Practical Analysis of Symmetric and Asymmetric Standards in the White-Box Context

Agathe Houzelot

▶ **To cite this version:**

Agathe Houzelot. Practical Analysis of Symmetric and Asymmetric Standards in the White-Box Context. Computer science. Université de Bordeaux, 2024. English. NNT : 2024BORD0217 . tel-04819670

HAL Id: tel-04819670

https://theses.hal.science/tel-04819670v1

Submitted on 4 Dec 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE PRÉSENTÉE
POUR OBTENIR LE GRADE DE

DOCTEURE DE L'UNIVERSITÉ DE BORDEAUX

ECOLE DOCTORALE MATHÉMATIQUES ET INFORMATIQUE

SPECIALITÉ INFORMATIQUE

Par **Agathe HOUZELOT**

Analyse pratique de standards symétriques et asymétriques dans le contexte de la boîte blanche

Practical Analysis of Symmetric and Asymmetric Standards in the White-Box Context

---

Sous la direction de : **Arnaud CASTEIGTS**

Soutenue le 18 octobre 2024

Membres du jury :

M. Louis GOUBIN	Professeur, Université Versailles-St-Quentin-en-Yvelines	Rapporteur
M. Matthieu RIVAIN	Ingénieur de Recherche, CryptoExperts	Rapporteur
Mme. Marine MINIER	Professeure, Université de Lorraine	Examinatrice
Mme. Alice PELLET-MARY	Chargée de Recherche, Université de Bordeaux, CNRS	Examinatrice
M. Pascal DESBARATS	Professeur, Université de Bordeaux	Président du jury
M. Arnaud CASTEIGTS	Professeur, Université de Genève	Directeur

Membres invités :

Mme. Emmanuelle DOTTAX	Ingénieure de Recherche, IDEMIA	Co-encadrante
M. Christophe GIRAUD	Ingénieur de Recherche, IDEMIA	Co-encadrant

Remerciements

Au cours de cette thèse, j'ai eu la chance d'être entourée par des personnes formidables, dont la présence et le soutien ont été essentiels à la réalisation de ce projet.

Je tiens tout d'abord à exprimer ma profonde reconnaissance à mon directeur de thèse, Arnaud Casteigts, ainsi qu'à mes deux encadrants, Emmanuelle Dottax et Christophe Giraud, pour la confiance qu'ils m'ont accordée. Leurs conseils avisés, leur suivi régulier et nos nombreuses discussions m'ont beaucoup apporté tout au long de ces trois années.

Je remercie également Pascal Desbarats, Louis Goubin, Marine Minier, Alice Pellet-Mary, et Matthieu Rivain pour leur participation en tant que membres de mon jury de thèse. Un grand merci à Louis Goubin et Matthieu Rivain qui ont accepté de relire mon manuscrit durant les périodes de vacances.

Je suis également reconnaissante envers tous les membres du LaBRI, doctorants et permanents, avec qui j'ai partagé de nombreux repas et de grandes parties de bluff. Un merci tout particulier à Timothée Corsini, dont l'humour unique et les dessins improbables sur mon pauvre tableau blanc ont égayé mes journées au laboratoire.

Bien sûr, mes remerciements vont aussi à tous mes collègues du côté entreprise. C'est un véritable plaisir de les côtoyer chaque jour, même ceux où je ne fais que perdre au baby et où il n'y a pas de chocolatines au petit-déjeuner. En particulier, j'adresse un grand merci à tous ceux de la zone 4 – Guillaume Barbu, Yannick Bequer, Loana Brignon, Laurent Castelnovi, Thomas Chabrier, Nicolas Debande, Laurent Grémy, Claire Jauvion, Mael Lhostis, Sarah Lopez et Vladimir Sarde – pour la multitude de choses qu'ils m'ont apprises au niveau professionnel, mais aussi pour les sorties, les nombreux déjeuners, les pichets de chouffe partagés le soir au central et plus généralement tous les bons moments passés ensemble. Bien qu'il ne fasse plus partie de cette équipe, je tiens également à remercier Alberto Battistello, sans qui je n'aurais sans doute pas eu la chance de rentrer chez Idemia et de découvrir le monde de la boîte blanche.

À mes co-auteurs, et en particulier Laurent Castelnovi, Emmanuelle Dottax, et Christophe Giraud, je tiens à exprimer ma gratitude pour nos enrichissantes sessions de recherche. Les débriefings sur les potins qui précédaient chaque séance de travail avec Laurent étaient également très instructifs.

Sur un plan plus personnel, je suis infiniment reconnaissante envers mes amis et ma famille qui m'ont soutenue et rendue heureuse pendant ces trois années et depuis bien plus longtemps pour certains. Merci à mes parents, qui, en plus de m'avoir donné la vie, l'ont rendue merveilleuse. Je ne peux pas dire qu'ils m'aient transmis le goût des maths mais je leur dois presque tout le reste. Merci à mon frère, à ma sœur, à leurs conjoints et à leurs enfants pour les nombreux moments partagés. Merci également à mes grands-parents, avec une pensée particulière pour Pitel, qui nous a quittés avant la fin de ce parcours. Merci à Mallow et Murphy pour le réconfort qu'ils ne manquent jamais de m'apporter.

Et surtout, merci à Gautier, qui a su m'accompagner avec patience et amour, malgré le stress combiné de la thèse et de la préparation de notre mariage. Je ne pourrais rêver d'un meilleur partenaire de vie.

Analyse pratique de standards symétriques et asymétriques dans le contexte de la boîte blanche

Résumé : La cryptographie en boîte blanche vise à sécuriser les implémentations des algorithmes cryptographiques dans des environnements hostiles où l'adversaire peut potentiellement avoir un accès complet à l'implémentation et à son environnement d'exécution. Face à cet attaquant quasi omnipotent, toutes les solutions proposées à ce jour dans la littérature pour des cryptosystèmes standards sont considérées comme vulnérables. Cependant, dans la pratique, l'adversaire peut se heurter à certains obstacles pouvant compliquer l'application d'attaques théoriquement efficaces, tels qu'une limite sur le nombre d'exécutions avec une clé donnée ou des couches d'obfuscation obligeant à entreprendre une longue étape de rétro-ingénierie. Ainsi, le modèle de la boîte blanche semble parfois définir un attaquant excessivement puissant pour certains cas d'usage. Dans ce contexte, les entreprises développent des solutions propriétaires dont la conception reste secrète et qui sont spécifiquement adaptées à leurs besoins. Il est donc primordial d'étudier les attaques et les contre-mesures utilisables en pratique pour ces implémentations.

Dans cette thèse, nous nous concentrons sur deux cryptosystèmes standards largement utilisés en cryptographie symétrique et asymétrique, à savoir AES et ECDSA. Alors que la littérature sur AES est abondante, très peu de publications concernent les implémentations en boîte blanche d'ECDSA, malgré leur grande pertinence pour l'industrie. Pour ces deux cryptosystèmes, nous présentons des attaques efficaces en pratique, mettant en avant des caractéristiques telles que la possibilité d'automatisation, un nombre réduit d'exécutions et la non-nécessité de choisir les entrées du programme. En particulier, nous examinons les diverses vulnérabilités potentielles des boîtes blanches ECDSA et montrons que la plupart d'entre elles sont dues à l'absence de sources d'aléa fiables dans le contexte de la boîte blanche. Nous détaillons les attaques que nous avons effectuées pour casser les 97 implémentations candidates du concours WhibOx 2021. Nous montrons également comment des injections de fautes peuvent permettre à un attaquant de casser la toute première implémentation boîte blanche d'ECDSA publiée en 2020 par Zhou et coll., et proposons une contre-mesure qui n'augmente pas la taille du code. Étant donné qu'il n'existe aucune autre implémentation publique d'ECDSA, nous examinons également divers brevets pour nous donner une idée des contremesures utilisées en pratique dans les produits.

Concernant les boîtes blanches AES, nous proposons une nouvelle attaque très efficace qui nécessite très peu d'exécutions sur des entrées aléatoires. Nous étudions également la protection fournie par les encodages internes contre les attaques par canaux auxiliaires. Cette contre-mesure courante est utilisée sur les implémentations tabulées et consiste à appliquer des permutations aléatoires sur les variables sensibles pour les cacher à l'attaquant. Bien qu'il soit de notoriété publique que des encodages aléatoires puissent être cassés avec une grande probabilité, la question de l'existence d'une classe particulière d'encodages qui pourrait prévenir les attaques par canaux auxiliaires était toujours ouverte. Dans cette thèse, nous y répondons négativement et montrons que construire des encodages avec des propriétés spécifiques n'est pas une solution viable.

Mots-clés : Cryptographie en boîte blanche, cryptographie symétrique, cryptographie asymétrique, cryptosystèmes standards, sécurité pratique, canaux auxiliaires, injection de fautes.

Unité de recherche
LaBRI UMR 5800, 351 Cours de la Libération, 33405 Talence, France.

Practical Analysis of Symmetric and Asymmetric Standards in the White-Box Context

Abstract: White-box cryptography aims to secure implementations of cryptographic algorithms in hostile environments where the adversary may potentially gain full access to the implementation and its execution environment. Against this nearly omnipotent attacker, all solutions proposed to date in the literature for standard cryptosystems are considered vulnerable. However, in practice, the adversary may encounter obstacles that complicate the application of theoretically effective attacks, such as a limit on the number of executions with a given key or obfuscation layers forcing him to undertake a costly reverse-engineering phase. Therefore, the white-box model seems to define an attacker who is excessively powerful for several use-cases. In this context, companies develop proprietary solutions whose designs remain secret and are specifically tailored to their needs. It is thus crucial to study the attacks and countermeasures that can be practically applied to these implementations.

In this thesis, we focus on two widely used standard cryptosystems in both symmetric and asymmetric cryptography, namely AES and ECDSA. While the literature on AES is abundant, very few publications address white-box implementations of ECDSA, despite their high relevance for the industry. For both cryptosystems, we present real-life attacks, focusing on features such as the possibility of automation, a reduced number of white-box executions and no requirement for chosen inputs. Specifically, we examine the various potential vulnerabilities of ECDSA white-boxes and show that most of them stem from the lack of reliable sources of randomness in the white-box context. We detail the attacks that we carried out to break the 97 candidate implementations of the 2021 WhibOx contest. We also demonstrate how fault injections can break the very first white-box implementation of ECDSA published in 2020 by Zhou et al., and we propose a countermeasure that does not increase the size of the code. Given that there is no other public ECDSA implementation, we also review various patents to gain insights into countermeasures used in practice in products.

Regarding AES white-boxes, we propose a new and highly efficient attack that requires very few executions on random plaintexts. We also investigate the protection provided by internal encodings against side-channel attacks. This common countermeasure is used on table-based implementations and consists in applying random permutations on sensitive variables to obfuscate them. Although it is widely known that random encodings are broken with high probability, the question of whether a particular class of encodings could prevent side-channel attacks remained open. In this thesis, we answer it negatively and show that carefully crafting encodings with a specific property is not a viable solution.

