workshop.

Abstract. In our paper at PROOFS 2013, we formally studied a few known countermeasures to protect CRT-RSA against the BellCoRe fault injection attack. However, we left Vigilant's countermeasure and its alleged repaired version by Coron et al. as future work, because the arithmetical framework of our tool was not sufficiently powerful. In this chapter we bridge this gap and then use the same methodology to formally study both versions of the countermeasure. We obtain surprising results, which we believe demonstrate the importance of formal analysis in the field of implementation security. Indeed, the original version of Vigilant's countermeasure is actually broken, but not as much as Coron et al. thought it was. As a consequence, the repaired version they proposed can be simplified. It can actually be simplified even further as two of the nine modular verifications happen to be unnecessary. Fortunately, we could formally prove the simplified repaired version to be resistant to the BellCoRe attack, which was considered a "challenging issue" by the authors of the countermeasure themselves.

Contents

6.1 Introduction

Private information protection is a highly demanded feature, especially in the context of global defiance against most infrastructures, assumed to be controlled by governmental agencies. Properly used cryptography is known to be a key building block for secure information exchange. However, in addition to the threat of cyber-attacks, implementation-level hacks are also to be considered seriously. This chapter deals specifically with the protection of a *decryption* or *signature* crypto-system (called *Rivest–Shamir–Adleman* (RSA) [RSA78]) in the presence of hardware attacks (e.g., we assume the attacker can alter the RSA computation while it is being executed).

It is known since 1997 [BDL97] that injecting faults during the computation of CRT-RSA could yield to malformed signatures that expose the prime factors (p and q) of the public modulus ($N = p \cdot q$). Notwithstanding, computing without the four-fold acceleration conveyed by the Chinese Remainder Theorem (CRT) is definitely not an option in practical applications. Therefore, many countermeasures have appeared that consist in step-wise internal checks during the CRT computation. Last year we formally studied some of them [RG14a]. We were able to formally prove that the unprotected implementation of CRT-RSA as well as the countermeasure proposed by Shamir [Sha99] are broken, and that Aumüller et al.'s countermeasure [ABF+02] is resistant to the BellCoRe attack. However, we were not able to study Vigilant's countermeasure [Vig10, Vig08a] or its repaired version by Coron et al. [CGM+10], because the tool we developed lacked the ability to work with arithmetical properties necessary to handle these countermeasures. In particular, our framework was unaware of the easiness to compute the discrete logarithm in some composite degree residuosity classes. These difficulties were foreseen by Coron et al. themselves in the conclusion of their paper [CGM+10]:

> *"Formal proof of the FA-resistance of Vigilant's scheme including our countermeasures is still an open (and challenging) issue."*

This is precisely the purpose of this chapter. The state-of-the-art of formal proofs of Vigilant's countermeasure is the work of Christofi, Chetali, Goubin, and Vigilant [CCGV13]. However, for tractability reasons, the proof is conducted on reduced versions of the algorithms parameters. One fault model is chosen authoritatively (*viz.* the zeroization of a complete intermediate data), which is a strong assumption. In addition, the verification is conducted on a specific implementation in pseudocode, hence concerns about its portability after compilation into machine-level code.

Contributions. We improved finja[1] (our tool based on symbolic computation in the framework of modular arithmetic [RG14a], see App. B.2) to enable the formal study of CRT-RSA Vigilant's countermeasure against the BellCoRe attack. The finja tool allows a full fault coverage of CRT-RSA algorithm, thereby keeping the proof valid even if the code is transformed (e.g., optimized, compiled, partitioned in software/hardware, or equipped with dedicated countermeasures). We show that the original countermeasure [Vig08a] is indeed broken, but not as much as Coron et al. thought it was when they proposed a *manually* repaired version of it [CGM+10].

[1] http://pablo.rauzy.name/sensi/finja.html

We simplify the repaired version accordingly, and formally prove it resistant to the BellCoRe attack under a fault model that considers *permanent faults* (in memory) and *transient faults* (one-time faults, even on copies of the secret key parts, e.g., during their transit on buses or when they reside on register banks), with or without forcing at zero, and with possibly faults at various locations. Thanks to the formal analysis, we are able to simplify the countermeasure even further: two of the nine checks are unnecessary.

Organization of the chapter. Vigilant's countermeasure and its variant by Coron et al. are exposed in Sec. 6.2. In Sec. 6.3 we detail the modular arithmetic framework used by our tool. The results of our analysis are presented in Sec. 6.4. Finally, conclusions and perspectives are given in Sec. 6.5. After that, Appx. 6.A details the practical issues related to fault injection analysis, for single- and multiple-fault attacks.

6.2 Vigilant's Countermeasure Against the BellCoRe Attack

Fault attacks on RSA can be thwarted simply by refraining from implementing the CRT. If this is not affordable, then the signature can be verified before being outputted. Such protection is efficient in practice, but is criticized for two reasons. First of all, it requires an access to e; second, the performances are incurred by the extra exponentiation needed for the verification. This explains why several other countermeasures have been proposed.

Until recently, none of these countermeasures had been proved. In 2013, Christofi, Chetali, Goubin, and Vigilant [CCGV13] formally proved an implementation of Vigilant's countermeasure. However, for tractability purposes, the proof is conducted on reduced versions of the algorithms parameters. One fault model is chosen authoritatively (the zeroization of a complete intermediate data), which is a strong assumption. In addition, the verification is conducted on a specific implementation in pseudocode, hence concerns about its portability after compilation into machine-level code. The same year, we formally studied [RG14a] the ones of Shamir [Sha99] and Aumüller et al. [ABF+02] using a tool based on the framework of modular arithmetic, thus offering a full fault coverage on CRT-RSA algorithm, thereby keeping the proof valid even if the code is transformed (e.g., optimized, compiled or partitioned in software/hardware). We proved the former to be broken and the latter to be resistant to the BellCoRe attack.

6.2.1 Vigilant's Original Countermeasure

In his paper at CHES 2008, Vigilant [Vig08a] proposed a new method to protect CRT-RSA. Its principle is to compute $M^d \mod N$ in $\mathbb{Z}_{Nr^2}$ where r is a random integer that is coprime with N. Then M is transformed into M^* such that

$$M^* \equiv \begin{cases} M \mod N, \\ 1 + r \mod r^2, \end{cases}$$

which implies that

$$S^* = M^{*d} \mod Nr^2 \equiv \begin{cases} M^d \mod N, \\ 1 + dr \mod r^2. \end{cases}$$

The latter results are based on the binomial theorem, which states that:

$$\begin{aligned} (1+r)^d &= \sum_{k=0}^{d} \binom{d}{k} r^k \\ &= 1 + dr + \binom{d}{2} r^2 + \text{higher powers of } r, \end{aligned} \tag{6.1}$$

which simplifies to $1+dr$ in the ring $\mathbb{Z}_{r^2}$. This property can be seen as a special case of an easy computation of a discrete logarithm in a composite degree residuosity class (refer for instance to [Pai99]). Therefore the result S^* can be checked for consistency modulo r^2. If the verification $S^* \stackrel{?}{=} 1 + dr \mod r^2$ succeeds, then the final result $S = S^* \mod N$ is returned. Concerning the random numbers used in the countermeasure, its author recommend using a 32 bits integer for r, and 64 bits for the R_is. The original algorithm for Vigilant's countermeasure is presented in Alg. 6.1.

6.2.2 Coron et al. Repaired Version

At FDTC 2010, Coron et al. [CGM+10] proposed some corrections to Vigilant's original countermeasure. They claimed to have found three weaknesses in integrity verifications:

1. one for the computation modulo $p - 1$;
2. one for the computation modulo $q - 1$;
3. and one in the final check modulo Nr^2.

They propose a fix for each of these weaknesses. To correct weakness #1, lines 9 to 11 in Alg. 6.1 are replaced with the content of Alg. 6.2.

Similarly, for weakness #2, lines 22 to 24 in Alg. 6.1 are replaced with the content of Alg. 6.3.

These two fixes immediately looked suspicious to us. Indeed, Coron et al. suppose that the computations of $p - 1$ and $q - 1$ are factored as an optimization, while they are not in the original version of Vigilant's countermeasure. This kind of optimization, called "common subexpression elimination", is very classical but is, as for most speed-oriented optimization, not a good idea in security code. We will see in Sec. 6.4 that the original algorithm, which did not include this optimization, was not at fault here.

The third fix proposed by Coron et al. consists in replacing lines 28 to 29 of Alg. 6.1 by Alg. 6.4.

What has changed is the recomputation of $p \cdot q$ instead of reusing N in the verification. The idea is that if p or q is faulted when N is computed then error will be returned. In the original countermeasure, if p or q were to be faulted, the faulted version of N would appear in the condition twice, simplifying itself out and thus hiding the fault.

Algorithm 6.1: Vigilant's CRT-RSA

Input : Message M, key (p, q, d_p, d_q, i_q)
Output: Signature $M^d \mod N$, or error

Choose random numbers r, R_1, R_2, R_3, and R_4.
$p' = pr^2$
$M_p = M \mod p'$
$i_{pr} = p^{-1} \mod r^2$
$B_p = p \cdot i_{pr}$
$A_p = 1 - B_p \mod p'$
$M'_p = A_p M_p + B_p \cdot (1 + r) \mod p'$
if $M'_p \not\equiv M \mod p$ **then return** error
$d'_p = d_p + R_1 \cdot (p - 1)$
$S_{pr} = {M'_p}^{d'_p} \mod p'$
if $d'_p \not\equiv d_p \mod p - 1$ **then return** error
if $B_p S_{pr} \not\equiv B_p \cdot (1 + d'_p r) \mod p'$ **then return** error
$S'_p = S_{pr} - B_p \cdot (1 + d'_p r - R_3)$
$q' = qr^2$
$M_q = M \mod q'$
$i_{qr} = q^{-1} \mod r^2$
$B_q = q \cdot i_{qr}$
$A_q = 1 - B_q \mod q'$
$M'_q = A_q M_q + B_q \cdot (1 + r) \mod q'$
if $M'_q \not\equiv M \mod q$ **then return** error
if $M_p \not\equiv M_q \mod r^2$ **then return** error
$d'_q = d_q + R_2 \cdot (q - 1)$
$S_{qr} = {M'_q}^{d'_q} \mod q'$
if $d'_q \not\equiv d_q \mod q - 1$ **then return** error
if $B_q S_{qr} \not\equiv B_q \cdot (1 + d'_q r) \mod q'$ **then return** error
$S'_q = S_{qr} - B_q \cdot (1 + d'_q r - R_4)$
$S = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod p')$
$N = pq$
if $N \cdot (S - R_4 - q \cdot i_q \cdot (R_3 - R_4)) \not\equiv 0 \mod Nr^2$ **then return** error
if $q \cdot i_q \not\equiv 1 \mod p$ **then return** error
return $S \mod N$

Algorithm 6.2: Vigilant's CRT-RSA: Coron et al. fix #1

$p_{minusone} = p - 1$
$d'_p = d_p + R_1 \cdot p_{minusone}$
$S_{pr} = {M'_p}^{d'_p} \mod p'$
$p_{minusone} = p - 1$
if $d'_p \not\equiv d_p \mod p_{minusone}$ **then return** error

Algorithm 6.3: Vigilant's CRT-RSA: Coron et al. fix #2

1 $q_{minusone} = q - 1$
2 $d'_q = d_q + R_2 \cdot q_{minusone}$
3 $S_{qr} = M_q'^{d'_q} \mod q'$
4 $q_{minusone} = q - 1$
5 **if** $d'_q \not\equiv d_q \mod q_{minusone}$ **then return** error

Algorithm 6.4: Vigilant's CRT-RSA: Coron et al. fix #3

1 $N = p \cdot q$
2 **if** $p \cdot q \cdot (S - R_4 - q \cdot iq \cdot (R_3 - R_4)) \not\equiv 0 \mod Nr^2$ **then return** error

6.3 Formal Methods

Here we briefly recall what has been introduced in Sec. 5.3 of the previous chapter.

6.3.1 CRT-RSA and Fault Injection

Definition 6.1 (CRT-RSA)**.** The CRT-RSA computation takes as input a message M, assumed known by the attacker but which we consider to be random, and a secret key (p, q, d_p, d_q, i_q). Then, the implementation is free to instantiate any variable, but must return a result equal to $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$, where:
- $S_p = M^{d_p} \mod p$, and
- $S_q = M^{d_q} \mod q$.

Definition 6.2 (Fault injection)**.** An attacker is able to request RSA computations, as per Def. 6.1. During the computation, the attacker can modify any intermediate value by setting it to either a *random value* or *zero*. At the end of the computation the attacker can read the result.

Remark 6.1. We notice that the fault injection model of Def. 6.2 corresponds to that of Vigilant [Vig08a], with the exception that the conditional tests can also be faulted. To summarize, an attacker can modify a value in the global memory (*permanent fault*), and modify a value in a local register or bus (*transient fault*), but cannot inject a permanent fault in the input data (message and secret key), nor modify the control flow graph.

6.3.2 Improvement to finja

The working principle of finja have been described in Sec. 5.3.2 of the previous chapter. A few improvements have been necessary to enable the analysis of Vigilant's countermeasure.

The rewriting engine of finja now knows about two these arithmetic theorems:
- the Chinese Remainder Theorem (CRT); and
- a particular case of the binomial theorem (see Sec. 6.2.1).

In addition it also implements a lemma that is used for modular (in)equality verifications (note that a or b can be 0): $a \times x \stackrel{?}{\equiv} b \times x \mod N \times x$ is simplified to the equivalent test $a \stackrel{?}{\equiv} b \mod N$.

6.4 Analysis of Vigilant's Countermeasure

This section presents and discusses the results of our formal analysis of Vigilant's countermeasure.

6.4.1 Original and Repaired Version

The original version of Vigilant's countermeasure in our tool's input language can be found in Appx. 6.B.

In a single fault attack model, we found that the countermeasure has a single point of failure: if transient fault are possible, then if p or q are randomized in the computation of N at line 44, then a BellCoRe attack works.

The repaired version by Coron et al. thus does unnecessary modifications, since it fixes three spotted weaknesses (as seen in Sec. 6.2.2) while there is only one to fix. Only implementing the modification of Alg. 6.4 is sufficient to be provably protected against the BellCoRe attack.

After that, we went on to see what would happen in a multiple fault model. What we found is the object of the next section.

6.4.2 Our Fixed and Simplified Version

Attacking with multiple faults when at least one of the fault is a zeroing fault, means that some verification can be skipped by having their condition zeroed. We found that two of the tests could be skipped without impacting the protection of the computation (the lines 40 and 49). Both tests have been removed from our fixed and simplified version, which can be found in Appx. 6.C. We have proved that removing any other verification makes the computation vulnerable to single fault injection attacks, thereby proving the *minimality* of the obtained countermeasure.

Vigilant's original fault model did not include the possibility to inject fault in the conditions of the verifications. If we protect them from fault injection, we found that it is still possible to perform a BellCoRe attack exploiting two faults; Woudenberg et al. [vWWM11] showed that two-fault attacks are realistic. Both faults must be zeroing faults. One on R_3 (resp. R_4), and one on S'_p (resp. S'_q). It works because the presence of R_3 (resp. R_4) in the last verification is not compensated by its presence in S'_p (resp. S'_q). This shows that even security-oriented optimizations cannot be applied mindlessly in security code, which once again proves the importance of formal methods.

6.4.3 Comparison with Aumüller et al.'s Countermeasure

Now that we have (in Appx. 6.C) a version of Vigilant's countermeasure formally proven resistant to BellCoRe attack, it is interesting to compare it with the only other one in this case, namely Aumüller et al.'s countermeasure. Both countermeasures were developed without the help of formal methods, by trial-and-error engineering, accumulating layers of intermediate computations and verifications to patch weaknesses found at the previous iteration. This was shown, unsurprisingly, to be an ineffective development method. On one hand, there is no guarantee that all weaknesses are eliminated, and there was actually a point of failure left in the original version of Vigilant's countermeasure. On the other hand, there is no guarantee

that the countermeasure is minimal. Vigilant's countermeasure could be simplified by formal analysis, we removed two out of its nine verifications. This proves that "visual" code verification as is general practice in many test/certification labs, even for such a concise algorithm is far from easy in practice. Moreover, the difficulty of this task is increased by the multiplicity of the fault model (permanent vs. transient, randomizing vs. zeroing, single or double).

Apart from these saddening similarities in the development process, there are notable distinctions between the two countermeasures. Vigilant's method exposes the secret values p and q at two different places, namely during recombination and reduction from modulo pqr^2 to pq. Generally speaking, it is thus less safe than Aumüller et al.'s method. In terms of computational complexity the two countermeasures are roughly equivalent. Aumüller et al.'s one does 4 exponentiations instead of 2 for Vigilant's, but the 2 additional ones are with 16 bits numbers, which is entirely negligible compared to the prime factors (p and q are $1,024$ bits numbers). Vigilant's countermeasure requires more random numbers, which may be costly but is certainly negligible too compared to the exponentiations.

6.5 Conclusions and Perspectives

We have formally proven the resistance of a fixed version of Vigilant's CRT-RSA countermeasure against the BellCoRe fault injection attack. Our research allowed us to remove two out of nine verifications that were useless, thereby simplifying the protected computation of CRT-RSA while keeping it formally proved. Doing so, we believe that we have shown the importance of formal methods in the field of implementation security. Not only for the development of trustable devices, but also as an *optimization enabler*, both *for speed and security*. Indeed, Coron et al.'s repaired version of Vigilant's countermeasure included fixes for weaknesses that have been introduced by a hasty speed-oriented optimization, (which is never a good idea in the security field), and two faults attacks are possible only because of random numbers that have been introduced to reinforce the countermeasure.

As a first perspective, we would like to further improve our tool. It is currently only able to inject faults in the data. It would be interesting to be able to take into account faults on instructions such as studied by Heydemann et al. [HMER13]. It would be interesting as well to be able to automatically refine some properties of the variables, for instance to study attacks using a chosen message as input of the algorithm. Also, multiple fault analyses can take up to several dozens of minutes to compute. Since each attack is independent, our tool would greatly benefit (2 to 8 times speed-up on any modern multi-core computer) from a parallelization of the computations. Finally, it is worthwhile to note that our tool is *ad hoc* in the sense that we directly inject the necessary knowledge into its core. It would be interesting to see if general purpose tool such as EasyCrypt [BGZB09] could be a good fit for this kind of work.

Acknowledgements

I would like to thank David Vigilant for his insightful answers to our questions.

6.A Practical Feasibility of the Identified Attacks

In this section, we discuss the key features that an attacker must control for his attacks to succeed. We first describe in Sec. 6.A.1 the timing characteristics of a CRT-RSA computation implemented on a smartcard. Second, we analyze in Sec. 6.A.2 the performance of a fault injection bench. Third, we analyse in Sec. 6.A.3 which fault injection is realistic or, at the contrary, challenging practically speaking.

6.A.1 CRT-RSA Algorithm in a Smartcard

We consider a smartcard implementation of CRT-RSA. It consists in an arithmetic hardware accelerator, capable of computing multiplications and additions. It is thus termed a Multiplier-Accumulator Circuit (MAC). This MAC can manipulate data on 64 bits. In 65 nm and lower technologies, such a MAC can run at the frequency of 100 MHz (after logical synthesis in a low power standard cell library). Let us consider that N is a 2048 bits number, i.e., p and q hold on 1024 bits. The computations are carried out in the so-called Montgomery representation. For instance, when computing a modular multiplication, the multiplication and the reductions are interleaved. Thus, we derive some approximate timings for the unitary operations.

- One schoolbook addition on $\mathbb{Z}_p$ (or equivalently on $\mathbb{Z}_q$) of two large numbers takes $1024/64 = 16$ clock cycles, i.e., 160 ns;
- One schoolbook squaring or multiplication on $\mathbb{Z}_p$ of two large numbers takes $2 \times (1024/64)^2 = 512$ clock cycles, i.e., 5.12 μs; The factor two takes into the account the reduction steps.
- One schoolbook exponentiation on $\mathbb{Z}_p$ of one large number by another large number takes $512 \times (1024 \times 3/2) = 786432$ clock cycles, i.e., about 7.8 ms; This estimation assumes the square-and-multiply exponentiation algorithm, in which there are as many squarings as bits in p, but only half (in average) multiplications.

Besides, we notice that some operations in Alg. 6.1 are hybrid: they involve one large number and one small number (e.g., 64 bits number, of the size of r, R_1, R_2, R_3 or R_4). Obviously, those operations unfold more quickly than operations with two large numbers. Some operations in Alg. 6.2 can even be implemented to be extremely fast. For instance, the subtraction on line 1 or 4 of Alg. 6.2 (resp. Alg. 6.3) can be imagined to be done in one clock cycle, i.e., 10 ns. Indeed, as p and q are odd (because they are large prime numbers), this "decrementation" consists simply in resetting the least significant bit of the operand p (resp. q).

So, concluding, the operations in the CRT-RSA algorithm can be of diverse durations, ranging 6 orders of magnitude, from 10 ns for operations computable in one clock cycle, to about 8 ms for one modular exponentiation.

6.A.2 Fault Injection Setup

An efficient fault injection method consists in the production of an Electromagnetic (EM) pulse of high energy in the vicinity of the circuit that runs the CRT-RSA. This kind of fault is termed Electromagnetic Fault Injection (EMFI). Some setups can feature the possibility of multiple fault injections. Simple, double and triple EM pulses are illustrated respectively in Fig. 6.1, 6.2 and 6.3.

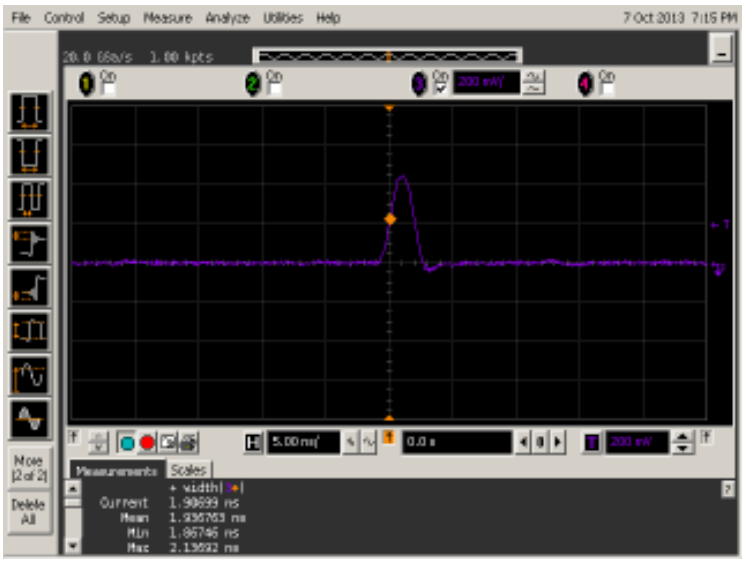


Figure 6.1: First-order fault injection attack — proof of concept.

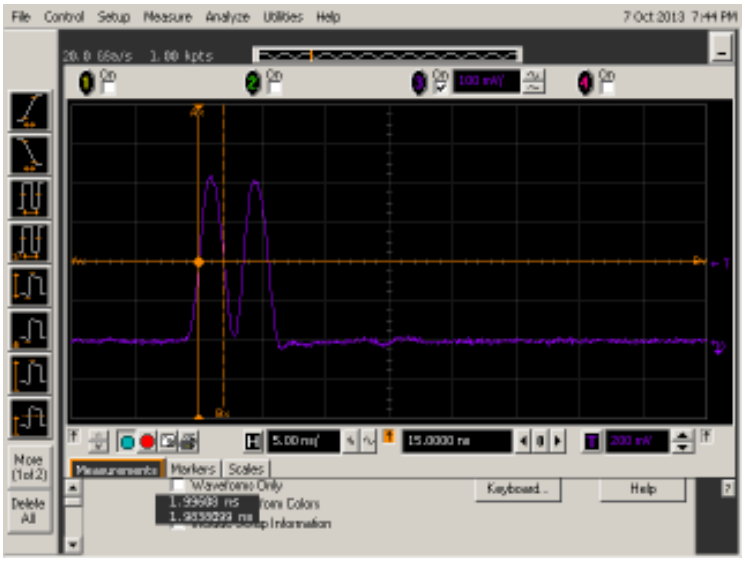


Figure 6.2: Second-order fault injection attack — proof of concept.

We detail below the technical characteristics of a fairly high-end bench[2]:

- Pulse amplitude: ..... 0—500 mV.
- Pulse duration: ..... 2 ns.
- Repeatability: ..... 2 ns $\iff$ 500 MHz.

6.A.3 Practicality of the Attacks Identified by finja

The *pulse amplitude* (or "*energy*") shall be sufficiently high for the EMFI to have an effect, and not too large to avoid chip damage. A typical method to set the adequate level of energy is to run the CRT-RSA computations iteratively, while increasing the energy from zero, with small increments (e.g., a few millivolts). After a given number of iterations, the CRT-RSA signature will be faulty or the circuit

[2]We credit Laurent Sauvage for this information, that comes from the specification of one plug-in of the Smart-SIC Analyzer tool of Secure-IC S.A.S.

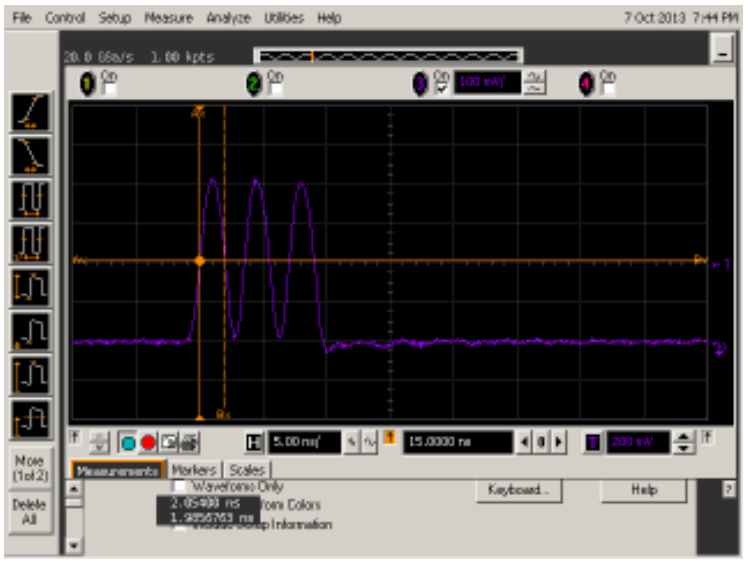


Figure 6.3: Third-order fault injection attack — proof of concept.

under attack will refuse to give an answer. From this value on, the EMFI starts to have an effect on the chip. Perhaps this fault injection is probabilistic, meaning that it does not always have an effect. So, it is safe to increase the energy level of a few percents to raise the injection probability of success as close to one as possible.

The *pulse duration* is an important parameter, because it conditions the accuracy in time of the stress. As mentioned in Sec. 6.A.1, there is a great variability in the duration of the operations. Therefore, some attacks will be more difficult to set up successfully. For exemple, the two corrections (Alg. 6.2 and 6.4) finally repair a flaw that is fairly difficult to fault, because the operations last:

- 1 clock cycle, i.e., 10 ns, and
- 512 clock cycles, i.e., 5.12 μs respectively.

Implementations of CRT-RSA are generally working in constant time, to prevent timing attacks [Koc96]. Those attacks are indeed to be taken very seriously, as they can even be launched without a physical access to the device. Paradoxically, this helps carry out timely attacks on focused stages in the algorithm. So, with the setup described in Sec. 6.A.2, such fault attacks remain possible. Also, multiple faults can be injected easily if the algorithm timing is deterministic, because the *repetition rate* of the setup is greater than once per clock cycle. Indeed, a new pulse can be generated every other period of 2 ns, i.e., at a maximal rate of 500 MHz.

However, a bench of lower quality would deny some exploits, for instance those that require: (*a*) to fault an operation of small duration; (*b*) for high-order fault attacks, operations that are too close one from each other.

6.B Vigilant's Original Countermeasure

```
noprop error, M, e, r, R1, R2, R3, R4 ;
prime {p}, {q} ;

dp := { e^-1 mod (p-1) } ;
dq := { e^-1 mod (q-1) } ;
iq := { q^-1 mod p } ;

--- p part ---
p' := p * r * r ;
Mp := M mod p' ;
ipr := p^-1 mod (r * r) ;
Bp := p * ipr ;
Ap := 1 - Bp mod p' ;
M'p := Ap * Mp + Bp * (1 + r) mod p' ;
if M'p !=[p] M abort with error ;

d'p := dp + R1 * (p - 1) ;
Spr := M'p^d'p mod p' ;
if d'p !=[p - 1] dp abort with error ;
if Bp * Spr !=[p'] Bp * (1 + d'p * r) abort with error ;

S'p := Spr - Bp * (1 + d'p * r - R3) ;

--- q part ---
q' := q * r * r ;
Mq := M mod q' ;
iqr := q^-1 mod (r * r) ;
Bq := q * iqr ;
Aq := 1 - Bq mod q' ;
M'q := Aq * Mq + Bq * (1 + r) mod q' ;
if M'q !=[q] M abort with error ;

d'q := dq + R2 * (q - 1) ;
Sqr := M'q^d'q mod q' ;
if d'q !=[q - 1] dq abort with error ;
if Bq * Sqr !=[q'] Bq * (1 + d'q * r) abort with error ;

S'q := Sqr - Bq * (1 + d'q * r - R4) ;

if Mp !=[r * r] Mq abort with error ;

--- recombination ---
S := S'q + q * (iq * (S'p - S'q) mod p') ;
N := p * q ;

if N * (S - R4 - q * (iq * (R3 - R4)))
  !=[N * r * r] 0 abort with error ;

if q * iq !=[p] 1 abort with error ;

return S mod N ;

%%

_ != @ /\ ( _ =[p] @ \/ _ =[q] @ )
```

6.C Fixed Vigilant's Countermeasure

```
noprop error, M, e, r, R1, R2, R3, R4 ;
prime {p}, {q} ;

dp := { e^-1 mod (p-1) } ;
dq := { e^-1 mod (q-1) } ;
iq := { q^-1 mod p } ;

--- p part ---
p' := p * r * r ;
Mp := M mod p' ;
ipr := p^-1 mod (r * r) ;
Bp := p * ipr ;
Ap := 1 - Bp mod p' ;
M'p := Ap * Mp + Bp * (1 + r) mod p' ;
if M'p !=[p] M abort with error ;

d'p := dp + R1 * (p - 1) ;
Spr := M'p^d'p mod p' ;
if d'p !=[p - 1] dp abort with error ;
if Bp * Spr !=[p'] Bp * (1 + d'p * r) abort with error ;

S'p := Spr - Bp * (1 + d'p * r - R3) ;

--- q part ---
q' := q * r * r ;
Mq := M mod q' ;
iqr := q^-1 mod (r * r) ;
Bq := q * iqr ;
Aq := 1 - Bq mod q' ;
M'q := Aq * Mq + Bq * (1 + r) mod q' ;
if M'q !=[q] M abort with error ;

d'q := dq + R2 * (q - 1) ;
Sqr := M'q^d'q mod q' ;
if d'q !=[q - 1] dq abort with error ;
if Bq * Sqr !=[q'] Bq * (1 + d'q * r) abort with error ;

S'q := Sqr - Bq * (1 + d'q * r - R4) ;

-- useless verification removed

--- recombination ---
S := S'q + q * (iq * (S'p - S'q) mod p') ;
N := p * q ;

if p * q * (S - R4 - q * (iq * (R3 - R4)))
  !=[N * r * r] 0 abort with error ;

-- useless verification removed

return S mod N ;

%%

_ != @ /\ ( _ =[p] @ \/ _ =[q] @ )
```

0x7

High-Order Countermeasures Against Fault Attacks on CRT-RSA

This chapter presents a joint work with Sylvain Guilley as my adviser. This work was first published at FDTC 2014.

Abstract. In this chapter we study the existing CRT-RSA countermeasures against fault-injection attacks. In an attempt to classify them we get to achieve deep understanding of how they work. We show that the many countermeasures that we study (and their variations) actually share a number of common features, but optimize them in different ways. We also show that there is no conceptual distinction between test-based and infective countermeasures and how either one can be transformed into the other. Furthermore, we show that faults on the code (skipping instructions) can be captured by considering only faults on the data. These intermediate results allow us to improve the state of the art in several ways: (a) we fix an existing and that was known to be broken countermeasure (namely the one from Shamir); (b) we drastically optimize an existing countermeasure (namely the one from Vigilant) which we reduce to 3 tests instead of 9 in its original version, and prove that it resists not only one fault but also an arbitrary number of randomizing faults; (c) we also show how to upgrade countermeasures to resist any given number of faults: given a correct first-order countermeasure, we present a way to design a provable high-order countermeasure (for a well-defined and reasonable fault model). Finally, we pave the way for a generic approach against fault attacks for any modular arithmetic computations, and thus for the automatic insertion of countermeasures.

Contents

7.1 Introduction

Private information protection is a highly demanded feature, especially in the current context of global defiance against most infrastructures, assumed to be controlled by governmental agencies. Properly used cryptography is known to be a key building block for secure information exchange. However, in addition to the threat of cyber-attacks, implementation-level hacks must also be considered seriously. This chapter deals specifically with the protection of a *decryption* or *signature* crypto-system (called RSA [RSA78]) in the presence of hardware attacks (e.g., we assume the attacker can alter the Rivest–Shamir–Adleman (RSA) computation while it is being executed).

It is known since 1997 (with the BellCoRe attack by Boneh et al. [BDL97]) that injecting faults during the computation of Chinese Remainder Theorem (CRT) optimized RSA algorithm could yield to malformed signatures that expose the prime factors (p and q) of the public modulus ($N = p \cdot q$). Notwithstanding, computing without the fourfold acceleration conveyed by the CRT optimization is definitely not an option in practical applications. Therefore, many countermeasures have appeared. Most of the existing countermeasures were designed with an attack-model consisting in a single fault injection. The remaining few attempts to protect against second-order fault attacks (i.e., attacks with two faults).

Looking at the history of the development of countermeasures against the Bell-CoRe attack, we see that many countermeasures are actually broken in the first place. Some of them were fixed by their authors and/or other people, such as the countermeasure proposed by Vigilant [Vig08a], which was fixed by Coron et al. [CGM+10] and then simplified by Rauzy & Guilley [RG14b]; some simply abandoned, such as the one by Shamir [Sha99]. Second-order countermeasures are no exception to that rule, as demonstrated with the countermeasure proposed by Ciet & Joye [CJ05], which was fixed later by Dottax et al. [DGRS09]. Such mistakes can be explained by two main points:

- the almost nonexistent use of formal methods in the field of implementation security, which can itself be explained by the difficulty to properly model the physical properties of an implementation which are necessary to study side-channel leakages and fault-injection effects;
- the fact that most countermeasures were developed by trial-and-error engineering, accumulating layers of intermediate computations and verifications to patch weaknesses until a fixed point was reached, even if the inner workings of the countermeasure were not fully understood.

Given their development process, it is likely the case that existing second-order countermeasures would not resist third-order attacks, and strengthening them against such attacks using the same methods will not make them resist fourth-order, etc.

The purpose of this chapter is to remedy to these problems. First-order countermeasures have started to be formally studied by Christofi et al. [CCGV13], who have been followed by Rauzy & Guilley [RG14a, RG14b], and Barthe et al. [BDF+14a]. To our best knowledge, no such work has been attempted on high-order countermeasures. Thus, we should understand the working factors of a countermeasure, and use that knowledge to informedly design a generic high-order countermeasure, either one resisting any number of faults, or one which could be customized to protect against n faults, for any given $n \geqslant 1$.

Notice that we consider RSA used in a mode where the BellCoRe attack is applicable; this means that we assume that the attacker can choose (but not necessarily knows) the message that is exponentiated, which is the case in *decryption* mode or in (outdated) *deterministic signature* mode (e.g., PKCS #1 v1.5). In some other modes, formal proofs of security have been conducted [CM09, BDF+14a].