Keywords: White-Box cryptography, symmetric cryptography, asymmetric cryptography, standard cryptosystems, practical security, side-channel attack, fault injection.

Research Unit
LaBRI UMR 5800, 351 Cours de la Libération, 33405 Talence, France.

Contents

Résumé étendu en français

Boîte noire, grise ou blanche

Les algorithmes cryptographiques sont principalement conçus pour être sécurisés dans le *modèle de la boîte noire*, où un attaquant est limité à l'observation de leurs entrées et sorties et ne dispose d'aucune information sur les variables intermédiaires. Ce modèle est pertinent lorsque les opérations cryptographiques se déroulent dans un environnement sécurisé et que seul le canal de transmission n'est pas fiable. Cependant, dans de nombreux scénarios, l'attaquant peut avoir accès à bien plus d'informations. En particulier, il peut parfois mesurer le temps d'exécution de l'algorithme, la consommation de courant de l'appareil dans lequel il est implémenté ou toute autre donnée obtenue à partir d'un canal auxiliaire [KJJ99]. Etant liées aux valeurs manipulées lors de l'exécution, ces informations peuvent permettre de retrouver la clé secrète si l'implémentation n'est pas protégée. Alternativement, l'adversaire peut intentionnellement altérer le comportement du dispositif physique et induire des erreurs lors de l'exécution de l'algorithme pour obtenir des informations supplémentaires [BDL97]. Par conséquent, les développeurs doivent mettre en place des contre-mesures spécifiques pour atteindre le niveau de sécurité initialement attendu. Ce contexte, où l'adversaire a un accès limité au dispositif et peut effectuer des attaques physiques, est décrit par le *modèle de la boîte grise*. Il est particulièrement pertinent dans le contexte des systèmes embarqués où la cryptographie est déployée sur des éléments sécurisés.

Enfin, le *modèle de la boîte blanche*, introduit il y a une vingtaine d'année par Chow et al. [CEJv02,CEJv03], considère un attaquant encore plus puissant. En effet, le développement récent d'algorithmes cryptographiques sur des dispositifs ouverts ne contenant pas nécessairement de matériel sécurisé, tels que les smartphones ou les objets connectés, offre de nombreuses opportunités à l'attaquant. Dans le modèle de la boîte blanche, l'adversaire contrôle tous les aspects de l'implémentation. La surface d'attaque est donc très large : il peut analyser le binaire, altérer l'exécution ou observer toutes les informations d'exécution, telles que les variables intermédiaires ou les adresses mémoire accédées, et ce pour toute entrée choisie. Bien sûr, toutes les attaques qui peuvent être effectuées dans les modèles de la boîte noire ou grise sont également des menaces potentielles. Dans un environnement si hostile, l'implémentation elle-même est la dernière ligne de défense contre l'extraction de clé. Par abus de langage, une implémentation d'un algorithme cryptographique conçue pour résister à un attaquant dans le modèle de la boîte blanche est communément appelée une *boîte blanche*. Elle doit être méticuleusement générée pour que les valeurs secrètes, telles que les clés, soient cachées à l'intérieur et difficiles à calculer.

Cas d'usage des boîtes blanches

Les cas d'usage des boîtes blanches sont aujourd'hui variés. Elles sont employées dans diverses applications fonctionnant dans des environnements non sécurisés, en particulier sur des téléphones mobiles qui ne disposent pas toujours de matériel sécurisé pour stocker les clés secrètes et effectuer des opérations cryptographiques. Les exemples typiques incluent – mais ne se limitent pas à – la gestion des droits numériques (DRM), les applications de paiement mobile et les portefeuilles de cryptomonnaie. Chaque utilisation correspond à un type d'attaquant avec des capacités potentiellement différentes. Par exemple, dans le cadre des DRM, le propriétaire légitime de l'implémentation, un client ayant payé un abonnement, pourrait être un attaquant potentiel et chercher à revendre la clé permettant de lire le contenu chiffré stocké sur la plateforme. En revanche, dans le cas du paiement mobile, le propriétaire légitime de l'application n'a aucun intérêt à ce que ses secrets soient divulgués à d'autres. L'attaquant est plutôt une personne extérieure, ayant potentiellement installé un virus sur le téléphone, mais n'étant probablement pas omnipotent, surtout si l'application est protégée par des mécanismes d'authentification ou d'autres mesures de sécurité.

Approche théorique

Une branche de la cryptographie en boîte blanche s'efforce de définir des notions de sécurité précises [SWP09, DLPR14, BI15, AABM20]. En effet, la résistance à l'extraction de clé est indispensable mais insuffisante pour de nombreux cas d'usage. Des propriétés telles que la confidentialité et l'intégrité des données peuvent également être nécessaires. De plus, les boîtes blanches doivent résister aux attaques par code-lifting, où le code est volé et utilisé sur un autre appareil sans chercher à comprendre son fonctionnement ou à récupérer la clé secrète. Des travaux théoriques définissant précisément les objectifs de la cryptographie en boîte blanche sont donc essentiels. Il reste ensuite à déterminer si les notions de sécurité ainsi définies sont atteignables ou non.

La plupart des recherches s'appuient sur des études théoriques dans un domaine connexe : l'obfuscation de programmes. De manière informelle, un obfuscateur est un compilateur qui prend en entrée un programme et en produit un nouveau ayant la même fonctionnalité mais qui est, en un certain sens, "inintelligible". Il existe différentes branches de l'obfuscation, dont l'obfuscation en boîte noire virtuelle (VBB), qui vise à produire un programme ne fournissant aucune information sur le programme original au-delà de ses entrées et sorties. En 2001, Barak et al. [BGI+01] ont démontré que certains programmes sont intrinsèquement non obfusquables au sens VBB. Cependant, il a été prouvé par la suite que d'autres programmes simples le sont effectivement [Wee05]. Déterminer si certains algorithmes cryptographiques sont VBB-obfusquables serait donc d'un grand intérêt, bien que cela semble peu probable. Néanmoins, il est à noter que l'obfuscation VBB n'est pas exactement ce que nous visons en cryptographie en boîte blanche. Si seule la clé secrète doit être protégée, fournir des informations "inutiles" à l'attaquant peut être acceptable. Saxena et al. [SWP09] ont tenté de formaliser la distinction entre les informations non boîte noire "utiles" et "inutiles", prouvant également certains résultats d'(im)possibilité. Ils ont montré que pour la plupart des programmes, certaines notions de sécurité peuvent être satisfaites dans le contexte de la boîte noire mais pas dans celui de la boîte blanche. Cependant, ils ont aussi démontré que certaines notions de sécurité intéressantes pouvaient être atteignables avec des chiffrements ad-hoc.

Approche pratique

En 2002 et 2003, Chow et al. ont proposé les deux premiers designs de boîtes blanches, respectivement pour le Data Encryption Standard (DES) [CEJv02] et l'Advanced Encryption Standard (AES) [CEJv03]. En un mot, leur idée consiste à protéger la clé secrète en l'intégrant dans des tables pré-calculées utilisées pendant l'exécution pour effectuer des opérations sensibles. Toutes les variables intermédiaires, les entrées et les sorties des tables, sont ensuite obscurcies par des bijections aléatoires appelées *encodages*. Ces dernières peuvent être classées en deux catégories : les *encodages internes* qui s'annulent les uns avec les autres, et les *encodages externes* appliqués à l'entrée et à la sortie de l'algorithme. Bien qu'ils augmentent la sécurité globale, les encodages externes modifient le comportement d'entrée/sortie du programme, ce qui est prohibitif dans de nombreux cas d'usage où des cryptosystèmes standards sont nécessaires pour des raisons d'interopérabilité.

Les deux premières implémentations de Chow et al. ont été cassées quelques années après leur publication [BGEC04, WMGP07, GMQ07]. Elles ont rapidement été suivies par de nombreux nouveaux designs [BCD06, XL09, Kar11, LLY14, BCH16] et attaques [DWP10, DRP13a, DRP13b, MGH09, DFLM18, BHMT16], la plupart concernant l'AES. Malheureusement, il n'existe à ce jour aucun design publié d'un algorithme cryptographique standard qui soit considéré comme sécurisé dans le modèle de la boîte blanche.

Pour des raisons d'interopérabilité et parce que plusieurs parties prenantes sont généralement impliquées dans une solution spécifique, l'industrie est peu susceptible d'adopter des cryptosystèmes ad-hoc pour de nombreux cas d'usage. Dans de telles circonstances, les entreprises sont contraintes de concevoir des solutions personnalisées pour répondre à la demande croissante de logiciels sécurisés. Leur objectif est de fournir une sécurité pratique en protégeant la boîte blanche contre les attaques automatisées, telles que celles héritées de la boîte grise, et en obfusquant le code pour forcer l'adversaire à passer par une phase de rétro-ingénierie longue et ardue avant de pouvoir rechercher des vulnérabilités.

Même si ces solutions pratiques peuvent être moins sécurisées que ce qui est attendu dans le modèle de la boîte blanche en théorie, il est important de noter que les adversaires réels peuvent également être beaucoup moins puissants que prévu initialement. En effet, les capacités et les objectifs des attaquants varient largement en fonction du cas d'usage. Par exemple, dans les applications de paiement mobile, l'adversaire est généralement une menace externe qui peut introduire un virus mais n'est probablement pas omnipotent. Des mécanismes d'authentification ou une mise à jour régulière de la clé peuvent permettre de limiter le nombre d'exécutions observables dans un délai donné.

Dans ce contexte, les concours WhibOx [Whi17, Whi19, Whi21, Whi24] ont été organisés dans le cadre des ateliers associés à CHES pour motiver la recherche pratique sur les boîtes blanches. À chaque édition, les développeurs étaient invités à soumettre le code source d'une implémentation conçue secrètement, tandis que les attaquants devaient tenter de la casser. Alors que les deux premiers concours ciblaient l'AES, les organisateurs ont choisi l'Elliptic Curve Digital Signature Algorithm (ECDSA) pour les deux dernières éditions. En effet, il existe un fort besoin de solutions ECDSA en boîte blanche pour l'industrie, et sécuriser ce schéma pose plusieurs nouveaux défis [DGH21]. Ces concours ont mis en avant une grande différence de maturité pour le design des boîtes blanches AES versus ECDSA : alors que trois implémentations de l'AES avient résisté jusqu'à la fin de l'édition de 2019, une vingtaine de jours après leur publication, la meilleure implémentation de l'ECDSA soumise à l'édition de 2021 n'a tenu que 35 heures. Il parait donc primordial de continuer à étudier les attaques et contremesures applicables à ces implémentations pratiques.

Contributions

Dans cette thèse, nous avons étudié à la fois l'AES et l'ECDSA, identifiant des attaques réalisables en pratique et analysant l'efficacité d'éventuelles contre-mesures.

Après quelques rappels sur les attaques par canaux auxiliaires et injection de fautes, nous présentons un court état de l'art sur les boîtes blanches AES, qui ont été largement étudiées dans la littérature. Nous proposons ensuite une nouvelle attaque qui nécessite avantageusement très peu d'exécutions sur des entrées aléatoires, puis nous étudions les attaques par canaux auxiliaires et en particulier la protection offerte par les encodages internes. Bien qu'il soit généralement admis qu'une implémentation de l'AES protégée par de petites bijections aléatoires puisse être cassée avec grande probabilité [RW19] par la Differencial Computation Analysis (DCA), une attaque par canaux auxiliaires, la question de l'existence d'une classe particulière d'encodages pouvant empêcher l'attaque restait ouverte. Dans cette thèse, nous y répondons négativement. Nous montrons que la corrélation entre les valeurs prédites et celles qui sont effectivement manipulées dépend en fait du spectre de Walsh des encodages utilisés, et bien que la sélection de bijections avec de petites transformées de Walsh puisse prévenir l'attaque classique, il est toujours possible de casser l'implémentation avec une variante. Par conséquent, les encodages internes sont insuffisants pour protéger les boîtes blanches AES contre les attaques par canaux auxiliaires, qu'ils soient choisis aléatoirement ou soigneusement sélectionnés.

La seconde partie de cette thèse concerne l'ECDSA et met en évidence les complexités supplémentaires inhérentes à ce schéma. Nous décrivons plusieurs attaques efficaces, notamment celles réalisées par l'équipe TheRealIdefix, dont nous faisions partie, et qui a cassé le plus grand nombre de candidats au concours WhibOx 2021. Nous montrons que les attaques par fautes sont particulièrement dévastatrices pour les versions déterministes de l'ECDSA, comme la plupart de celles implémentées lors du concours. De plus, nous fournissons un aperçu de la conception des deux candidats gagnants, dont les contremesures sont théoriquement intéressantes bien qu'insuffisantes pour contrer toutes les attaques automatisées.

Nous étudions également la seule autre source d'information publiquement accessible sur les boîtes blanches ECDSA standards : les brevets déposés par certaines entreprises. Bien que ces brevets puissent omettre des détails cruciaux sur l'implémentation ou le modèle de sécurité, ils restent des sources précieuses d'informations sur les méthodes mises en œuvre sur le terrain. Nous exposons donc dans cette thèse les principales idées pour contrer les diverses attaques pouvant cibler les boîtes blanches ECDSA, en soulignant leurs avantages et leurs limitations en termes de performances et de sécurité.

Enfin, nous présentons la première attaque sur le seul design de boîte blanche ECDSA publié à ce jour [ZBJ20]. Ce dernier nécessite l'aide d'un serveur lors du calcul d'une signature, ce qui limite son utilisation mais facilite sa sécurisation. La clé, entièrement contenue du côté du client, est protégée par des tables encodées dont la taille reste relativement petite grâce à l'utilisation du Residue Number System (RNS). Nous montrons que cette clé peut être efficacement récupérée par un attaquant induisant des fautes dans les calculs. Plus précisément, nous présentons deux attaques différentes impliquant la réinjection de valeurs spécifiques stockées lors d'une exécution précédente dans une nouvelle exécution. Nous proposons également une contre-mesure basée sur des encodages joints qui empêche les deux attaques sans impacter les performances du côté client.

En conclusion, cette thèse vise à faire progresser la recherche sur la cryptographie en boîte blanche en étudiant des attaques pratiques sur l'AES et l'ECDSA et des contremesures alignées avec les besoins industriels.

Abbreviations and Notations

List of Abbreviations

AES	Advanced Encryption Standard
ASLR	Address Space Layout Randomization
DCA	Differential Computation Analysis
DFA	Differential Fault Analysis
DLP	Discrete Logarithm Problem
DPA	Differential Power Analysis
DRM	Digital Rights Management
ECDSA	Elliptic Curve Digital Signature Algorithm
FA	Fault Attack
gcd	Greatest common divisor
iO	indistinguishability Obfuscation
PRNG	Pseudo Random Number Generator
RNG	Random Number Generator
RNS	Residue Number System
SCA	Side-Channel Attack
SPA	Simple Power Analysis
SPN	Substitution Permutation Network
SVP	Shortest Vector Problem
TTP	Trusted Third Party
VBB	Virtual Black Box
VBF	Vectorial Boolean Function

List of Notations

Cor	Pearson's correlation coefficient
$\mathrm{HW}(x)$	Hamming weight of x
$\langle x, y \rangle$	Scalar product between x and y
$\mathrm{S}(x)$	Result of the application of the AES S-box on x
$\#A$	Cardinality of the set A
W_F	Walsh transform of a function F
$\mathcal{W}(F)$	Walsh spectrum of a function F
$\mathcal{W}_1(F)$	subset of $\mathcal{W}(F)$ restricted to inputs of Hamming weight 1
$r \xleftarrow{\$} S$	r takes a random value in the set S
$\mathcal{HG}$	Hypergeometric distribution
$a \wedge b$	Greater common divisor of a and b
$\bar{x}$	Binary complement of x

Chapter 1

Introduction

Contents

1.1 Cryptography and Cryptanalysis

The word *cryptography* is derived from *kryptos*, which means "hidden" or "secret" in ancient Greek, and *graphein*, which means "writing". The primary objective of cryptography is thus to protect the confidentiality of written messages but it can also be used to achieve other desirable security properties such as authenticity or integrity. While originally deployed for military and diplomatic purposes, it is now ubiquitous: banks, hospitals, and many businesses use it to protect their data. It is also present in our everyday lives, even if we are not always aware of it, when we make a payment by credit card for instance.

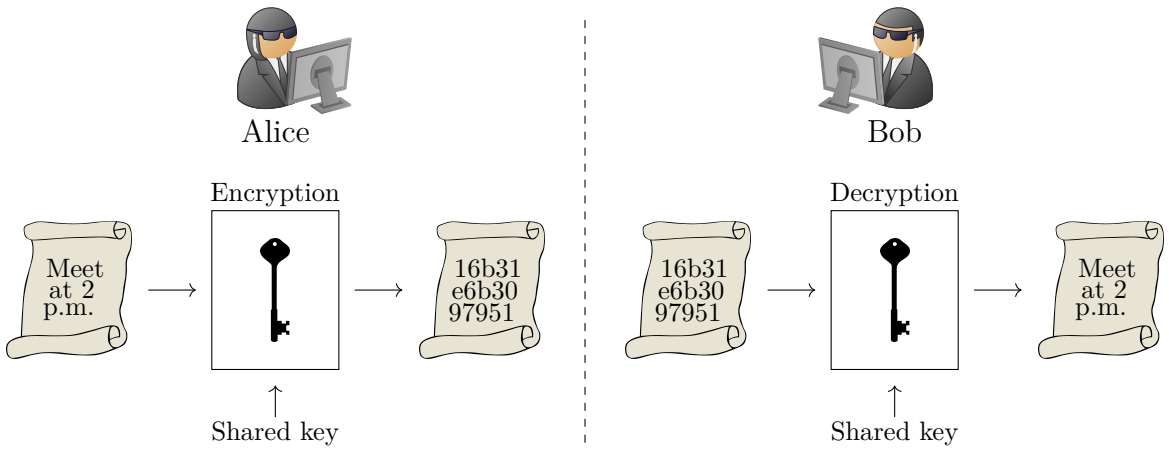


Figure 1.1: Symmetric encryption.

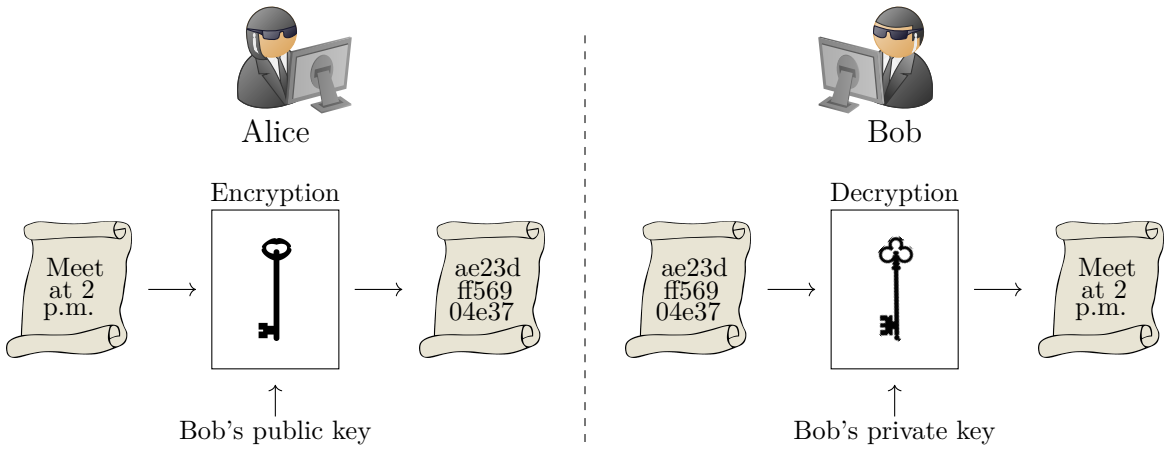


Figure 1.2: Asymmetric encryption.

The process of transforming a readable message, the *plaintext*, into a seemingly incomprehensible sequence of characters, the *ciphertext*, is called *encryption*. Of course, the reverse transformation, called *decryption*, should only be possible for the intended recipient of the message. It thus requires the knowledge of a secret, usually referred to as the *key*. In the case of *symmetric* cryptography (Figure 1.1), this secret is shared between the recipient and the sender of the message, and is used both in the encryption and decryption algorithms. Consequently, it must be securely exchanged between the communicating parties prior to communication. In contrary, in *asymmetric* cryptography (Figure 1.2), each party generates a pair of mathematically related keys: a public key and a private key. The first one is widely distributed and used for encryption, while the second one is kept secret and used for decryption. This eliminates the need for secure key distribution, making asymmetric encryption ideal for scenarios where key exchange is challenging or impractical. Additionally, asymmetric cryptography is useful to ensure the authenticity of messages via digital signatures, which are generated with a private key and can be verified with the corresponding public key. However, asymmetric schemes tends to be computationally more intensive compared to symmetric ones.

The counterpart of cryptography is called *cryptanalysis* and consists in the study of all the techniques allowing to bypass protections and recover the secret key, or at least compromise a security requirement such as confidentiality or authenticity. To effectively

prevent potential attacks, one must have a precise idea of what the adversary is capable of. Indeed, the capabilities of the attacker, and therefore the security measures needed, vary greatly depending on the context. Three main models are used to characterize the type of attacker considered: the black-box, grey-box, and white-box models.