Contributions. In this chapter we propose a classification of the existing CRT-RSA countermeasures against the BellCoRe fault-injection attacks. Doing so, we raise questions whose answers lead to a deeper understanding of how the countermeasures work. We show that the many countermeasures that we study (and their variations) are actually applying a common protection strategy but optimize it in different ways (Sec. 7.4). We also show that there is no conceptual distinction between test-based and infective countermeasures and how either one can be transformed into the other (Prop. 7.1). Furthermore, we show that faults on the code (skipping instructions) can be captured by considering only faults on the data (Lem. 7.1). These intermediate results allow us to improve the state of the art in several ways:

- we fix an existing and that is known to be broken countermeasure (Alg. 7.10);
- we drastically optimize an existing countermeasure, while at the same time we transform it to be infective instead of test-based (Alg. 7.11);
- we also show how to upgrade countermeasures to resist any given number of faults: given a correct first-order countermeasure, we present a way to design a provable high-order countermeasure for a well defined and reasonable fault model (Sec. 7.4.2).

Finally, we pave the way for a generic approach against fault attacks for any modular arithmetic computations, and thus for the automatic insertion of countermeasures.

Organization of the chapter. We recall the CRT-RSA cryptosystem and the BellCoRe attack in Sec. 7.2. Then, to better understand the existing countermeasures, we attempt to classify them in Sec. 7.3, which also presents the state of the art. We then try to capture what make the essence of a countermeasure in Sec. 7.4, and use that knowledge to determine how to build a high-order countermeasure. We last use our findings to build better countermeasures by fixing and simplifying existing ones in Sec. 7.5. A discussion about possible attacks that would circumvent the assumptions of our formal model is given in Sec. 7.6. Conclusions and perspectives are drawn in Sec. 7.7. The appendices contain the detail of some secondary results.

7.2 Fault Model

The CRT-RSA cryptosystem and the BellCoRe fault injection attack have been described in Sec. 5.2 of Chap. 5. In this section, we formalize our attack model by defining what is a fault injection (we extend the our previous fault model to an even broader one) and what is the order of an attack.

Definition 7.1 (Fault injection)**.** During the execution of an algorithm, the attacker can:

- modify any intermediate value by setting it to either a *random value* (*randomizing fault*) or *zero* (*zeroing fault*); such a fault can be either *permanent* (e.g., in memory) or *transient* (e.g., in a register or a bus);
- skip any number of consecutive instructions (*skipping fault*).

At the end of the computation the attacker can read the result returned by the algorithm.

Remark 7.1. This fault injection model implies that faults can be injected very accurately in timing (the resolution is the clock period), whereas the fault locality in space is poor (the attacker cannot target a specific bit). This models an attacker who is able to identify the sequence of operations by a simple side-channel analysis, but who has no knowledge of the chip internals. Such attack model is realistic for designs where the memories are scrambled and the logic gates randomly routed (in a sea of gates).

Lemma 7.1. *The effect of a skipping fault (i.e., fault on the code) can be captured by considering only randomizing and zeroing faults (i.e., fault on the data).*

Proof. Indeed, if the skipped instructions are part of an arithmetic operation:

- either the computation has not been done at all and the value in memory where the result is supposed to be stays zero (if initialized) or random (if not),
- or the computation has partly been done and the value written in memory as its result is thus pseudo-randomized (and considered random at our modeling level).

If the skipped instruction is a branching instruction, then it is equivalent to do a zeroing fault on the result of the branching condition to make it false and thus avoid branching. □

Definition 7.2 (Attack order)**.** We call *order* of the attack the number of fault injections in the computation. An attack is said to be *high-order* if its order is strictly more than 1.

7.3 Classifying Countermeasures

The goal of a countermeasure against fault-injection attacks is to avoid returning a compromised value to the attacker. To this end, countermeasures attempt to verify the integrity of the computation before returning its result. If the integrity is compromised, then the returned value should be a random number or an error constant, in order not to leak any information.

An obvious way of achieving that goal is to repeat the computation and compare the results, but this approach is very expensive in terms of computation time. The same remark applies to the verification of the signature (notice that e can be recovered for this purpose from the 5-tuple (p, q, d_p, d_q, i_q), as explained in App. 7.A). In this section we explore the different methods used by the existing countermeasures to verify the computation integrity faster than $(M^d)^e \stackrel{?}{\equiv} M \mod N$. Besides, we recall that such a verification is prone to other attacks [YJ00], and must thus be avoided.

7.3.1 Shamir's or Giraud's Family of Countermeasures

To the authors knowledge, there are two main families of countermeasures: those which are descendants of Shamir's countermeasure [Sha99], and those which are descendants of Giraud's [Gir06].

The countermeasures in Giraud's family avoid replicating the computations using particular exponentiation algorithms. These algorithms keep track of variables involved in intermediate steps; those help verifying the consistency of the final results by a consistency check of an invariant that is supposed to be spread till the last steps. This idea is illustrated in Alg. 7.1, which resembles the one of Giraud. The test at line 5 verifies that the recombined values S and S' (recombination of intermediate steps of the exponentiation) are consistent. Example of other countermeasures in this family are the ones of Boscher et al. [BNP07], Rivain [Riv09] (and its recently improved version [LRT14]), or Kim et al. [KKHH11]. The former two mainly optimize Giraud's, while the latter introduce an infective verification based on binary masks. The detailed study of the countermeasures in Giraud's family is left as future work.

Algorithm 7.1: CRT-RSA with a Giraud's family countermeasure

Input : Message M, key (p, q, d_p, d_q, i_q)
Output: Signature $M^d \mod N$, or error

1 $(S_p, S'_p) = \mathsf{ExpAlgorithm}(M, d_p)$ `// ExpAlgorithm(a, b) returns` (a^b, a^{b-1})
2 $(S_q, S'_q) = \mathsf{ExpAlgorithm}(M, d_q)$
3 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ `// Recombination`
4 $S' = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod p)$ `// Recombination for verification`
5 **if** $M \cdot S' \not\equiv S \mod pq$ **then return** error
6 **return** S

Indeed, the rest of our chapter is mainly concerned with Shamir's family of countermeasures. The countermeasures in Shamir's family rely on a kind of "checksum" of the computation using smaller numbers (the checksum is computed in rings smaller than the ones of the actual computation). The base-two logarithm of the smaller rings cardinal is typically equal to 32 or to 64 (bits): therefore, assuming that the faults are randomly distributed, the probability of having an undetected fault is 2^{-32} or 2^{-64}, i.e., very low. In the sequel, we will make a language abuse by considering that such probability is equal to zero. We also use the following terminology:

Notation 7.1. Let a a big number and b a small number, such that they are coprime. We call the ring $\mathbb{Z}_{ab}$ an *overring* of $\mathbb{Z}_a$, and the ring $\mathbb{Z}_b$ a *subring* of $\mathbb{Z}_{ab}$.

Remark 7.2. RSA is friendly to protections by checksums because it computes in rings $\mathbb{Z}_a$ where a is either a large prime number (e.g., $a = p$ or $a = q$) or the product of large prime numbers (e.g., $a = p \cdot q$). Thus, any small number $b > 1$ is coprime with a, and so we have an isomorphism between the *overring* $\mathbb{Z}_{ab}$ and the direct product of $\mathbb{Z}_a$ and $\mathbb{Z}_b$, i.e., $\mathbb{Z}_{ab} \cong \mathbb{Z}_a \times \mathbb{Z}_b$. This means that the Chinese Remainder Theorem applies. Consequently, the nominal computation and the checksum can be conducted in parallel in $\mathbb{Z}_{ab}$.

The countermeasures attempt to assert that some invariants on the computations and the checksums hold. There are many different ways to use the checksums and to verify these invariants. In the rest of this section we review these ways while we attempt to classify countermeasures and understand better what are the necessary invariants to verify.

7.3.2 Test-Based or Infective

A first way to classify countermeasures is to separate those which consist in stepwise internal checks during the CRT computation and those which use an infective computation strategy to make the result unusable by the attacker in case of fault injection.

Definition 7.3 (Test-based countermeasure)**.** A countermeasure is said to be *test-based* if it attempts to detect fault injections by verifying that some arithmetic invariants are respected, and branch to return an error instead of the numerical result of the algorithm in case of invariant violation. Examples of test-based countermeasures are the ones of Shamir [Sha99], Aumüller et al. [ABF+02], Vigilant [Vig08a], or Joye et al. [JPY01].

Definition 7.4 (Infective countermeasure)**.** A countermeasure is said to be *infective* if rather than testing arithmetic invariants it uses them to compute a neutral element of some arithmetic operation in a way that would not result in this neutral element if the invariant is violated. It then uses the results of these computations to infect the result of the algorithm before returning it to make it unusable by the attacker (thus, it does not need branching instructions). Examples of infective countermeasures are the ones by Blömer et al. [BOS03] (and the variant by Liu et al. [LKW06]), Ciet & Joye [CJ05], or Kim et al. [KKHH11].

The extreme similarity between the verifications in the test-based countermeasure of Joye et al. [JPY01] (see Alg. 7.2, line 9) and the infective countermeasure of Ciet & Joye [CJ05] (see Alg. 7.3, lines 10 and 11) is striking, but it is actually not surprising at all, as we will discover in Prop. 7.1.

Proposition 7.1 (Equivalence between test-based and infective countermeasures)**.** *Each test-based (resp. infective) countermeasure has a direct equivalent infective (resp. test-based) countermeasure.*

Proof. The invariants that must be verified by countermeasures are modular equality, so they are of the form $a \stackrel{?}{\equiv} b \mod m$, where a, b and m are arithmetic expressions.

It is straightforward to transform this invariant into a Boolean expression usable in test-based countermeasures: `if a != b [mod m] then return` *error*.

To use it in infective countermeasures, it is as easy to verify the same invariant by computing a value which should be 1 if the invariant holds: `c := a - b + 1 mod m`. The numbers obtained this way for each invariant can then be multiplied and their product c^*, which is 1 only if all invariants are respected, can be used as an exponent on the algorithm's result to infect it if one or more of the tested invariants are violated. Indeed, when the attacker perform the BellCoRe attack by computing $\gcd(N, S - \widehat{S^{c^*}})$ as defined in Prop. 5.1, then if c^* is not 1 the attack would not work. □

Algorithm 7.2: CRT-RSA with Joye et al.'s countermeasure [JPY01]

Input : Message M, key (p, q, d_p, d_q, i_q)
Output: Signature $M^d \mod N$, or error

1 Choose two small random integers r_1 and r_2.
2 Store in memory $p' = p \cdot r_1$, $q' = q \cdot r_2$, $i'_q = q'^{-1} \mod p'$, $N = p \cdot q$.
3 $S'_p = M^{d_p \mod \varphi(p')} \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr_1}$
4 $S_{pr} = M^{d_p \mod \varphi(r_1)} \mod r_1$ // Checksum in $\mathbb{Z}_{r_1}$
5 $S'_q = M^{d_q \mod \varphi(q')} \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr_2}$
6 $S_{qr} = M^{d_q \mod \varphi(r_2)} \mod r_2$ // Checksum in $\mathbb{Z}_{r_2}$
7 $S_p = S'_p \mod p$ // Retrieve intermediate signature in $\mathbb{Z}_p$
8 $S_q = S'_q \mod q$ // Retrieve intermediate signature in $\mathbb{Z}_q$
9 **if** $S'_p \not\equiv S_{pr} \mod r_1$ **or** $S'_q \not\equiv S_{qr} \mod r_2$ **then return** error
10 **return** $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ // Recombination in $\mathbb{Z}_N$

Algorithm 7.3: CRT-RSA with Ciet & Joye's countermeasure [CJ05]

Input : Message M, key (p, q, d_p, d_q, i_q)
Output: Signature $M^d \mod N$, or a random value in $\mathbb{Z}_N$

1 Choose small random integers r_1, r_2, and r_3.
2 Choose a random integer a.
3 Initialize γ with a random number
4 Store in memory $p' = p \cdot r_1$, $q' = q \cdot r_2$, $i'_q = q'^{-1} \mod p'$, $N = p \cdot q$.
5 $S'_p = a + M^{d_p \mod \varphi(p')} \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr_1}$
6 $S_{pr} = a + M^{d_p \mod \varphi(r_1)} \mod r_1$ // Checksum in $\mathbb{Z}_{r_1}$
7 $S'_q = a + M^{d_q \mod \varphi(q')} \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr_2}$
8 $S_{qr} = a + M^{d_q \mod \varphi(r_2)} \mod r_2$ // Checksum in $\mathbb{Z}_{r_2}$
9 $S' = S'_q + q' \cdot (i'_q \cdot (S'_p - S'_q) \mod p')$ // Recombination in $\mathbb{Z}_{Nr_1r_2}$
10 $c_1 = S' - S_{pr} + 1 \mod r_1$ // Invariant for the signature modulo p
11 $c_2 = S' - S_{qr} + 1 \mod r_2$ // Invariant for the signature modulo q
12 $\gamma = (r_3 \cdot c_1 + (2^l - r_3) \cdot c_2)/2^l$ // $\gamma = 1$ if c_1 and c_2 have value 1
13 **return** $S = S' - a^\gamma \mod N$ // Infection and result retrieval in $\mathbb{Z}_N$

By Prop. 7.1, we know that there is an equivalence between test-based and infective countermeasures. This means that in theory any attack working on one kind of countermeasure will be possible on the equivalent countermeasure of the other kind. However, we remark that in practice it is harder to do a zeroing fault on an intermediate value (especially if it is the result of a computation with big numbers) in the case of an infective countermeasure, than it is to skip one branching instruction in the case of a test-based countermeasure. We conclude from this the following rule of thumb: *it is better to use the infective variant of a countermeasure.* In addition, it is generally the case that code without branches is safer (think of timing attacks or branch predictor attacks on modern CPUs).

Note that if a fault occurs, c^* is not 1 anymore and thus the computation time required to compute S^{c^*} might significantly increase. This is not a security problem,

indeed, taking longer to return a randomized value in case of an attack is not different from rapidly returning an error constant without finishing the computation first as it is done in the existing test-based countermeasures. In the worst case scenario, the additional time would be correlated to the induced fault, but we assume the fault to be controlled by the attacker already.

7.3.3 Intended Order

Countermeasures can be classified depending on their order, i.e., the maximum order of the attacks (as per Def. 7.2) that they can protect against.

In the literature concerning CRT-RSA countermeasures against fault-injection attacks, most countermeasures claim to be first-order, and a few claim second-order resistance. For instance, the countermeasures by Aumüller et al. [ABF+02] and the one by Vigilant [Vig08a] are described as first-order by their authors, while Ciet & Joye [CJ05] describe a second-order fault model and propose a countermeasure which is supposed to resist to this fault model, and thus be second-order.

However, using the finja[1] tool which has been open-sourced by Rauzy & Guilley [RG14a] (See App. B.2), we found out that the countermeasure of Ciet & Joye is in fact vulnerable to second-order attacks (in our fault model of Def. 7.1). This is not very surprising. Indeed, Prop. 7.1 proves that injecting a fault, and then skipping the invariant verification which was supposed to catch the first fault injection, is a second-order attack strategy which also works for infective countermeasures, except the branching-instruction skip has to be replaced by a zeroing fault. As expected, the attacks we found using finja did exactly that. For instance a zeroing fault on S'_p (resp. S'_q) makes the computation vulnerable to the BellCoRe attack, and a following zeroing fault on S_{pr} (resp. S_{qr}) makes the verification pass anyway. To our knowledge our attack is new. It is indeed different from the one Dottax et al. [DGRS09] found and fixed in their paper, which was an attack on the use of γ (see line 12 of Alg. 7.3). It is true that their attack model only allows skipping faults (as per Def. 7.1) for the second injection, but we have concerns about this:

- What justifies this limitation on the second fault? Surely if the attackers are able to inject two faults and can inject a zeroing fault once they can do it twice.
- Even considering their attack model, a zeroing fault on an intermediate variable x can in many cases be obtained by skipping the instructions where the writing to x happens.
- The fixed version of the countermeasure by Dottax et al. [DGRS09, Alg. 8, p. 13] makes it even closer to the one of Joye et al. by removing the use of a and γ. It also removes the result infection part and instead returns S along with values that should be equal if no faults were injected, leaving "out" of the algorithm the necessary comparison and branching instructions which are presented in a separate procedure [DGRS09, Proc. 1, p. 11]. The resulting countermeasure is second-order resistant (in their attack model) only because the separate procedure does the necessary tests twice (it would indeed break at third-order unless an additional repetition of the test is added, etc.).

[1] http://pablo.rauzy.name/sensi/finja.html (we used the commit 782384a version of the code).

An additional remark would be that the algorithms of intended second-order countermeasures does not look very different from others. Moreover, Rauzy & Guilley [RG14a, RG14b] exposed evidence that the intendedly first-order countermeasures of Aumüller et al. and Vigilant actually offer the same level of resistance against second-order attacks, i.e., they resist when the second injected fault is a randomizing fault (or a skipping fault which amounts to a randomizing fault).

7.3.4 Usage of the Small Rings

In most countermeasures, the computation of the two intermediate signatures modulo p and modulo q of the CRT actually takes place in overrings. The computation of S_p (resp. S_q) is done in $\mathbb{Z}_{pr_1}$ (resp. $\mathbb{Z}_{qr_2}$) for some small random number r_1 (resp. r_2) rather than in $\mathbb{Z}_p$ (resp. $\mathbb{Z}_q$). This allows the retrieval of the results by reducing modulo p (resp. q) and verifying the signature modulo r_1 (resp. r_2), or, if it is done after the CRT recombination, the results can be retrieved by reducing modulo $N = p \cdot q$. The reduction in the *small subrings* $\mathbb{Z}_{r_1}$ and $\mathbb{Z}_{r_2}$ is used as the checksums for verifying the integrity of the computation. It works because small random numbers are necessarily coprime with a big prime number.

An interesting part of countermeasures is how they use the small subrings to verify the integrity of the computations. Almost all the various countermeasures we studied had different ways of using them. However, they can be divided in two groups. On one side there are countermeasures which use the small subrings to verify the integrity of the intermediate CRT signatures and of the recombination directly but using smaller numbers, like Blömer et al.'s countermeasure [BOS03], or Ciet & Joye's one [CJ05]. On the other side, there are countermeasures which use some additional arithmetic properties to verify the necessary invariants indirectly in the small subrings. Contrary to the countermeasures in the first group, the ones in the second group use the same value r for r_1 and r_2. The symmetry obtained with $r_1 = r_2$ is what makes the additional arithmetic properties hold, as we will see.

Verification of the Intermediate CRT Signatures

The countermeasure of Blömer et al. [BOS03] uses the small subrings to verify the intermediate CRT signatures. It is exposed in Alg. 7.4. This countermeasure needs access to d directly rather than d_p and d_q as the standard interface for CRT-RSA suggests, in order to compute $d'_p = d \mod \varphi(p \cdot r_1)$ and $d'_q = d \mod \varphi(q \cdot r_2)$, as well as their inverse $e'_p = {d'_p}^{-1} \mod \varphi(p \cdot r_1)$ and $e'_q = {d'_q}^{-1} \mod \varphi(q \cdot r_2)$ to verify the intermediate CRT signatures.

We can see in Alg. 7.4 that these verifications (lines 6 and 7) happen after the recombination (line 5) and retrieve the checksums in $\mathbb{Z}_{r_1}$ (for the p part of the CRT) and $\mathbb{Z}_{r_2}$ (for the q part) from the recombined value S'. It allows these tests to verify the integrity of the recombination at the same time as they verify the integrity of the intermediate CRT signatures.

Checksums of the Intermediate CRT Signatures

The countermeasure of Ciet & Joye [CJ05] uses the small subrings to compute checksums of the intermediate CRT signatures. It is exposed in Alg. 7.3. Just as the previous one, the verifications (lines 10 and 11) take place after the recombination

Algorithm 7.4: CRT-RSA with Blömer et al.'s countermeasure [BOS03]

Input : Message M, key (p, q, d, i_q)
Output: Signature $M^d \mod N$, or a random value in $\mathbb{Z}_N$

1 Choose two small random integers r_1 and r_2.
2 Store in memory $p' = p \cdot r_1$, $q' = q \cdot r_2$, $i'_q = q'^{-1} \mod p'$, $N = p \cdot q$, $N' = N \cdot r_1 \cdot r_2$, d'_p, d'_q, e'_p, e'_q.
3 $S'_p = M^{d'_p} \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr_1}$
4 $S'_q = M^{d'_q} \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr_2}$
5 $S' = S'_q + q' \cdot (i'_q \cdot (S'_p - S'_q) \mod p')$ // Recombination in $\mathbb{Z}_{Nr_1r_2}$
6 $c_1 = M - S'^{e'_p} + 1 \mod r_1$ // Invariant for the signature modulo p
7 $c_2 = M - S'^{e'_q} + 1 \mod r_2$ // Invariant for the signature modulo q
8 **return** $S = S'^{c_1 c_2} \mod N$ // Infection and result retrieval in $\mathbb{Z}_N$

(line 9) and retrieve the checksums in $\mathbb{Z}_{r_1}$ (for the p part of the CRT) and $\mathbb{Z}_{r_2}$ (for the q part) from the recombined value S', which enables the integrity verification of the recombination at the same time as the integrity verifications of the intermediate CRT signatures.

We note that this is missing from the protection of Joye et al. [JPY01], presented in Alg. 7.2, which does not verify the integrity of the recombination at all and is thus as broken as Shamir's countermeasure [Sha99]. The countermeasure of Ciet & Joye is a clever fix against the possible fault attacks on the recombination of Joye et al.'s countermeasure, which also uses the transformation that we described in Prop. 7.1 from a test-based to an infective countermeasure.

Overrings for CRT Recombination

In Ciet & Joye's countermeasure the CRT recombination happens in an overring $\mathbb{Z}_{Nr_1r_2}$ of $\mathbb{Z}_N$ while Joye et al.'s countermeasure extracts in $\mathbb{Z}_p$ and $\mathbb{Z}_q$ the results S_p and S_q of the intermediate CRT signatures to do the recombination in $\mathbb{Z}_N$ directly.

There are only two other countermeasures which do the recombination in $\mathbb{Z}_N$ that we know of: the one of Shamir [Sha99] and the one of Aumüller et al. [ABF+02]. The first one is known to be broken, in particular because it does not check whether the recombination has been faulted at all. The second one seems to need to verify 5 invariants to resist the BellCoRe attack[2], which is more than the only 2 required by the countermeasure of Ciet & Joye [CJ05] or by the one of Blömer et al. [BOS03], while offering a similar level of protection (see [RG14a]). This fact led us to think that the additional tests are necessary because the recombination takes place "in the clear". But we did not jump right away to that conclusion. Indeed, Vigilant's countermeasure [Vig08a] does the CRT recombination in the $\mathbb{Z}_{Nr^2}$ overring of $\mathbb{Z}_N$ and seems to require 7 verifications[3] to also offer that same level of security (see [RG14b]). However, we remark that Shamir's, Aumüller et al.'s, and Vigilant's

[2]The original Aumüller et al.'s countermeasure uses 7 verifications because it also needs to check the integrity of intermediate values introduced against simple power analysis, see [RG14a, Remark 1].

[3]Vigilant's original countermeasure and its corrected version by Coron et al. [CGM+10] actually use 9 verifications but were simplified by Rauzy & Guilley [RG14b] who removed 2 verifications.

countermeasures use the same value for r_1 and r_2.

Identity of r_1 and r_2

Some countermeasures such as the ones of Shamir [Sha99], Aumüller et al. [ABF+02], and Vigilant [Vig08a] use a single random number r to construct the overrings used for the two intermediate CRT signatures computation. The resulting symmetry allows these countermeasures to take advantage of some additional arithmetic properties.

Shamir's countermeasure. In his countermeasure, which is presented in Alg. 7.5, Shamir uses a clever invariant property to verify the integrity of both intermediate CRT signatures in a single verification step (line 9). This is made possible by the fact that he uses d directly instead of d_p and d_q, and thus the checksums in $\mathbb{Z}_r$ of both the intermediate CRT signatures are supposed to be equal if no fault occurred. Unfortunately, the integrity of the recombination is not verified at all. We will see in Sec. 7.5.1 how to fix this omission. Besides, we notice that d can be reconstructed from a usual CRT-RSA key (p, q, d_p, d_q, i_q); we refer the reader to Appendix 7.A.

Algorithm 7.5: CRT-RSA with Shamir's countermeasure [Sha99]

Input : Message M, key (p, q, d, i_q)
Output: Signature $M^d \mod N$, or error

1 Choose a small random integer r.
2 $p' = p \cdot r$
3 $S'_p = M^{d \mod \varphi(p')} \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr}$
4 $q' = q \cdot r$
5 $S'_q = M^{d \mod \varphi(q')} \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr}$
6 $S_p = S'_p \mod p$ // Retrieve intermediate signature in $\mathbb{Z}_p$
7 $S_q = S'_q \mod q$ // Retrieve intermediate signature in $\mathbb{Z}_q$
8 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ // Recombination in $\mathbb{Z}_N$
9 **if** $S'_p \not\equiv S'_q \mod r$ **then return** error
10 **return** S

Aumüller et al.'s countermeasure. Contrary to Shamir, Aumüller et al. do verify the integrity of the recombination in their countermeasure, which is presented in Alg. 7.6. To do this, they straightforwardly check (line 10) that when reducing the result S of the recombination modulo p (resp. q), the obtained value corresponds to the intermediate signature in $\mathbb{Z}_p$ (resp. $\mathbb{Z}_q$). However, they do not use d directly but rather conform to the standard CRT-RSA interface by using d_p and d_q. Thus, they need another verification to check the integrity of the intermediate CRT signatures. Their clever strategy is to verify that the checksums of S_p and S_q in $\mathbb{Z}_r$ are conform to each other (lines 11 to 13). For that they check whether ${S_p}^{d_q}$ is equal to ${S_q}^{d_p}$ in $\mathbb{Z}_r$, that is, whether the invariant $(M^{d_p})^{d_q} \equiv (M^{d_q})^{d_p} \mod r$ holds.

The two additional tests on line 4 verify the integrity of p' and q'. Indeed, if p or q happen to be randomized when computing p' or q' the invariant verifications in $\mathbb{Z}_r$

Algorithm 7.6: CRT-RSA with Aumüller et al.'s countermeasure[4] [ABF+02]

Input : Message M, key (p, q, d_p, d_q, i_q)
Output: Signature $M^d \mod N$, or error

1 Choose a small random integer r.
2 $p' = p \cdot r$
3 $q' = q \cdot r$
4 **if** $p' \not\equiv 0 \mod p$ **or** $q' \not\equiv 0 \mod q$ **then return** error
5 $S'_p = M^{d_p \mod \varphi(p')} \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr}$
6 $S'_q = M^{d_q \mod \varphi(q')} \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr}$
7 $S_p = S'_p \mod p$ // Retrieve intermediate signature in $\mathbb{Z}_p$
8 $S_q = S'_q \mod q$ // Retrieve intermediate signature in $\mathbb{Z}_q$
9 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ // Recombination in $\mathbb{Z}_N$
10 **if** $S \not\equiv S'_p \mod p$ **or** $S \not\equiv S'_q \mod q$ **then return** error
11 $S_{pr} = S'_p \mod r$ // Checksum of S_p in $\mathbb{Z}_r$
12 $S_{qr} = S'_q \mod r$ // Checksum of S_q in $\mathbb{Z}_r$
13 **if** ${S_{pr}}^{d_q \mod \varphi(r)} \not\equiv {S_{qr}}^{d_p \mod \varphi(r)} \mod r$ **then return** error
14 **return** S

would pass but the retrieval of the intermediate signatures in $\mathbb{Z}_p$ or $\mathbb{Z}_q$ would return random values, which would make the BellCoRe attack work. These important verifications are missing from all the previous countermeasures in Shamir's family.

Vigilant's countermeasure. Vigilant takes another approach. Rather than doing the integrity verifications on "direct checksums" that are the representative values of the CRT-RSA computation in the small subrings, Vigilant uses different values that he constructs for that purpose. The clever idea of his countermeasure is to use sub-CRTs on the values that the CRT-RSA algorithm manipulates in order to have in one part the value we are interested in and in the other the value constructed for the verification (lines 8 and 17).

To do this, he transforms M into another value M' such that:

$$M' \equiv \begin{cases} M & \mod N, \\ 1 + r & \mod r^2, \end{cases}$$

which implies that:

$$S' = M'^d \mod Nr^2 \equiv \begin{cases} M^d & \mod N, \\ 1 + dr & \mod r^2. \end{cases}$$

The latter results are based on the binomial theorem, which states that $(1+r)^d = \sum_{k=0}^{d} \binom{d}{k} r^k = 1 + dr + \binom{d}{2} r^2 + \ldots$, which simplifies to $1 + dr$ in the $\mathbb{Z}_{r^2}$ ring.

This property is used to verify the integrity of the intermediate CRT signatures on lines 11 and 20. It is also used on line 24 which tests the recombination using the same technique but with random values inserted on lines 21 and 22 in place of the constructed ones. This test also verifies the integrity of N.

Algorithm 7.7: CRT-RSA with Vigilant's countermeasure[4] [Vig08a]
with Coron et al.'s fixes [CGM+10] and Rauzy & Guilley's simplifications [RG14b]

Input : Message M, key (p, q, d_p, d_q, i_q)
Output: Signature $M^d \mod N$, or error

1 Choose small random integers r, R_1, and R_2.
2 $N = p \cdot q$
3 $p' = p \cdot r^2$
4 $i_{pr} = p^{-1} \mod r^2$
5 $M_p = M \mod p'$
6 $B_p = p \cdot i_{pr}$
7 $A_p = 1 - B_p \mod p'$
8 $M'_p = A_p \cdot M_p + B_p \cdot (1 + r) \mod p'$ // CRT embedding in M'_p
9 $S'_p = {M'_p}^{d_p \mod \varphi(p')} \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr^2}$
10 **if** $M'_p \not\equiv M \mod p$ **then return** error
11 **if** $B_p \cdot S'_p \not\equiv B_p \cdot (1 + d_p \cdot r) \mod p'$ **then return** error
12 $q' = q \cdot r^2$
13 $i_{qr} = q^{-1} \mod r^2$
14 $M_q = M \mod q'$
15 $B_q = q \cdot i_{qr}$
16 $A_q = 1 - B_q \mod q'$
17 $M'_q = A_q \cdot M_q + B_q \cdot (1 + r) \mod q'$ // CRT embedding in M'_q
18 $S'_q = {M'_q}^{d_q \mod \varphi(q')} \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr^2}$
19 **if** $M'_q \not\equiv M \mod q$ **then return** error
20 **if** $B_q \cdot S'_q \not\equiv B_q \cdot (1 + d_q \cdot r) \mod q'$ **then return** error
21 $S_{pr} = S'_p - B_p \cdot (1 + d_p \cdot r - R_1)$ // Verification value of S'_p swapped with R_1
22 $S_{qr} = S'_q - B_q \cdot (1 + d_q \cdot r - R_2)$ // Verification value of S'_q swapped with R_2
23 $S_r = S_{qr} + q \cdot (i_q \cdot (S_{pr} - S_{qr}) \mod p')$ // Recombination in $\mathbb{Z}_{Nr^2}$
// Simultaneous verification of lines 2 and 23
24 **if** $pq \cdot (S_r - R_2 - q \cdot i_q \cdot (R_1 - R_2)) \not\equiv 0 \mod Nr^2$ **then return** error
25 **return** $S = S_r \mod N$ // Retrieve result in $\mathbb{Z}_N$

Two additional tests are required by Vigilant's arithmetic trick. The verifications at lines 10 and 19 ensure that the original message M has indeed been CRT-embedded in M'_p and M'_q.

7.4 The Essence of a Countermeasure

Our attempt to classify the existing countermeasures provided us with a deep understanding of how they work. To ensure the integrity of the CRT-RSA computation,

[4] For the sake of simplicity we removed some code that served against SPA (simple power analysis) and only kept the necessary code against fault-injection attacks.

the algorithm must verify 3 things: the integrity of the computation modulo p, the integrity of the computation modulo q, and the integrity of the CRT recombination (which can be subject to transient fault attacks). This fact has been known since the first attacks on Shamir's countermeasure. Our study of the existing countermeasures revealed that, as expected, those which perform these three integrity verifications are the ones which actually work. This applies to Shamir's family of countermeasures, but also for Giraud's family. Indeed, countermeasures in the latter also verify the two exponentiations and the recombination by testing the consistency of the exponentiations indirectly on the recombined value.

7.4.1 A Straightforward Countermeasure

The result of these observations is a very straightforward countermeasure, presented in Alg. 7.8. This countermeasure works by testing the integrity of the signatures modulo p and q by replicating the computations (lines 1 and 3) and comparing the results, and the integrity of the recombination by verifying that the two parts of the CRT can be retrieved from the final result (line 5). This countermeasure is of course very expensive since the two big exponentiations are done twice, and is thus not usable in practice. Note that it is nonetheless still better in terms of speed than computing RSA without the CRT optimization.

Algorithm 7.8: CRT-RSA with straightforward countermeasure

Input : Message M, key (p, q, d_p, d_q, i_q)
Output: Signature $M^d \mod N$, or error

1 $S_p = M^{d_p \mod \varphi(p)} \mod p$ // Intermediate signature in $\mathbb{Z}_p$
2 **if** $S_p \not\equiv M^{d_p} \mod p$ **then return** error
3 $S_q = M^{d_q \mod \varphi(q)} \mod q$ // Intermediate signature in $\mathbb{Z}_q$
4 **if** $S_q \not\equiv M^{d_q} \mod q$ **then return** error
5 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ // Recombination in $\mathbb{Z}_N$
6 **if** $S \not\equiv S_p \mod p$ **or** $S \not\equiv S_q \mod q$ **then return** error
7 **return** S

Proposition 7.2 (Correctness)**.** *The straightforward countermeasure (and thus all the ones which do equivalent verifications) is secure against first-order fault attacks as per Def. 7.1 and 7.2.*

Proof. The proof is in two steps. First, prove that if the intermediate signatures are not correct, then the tests at lines 2 and 4 returns error. Second, prove that if both tests passed then either the recombination is correct or the test at line 6 returns error.

If a fault occurs during the computation of S_p (line 1), then it either has the effect of zeroing its value or randomizing it, as shown by Lem. 7.1. Thus, the test of line 2 detects it since the two compared values won't be equal. If the fault happens on line 2, then either we are in a symmetrical case: the repeated computation is faulted, or the test is skipped: in that case there are no faults affecting the data so the test is unnecessary anyway. It works similarly for the intermediate signature in $\mathbb{Z}_q$.

If the first two tests pass, then the tests at line 6 verify that both parts of the CRT computation are indeed correctly recombined in S. If a fault occurs during the recombination on line 5 it will thus be detected. If the fault happens at line 6, then either it is a fault on the data and one of the two tests returns error, or it is a skipping fault which bypasses one or both tests but in that case there are no faults affecting the data so the tests are unnecessary anyway. □

7.4.2 High-Order Countermeasures

Using the finja[1] tool we were able to verify that removing one of the three integrity checks indeed breaks the countermeasure against first-order attacks. Nonetheless, each countermeasure which has these three integrity checks, plus those that may be necessary to protect optimizations on them, offers the same level of protection.