1.2 Black-Box, Grey-Box, White-Box

Cryptographic algorithms are primarily designed to resist key recovery attacks in the *black-box model* where an attacker is restricted to the observation of their inputs and outputs. While the adversary knows which algorithm is employed, he cannot observe any intermediate variables. This model is sound when the cryptographic operations occur within secure environments and only the transmission channel cannot be trusted. However, in many real-life scenarios, the attacker may have access to more information. In particular, he may measure the execution timing of the algorithm, the power consumption of the device in which it is implemented or any other side-channel leakage [KJJ99]. Since these values are related to the data being processed, they could lead the attacker to the secret key if the implementation is not protected. Alternatively, the adversary may physically tamper with the execution of the algorithm to obtain sensitive information [BDL97]. As a consequence, developers have to put countermeasures in place to reach the originally expected security level. This setting, where the adversary has limited access to the device and may perform physical attacks is depicted by the *grey-box model*. It is particularly relevant in the context of embedded systems.

Finally, for about twenty years, we have been facing a new model, the *white-box model*, which considers an even more powerful attacker. Indeed, the recent development of cryptographic algorithms on open devices that do not necessarily contain secure hardware, such as smartphones or connected objects, provides numerous opportunities to the attacker. The *white-box model* assumes that the adversary controls every aspect of the implementation. The attack surface is very large: one can analyze the binary, tamper with the execution, or observe all the runtime information, such as the intermediate variables or the accessed memory addresses, for any chosen input. Of course, all the attacks that can be performed in the black-box or grey-box models are also potential threats. In such a hostile environment, the implementation itself is the last line of defense against key extraction. By misuse of language, an implementation of a cryptographic algorithm that is conceived to resist an attacker in the white-box model is commonly referred to as *a white-box*. It must be meticulously generated in what we call the *offline phase* so that secret values, such as keys, are hidden inside it and difficult to extract.

1.3 White-Box Use Cases

White-box implementations are currently employed in a variety of applications that run in insecure environments. In particular, mobile applications deployed across diverse devices cannot rely on secure hardware for storing secret keys and performing cryptographic operations, as many mobile phones lack a secure element. Typical use-cases include – but are not limited to – Digital Right Management (DRM) technologies, mobile payment or key-less applications and cryptocurrency wallets.

1.3.1 Digital Right Management

One of the first motivations for white-box cryptography is to be able to restrict the access and distribution of content under copyright. Let us take the example of a movie available on a streaming platform. Naturally, the platform does not want just anyone to be able to watch it without having paid for a subscription, so the movie is encrypted. To allow legitimate users access to the content, one might consider giving them the decryption key directly. However, in that case, nothing would prevent them from selling this key or sharing it with a friend. It is therefore preferable to provide legitimate users with a white-box program that allows them to decrypt the movie without knowing the secret key. Copying and reselling the code should then be more complicated than distributing a simple key. To protect against such *code-lifting* attacks, the implementation can for example be designed to work only on a specific device or to include hidden data that can identify the person who illegally resells it.

1.3.2 Mobile Payment

Mobile payment applications enable users to utilize their mobile phones instead of credit cards to make payments at NFC terminals [Pay]. During each payment, some transaction credentials, stored encrypted, need to be decrypted to generate a valid transaction request. Both the credential and the secret key used for decryption are sensitive information that need to be protected. Note that the context is a bit different than the one of DRM: the application owner has no interest in its secrets being disclosed to others. The goal of white-box cryptography is rather to protect against a malicious third party who could try to steal sensitive data, potentially through a malware installed on the phone. Once again, preventing key extraction is necessary but not sufficient. First, it must be impossible for the adversary to recover the sensitive data exchanged between the mobile and the terminal, so confidentiality is required. It implies security against key-extraction but is not restricted to it. Second, one should aim for integrity, meaning that the adversary should not be able to alter the data. Finally, code-lifting attacks should also be prevented since an adversary who has copied the code on his own device could use the latter to steal money from the white-box owner by making a payment on a terminal of his choice.

1.3.3 Cryptocurrency Wallets

Cryptocurrencies are digital currencies that operate independently of central authorities, such as governments or banks. They use cryptographic techniques to secure transactions and control the creation of new coins. Cryptocurrencies can be stored either on hardware-based or software-based wallets. While the use of USB sticks or smart cards gives less power to a potential attacker than relying on a mobile application, it can be inconvenient in practice. In the cases were pure-software cryptocurrency wallets are preferred, the use of white-box cryptography is essential. As for mobile payment applications, the adversary to consider is an external threat that goes against the interests of the white-box owner and has potentially installed a malware designed to steal information on his phone.

1.3.4 Digital Car Keys

Digital key solutions [Ide] enable car access and engine start via mobile devices. They allow users to easily share digital keys and restrict car usage when necessary, thereby

simplifying car sharing and rental services. Similar to mobile payment solutions, these applications must be protected against a white-box adversary who is not their rightful owner. The attacker seeks to extract the code or steal the secrets that are stored on the mobile phone and used for the authentication with the car in order to gain unauthorized access to the vehicle.

1.4 Theoretical Approach

For about twenty years, several authors (e.g. [SWP09, DLPR14, BI15, AABM20]) tried to define useful security notions for white-box cryptography. While resistance against key-extraction is straightforward, we have seen that it is not sufficient for many use-cases and that a white-box should also resist other threats, including code-lifting attacks. Theoretical works defining precisely the objectives of white-box cryptography and proving (im)possibility results are thus needed. In the following, we give a brief overview of the literature on that topic, starting with the results that are inherited from the field of program obfuscation.

1.4.1 Obfuscation and (Im)possibility Results

White-box cryptography is closely related to program obfuscation [HB15, BGI+01], their common goal being to protect software implementations. Informally, an obfuscator is a compiler that takes as input a program and produces a new one that has the same functionality but is in a sense "unintelligible". There are different ways to define more precisely that idea of unintelligibility, leading to different branches of obfuscation, the most famous ones being Virtual Black Box (VBB) and indistinguishability obfuscation (iO). Intuitively, a VBB obfuscator intends to produce a program that provides no information about the original one that cannot already be obtained from its inputs and outputs. An iO obfuscator, on the other hand, produces indistinguishable programs, meaning that given two programs P_1 and P_2 with the same functionality, and the obfuscation Q of one of them, it is impossible to determine whether Q was generated from P_1 or P_2.

Theoretical research on obfuscation was initiated by Barak et al. [BGI+01], who demonstrated the impossibility of creating a generic VBB obfuscator. While their work confirms the existence of programs that are inherently not VBB-obfuscatable, it does not imply that no algorithm can be implemented in a way that reveals no information beyond its inputs and outputs. Indeed, it has later been proven that some simple programs, such as point functions, are indeed VBB-obfuscatable [Wee05]. Even though it seems unlikely, it would thus be of high interest to determine if some cryptosystems are actually VBB-obfuscatable. Nevertheless, note that VBB-obfuscation is not exactly what we want to achieve in white-box cryptography. Indeed, if only the secret key has to be protected, one could accept giving some "useless" information to the attacker in addition to the inputs and outputs of the algorithm. Saxena et al. [SWP09] tried to formalize the distinction between "useful" and "useless" non-black-box information by adapting standard public-key security notions such as indistinguishability under chosen plaintext attacks (IND-CPA) in the context of white-box cryptography. They also managed to prove some (im)possibility results. In particular, they showed that for most programs, there are security notions that can be satisfied in the black-box model but not in the white-box context. On the positive side, they showed that there exists an implementation of an ad-hoc symmetric encryption scheme that satisfies IND-CPA. Saxena et al.'s work highlighted the lack of

theoretical foundations for white-box cryptography and motivated the search for useful security notions.

1.4.2 White-Box Security Notions

Since the work of Saxena et al. [SWP09], many different security notions have been defined in the literature for the white-box model. In the following, we briefly describe the most common ones and gives information on their usefulness in practice.

Resistance Against Key Extraction (or Unbreakability). This first security notion, formalized by Delerablée et al. in [DLPR14], captures the idea that an efficient adversary should not be able to recover the secret key embedded in a white-box implementation. While obviously necessary, unbreakability is often not sufficient.

One-Wayness. The idea behind this security notion is that a white-box implementation of an encryption algorithm should not facilitate the decryption of an arbitrary ciphertext. An interesting fact about this property is that it enables the creation of an asymmetric scheme from a symmetric one. The white-box implementation is then seen as the public key for encryption (resp. verification) while the secret key enables the other party to decrypt (resp. sign) messages.

Incompressibility. An implementation is called incompressible when it is difficult to reduce its size without modifying its functionality. The motivation behind the definition of such a security notion is that a large white-box implementation should be more difficult to distribute, in particular online, than a short secret key. This property is therefore intended to mitigate code-lifting attacks. While incompressibility might be thinkable for use-cases such as DRM, it is not suited for mobile payment or any application that runs on resource-constrained devices. Furthermore, note that it does not prevent code-lifting attacks but rather minimizes their impact.

Space-Hardness. This notion introduced in [BI15] is very close to incompressibility. An implementation is (M, Z)-space hard if an adversary having lifted a portion M of the code has a probability lower than 2^{-Z} of correctly encrypting/decrypting a random input. The idea is again to complicate the work of an adversary trying to steal and sell the code.

Traceability. Another approach to mitigate code-lifting attacks involves watermarking the implementation so that if unauthorized copies are distributed, the original program owner can be identified. Again, this property may be useful in the context of DRM, where the adversary may try to sell copies of his own decryption program. Nevertheless, it is pointless for mobile payment applications, where the white-box has to be protected against external adversaries trying to steal the legitimate owner of the program and not against the latter trying to sell copies of its own implementation.

Hardware-Binding. In [AABM20], Alpirez Bock et al. pointed out the lack of security notions addressing code-lifting attacks that are interesting for mobile payment applications. In that context, they suggest to rely on hardware-binding, which involves making the implementation functional only when executed on a specific hardware device. This

can be achieved, for instance, by incorporating a unique device fingerprint within the white-box.

Application-Binding. A last possibility to mitigate code-lifting attacks consists in making the white-box inseparable from a particular application instead of a specific hardware device. This idea presents several advantages. First, it can enhance security, especially if the application incorporates authentication mechanisms. Second, a white-box solely bound to an application, without any hardware-binding features, allows the program owner to select the device on which he wishes to use it. However, as noted by Alpirez Bock et al. in [AABM20], application-binding is difficult to formalize properly.

1.5 Practical Approach

Since the first white-box implementation introduced in 2002 by Chow et al. [CEJv02], many designs and attacks were proposed. In the following, we give an overview of the history of white-box cryptography.

1.5.1 White-Box Implementations of Block Ciphers

In 2002 and 2003, Chow et al. proposed two white-box designs intended to prevent key extraction attacks in DRM applications. These implementations, of the Data Encryption Standard (DES) [CEJv02] and the Advanced Encryption Standard (AES) [CEJv03] respectively, initiated a long line of study on white-boxed block ciphers. In short, Chow et al.'s idea consists in protecting the secret key by embedding it inside precomputed look-up tables that are used during the execution to perform the sensitive operations. All the intermediate variables, the inputs and outputs of the tables, are then obfuscated by random bijections called *encodings*. The latter can be classified into two categories, namely the *internal encodings*, that cancel each other, and the *external encodings* that are the ones applied to the input and output of the cryptosystem. While they increase the overall security, external encodings modify the input/output behavior of the program, which is prohibitive for many use-cases where standard cryptosystems are required for interoperability reasons.

The first two implementations of Chow et al. were proven insecure a few years after their publication [BGEC04, WMGP07, GMQ07]. They were quickly followed by many new designs [BCD06, XL09, Kar11, LLY14, BCH16] and attacks [DWP10, DRP13a, DRP13b, MGH09, DFLM18, BHMT16], most of them concerning AES. Unfortunately, there is no published implementation of either DES or AES that stands unbroken to this day. Even the existence of a secure implementation is currently unknown.

In this context, the WhibOx contests of 2017 [Whi17] and 2019 [Whi19] were organized as part of the CHES workshops to motivate practical research on AES white-boxes. Each time, developers were invited to submit the source code of a (secretly designed) implementation and attackers to try and break it. The first contest ended with a victory for the attackers, with all the challenges being broken at the end of the allocated time. On the contrary, three implementations – all due to Biryukov and Udovenko – survived the 2019 WhibOx contest, which illustrates the advancement of the underlying white-box techniques. Note however that they were all broken in the months following the competition.

1.5.2 Study of Other Standards

Apart from AES and DES, there is relatively little literature on white-box designs for known crypstosystems. In 2022, Galissant and Goubin [GG22] took interest in multivariate cryptography and proposed an implementation of HFE [Pat96] that has a claimed level of security of 2^{80} but is unfortunately impractical since it requires about 256GB of memory. In the meantime, Ranea, Vandersmissen and Preneel [VRP22, RVP22] proposed two solutions for the protection of ARX ciphers in the white-box model. Unfortunately, the first one was broken by the authors themselves and the second one by Biryukov, Lambin and Udovenko in [BLU23].

Recently, the ECDSA signature scheme benefited from quite a lot of attention. Indeed, there is a strong need for ECDSA white-box solutions for the industry and securing this scheme poses several new challenges [DGH21]. In particular, encoded look-up tables are not well suited for operations on a large number of bits such as modular inversions or scalar multiplications, so new countermeasures need to be invented. Furthermore, the only design published in the literature [ZBJ20] relies on a cloud server during the computation of the signatures, which is prohibitive for many use-cases, and has been proven insecure [GH23]. In this context, the 2021 [Whi21] and 2024 [Whi24] editions of the WhibOx contest switched the target from AES to ECDSA. The principle was similar to previous editions: developers were invited to submit candidate implementations and attackers to try to recover the corresponding secret keys. In 2021, all 97 challenges ended up broken in less than 35 hours, yielding a clear victory for the attackers and showing the difficulty of protecting ECDSA in the white-box context. This edition led to the publication of two papers [BBD+22, BDG+22] discussing the attacks and countermeasures that were used. As for the one of 2024, it is still ongoing, but less than a month before the submission deadline, the best candidate also survived approximately 35 hours.

1.5.3 Ad-Hoc Cryptosystems

To this day, no practical public implementation of a standard algorithm has proven to be resistant against key extraction. However, there are several ad-hoc cryptosystems in the literature [BI15, FKKM16, BIT16, CCD+17, Bar20, KI21, LRH+22] that were specifically tailored for the white-box context and some satisfy certain security notions presented in Section 1.4.2. A large majority of these schemes focus on protection against key extraction and code-lifting via incompressibility or space-hardness.

1.5.4 White-Box Cryptography in Practice

For reasons of interoperability and because multiple stakeholders are typically involved in a specific solution, the industry is unlikely to adopt ad-hoc cryptosystems for many use cases. Nevertheless, as we already mentioned, all public designs of standard algorithms are currently considered broken. In such circumstances, the industry is compelled to devise custom solutions to address the escalating demand for secure cryptographic software. Their first objective is to provide practical security by shielding the white-box against automated attacks, such as grey-box inherited ones. Obfuscation is then used to force the adversary to go through a time-consuming and arduous reverse-engineering phase before being able to search for vulnerabilities. Note that here, we do not talk about program obfuscation (as described in Section 1.4.1) since it is unfortunately unpractical to this day, but rather about code obfuscation that consists in transforming the code so that it

remains functionally equivalent but becomes significantly more complex. Common techniques include control flow obfuscation, which makes the execution path complex and hard to follow, and instruction substitution, which replaces straightforward instructions with more convoluted alternatives. The secrecy surrounding the design of such obfuscated implementations enhances their security, despite going against Kerckhoffs' principle [Ker83], a fundamental concept in cryptography which asserts that everything about the cryptosystem, except the key, must be considered publicly known. Furthermore, the obfuscation layer might be different for each white-box, so that at least a part of the reverse-engineering phase must be performed for each distinct target, complicating mass attacks.

Even if these practical solutions might be less secure than what was initially expected, it is important to note that the security model of white-box cryptography is too strong for many use-cases, and real-life adversaries may be far less powerful than anticipated. Indeed, the capabilities and objectives of attackers vary widely depending on the use-case. For example, in mobile payment applications, the adversary is typically an external threat who may introduce a virus but is unlikely to possess omnipotent capabilities. If the white-box is bound to an application implementing authentication mechanisms, the attacker should be unable to execute the code himself, making chosen plaintext attacks impractical and limiting the number of side-channel traces obtainable within a given time-frame. If the adversary were to induce a high number of faults, the legitimate user of the application might detect that something is amiss. Moreover, for some use-cases, the number of uses and lifespan of the key can be restricted, and the white-box can be regularly updated to further reduce the number of executions observable by the attacker. Finally, in addition to obfuscation techniques, other software protection mechanisms such as anti-debug can also be implemented.

Practical white-box designs should thus be constructed for a specific purpose and come with a detailed adversary model so that relevant analysis can be done. Similarly, when a new attack is discovered, its practicability should be evaluated in terms of several parameters, including the possibility of automation, the required number of executions and the need to choose specific inputs or to induce faults.

1.6 Contributions and organization of the manuscript

This PhD thesis gathers the results of the four articles that I published in conferences. They are listed in this section together with my other contributions, including the redaction of a popular science article on white-box cryptography for the *Interstices* journal and a patent on a countermeasure against fault attacks for ECDSA white-boxes.

1.6.1 Publications

In International Conferences/Journals

[**DGH21**] Emmanuelle Dottax, Christophe Giraud, and Agathe Houzelot. White-box ECDSA: Challenges and Existing Solutions. In Shivam Bhasin and Fabrizio De Santis, editors, *COSADE 2021*, volume 12910 of *LNCS*, pages 184–201. Springer, Heidelberg, October 2021.

[**BBD+22**] Guillaume Barbu, Ward Beullens, Emmanuelle Dottax, Christophe Giraud, Agathe Houzelot, Chaoyun Li, Mohammad Mahzoun, Adrián Ranea, and Jianrui

Xie. ECDSA White-Box Implementations: Attacks and Designs from CHES 2021 Challenge. *IACR TCHES*, 2022(4):527–552, 2022.

[**GH23**] Christophe Giraud and Agathe Houzelot. Fault Attacks on a Cloud-Assisted ECDSA White-Box Based on the Residue Number System. In *2023 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, pages 72–80. IEEE, 2023.

[**CH24**] Laurent Castelnovi and Agathe Houzelot. On the (Im)possibility of Preventing Differential Computation Analysis with Internal Encodings. Accepted for publication in *TCHES*, 2024(3), 2024.

In Popular Science Reviews

[**CH23**] Arnaud Casteigts and Agathe Houzelot. Vers une cryptographie en boîte blanche ? In *Interstices*, INRIA, 2023. `https://interstices.info/vers-une-cryptographie-en-boite-blanche`.

1.6.2 Patents

During my PhD, I contributed in the design of a proprietary ECDSA white-box for IDEMIA. I also implemented the proof of concept in python and C. Our work on ECDSA white-boxes lead to the following patent:

[**DGH22**] Emmanuelle Dottax, Christophe Giraud and Agathe Houzelot. Procédé de signature cryptographique d'une donnée, dispositif électronique et programme d'ordinateur associés. Patent FR3133251A1, 2022.

1.6.3 Thesis Outline

This thesis contributes to the field of practical white-box cryptography. We focused on two widely-used standard cryptosystems, namely AES and ECDSA, trying to identify attacks that could be performed by real-life adversaries and determine countermeasures that could prevent them. The chapters are organized as follows:

Chapter 2: Grey-box Attacks and Countermeasures

While initially developed for the grey-box model, side-channel and fault attacks have shown to be extremely effective in the white-box model, where the adversary has greater capabilities. Given their significance in the subsequent parts of this thesis, this chapter delves deeper into these attacks, discussing common countermeasures and highlighting the differences in their implementation between the grey-box and white-box models.

Chapter 3: The Advanced Encryption Standard in the White-Box Model

Although all public white-box implementations of AES have been compromised, some of the attacks are arguably difficult to execute in practice. Similarly, there are common countermeasures in the literature, as external encodings, that are unlikely to be adopted by the industry because they alter the input/output behavior of the algorithm. It is therefore crucial to study attacks that are feasible in real-life scenarios and to develop practical countermeasures. In particular, finding a method to prevent side-channel attacks without

relying on random values would be a significant breakthrough in white-box cryptography. Internal encodings were once considered a potential solution, but Rivain and Wang demonstrated in [RW19] that encoded AES implementations could be effectively broken using Differential Computation Analysis (DCA) [BHMT16] with high probability.

In this chapter, we begin by reviewing the state-of-the-art on AES white-box implementations. We start with the designs that have been published in the literature and then provide a brief overview of the various algebraic attacks. We then introduce a new collision attack on practical AES implementations that requires very few executions. Next, we delve into side-channel analysis, discussing previous works before presenting the main contribution of this chapter: we identify the small class of encodings that cause DCA to fail according to [RW19] and explain how to adapt the attack to defeat even these specific bijections. We thereby demonstrate that no class of encodings can effectively protect an AES white-box against side-channel attacks. Crafting bijections with specific properties, rather than selecting them at random, is not a viable solution.

Chapter 4: The Elliptic Curve Digital Signature Algorithm in the White-Box Model

Despite the limited academic literature and the fact that techniques used in symmetric white-box cryptography cannot be directly applied to asymmetric algorithms, many ECDSA white-boxes [SAS, Tha, Qua, Ird, Int, Inc, Ver, Dig] have been developed and sold by companies. Understanding the challenges in designing such a white-box and studying practical attacks and countermeasures is therefore essential.

This chapter begins with a discussion on the WhibOx 2021 contest. As participants on the attacking team TheRealIdefix, we successfully broke the most submissions and ranked third at the end of the contest. We detail the various attacks we employed, ranging from the exploitation of biases in the nonce generation to fault attacks, and assess their effectiveness against all the candidates. We also give an overview of the two best designs that, although broken, contain new ideas that are theoretically interesting.

Next, we explore the most interesting countermeasures that we found in the various patents filled by companies. Since their products' documentation lacks technical details, we examined patents to glean insights into the white-box designs. While they may omit critical details and not fully explain the security context, we believe that patents still provide valuable information on methods implemented in the field. As already suggested by the results of the WhibOx contest, this study shows that known countermeasures seem to be insufficient to prevent all the attacks that threaten ECDSA white-boxes, especially the ones based on fault injections.