Proposition 7.3 (High-order countermeasures)**.** *Against randomizing faults, all correct countermeasures (as per Prop. 7.2) are high-order. However, there are no generic high-order countermeasures if the three types of faults in our attack model are taken into account, but it is possible to build nth-order countermeasures for any n.*

Proof. Indeed, if a countermeasure is able to detect a single randomizing fault, then adding more faults will not break the countermeasure, since a random fault cannot induce a verification skip. Thus, *all working countermeasures are high-order against randomizing faults.*

However, if after one or more faults which permit an attack, there is a skipping fault or a zeroing fault which leads to skip the verification which would detect the previous fault injections, then the attack will work. As Lem. 7.1 and Prop. 7.1 explain, this is true for all countermeasures, not only those which are test-based but also the infective ones. It seems that the only way to protect against that is to replicate of the integrity checks. If each invariant is verified n times, then the countermeasure will resist at least n faults in the worst case scenario: a single fault is used to break the computation and the n others to avoid the verifications which detect the effect of the first fault. Thus, *there are no generic high-order countermeasures* if the three types of faults in our attack model are taken into account, ***but it is possible to build a*** n**th*-order countermeasure for any*** n **by replicating the invariant verifications** n **times**. □

Existing first-order countermeasures such as the ones of Aumüller et al. (Alg. 7.6 and 7.13), Vigilant (Alg. 7.7 and 7.11), or Ciet & Joye (Alg. 7.3) can thus be transformed into nth-order countermeasures, in the attack model described in Def. 7.1 and 7.2. As explained, the transformation consists in replicating the verifications n times, whether they are test-based or infective. An example "macro" generating the code for a countermeasure of order D based on Vigilant's countermeasure is presented in Alg. 7.9.

This result means that it is very important that the verifications be cost effective. Fortunately, as we saw in Sec. 7.3 and particularly in Sec. 7.3.4 on the usage of the small rings, the existing countermeasures offer exactly that: optimized versions of Alg. 7.8 that use a variety of invariant properties to avoid replicating the two big exponentiations of the CRT computation.

Algorithm 7.9: Generation of CRT-RSA Vigilant's countermeasure of order D

Input : order D
Output: CRT-RSA algorithm protected against fault injection attack of order D

print Choose a small random integer r.
print $N = p \cdot q$
print $p' = p \cdot r^2$
print $i_{pr} = p^{-1} \bmod r^2$
print $M_p = M \bmod p'$
print $B_p = p \cdot i_{pr}$
print $A_p = 1 - B_p \bmod p'$
print $M'_p = A_p \cdot M_p + B_p \cdot (1 + r) \bmod p'$
print $q' = q \cdot r^2$
print $i_{qr} = q^{-1} \bmod r^2$
print $M_q = M \bmod q'$
print $B_q = q \cdot i_{qr}$
print $A_q = 1 - B_q \bmod q'$
print $M'_q = A_q \cdot M_q + B_q \cdot (1 + r) \bmod q'$
print $S'_p = {M'_p}^{d_p \bmod \varphi(p')} \bmod p'$
print $S'_q = {M'_q}^{d_q \bmod \varphi(q')} \bmod q'$
print $S_{pr} = 1 + d_p \cdot r$
print $S_{qr} = 1 + d_q \cdot r$
print $S_r = S_{qr} + q \cdot (i_q \cdot (S_{pr} - S_{qr}) \bmod p')$
print $S' = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \bmod p')$
print $S_0 = S' \bmod N$
for $i \leftarrow 1$ ***to*** D **do**
  print if $M'_p + N \not\equiv M \mod p$ **then** S; **print** $_{i-1}$; **print** $= 0$
  print S'; **print** $_i$ **print** $= S$; **print** $_{i-1}$
  print if $M'_q + N \not\equiv M \mod q$ **then** S'; **print** $_i$; **print** $= 0$
  print S''; **print** $_i$ **print** $= S'$; **print** $_i$
  print if $S \not\equiv S_r \mod r^2$ **then** S''; **print** $_i$; **print** $= 0$
  print S; **print** $_i$ **print** $= S''$; **print** $_i$
end
print return S; **print** $_D$

7.5 Building Better Countermeasures

In the two previous sections we learned a lot about current countermeasures and how they work. We saw that to reduce their cost, most countermeasures use invariant properties to optimize the verification speed by using checksums on smaller numbers than the big ones which are manipulated by the protected algorithm. Doing so, we understood how these optimizations work and the power of their underlying ideas. In this section apply our newly acquired knowledge on the *essence of countermeasures* in order to build the *quintessence of countermeasures*. Namely, we leverage our findings to fix Shamir's countermeasure, and to drastically simplify the one of Vigilant, while at the same time transforming it to be infective instead of test-based.

7.5.1 Correcting Shamir's Countermeasure

We saw that Shamir's countermeasure is broken in multiple ways, which has been known for a long time now. To fix it without denaturing it, we need to verify the integrity of the recombination as well as the ones of the overrings moduli. We can directly take these verifications from Aumüller et al.'s countermeasure. The result can be observed in Alg. 7.10.

Algorithm 7.10: CRT-RSA with a fixed version of Shamir's countermeasure (new algorithm contributed in this chapter)

Input : Message M, key (p, q, d, i_q)
Output: Signature $M^d \mod N$, or error

1 Choose a small random integer r.
2 $p' = p \cdot r$
3 $q' = q \cdot r$
4 **if** $p' \not\equiv 0 \mod p$ **or** $q' \not\equiv 0 \mod q$ **then return** error
5 $S'_p = M^{d \mod \varphi(p')} \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr}$
6 $S'_q = M^{d \mod \varphi(q')} \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr}$
7 **if** $S'_p \not\equiv S'_q \mod r$ **then return** error
8 $S_p = S'_p \mod p$ // Retrieve intermediate signature in $\mathbb{Z}_p$
9 $S_q = S'_q \mod q$ // Retrieve intermediate signature in $\mathbb{Z}_q$
10 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ // Recombination in $\mathbb{Z}_N$
11 **if** $S \not\equiv S'_p \mod p$ **or** $S \not\equiv S'_q \mod q$ **then return** error
12 **return** S

The additional tests on line 4 protect against transient faults on p (resp. q) while computing p' (resp. q'), which would amount to a randomization of S'_p (resp. S'_q) while computing the intermediate signatures. The additional test on line 7 verifies the integrity of the intermediate signature computations.

7.5.2 Simplifying Vigilant's Countermeasure

The mathematical tricks used in the Vigilant countermeasure are very powerful. Their understanding enabled the optimization of his countermeasure to only need 3 verifications, while the original version has 9. Our simplified version of the countermeasure can be seen in Alg. 7.11. Our idea is that it is not necessary to perform the checksum value replacements at lines 21 and 22 of Alg. 7.7 (see Sec. 7.3.4). What is more, if these replacements are not done, then the algorithm's computations carry the CRT-embedded checksum values until the end, and *the integrity of the whole computation can be tested with a single verification* in $\mathbb{Z}_{r^2}$ (line 23 of Alg. 7.11).

This idea not only reduces the number of required verifications, which is in itself a security improvement as shown in Sec. 7.3.2, but it also optimizes the countermeasure for speed and reduces its need for randomness (the computations of lines 21 and 22 of Alg. 7.7 are removed).

The two other tests that are left are the ones of lines 10 and 19 in Alg. 7.7, which ensure that the original message M has indeed been CRT-embedded in M'_p and M'_q.

Algorithm 7.11: CRT-RSA with our simplified Vigilant's countermeasure, under its infective avatar
(new algorithm contributed in this chapter)

Input : Message M, key (p, q, d_p, d_q, i_q)
Output: Signature $M^d \mod N$, or a random value in $\mathbb{Z}_N$

1 Choose a small random integer r.
2 $N = p \cdot q$
3 $p' = p \cdot r^2$
4 $i_{pr} = p^{-1} \mod r^2$
5 $M_p = M \mod p'$
6 $B_p = p \cdot i_{pr}$
7 $A_p = 1 - B_p \mod p'$
8 $M'_p = A_p \cdot M_p + B_p \cdot (1 + r) \mod p'$ // CRT embedding in M'_p
9 $q' = q \cdot r^2$
10 $i_{qr} = q^{-1} \mod r^2$
11 $M_q = M \mod q'$
12 $B_q = q \cdot i_{qr}$
13 $A_q = 1 - B_q \mod q'$
14 $M'_q = A_q \cdot M_q + B_q \cdot (1 + r) \mod q'$ // CRT embedding in M'_q
15 $S'_p = {M'_p}^{d_p \mod \varphi(p')} \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr^2}$
16 $S_{pr} = 1 + d_p \cdot r$ // Checksum in $\mathbb{Z}_{r^2}$ for S'_p
17 $c_p = M'_p + N - M + 1 \mod p$
18 $S'_q = {M'_q}^{d_q \mod \varphi(q')} \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr^2}$
19 $S_{qr} = 1 + d_q \cdot r$ // Checksum in $\mathbb{Z}_{r^2}$ for S'_q
20 $c_q = M'_q + N - M + 1 \mod q$
21 $S' = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod p')$ // Recombination in $\mathbb{Z}_{Nr^2}$
22 $S_r = S_{qr} + q \cdot (i_q \cdot (S_{pr} - S_{qr}) \mod p')$ // Recombination checksum in $\mathbb{Z}_{r^2}$
23 $c_S = S' - S_r + 1 \mod r^2$
24 **return** $S = S'^{c_p c_q c_S} \mod N$ // Retrieve result in $\mathbb{Z}_N$

We take advantage of these two tests to verify the integrity of N both modulo p and modulo q (lines 17 and 20 of Alg. 7.11).

Remark 7.3. Note that we also made this version of the countermeasure infective, using the transformation method that we exposed in Sec. 7.3.2. As we said, any countermeasure can be transformed this way, for instance Alg. 7.13 in the Appendix 7.B presents an infective variant of Aumüller et al.'s countermeasure.

7.6 Discussion: Limitations of our Formalism

Some CRT-RSA implementations that are proved secure in this chapter have been subject to published attacks. Such attacks actually exploit artifacts not captured by our modelization.

We mention below two of them, which are quite insightful about the difficulty to formally capture a security notion:

- one where the inputs are correlated (Vigilant, in [Vig08b, Slides 13–14]), which allows to make the redundant verifications ineffective;
- one where the effect of the fault is considered small enough to enumerate all of them, coupled with an inversion of an infective protection.

Of course, it is difficult to resist an omnipotent attacker. Despite our proof framework's ability to adapt to different attacks (by using different *attack success condition*), it still only captures the fault model we described in Def. 7.1 and 7.2, which may not always match reality. We nevertheless notice that the use of formal methods helps guarantee a baseline security level.

7.6.1 Correlated Inputs

We suppose that the inputs can be chosen by the attacker[5] and that the redundancy parameter r is known[6]. In this case, Vigilant has shown in his slides at CHES '08 that the countermeasure of Aumüller et al. [ABF+02] can be defeated as if unprotected by using for input m a multiple of r. Indeed, all the computations happening in the overrings $\mathbb{Z}_{pr}$ or $\mathbb{Z}_{qr}$ are then equal to zero modulo r, and thus alterations of the control flow graph remain undetected.

7.6.2 Low Entropy Infection

In [Wag04], Wagner attacks an infective protection, namely BOS [BOS03]. By assuming that the error is not uniformly distributed, but rather of low Hamming weight, he shows how to exploit the infected output by an exhaustive enumeration of plausible faults. Clearly, this is a drawback of infective protections. We can ignore it provided two hypotheses are formulated:

1. we assume that the mixing between the result and the fault-dependent quantity (denoted checks c_i, e.g., in Alg. 7.4) is one-way, i.e., the non-infected value cannot be recovered from the infected value and exhaustive guesses on the c_i, and/or
2. we assume that the attacker cannot accurately choose his fault model.

7.7 Conclusions and Perspectives

We studied the existing CRT-RSA algorithm countermeasures against fault-injection attacks, in particular the ones of Shamir's family. In so doing, we got a deeper understanding of their ins and outs. We obtained a few intermediate results: the absence of conceptual distinction between test-based and infective countermeasures, the fact that faults on the code (skipping instructions) can be captured by considering only faults on the data, and the fact that the many countermeasures that we studied (and their variations) were actually applying a common protection strategy but optimized it in different ways. These intermediate results allowed us to describe the design of a high-order countermeasure against our very generic fault model (comprised of randomizing, zeroing, and skipping faults). Our design allows to build a countermeasure resisting n faults for any n at a very reduced cost (it consists in adding

[5] Chosen plaintext is an example of threat that is not captured by our model.

[6] In practice, r can be chosen randomly for each execution, thus mitigating this attack.

$n-1$ comparisons on small numbers). We were also able to fix Shamir's countermeasure, and to drastically improve the one of Vigilant, going from 9 verifications in the original countermeasure to only 3, removing computations made useless, and reducing its need for randomness, while at the same time making it infective instead of test-based.

Except for those which rely on the fact that the protected algorithm takes the form of a CRT computation, the ideas presented in the various countermeasures can be applied to any modular arithmetic computation. For instance, it could be done using the idea of Vigilant consisting in using the CRT to embed a known subring value in the manipulated numbers to serve as a checksum. That would be the most obvious perspective for future work, as it would allow a generic approach against fault attacks and even automatic insertion of the countermeasure.

A study of Giraud's family of countermeasures in more detail would be beneficial to the community as well.

Acknowledgment

We would like to thank Antoine Amarilli for his proofreading which greatly improved the editorial quality of our manuscript.

7.A Recovering d and e from (p, q, d_p, d_q, i_q)

We prove here the following proposition:

Proposition 7.4. *It is possible to recover the private exponent d and the public exponent e from the 5-tuple (p, q, d_p, d_q, i_q) described in Sec. 5.2.2.*

Proof. Clearly, $p-1$ and $q-1$ are neither prime, nor coprimes (they have at least 2 as a common factor). Thus, proving Prop. 7.4 is not a trivial application of the Chinese Remainder Theorem. The proof we provide is elementary, but to our best knowledge, it has never been published before.

The numbers $p_1 = \frac{p-1}{\gcd(p-1,q-1)}$ and $q_1 = \frac{q-1}{\gcd(p-1,q-1)}$ are coprime, but there product is not equal to $\lambda(N)$. There is a factor $\gcd(p-1, q-1)$ missing, since $\lambda(N) = p_1 \cdot q_1 \cdot \gcd(p-1, q-1)$.

Now, $\gcd(p-1, q-1)$ is expected to be small. Thus, the following Alg. 7.12 can be applied efficiently. In this algorithm, the invariant is that p_2 and q_2, initially equal to p_1 and p_2, remain coprime. Moreover, they keep on increasing whereas r_2, initialized to $r_1 = \gcd(p-1, q-1)$, keeps on decreasing till 1.

Algorithm 7.12: Factorization of $\lambda(N)$ into two coprimes, multiples of p_1 and q_1 respectively

Input : $p_1 = \frac{p-1}{\gcd(p-1,q-1)}$, $q_1 = \frac{q-1}{\gcd(p-1,q-1)}$ and $r_1 = \gcd(p-1, q-1)$
Output: (p_2, q_2), coprime, such as $p_2 \cdot q_2 = \lambda(N)$

1 $(p_2, q_2, r_2) \leftarrow (p_1, q_1, r_1)$
2 $g \leftarrow \gcd(p_2, r_2)$
3 **while** $g \neq 1$ **do**
4 $\quad p_2 \leftarrow p_2 \cdot g$
5 $\quad r_2 \leftarrow r_2 / g$
6 $\quad g \leftarrow \gcd(p_2, r_2)$
7 **end**
8 $g \leftarrow \gcd(q_2, r_2)$
9 **while** $g \neq 1$ **do**
10 $\quad q_2 \leftarrow q_2 \cdot g$
11 $\quad r_2 \leftarrow r_2 / g$
12 $\quad g \leftarrow \gcd(q_2, r_2)$
13 **end**
// p_2, q_2 and r_2 are now coprime
14 $q_2 \leftarrow q_2 \cdot r_2$ // $p_2 \leftarrow p_2 \cdot r_2$ would work equally
15 $(r_2 \leftarrow r_2 / r_2 = 1)$ // For more pedagogy
16 **return** (p_2, q_2)

Let us denote p_2 and q_2 the two outputs of Alg. 7.12, we have:
- $d_{p_2} = d_p \mod p_2$, since $p_2 | (p-1)$;
- $d_{q_2} = d_q \mod q_2$, since $q_2 | (q-1)$;
- $i_{12} = {p_2}^{-1} \mod q_2$, since p_2 and q_2 are coprime.

We can apply Garner's formula to recover d:

$$d = d_{p_2} + p_2 \cdot ((i_{12} \cdot (d_{q_2} - d_{p_2})) \mod q_2) \ . \tag{7.1}$$

By Garner, we know that $0 \leq d < p_2 \cdot q_2 = \lambda(N)$, which is consistent with the remark made in the last sentence of Sec. 5.2.1.

Once we know the private exponent d, the public exponent e can be computed as the inverse of d modulo $\lambda(N)$. □

7.B Infective Aumüller CRT-RSA

The infective variant of Aumüller protection for CRT-RSA is detailed in Alg. 7.13.

Algorithm 7.13: CRT-RSA with Aumüller et al.'s countermeasure[4], under its infective avatar
(new algorithm contributed in this chapter)

Input : Message M, key (p, q, d_p, d_q, i_q)
Output: Signature $M^d \mod N$, or a random value

1 Choose a small random integer r.

2 $p' = p \cdot r$
3 $c_1 = p' + 1 \mod p$

4 $q' = q \cdot r$
5 $c_2 = q' + 1 \mod q$

6 $S'_p = M^{d_p \mod \varphi(p')} \mod p'$ // Intermediate signature in $\mathbb{Z}_{pr}$
7 $S'_q = M^{d_q \mod \varphi(q')} \mod q'$ // Intermediate signature in $\mathbb{Z}_{qr}$

8 $S_p = S'_p \mod p$ // Retrieve intermediate signature in $\mathbb{Z}_p$
9 $S_q = S'_q \mod q$ // Retrieve intermediate signature in $\mathbb{Z}_q$

10 $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \mod p)$ // Recombination in $\mathbb{Z}_N$
11 $c_3 = S - S'_p + 1 \mod p$
12 $c_4 = S - S'_q + 1 \mod q$

13 $S_{pr} = S'_p \mod r$ // Checksum of S_p in $\mathbb{Z}_r$
14 $S_{qr} = S'_q \mod r$ // Checksum of S_q in $\mathbb{Z}_r$
15 $c_5 = {S_{pr}}^{d_q \mod \varphi(r)} - {S_{qr}}^{d_p \mod \varphi(r)} + 1 \mod r$

16 **return** $S^{c_1 c_2 c_3 c_4 c_5}$

0x8

Protecting Asymmetric Cryptography Against Fault Attacks

This chapter presents a joint work with Sylvain Guilley as my adviser, Martin Moreau as my intern, and Zakaria Najm, the research engineer who helped me with the lab work for the case study. A part of this work, namely Sec. 8.3.3, has been done under he supervision of Gilles Barthe, François Dupressoir, and Pierre-Yves Strub while I was visiting them at the IMDEA Software Institute in Madrid, Spain. This work has been submitted to a conference and is currently under review.

Abstract. Asymmetric cryptography is essential to secure key exchange and digital signature, for instance. However, its mathematical strongness is victim of implementation-level hacks such as fault injection attacks. Countermeasures against them are still a matter of handcrafted artisanal work: there is no automation, parameter choices are scarcely motivated, and there are seldom formal security proofs. In this chapter, we rigorously generalize modular extension based countermeasure to all computations taking place in finite rings, enabling to protect all asymmetric cryptography. We present a formal model which enables us to prove the correctness of the induced code transformation and a compiler which implements it. We also state a rigorous security property and compute the fault non-detection probability ($\mathbb{P}_{\text{n.d.}}$) using a non-trivial result regarding the number of roots of polynomials representing fault propagations in the algorithm. Namely, $\mathbb{P}_{\text{n.d.}}$ is inversely proportional to the security parameter. We use our compiler to protect an elliptic curve scalar multiplication algorithm, and provide figures for practical performances (on an ARM Cortex-M4 microcontroller) for several values of the security parameter. We confirm the security of the obtained implementation by a systematic fault injection campaign.

Contents

8.1 Introduction

The necessity to protect information exchange does not have to be explained anymore. Properly used cryptography is of course a key building block for secure information exchange. Thus, implementation-level hacks must also be considered seriously in addition to the threat of cyber-attacks. In particular, fault injection attacks target physical implementations of secured devices in order to induce exploitable errors.

Formal methods. In cryptology, formal methods aim at providing a mathematical/mechanical proof of security, which helps in building trust into proved cryptosystems. However, their use is still limited in the field of fault injection and side channel attacks, as formal methods rely on models and implementations are difficult to model properly.

Asymmetric cryptography. Asymmetric cryptography addresses different needs such as key exchange and digital signature. RSA [RSA78], Diffie-Hellman [DH76], and ElGammal [Elg85] have been used for decades, and Elliptic-Curve Cryptography (ECC) algorithms are more and more deployed. ECC signature algorithms such as Elliptic-Curve Digital Signature Algorithm (ECDSA) [JMV01] are typical examples of ECC based cryptographic algorithms. ECC pairing-based cryptography have recently been accelerated in practice and is thus becoming practical [NNS10]. For example, the construction of "pairing-friendly" elliptic curves is an active subject [GV13]. Homomorphic encryption schemes, even if not practical yet, are progressively becoming a real need for privacy and are another example of asymmetric cryptography usage. The common point between all these algorithms is that the computations use large numbers and take place in mathematical structures such as finite rings and fields. This property enables the use of formal methods but also facilitates attacks.

Fault Attacks. As put forward in the reference book on fault analysis in cryptography [JT11, Chp. 9], there are three main categories of fault attacks.
1) *Safe-Error (SE) attacks* consist in testing whether an intermediate variable is dummy (usually introduced against simple power analysis, or SPA [KJJ99]) or not, by faulting it and looking whether there is an effect on the final result.
2) *Cryptosystem parameter alterations* with the goal of weakening the algorithm in order to facilitate key extraction. For example in ECC, invalid-curve fault attacks consist in moving the computation to a weaker curve, enabling the attacker to use cryptanalysis attacks exploiting the faulty outputs.
3) Finally, the most serious attacks are the one in the *Differential Fault Analysis* (DFA) category. Often the attack path consists in comparing correct and faulted outputs, like in the well-known BellCoRe attack on CRT-RSA (Rivest–Shamir–Adleman (RSA) optimized using the Chinese Remainder Theorem (CRT)), or the sign-change fault attack on ECC.

The *BellCoRe attack* [BDL97] on CRT-RSA introduced the concept of fault injection attacks. It is very powerful: faulting the computation even in a very random way yields almost certainly an exploitable result allowing to recover the secret primes of the RSA modulus $N = pq$. Indeed, in the CRT-RSA algorithm,

which is described in Alg. 8.1, most of the time is spent in the exponentiation algorithm. If the intermediate variable S_p (resp. S_q) is returned faulted as $\widehat{S_p}$ (resp. $\widehat{S_q}$), then the attacker gets an erroneous signature $\widehat{S}$, and is able to recover q (resp. p) as $\gcd(N, S - \widehat{S})$. For any integer x, $\gcd(N, x)$ can only be either 1, p, q, or N. In Alg. 8.1, if S_p is faulted (i.e., replaced by $\widehat{S_p} \neq S_p$), then $S - \widehat{S} = q \cdot ((i_q \cdot (S_p - S_q) \bmod p) - (i_q \cdot (\widehat{S_p} - S_q) \bmod p))$, and thus $\gcd(N, S - \widehat{S}) = q$. If S_q is faulted (i.e., replaced by $\widehat{S_q} \neq S_q$), then $S - \widehat{S} \equiv (S_q - \widehat{S_q}) - (q \bmod p) \cdot i_q \cdot (S_q - \widehat{S_q}) \equiv 0 \mod p$ because $(q \bmod p) \cdot i_q \equiv 1 \mod p$, and thus $S - \widehat{S}$ is a multiple of p. Additionally, the difference $(S - \widehat{S})$ is not a multiple of q. So, $\gcd(N, S - \widehat{S}) = p$.

Algorithm 8.1: Unprotected CRT-RSA

Input : Message M, key (p, q, d_p, d_q, i_q)
Output: Signature $M^d \bmod N$

1 $S_p = M^{d_p} \bmod p$ `// Intermediate signature in` $\mathbb{Z}_p$
2 $S_q = M^{d_q} \bmod q$ `// Intermediate signature in` $\mathbb{Z}_q$
3 $S = \mathrm{CRT}(S_p, S_q)$ `// Recombination in` $\mathbb{Z}_N$
4 **return** S

Since then, other attacks on CRT-RSA have been found, including as recently as last year, when Barthe et al. [BDF+14b] exposed two new families of fault injections on CRT-RSA: *"almost full" linear combinations of p and q*, and *"almost full" affine transforms of p or q*. Both target intermediate variables of the Montgomery multiplication algorithm (namely Coarsely Integrated Operand Scanning (CIOS) [MMM04]) used to implement the exponentiations of the CRT-RSA computation, and both leads to attacks based on the Lenstra–Lenstra–Lovász (LLL) lattice basis reduction algorithm [LLL82].

The *sign-change attack* [BOS06] on ECC consists, as its name suggests, in changing the sign of an intermediate elliptic curve point in the midst of an *Elliptic-Curve Scalar Multiplication* (ECSM). This means that the faulted resulting point is still on the curve so the fault is not detected by traditional point validation countermeasures. Such a fault can be achieved by for instance changing the sign in the double operation of the ECSM algorithm, see Alg. 8.2, on line 3. If the fault injection occurs during the last iteration of the loop, then the final result $\widehat{Q} = \sum_{i=1}^{n-1} k_i 2^i P + k_0 P = -Q + 2k_0 P$, i.e., either $\widehat{Q} = -Q$ or $\widehat{Q} = -Q + 2P$ depending on k_0, which reveals the value of k_0 to the attacker. This process can be iterated to find the other bits of the key, and optimizations exist that trade-off between the number of necessary faulted results and the required exhaustive search.

Algorithm 8.2: Double-and-add scalar multiplication

Input : $P \in E$, $k = \sum_{i=0}^{n} k_i 2^i$
Output: $[k]P$

1 $Q \leftarrow \mathcal{O}$
2 **for** $i \leftarrow n - 1$ ***down to*** 0 **do**
3 $\quad Q \leftarrow 2Q$
4 $\quad$ **if** $k_i = 1$ **then** $Q \leftarrow Q + P$
5 **end**
6 **return** Q

Both RSA and ECC algorithms continue to be the target of **many** new fault injection attacks: see [BDF+14b, LPEM+14, BGDSG+14, BGL14, EMFGL14] just for some 2014 papers. Besides, this topic is emerging and other new fault attacks will appear sooner or later. Hence, the need for efficient and practical generic countermeasures against fault attacks is obvious. David Wagner from UC Berkeley concurs in [Wag04]:

> "*It is a fascinating research problem to establish a principled foundation for security against fault attacks and to find schemes that can be proven secure within that framework.*"

In this chapter, we focus on countermeasures which guarantee the integrity of the computation result, hence covering most existing and future DFA attacks. Integrity verification can be carried out by classical methods such as parity or crafted codes (e.g., in [TNK+14]). These techniques are necessarily blind to the mathematical structures, and are favored for low-level verifications which are typically implemented in hardware.

Countermeasures. Verifications compatible with mathematical structures can be applied either at computational or at algorithmic level.
Algorithmic protections have been proposed by Giraud [Gir06] (and many others [BNP07, LRT14, KKHH11]) for CRT-RSA, which naturally transpose to ECC, as shown in [KFSV11]. These protections are implementation specific (e.g., depend on the chosen exponentiation algorithm) and are thus difficult to automate, requiring specialized engineering skills.
Computational protections have been pioneered by Shamir in [Sha99] using modular extension, initially to protect CRT-RSA. The idea is to carry out the same computation in two different algebraic structures allowing to check the computation before disclosing its result. For example protecting a computation in $\mathbb{Z}_p$ consists in carrying out the same computation in $\mathbb{Z}_{pr}$ and $\mathbb{Z}_r$, where r is a small number ($r \ll p$); the computation in $\mathbb{Z}_{pr}$ must match that of $\mathbb{Z}_r$ when reduced modulo r, if not an error is returned, otherwise the result in $\mathbb{Z}_{pr}$ is reduced modulo p and returned. This method operates at low level (integer arithmetic), thereby enabling countermeasures (and optimizations) to be added on top of it. They are thus easily maintained, which explains why this method is quite popular. Indeed, there is a wealth of variants for CRT-RSA stemming from this idea [ABF+02, Vig08a, JPY01, BOS03, CJ05, DGRS09], as well as a few proofs-of-concept transposing it to ECC [BOS06, BV07]. Despite the nonexistence of literature, the same idea could apply to post-quantum code-based cryptography, pairing, and homomorphic computation for instance. Therefore, our chapter focuses on computational countermeasures.

On the one hand, the variety of CRT-RSA countermeasures shows that fault attacks are a threat that is taken seriously by both the academic and the industrial communities. On the other hand, it bears witness to the artisanal way these countermeasures were put together. Indeed, the absence of formal security claims and of proofs added to the necessity of writing implementations by hand results in many weaknesses in existing countermeasures and thus in many attempts to create better ones.

The two attempts at porting the modular extension idea from Shamir's CRT-RSA countermeasure to ECC are not exempt from these problems. For Blömer et al. [BOS06], the computation in small ring $\mathbb{Z}_r$ must stay on an elliptic curve, and thus the parameter r is a known fixed prime number chosen at production time. But we claim that the computation in the small structure is actually not required to "make sense" (i.e., it only has to compute the same thing as to the one in the overring $\mathbb{Z}_{pr}$, without any regard for what it is). And for Baek and Vasyltsov [BV07], r is a random number chosen at runtime. In practice, the lack of formal study of the fault non-detection probability makes both approaches blind to a critical vulnerability. In the first one, the attacker can use the knowledge of r to provoke collisions in the small structure (e.g., if the attacker controls the input point, and/or by injecting faults). In the second one, the fact that r is friable means that some computations involving divisions (as ECC computations do) in $\mathbb{Z}_r$ may fail in manners that were not expected by the implementation, which can reveal information to the attacker. Also, neither of the two countermeasures discuss the problem of divisions in $\mathbb{Z}_{pr}$, which we address in Sec. 8.2.3. Finally, they only claim to protect against sign-change fault attacks[1] while the principle of modular extension offers a more general protection when properly instantiated, as we will explain in more details.

Contributions. To this day, countermeasures against fault injection attacks are still a matter of handcrafted artisanal work: there is no automation, parameter choices are scarcely motivated, and there are seldom rigorous security proofs. Most notably, the fault non-detection probability is never formally established. Indeed, we just exhibited a functional and a security problem in the sole two ECC papers discussing computational countermeasures.

In this chapter we are the first to rigorously generalize modular extension to all computations taking place in finite rings. Precisely, we present in Sec. 8.2 a protection method that we call *entanglement* and claim to be both *generic* (works against all DFA attacks) and *universal* (applies to all asymmetric cryptography).

In Sec. 8.3, we present a formal model which enables us to prove the correctness of the induced code transformation (i.e., it preserves the semantics), and a compiler called enredo which automates the entanglement protection. We also state a rigorous security property and compute the fault non-detection probability using a non-trivial result regarding the number of roots of polynomials representing fault propagations in the algorithm. This result allows us to confirm figures announced for CRT-RSA and to provide figures for ECC and asymmetric cryptography computation in general.

We use our compiler in Sec. 8.4 to generate a protected elliptic curve scalar multiplication (ECSM) algorithm, which enables us to provide actual figures for practical performances (on ARM) for this type of countermeasures on ECC. In addition, we confirm that the non-detection probability is inversely proportional to the security parameter on the obtained implementation by a systematic fault injection campaign. Our experiments allow to refine the specific case of ECC: a proportionality constant is put forward in practice and accounted for in theory. Interestingly, this constant has been completely overlooked in the previous publications about modular extension based countermeasures for ECC.

[1] Although, they do point validation for protection against other attack paths.

8.2 Integrity Verification

8.2.1 Redundancy

The obvious way to verify the integrity of a computation is to repeat it and compare the results. There are two drawbacks to this approach: 1. with a suitably calibrated setup, the attacker could be able to inject the exact same fault in both computations; and 2. more importantly, repeating the entire computation is costly. For example CRT-RSA is used when the 4× speed-up enabled by the CRT optimization is deemed necessary, so we don't want to double the computation time.

8.2.2 Entanglement

Fortunately, there is a solution that solves both problems. It consists on the one hand in embedding a form of redundancy in the computation itself by *entangling* the original values and smaller ones, such that computations are performed on the resulting entangled values which cannot be faulted independently but only as a whole. And on the other hand in performing a small computation (of comparatively negligible cost) in order to verify the small redundant part (once extracted) of the main computation.

This idea has almost as many different implementations as there are CRT-RSA countermeasures against the BellCoRe attack. Indeed, the redundancy embedding can be implemented in multiple ways. We found that the term *entanglement* nicely captures the multiplicity of such proteiform protections.

The general idea is to lift the computation into an over-structure (e.g., an over-ring) which allows to quotient the result of the computation back to the original structure, as well as quotienting a "checksum" of the computation to a smaller structure. What has just been described is the *direct product* of the underlying algebraic structures. If an equivalent computation is performed in parallel in the smaller structure, its result can be compared with the checksum of the main computation. If they are the same, we have a high confidence in the integrity of the main computation. The confidence degree depends directly from the size of the smaller structure, which is thus a security parameter: the larger it is, the less probable it is to have an unwanted collision, but the more costly the redundancy will be. It is also possible to use several different small structures to lower the collision probability. The only requirement is that all the structures (the original one and the small ones) are pairwise compatible for direct product. In the case of rings for instance, it means that their characteristics must be pairwise coprime.

This method is indeed a solution to the first of our problems: faulting the small computation in a way that will have the same effect as a fault on the main computation is much more complicated, especially since the smaller structure ($\mathbb{Z}_r$ in Ex. 8.1) can be chosen randomly at each execution.
It also solves our second problem as the overhead induced by the redundancy is almost negligible: for instance in Ex. 8.1, p and q are typically $1,024$ or $2,048$ bits long, while r can be as small as 32 or 64 bits long.

Example 8.1 (Protecting the p part of a CRT-RSA computation using the entanglement scheme)**.** Let p, q be big prime numbers, and M, $d \in \mathbb{Z}_{pq}$. We want to

protect the computation

$$S_p = M^d \text{ in the } \mathbb{Z}_p \text{ ring, i.e., } S_p = (M \bmod p)^{d \bmod \varphi(p)} \bmod p.$$

We lift it in the $\mathbb{Z}_{pr}$ overring where r is a small random number, thus we compute

$$S_{pr} = M^d \text{ in } \mathbb{Z}_{pr}, \text{ i.e., } S_{pr} = (M \bmod pr)^{d \bmod \varphi(pr)} \bmod pr.$$

Then, the "checksum" value of

$$S_r = S_{pr} \bmod r$$

can be compared with the result of computing

$$S'_r = M^d \text{ in } \mathbb{Z}_r, \text{ i.e., } S'_r = (M \bmod r)^{d \bmod \varphi(r)} \bmod r,$$

which is an inexpensive computation. If they are equal, there is only a small probability that the computation has been tampered with[2]. Eventually, the result in the $\mathbb{Z}_p$ ring can be retrieved as $S_p = S_{pr} \bmod p$. Otherwise, we return an error constant (denoted $\perp$) to avoid the risk of revealing useful information to the attacker.

8.2.3 Inversions in Direct Products

Many asymmetric cryptography computations take place in an $\mathbb{F}_p$ field and rely on the possibility of performing divisions. In the entanglement protection scheme, we lift the computation to a direct product $\mathbb{Z}_{pr}$, where it might happen that inversing a number is not possible. It is actually possible to circumvent this problem in the entanglement setting. Indeed, divisions can be optimized, as expressed in the following proposition.

Proposition 8.1 (Divisions optimization)**.** *To get the inverse of z in $\mathbb{F}_p$ while computing in $\mathbb{Z}_{pr}$, one has:*

- $z = 0 \bmod r \implies (z^{p-2} \bmod pr) \equiv z^{-1} \mod p$,
- *otherwise* $(z^{-1} \bmod pr) \equiv z^{-1} \mod p$.