Finally, we analyze the security of the first published ECDSA white-box, that is Zhou et al.'s cloud-based implementation [ZBJ20]. We present two efficient fault attacks based on a common principle to re-inject some specific values from one signature execution to another. We also propose a countermeasure that prevents these attacks without increasing the size of the white-box on the clients' side. It only has a slight impact on the performances on the server's side.

Chapter 2

Grey-Box Attacks and Countermeasures

Contents

The grey-box model considers an adversary who has access to a physical device implementing a cryptosystem and may exploit information leaked during an execution, such as the power consumption, the electromagnetic radiations or the execution timing. Since this side-channel information directly depends on the operations performed and the processed data, it may be used to extract sensitive values, including cryptographic keys, from unprotected implementations.

By manipulating the voltage, clock frequency, or other parameters of the device, attackers can also intentionally introduce errors in the computation or disrupt the normal execution flow. The obtained faulty output may then be used to obtain information on the secret key if the implementation lacks adequate protection.

While initially introduced for the grey-box model, side-channel and fault attacks have logically proven to be highly effective in the white-box model where the adversary is more powerful [BHMT16]. This chapter provides more details on these attacks, including information on typical countermeasures and the differences in their application between the two models.

2.1 Side-Channel Attacks

Side-channel attacks (SCA) were first introduced in 1996 by Paul Kocher [Koc96], who proposed to use differences in the execution timing to break unprotected implementations

of several cryptosystems such as Diffie-Hellman and RSA. Since then, a variety of attacks exploiting different side-channels, such as power consumption [KJJ99] or electromagnetic emanations [GMO01] have been described. They proved to be highly effective in practice against a wide variety of cryptosystems.

Side-channel attacks all start with the acquisition of a set of traces during one or several executions. These can consist of power consumption or any other side-channel information over time. Then, the analysis of these traces can take various forms. Side-channel attacks can be categorized into two groups based on the type of information exploited: some analyze the operation flow, while others concentrate on the processed data. They can also be classified according to whether they involve a *profiling phase* or not. In this section, we give the basic principle of those attacks and discuss their countermeasures.

2.1.1 Analyzing the Execution Flow

The success of some side-channel attacks, such as timing attacks [Koc96] or Simple Power Analysis (SPA) [KJJ99] hinges on the dependence of the execution flow on a sensitive variable. Different operations may take different amounts of time and generate distinct power consumption or electromagnetic radiation patterns. Consequently, an attacker who can observe these variations may gain insights into the secret key.

Figure 2.1 shows an example of SPA targeting the exponent of a modular exponentiation implemented using the famous *square-and-multiply* algorithm. In this scenario, variations in power consumption enable the attacker to discern whether a multiplication operation occurred during each iteration of the loop, thereby allowing the recovery of the exponent with a single power trace.

Algorithm 1: Square-and-multiply

Input: x, $e = (e_0, e_1, \ldots, e_\ell)_2$ and n
Output: $y = x^e \bmod n$
$y \leftarrow 1$
for i *in* $[\![0, \ell]\!]$ **:**
    $y \leftarrow y^2 \bmod n$
    if $e_i = 1$ **:**
        $y \leftarrow y \cdot x \bmod n$
return y

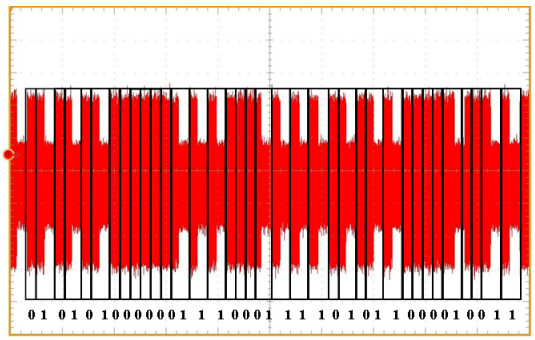


Figure 2.1: Left: the square-and-multiply algorithm for modular exponentiation. Right: a power trace acquired during an execution of the algorithm, taken from [Riv09]. Since square and multiply operations can easily be distinguished, the exponent is leaked as shown under the curve.

Timing attacks are more sophisticated. For example, with the square-and-multiply algorithm, an exponent with more 1's in its binary representation results in more operations and thus longer execution timing. An attacker can thus infer sensitive information by observing multiple executions and applying statistical analysis to the collected data.

2.1.2 Analyzing the Processed Data

Some side-channel attacks exploit the disparity in power consumption or electromagnetic emanations when manipulating a bit set to 0 or 1. Despite inherent noise, the intermediate variables being processed at any time during an execution are somehow correlated to the side-channel information, and this can be exploited to recover secret values if the implementation lacks adequate protection. The attacks all consist of the following steps:

1. Acquire a set of traces by observing multiple executions on different inputs.
2. Select an intermediate variable that depends on a few key bits only. Such a variable is said to be *sensitive*.
3. For all key guess, predict the said intermediate variable.
4. Choose a score function, called a *distinguisher*, and apply it to the predictions and each point of the traces.
5. Validate the key hypothesis that obtains the highest score.
6. Repeat with another sensitive variable depending on other key bits until the whole secret is recovered.

In the seminal paper of Kocher et al. [KJJ99], the proposed distinguisher was the *difference-of-means*. It consists in dividing the collected traces into two sets based on the predicted sensitive variable, and computing the mean of each set for each point. For a wrong key guess and a correctly chosen sensitive variable, the sorting can be modeled as random so the difference of means should approach 0 when the number of traces is high enough. The correct key hypothesis can then be determined as the one that leads to the highest absolute value of the difference-of-means (see Figure 2.2).

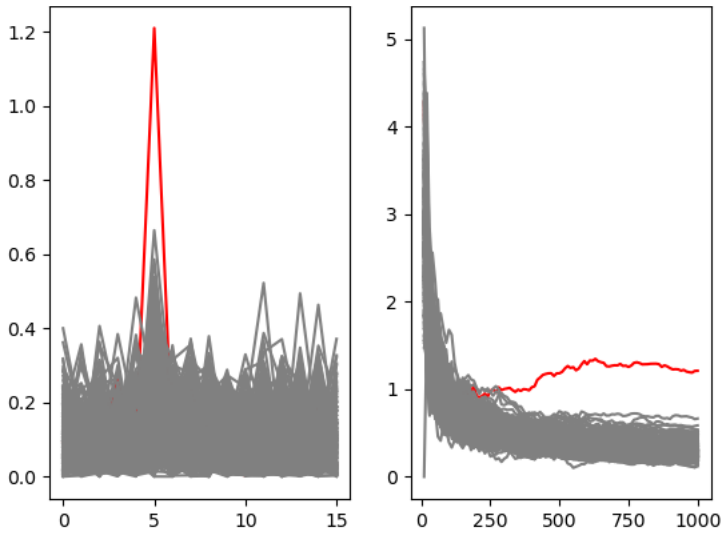


Figure 2.2: DPA on simulated AES traces. Left: difference-of-means over time for wrong (resp. good) key guesses in grey (resp. red). Right: highest absolute value of the difference-of-means for each key guess (good hypothesis highlighted in red) over the number of leakage measurements.

Since the introduction of DPA, several other statistical tools have been suggested, including Pearson correlation coefficient [BCO04], mutual information [GBTP08] and maximum likelihood [CRR03]. The optimal choice of distinguisher largely depends on the attacker's knowledge of the leakage. If the latter is non-existent, mutual information might be the most effective option. Conversely, likelihood-based attacks require a high degree of knowledge.

2.1.3 Profiled Analysis

Unlike other SCA, profiled attacks require full control over a device which has to be similar to the target one. It is used in what is called the profiling phase, where a large number of traces are recorded for different known key values. These traces are employed to either construct templates [CRR03] or train a deep-learning network [HGD+11] to learn a leakage profile of the cryptosystem under attack. Subsequently, in the key extraction stage, a small number of traces collected from the target device are used to find the correct key based on the template or trained model. Note that even though profiled attacks are considered very powerful, their application is sometimes complicated in practice since the attacker needs to have a similar device that he can configure with different known key values.

2.1.4 Countermeasures

A first idea to complicate side-channel attacks is to use hardware modules to flatten the power consumption or increase the noise, but this is often insufficient to thwart practical attacks. Preventing SPA or similar threats is relatively straightforward: one must ensure that the execution flow remains independent of any sensitive data by employing *atomic* or *regular* algorithms [CCJ03, Joy07, JT09, Riv11]. However, DPA-like attacks are more challenging to mitigate. Since they typically require the analysis of many side-channel traces, a first strategy could be to frequently update the secret key if the use-case allows it. This can be done in combination with other countermeasures, such as masking or shuffling, which help remove or at least reduce the dependence between trace points and sensitive variables.

Masking

Masking is a widely-deployed countermeasure that has been proven efficient against side-channel attacks [CJRR99, PR13]. It consists in splitting all the sensitive variables into several shares and processing them individually. More precisely, a sensitive variable x is represented as

$$x = x_0 \star x_1 \star x_2 \star \ldots \star x_n \tag{2.1}$$

where $\star$ is a group operation and n is called the masking order. The masks x_i are drawn at random for $1 \leq i \leq n$ and the masked variable x_0 is computed so that (2.1) is satisfied. The two most predominant types of masking are *Boolean masking* and *arithmetic masking* for which $\star$ is respectively the XOR operation and the modular addition. The choice of the masking type depends on the operations to be protected, and if needed, there exists conversion methods from Boolean to arithmetic masking and the other way around [Gou01, CT03, Deb12, BCZ18]. In any case, the computations must be performed in a way that ensures that masks and masked variables are processed separately. Several

methods, called *masking schemes*, have been described in the literature (e.g. [ISW03, GPQ11, CGP+12]).

Assuming that the masks are uniformly distributed, recovering up to n shares does not give any information on the sensitive variable. Since the shares are manipulated separately, the processed data is statistically independent from any secret value. Traditional SCA that rely on the leakage from a single intermediate variable are thus no longer applicable. However, note that masking remains susceptible to *higher-order* side-channel attacks, which exploit leakages from multiple intermediate variables simultaneously by combining samples within a trace [Mes00, OMHT06, CPR07]. A masking scheme of order n is theoretically vulnerable to an attack of order $n+1$. Nevertheless, because of the noise, the complexity of higher-order SCA increases exponentially with the order [CJRR99], so masking remains a sound countermeasure.

Shuffling

In order to increase the noise on the traces and complicate the attacks, one can implement shuffling techniques [VMKS12], which involve randomly permuting the order of independent operations at each execution of the program. Since the obtained traces are misaligned, one has to observe a higher number of executions in order to compensate the noise. Dummy operations or random delays can also be added for better results. In combination with a masking scheme, shuffling can drastically increase the complexity of higher-order side-channel attacks.

2.1.5 Particularities in the White-Box Model

The key distinction between side-channel attacks conducted in the grey-box versus the white-box model lies in the nature of the traces. Indeed, in the white-box context, the adversary has access to the memory, so he can trace the addresses that are accessed during the execution or even the intermediate variables themselves. This way, he may obtain noise-free traces, significantly enhancing the efficiency of the attacks.

Furthermore, the countermeasures that are usually deployed in the grey-box context are not always straightforward to implement in the white-box model. Indeed, both masking and shuffling rely on the generation of random values, which is typically done by a Random Number Generator (RNG). However, the output of such an external source of randomness could easily be identified and fixed by a white-box adversary who controls the execution environment. In this context, random variables are therefore usually computed with a Pseudo-Random Number Generator (PRNG) applied to the input of the program. Nevertheless, the implementation of such a PRNG can be challenging. Indeed, the adversary should neither be able to find nor fix its output. Moreover, this value should be unpredictable to the adversary. While using a keyed PRNG might seem like a good idea, such a solution creates a vicious circle as the generator then has to be resilient against side-channel attacks. The latter are thus both more efficient and more difficult to prevent in the white-box model.

2.2 Fault Attacks

The idea of deliberately inducing errors in computations was introduced in 1996 by Boneh et al. [BDL97]. There are many ways of perturbing a component to perform a fault attack,

but the most common ones consist in inducing a *glitch* [BECN+04] (i.e. temporarily modifying the power supply voltage or the clock frequency) or exposing the component to a brief and intensive pulse of light [SA03, BECN+04]. The effect of a fault can differ depending on the way it was induced, but also on the timing of the attack or on the area of the chip targeted in the case of a laser-based fault injection. It may for instance alter the value of an intermediate variable, skip an instruction or perform a jump in the code. Some fault attacks are highly generic, but most rely on a precise fault model that specifies which data has been corrupted and how the fault impacts it. Common models consider the modification of a bit, a byte or a data word, to either a random value or a constant.

The treatment of the faulty outputs also varies widely depending on the targeted scheme. Some attacks, as for instance the Bellcore attack against RSA [BDL97] or the Differential Fault Analysis (DFA) against block ciphers [BS97, PQ03], analyze the difference between correct and faulty outputs in order to retrieve information on the secret key. Others simply exploit the fact that the injected fault yields, or not, an erroneous result [Cla07, DEK+18, YJ00]. Advantageously, such attacks may bypass some classical countermeasures based on fault detection.

2.2.1 Countermeasures

First of all, fault attacks can be mitigated by hardware countermeasures, such as physical sensors included in the chips to detect light or significant voltage and frequency variations. Of course, there also exist software countermeasures. While they may vary depending on the targeted scheme and the employed attack, most of them rely on redundancy. They can be classified into two categories: detective and infective solutions.

Detective countermeasures rely on consistency checks. For example, performing a sensitive operation twice and comparing the results can be effective for a large variety of attacks and cryptosystems. Another method involves implementing the inverse transformation and applying it to the output of the algorithm, then verifying that it returns to the original input. If a fault is detected, an error or an unusable output, such as a random value independent of the computation, should be returned. This type of countermeasure is straightforward to implement, but it doubles the execution time, which may be prohibitive for some use cases. Furthermore, consistency checks can potentially be bypassed with another fault, even though the complexity of executing such a double-injection attack is already significantly higher.

Instead of checking the consistency of the computations, one can implement the algorithm in such a way that any fault on a sensitive variable also corrupts other ones, resulting in an output that is not exploitable by an attacker. Infective countermeasures [YKLM02, GST12, LRT12] are less susceptible to double injections compared to detective ones, but they are challenging to design and to generalize to a large variety of cryptosystems. It is essential to ensure that compromised outputs do not disclose any sensitive information or some attacks may still succeed [YKM06, LRT12, BG13].

Note that some countermeasures employed against SCA (see Section 2.1.4) can also be used to mitigate several fault attacks. For instance, unlike detection-based countermeasures, masking or other randomization techniques can be used to prevent some attacks that exploit whether the fault injection produced an error or not. Shuffling, on the other hand, can complicate the attacks that require precise fault injections.

2.2.2 Particularities in the White-Box Model

Fault attacks are far easier to mount in the white-box context. Indeed, since the adversary controls the execution environment, there is no need to use lasers or glitches to induce errors. Instead, he can directly modify the executable or even use debugging tools to stop the execution and, for example, skip an instruction or alter a specific register's value. Very precise faults are thus made possible, offering new possibilities for attackers. In particular, it is easier to induce several faults and circumvent consistency checks. It is thus often preferable to use infective countermeasures that propagate the error until the result is unusable rather than relying on redundancy and fault detection.

Chapter 3

The Advanced Encryption Standard in the White-Box Model.

Contents

3.1 Introduction

Selected by the NIST in 2001 after a rigorous evaluation process, the Advanced Encryption Standard (AES) [AES23] has since become one of the most popular symmetric ciphers.

It is indeed known for its efficiency in both software and hardware, straightforward implementation, and minimal memory requirements.

AES is also by far the most studied cryptosystem in the white-box model. This study was initiated by Chow et al., who introduced the first implementation in 2002 [CEJv03]. Their main idea was to implement all the operations with the help of look-up tables protected by random bijections, called *encodings*, applied to their inputs and outputs. Since Chow et al.'s pioneering work, numerous attacks [BGEC04, DWP10, DRP13a, LRD+14, BHMT16] and new designs [BCD06, XL09, Kar11] have emerged. While all of the public implementations are currently broken, some of the attacks can be argued to be difficult to execute in practice. Similarly, some of the countermeasures proposed are unlikely to be used by the industry since they modify the input/output behavior of the algorithm, therefore making it not standard anymore. It is thus of high interest to study more deeply the attacks that are easy to perform in real-life scenarios and the countermeasures that could be deployed in practice. In particular, finding a way to prevent side-channel attacks without relying on a PRNG, unlike masking and shuffling, would be a significant advancement in white-box cryptography. Encodings were once believed to be able to fulfill this role but Rivain and Wang showed in [RW19] that encoded AES implementations could actually be broken by a Differential Computation Analysis (DCA) [BHMT16] with high probability.

In this chapter, we first go through the state-of-the-art on AES white-boxes. We start with the designs that were published in the literature, and continue with a short description of the diverse algebraic attacks. We take this opportunity to present a new collision attack on practical AES implementations that requires a very low number of executions. We then move to side-channel analysis and discuss previous works before arriving to the main contribution of this chapter: we characterize the small class of encodings that prevents DCA according to [RW19] and explain how to adapt the attack so that even these specific bijections can be defeated. We thus demonstrate that no class of encodings can efficiently protect an AES white-box against side-channel attacks.

3.2 Preliminaries

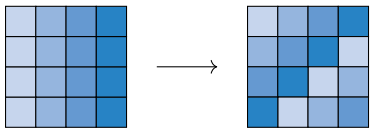
3.2.1 Description of AES

The Advanced Encryption Standard [AES23] is a Substitution Permutation Network (SPN) that works on 128-bit blocks. Three different sets of parameters corresponding to key lengths of 128, 192 or 256 bits are supported, but in this work as in many others, we focus on AES-128 which uses a 128-bit key. The encryption algorithm then consists of 10 rounds during which the four following operations are successively applied on a 4×4 array of bytes denoted as the *state*.

1. AddRoundKey: Binary addition between the state and a 128-bit roundkey k^r derived from the master key through a key-scheduling.

2. SubBytes: Parallel application of a non-linear bijection, the AES S-box S, to each byte of the state.

3. ShiftRows: Permutation of the indexes of the state bytes. For $i \in [\![0, 3]\!]$, the i^{th} row is shifted by i positions to the left in the 4×4 array.



4. MixColumns: Multiplication of each column of the state, seen as a 4-byte vector with elements in $\mathbb{F}_{2^8}$, by a constant matrix

$$MC = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} .$$

Note that the MixColumns operation of the last round is replaced by an exclusive-or with a post-whitening key. The AES-128 encryption can thus be described as in Algorithm 2.

Algorithm 2: AES-128 encryption

Input: Plaintext m and roundkeys $k^1, k^2, \ldots, k^{11}$
Output: Ciphertext c
state $\leftarrow m$
for r *in* $[\![1, 10]\!]$ **:**
    AddRoundKey(state, k^r)
    SubBytes(state)
    ShiftRows(state)
    if $r \neq 10$ **:**
        MixColumns(state)
AddRoundKey(state, k^{11})
$c \leftarrow$ state
return c

As for the decryption algorithm, it simply applies the inverse of the operations in a reversed order (see Algorithm 3).

Algorithm 3: AES-128 decryption

Input: Ciphertext c and roundkeys $k^1, k^2, \ldots, k^{11}$
Output: Plaintext m
state $\leftarrow c$
AddRoundKey(state, k^{11})
for r *in* $[\![1, 10]\!]$ **:**
    if $r \neq 1$ **:**
        MixColumnsInv(state)
    ShiftRowsInv(state)
    SubBytesInv(state)
    AddRoundKey(state, k^{11-r})
$m \leftarrow$ state
return m

3.2.2 Measuring the Dependence between Random Variables

Several tools can be used to measure the dependence between random variables. In this chapter, we will use two of them, namely Pearson's correlation coefficient and mutual information.

Pearson's Coefficient. this coefficient can be used to measure the linear correlation between two random variables. In particular, as we will see in Section 3.5.1, it is used in some side-channel attacks [BCO04, BHMT16] to assess the correlation between trace points and predicted values.

Given two random variables X and Y, Pearson's correlation coefficient is defined as

$$\mathrm{Cor}(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y} \ ,$$

with $\mathrm{Cov}(X,Y)$ denoting the covariance between X and Y and σ_X (resp. σ_Y) being the standard deviation of X (resp. Y).

Mutual Information. The mutual information of two random variables X and Y allows to measure their dependence, that is to say the quantity of information that one obtains on X by observing Y, or the opposite. The advantage compared to Pearson's coefficient is that it gives information about all dependences, whether linear or not. As we will see in Section 3.5.1, mutual information is also used in some side-channel attacks [GBTP08] to assess the dependence between trace points and predicted values.

Given two variables $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$, their mutual information is defined as

$$\mathrm{I}(X,Y) = -\sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \mathbb{P}(X=x, Y=y) \cdot \log_2\left(\frac{\mathbb{P}(X=x, Y=y)}{\mathbb{P}(X=x) \cdot \mathbb{P}(Y=y)}\right) \ .$$

3.2.3 The Hypergeometric Distribution

The hypergeometric distribution $\mathcal{HG}(\alpha, \beta, \tau)$ is a discrete distribution that describes the probability of having t successes in τ draws without replacement from a population of size β containing α successes, that is

$$\mathbb{P}_{\alpha,\beta,\tau}(t) = \frac{\binom{\alpha}{t}\binom{\beta-\alpha}{\tau-t}}{\binom{\beta}{\tau}} \ .$$

In the following, we denote by $\widetilde{\mathcal{HG}}(n)$ the distribution $\mathcal{HG}(2^{n-1}, 2^n, 2^{n-1})$.