Proof. If $z = 0 \mod r$, then z is not invertible in $\mathbb{Z}_{pr}$. However, z^{p-2} exists in $\mathbb{Z}_{pr}$, and $(z^{p-2} \bmod pr) \bmod p = z^{p-2} \bmod p = z^{-1} \bmod p$.

Otherwise, when $z \neq 0 \mod r$, we have in $\mathbb{Z}_{pr}$ that $z^{-1} = z^{\varphi(pr)-1} = z^{pr-p-r} \bmod pr$. Now, $(z^{-1} \bmod pr) \bmod p = z^{-1} \bmod p$ if and only if: $\varphi(p)$ divides $(pr - p - r) - (-1)$. But $(pr - p - r) - (-1) = (p-1)(r-1)$, which is indeed a multiple of $\varphi(p) = p - 1$. □

The modular inverse of a number can be efficiently obtained thanks to an extended Euclid algorithm [MvOV96, Algorithm 2.107 at §2.4, p. 67]. The complexity of this algorithm is quadratic in the size in bits of the modulo (i.e., it is $O(\log_2(p))$ when the ring is $\mathbb{Z}_p$), like the modular multiplication (see [MvOV96, Table 2.8 in Chap. 2, page 84]). However, in practice, for moduli of cryptographic size (i.e., $192 \leq \log_2(p) \leq 521$ for ECC, see [U.S99]) the duration of a division lasts from 4 to 10 times the duration of a multiplication (as benchmarked by OpenSSL version

[2] This probability is detailed in the security proof in Sec. 8.3.4.

1.0.1f `BN_mod_mul` and `BN_mod_inverse` functions). Prop. 8.1 shows that divisions can also be implemented efficiently in $\mathbb{Z}_{pr}$, provided the division exists. If not, an exponentiation $z \mapsto z^{p-2}$ is necessary. Assuming that the binary representation of p consists of as many ones as zeros and that the exponentiation is done with a double-and-add algorithm, the cost of $z \mapsto z^{p-2}$ is about $\frac{3}{2}\lfloor \log_2 p \rfloor$, that is 288 multiplications when p is a 192-bit prime number.

However, multiples of r occur only with probability $\frac{1}{r}$ in computations. Thus, an upper-bound for the expected overhead is $(10 \times (1 - \frac{1}{r}) + 384 \times \frac{1}{r})/10 \approx 1 + 10^{-8}$ when r is a 32 bit number, which is negligible in practice.

Remark 8.1. The entanglement protection scheme reminds of *error detecting codes* (or simply "codes") [WK11, TNK+14]. Now, codes are designed to armor a data before transmission over a noisy channel, in order to detect and possibly correct faults striking a codeword. However, in general, codewords are not meant to be used in computations; in particular, the set of codewords does not form a ring, i.e., it is not stable by both addition and multiplication. Linear codes are indeed stable by addition, but do not feature any multiplicative structure. Therefore, using codes to detect faults in public key cryptography is restricted to checking the integrity of each state, and not of operations. But operations are security-critical steps which deserve a proper protection; hence the need for the integrity verification by entanglement.

8.2.4 Generic and Universal Protection

While not being formalized at this high level of abstraction, the protection strategy used by most of the countermeasures which claim to defend against the BellCoRe attack can actually be seen as an instance of the one we just described [RG14c]. Therefore, we claim that the existing countermeasures actually protect against other attacks than the BellCoRe one. We also claim that implementations equipped with one of these countermeasures are protected regardless of the technical implementation of the arithmetic computation.

Proposition 8.2 (Universality)**.** *Algorithms that implement the* entanglement *protection scheme described in Sec. 8.2.2 are protected against all randomizing fault injection attacks.*

Intuition. If the attacker performs a DFA, then either the result of the arithmetic computation is not corrupted and everything is fine, or the integrity of the result has been compromised and the algorithm will return an error with overwhelming probability. □

Proposition 8.3 (Genericity)**.** *The* entanglement *protection scheme can protect any arithmetic computation taking place in an algebraic structure which supports the lifting and quotienting operations described in Sec. 8.2.2.*

Intuition. The entanglement protection scheme only depends on the properties of the algebraic structure in which the arithmetic computation to be protected takes place. Thus, any operation properly defined on the algebraic structure can be performed any number of time during the computation. The countermeasure will be effective as long as equivalent operations are performed in the smaller structure. □

```
Algo   ::= Decl* Stmt
Decl   ::= "Field" Struct ";"
         | "Ring" Struct "=" Struct ("*" Struct)* ";"
         | Struct Var ";"
Stmt   ::= "skip"
         | Var ":=" Expr
         | "if" Cond "then" Stmt+ "else" Stmt+ "end"
         | "while" Cond "do" Stmt+ "end"
         | "return" Var
         | Stmt ; Stmt
Expr   ::= "(" Expr ")"
         | Expr "+" Expr
         | Expr "-" Expr
         | Expr "*" Expr
         | Expr "^" Expr
         | Expr "[" Struct "]"
         | Var
         | <int>
Cond   ::= "(" Cond ")"
         | Cond "/\" Cond
         | Cond "\/" Cond
         | "!" Cond
         | Expr "=" Expr
         | Expr "!=" Expr
Struct ::= <uident>
Var    ::= <lident>
```

Figure 8.1: enredo's input language.

8.3 The enredo Compiler

We implemented the entanglement protection scheme (see Sec. 8.2.2) in a compiler called enredo. The compiler takes an algorithm and the number of small structures to use as an argument and generate an equivalent protected algorithm.

8.3.1 Input Language

The input language of enredo is a "while language". This ensures that the language is generic enough to express all the algorithms one would want to, while keeping the language simple enough for formal study. This language, which Backus–Naur Form (BNF) can be seen on Fig. 8.1, is used to describe the algorithms that we want enredo to protect by applying the entanglement protection scheme.

Algorithm descriptions start with the declarations of the structures (`Fields` and `Rings`) and of the variables, which are members of these structures. Then the algorithm itself is stated using an imperative style, i.e., a list of statements that can be assignments of expressions to variables, conditional branching or looping. An expression can be a number or a reference to a variable, or the sum of two expressions, or their product, or a projection of an expression into a given structure

$$\frac{\text{“}\texttt{Field F}\text{”}}{\Gamma \vdash \texttt{F} : p} \text{ field } (p \text{ is a fresh prime number})$$

$$\frac{\text{“}\texttt{Ring R=F}_1 \star \texttt{F}_2 \star \ldots \star \texttt{F}_n\text{”} \quad \Gamma \vdash \texttt{F}_1 : p_1 \;\; \Gamma \vdash \texttt{F}_2 : p_2 \;\ldots\; \Gamma \vdash \texttt{F}_n : p_n}{\Gamma \vdash \texttt{R} : \prod_{n=1}^{i} p_i} \text{ ring}$$

$$\frac{\text{“}\texttt{S v}\text{”} \quad \Gamma \vdash \texttt{S} : s}{\Gamma \vdash \texttt{v} : s} \text{ variable}$$

Figure 8.2: Typing judgments.

that has to be compatible with the type of the expression. A condition can be an equality or an inequality between expressions, or the negation of the condition, or the conjunction or disjunction of expressions. A structure is said to be compatible with the type of an expression if either of the following is true:

- the type of the expression is $\mathbb{Z}$,
- the structure is $\mathbb{Z}$,
- both are the same field $\mathbb{Z}_p$,
- the type of the expression is $\mathbb{Z}_p$ and the structure is $\mathbb{Z}_{\prod_i p_i}$, where $\exists i, p_i = p$,
- the type of the expression is $\mathbb{Z}_{\prod_i p_i}$ and the structure is $\mathbb{Z}_p$, where $\exists i, p_i = p$,
- the type of the expression is $\mathbb{Z}_{\prod_i p_i}$ and the structure is $\mathbb{Z}_{\prod_j q_j}$, where $\forall i, \exists j, p_i = q_j$.

8.3.2 Typing and Semantics

The typing environment Γ of an algorithm written in enredo's input language is obtained by judging its types and variables declarations (i.e., the `Decl` part in the BNF provided on Fig. 8.1). Typing judgments are described in Fig. 8.2 using sequent notation.

Let `a` be an algorithm expressed in enredo's input language. Let Γ be a typing of algorithm `a`. Let m be a memory with the argument variables set to their input values and all other variables to the default value zero.

The semantic of `a` is $[\![\texttt{a}]\!]_\Gamma\ m$, defined by:

$$\begin{aligned}
[\![\texttt{skip}]\!]_\Gamma &= m \mapsto m \\
[\![\texttt{v := e}]\!]_\Gamma &= m \mapsto m\{\texttt{v} \leftarrow [\![\texttt{e}]\!]_{expr(\Gamma,m)} \bmod \Gamma[\texttt{v}]\} \\
[\![\texttt{if c t e}]\!]_\Gamma &= \text{if } [\![\texttt{c}]\!]_{cond(\Gamma,m)} \text{ then } [\![\texttt{t}]\!]_\Gamma \text{ else } [\![\texttt{e}]\!]_\Gamma \\
[\![\texttt{while c d}]\!]_\Gamma &= \text{if } [\![\texttt{c}]\!]_{cond(\Gamma,m)} \\
&\quad \text{then } [\![\texttt{d ; while c d}]\!]_\Gamma \\
&\quad \text{else } [\![\texttt{skip}]\!]_\Gamma \\
[\![\texttt{return v}]\!]_\Gamma &= m[\texttt{v}], \\
&\quad \text{note that a value and not a memory is returned here;} \\
[\![\texttt{s ; a}]\!]_\Gamma &= [\![\texttt{a}]\!]_\Gamma \mathbin{\vec{\circ}} [\![\texttt{s}]\!]_\Gamma, \\
&\quad \text{where } f \mathbin{\vec{\circ}} g \text{ is } f \circ g \text{ if } g \text{ returns a} \\
&\quad \text{memory, and } g \text{ otherwise;}
\end{aligned}$$

where $[\![\mathtt{e}]\!]_{expr(\Gamma,m)}$ is defined by:

$$\begin{aligned}
[\![\mathtt{e}_1 + \mathtt{e}_2]\!]_{expr(\Gamma,m)} &= [\![\mathtt{e}_1]\!]_{expr(\Gamma,m)} + [\![\mathtt{e}_2]\!]_{expr(\Gamma,m)} \\
[\![\mathtt{e}_1 - \mathtt{e}_2]\!]_{expr(\Gamma,m)} &= [\![\mathtt{e}_1]\!]_{expr(\Gamma,m)} - [\![\mathtt{e}_2]\!]_{expr(\Gamma,m)} \\
[\![\mathtt{e}_1 * \mathtt{e}_2]\!]_{expr(\Gamma,m)} &= [\![\mathtt{e}_1]\!]_{expr(\Gamma,m)} \times [\![\mathtt{e}_2]\!]_{expr(\Gamma,m)} \\
[\![\mathtt{e}_1 \hat{} \mathtt{e}_2]\!]_{expr(\Gamma,m)} &= [\![\mathtt{e}_1]\!]_{expr(\Gamma,m)}^{[\![\mathtt{e}_2]\!]_{expr(\Gamma,m)}} \\
[\![\mathtt{e}\ \mathtt{[S]}]\!]_{expr(\Gamma,m)} &= [\![\mathtt{e}]\!]_{expr(\Gamma,m)} \bmod \Gamma[\mathtt{S}] \\
[\![\mathtt{v}]\!]_{expr(\Gamma,m)} &= m[\mathtt{v}] \bmod \Gamma[\mathtt{v}] \\
[\![\mathtt{n}]\!]_{expr(\Gamma,m)} &= n;
\end{aligned}$$

and $[\![\mathtt{c}]\!]_{cond(\Gamma,m)}$ is defined by:

$$\begin{aligned}
[\![\mathtt{c}_1\ \mathtt{/\backslash}\ \mathtt{c}_2]\!]_{cond(\Gamma,m)} &= [\![\mathtt{c}_1]\!]_{cond(\Gamma,m)} \wedge [\![\mathtt{c}_2]\!]_{cond(\Gamma,m)} \\
[\![\mathtt{c}_1\ \mathtt{\backslash/}\ \mathtt{c}_2]\!]_{cond(\Gamma,m)} &= [\![\mathtt{c}_1]\!]_{cond(\Gamma,m)} \vee [\![\mathtt{c}_2]\!]_{cond(\Gamma,m)} \\
[\![\mathtt{!\ c}]\!]_{cond(\Gamma,m)} &= \neg\, [\![\mathtt{c}]\!]_{cond(\Gamma,m)} \\
[\![\mathtt{e}_1 = \mathtt{e}_2]\!]_{cond(\Gamma,m)} &= [\![\mathtt{e}_1]\!]_{expr(\Gamma,m)} = [\![\mathtt{e}_2]\!]_{expr(\Gamma,m)} \\
[\![\mathtt{e}_1\ \mathtt{!=}\ \mathtt{e}_2]\!]_{cond(\Gamma,m)} &= [\![\mathtt{!\ (e}_1 = \mathtt{e}_2\mathtt{)}]\!]_{cond(\Gamma,m)}.
\end{aligned}$$

8.3.3 Correctness of the Transformation

Let a be an algorithm. We define the enredo transformation $\langle a \rangle_r$ for any field $\mathbb{Z}_r$. Let Γ be a typing of algorithm a. We define $\langle \Gamma \rangle_r$ as $\Gamma' \uplus \Gamma_r$, where, $\forall\ \mathtt{v} \in Dom(\Gamma)$,

- $\Gamma_r[\langle \mathtt{v} \rangle_v] = r$, $\Gamma_r[\mathbb{Z}_r] = r$, and
- $\Gamma'[\mathtt{v}] = \Gamma[\mathtt{v}] \times r$.

Let m be a memory with the argument variables set to their input values and all other variables to the default value zero. We define $\langle m \rangle_r$ as $m' \uplus m_r$, where $\forall\ \mathtt{v} \in Dom(m)$,

- $m_r[\langle \mathtt{v} \rangle_v] = m[\mathtt{v}] \bmod r$, and
- $m'[\mathtt{v}] = \mathrm{CRT}(m[\mathtt{v}], m_r[\langle \mathtt{v} \rangle_v])$.

The enredo transformation of a is $\langle \mathtt{a} \rangle_{r,\Gamma}$, defined by:

$$\begin{aligned}
\langle \mathtt{v := e} \rangle_{r,\Gamma} &= \langle \mathtt{v} \rangle_v \ \mathtt{:=}\ \langle \mathtt{e} \rangle_e \ \mathtt{;}\ \mathtt{v} \ \mathtt{:=}\ \langle \mathtt{e} \rangle_E \ \mathtt{;} \\
\langle \mathtt{return\ v} \rangle_{r,\Gamma} &= \mathtt{if\ v\ [}\mathbb{Z}_r\mathtt{]} = \langle \mathtt{v} \rangle_v \\
&\quad\ \mathtt{then\ return\ v\ [}\Gamma[\mathtt{v}]\mathtt{]\ ;} \\
&\quad\ \mathtt{else\ return}\ \bot\ \mathtt{;} \\
\langle \mathtt{s\ ;\ a} \rangle_{r,\Gamma} &= \langle \mathtt{s} \rangle_{r,\Gamma}\ \langle \mathtt{a} \rangle_{r,\Gamma},
\end{aligned}$$

where $\langle \mathtt{e} \rangle_e$ is defined by:

$$\begin{aligned}
\langle \mathtt{e}_1 + \mathtt{e}_2 \rangle_e &= \langle \mathtt{e}_1 \rangle_e + \langle \mathtt{e}_2 \rangle_e \\
\langle \mathtt{e}_1 - \mathtt{e}_2 \rangle_e &= \langle \mathtt{e}_1 \rangle_e - \langle \mathtt{e}_2 \rangle_e \\
\langle \mathtt{e}_1 * \mathtt{e}_2 \rangle_e &= \langle \mathtt{e}_1 \rangle_e * \langle \mathtt{e}_2 \rangle_e \\
\langle \mathtt{e}_1 \hat{} \mathtt{e}_2 \rangle_e &= \langle \mathtt{e}_1 \rangle_e \hat{} \langle \mathtt{e}_2 \rangle_e \\
\langle \mathtt{e\ [S]} \rangle_e &= \langle \mathtt{e} \rangle_e \\
\langle \mathtt{v} \rangle_e &= \langle \mathtt{v} \rangle_v \\
\langle \mathtt{n} \rangle_e &= \mathtt{n},
\end{aligned}$$

and $\langle \mathtt{e} \rangle_E$ is defined by:

$$\begin{aligned}
\langle \mathtt{e}_1 + \mathtt{e}_2 \rangle_E &= \langle \mathtt{e}_1 \rangle_E + \langle \mathtt{e}_2 \rangle_E \\
\langle \mathtt{e}_1 - \mathtt{e}_2 \rangle_E &= \langle \mathtt{e}_1 \rangle_E - \langle \mathtt{e}_2 \rangle_E \\
\langle \mathtt{e}_1 * \mathtt{e}_2 \rangle_E &= \langle \mathtt{e}_1 \rangle_E * \langle \mathtt{e}_2 \rangle_E \\
\langle \mathtt{e}_1 \hat{} \mathtt{e}_2 \rangle_E &= \langle \mathtt{e}_1 \rangle_E \hat{} \langle \mathtt{e}_2 \rangle_E \\
\langle \mathtt{e\ [S]} \rangle_E &= \langle \mathtt{e} \rangle_E\ \mathtt{[}\langle \mathtt{S} \rangle_s\mathtt{]} \\
\langle \mathtt{v} \rangle_E &= \mathtt{v} \\
\langle \mathtt{n} \rangle_E &= \mathtt{n}.
\end{aligned}$$

Proposition 8.4 (Correctness of the enredo transformation)**.** *The enredo transformation is* correct *if at all time during the execution the invariant defining the transformation of the memory holds, and when a value is returned, it is either the same as in the original program, or* $\perp$ *(meaning that no information is output since an error has been detected).*

Formally, let a *be an algorithm,* Γ *be a typing of algorithm* a*, and* r *a random prime.* $\forall$ *initial memory* m*, for each statement* s *in* a*, we have either*

$$\begin{array}{ccc} m & \xrightarrow{[\![s]\!]_\Gamma} & m' \\ {\scriptstyle\langle\cdot\rangle_r}\downarrow & & \downarrow{\scriptstyle\langle\cdot\rangle_r} \\ \langle m\rangle_r & \xrightarrow[[\![\langle s\rangle_{r,\Gamma}]\!]_{\langle\Gamma\rangle_r}]{} & \langle m'\rangle_r \end{array} \quad \textit{or} \quad \begin{array}{ccc} m & \xrightarrow{[\![s]\!]_\Gamma} & v \\ {\scriptstyle\langle\cdot\rangle_r}\downarrow & & \| \\ \langle m\rangle_r & \xrightarrow[[\![\langle s\rangle_{r,\Gamma}]\!]_{\langle\Gamma\rangle_r}]{} & v' \end{array}$$

during the execution *when the algorithm terminates.*

Proof. The proof is done by structural induction on algorithm a.

Empty program. $[\![\langle \texttt{skip}\rangle_{r,\Gamma}]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r = \langle m\rangle_r$ and $[\![\texttt{skip}]\!]_\Gamma\ m = m$, in which case the first diagram trivially holds.

Assignments. For assignments we have

$$\begin{aligned} &[\![\langle \mathtt{v} := \mathtt{e}\rangle_{r,\Gamma}]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r \\ &= [\![\langle \mathtt{v}\rangle_v := \langle \mathtt{e}\rangle_e \texttt{ ; } \mathtt{v} := \langle \mathtt{e}\rangle_E]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r \\ &= [\![\mathtt{v} := \langle \mathtt{e}\rangle_E]\!]_{\langle\Gamma\rangle_r} \vec{\circ} [\![\langle \mathtt{v}\rangle_v := \langle \mathtt{e}\rangle_e]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r \\ &= [\![\mathtt{v} := \langle \mathtt{e}\rangle_E]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r\{\langle \mathtt{v}\rangle_v \leftarrow [\![\langle \mathtt{e}\rangle_e]\!]_{expr(\Gamma,m)} \bmod \langle\Gamma\rangle_r[\langle \mathtt{v}\rangle_v]\} \\ &= \langle m\rangle_r\{\mathtt{v} \leftarrow [\![\langle \mathtt{e}\rangle_E]\!]_{expr(\langle\Gamma\rangle_r,\langle m\rangle_r)} \bmod \langle\Gamma\rangle_r[\mathtt{v}]\} \\ &\qquad \{\langle \mathtt{v}\rangle_v \leftarrow [\![\langle \mathtt{e}\rangle_e]\!]_{expr(\langle\Gamma\rangle_r,\langle m\rangle_r)} \bmod \langle\Gamma\rangle_r[\langle \mathtt{v}\rangle_v]\} \\ &= \langle m\rangle_r\{\mathtt{v} \leftarrow [\![\langle \mathtt{e}\rangle_E]\!]_{expr(\langle\Gamma\rangle_r,\langle m\rangle_r)} \bmod \Gamma[\mathtt{v}] \times r\} \\ &\qquad \{\langle \mathtt{v}\rangle_v \leftarrow [\![\langle \mathtt{e}\rangle_e]\!]_{expr(\langle\Gamma\rangle_r,\langle m\rangle_r)} \bmod r\} \end{aligned}$$

and $[\![\mathtt{v} := \mathtt{e}]\!]_\Gamma\ m = m\{\mathtt{v} \leftarrow [\![\mathtt{e}]\!]_{expr(\Gamma,m)} \bmod \Gamma[\mathtt{v}]\}$, which satisfies the first diagram too.

Returns. When returning a value we have

$$\begin{aligned} &[\![\langle \texttt{return v}\rangle_{r,\Gamma}]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r \\ &= [\![\texttt{if v[}\mathbb{Z}_r\texttt{]} = \langle \mathtt{v}\rangle_v \texttt{ then return v[}\Gamma[\mathtt{v}]\texttt{] ; else return } \perp \texttt{ ;}]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r \\ &= (\text{if } [\![\mathtt{v}\texttt{[}\mathbb{Z}_r\texttt{]} = \langle \mathtt{v}\rangle_v]\!]_{cond(\langle\Gamma\rangle_r,\langle m\rangle_r)} \text{ then } [\![\texttt{return v[}\Gamma[\mathtt{v}]\texttt{]}]\!]_{\langle\Gamma\rangle_r} \\ &\quad \text{else } [\![\texttt{return } \perp]\!]_{\langle\Gamma\rangle_r})\langle m\rangle_r \\ &= (\text{if } [\![\mathtt{v}\texttt{[}\mathbb{Z}_r\texttt{]}]\!]_{expr(\langle\Gamma\rangle_r,\langle m\rangle_r)} = [\![\langle \mathtt{v}\rangle_v]\!]_{expr(\langle\Gamma\rangle_r,\langle m\rangle_r)} \text{ then } [\![\texttt{return v[}\Gamma[\mathtt{v}]\texttt{]}]\!]_{\langle\Gamma\rangle_r} \\ &\quad \text{else } [\![\texttt{return } \perp]\!]_{\langle\Gamma\rangle_r})\langle m\rangle_r \\ &= (\text{if } \langle m\rangle_r[\mathtt{v}] \bmod r = \langle m\rangle_r[\langle \mathtt{v}\rangle_v] \text{ then } [\![\texttt{return v[}\Gamma[\mathtt{v}]\texttt{]}]\!]_{\langle\Gamma\rangle_r} \\ &\quad \text{else } [\![\texttt{return } \perp]\!]_{\langle\Gamma\rangle_r})\langle m\rangle_r \\ &= [\![\texttt{return v[}\Gamma[\mathtt{v}]\texttt{]}]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r \text{ by induction} \\ &= \langle m\rangle_r[\mathtt{v}] \bmod \Gamma[\mathtt{v}] \\ &= m[\mathtt{v}] \end{aligned}$$

and $[\![\texttt{return v}]\!]_\Gamma\ m = m[\texttt{v}]$, which satisfies the second diagram.

Sequential composition. For the sequential composition we have

$[\![\langle \texttt{s ; a}\rangle_{r,\Gamma}]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r = [\![\langle \texttt{a}\rangle_{r,\Gamma}]\!]_{\langle\Gamma\rangle_r} \vec{\circ}\ [\![\langle \texttt{s}\rangle_{r,\Gamma}]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r$, and
$[\![\texttt{s ; a}]\!]_\Gamma\ m = [\![\texttt{a}]\!]_\Gamma \vec{\circ}\ [\![\texttt{s}]\!]_\Gamma\ m$

If $[\![\texttt{s}]\!]_\Gamma\ m$ returns a value, then by induction we know that $[\![\langle \texttt{s}\rangle_{r,\Gamma}]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r$ does return the same value. In that case by definition of $\vec{\circ}$ it works.

If $[\![\texttt{s}]\!]_\Gamma\ m$ returns a memory m', then by induction we know that $[\![\langle \texttt{s}\rangle_{r,\Gamma}]\!]_{\langle\Gamma\rangle_r}\langle m\rangle_r$ does return the memory $\langle m'\rangle_r$. In that case by induction on the structure of a it works. □

8.3.4 Security Proof

Proposition 8.5 (Security)**.** *Let a be an algorithm, expressed in* enredo*'s input language, that can be unrolled. Let* Γ *be its typing and* r*, a random prime, used as security parameter. The* enredo *transformation is secure if* $\langle a\rangle_{r,\Gamma}$ *returns either the same result as a or* $\perp$ *with a probability asymptotically inversely proportional to* r *(when* r *grows larger and larger).*

Proof. In the absence of fault injection attacks, the correctness proof of Prop. 8.4 is sufficient as a security proof as it proves that $\langle \texttt{a}\rangle_{r,\Gamma}$ always returns the same result as a does.

Cryptographic algorithms are finite sequences of assignments. To carry out the security proof in the presence of fault injection attacks, we start by introducing a fault injection formalism. We have given a small-step semantic to enredo's input language in Sec. 8.3.2. In this semantic, we can easily model randomizing faults.

- Permanent faults are modeled by inserting a new statement assigning random to the targeted variable at the desired moment. For example, a permanent fault on the intermediate variable x in $[\![\texttt{v := x + y}]\!]_\Gamma$ is written $[\![\texttt{v := x + y}]\!]_\Gamma \vec{\circ}\ [\![\texttt{x :=}\,\mathsf{random}]\!]_\Gamma$.
- Transient faults are modeled by temporarily modifying the variable. For example, a transient fault on the intermediate variable x in $[\![\texttt{v := x + y}]\!]_\Gamma$ is written $[\![\texttt{x := \#}]\!]_\Gamma \vec{\circ}\ [\![\texttt{v := x + y}]\!]_\Gamma \vec{\circ}\ [\![\texttt{x :=}\,\mathsf{random}]\!]_\Gamma \vec{\circ}\ [\![\texttt{\# := x}]\!]_\Gamma$, where # is a special variable unused elsewhere.

Also, as a convenience, we consider that algorithms are finite sequences of assignments. Indeed, since cryptographic computation terminates, we assume that we unroll each while loops (as the small-step semantics given in Sec. 8.3.2 does) to a finite sequence of assignments. Therefore, as detailed in the two previous bullets, a faulted algorithm can also be formally described as a finite sequence of assignments.

Faulted results are polynomials of faults. The result can also be written as a function of a subset of the intermediate variables, plus some inputs if the intermediate variables do not suffice to finish the computation. We are interested in the expression of the result as a function of the intermediate variables which are the target of a transient or permanent fault injection. We give the formal name $\widehat{x}$ to any

faulted variable x. For convenience, we denote them by $\widehat{x_i}$, $1 \leq i \leq n$, where $n \geq 1$ is the number of injected faults. The result consists in additions, subtractions, and multiplications of those formal variables (and inputs). Such expression is a multivariate polynomial.

If the inputs are fixed, then the polynomial has only n formal variables. We call it $P(\widehat{x_1}, \ldots, \widehat{x_n})$.

For now, let us assume that $n = 1$, i.e., that we face a single fault. Then P is a monovariate polynomial. Its degree is the multiplicative depth of $\widehat{x_1}$ in the result.

A fault is not detected (i.e., the algorithm $\langle \mathtt{a} \rangle_{r,\Gamma}$ would return neither the same result as a nor $\perp$) if and only if:

- $P(\widehat{x_1}) = P(x_1) \mod r$, whereas
- $P(\widehat{x_1}) \neq P(x_1) \mod p$.

Notice that the second condition is superfluous insofar, as if it is negated the effect of the fault does not alter the result in $\mathbb{Z}_p$ nor $\mathbb{Z}_r$, and hence in $\mathbb{Z}_{pr}$ either.

Non-detection probability is inversely proportional to r. As the faulted values $\widehat{x_1}$ are uniformly distributed over $\mathbb{Z}_{pr}$, the non-detection probability $\mathbb{P}_{\text{n.d.}}$ is given by:

$$\mathbb{P}_{\text{n.d.}} = \frac{1}{pr-1} \cdot \sum_{\widehat{x_1} \in \mathbb{Z}_{pr} \setminus \{x_1\}} \delta_{P(\widehat{x_1}) \,=\, P(x_1) \mod r} \tag{8.1}$$

$$= \frac{1}{pr-1} \cdot \left(-1 + p \sum_{\widehat{x_1}=0}^{r-1} \delta_{P(\widehat{x_1}) \,=\, P(x_1) \mod r} \right) . \tag{8.2}$$

In this equation, $\delta_{\text{condition}}$ is equal to 1 (resp. 0) if the condition is true (resp. false). We can explicit the steps between lines 8.1 and 8.2:

$$\begin{aligned}
&\textstyle\sum_{\widehat{x_1} \in \mathbb{Z}_{pr} \setminus \{x_1\}} \delta_{P(\widehat{x_1}) \,=\, P(x_1) \mod r} \\
&\quad = -1 + \textstyle\sum_{\widehat{x_1} \in \mathbb{Z}_{pr}} \delta_{P(\widehat{x_1}) \,=\, P(x_1) \mod r} \\
&\quad = -1 + \textstyle\sum_{k=0}^{p-1} \sum_{\widehat{x_1}=kr}^{(k+1)r-1} \delta_{P(\widehat{x_1} \bmod r) \,=\, P(x_1) \mod r} \\
&\quad = -1 + \textstyle\sum_{k=0}^{p-1} \sum_{\widehat{x_1}=0}^{r-1} \delta_{P(\widehat{x_1}) \,=\, P(x_1) \mod r} \\
&\quad = -1 + p \textstyle\sum_{\widehat{x_1}=0}^{r-1} \delta_{P(\widehat{x_1}) \,=\, P(x_1) \mod r} \ .
\end{aligned}$$

Let $\widehat{x_1} \in \mathbb{Z}_r$, if $P(\widehat{x_1}) = P(x_1) \mod r$, then $\widehat{x_1}$ is a root of the polynomial $\Delta P(\widehat{x_1}) = P(\widehat{x_1}) - P(x_1)$ in $\mathbb{Z}_r$. We denote by $\#\text{roots}(\Delta P)$ the number of roots of ΔP over $\mathbb{Z}_r$. Thus 8.2 computes $(p \times \#\text{roots}(\Delta P) - 1)/(pr-1) \approx \#\text{roots}(\Delta P)/r$.

Study of the proportionality constant. A priori, bounds on this value are broad since $\#\text{roots}(\Delta P)$ can be as high as the degree of ΔP in $\mathbb{Z}_r$, i.e., $\min(d, r-1)$. However, in practice, ΔP looks like a random polynomial over the finite field $\mathbb{Z}_r$, for several reasons:

- inputs are random numbers in most cryptographic algorithms, such as probabilistic signature schemes,
- the coefficients of ΔP in $\mathbb{Z}_r$ are randomized owing to the modular reduction by r.

In such case, the number of roots is very small, despite the possibility of d being large. See for instance [Leo06] for a proof that the number of roots tends to 1 as $r \to \infty$. Interestingly, the random polynomials are still friable (i.e., they are clearly not irreducible) in average, but most factors of degree greater than one happen not to have roots in $\mathbb{Z}_r$. Thus, we have $\mathbb{P}_{\text{n.d.}} \gtrsim \frac{1}{r}$, meaning that $\mathbb{P}_{\text{n.d.}} \geq \frac{1}{r}$ but is close to $\frac{1}{r}$.

It is interesting to study the theoretical upper bound on the number of roots in practical cases. Leont'ev proved in [Leo06] that if P is a random polynomial in $\mathbb{F}_p$ then $\#\text{roots}(P) \sim \text{Poisson}(\lambda = 1)$, i.e., $\mathbb{P}(\#\text{roots}(P) = k) = \frac{1}{ek!}$. In the case of $\Delta P \bmod r$, we know that there is always at least one root, when $\widehat{x_1} = x_1$, so we can rewrite $\Delta P(\widehat{x_1}) = P(\widehat{x_1}) - P(x_1) = R(\widehat{x_1}) \cdot a(\widehat{x_1} - x_1)$, where a is some constant, and R is indeed a random polynomial of degree $r-2$, owing to the modular reduction of ΔP by r. So we know that $\#\text{roots}(\Delta P) = 1 + \#\text{roots}(R)$, hence $\mathbb{P}(\#\text{roots}(\Delta P) = k) = \mathbb{P}(\#\text{roots}(R) = k-1)$, which is 0 if $k = 0$ and $\frac{1}{e(k-1)!}$ otherwise. We want the maximum value of k which has a "plausible" probability, let us say that is 2^{-p}, e.g., 2^{-256}. Since the values of a Poisson distribution of parameter $\lambda = 1$ are decreasing, we are looking for k such that: $\mathbb{P}(\#\text{roots}(R) = k-1) = \frac{1}{e(k-1)!} \leq 2^{-256}$. This would suggest that $k \gtrsim 58$.

This result means that $\mathbb{P}_{\text{n.d.}}$ is predicted to be at most $\frac{57}{r}$, with r being at least a 32-bit number, i.e., that $\mathbb{P}_{\text{n.d.}}$ is at maximum $\approx 2^{-26}$, and that this worst-case scenario has a probability of $\approx 2^{-256}$ of happening, in theory. Fig. 8.3 shows typical number of roots (obtained with SAGE) for practical cases in ECC, and compare them to the theoretical predictions. In this figure, we chose values of k of the form $2^j - 1$, which maximize the number of operations, and thus the size and degree of the resulting ΔP polynomials. For each value of k, we expressed the polynomial ΔP corresponding to the ECSM $[k]G$, and did so for a thousand random G. We then plotted for $i = 0$ to 8 the number of $[k]G$ for which $\#\text{roots}(\Delta P) = i + 1$ divided by 1000, that is the estimated probability $\hat{\mathbb{P}}(\#\text{roots}(\Delta P) = i + 1)$. Let us denote by Z the Boolean random variable which is equal to one if ΔP has a $(i+1)$ roots, and zero otherwise. Our estimation of $\hat{\mathbb{P}}(\#\text{roots}(\Delta P) = i + 1)$ is thus the expectation of $\frac{1}{1000} \sum_{j=1}^{1000} Z_j$. This random variable follows a binomial distribution, of mean $p = \mathbb{P}(\#\text{roots}(\Delta P) = i + 1)$ and variance $p(1-p)/1000$. The later values are used for the error bars ($[p - \sqrt{p(1-p)/1000}, p + \sqrt{p(1-p)/1000}]$).

The two graphs in Fig. 8.3 correspond to two corner-cases:

1. $k = 3 = (11)_2$: the number of roots is small because the polynomial degree is small (it is 13). (recall that $\#\text{roots}(P)$ cannot exceed the degree of P).
2. $k = 15 = (1111)_2$: the number of roots is also small, but this times because the result of Leont'ev applies. Indeed, the degree is 7213, thus the polynomial is much more random-looking.

Actually, it is computationally hard to count the roots of polynomials of degree greater than 7213. But it can be checked that the degree of the polynomials is growing exponentially with k. This is represented in Fig. 8.4, where we see that the degree is about equal to $k^{3.25}$ (of course, when k has a large Hamming weight, as in $(11\ldots1)_2$, the degree is larger than when k is hollow, as in $(10\ldots0)_2$). In particular, the polynomial ΔP reaches degree 2^{32} (resp. 2^{64}) when k has about 10 (resp. 18) bits. Thus, modulo r (recall Eqn. 8.2), the polynomial ΔP has maximal degree as long as the fault is injected before the last 10 (resp. 18) elliptic curve operations when r fits on 32 bits (resp. 64 bits).

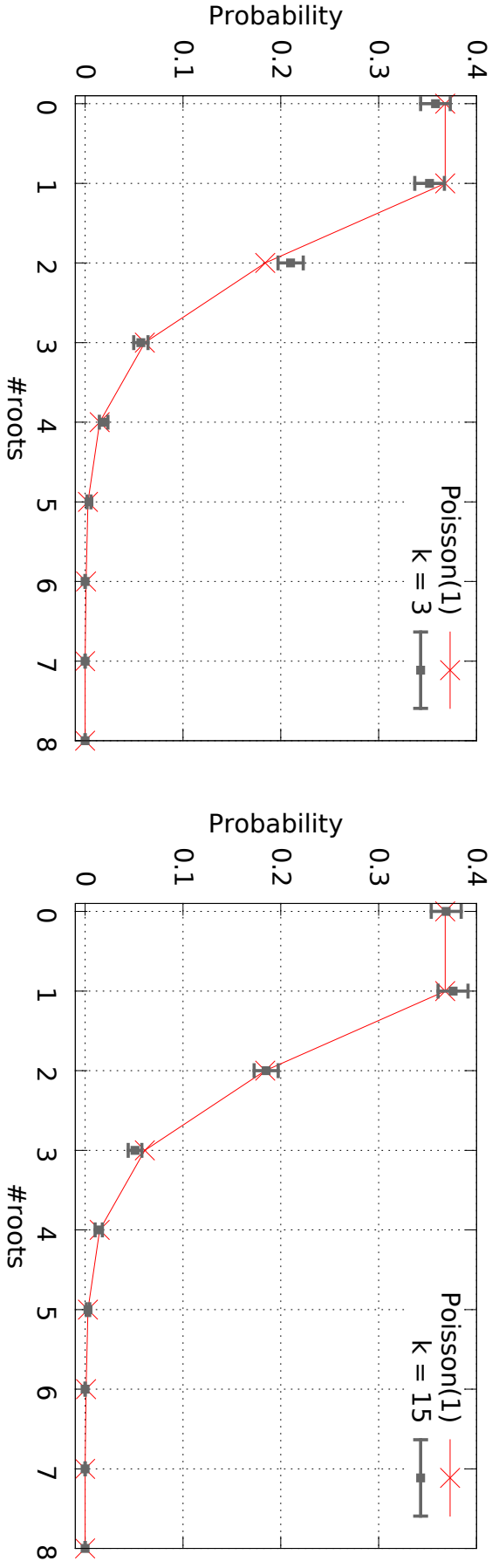


Figure 8.3: #roots probability for ECSM $[k]G$.

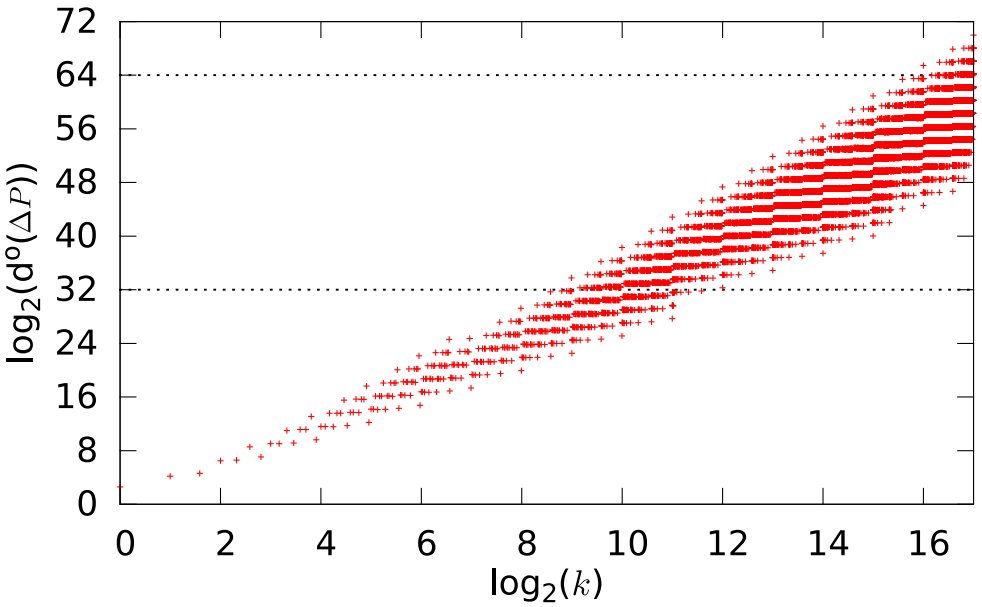


Figure 8.4: Degree of the polynomial ΔP against the value of k (in log-log scale).

The same law applies to multiple faults. In the case of multiple faults ($n > 1$), then the probability of non-detection generalizes to:

$$\begin{aligned}
\mathbb{P}_{\text{n.d.}} &= \frac{1}{(pr-1)^n} \cdot \sum_{\widehat{x_1},\ldots,\widehat{x_n} \in \mathbb{Z}_{pr}\setminus\{x_1\}\times\ldots\times\mathbb{Z}_{pr}\setminus\{x_n\}} \delta_{P(\widehat{x_1},\ldots,\widehat{x_n})=P(x_1,\ldots,x_n) \mod r} \\
&= \frac{1}{(pr-1)^n} \cdot \sum_{\widehat{x_2},\ldots,\widehat{x_n} \in \prod_{i=2}^{n} \mathbb{Z}_{pr}\setminus\{x_i\}} \left[\sum_{\widehat{x_1} \in \mathbb{Z}_{pr}\setminus\{x_1\}} \delta_{P(\widehat{x_1},\widehat{x_2},\ldots,\widehat{x_n})=P(x_1,x_2,\ldots,x_n) \mod r} \right] \qquad (8.3) \\
&= \frac{1}{(pr-1)^n} \cdot \sum_{\widehat{x_2},\ldots,\widehat{x_n} \in \prod_{i=2}^{n} \mathbb{Z}_{pr}\setminus\{x_i\}} \Big[p \times \#\text{roots}(\Delta P) - 1 \Big] \\
&= \frac{1}{(pr-1)^n} \cdot (pr-1)^{n-1} \left[p \times \#\text{roots}(\Delta P) - 1 \right] \\
&= \frac{p \times \#\text{roots}(\Delta P) - 1}{pr-1} \ . \qquad (8.4)
\end{aligned}$$

Indeed, at line 8.3, the expression in square brackets is the number of roots in $\mathbb{Z}_r$ of a monovariate polynomial in $\widehat{x_1}$, namely:

$$\widehat{x_1} \ \mapsto \ P(\widehat{x_1}, \widehat{x_2}, \ldots, \widehat{x_n}) - P(x_1, x_2, \ldots, x_n) \bmod r \ . \qquad (8.5)$$

As per the previous analysis in the case of a single fault ($n = 1$), the quantity in square brackets is equal to $p \times \#\text{roots}(\Delta P) - 1$, where ΔP is a random polynomial in $\mathbb{Z}_r$, as that of Eqn. 8.5. Therefore, the probability not to detect a fault when $n > 1$ is identical to that for $n = 1$. Thus, we also have $\mathbb{P}_{\text{n.d.}} \approx \frac{1}{r}$ in the case of multiple faults.

This concludes the security proof of Prop. 8.5. Indeed, we have shown that in the case of fault injection attacks, $\langle \mathtt{a} \rangle_{r,\Gamma}$ returns either the same result as $\mathtt{a}$ or $\perp$ with probability $1 - \mathbb{P}_{\text{n.d.}} \approx \frac{r-1}{r}$. □

Example 8.2 ($\mathbb{P}_{\text{n.d.}}$ for CRT-RSA)**.** From Prop. 8.5's proof, we can derive that for proven CRT-RSA countermeasures such as [ABF+02, Vig08a, RG14c], we have $\mathbb{P}_{\text{n.d.}} = \frac{1}{r}$.
Indeed, in CRT-RSA, the computation mainly consists of two exponentiations. In an exponentiation, ΔP takes on the form $\widehat{m}^k \cdot m^{d-k} - m^d = (\widehat{m}^k - m^k) \cdot m^{d-k}$. Assuming the message $m \neq 0$, we have $\#\text{roots}(\Delta P) = 1$ (that is $\widehat{m} = m \mod r$), hence a non-detection probability of $\frac{1}{r}$ (in the case RSA is computed with CRT). Otherwise, after the Garner recombination [Gar65] ΔP is of the form $m^{d_q} + q \cdot (i_q \cdot (m^{d_p-k}\widehat{m}^k - m^{d_q}) \bmod p) - m^{d_q} + q \cdot (i_q \cdot (m^{d_p} - m^{d_q}) \bmod p) = q \cdot (i_q \cdot (m^{d_p-k}\widehat{m}^k - m^{d_p}))$, if the fault is on the p part; or $m^{d_q-k}\widehat{m}^k + q \cdot (i_q \cdot (m^{d_p} - m^{d_q-k}\widehat{m}^k) \bmod p) - m^{d_q} + q \cdot (i_q \cdot (m^{d_p} - m^{d_q}) \bmod p) = (m^{d_q-k}\widehat{m}^k - m^{d_q}) + q \cdot (i_q \cdot (m^{d_q-k}\widehat{m}^k - m^{d_q}))$, if it is on the q part.
We conclude that $\#\text{roots}(\Delta P)$ is still 1 in both cases and thus that $\mathbb{P}_{\text{n.d.}} = \frac{1}{r}$.

It can be noticed that this result had been used in most previous articles dealing with fault protection on CRT-RSA without being formally proved. So, we now formally confirm those results were indeed correct.

Example 8.3 ($\mathbb{P}_{\text{n.d.}} = \frac{\alpha}{r}$ with $\alpha > 1$)**.** Let us assume the computation $P(a, b, c) = (a + b) \cdot (b + c)$. If a single fault strikes b, then the polynomial ΔP is equal to $P(a, \hat{b}, c) - P(a, b, c) \mod r$. Its degree is equal to 2, and has 2 distinct roots provided $b \neq -(a + c)/2 \mod r$, or 1 double root otherwise. Thus, in the general case where the nominal inputs satisfy $b \neq -(a + c)/2 \mod r$ (which occurs also with probability $\frac{1}{r}$), the non-detection probability is $\frac{2}{r}$. Namely, the $2p - 1$ values of $\hat{b} \in \mathbb{Z}_{pr}$ causing an undetected fault are $b + kr$, with $k \in \{1, \ldots, p - 1\}$, and $-(a + c)/2 + lr$, with $l \in \{0, \ldots, p - 1\}$.

In conclusion, examples 8.2 and 8.3 illustrate the security property: indeed, $\mathbb{P}_{\text{n.d.}}$ is inversely proportional to r, with a proportionality constant which depends on the specific algorithm. The purpose of the next section is to show that the product $\mathbb{P}_{\text{n.d.}} \times r$ is constant in practice. Moreover, we explicit this constant for ECSM computations.

8.4 Practical Case Study: Protecting an ECSM Implementation

In order to practically validate our theoretical results, we implemented an ECSM algorithm on an ARM Cortex-M4 microcontroller (specifically the STM32F417VGT6). Our choice of implementing an ECSM rather than a full ECC cryptosystem is based on the lower complexity of such an implementation coupled to the fact that virtually all fault attacks on ECC based algorithms rely on faulting an ECSM. We protected our implementation using the entanglement methodology described in Sec. 8.2.2, and then performed a fault injection attack campaign on it.

8.4.1 Setup

The elliptic curve we used is P-192 from NIST [U.S13, D.1.2.1]. It uses a Weierstrass equation, that is $y^2 = x^3 + ax + b$, over the field $\mathbb{F}_p$. Parameter values are listed in

Field order	$p =$ `0xfffffffffffffffffffffffffffffffeffffffffffffff`
Curve equation coefficients	$a =$ `0xfffffffffffffffffffffffffffffffefffffffffffffc`
	$b =$ `0x64210519e59c80e70fa7e9ab72243049feb8deecc146b9b1`
Point coordinates	$Gx =$ `0x188da80eb03090f67cbf20eb43a18800f4ff0afd82ff1012`
	$Gy =$ `0x07192b95ffc8da78631011ed6b24cdd573f977a11e794811`
Point order	$ord =$ `0xffffffffffffffffffffffff99def836146bc9b1b4d22831`

Figure 8.5: Parameters of our ECSM implementation (namely NIST P-192).

Fig. 8.5. Gx and Gy define the coordinates of a ord-order point of the P-192 elliptic curve. We implement the ECSM algorithm with projective coordinates, as depicted in Alg. 8.3. We code it in C, using the arbitrary precision arithmetic library GMP[3].

Algorithm 8.3: Protected ECSM implementation

Input : G, $k \in Z_p$
Output: $[k]G$ or $\perp$

1 $(X, Y, Z) = [k]G$ in $\mathbb{Z}_{pr}$
2 $(x, y, z) =$ projective-to-affine-coordinates(X, Y, Z) `// with Prop. 8.1`
3 $(X', Y', Z') = [k]G$ in $\mathbb{Z}_r$
4 $(x', y', z') =$ projective-to-affine-coordinates(X', Y', Z')
5 **if** $(x \equiv x' \mod r)$ *and* $(y \equiv y' \mod r)$ *and* $(z \equiv z' \mod r)$ **then**
6 | **return** $(x \bmod p, y \bmod p, z \bmod p)$
7 **else**
8 | **return** $\perp$
9 **end**

Figure 8.6 shows an architectural overview of the Electromagnetic Fault Injection (EMFI) analysis platform we used for our experiments. The platform includes a signal generator able to generate pulses of 1.5 ns width amplified by a broadband class A amplifier (400 MHz, 300 Watt max), and an Electromagnetic (EM) probe. An oscilloscope and a data timing generator are also present, so that we can precisely (with 1 ps precision) control the delay before the injection. All experiments have been performed at a settled spatial location of the EM probe relative to the ARM microcontroller: a fixed position and a fixed angular orientation. A boundary-scan (also known as JTAG) probe has been used to dump internal registers and memory contents after injection (for attack analysis purpose only).

We manually explored the effect of different width and power of the EM pulse, and chose values which maximize the faulting success rate. Then, we manually tuned the delay before the injection happens in order to maximize the probability of obtaining an exploitable fault while the protection is disabled (when $r = 1$, as explained in Sec. 8.4.2).

8.4.2 Method

In order to assess our theoretical results, we performed multiple attack campaigns with different values for r (the order of the small ring). This allowed us to verify our theoretical prediction on the probability of non-detection $\mathbb{P}_{\text{n.d.}}$ to be inversely

[3]We used the `mini-gmp` implementation for easy portability onto the ARM microcontroller.

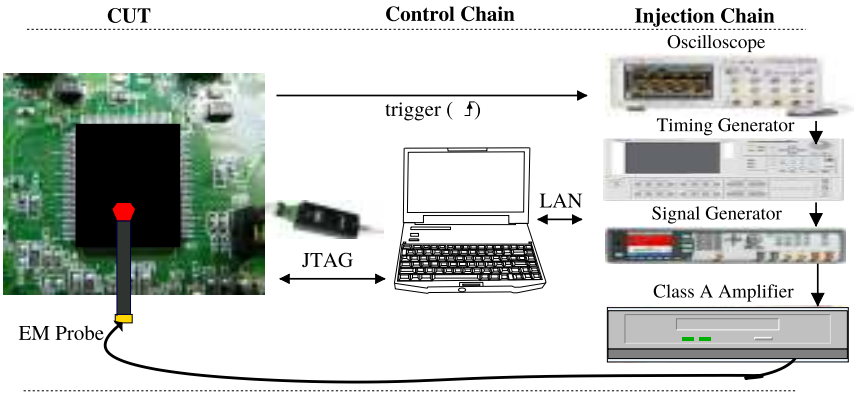


Figure 8.6: EMFI platform.

proportional to r (see Sec. 8.3.4). At the same time we were able to measure the cost of the countermeasure and confirm that the size of r is a security parameter that trades speed off for security.

- The value $r = 1$ basically means that there is no countermeasure, since $\mathbb{Z}_r = \mathbb{Z}_1 = \{0\}$. It helps verify that the platform is indeed injecting faults effectively, i.e., that most of the fault injection attempts are successful.
- The small values of r (on 8 to 16 bits) aim at verifying that the probability of detection / non-detection follow our theoretical prediction.
- The values of r on 32 and 64 bits represent realistic values for an operational protection.

Each value of r is chosen to be the largest prime number of its size. That is, if n is the size of r in bits, then r is the largest prime number such that $r < 2^n$.

8.4.3 Security Results

The table presented in Tab. 8.1 shows the security assessment of the entanglement countermeasure. For each value of r (lines of the table) we ran and injected random faults in approximately[4] 1000 ECSM $[k]G$ using a random 192-bit k. In total, the execution of the tests we present took approximately 6 hours. The results of our attack campaign are depicted in the last four columns.

- Correct results for which there is no error detection are simply fault injections without effect (*true negatives*).
- Correct results for which an error is detected are *false positives*, and should be minimized. Those false positive alarms are annoyances, as they warn despite no secret is at risk security-wise.
- The incorrect results for which an error is detected (*true positives*) should appear with probability $1 - \frac{1}{r}$.
- The incorrect results for which there is no error detection are *false negatives*, and should really be minimized: otherwise, the countermeasure is bypassed without notice and sensitive information may leak.

Once renormalized to remove the true negatives, the last column of Tab. 8.1 (false negatives) represents the non-detection probability $\mathbb{P}_{\text{n.d.}}$. The relationship between r and $\mathbb{P}_{\text{n.d.}}$ is plotted in Fig. 8.7.

[4] A bit less in practice: a few attempts were lost due to communication errors between our computer and the JTAG probe `gdb-server`.

Table 8.1: Entanglement security assessment results.

r value	r size (bit)	Positives (%)		Negatives (%)	
		true	false	true	false
1	1	0.00	0.00	2.74	97.26
251	8	63.65	0.00	2.56	33.79
1021	10	89.09	0.00	2.96	7.95
2039	11	98.82	0.00	0.00	1.18
4093	12	97.61	0.00	1.91	0.48
65521	16	97.79	0.00	2.21	0.00
4294967291	32	97.19	0.00	2.81	0.00
18446744073709551557	64	99.79	0.00	0.21	0.00

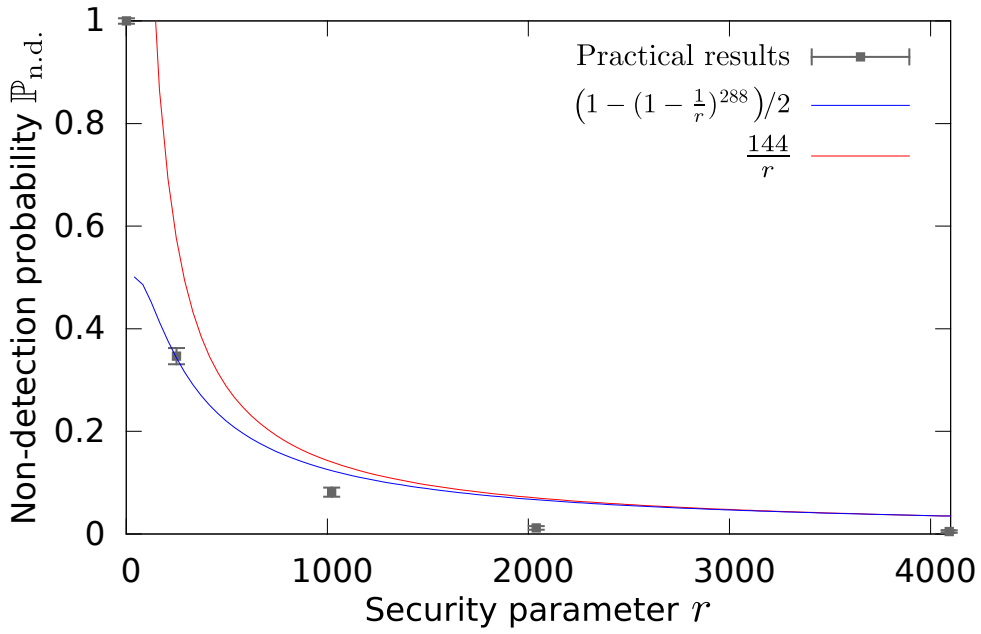


Figure 8.7: Relationship between $\mathbb{P}_{\text{n.d.}}$ and r.

Proposition 8.6 (ECSM proportionality factor)**.** *In the case of ECSM on curve P-192, the proportionality constant is* ≈ 144.

Lemma 8.1. *During ECSM computation (See Alg. 8.2), if the coordinates of the intermediate point becomes a multiple of r, they stay so until the end. Notably, if the Y coordinate becomes a multiple of r, then all coordinates become so in two loop iterations.*

Proof. Let $Q = (X_1, Y_1, Z_1)$, the point doubling $2Q = (X_3, Y_3, Z_3)$ (line 3 of Alg. 8.2) is computed as:

$$X_3 = 2Y_1Z_1(A^2 - 8X_1Z_1Y_1^2) \tag{8.6}$$
$$Y_3 = A(12X_1Z_1Y_1^2 - A^2) - 8Z_1^2Y_1^4 \tag{8.7}$$
$$Z_3 = 8Z_1^3Y_1^3 \tag{8.8}$$

where $A = 3(X_1^2 + 2aZ_1(X_1 + Z_1))$.

Let $P = (X_2, Y_2, Z_2)$, the point addition $Q + P = (X_3, Y_3, Z_3)$ (line 4 of Alg. 8.2) is computed as:

$$X_3 = BC \tag{8.9}$$
$$Y_3 = A(X_1Z_2B^2 - C) - Y_1Z_2B^3 \tag{8.10}$$
$$Z_3 = Z_1Z_2B^3 \tag{8.11}$$

where $A = Y_2Z_1 - Y_1Z_2$, $B = X_2Z_1 - X_1Z_2$, and $C = Z_1Z_2A^2 - (X_1Z_2 + X_2Z_1)B^2$.

In both cases, it is trivial to see that if $(X_1, Y_1, Z_1) = (0, 0, 0)$ in $\mathbb{Z}_r$ then $(X_3, Y_3, Z_3) = (0, 0, 0)$ in $\mathbb{Z}_r$.

Now, we can see that except for Z_3 in the doubling operation which is only composed of multiplications (Eqn. 8.8), any coordinates might become a multiple of r. We estimate that such an event occurs with probability $\frac{1}{r}$. We do not consider the case where two coordinates separately become multiple of r at the same time as it is deemed unlikely in practice (the probability is $\approx 1/r^2 \ll 1/r$).

In the doubling operation: 1. if $X_1 \equiv 0 \mod r$, the other coordinates are not contaminated as they have sum terms depending solely on Y_1 and Z_1; 2. if $Y_1 \equiv 0 \mod r$, we see that X_3 and Z_3 immediately become multiples of r (Eqn. 8.6 and 8.8), and this will lead Y_3 to become so at the next double operation (Eqn. 8.7); 3. as said before, the Z coordinate cannot become a multiple of r on its own.

In the addition operation, where the coordinates of the point P are assumed coprime with r, we can see that all three equations have at least one sum term that do not depend on one of the three coordinates of Q. However, we can see that if both X_1 and Z_1 are multiples of r then X_3 and Z_3 will stay so, making sure that the Y coordinate will become so at the next double operation as said in the second of the previous bullet points. Moreover, we remark that if the Z coordinate becomes a multiple of r, it is necessarily because B does, which implies that the X coordinate also becomes a multiple of r. □

Proof. Using Lem. 8.1, we can now prove Prop. 8.6. If a fault occurs and then either 1. the Y coordinate becomes a multiple of r, or 2. the Z coordinate does, then the fault is not detected. Indeed in these cases, both the result in $\mathbb{Z}_{pr} \bmod r$ and the result in $\mathbb{Z}_r$ are null. This happens with probability $\frac{1}{2}(1 - \frac{1}{r})^{192+96}$, where 192 is the number of double operations (accounting for event 1) and 96 the average number of

Table 8.2: Entanglement performance results.

r value	r size (bit)	time (ms) $\mathbb{Z}_{pr}$	$\mathbb{Z}_r$	test	overhead
1	1	683	24	$\ll 1$	$\times 1.04$
251	8	883	91	$\ll 1$	$\times 1.43$
1021	10	899	100	$\ll 1$	$\times 1.46$
2039	11	902	197	$\ll 1$	$\times 1.61$
4093	12	903	197	$\ll 1$	$\times 1.61$
65521	16	883	189	$\ll 1$	$\times 1.56$
4294967291	32	832	172	$\ll 1$	$\times 1.47$
18446744073709551557	64	996	246	$\ll 1$	$\times 1.82$

addition operations (accounting for event 2). The $\frac{1}{2}$ factor accounts for the faulting to follow event 1 or 2, where it is likely to be absorbed by a subsequent product with a multiple of r. For large values of r, such events are rare: their probability is $\frac{1}{2}(1-\frac{1}{r})^{288} = \frac{288/2}{r} + O(\frac{1}{r})$. □

As Tab. 8.1 shows, practical values of r are sufficiently large for the latter equality to be true, and thus for the security to be highly efficient.

8.4.4 Performance Results

The table presented in Tab. 8.2 shows the cost of the entanglement countermeasure in terms of speed[5]. For each value of r (lines of the table) we list the execution time of the ECSM computation in $\mathbb{Z}_{pr}$, of the one in $\mathbb{Z}_r$, of the test (comprising the extraction modulo r of the $\mathbb{Z}_r$ value entangled in the result of the computation in $\mathbb{Z}_{pr}$ and its comparison with the result of the computation in $\mathbb{Z}_r$), and eventually the overhead of the countermeasure.

In the unprotected implementation, the ECSM computation in $\mathbb{Z}_p$ took 683 ms (which naturally corresponds to the 683 ms in $\mathbb{Z}_{pr}$ when $r = 1$ as shown in Tab. 8.2, except that there is no need for the 24 ms needed by the computation in $\mathbb{Z}_r$ which is mathematically trivial, but not optimized by `gcc`). We can see that when r is on 32 bits, the alignment with `int` makes `mini-gmp` faster, resulting in the protected algorithm running for 1004 ms, incurring a factor of only ≈ 1.47 in the run time compared to the unprotected algorithm. This is a particularly good performance result. Indeed, the curve P-192 that we use is among the smallest standardized curves, and the performance factor is directly tied to the increase in the size of the ring in which the computations are performed: when $\mathbb{F}_p$ grows, the countermeasure gets cheaper as $\frac{\log_2(pr)}{\log_2(p)}$ will be smaller.

Finally, we underline the advantage of choosing r the largest prime smaller than a given power of two, so as to increase the detection probability at given cost in terms of code speed. Notice that this is also the strategy used by the P-192 ECC curve, where p is very close to a power of two.

[5]Note that we compiled the code with `gcc -O0` option.

8.5 Conclusion and Perspectives

In this chapter, we have shown how well-known techniques to protect CRT-RSA can be mobilized to protect any kind of asymmetric cryptographic algorithms.

We presented a protection method that we call *entanglement* and we illustrated its *genericity* (it works against all DFA attacks) and its *universality* (it applies to all asymmetric cryptography). The protection consists in introducing a redundant computation that can be verified cost-effectively, and entangling it with the main computation into an over-structure (direct product), so that the integrity of the computation can be verified before returning its result.

We proved the correctness of the induced code transformation (i.e., it preserves the semantics), and we presented a compiler which automates the entanglement protection. We also proved a rigorous security property and computed the fault non-detection probability of the countermeasure. This result allowed us to confirm the figures announced for CRT-RSA and to provide figures for ECC and every asymmetric cryptography computation in general.

We used our compiler to generate a protected ECSM algorithm. To our best knowledge, this is the first ECC implementation to be provably protected against fault injection attacks. We used it to show that the cost of the entanglement protection scheme is extremely reasonable: **with a 32-bit value for the security parameter, the code is less than** 1.5 **times slower**. A systematic fault injection campaign revealed that it is also very efficient: using the same 32-bit security parameter, **100% of the fault injections were detected**.

As a perspective, it would be beneficial to adapt our proofs to other fault models than randomizing fault. An interesting idea would be to measure the additional cost of performing intermediate verifications against safe-error attacks. Intermediate verifications would also decrease the constant of Proposition 8.6.

A natural extension would be the application of the entanglement protection to pairing. In this case, some computations must be carried out in *field extensions*, typically with embedding degree $m = 12$.

Finally, the security parameter r can be chosen randomly at execution time. As already pointed out in [BV07], this can make a natural protection against side-channel analyses. The formalization of this nice side-effect of the entanglement protection would be welcomed.

Acknowledgment

The authors would like to thank Gilles Barthe, François Dupressoir, and Pierre-Yves Strub from IMDEA Software Institute for their guidance concerning the correctness proof (Prop. 8.4).

0x9

Conclusions and Perspectives

The two main goals of my thesis were:

- to show that formal methods can be used to prove not only the principle of countermeasures according to a model but also their implementation, as it is where the physical vulnerabilities are exploited; and
- the proof and the automation of the protection techniques themselves, because handwritten security code is error-prone.

I hope that my research work is a step forward in these directions. There are many things left to do based on it. I first recall my contributions and then lay out some perspectives.

9.1 Conclusions

Physical attacks can be classified into two distinct categories. Passive attacks, where the attacker only reads information that leaks through *side channels* (such as power consumption or electromagnetic emanations). And active attacks, where the attacker tampers with the system to have it reveal secrets through its "normal" output channel. Therefore, I have pursued my goals in both settings: on a countermeasure that diminishes side-channel leakage, and on countermeasures which detect *fault injection* attacks.

As there already exists a rigorous security property for protections against side-channel leakage [MOP06, Chp. 7 & 9], my contributions concentrate on formal methods for design and verification of protected implementations of algorithms. In chapter 4, there was:

- A methodology to protect an implementation against side-channel attacks by generating an improved version of it which has a null side-channel signal-to-noise ratio, as its leakage is made constant (in particular, it does not depend on the secret values) by using a *software* Dual-rail with Precharge Logic (DPL) balancing protocol.
- A proof that the induced code transformation is correct (semantic preserving).
- A compiler named paioli that takes assembly code and protect it by applying the aforementioned code transformation.
- A tool, integrated with the compiler, able to independently prove that this constant leakage property holds for a given implementation, which can be used to assess the security of the output of the compiler.

To my best knowledge, paioli is the first tool which can automatically protect an implementation against side-channel attacks by provably balancing its leakage.

In contrast, the definition of the security objectives was not clearly established for fault injection attacks before I started my PhD. Many countermeasures have been published without any proofs, as the necessary properties to prove were not formally expressed. It is only during my PhD that so-called "attack success conditions" were introduced. In chapters 5, 6, and 7, there was:

- A theoretical framework based on arithmetic to evaluate existing countermeasure with respect to these attack success conditions.
- A tool named finja which does symbolic evaluation by rewriting according to arithmetic rules, which allowed to study, prove, and simplify existing countermeasures.
  Example: the original Vigilant's CRT-RSA[1] countermeasure was broken, used 9 tests, and needed 5 random numbers; our final version can be proved, works with less code, uses 3 tests, and needs only 1 random number.
- A classification of a family of existing countermeasures which allowed to extract protection principles from the employed techniques.
- A method for designing a countermeasure that can resist an arbitrary (but fixed) number of faults.

The method I have developed and the finja tool I have implemented have also been used by other researchers to study another family of countermeasures and to obtain new results [Kis14].

I have also noticed that all the countermeasures I have studied are variations of optimization for the same base technique consisting in verifying the integrity of the computation using a form of redundancy. This technique is independent of the algorithm it is applied to and of the attack success conditions, as it only depends on the mathematical structure of the sensitive data, namely modular arithmetic. I have thus proposed a property of resistance against fault injection attacks which goes beyond attack success conditions. In chapter 8, there was:

- An abstraction of the principle of the CRT-RSA countermeasures against Bell-CoRe attacks into a protection scheme that we call *entanglement*, and that is shown to be applicable to all of asymmetric cryptographic computations.
- A provably correct compiler named enredo which automates the protection and allowed to obtain protected implementations of cryptographic algorithms for which no countermeasures were known until now, but which were already victims of fault injection attacks.

An additional advantage of the enredo compiler is that it allows to study the trade-off between security and complexity.

9.2 Perspectives

My work on side-channel attacks opens a few perspectives. An obvious idea would be to use the same methods I developed to protect other cryptographic algorithms, to see how much the countermeasure cost and if we can generalize the low cost that we observed on PRESENT. More importantly, it would be interesting to target other hardware platforms, where the characterization step might need to be different, and where the architecture adds new constraints on the countermeasure. For instance,

[1] "CRT" stands for "Chinese Remainder Theorem", which is a common optimization technique for RSA.

it is an open issue whether it is possible to use a software balancing countermeasure along with hardware caches without a prohibitive cost due to necessary locks.

My work on fault injection attacks also opens a few direct perspectives.

The main one is obviously to continue the linear progression of the research, i.e., to adapt the entanglement protection scheme and the enredo compiler to protect Elliptic-Curve Cryptography (ECC) pairing computations, which take place in field extensions. Martin Moreau, an M2 research intern that I co-supervise with Sylvain Guilley is currently tackling this perspective.

Another perspective is the formalization and the proof of the finja tool itself, i.e., a formal study of the arithmetic rewriting system leveraged by finja to compute its proofs in order to prove properties such as its termination or the fact that the exploration is complete. This proof work could be carried out in a proof assistant such as EasyCrypt [BGZB09], which would move the trust needed away from finja's code to a more established, more mature, and better scrutinized technology.

The classification work presented in chapter 7 only covers what I called "Shamir's family" of countermeasures and it would be interesting to study the countermeasures of "Giraud's family". After my publication at FDTC 2014, this work has been largely started by Agnès Kiss, under the supervision of Juliane Krämer, at TU Berlin [Kis14].

Finally, another important perspective would be to add chosen message attacks to the threat model and see what additions to the countermeasures would be necessary against these more powerful but still realistic attackers.

More broadly, I believe that the physical attacks and countermeasures on cryptography would benefit a lot from a wider use of formal methods and even more from the possible automation brought by them, and I hope that my thesis work will have a positive impact toward this goal.

0xA

Bibliography

[ABF+02] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and Jean-Pierre Seifert. Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures. In Burton S. Kaliski, Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2002.

[BCDG12] Alexandre Berzati, Cécile Canovas-Dumas, and Louis Goubin. A Survey of Differential Fault Analysis Against Classical RSA Implementations. In Marc Joye and Michael Tunstall, editors, *Fault Analysis in Cryptography*, Information Security and Cryptography, pages 111–124. Springer, 2012.

[BCO04] Éric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In *CHES*, volume 3156 of *LNCS*, pages 16–29. Springer, August 11–13 2004. Cambridge, MA, USA.

[BCS08] Eli Biham, Yaniv Carmeli, and Adi Shamir. Bug attacks. In *CRYPTO*, volume 5157 of *LNCS*, pages 221–240. Springer, 2008. Santa Barbara, CA, USA.

[BDF+14a] Gilles Barthe, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. Making RSA-PSS Provably Secure Against Non-Random Faults. *IACR Cryptology ePrint Archive*, 2014:252, 2014.

[BDF+14b] Gilles Barthe, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Jean-Christophe Zapalowicz. Synthesis of Fault Attacks on Cryptographic Implementations. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1016–1027. ACM, 2014.

[BDGN14a] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. NICV: Normalized Inter-Class Variance for Detection of Side-Channel Leakage. In *International Symposium on Electromagnetic Compatibility (EMC '14 / Tokyo)*. IEEE, May 12-16 2014. Session OS09: EM Information Leakage. Hitotsubashi Hall (National Center of Sciences), Chiyoda, Tokyo, Japan.

[BDGN14b] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Side-channel Leakage and Trace Compression Using Normalized Inter-class Variance. In *Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '14, pages 7:1–7:9, New York, NY, USA, 2014. ACM.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *Proceedings of Eurocrypt'97*, volume 1233 of *LNCS*, pages 37–51. Springer, May 11-15 1997. Konstanz, Germany. DOI: 10.1007/3-540-69053-0_4.

[Ben03] Charles H. Bennett. Notes on Landauer's principle, Reversible Computation and Maxwell's Demon. *Studies in History and Philosophy of Modern Physics*, 34:501–510, 2003. http://arxiv.org/abs/physics/0210005.

[BG13] Alberto Battistello and Christophe Giraud. Fault Analysis of Infective AES Computations. In Wieland Fischer and Jörn-Marc Schmidt, editors, *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, pages 101–107. IEEE, 2013. Santa Barbara, CA, USA.

[BGDSG+14] Johannes Blömer, Ricardo Gomes Da Silva, Peter Gunther, Juliane Krämer, and Jean-Pierre Seifert. A Practical Second-Order Fault Attack against a Real-World Pairing Implementation. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on*, pages 123–136, Sept 2014. Busan, Korea.

[BGL14] Johannes Blömer, Peter Günther, and Gennadij Liske. Tampering Attacks in Pairing-Based Cryptography. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on*, pages 1–7, Sept 2014. Busan, Korea.

[BGZB09] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella-Béguelin. Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009*, pages 90–101. ACM, 2009.

[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.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES*, volume 4727 of *LNCS*, pages 450–466. Springer, September 10-13 2007. Vienna, Austria.

[Bla] Bruno Blanchet. ProVerif: Cryptographic protocol verifier in the formal model. http://prosecco.gforge.inria.fr/personal/bblanche/proverif/.

[BNP07] Arnaud Boscher, Robert Naciri, and Emmanuel Prouff. CRT RSA Algorithm Protected Against Fault Attacks. In Damien Sauveron, Constantinos Markantonakis, Angelos Bilas, and Jean-Jacques Quisquater, editors, *WISTP*, volume 4462 of *Lecture Notes in Computer Science*, pages 229–243. Springer, 2007.

[BOS03] Johannes Blömer, Martin Otto, and Jean-Pierre Seifert. A new CRT-RSA algorithm secure against bellcore attacks. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM Conference on Computer and Communications Security*, pages 311–320. ACM, 2003.

[BOS06] Johannes Blömer, Martin Otto, and Jean-Pierre Seifert. Sign Change Fault Attacks on Elliptic Curve Cryptosystems. In Luca Breveglieri, Israel Koren, David Naccache, and Jean-Pierre Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography*, volume 4236 of *Lecture Notes in Computer Science*, pages 36–52. Springer Berlin Heidelberg, 2006.

[BS97] Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *CRYPTO*, volume 1294 of *LNCS*, pages 513–525. Springer, August 1997. Santa Barbara, California, USA. DOI: 10.1007/BFb0052259.

[BV07] Yoo-Jin Baek and Ihor Vasyltsov. How to Prevent DPA and Fault Attack in a Unified Way for ECC Scalar Multiplication - Ring Extension Method. In Ed Dawson and DuncanS. Wong, editors, *Information Security Practice and Experience*, volume 4464 of *Lecture Notes in Computer Science*, pages 225–237. Springer Berlin Heidelberg, 2007.

[CCGV13] Maria Christofi, Boutheina Chetali, Louis Goubin, and David Vigilant. Formal verification of a CRT-RSA implementation against fault attacks. *Journal of Cryptographic Engineering*, 3(3):157–167, 2013.

[CESY14] Cong Chen, Thomas Eisenbarth, Aria Shahverdi, and Xin Ye. Balanced Encoding to Mitigate Power Analysis: A Case Study. In *CARDIS*, Lecture Notes in Computer Science. Springer, November 2014. Paris, France.

[CFGR12] Claude Carlet, Jean-Charles Faugère, Christopher Goyet, and Guénaël Renault. Analysis of the algebraic side channel attack. *J. Cryptographic Engineering*, 2(1):45–62, 2012.

[CGM+10] Jean-Sébastien Coron, Christophe Giraud, Nicolas Morin, Gilles Piret, and David Vigilant. Fault Attacks and Countermeasures on Vigilant's RSA-CRT Algorithm. In Luca Breveglieri, Marc Joye, Israel Koren, David Naccache, and Ingrid Verbauwhede, editors, *FDTC*, pages 89–96. IEEE Computer Society, 2010.

[CGP+12] Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-Order Masking Schemes for S-Boxes. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 366–384. Springer, 2012.

[CHM11] Nicolas Courtois, Daniel Hulme, and Theodosis Mourouzis. Solving Circuit Optimisation Problems in Cryptography and Cryptanalysis. *IACR Cryptology ePrint Archive*, 2011:475, 2011. (Also presented in SHARCS 2012, Washington DC, 17-18 March 2012, on page 179).

[CJ05] Mathieu Ciet and Marc Joye. Practical fault countermeasures for chinese remaindering based RSA. In *Fault Diagnosis and Tolerance in Cryptography*, pages 124–131, Friday September 2nd 2005. Edinburgh, Scotland.

[CM09] Jean-Sébastien Coron and Avradip Mandal. PSS Is Secure against Random Fault Attacks. In *ASIACRYPT*, volume 5912 of *LNCS*, pages 653–666. Springer, December 6-10 2009. Tōkyō, Japan.

[Con13] Common Criteria Consortium. Common Criteria (*aka* CC) for Information Technology Security Evaluation (ISO/IEC 15408), 2013.
Website: http://www.commoncriteriaportal.org/.

[CPR07] Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side Channel Cryptanalysis of a Higher Order Masking Scheme. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *LNCS*, pages 28–44. Springer, 2007.

[CSS13] Zhimin Chen, Ambuj Sinha, and Patrick Schaumont. Using Virtual Secure Circuit to Protect Embedded Software from Side-Channel Attacks. *IEEE Trans. Computers*, 62(1):124–136, 2013.

[DFK+13] Goran Doychev, Dominik Feld, Boris Köpf, Laurent Mauborgne, and Jan Reineke. CacheAudit: A Tool for the Static Analysis of Cache Side Channels. *IACR Cryptology ePrint Archive*, 2013:253, 2013.

[DGRS09] Emmanuelle Dottax, Christophe Giraud, Matthieu Rivain, and Yannick Sierra. On Second-Order Fault Analysis Resistance for CRT-RSA Implementations. In Olivier Markowitch, Angelos Bilas, Jaap-Henk Hoepman, Chris J. Mitchell, and Jean-Jacques Quisquater, editors, *WISTP*, volume 5746 of *Lecture Notes in Computer Science*, pages 68–83. Springer, 2009.

[DH76] Whitfield Diffie and Martin Edward Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[DSE+12] Nicolas Debande, Youssef Souissi, Moulay Abdelaziz Elaabid, Sylvain Guilley, and Jean-Luc Danger. Wavelet Transform Based Pre-processing for Side Channel Analysis. In *HASP*, pages 32–38. IEEE, December 2nd 2012. Vancouver, British Columbia, Canada. DOI: 10.1109/MICROW.2012.15.

[EGG+12] Thomas Eisenbarth, Zheng Gong, Tim Güneysu, Stefan Heyse, Sebastiaan Indesteege, Stéphanie Kerckhof, François Koeune, Tomislav Nad, Thomas Plos, Francesco Regazzoni, François-Xavier Standaert, and Loïc van Oldeneel tot Oldenzeel. Compact Implementation and Performance Evaluation of Block Ciphers in ATtiny Devices. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *AFRICACRYPT*, volume 7374 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2012.

[Elg85] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *Information Theory, IEEE Transactions on*, 31(4):469–472, Jul 1985.

[EMFGL14] Nadia El Mrabet, Jacques J.A. Fournier, Louis Goubin, and Ronan Lashermes. A survey of fault attacks in pairing based cryptography. *Cryptography and Communications*, pages 1–21, 2014.

[Gar65] Harvey L. Garner. Number Systems and Arithmetic. *Advances in Computers*, 6:131–194, 1965.

[GCS+08] Sylvain Guilley, Sumanta Chaudhuri, Laurent Sauvage, Philippe Hoogvorst, Renaud Pacalet, and Guido Marco Bertoni. Security Evaluation of WDDL and SecLib Countermeasures against Power Attacks. *IEEE Transactions on Computers*, 57(11):1482–1497, nov 2008.

[GHMP05] Sylvain Guilley, Philippe Hoogvorst, Yves Mathieu, and Renaud Pacalet. The "Backend Duplication" Method. In *CHES*, volume 3659 of *LNCS*, pages 383–397. Springer, 2005. August 29th – September 1st, Edinburgh, Scotland, UK.

[Gir06] Christophe Giraud. An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis. *IEEE Trans. Computers*, 55(9):1116–1120, 2006.

[GM11] Tim Güneysu and Amir Moradi. Generic side-channel countermeasures for reconfigurable devices. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES*, volume 6917 of *LNCS*, pages 33–48. Springer, 2011.

[GMK12] Xiaofei Guo, Debdeep Mukhopadhyay, and Ramesh Karri. Provably secure concurrent error detection against differential fault analysis. Cryptology ePrint Archive, Report 2012/552, 2012. http://eprint.iacr.org/2012/552/.

[GV13] Aurore Guillevic and Damien Vergnaud. Genus 2 Hyperelliptic Curve Families with Explicit Jacobian Order Evaluation and Pairing-Friendly Constructions. In Michel Abdalla and Tanja Lange, editors, *Pairing-Based Cryptography — Pairing 2012*, volume 7708 of *Lecture Notes in Computer Science*, pages 234–253. Springer Berlin Heidelberg, 2013.

[HDD11] Philippe Hoogvorst, Jean-Luc Danger, and Guillaume Duc. Software Implementation of Dual-Rail Representation. In *COSADE*, February 24-25 2011. Darmstadt, Germany.

[HMER13] Karine Heydemann, Nicolas Moro, Emmanuelle Encrenaz, and Bruno Robisson. Formal Verification of a Software Countermeasure Against Instruction Skip Attacks. Cryptology ePrint Archive, Report 2013/679, 2013. http://eprint.iacr.org/2013/679.

[INR] INRIA. OCaml, a variant of the Caml language. http://caml.inria.fr/ocaml/index.en.html.

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private Circuits II: Keeping Secrets in Tamperable Circuits. In *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 308–327. Springer, May 28 – June 1 2006. St. Petersburg, Russia.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, August 17–21 2003. Santa Barbara, California, USA.

[JLQ99] Marc Joye, Arjen K. Lenstra, and Jean-Jacques Quisquater. Chinese Remaindering Based Cryptosystems in the Presence of Faults. *J. Cryptology*, 12(4):241–245, 1999.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001.

[Joy09] Marc Joye. Protecting RSA against Fault Attacks: The Embedding Method. In Luca Breveglieri, Israel Koren, David Naccache, Elisabeth Oswald, and Jean-Pierre Seifert, editors, *FDTC*, pages 41–45. IEEE Computer Society, 2009.

[JP03] Marc Joye and Pascal Paillier. GCD-Free Algorithms for Computing Modular Inverses. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2779 of *Lecture Notes in Computer Science*, pages 243–253. Springer, 2003.

[JPY01] Marc Joye, Pascal Paillier, and Sung-Ming Yen. Secure evaluation of modular functions, 2001.

[JT11] Marc Joye and Michael Tunstall. *Fault Analysis in Cryptography*. Springer LNCS, March 2011. http://joye.site88.net/FAbook.html. DOI: 10.1007/978-3-642-29656-7 ; ISBN 978-3-642-29655-0.

[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, *ACM Conference on Computer and Communications Security*, pages 286–296. ACM, 2007.

[KD09] Boris Köpf and Markus Dürmuth. A provably secure and efficient countermeasure against timing attacks. In *CSF*, pages 324–335. IEEE Computer Society, 2009.

[KFSV11] Dusko Karaklajic, Junfeng Fan, Jörn-Marc Schmidt, and Ingrid Verbauwhede. Low-cost fault detection method for ECC using montgomery powering ladder. In *Design, Automation and Test in Europe, DATE 2011, Grenoble, France, March 14-18, 2011*, pages 1016–1021. IEEE, 2011.

[Kis14] Ágnes Kiss. Testing Self-Secure Exponentiation Countermeasures Against the Bellcore Attack, 2014. Master thesis, http://math.bme.hu/~kissa/diplom_Agnes_Kiss.pdf.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Proceedings of CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer-Verlag, 1999.

[KKHH11] Sung-Kyoung Kim, Tae Hyun Kim, Dong-Guk Han, and Seokhie Hong. An efficient CRT-RSA algorithm secure against power and fault attacks. *J. Syst. Softw.*, 84:1660–1669, October 2011.

[Koç94] Çetin Kaya Koç. High-Speed RSA Implementation, November 1994. Version 2, ftp://ftp.rsasecurity.com/pub/pdfs/tr201.pdf.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proceedings of CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer-Verlag, 1996.

[Leo06] V.K. Leont'ev. Roots of random polynomials over a finite field. *Mathematical Notes*, 80(1-2):300–304, 2006.

[LKW06] Sining Liu, Brian King, and Wei Wang. A CRT-RSA algorithm secure against hardware fault attacks. In *Second International Symposium on Dependable Autonomic and Secure Computing (DASC 2006), 29 September - 1 October 2006, Indianapolis, Indiana, USA*, pages 51–60. IEEE Computer Society, 2006.

[LLL82] A.K. Lenstra, H.W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[LPEM+14] Ronan Lashermes, Marie Paindavoine, Nadia El Mrabet, Jacques J.A. Fournier, and Louis Goubin. Practical Validation of Several Fault Attacks against the Miller Algorithm. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on*, pages 115–122, Sept 2014. Busan, Korea.

[LRT14] Duc-Phong Le, Matthieu Rivain, and Chik How Tan. On double exponentiation for securing RSA against fault analysis. In Josh Benaloh, editor, *CT-RSA*, volume 8366 of *Lecture Notes in Computer Science*, pages 152–168. Springer, 2014.

[MAM+03] Simon Moore, Ross Anderson, Robert Mullins, George Taylor, and Jacques J.A. Fournier. Balanced Self-Checking Asynchronous Logic for Smart Card Applications. *Journal of Microprocessors and Microsystems*, 27(9):421–430, October 2003.

[MMM04] Máire McLoone, Ciaran McIvor, and John V. McCanny. Coarsely integrated operand scanning (CIOS) architecture for high-speed Montgomery modular multiplication. In Oliver Diessel and John Williams, editors, *Proceedings of the 2004 IEEE International Conference on Field-Programmable Technology, Brisbane, Australia, December 6-8, 2004*, pages 185–191. IEEE, 2004.

[MO12] Luke Mather and Elisabeth Oswald. Pinpointing side-channel information leaks in web applications. *J. Cryptographic Engineering*, 2(3):161–177, 2012.

[MOP06] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, December 2006. ISBN 0-387-30857-1, http://www.dpabook.org/.

[MOPT12] Andrew Moss, Elisabeth Oswald, Dan Page, and Michael Tunstall. Compiler Assisted Masking. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES*, volume 7428 of *LNCS*, pages 58–75. Springer, 2012.

[MOS11] Stefan Mangard, Elisabeth Oswald, and François-Xavier Standaert. One for All - All for One: Unifying Standard DPA Attacks. *Information Security, IET*, 5(2):100–111, 2011. ISSN: 1751-8709 ; Digital Object Identifier: 10.1049/iet-ifs.2010.0096.

[MS06] Stefan Mangard and Kai Schramm. Pinpointing the Side-Channel Leakage of Masked AES Hardware Implementations. In *CHES*, volume 4249 of *LNCS*, pages 76–90. Springer, October 10-13 2006. Yokohama, Japan.

[MSGR10] Marcel Medwed, François-Xavier Standaert, Johann Großschädl, and Francesco Regazzoni. Fresh Re-Keying: Security against Side-Channel and Fault Attacks for Low-Cost Devices. In *AFRICACRYPT*, volume 6055 of *LNCS*, pages 279–296. Springer, May 03-06 2010. Stellenbosch, South Africa. DOI: 10.1007/978-3-642-12678-9_17.

[MvOV96] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, October 1996.

[NBD+10] Maxime Nassar, Shivam Bhasin, Jean-Luc Danger, Guillaume Duc, and Sylvain Guilley. BCDL: A high performance balanced DPL with global precharge and without early-evaluation. In *DATE'10*, pages 849–854. IEEE Computer Society, March 8-12 2010. Dresden, Germany.

[NNS10] Michael Naehrig, Ruben Niederhagen, and Peter Schwabe. New software speed records for cryptographic pairings. In Michel Abdalla and Paulo S.L.M. Barreto, editors, *Progress in Cryptology – LATINCRYPT 2010*, volume 6212 of *Lecture Notes in Computer Science*, pages 109–123. Springer-Verlag Berlin Heidelberg, 2010. Updated version: http://cryptojedi.org/papers/#dclxvi.

[Pai99] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, May 2-6 1999. Prague, Czech Republic.

[PM05] Thomas Popp and Stefan Mangard. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *LNCS*, pages 172–186. Springer, 2005.

[RG14a] Pablo Rauzy and Sylvain Guilley. A formal proof of countermeasures against fault injection attacks on CRT-RSA. *Journal of Cryptographic Engineering*, 4(3):173–185, 2014.

[RG14b] Pablo Rauzy and Sylvain Guilley. Formal Analysis of CRT-RSA Vigilant's Countermeasure Against the BellCoRe Attack. In *3rd ACM SIGPLAN Program Protection and Reverse Engineering Workshop (PPREW 2014)*, January 25 2014. San Diego, CA, USA. ISBN: 978-1-4503-2649-0.

[RG14c] Pablo Rauzy and Sylvain Guilley. Countermeasures Against High-Order Fault-Injection Attacks on CRT-RSA. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on*, pages 68–82, Sept 2014. Busan, Korea.

[Riv09] Matthieu Rivain. Securing RSA against Fault Analysis by Double Addition Chain Exponentiation. Cryptology ePrint Archive, Report 2009/165, 2009. http://eprint.iacr.org/2009/165/.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably Secure Higher-Order Masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.

[RS09] Mathieu Renauld and François-Xavier Standaert. Algebraic Side-Channel Attacks. In Feng Bao, Moti Yung, Dongdai Lin, and Jiwu Jing, editors, *Inscrypt*, volume 6151 of *Lecture Notes in Computer Science*, pages 393–410. Springer, 2009.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[RSVC09] Mathieu Renauld, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Algebraic Side-Channel Attacks on the AES: Why Time also Matters in DPA. In *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 97–111. Springer, September 6-9 2009. Lausanne, Switzerland.

[SBG+09] Nidhal Selmane, Shivam Bhasin, Sylvain Guilley, Tarik Graba, and Jean-Luc Danger. WDDL is Protected Against Setup Time Violation Attacks. In *FDTC*, pages 73–83. IEEE Computer Society, September 6th 2009. In conjunction with CHES'09, Lausanne, Switzerland. DOI: 10.1109/FDTC.2009.40; Online version: http://hal.archives-ouvertes.fr/hal-00410135/en/.

[SDMB14] Victor Servant, Nicolas Debande, Houssem Maghrebi, and Julien Bringer. Study of a Novel Software Constant Weight Implementation. In *CARDIS*, Lecture Notes in Computer Science. Springer, November 2014. Paris, France.

[SED+11] Youssef Souissi, Moulay Aziz Elaabid, Jean-Luc Danger, Sylvain Guilley, and Nicolas Debande. Novel Applications of Wavelet Transforms based Side-Channel Analysis, September 26-27 2011. Non-Invasive Attack Testing Workshop (NIAT 2011), co-organized by NIST & AIST. Todai-ji Cultural Center, Nara, Japan. (PDF).

[SEE98] Maitham Shams, Jo. C. Ebergen, and Mohamed I. Elmasry. Modeling and comparing CMOS implementations of the C-Element. *IEEE Transactions on VLSI Systems*, 6(4):563–567, December 1998.

[Sha99] Adi Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks, November 1999. US Patent Number 5,991,415; also presented at the rump session of EUROCRYPT '97 (May 11–15, 1997, Konstanz, Germany).

[SP06] Kai Schramm and Christof Paar. Higher Order Masking of the AES. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *LNCS*, pages 208–225. Springer, 2006.

[TNK+14] V. Tomashevich, Y. Neumeier, R. Kumar, O. Keren, and I. Polian. Protecting cryptographic hardware against malicious attacks by nonlinear robust codes. In *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2014 IEEE International Symposium on*, pages 40–45, Oct 2014.

[TPR13] Adrian Thillard, Emmanuel Prouff, and Thomas Roche. Success through Confidence: Evaluating the Effectiveness of a Side-Channel Attack. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES*, volume 8086 of *Lecture Notes in Computer Science*, pages 21–36. Springer, 2013.

[TV04a] Kris Tiri and Ingrid Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *DATE'04*, pages 246–251. IEEE Computer Society, February 2004. Paris, France. DOI: 10.1109/DATE.2004.1268856.

[TV04b] Kris Tiri and Ingrid Verbauwhede. Place and Route for Secure Standard Cell Design. In Kluwer, editor, *Proceedings of WCC / CARDIS*, pages 143–158, Aug 2004. Toulouse, France.

[TV06] Kris Tiri and Ingrid Verbauwhede. A digital design flow for secure integrated circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(7):1197–1208, 2006.

[TW12] Mohammad Tehranipoor and Cliff Wang, editors. *Introduction to Hardware Security and Trust*. Springer, 2012. ISBN 978-1-4419-8079-3.

[U.S99] U.S. Department of Commerce, National Institute of Standards and Technology. Recommended Elliptic Curves For Federal Government Use, July 1999. http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf.

[U.S13] U.S. Department of Commerce, National Institute of Standards and Technology. FIPS PUB 186-4, FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION: Digital Signature Standard (DSS), July 2013. https://oag.ca.gov/sites/all/files/agweb/pdfs/erds1/fips_pub_07_2013.pdf.

[Vig08a] David Vigilant. RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2008.

[Vig08b] David Vigilant. RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks. In *CHES*, 2008. Slides presented at CHES [Vig08a] (personal communication).

[Vig10] David Vigilant. Countermeasure securing exponentiation based cryptography, Feb 17 2010. European patent, EP 2154604 A1.

[vWWM11] Jasper G. J. van Woudenberg, Marc F. Witteman, and Federico Menarini. Practical Optical Fault Injection on Secure Microcontrollers. In Luca Breveglieri, Sylvain Guilley, Israel Koren, David Naccache, and Junko Takahashi, editors, *FDTC*, pages 91–99. IEEE, 2011.

[Wag04] David Wagner. Cryptanalysis of a provably secure CRT-RSA algorithm. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick Drew McDaniel, editors, *ACM Conference on Computer and Communications Security*, pages 92–97. ACM, 2004.

[WK11] Zhen Wang and M. Karpovsky. Algebraic manipulation detection codes and their applications for design of secure cryptographic devices. In *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*, pages 234–239, July 2011.

[YJ00] Sung-Ming Yen and Marc Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Trans. Computers*, 49(9):967–970, 2000. DOI: 10.1109/12.869328.

[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, *ACM Conference on Computer and Communications Security*, pages 305–316. ACM, 2012.

0xB

The sensi Toolbox

The sensi (*SEN Security Inspector*) toolbox is composed of the tools I develop as part of my research on applying formal methods to cryptosystems implementation security. It is composed of three tools: paioli, finja, and enredo. All three tools are easy to build and install thanks to OPAM[1] and `ocamlbuild`[2].

B.1 paioli

The goal of paioli[3] (*Power Analysis Immunity by Offsetting Leakage Intensity*) is to protect assembly code against power analysis attacks such as DPA (differential power analysis) and CPA (correlation power analysis), and to formally prove the efficiency of the protection. To this end, it implements the automatic insertion of a balancing countermeasure, namely DPL (dual-rail with precharge logic), in assembly code (for now limited to bitsliced block-cipher type of algorithms). Independently, it is able to statically verify if the power consumption of a given assembly code is correctly balanced with regard to a leakage model (e.g., the Hamming weight of values, or the Hamming distance of values updates).

```
paioli [options] <input-file>
    -bf Bit to use as F is DPL macros (default: 1)
    -bt Bit to use as T is DPL macros (default: 0)
    -po Less significant bit of the DPL pattern for DPL LUT access
        (default: 0)
    -cl Compact the DPL look-up table (LUT) if present
    -la Address in memory where to put the DPL LUT (default: 0)
    -r1 Register number of one of the three used by DPL macros
        (default: 20)
    -r2 Register number of one of the three used by DPL macros
        (default: 21)
    -r3 Register number of one of the three used by DPL macros
        (default: 22)
    -a  Adapter for custom assembly language
    -o  asm output (default: no output)
    -l  Only check syntax if present
    -d  Perform DPL transformation of the code if present
    -v  Perform leakage verification if present
    -s  Perform simulation if present
    -r  Register count for simulation (default: 32)
```

[1] https://opam.ocaml.org/
[2] https://ocaml.org/learn/tutorials/ocamlbuild/
[3] http://pablo.rauzy.name/sensi/paioli.html

```
-m  Memory size for simulation (default: 1024)
-M  range of memory to display after simulation
-R  range of registers to display after simulation
```

The rest of this section details its features.

Adapters. To easily adapt it to any assembly language, it has a system of plugins (which we call "adapters") that allows to easily write a parser and a pretty-printer for any language and to use them instead of the internal parser and pretty-printer (which are made for the internal language we use, see Sec. 4.3.1) without having to recompile the whole tool.

DPL transformation. If asked so, paioli is able to automatically apply the DPL transformation as explained in Sec. 4.3.2. It takes as arguments which bits to use for the DPL protocol, the offset at which to place the pattern for look-up tables (for example, we used an offset of 1 to avoid resorting to the least significant bit which leaks differently), and where in memory should the look-up tables start. Given these parameters, the tool verifies that they are valid and consistent according to the DPL protocol, and then it generates the DPL balanced code corresponding to the input code, including the code for look-up tables initialization. Optionally, the tool is able to compact the look-up tables (since they are sparse), still making sure that their addresses respect the DPL protocol (Sec. 4.2.2).

Simulation. If asked so, paioli can simulate the execution of the code after its optional DPL transformation. The simulator is equipped to do the balance verification proof (see Sec. 4.4) but it is not mandatory to do the balance analysis when running it. It takes as parameters the size of the memory and the number of register to use, and initializes them to the set of two DPL encoded values of 1 and 0 corresponding to the given DPL parameters. The tool can optionally display the content of selected portions of the memory or of chosen registers after execution, which is useful for inspection and debugging purpose for example.

Balance verification. The formal verification of the balance of the code is an essential functionality of the tool. Indeed, bugs occur even when having a thorough and comprehensive specification, thus we believe that it is not sufficient to have a precise and formally proven method for generating protected code, but that the results should be independently verified (see Sec. 4.4).

B.2 finja

The goal of finja[4] (*Fault INJection Analysis*) is to formally analyze fault injection attacks and countermeasures against them. It knows about several fault models (random or zero, permanent or transient) and performs term rewriting using arithmetic rules to symbolically reduce faulted expressions and verify if an *attack success condition* holds. Doing that for all possible fault injections permits to find (sometimes new) attacks, or to show the security of the tested countermeasure otherwise.

finja has also been used by other researchers (namely Ágnes Kiss and Juliane Krämer in [Kis14]).

```
finja [options] <input-file>
    -o <output-file> HTML report (defaults to input-file.html)
    -l Only check syntax
    -a Only simplify the input term (no attacks)
    -s Print only successful attacks in the html report
    -t Enable transient faults (default is only permanent fault)
    -r Inject randomizing fault (default)
    -z Inject zeroing fault
    -n Specify the number of faults (default is 1).
       If specified, you can use the -r or -z option for each
       fault (last one is repeated).
```

Input file formats. Input files are text files (which use the .fia extension by convention) which first contain a description of the computation term to analyze, then a line with a single `%%` symbol, then an attack success condition.

The BNF of the syntax is given below.

```
fia      ::= term '%%' cond
term     ::= ( stmt )* 'return' mp_expr ';'
stmt     ::= ( decl | assign | verif ) ';'
decl     ::= 'noprop' mp_var ( ',' mp_var )*
           | 'prime' mp_var ( ',' mp_var )*
assign   ::= var ':=' mp_expr
verif    ::= 'if' mp_cond 'abort with' mp_expr
mp_expr  ::= '{' expr '}' | expr
expr     ::= '(' mp_expr ')'
           | '0' | '1' | var
           | '-' mp_expr
           | mp_expr '+' mp_expr
           | mp_expr '-' mp_expr
           | mp_expr '*' mp_expr
           | mp_expr '^' mp_expr
           | mp_expr 'mod' mp_expr
mp_cond  ::= '{' cond '}' | cond
cond     ::= '(' mp_cond ')'
           | mp_expr '=' mp_expr
           | mp_expr '!=' mp_expr
           | mp_expr '=[' mp_expr ']' mp_expr
           | mp_expr '!=[' mp_expr ']' mp_expr
           | mp_cond '/\' mp_cond
           | mp_cond '\/' mp_cond
mp_var   ::= '{' var '}' | var
var      ::= [a-zA-Z][a-zA-Z0-9_']*
```

[4] http://pablo.rauzy.name/sensi/finja.html

The `mp` in `mp_var`, `mp_expr` and `mp_cond` stands for "maybe protected". A variable name at declaration time, an expression, or a condition surrounded by `{` and `}` can't be faulted.

The attack success condition can use all the variables introduced in the computation term, plus two special variables `_` and `@` which respectively represent the returned expression of the computation term as given in the input file, and the returned expression of the computation term with injected faults.

Output. Output reports of finja are printed in HTML format for easy reviewing.

finja report for "tests/crt-rsa_shamir.fia"

Options:
transient faults: disabled.
fault types: randomizing.
Maximum number of faults: 1.

Summary BROKEN
Total number of different fault injections: 31.
Total number of successful attack: 7 (show all).

Computation

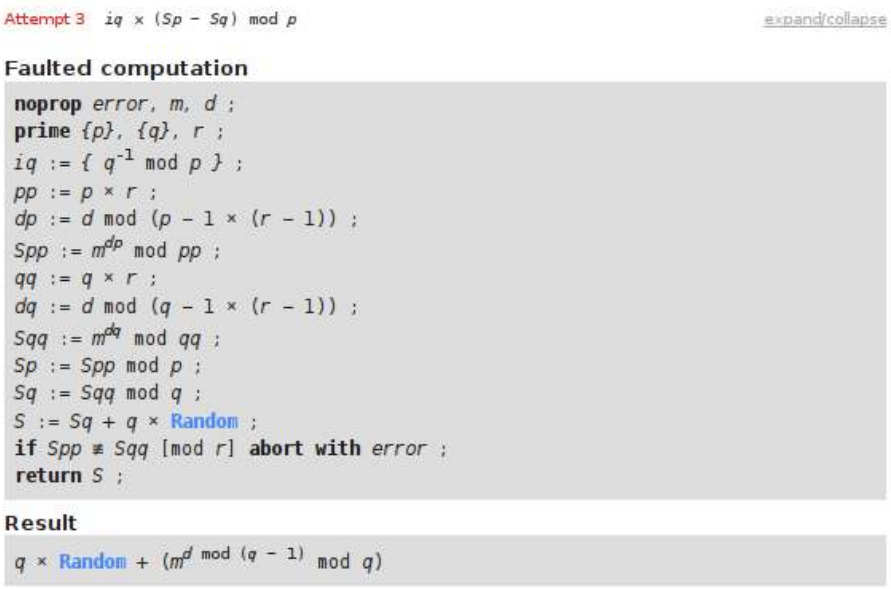
```
noprop error, m, d ;
prime {p}, {q}, r ;
iq := { q^-1 mod p } ;
pp := p × r ;
dp := d mod (p − 1 × (r − 1)) ;
Spp := m^dp mod pp ;
qq := q × r ;
dq := d mod (q − 1 × (r − 1)) ;
Sqq := m^dq mod qq ;
Sp := Spp mod p ;
Sq := Sqq mod q ;
S := Sq + q × (iq × (Sp − Sq) mod p) ;
if Spp ≠ Sqq [mod r] abort with error ;
return S ;
```

Attack success condition:
$S \neq @ \wedge (S \equiv @ \,[\mathrm{mod}\, p] \vee S \equiv @ \,[\mathrm{mod}\, q])$

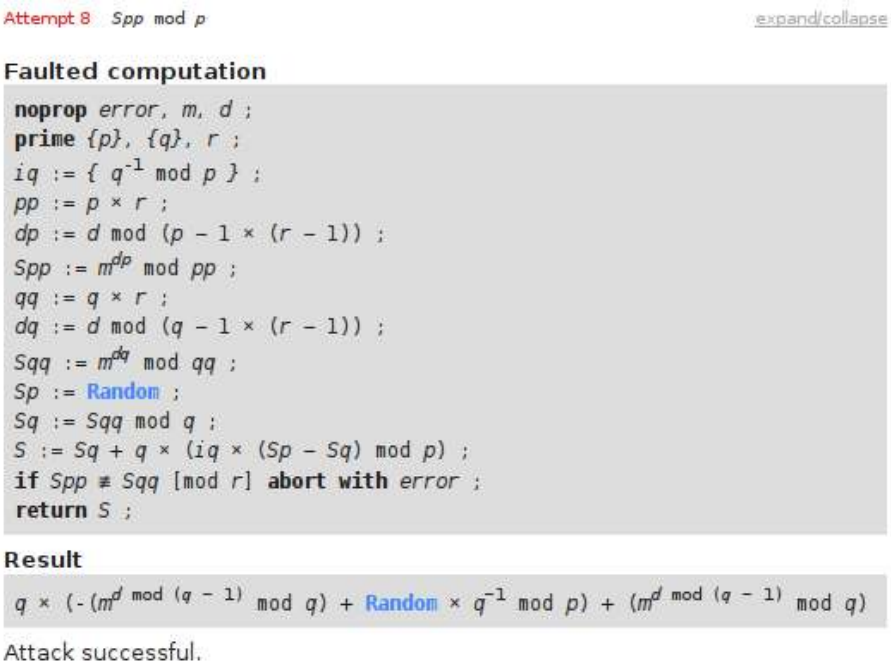
Reduced computation

$$q \times (-(m^{d \bmod (q-1)} \bmod q) + m^{d \bmod (p-1)} \times q^{-1} \bmod p) + (m^{d \bmod (q-1)} \bmod q)$$

Figure B.1: Header of finja report for Shamir's countermeasure [Sha99].



(a) Example of attack found by finja on Shamir's countermeasure [Sha99].



(b) Another example of attack found by finja on Shamir's countermeasure [Sha99].

finja report for "crt-rsa_vigilant-fixed_simplified.fia"

Options:
transient faults: disabled.
fault types: randomizing.
Maximum number of faults: 1.

Summary PROTECTED
Total number of different fault injections: 62.
Total number of successful attack: 0 (hide others).

Computation

```
noprop ERR1, ERR2, ERR3, M, e, r ;
prime {p}, {q} ;
dp := { e^-1 mod (p - 1) } ;
dq := { e^-1 mod (q - 1) } ;
iq := { q^-1 mod p } ;
N := p × q ;
p' := p × r × r ;
Mp := M mod p' ;
ipr := p^-1 mod (r × r) ;
Bp := p × ipr ;
Ap := 1 - Bp mod p' ;
M'p := Ap × Mp + Bp × (1 + r) mod p' ;
if M'p + N ≠ M [mod p] abort with ERR1 ;
Spr := M'p^dp mod p' ;
miniSp := 1 + dp × r ;
q' := q × r × r ;
Mq := M mod q' ;
iqr := q^-1 mod (r × r) ;
Bq := q × iqr ;
Aq := 1 - Bq mod q' ;
M'q := Aq × Mq + Bq × (1 + r) mod q' ;
if M'q + N ≠ M [mod q] abort with ERR2 ;
Sqr := M'q^dq mod q' ;
miniSq := 1 + dq × r ;
S := Sqr + q × (iq × (Spr - Sqr) mod p') ;
miniS := miniSq + q × iq × (miniSp - miniSq) ;
if S ≠ miniS [mod r × r] abort with ERR3 ;
return S mod N ;
```

Attack success condition:
$\_ \neq @ \wedge (\_ \equiv @ \ [\mathrm{mod}\ p] \vee \_ \equiv @ \ [\mathrm{mod}\ q])$

Reduced computation

$$p \times M^{e^{-1} \bmod (q-1)} \times (p^{-1} \bmod q) + q \times M^{e^{-1} \bmod (p-1)} \times (q^{-1} \bmod p) \bmod (p \times q)$$

Figure B.3: Header of finja report for our fixed and simplified version of Vigilant's countermeasure [RG14c].

B.3 enredo

The goal of enredo[5] (*ENtanglement REdundancy Defense Op*) is to protect asymmetric cryptography algorithms against fault injection attacks. It is a formally proved compiler (the semantic of its input and output is proved to be the same) which takes an algorithm as input and outputs a protected equivalent algorithm. The protection technique, that we call *entanglement*, is based on efficient redundancy for integrity verification, and has also been formally proven.

```
enredo [options] <input-file>
    -n Number of small structure to have redundancy with.
    -p Output Python code.
```

More information about enredo can be found in chapter 8.

[5] http://pablo.rauzy.name/sensi/enredo.html

0xC

Publications

Before my PhD

- **Can a Program Reverse-Engineer Itself?**
  Antoine Amarilli, David Naccache, Pablo Rauzy, Emil Simion.
  — IMACC 2011: *13th IMA International Conference on Cryptography and Coding.*

- **Can Code Polymorphism Limit Information Leakage?**
  Antoine Amarilli, Sascha Müller, David Naccache, Daniel Page, Pablo Rauzy, Michael Tunstall.
  — WISTP 2011: *Workshop in Information Security Theory and Practice.*

- **From Rational Number Reconstruction to Set Reconciliation and File Synchronization**
  Antoine Amarilli, Fabrice Ben Hamouda, Florian Bourse, Robin Morisset, David Naccache, Pablo Rauzy.
  — TGC 2012: *7th International Symposium on Trustworthy Global Computing.*

During my PhD

- **A Formal Proof of Countermeasures Against Fault Injection Attacks on CRT-RSA**
  Pablo Rauzy, Sylvain Guilley.
  — PROOFS 2013: *2nd Workshop on Security Proofs for Embedded Systems.*
  — *Journal of Cryptographic Engineering* (2014).

- **Formal Analysis of CRT-RSA Vigilant's Countermeasure Against the BellCoRe Attack**
  Pablo Rauzy, Sylvain Guilley.
  — PPREW 2014: *3rd SIGPLAN Program Protection and Reverse Engineering Workshop.*

- **Formally Proved Security of Assembly Code Against Power Analysis**
  Pablo Rauzy, Sylvain Guilley, Zakaria Najm.
  — PROOFS 2014: *3rd Workshop on Security Proofs for Embedded Systems.*
  — *Journal of Cryptographic Engineering*, (2015).

- **Countermeasures Against High-Order Fault-Injection Attacks on CRT-RSA**
  Pablo Rauzy, Sylvain Guilley.
  — FDTC 2014: *11th IACR Workshop on Fault Diagnosis and Tolerance in Cryptography.*

- **High Precision Fault Injections on the Instruction Cache of ARMv7-M Architectures**
  Lionel Rivière, Zakaria Najm, Pablo Rauzy, Jean-Luc Danger, Julien Bringer, Laurent Sauvage.
  — HOST 2015: *IEEE International Symposium on Hardware-Oriented Security and Trust.*

To Appear

- **Algorithmic Countermeasures Against Fault Attacks and Power Analysis for RSA-CRT**
  Ágnes Kiss, Juliane Krämer, Pablo Rauzy, Jean-Pierre Seifert.
  Currently under review.

- **Generic Countermeasures Against Fault Injection Attacks on Asymmetric Cryptography**
  Pablo Rauzy, Sylvain Guilley, Martin Moreau, Zakaria Najm.
  Currently under review.

- **Protecting Pairings Against Fault Injection Attacks**
  Nadia El Mrabet, Jacques Fournier, Louis Goubin, Sylvain Guilley, Ronan Lashermes, Martin Moreau, Pablo Rauzy.
  — Book chapter in *Handbook of Pairing Based Cryptography*, Nadia El Mrabet and Marc Joye, editors, CRC Press Taylor and Francis group.

0xD

Acronyms

AES Advanced Encryption Standard

ASCA Algebraic Side-Channel Attack

ASIC Application-Specific Integrated Circuit

BCDL Balanced Cell-based Differential Logic

BNF Backus–Naur Form

CIOS Coarsely Integrated Operand Scanning

CPA Correlation Power Analysis

CPU Central Processing Unit

CRT Chinese Remainder Theorem

DES Data Encryption Standard

DFA Differential Fault Analysis

DH Diffie-Hellman

DPA Differential Power Analysis

DPL Dual-rail with Precharge Logic

DSA Digital Signature Algorithm

ECC Elliptic-Curve Cryptography

ECDSA Elliptic-Curve Digital Signature Algorithm

ECSM Elliptic-Curve Scalar Multiplication

EG ElGammal

EM Electromagnetic

EMFI Electromagnetic Fault Injection

FPGA Field-Programmable Gate Array

JTAG Joint Test Action Group

LLL Lenstra–Lenstra–Lovász

MAC Multiplier-Accumulator Circuit

MDPL Masked Dual-rail with Precharge Logic

NICV Normalized Inter-Class Variance

RSA Rivest–Shamir–Adleman

SE Safe-Error

SNR Signal-to-Noise Ratio

SPA Simple Power Analysis

WDDL Wave Dynamic Differential Logic

0xE

Figures, Tables, and Algorithms

E.1 List of Figures

E.2 List of Tables

E.3 List of Algorithms

0xF

Le libre accès à la recherche : une introduction

F.1 Le fonctionnement de la recherche

Pour comprendre le problème qui se pose actuellement avec la publication des résultats de la recherche, en particulier de la recherche scientifique, il faut d'abord s'intéresser à son fonctionnement.

Schématiquement, la recherche se déroule en une succession d'étapes identifiables, que l'on va essayer de décrire ici :

1. La recherche d'une question.

   Les chercheurs ont besoin d'avoir accès à tous les résultats de la recherche jusqu'à présent, afin de connaître l'*état de l'art* de leur champ de recherche. Cela leur permet de trouver, parmi toutes les choses qu'il y reste à faire, laquelle serait une bonne prochaine étape pour faire progresser le savoir et la technique de leur domaine. C'est là que se passe une phase très importante : la formulation d'une question.

2. La recherche d'une réponse.

   Une fois que cette question est formulée, les chercheurs, souvent en groupe, cherchent à y répondre. Pour cela, ils vont devoir recenser toutes les connaissances de leur domaine et des domaines voisins, afin de repérer les outils qui pourraient leur être utiles pour résoudre le problème qu'ils se sont posé.
   Une fois que cela est fait, la façon de travailler dépend beaucoup du domaine de recherche, et il serait difficile de généraliser cela ici sans dire de bêtises. Cependant, sans parler de méthode, on peut affirmer que ce travail peut aboutir de plusieurs manières : la réponse à la question peut être trouvée, des réponses à d'autres questions peuvent émerger, de nouvelles méthodes ou de nouveaux outils de travail peuvent être découverts, etc. La liste des possibilités est longue, mais dans tous les cas, des résultats sont obtenus, et de nouvelles questions se posent.

3. L'obtention de résultats et la rédaction de l'article.

   Les résultats obtenus peuvent être ceux attendus, ou ils peuvent être différents. Dans les deux cas, quand ces résultats font significativement avancer l'état de l'art, par exemple quand les chercheurs ont réussi à prouver une conjecture, les chercheurs décident de les décrire dans un *article* (on dit aussi *papier*) afin

de les partager avec leur communauté, afin que les autres puissent à leur tour faire progresser la connaissance en se basant sur ces nouveaux résultats.

4. La *soumission* de l'article.

   Une fois rédigé, l'article détaillant les méthodes utilisées et les résultats obtenus est soumis pour publication dans une *revue* (on dit aussi un *journal*) ou une *conférence* (on parle aussi de *symposium*, de *colloque*, ...) correspondant au domaine de recherche concerné (pour simplifier les choses, on ne va plus parler que de revues dans la toute la suite de ce document, mais ce qui est dit s'applique tout autant aux conférences et à la publication de leur *actes*, ou *proceedings* en anglais). Cela signifie que l'article est envoyé à la revue, pour que celle-ci le distribue le plus largement possible si l'article est accepté.
   → L'article à ce stade est appelé un *preprint*.

5. Le processus d'*évaluation par les pairs* de l'article.

   Afin de décider de la diffusion ou non d'un article, sa qualité et sa nouveauté doivent être vérifiées par le *comité éditorial* (*editorial board* en anglais, ou *program committee* pour les conférences) de la revue dans laquelle il a été soumis. Pour cela, l'article est envoyé à des *relecteurs* (*reviewers* ou *referees* en anglais), qui forment un groupe qu'on appelle le *comité de lecture* de la revue. On appelle cela l'évaluation, l'examen, la validation, la relecture, ou encore l'arbitrage, par les pairs (*peer-review* en anglais). C'est une étape très importante. Le principe est que les relecteurs, qui sont d'autres chercheurs des domaines concernés par l'article, vont le relire attentivement, en vérifiant que les méthodes employées sont raisonnables et rigoureuses, et que les résultats obtenus le sont tout autant. Ils s'assurent aussi de la pertinence de la diffusion de l'article (est-ce qu'il contribue vraiment de manière significative à faire progresser l'état de l'art ?) et de la qualité de sa rédaction (inutile de diffuser un texte incompréhensible).
   Une fois que les relecteurs ont fait leur rapport sur l'article, les auteurs intègrent les éventuelles remarques dans leur article et resoumettent l'article pour publication (parfois dans la même revue, parfois ailleurs plus tard, si vraiment il y a beaucoup de choses à changer ou à refaire). Cela recommence jusqu'à ce que l'article soit accepté ou que ses auteurs renoncent à le publier.
   → L'article à ce stade est appelé un *postprint*.

6. La *publication* de l'article.

   Une fois que l'article a été accepté pour publication, la *maison d'édition* de la revue le met en forme selon les conventions typographiques et la charte de présentation de la revue. À ce moment là, l'article est "*publié*".
   → L'article à ce stade est dit en *version publiée* (*published version* ou *publisher version* en anglais).

Le mot « publié » est entre guillemets parce que littéralement, cela signifie « rendu public », et comme on va le voir, ce n'est pas aussi simple dans le système de publication tel qu'il existe actuellement.

J'utilise dans ce dossier le terme « *maisons d'édition* », faute de mieux en français, pour désigner ce qu'on appelle en anglais les « *publishers* » (certains françisent en « publicheurs »). Il est important de faire la différence entre les maisons d'édition

classiques et les maisons d'édition scientifiques, mais aussi et surtout de ne pas faire un amalgame en utilisant le mot « éditeurs ». En effet, ce mot désigne aussi bien les « *publishers* », c'est à dire ceux que j'appelle « maisons d'édition », que les « *editors* », c'est à dire les comités éditoriaux des revues, qui sont un collège de chercheurs qui s'occupent de tout le travail d'édition scientifique, tel que l'évaluation par les pairs.

F.2 Le système de publication

Pour comprendre le système de publication à l'heure actuelle, il est intéressant d'avoir en tête un bref historique de ce système. Comme pour le fonctionnement de la recherche ci-dessus, cet historique est schématique, voire grossier par certains aspects.

F.2.1 Historique rapide du système de publication

Suite à une longue période d'échanges informels de lettres entre ce qu'on appellerait aujourd'hui des chercheurs, la première revue littéraire et scientifique (d'Europe) apparaît début 1665 à Paris, il s'agit du *Journal des sçavans*[1]. Son but déclaré est de faire connaître « ce qui se passe de nouveau dans la République des lettres ». Elle suscite beaucoup d'intérêt et rapidement d'autres revues voient le jour, notamment *Philosophical Transactions of the Royal Society*[2], trois mois plus tard à Londres. Il est amusant de noter que ces deux revues continuent d'exister de nos jours.

De multiples revues apparaissent au XVII^{e} siècle dans le but de résoudre les problèmes de la rapidité de diffusion des connaissances, de l'impartialité, de la priorité, et de la visibilité des travaux de recherche.

Au XVIII^{e} siècle, les revues s'imposent comme le moyen de communication prioritaire de diffusion des nouvelles connaissances. À cette époque, les revues sont publiées par des sociétés savantes ou des instituts publics (par exemple ceux créés à la fin du siècle suite à la Révolution française dans le but de reconstituer et de conserver le patrimoine scientifique).

Au XIX^{e} siècle commencent à apparaître de nombreuses maisons d'édition spécialisées dans la communication scientifique et littéraire. Parmi elles, certaines que l'on connaît encore aujourd'hui comme Masson (1804), Wiley (1807), Springer-Verlag (1842), Dunod (1876), ou encore Elsevier (1880).

Au XX^{e} siècle, suite à la Première Guerre mondiale, et aussi à la Grande Dépression (crise économique de 1929), beaucoup de ces petites maisons d'édition disparaissent ou se font racheter par d'autres. Les plus importantes commencent à cette époque à faire des universités leur cœur de marché.

Après la Seconde Guerre mondiale, on assiste, parallèlement à la reprise économique (Trente Glorieuses), à une forte croissance de l'activité scientifique. Automatiquement, l'industrie d'édition scientifique connaît une accélération, passant de petites maisons d'édition à de grosses entreprises multinationales, cherchant naturellement à maximiser leurs profits avant tout.

[1] https://fr.wikipedia.org/wiki/Journal_des_savants

[2] https://fr.wikipedia.org/wiki/Philosophical_Transactions_of_the_Royal_Society

À partir de 1960, une multitude de nouvelles revues sont créées et les pratiques tarifaires de cette industrie évoluent. On commence à voir les prix des abonnements augmenter. Parallèlement à ça, la délocalisation vers des pays à faible coût de main d'œuvre et l'utilisation des nouvelles technologies par les maisons d'édition font baisser les prix de revient de l'édition et de la distribution pour ces dernières. On constate ici les effets mécaniques de la **privatisation de la diffusion des connaissances** de l'humanité.

À la même période, les bibliothèques universitaires ressentent pour la première fois des problèmes financiers par rapport aux prix des abonnements aux revues. En conséquence, les bibliothèques mettent en place, au nom du prêt entre bibliothèques, la diffusion d'articles de recherche photocopiés à bas coûts.

F.2.2 Qu'en est-il aujourd'hui ?

On rappelle ici que, même si ces revues remplissent aujourd'hui des rôles supplémentaires que nous détaillerons par la suite, leur fonction première lors de leur création était la diffusion des nouvelles connaissances.

Depuis quelques années, grâce à Internet, la diffusion des articles peut être rendue bien plus simple, plus rapide, moins coûteuse, et certainement plus écologique qu'avec les versions systématiquement imprimées de l'intégralité des revues.

De plus, ces avantages d'Internet permettent l'auto-diffusion. Par les chercheurs eux-mêmes d'une part, mais aussi et surtout par les revues, qui n'ont plus besoin des services d'une maison d'édition pour être diffusées mondialement.

F.2.3 Le rôle d'une maison d'édition

Pour des raisons de simplicité, on parle dans toute la suite de ce document de maison d'édition, mais le rôle et les pratiques que l'on décrit s'appliquent tout autant à certaines sociétés savantes qui les imitent (à l'exception de la recherche du profit pour le profit, puisqu'elles n'ont pas d'actionnaires, ce qui est déjà un point positif). Le rôle scientifique d'une maison d'édition se décompose en plusieurs points :

1. La mise en page des articles.

   La mise en forme des articles est une des grosses parties du travail d'édition. Aujourd'hui, elle est de moins en moins nécessaire, voire quasi-inexistante dans certains domaines. En effet, des modèles et feuilles de styles sont fournis aux auteurs pour les traitements de texte (LaTeX, LibreOffice, Word) qu'ils utilisent, afin qu'ils puissent directement écrire leurs articles pour la charte de présentation de la revue. Cela dit, il reste encore des domaines dans lequel ce travail est nécessaire.

2. Distribuer et faire connaître le plus largement possible les articles qu'elle publie dans ses revues.

   Historiquement, cela voulait dire imprimer les articles et les envoyer dans le monde entier. Aujourd'hui, grâce à Internet, il s'agit principalement d'héberger en ligne des fichiers PDF. En effet, d'après le rapport d'activité[3] pour l'année

[3] http://www.reedelsevier.com/mediacentre/pressreleases/2014/Pages/publication-of-annual-reports-and-financial-statements-2013.aspx

2013 de la plus grosse maison d'édition (Reed-Elsevier), plus de 80% de leur revenu dus aux abonnements proviennent des formats électroniques, et ce chiffre est en augmentation constante depuis des années.

3. Faire (re)connaître les chercheurs.

   Lorsqu'une revue acquiert un certain prestige grâce à la qualité des articles qui y sont publiés, les chercheurs qui parviennent à y publier leurs articles par la suite profitent à leur tour de ce prestige. Il est aujourd'hui impossible de faire carrière dans la recherche sans avoir publié plusieurs articles dans des revues prestigieuses. Ce n'est pas un rôle intrinsèque des revues, mais plutôt un effet secondaire automatique du mécanisme de publication dans des revues.

4. Définir les « tendances » thématiques des sujets de recherche.

   Là aussi, il s'agit d'un rôle dont les revues ont mécaniquement hérité, mais qui ne leur est pas intrinsèque. L'avancement des carrières des chercheurs, tout comme le financement de la recherche qui s'organise de plus en plus autour de projets à court-terme (en temps scientifique, même quand il s'agit de 3 voire 5 ans, on parle de très court terme), sont indirectement (mais fortement) influencés par les revues. En effet, les postes pour les chercheurs et les bourses de financement de leurs projets sont attribuées en fonction de critères qui à l'heure actuelle font la plus belle part à la bibliométrie[4], en prenant en compte les « facteur d'impact[5] » et autres « indice h[6] ». Les revues sont donc un des facteurs les plus importants de la bibliométrie et à ce titre participent de manière prépondérante à la définition des tendances thématiques des sujets de recherche.
   En plus d'être navré par la **précarisation de la recherche** induite par ce mode de financement, il est naturel de s'interroger sur **la légitimité du rôle (quoique indirect) des maisons d'édition privées dans le processus de décision de dépense de l'argent public**.

   Remarque F.1. Je vais me permettre ici un petit aparté sur le côté intrinsèquement néfaste de la bibliométrie. C'est très important d'en discuter car **le mouvement pour le libre accès et celui pour se sortir de la bibliométrie sont étroitement liés** comme on va le voir tout au long de ce document. La bibliométrie est forcément subjective en ce sens qu'elle tente de comparer sur une unique dimension des chercheurs, des activités, des revues, etc. qui existent dans un nombre non défini et non semblable de dimensions. De plus, comme tout système de notation, cela donne lieu à des phénomènes émergents inévitables et néfastes. Quels que soient les critères choisis, une bureaucratie se met inéluctablement en place pour les satisfaire. Par nécessité, on ne cherche plus alors à faire de la bonne recherche, mais de la recherche qui satisfasse au mieux ces critères. On se retrouve alors avec des critères qui ne peuvent plus remplir de manière satisfaisante leur rôle initial. Il ne s'agit donc pas seulement de dire que les critères bibliométriques actuels sont mauvais (tout le monde est d'accord sur ce point), mais plutôt qu'il n'est donc pas possible d'avoir de bons critères bibliométriques. En revanche, je tiens

[4] https://fr.wikipedia.org/wiki/Bibliom%C3%A9trie
[5] https://fr.wikipedia.org/wiki/Facteur_d'impact
[6] https://fr.wikipedia.org/wiki/Indice_h

à souligner que je ne suis pas en train de dire qu'il ne faut pas évaluer les chercheurs. Simplement, je pense que pour évaluer un chercheur il faut lire ses quelques articles les plus significatifs, et si ils étaient publiquement disponibles et signés, les rapports d'évaluation par les pairs de ces articles, plutôt que de se contenter de regarder des chiffres qui sortent du chapeau bibliométrique.

5. Gagner de l'argent.

   Quand on est une entreprise multinationale cotée en bourse, le but premier est nécessairement de rapporter des dividendes aux actionnaires. **Faire des bénéfices est donc le but principal des maisons d'édition**, bien plus que de conserver et diffuser les connaissances de l'humanité. Un témoignage édifiant intitulé « De la mondialisation appliquée à l'édition[7] », paru dans *Le Tigre*, raconte les effets du rachat de Masson par Elsevier, vu de l'intérieur.

Si on oublie le cinquième point, on remarque que les deux premiers points concernent réellement le travail d'une maison d'édition, mais qu'il serait tout à fait possible de remplir les deux autres rôles en dehors de ce cadre, par exemple avec des revues indépendantes et auto-publiées.

Une autre chose remarquable, est qu'il semble y avoir une déconnexion entre le rôle des maisons d'édition tel qu'il vient d'être décrit, et les différentes étapes du fonctionnement de la recherche tel que décrit au début de ce document.
Évidemment, ce n'est pas une erreur...

F.2.4 Le partage des coûts jusqu'à la publication

Les chercheurs, les laboratoires, les équipements, etc. sont en écrasante majorité financés par de l'argent public (et, au moins en France, même le peu qui est financé par des entreprises l'est souvent indirectement par de l'argent public grâce à des inventions type crédit d'impôt recherche[8] et autres joyeusetés).

D'autre part, il est important de réaliser que les personnes qui se cachent derrière les comités éditoriaux et les comités de lecture des revues ne sont autres que... des chercheurs ! Évidemment, puisqu'il y a besoin des experts du domaine pour juger de la qualité des articles et pouvoir y déceler des problèmes...

Si on reprend un peu ce qui a été dit jusqu'à présent, on se rend compte que la répartition des dépenses et des recettes est curieuse. Effectivement, des prémices du travail de recherche jusqu'à la rédaction d'un article, et ensuite le contrôle de sa qualité (c'est-à-dire le travail du comité éditorial ainsi que celui du comité de lecture), tout est assuré par des chercheurs et donc financés par de l'argent public.

Pour ne pas s'arrêter en si bon chemin, certaines revues font payer les auteurs à la page (parfois jusqu'à plusieurs centaines d'euros par page) pour la publication d'un article. C'est même la règle dans des domaines de recherche comme la médecine ou la pharmacie.

Et ensuite, pour parfaire le tout, les maisons d'édition s'assurent d'avoir un monopole sur un résultat de recherche en **exigeant des chercheurs qu'ils cèdent le copyright des articles et de leur contenu** (schémas, tableau, graphiques, etc.) à la maison d'édition. Gratuitement.

[7] http://www.le-tigre.net/De-la-mondialisation-appliquee-a-l.html
[8] https://fr.wikipedia.org/wiki/Cr%C3%A9dit_d%27imp%C3%B4t_recherche

Vous ne rêvez pas, le résultat de tout ce travail leur est cédé, en échange de rien du tout.

Remarque F.2. Il est important de noter qu'en droit français, la validité de la cession du copyright telle qu'elle est pratiquée est possiblement nulle. Il semble que cette cession de copyright ne puisse pas avoir valeur de contrat puisqu'elle n'est signée que par un seul côté, qu'il suffit qu'un des auteurs la signe pour engager tous les autres, et qu'il n'y a rien en échange. C'est du moins ce que l'on comprend à la lecture d'un « Avis pour le Comité d'Éthique du CNRS sur les relations entre les chercheurs et les maisons d'édition scientifique[9] » de 2011.

La maison d'édition devait ensuite imprimer et envoyer l'article dans le monde entier, mais cela est de moins en moins vrai, comme on l'a vu dans le second point de la section F.2.3 « Le rôle d'une maison d'édition ». Le coût pour la maison d'édition est alors celui d'héberger des fichiers PDF sur un serveur web, autant dire qu'il est faible. Faible, mais pas proche de zéro a priori, car ces maisons d'édition ont aussi le devoir moral de conserver de manière fiable les archives de la connaissance humaine, qu'elles accaparent depuis plusieurs siècles maintenant.

F.2.5 Le coût des abonnements

Les meilleurs résultats des travaux de recherche financés publiquement et cédés gracieusement aux maisons d'édition sont ensuite revendus par celles-ci, sous forme d'abonnement à leurs revues, aux universités et aux instituts de recherche, qui sont bien évidemment toujours autant financés par de l'argent public.

On utilise donc l'argent public pour racheter un article à une maison d'édition privée, et ce pour chacune des universités et des instituts de recherche qui sont intéressés par le domaine de l'article.

Pour vous donner un ordre d'idées, voilà ce qu'on peut trouver dans le rapport d'activité pour l'année 2011 du Service Commun de Documentation (le réseau des bibliothèques) de l'École normale supérieure[10] :

- Montant total des dépenses : 2 101 587,47 €.
- Dont dépenses documentaires : 1 083 885,16 €.

L'ENS consacre donc plus d'un million d'euros par an aux dépenses documentaires, dont la vaste majorité correspondent à des abonnements à des revues. Le rapport prend soin de préciser la chose suivante, qui révèle que les dépenses documentaires seraient encore bien plus élevées si l'ENS ne pouvait pas bénéficier des abonnements de certains de ses partenaires :

> [N]ous rappelons que les publics scientifiques de l'ENS bénéficient en outre des abonnements électroniques acquis par le CNRS au niveau national et mis à disposition via ses portails nationaux. Ceci constitue un **facteur d'économie important** pour certains départements de l'ENS, qui n'acquièrent pas ces ressources sur leurs budgets propres. Il en est de même pour les ressources consultables par les chercheurs de l'ENS auprès d'autres grands établissements parisiens, notamment l'UPMC.

[9] http://wavelets.ens.fr/PUBLICATIONS/ARTICLES/PDF/312.pdf
[10] http://www.ens.fr/

Malheureusement, l'ENS est loin de faire figure d'exception, l'ordre de grandeur des dépenses en abonnements est plutôt la norme pour les quelques centaines d'établissements d'études supérieures que l'on compte en France. On peut le constater en consultant la base de données ASIBU[11] (application statistique interactive des Bibliothèques universitaires, c'est très laborieux à utiliser, courage), qui malheureusement ne contient pas les informations concernant les bibliothèques de recherche et les abonnements électroniques propres aux instituts de recherche comme le CNRS, Inria, l'INSERM, etc. Pour donner un ordre de grandeur **les dépenses documentaires du CNRS seul s'élèvent aux alentours de 36 millions d'euros par an** d'après leurs rapports financiers et comptes consolidés[12].

Bien évidemment, les articles sont pas non plus disponibles pour le grand public (qui a pourtant tout financé avec ses impôts) y compris et surtout dans des domaines aussi importants que la médecine, ou encore la pharmacie qui a des lobbies puissant face auxquels l'information est le seul contre-pouvoir possible. Quand on n'est pas dans une institution qui a payé les abonnements et qu'on cherche à accéder à un article de recherche, celui-ci sera le plus souvent derrière un *mur à péage* (*paywall* en anglais), vous demandant de payer à la maison d'édition un montant entre 30 et 50 € (en général) pour pouvoir télécharger un PDF de quelques pages.

F.2.6 Les profits des maisons d'édition

On pourrait penser qu'avec la baisse conséquente des coûts de gestion et de distribution due au passage des revues au format électronique, les prix des abonnements seraient revus à la baisse. Évidemment, il n'en est rien. Au contraire, on assiste ces dernières années à des augmentations fulgurantes des tarifs. On le voit bien avec l'exemple de la bibliothèque de Télécom ParisTech[13] pour laquelle j'ai pu obtenir des chiffres sur les cinq dernières années. En effet, entre 2009 et 2014 on constate un désabonnement massif aux versions papier des revues (-55%) au profit des versions électroniques (+33%). Pourtant on assiste dans le même temps à une **augmentation significative des prix des abonnements** aux principales maisons d'édition : +21% pour Elsevier, +32% pour Springer, et **+61% pour l'IEEE**, en seulement cinq ans.

Cela donne des chiffres hallucinants du côté des maisons d'édition, comme le confirme le rapport financier de Reed-Elsevier (maison mère de Elsevier) pour l'année 2013. Dans ce rapport on apprend que les revues électroniques représentent maintenant plus de 80% de leur chiffre d'affaires, et que leurs profits sont aux environs de 30% avec des revenus de l'ordre de 7 milliards d'euros et un **bénéfice opérationnel de plus de 2 milliards d'euros**.

Ces profits sont évidemment l'écho de la ruine des bibliothèques de recherche. On voit ces dernières années pour les premières fois à des désabonnements massifs pour des raisons de manque de budget. Cela a par exemple été le cas avec l'Université Pierre et Marie Curie[14], pour qui l'abonnement au « Big Deal » d'Elsevier n'était plus soutenable (les maisons d'édition s'arrangent pour vendre des gros bouquets de revues plutôt que de laisser les établissement sélection-

[11] https://www.sup.adc.education.fr/asibu/
[12] http://www.dgdr.cnrs.fr/dcif/chiffres-cles/comptes-2013/default.htm
[13] http://www.telecom-paristech.fr/
[14] http://www.upmc.fr/

ner seulement celles dont ils auraient besoin). En 2009 cela faisait plus de dix ans que l'UPMC payait plus d'un million d'euros par an à Elsevier. L'abonnement a été reconduit un an suite à une concession significative d'Elsevier dans la négociation et par solidarité avec les autres membres de Couperin[15] (consortium unifié des établissements universitaires et de recherche pour l'accès aux publications numériques, qui est récemment passé de 147 à 642 établissements membres). Mais en 2010, suite à une large consultation des chercheurs et au vote à l'unanimité de son Conseil scientifique, l'UPMC a décidé de se désabonner d'Elsevier. Finalement, Elsevier est revenu à la charge en acceptant une réduction d'environ 30% (du prix, mais aussi du nombre de revues dans le bouquet), preuve que les universités ont le pouvoir de renverser en leur faveur le rapport de force dans les négociations avec les maisons d'édition. Depuis l'UPMC s'est désabonnée de *Science*[16] et l'Université Paris Descartes[17] de *Nature*[18] qui sont (à tort ou à raison) considérées comme les deux plus prestigieuses revues généralistes.

Ces cas ne sont pas isolés, il se passe la même chose en Belgique, en Angleterre, aux États-Unis, ..., partout dans le monde. En Allemagne, le cas de l'Université de Konstanz[19] est frappant. Constatant les prix exorbitants des revues Elsevier qui leur coûtent plus de 3400 € par an et par revue (ce qui est trois fois plus cher que la deuxième maison d'édition en terme de prix), et cela faisant suite à une augmentation de plus de 30% des tarifs de la maison d'édition sur les cinq dernières années, l'Université de Konstanz à décider de se désabonner des revues Elsevier. Le mot d'ordre était très clair : « Stop paying twice ». Un appel interne a été lancé pour demander aux chercheurs de suivre l'initiative The Cost of Knowledge[20], et à l'utilisation en priorité des versions des articles disponibles en libre accès, avant de se rabattre sur l'achat d'articles individuellement.

F.2.7 Les maisons d'édition sont des crapules

La mise en place de cette situation abracadabrante s'explique malheureusement assez bien. Depuis des années, contrairement à ce qu'on pourrait imaginer au vu de leur mission, **le cœur de métier des maisons d'édition est la négociation**. Elles font mener par des professionnels des négociations volontairement lentes et complexes, en ayant en face d'elles des individus qui le plus souvent ne sont pas négociateurs ni même commerçants de profession. Ces gens-là sont des bibliothécaires, des conservateurs, des chercheurs, etc. En plus de cela, les maisons d'édition, en faisant appel au secret commercial, rendent entièrement confidentielles leurs négociations avec les institutions publiques. Ainsi, les montants des contrats sont secrets, et révéler les abus des maisons d'édition en terme de tarification est passible de poursuites pour les personnes qui rompraient ce contrat. La confidentialité de ces contrats est possible car les négociations avec les maisons d'édition échappent aux règles des marchés publics, pour des raisons étranges de concurrence et de propriété

[15] http://www.couperin.org/
[16] https://fr.wikipedia.org/wiki/Science_%28revue%29
[17] http://www.univ-paris5.fr/
[18] https://fr.wikipedia.org/wiki/Nature_%28revue%29
[19] http://www.uni-konstanz.de/en/welcome/
[20] http://thecostofknowledge.com/

intellectuelle (cf. article 35 du Code des marchés publics[21]). Ce qui est encore plus fou est que la clause de confidentialité est elle-même sous le sceau du secret dans certains cas, comme on l'apprend avec horreur à la lecture du très bon article « Elsevier journals — some facts[22] » de Timothy Gowers[23]. Il faut se rendre compte que, jusqu'à récemment en tout cas (c'est sûrement toujours le cas), les seuls juristes du CNRS par exemple étaient des spécialistes du droit des brevets ; certainement pas des gens dont ce serait le métier de négocier et vérifier les contrats avec les maisons d'édition.

F.2.8 Liberté, égalité, sororité

En réalité, le coût en euros effrayant du système de publication n'est pas le vrai problème. Il s'agit en fait d'un symptôme du problème réel qui est la privatisation de la conservation et de la diffusion des découvertes et des inventions de l'humanité.

Il y a évidemment d'autres symptômes tout aussi importants que la dépense d'argent public au profit des actionnaires des maisons d'édition. En effet, en plus des symptômes financiers, on compte au moins deux importants symptômes sociaux. Le premier est la difficulté de l'accès citoyen aux résultats de la recherche, alors que celle-ci est financée par l'argent public, y compris dans des domaines très importants comme la médecine ou la pharmacie. Le second est l'inégalité entre les étudiants et les chercheurs dans les universités, non seulement à l'échelle d'un pays, mais aussi dans le monde : les études et la recherche dans les universités des pays émergents, qui ont peu de moyens, ne devraient pas en plus être pénalisées par l'impossibilité d'avoir accès aux résultats de la recherche ailleurs dans le monde.

Le libre accès se pose comme une solution à la plupart de ces symptômes, voire comme une solution au problème, selon comment et à quel point on décide de l'appliquer.

F.3 Le libre accès

Le libre accès, c'est la mise à disposition des articles en accès libre, gratuit et illimité via Internet, sans restriction de paiement ou d'abonnement pris auprès des maisons d'édition.

Il est évident que cela résout presque entièrement les symptômes que l'on a soulevés jusque là : la dépense immodérée d'argent public, l'accès citoyen, l'égalité entre étudiants et chercheurs des différentes universités à travers le monde. Cependant, il serait mentir de dire que ça résout entièrement les problèmes, comme nous allons le voir en étudiant ce qu'est le libre accès en pratique, mais aussi ce que ce n'est pas.

F.3.1 Comment le libre accès est-il rendu possible ?

Avant de regarder en détail de quoi il s'agit en pratique, il est nécessaire de s'intéresser à la faisabilité du libre accès. En effet, il faut réunir deux conditions pour rendre

[21] http://www.legifrance.gouv.fr/affichCodeArticle.do?cidTexte=LEGITEXT000005627819&idArticle=LEGIARTI000024506918

[22] http://gowers.wordpress.com/2014/04/24/elsevier-journals-some-facts/

[23] http://gowers.wordpress.com/

le libre accès possible.

La première, c'est de pouvoir diffuser les articles à bas coût dans le monde entier. Cela est comme on l'a déjà dit possible grâce à Internet.

La seconde, c'est le consentement des détenteurs des droits des articles. Pour les anciens articles, ces droits ont été cédés aux maisons d'édition comme on l'a vu dans la section F.2.4 « Le partage des coûts jusqu'à la publication » (encore que légalement ça n'est peut-être pas clair partout, cf. la remarque F.2 sur la cession du copyright en France). Si on veut faire ça dans les règles (on verra plus tard que ça n'est pas nécessairement la bonne solution), il faut donc l'accord des maisons d'édition pour diffuser ces articles, qu'elles possèdent jusqu'à 70 ans après la mort des auteurs à cause des lois absurdes sur le droit d'auteur.

En revanche, pour les nouveaux articles, les détenteurs du copyright sont par défaut les chercheurs qui les ont écrits (ou leurs institutions). Si ces derniers (ou celles-ci) choisissent de conserver leur copyright plutôt que de le céder à une maison d'édition, ils peuvent faire le choix eux-mêmes de mettre leur article en libre accès.

F.3.2 L'intérêt du libre accès pour les auteurs

La question qui reste à se poser est alors de savoir pourquoi les chercheurs, auteurs des articles, auraient intérêt à les mettre en libre accès plutôt que de les céder à une maison d'édition.

Il faut comprendre que pour les chercheurs il n'y a aucun intérêt financier en jeu. Dans aucun des deux cas les chercheurs ne perçoivent d'argent, ni à la publication d'un article, ni lorsque celui-ci est vendu.

En fait, un chercheur écrit des articles pour plusieurs raisons. La première est de faire connaître ses résultats, parce qu'il les pense intéressants et importants au moins pour la communauté de son domaine de recherche. La seconde, c'est de se faire connaître lui. La carrière d'un chercheur, et donc les moyens dont il va disposer pour mener à bien ses recherches, et la liberté qu'il aura de faire vraiment ce qu'il désire (grâce à sa renommée et/ou à l'obtention d'un poste permanent par exemple), avancent avec la reconnaissance de ses travaux par la communauté des chercheurs, en particulier par ceux de son domaine.

Vous ne trouverez donc aucun chercheur (ou presque ?) qui refuserait que ses articles soient disponibles plus facilement et à plus de monde. Aucun ne refusera d'être plus lu et donc d'avoir plus de retours pour améliorer ses articles, et par un cercle vertueux de gagner ainsi plus de reconnaissance. C'est ça que leur offre l'option du libre accès.

F.3.3 Les difficultés de la transition vers le libre accès

Les chercheurs, surtout les jeunes chercheurs qui doivent encore faire leurs preuves, ont besoin pour leur carrière de publier des articles dans des revues prestigieuses de leur domaine. Malheureusement, le prestige est quelque chose qui a beaucoup d'inertie, et les revues les plus prestigieuses sont souvent les plus anciennes et sont donc pour la plupart détenues par des maisons d'édition.

Quand c'est le cas, ces revues ne sont que très rarement en libre accès. À l'heure actuelle, il est malheureusement déraisonnable pour un chercheur débutant de résoudre ce conflit d'intérêts en faveur du libre accès. C'est malheureux, comme

on l'a expliqué précédemment dans un aparté sur la bibliométrie (cf. Remarque F.1), mais nécessaire. Le problème, c'est qu'il est rare qu'un article soit écrit par un seul chercheur, et c'est encore plus rare pour un jeune chercheur (peut-être à l'exception des mathématiques), qui co-écrit le plus souvent avec ses encadrants (de thèse[24] ou de post-doc[25]). Ces encadrants sont des chercheurs qui ont déjà fait leurs preuves mais qui se retrouvent dans la situation de devoir choisir entre publier dans des revues fermées mais prestigieuses, ou de pénaliser leurs jeunes co-auteurs. Il y a donc **une situation de prise en otage par les maisons d'édition, à cause de l'utilisation de la bibliométrie**.

Il ne faut pas perdre espoir pour autant. D'une part, le passage au libre accès est justement l'occasion de remettre en question les méthodes bibliométriques, voire de remettre en question leur utilisation. D'autre part, comme on va le voir dans la suite, il y a plusieurs façons de mettre ses travaux de recherche en libre accès, et certaines ne dépendent pas de la revue dans laquelle on publiera finalement. Mais avant, il est important de répondre aux craintes les plus courantes sur le libre accès.

F.3.4 Ce que le libre accès n'est pas

Dans cette section, on essaye de lister les craintes qui existent vis-à-vis du libre accès et d'y répondre.

1. Ce n'est pas un moyen de se passer de l'évaluation par les pairs.

   Ceci est très important. Certains détracteurs du libre accès essayent de faire croire que le libre accès est synonyme de publications de mauvaise qualité puisqu'elles ne subiraient à aucun moment d'évaluation par les autres experts des domaines concernés. Il est évident que ceci est entièrement faux, puisque le libre accès est une question indépendante, totalement orthogonale à la façon de sélectionner les articles. Le libre accès est compatible avec toutes sortes d'évaluations par les pairs : la disponibilité des articles en libre accès n'exclut en aucun cas de les évaluer et de mettre en avant les meilleurs.

   Remarque F.3. John Bohannon essaye de prouver le contraire dans un torchon intitulé « Who's afraid of peer-review?[26] » qu'il a écrit pour le magazine d'actualité scientifique *Science*. Sa méthode de comparaison des revues en libre accès et de celles qui ne le sont pas est assez remarquable. Son test consiste à envoyer un article de mauvaise qualité avec de faux résultats à plein de revues en libre accès avec frais de publication pour les auteurs, et de voir combien vont l'accepter. Ce qu'il constate, c'est que plus de la moitié à peu près de ces revues ont accepté l'article sans signe d'évaluation par les pairs. Il en conclue donc que les revues en libre accès sont pour la plupart de mauvaise qualité.
   Cette conclusion est éminemment ridicule. Premièrement, s'il s'agit de comparer les revues en libre accès et celles qui ne le sont pas, il faudrait aussi faire le test sur celles qui ne le sont pas, or ici ce n'est pas fait, on suppose simplement sans argumenter (et ça serait difficile) que les revues en accès fermé sont de bonne qualité. Deuxièmement, le test qui est fait n'a aucun sens. Aucun

[24] https://fr.wikipedia.org/wiki/Doctorat_%28France%29
[25] https://fr.wikipedia.org/wiki/Chercheur_postdoctoral
[26] http://www.sciencemag.org/content/342/6154/60.full

chercheur n'envoie ses articles au hasard dans des revues inconnues. Quand un article est soumis à une revue, on la choisit en connaissance de cause, selon deux critères aussi importants l'un que l'autre (et qui vont souvent de pair) : d'un côté le prestige de la revue, car plus elle est prestigieuse plus elle sera lue, et le but est évidemment d'être le plus lu possible par les autres chercheurs de son domaine, qui sont au courant de la qualité des différentes revues ; de l'autre côté pour le sérieux de la revue, parce que l'étape de relecture et d'évaluation est très important pour les auteurs. D'une part pour les rassurer sur la qualité de leurs travaux (on a toujours peur d'avoir raté quelque chose), et d'autre part car les auteurs d'un article ne voudraient surtout pas qu'il soit publié dans une grande revue s'il reste des erreurs dedans. On note que des maisons d'édition comme Elsevier estiment aujourd'hui que travailler le texte n'est plus de leur ressort et que « s'il y a des coquilles, des images dégueulasses, des erreurs dans les références, c'est la faute de l'auteur » comme on l'apprend dans ce témoignage édifiant[27]. Troisièmement, la conclusion à tirer de son expérience est en fait que **la pression bibliométrique qui pèse sur les chercheurs** (« *publish or perish* »), couplée à leur désir naturel et souhaitable d'être plus lus et donc de publier en libre accès, **a provoqué le développement d'un business malsain de la publication** (on rappelle qu'il a choisi des revues avec frais de publication pour les auteurs !) qui nuit directement au monde de la recherche.

2. Ce n'est pas une réforme du copyright ou du droit d'auteur.

   Aucune réforme du copyright n'est nécessaire pour passer au libre accès. Même les *licences libres* que l'on peut choisir d'utiliser (on verra cela dans la suite du dossier) se fondent sur le copyright tel qu'il existe actuellement. En ce qui concerne le droit d'auteur, il n'est absolument pas remis en question non plus puisqu'on ne parle pas ici d'œuvres qui font toucher une *redevance* (*royalties* en anglais) à leurs auteurs. Cependant, il n'est pas totalement inenvisageable de faire coexister le libre accès et les redevances du droit d'auteur : il existe des livres qui se vendent en version imprimée de manière traditionnelle mais qui sont en libre accès au format électronique, y compris des livres académiques, et cela n'empêche pas les versions imprimées de se vendre convenablement, comme l'ont montré le best-seller de Cory Doctorrow, Little Brother[28], ou plus modestement les livres libres de la collection Framabook[29].

3. Ce n'est pas un déni des coûts de publication.

   Évidemment, personne ne prétend que publier n'a aucun coût. L'accent est mis sur le fait que ces coûts sont déjà en grande partie assumés publiquement puisqu'ils font partie du travail des chercheurs (recherche, rédaction, comité éditoriaux, comité de lecture), et qu'il est aujourd'hui possible de baisser considérablement les coûts sur le reste de l'activité de publication, notamment grâce à Internet et en ne cherchant pas à maintenir un certain taux de profit.

4. Ce n'est pas un façon de diminuer le contrôle des auteurs sur leur travail.

[27] http://www.le-tigre.net/De-la-mondialisation-appliquee-a-l.html
[28] https://en.wikipedia.org/wiki/Little_Brother_%28Doctorow_novel%29
[29] http://framabook.org/

Au contraire, puisqu'il s'agit de leur laisser le copyright de leurs articles plutôt que de le céder à une maison d'édition. De plus, certaines des maisons d'édition en accès fermé tiennent tellement au monopole qu'elles ont sur les articles qu'elles publient, que leur politique est d'interdire la soumission d'articles qui ont déjà été rendu public sur Internet, comme on l'apprend ici[30]. Avec le libre accès, les chercheurs peuvent mettre au plus tôt leurs articles en ligne (avant même l'évaluation par les pairs, en prenant soin de préciser qu'il s'agit d'un preprint) et ainsi s'assurer d'avoir la paternité des idées qu'ils y développent afin d'en être crédités.

5. Ce n'est pas une façon de banaliser le plagiat.

   Ce n'est pas parce qu'un article est disponible en libre accès qu'il est dans le domaine public. Et donc il est tout aussi interdit de plagier son contenu ou de le réutiliser sans attribution, c'est-à-dire sans mentionner les auteurs et citer l'article d'où le contenu provient. Le plagiat est de toutes façons puni bien plus sévèrement par les mœurs que par la loi, et à ce niveau-là non plus le libre accès ne change rien. De plus, même si un article tiers est dans le domaine public, il est généralement interdit d'en copier des morceaux, par exemple dans une thèse. Cette interdiction est le fait des règles de l'institution où la thèse est effectuée, et pas du copyright ou même des mœurs.

6. Ce n'est pas un appel généralisé au boycott des maisons d'édition.

   Les maisons d'édition traditionnelles, même si elles vont être amenées à s'adapter ou à disparaître, peuvent tout à fait coexister avec le libre accès comme on le verra dans la section suivantes sur les différents modèles de libre accès.

7. Ce n'est pas uniquement un moyen d'ouvrir la recherche aux citoyens.

   Même si cet aspect est très important, il y a toujours des gens pour dire que finalement, les gens qui ne sont ni étudiants ni chercheurs ne s'intéressent pas aux résultats de la recherche. Ils en concluent donc le libre accès ne sert à rien puisque les étudiants et les chercheurs ont déjà accès aux publications via leurs universités ou instituts de recherche. Le problème avec ce raisonnement, c'est que même si la première proposition était vraie, ce qui n'est pas le cas, la seconde serait fausse quand même. En effet, les universités ne sont pas sur un pied d'égalité vis-à-vis des abonnements aux revues dont elles disposent. C'est vrai entre universités d'un même pays et encore plus à l'échelle mondiale, avec des moyens conséquemment moins importants dans les pays en voie de développement. Il y a donc un problème de justice économique et sociale à résoudre, et le libre accès est un moyen d'y parvenir.

8. Ce n'est pas la solution à tous les problèmes.

   Malheureusement, le libre accès n'est pas la solution à tous les problèmes, sinon on parlerait d'*accès universel*. Le libre accès nécessite d'avoir accès à Internet. Il ne permet pas de contourner le filtrage et la censure du réseau. Il ne résout pas la barrière de la langue, la vaste majorité des articles étant écrit en anglais. Il y a bien sûr une barrière d'accessibilité, les sites web n'étant

[30] https://en.wikipedia.org/wiki/List_of_academic_journals_by_preprint_policy

pas toujours parfaitement conçus pour les personnes ayant un handicap, et le format PDF aujourd'hui dominant ne l'étant pas non plus.
Il faut remarquer que si le libre accès n'est pas la solution à ces problèmes, l'accès fermé traditionnel ne les résout pas non plus.

F.3.5 Différents modèles de libre accès

Un article peut être rendu accessible librement par trois voies principales.

1. La *voie verte* (*green open access* en anglais).

   Cette voie est celle de l'*auto-archivage* (*self-archiving* en anglais). Le principe est que les chercheurs eux-mêmes vont mettre leurs articles sur un *dépôt* (*repository* en anglais) prévu à cette fin. Ces dépôts peuvent être thématiques, institutionnels, ou nationaux. Ils sont pour la plupart gérés par des institutions publiques et ont pour mission la distribution et la conservation dans le temps des articles qui leur sont confiés. Certains d'entre eux acceptent tous les articles et permettent de préciser où ils ont été publiés quand c'est le cas, et d'autres n'acceptent d'héberger uniquement les articles qui ont déjà été publiés par ailleurs. Les dépôts sont capables de garder la trace des versions successives d'un article. Ils peuvent aussi communiquer entre eux (pour s'échanger des méta-données par exemple) grâce au protocole OAI-PMH[31]. La plupart des revues, y compris les revues en accès fermé, autorisent les auteurs à auto-archiver les preprints de leur article, parfois après une période d'embargo qui varie généralement entre six mois et un an après la parution de l'article dans la revue (on peut vérifier cela au cas par cas sur SHERPA/RoMEO[32]). C'est grâce à cette méthode que presque 100% des nouveaux articles (depuis plusieurs années) sont disponibles en libre accès dans certains domaines comme la physique théorique.
   Le dépôt le plus connu et le plus vieux est arXiv[33], qui est un dépôt thématique pour la physique, les maths, et l'informatique. En France le plus gros est HAL[34] (Hyper-Articles en Ligne), qui accepte tous les domaines de recherche, et dispose de sous-portails dédiés pour certains établissements et instituts de recherche.

2. La *voie dorée* (*gold open access* en anglais).

   Ici, le libre accès se fait directement par la revue. À la place de vendre l'article, la maison d'édition fait payer les auteurs (ou leurs institutions) pour sa publication. La plupart des revues se disant en libre accès ou proposant le choix du libre accès aux auteurs chez les maisons d'édition traditionnelles pratiquent la voie dorée. Ce n'est pas une solution satisfaisante car les frais pour les auteurs sont exorbitants (souvent aux alentours de 3000 €). De plus, les bibliothèques universitaires et de recherche sont toujours obligées de s'abonner aux revues même si elles proposent ce choix. En effet, tous les auteurs n'ont pas forcément choisi de payer pour l'option du libre accès, ce qui rend l'abonnement

[31] https://fr.wikipedia.org/wiki/Open_Archives_Initiative_Protocol_for_Metadata_Harvesting
[32] http://www.sherpa.ac.uk/romeo/
[33] http://arxiv.org/
[34] http://hal.archives-ouvertes.fr/

indispensable pour lire une partie des articles de la revue.
Malheureusement, **le lobby des maisons d'édition est très puissant et il y a à cause de lui une confusion entre le libre accès et la voie dorée**, à tel point qu'on trouve dans les bourses européennes de financement de projet de recherche des crédits servant à payer ces tarifs déraisonnables. Ces crédits importants, mobilisés dans l'intention louable d'aller vers plus de libre accès, seraient bien mieux employés à favoriser une vraie offre et non cette mascarade.

3. La *voie diamant* (*diamond open access* en anglais).

   Ici aussi, le libre accès se fait directement par la revue. En revanche, il n'y a pas de frais pour les auteurs. Les revues pratiquant la voie diamant sont quasiment toutes nées à l'ère d'Internet. Certaines sont ce qu'on appelle des *épi-revues*, c'est-à-dire qu'elles fonctionnent en collaboration avec les dépôts (comme ceux permettant la voie verte). Le principe des épi-revues est que les auteurs déposent leurs articles dans les dépôts, et c'est à partir de là que le processus d'évaluation par les pairs prend place, garantissant la qualité et la pertinence des articles. Les articles qui sont acceptés pour publication dans l'épi-revue sont ensuite mis à jour sur le dépôt : pour chacun, une nouvelle version prenant en compte l'évaluation par les pairs, incluant les méta-données de publication (ISSN, date, etc.), et estampillée par l'épi-revue comme la version acceptée pour publication, est alors téléversée sur le dépôt. Enfin, l'épi-revue publie une liste de liens pointant directement sur la version acceptée des articles. Ce modèle permet de se passer entièrement des maisons d'édition dans les domaines où il n'y a plus de travail de mise en page important (tous ceux où les chercheurs utilisent des traitements de texte avancés, comme LaTeX, pour mettre en page leurs articles), ce qui est le cas de plus en plus de domaines (notamment grâce au progrès fait par les logiciels de traitement de texte de type LibreOffice ou Word).

   Dans le modèle diamant du libre accès, d'une part **le comité éditorial est doué d'une existence légale et il détient la revue** et d'autre part **les auteurs conservent le copyright de leurs articles**. Actuellement, les comités éditoriaux n'ont pas d'existence propre, les revues sont détenues par les maisons d'édition, et leurs contenus aussi.
   C'est le modèle à préférer, celui pour lequel il faut lutter.

En parallèle de ces trois voies, on peut faire une autre distinction dans les modèles de libre accès, entre la gratuité seule d'un côté, et l'utilisation de licences libres de l'autre. Peut-être qu'il serait plus judicieux de parler d'*accès ouvert* pour le premier cas et d'*accès libre* pour le second.

Lorsqu'un article est mis à disposition gratuitement, par défaut, les détenteurs du copyright sur l'article ne concèdent aucun droit à ses lecteurs ; ces détenteurs sont les auteurs ou les institutions dans lesquelles ils travaillent, ou celles à qui ils auraient cédé leur copyright. Cela veut dire qu'il obéit à la règle « *tous droits réservés* ». Il est donc interdit d'en faire usage pour autre chose que ce pour quoi il a été mis à disposition : sa lecture. À l'exception des droits dit de « *fair use* » (« exceptions prévues par l'article L122-5 du Code de propriété intellectuelle » en français), c'est-à-dire les droits de courte citation, de parodie, de copie réservée à

l'usage privé, etc. sous réserve d'indiquer clairement les noms des auteurs, tout le reste est interdit sans avoir obtenu l'accord explicite des détenteurs du copyright. Cela signifie qu'il est par exemple interdit de donner une copie d'un article à un collègue ou à des étudiants.

Lorsqu'un article est distribué sous licence libre, les détenteurs du copyright donnent par avance bien plus de possibilités aux lecteurs. L'article obéit alors à la règle « *certains droits réservés* ». Les licences libres pour le contenu s'inspirent de celles qui ont fait le succès du logiciel libre[35]. Les plus connues et les plus utilisées sont les licences *Creative Commons*[36], car elles ont été rédigés par des juristes et sont donc valides dans la plupart des législations, en plus d'être avantageusement disponibles en trois versions : la version légale, compliquée mais précise ; la version simplifiée, lisible et compréhensible par tout le monde ; et la version électronique, permettant aux machines de comprendre la licence (par exemple pour les moteurs de recherche). La licence libre la plus appropriée aux articles de recherche, et celle qui est conseillée par la plupart des initiatives existantes pour le libre accès, est la licence « Creative Commons Attribution[37] », aussi appelée « CC-BY ». Cette licence permet de partager (copier, distribuer et communiquer le matériel par tous moyens et sous tous formats) et adapter (remixer, transformer et créer à partir du matériel), sous réserve d'attribution. C'est-à-dire qu'il est nécessaire pour jouir de ses droits supplémentaires de créditer l'article original, en donnant le nom de ces auteurs, son titre, et en précisant si il a été modifié et sans laisser entendre que les modifications sont soutenues par les auteurs originaux. Cela autorise par exemple les traductions, les changements de format (il faut penser dès aujourd'hui à l'après-PDF), une amélioration (par exemple, corriger des fautes de langue avant de le distribuer, ou encore améliorer la qualité de certaines figures), ou de faire des compilations d'articles sur un même sujet sous forme de livres.
Ce qui est intéressant, c'est que les licences Creative Commons peuvent aussi s'appliquer aux jeux de données, aux illustrations, aux vidéos, au code source, etc. qui peuvent à l'ère d'Internet accompagner les articles pour les rendre plus complets.

F.3.6 Comment passer au libre accès ?

Il y a plusieurs façons de passer au libre accès.

1. Par des efforts individuels.

   Chaque chercheur est responsable de ce qu'il choisit de faire avec ses articles. Il est toujours utile de discuter du libre accès avec ses co-auteurs quand ceux-ci ne sont pas déjà en sa faveur.
   Il est aussi toujours utile de remettre en question les contrats des maisons d'édition. Quand un article est accepté dans une revue à accès fermé, il faut toujours demander s'il serait possible, plutôt que de céder le copyright, de le conserver et de donner à la place une licence de distribution à la maison d'édition, de préférence une licence libre type CC-BY.
   Dans tous les cas, si la revue est en accès fermé il faut mettre au plus vite l'article à disposition sur sa page web et/ou l'auto-archiver dans un dépôt,

[35] https://fr.wikipedia.org/wiki/Logiciel_Libre
[36] https://creativecommons.org/
[37] https://creativecommons.org/licenses/by/4.0/deed.fr

sans nécessairement respecter la période d'embargo de la maison d'édition. En effet, les risques sont limités : d'une part, on a déjà vu que la cession du copyright a une valeur légale douteuse (cf. la remarque F.2) en France, d'autre part aucune maison d'édition n'oserait s'attaquer à un chercheur pour cette raison seulement, car cela lui donnerait une bien trop mauvaise image et elle risquerait une levée de boucliers de la part du monde académique. Ce n'est, à ma connaissance, jamais arrivé jusqu'à aujourd'hui.

2. Par des politiques d'institutions ou des politiques nationales.

   Quand les chercheurs d'une institution le décident collectivement dans les instances démocratiques de l'institution (Conseils de labo, Conseil scientifique), il est possible de mettre en place une politique de publication en libre accès. Par exemple l'institution peut décider qu'elle conserve le copyright des articles, et donc ne pas laisser d'autre choix aux maisons d'édition que d'accepter une licence de distribution plutôt que de récupérer le copyright. Cela permet à l'institution des chercheurs qui ont écrit un article de le mettre à disposition en libre accès sur un dépôt de l'institution quand l'article n'est pas dans une revue en libre accès.
   Il est important de noter que les maisons d'édition acceptent systématiquement ce genre de situation, car elles savent qu'elles sont bien plus dépendantes des auteurs que les auteurs ne sont dépendants d'elles. Le risque à s'engager dans une telle politique institutionnelle est donc faible, mais il est malgré tout évident qu'il appartient tout de même aux plus grosses et aux plus prestigieuses institutions d'ouvrir la voie, puisqu'elles ont plus de poid face aux maisons d'édition. Et justement, on peut observer que ce modèle fonctionne bien depuis qu'il a été mis en place dans certaines universités, y compris de grandes universités comme Harvard[38] ou le MIT[39], ou encore par des organismes comme l'OMS[40]. La liste est de plus en plus longue.

3. Par la prise d'indépendance par rapport aux maisons d'édition.

   Comme avec les épi-revues, il est possible de se passer entièrement des maisons d'édition, en auto-hébergeant les articles en ligne. Il est indispensable d'insister sur l'importance de la prise d'indépendance par rapport aux maisons d'édition. Actuellement, les maisons d'édition sont propriétaires des revues qu'elles distribuent. Cela signifie que le comité éditorial (qui on le rappelle, est composé de chercheurs ayant intérêt à passer au libre accès) ne peut pas partir avec la revue et son prestige. La seule solution actuellement pour prendre son indépendance est de créer une nouvelle revue, qui doit repartir de zéro.
   La réappropriation des revues par les chercheurs est donc indispensable pour mener à bien les deux batailles du mouvement pour le libre accès et du mouvement contre les effets néfastes de la bibliométrie. Les corrections et mises en forme typographiques ainsi que l'impression ne sont finalement que des services qui peuvent tout à fait être découplés du travail du comité éditorial et de celui du comité de lecture de la revue. Les économies conséquentes

[38] https://osc.hul.harvard.edu/policies

[39] https://libraries.mit.edu/scholarly/mit-open-access/open-access-at-mit/mit-open-access-policy/

[40] http://www.wellcome.ac.uk/News/Media-office/Press-releases/2014/WTP056351.htm

> que les universités et instituts de recherche pourraient réaliser en arrêtant de payer des abonnements à des prix indécents seraient largement suffisantes pour mener à bien ces missions par ailleurs. Cela peut être fait en interne par les presses universitaires, mais aussi par des entreprises de services spécialisées. Dans le cas de l'utilisation de licences libres, ces deux missions (que ce soit les corrections et mises en forme typographiques ou l'impression) pourraient même être menées à la demande, par ceux qui en ont besoin, puisque les auteurs ont déjà donné leur accord. Si la licence libre oblige ceux qui font des modifications à les partager avec la communauté (on appelle ces licences « *copyleft*[41] »), alors les frais induits par les corrections et mises en forme typographiques n'auront à être assumés qu'une seule fois, et seulement dans les cas où cela sera effectivement jugé utile (puisque sinon personne ne le fera). Une licence copyleft qui pourrait être utilisée pour les articles, les données, et le code issu de la recherche est la licence « Creative Commons Attribution et Partage dans les même conditions[42] », aussi appelée CC-BY-SA. On remarque que les licences qui ont fait le succès des plus grands logiciels libres (Linux, Mozilla Firefox, VLC, etc.) sont aussi des licences copyleft.

Tout ce que l'on vient de voir sont des moyens de se mettre au libre accès pour les nouveaux articles. Or il reste le problème de l'accès aux anciens articles. Pour cela, il y a besoin d'une coopération de la part des maisons d'édition qui détiennent légalement ces archives. Une solution possible serait de réformer le droit d'auteur, typiquement pour le limiter fortement dans le temps au moins pour les œuvres qui, comme les articles de recherche, sont principalement financées publiquement. Ce n'est pas irréaliste : par exemple, tout ce qui est produit par le gouvernement fédéral des États-Unis s'élève immédiatement dans le domaine public. Ce qui serait plus compliqué, c'est de rendre cela rétro-actif, car cela demanderait une volonté politique très forte et très interventionniste, ce qui n'est pas évident au niveau international. En attendant que le rapport de force vis-à-vis des lobbies de l'industrie de la publication scientifique penche en faveur du libre accès, il n'y a pas d'autre choix que de mener une guerilla[43] : ceux qui le peuvent doivent faire profiter les autres de leurs accès autant que possible, même si ça n'est pas censé être légal.

F.3.7 Quelques initiatives

En 2001, l'Initiative de Budapest pour l'accès ouvert[44] tente de définir ce qu'est le libre accès, et appelle à l'auto-archivage et à la création de revues alternatives en libre accès. Elle lance aussi les travaux de création d'un protocole, l'OAI-PMH (Open Archives Initiative Protocol for Metadata Harvesting[45]) qui va permettre aux différents dépôts de communiquer, notamment pour s'échanger des méta-données.

Cette initiative est suivie, deux ans plus tard en 2003, de la Déclaration de

[41] https://fr.wikipedia.org/wiki/Copyleft
[42] https://creativecommons.org/licenses/by-sa/4.0/deed.fr
[43] https://en.wikisource.org/wiki/Guerilla_Open_Access_Manifesto
[44] https://fr.wikipedia.org/wiki/Initiative_de_Budapest_pour_l%27acc%C3%A8s_ouvert
[45] https://fr.wikipedia.org/wiki/Open_Archives_Initiative_Protocol_for_Metadata_Harvesting

Berlin sur le libre accès à la connaissance[46]. Les signataires y réclament la mise à disposition en libre accès de la littérature scientifique mondiale et de l'ensemble des données et logiciels ayant permis de produire cette connaissance. Cette déclaration a aujourd'hui été signée par plus de 470 organisations[47] dont le CNRS, l'INSERM, l'Institut Pasteur, et l'UPMC.

À nouveau deux ans plus tard, naît la déclaration de Berlin III[48], qui prend des position plus fortes avec une exigence pour les chercheurs de déposer leurs travaux de recherche dans des dépôts, et de les encourager à publier dans des revues en libre accès. Elle a été signée entre autre par le CNRS, Inria, l'INSERM, et le CERN.

Ce genre d'initiatives se multiplie. En France le Centre pour la Communication Scientifique Directe[49] s'occupe de maintenir les dépôts nationaux HAL[50] et TEL[51] (« Thèse En Ligne »). Il est aussi responsable de projets comme Scienceconf[52], une plateforme de gestion de conférences qui permet de simplement les organiser de manière indépendante, ou Episciences[53], qui offre une plateforme de gestion d'épi-revues (y compris la gestion de l'évaluation par les pairs).

Dans les domaines des lettres, des sciences humaines et des sciences sociales, la situation actuelle est un peu meilleure. C'est en partie dû aux plus faibles enjeux financiers de ces domaines : en l'absence de lobbies industriels, les chercheurs disposent d'un champs d'action plus large. Mais ce n'est pas la seule raison : très tôt, certains chercheurs de ces domaines ont compris l'intérêt de fuir les maisons d'éditions traditionnelles, et l'opportunité offerte par l'arrivée d'Internet. Lorsque la bibliométrie a commencée à frapper, il n'a plus été question que des revues de « *rang A* ». Pour être de ce rang, une revue doit être internationale, et seule les revues en anglais étaient considérées comme pouvant l'être. Cela ne fait évidemment pas beaucoup de sens de publier en anglais quand on est spécialiste de l'ancien français, par exemple. De fait, les meilleures revues dans ces domaines étaient rarement en anglais. Des chercheurs ont donc profité de l'arrivée d'Internet et ont pris l'initiative dès 1999 de monter une plateforme leur permettant d'auto-gérer leurs revues, se passant ainsi totalement des maisons d'édition. Aujourd'hui, le projet revues.org[54] est international, et est même devenu incontournable. Il héberge plus de 400 revues dans 14 langues différentes, toutes en libre accès.

Dans mon domaine de recherche, en cryptologie, près de 100% des papiers sont disponibles en ligne avant leur publication (et sont mis à jour au fur et à mesure) sur le portail Cryptology ePrint Archive[55] de l'International Association for Cryptologic Research[56]. Cela permet d'avoir très tôt des retours de la communauté sur nos articles, avant même leur soumission à des revues, ce qui est très utile.

Cette idée est poussée encore plus loin par les partisans de l'« *open peer-review* »

[46] https://fr.wikipedia.org/wiki/D%C3%A9claration_de_Berlin_sur_le_libre_acc%C3%A8s_%C3%A0_la_connaissance
[47] http://openaccess.mpg.de/319790/Signatories
[48] http://www.eprints.org/events/berlin3/outcomes.html
[49] http://ccsd.cnrs.fr/
[50] http://hal.archives-ouvertes.fr/
[51] http://tel.archives-ouvertes.fr/
[52] http://www.sciencesconf.org/
[53] http://episciences.org/
[54] http://www.revues.org/
[55] http://eprint.iacr.org/
[56] http://www.iacr.org/

(*évaluation par les pairs ouverte* en français). Par exemple liberating research[57] propose aux chercheurs de mettre en ligne leurs preprints pour les faire évaluer par la communauté de manière ouverte, c'est-à-dire que les rapports d'évaluation des articles sont eux-mêmes publics, signés, et citables. Cela permet aux chercheurs d'être crédités pour leur travail d'évaluation. Le texte « Independent Peer-Review Initiative[58] » est une bonne lecture sur le sujet. Sur le sujet de l'auto-publication, qui va de pair avec l'évaluation ouverte, je conseille la lecture du texte « Academic self-publishing: a not-so-distant-future[59] ». Ces méthodes, en rendant public le processus d'évaluation par les pairs, permettent de le remettre au centre du système de publication scientifique, c'est à dire à sa place naturelle puisque il s'agit l'aspect le plus important scientifiquement du processus de publication.

F.4 Conclusions

Il y a deux problèmes principaux auxquels on s'est intéressé : d'un côté, celui de l'accès à tous à la connaissance, aux inventions et aux découvertes, aux savoirs et aux techniques ; de l'autre, celui de l'influence néfaste de la bibliométrie sur la recherche. On a vu que ces deux problèmes sont étroitement liés, et on a étudié les solutions offertes par le libre accès. Le libre accès permet de résoudre en majeure partie le premier problème, et est l'occasion de faire des avancées significatives sur le second, en particulier sous certaines des formes que le libre accès peut prendre (voie diamant, réappropriation des revues par les comités éditoriaux, utilisation de licences libres). Que ce soit grâce un mouvement politique ou à cause d'une nécessité économique, une chose est sûre : le monde de la publication scientifique est en train d'évoluer. Il ne tient qu'à nous d'en profiter pour le faire aller dans la bonne direction.

Rejoignez le mouvement et parlez-en autour de vous !

[57] http://www.liberatingresearch.org/
[58] http://www.openscholar.org.uk/independent-peer-review-initiative/
[59] http://www.openscholar.org.uk/academic-self-publishing-a-not-so-distant-future/

"Freedom is always the freedom of dissenters."

~ Rosa Luxembourg

Méthodes logicielles formelles pour la sécurité des implémentations cryptographiques

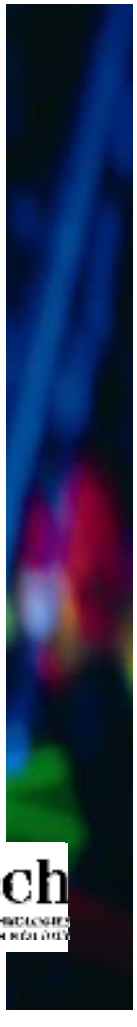
Pablo RAUZY

RÉSUMÉ : Les implémentations cryptographiques sont vulnérables aux attaques physiques, et ont donc besoin d'en être protégées. Bien sûr, des protections défectueuses sont inutiles. L'utilisation des méthodes formelles permet de développer des systèmes tout en garantissant leur conformité à des spécifications données. Le premier objectif de ma thèse, et son aspect novateur, est de montrer que les méthodes formelles peuvent être utilisées pour prouver non seulement les principes des contre-mesures dans le cadre d'un modèle, mais aussi leurs implémentations, étant donné que c'est là que les vulnérabilités physiques sont exploitées. Mon second objectif est la preuve et l'automatisation des techniques de protection elles-même, car l'écriture manuelle de code est sujette à de nombreuses erreurs, particulièrement lorsqu'il s'agit de code de sécurité.

MOTS-CLEFS: cryptologie, cannaux auxiliaires, injections de fautes, méthodes formelles.

ABSTRACT: Implementations of cryptosystems are vulnerable to physical attacks, and thus need to be protected against them. Of course, malfunctioning protections are useless. Formal methods help to develop systems while assessing their conformity to a rigorous specification. The first goal of my thesis, and its innovative aspect, is to show that formal methods can be used to prove not only the principle of the countermeasures according to a model, but also their implementations, as it is where the physical vulnerabilities are exploited. My second goal is the proof and the automation of the protection techniques themselves, because handwritten security code is error-prone.

KEY-WORDS: cryptology, side channels, fault injection, formal methods.



TELECOM ParisTech

EDITE

ParisTech