3.2.4 Boolean Functions

First, let us recall a few definitions. Let $n, m \geq 1$ be two integers. A function f from $\mathbb{F}_2^n$ to $\mathbb{F}_2$ is called a *Boolean function* while a function F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ is called a *vectorial Boolean function (VBF)*. If $F(x) = (F_0(x), F_1(x), \ldots, F_{m-1}(x))$, the Boolean functions $F_0, F_1, \ldots, F_{m-1}$ are often referred to as *coordinate functions* of F. The function F is said to be *balanced* if, for all $y \in \mathbb{F}_2^m$, $\#\{x \in \mathbb{F}_2^n \mid F(x) = y\} = 2^{n-m}$. The bias (or imbalance) of a Boolean function f is defined as $B(f) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)}$.

In the rest of this chapter, we will use the notions of Walsh transform and Walsh spectrum of Boolean and vectorial Boolean functions. We thus recall the following definitions.

Definition 1 (**Walsh transform, Walsh spectrum**).

- *The Walsh transform of a Boolean function* $f : \mathbb{F}_2^n \to \mathbb{F}_2$ *is defined as:*

$$W_f : \mathbb{F}_2^n \to \mathbb{Z}$$
$$u \mapsto \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)+\langle u,x \rangle} \ .$$

- *The Walsh transform of a VBF* $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ *is defined as:*

$$W_F : \mathbb{F}_2^n \times \mathbb{F}_2^m \to \mathbb{Z}$$
$$(u, v) \mapsto \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle v,F(x) \rangle+\langle u,x \rangle} \ .$$

- *The Walsh spectrum of a VBF* $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ *is the set of all the values that* $W_F(u, v)$ *can take:*

$$\mathcal{W}(F) = \{W_F(u, v) \mid (u, v) \in \mathbb{F}_2^n \times \mathbb{F}_2^m\} \ .$$

In this chapter, we only get interested in the case where the inputs u and v are of Hamming weight 1. In this case, the value $W_F(u, v)$ can be seen as a measure of the correlation between one input bit and one output bit of F. We will denote by $\mathcal{W}_1(F)$ the subset of the Walsh spectrum restricted to the values u and v of Hamming weight 1:

$$\mathcal{W}_1(F) = \{W_F(u, v) \mid (u, v) \in \mathbb{F}_2^n \times \mathbb{F}_2^m, \mathrm{HW}(u) = \mathrm{HW}(v) = 1\} \ .$$

In the following, we will focus on the Walsh spectrum of encodings. Since the latter are bijections, their coordinate functions are balanced Boolean functions and their Walsh transforms have the following property.

Proposition 1. *Let* $n \geq 2$ *and* $f : \mathbb{F}_2^n \to \mathbb{F}_2$ *be a balanced Boolean function. For all* $u \in \mathbb{F}_2^n$ *such that* $\mathrm{HW}(u) = 1$, $W_f(u)$ *is a multiple of* 4.

Proof. Let $n \geq 2$ and $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a balanced Boolean function. Let $u \in \mathbb{F}_2^n$ with $\mathrm{HW}(u) = 1$. Let us denote by A the biggest set $A \subseteq \mathbb{F}_2^n$ such that $\sum_{x \in A}(-1)^{f(x)+\langle x,u \rangle} = 0$ and by B the set $B = \mathbb{F}_2^n \setminus A$. We then have:

$$W_f(u) = \sum_{x \in B} (-1)^{f(x)+\langle x,u \rangle} \ .$$

The value $(-1)^{f(x)+\langle x,u \rangle}$ is constant over B since otherwise we could add at least two values in A. Therefore, proving that the cardinality of B is a multiple of 4 would conclude this proof. Since $\#B = 2^n - \#A$ and $n \geq 2$, it is also sufficient to prove that the cardinality of A is a multiple of 4.

If $f = x \mapsto \langle x, u \rangle$ or $f = x \mapsto \neg\langle x, u \rangle$, then we have $\#A = 0$, which concludes the proof. Otherwise, since f is balanced, there exist x_1, x_2, x_3 and x_4 in $\mathbb{F}_2^n$ such that:

$$\begin{array}{lll} \langle x_1, u \rangle = 0 & \text{and} & f(x_1) = 1, \\ \langle x_2, u \rangle = 0 & \text{and} & f(x_2) = 0, \\ \langle x_3, u \rangle = 1 & \text{and} & f(x_3) = 1, \\ \langle x_4, u \rangle = 1 & \text{and} & f(x_4) = 0. \end{array}$$

These four values x_1, x_2, x_3 and x_4 are different, so by definition of A, they belong to A. Now let $\mathcal{E} = \mathbb{F}_2^n \setminus \{x_1, x_2, x_3, x_4\}$. If $f(x) = \langle x, u \rangle$ or $f(x) = \neg\langle x, u \rangle$ for all x in $\mathcal{E}$, then $\#A = 4$. Otherwise, the argument above can be applied on $\mathcal{E}$ and the cardinality of A is increased by 4. Recursively, we get that $\#A$ is always a multiple of 4, then so are $\#B$ and $W_f(u)$. □

3.3 AES White-box Implementations

A first idea to secure AES in the white-box model could be to use a unique look-up table representing the bijection of $\mathbb{F}_2^{128}$ between plaintexts and ciphertexts for a given key. This would restrict the possible attacks to those that are feasible in the black-box context, but of course, it is totally impractical. Indeed, one would need to compute and store a $2^{128} \times 128$-bit array. The challenge of white-box cryptography is thus to find an implementation that is practical in terms of time and memory, even if it means giving a little more information to the adversary.

Even though the very first published AES white-box design [CEJv03] is now considered as insecure, the ideas that were introduced are still commonly employed in practice and seen as an important basic security brick. We thus start this section with the description of this implementation. We then briefly go through the other attempts that can be found in the literature.

3.3.1 Chow et al.'s Implementation

Chow et al.'s AES white-box [CEJv03] is inspired from the T-tables suggested by Daemen and Rijmen in their AES proposal [DR98]. The implementation is split into a network of small look-up tables that we describe in the following.

AddRoundKey and SubBytes. At each round r, these operations are computed by sixteen tables called T-boxes that map a byte to a byte and are defined as follows:

$$\begin{cases} T^r_{i,j}(x_{i,j}) = S(x_{i,j} \oplus k^r_{i,j}) & \text{for } i,j \in [0,3] \text{ and } r \in [1,9] \\ T^{10}_{i,j}(x_{i,j}) = S(x_{i,j} \oplus k^{10}_{i,j}) \oplus k^{11}_{i,j-i \bmod 4} & \text{for } i,j \in [0,3] \end{cases}$$

where $k^r_{i,j}$ denotes the byte in position (i,j) of the roundkey k^r.

MixColumns. First, note that the matrix multiplication between MC and a state column can be decomposed into three exclusive-or of 4-byte vectors:

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} x_{0,j} \\ x_{1,j} \\ x_{2,j} \\ x_{3,j} \end{pmatrix} = x_{0,j} \begin{pmatrix} 02 \\ 01 \\ 01 \\ 03 \end{pmatrix} \oplus x_{1,j} \begin{pmatrix} 03 \\ 02 \\ 01 \\ 01 \end{pmatrix} \oplus x_{2,j} \begin{pmatrix} 01 \\ 03 \\ 02 \\ 01 \end{pmatrix} \oplus x_{3,j} \begin{pmatrix} 01 \\ 01 \\ 03 \\ 02 \end{pmatrix}$$

Let us denote by MC_i the i^{th} column of MC, for i in $[\![0,3]\!]$. In Chow et al.'s implementation, each of the four vectors is computed by a table taking a one-byte input and returning a 4-byte output:

$$Ty^r_{i,j}(x_{i,j}) = x_{i,j} \cdot MC_i \ .$$

The results are then given as input to other tables that take two 4-bit values and return their exclusive-or. The reason behind the partitioning in nibbles instead of bytes for this operation is that each xor table would otherwise be quite big ($2^{16} \times 8$ bits = 65.536 kB).

Note that the tables $T^r_{i,j}$ and $Ty^r_{i,j}$ can be merged to improve the performances and reduce the number of intermediate variables. The data flow for the computation of one AES round of AES is depicted in Figure 3.1.

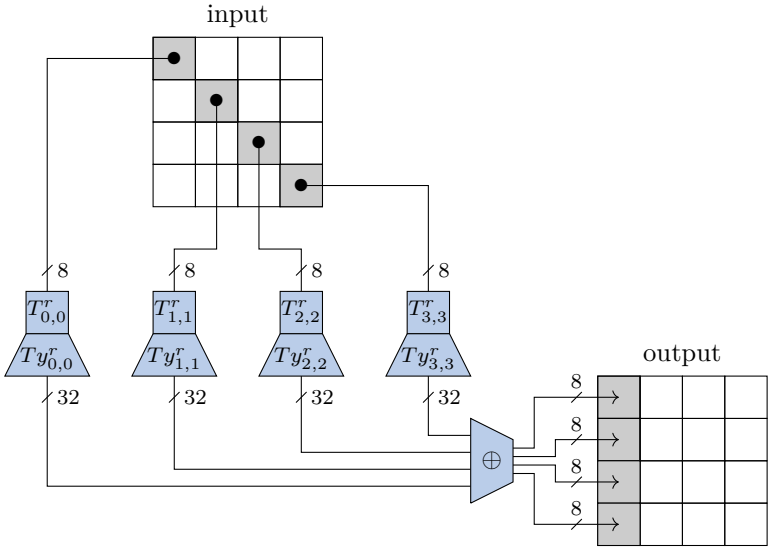


Figure 3.1: Data flow for the computation of an AES round. Only one column is shown but the implementation is similar for the other ones. The xor operation represented here is in fact composed of multiple little xor tables that take two 4-bit values as input.

Adding Encodings. In the implementation outlined previously, the outputs of the Ty tables are exposed to potential adversaries. Yet, these values depend on 8 key bits only, so they are highly sensitive and have to be protected. Otherwise, an attacker could compute all the 256 possible tables and compare them with the ones that are actually used to validate a key hypothesis. As a solution, Chow et al. suggested to add random bijections, called *encodings*, to the inputs and the outputs of the tables.

Definition 2 (**Encoding**). *Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$. Let E_0 an E_1 be bijections over $\mathbb{F}_2^n$ and $\mathbb{F}_2^m$ respectively. The function $\overline{F} = E_1 \circ F \circ E_0$ is called an* encoded function *of F, and E_0 and E_1 are called the* input *and* output encodings *respectively.*

Encodings are used to obfuscate the tables; an attacker reading the memory does not have access to a sensitive variable X anymore, but to $E(X)$ with E being an unknown bijection. They prevent the adversary from computing all possible tables and performing a brute-force attack since the number of bijections over $\mathbb{F}_2^n$ is $2^n!$, which quickly becomes prohibitive when n increases. Due to the design of the xor tables that take two nibbles as inputs, the encodings suggested by Chow et al. are bijections over $\mathbb{F}_2^4$. Output encodings are drawn randomly, independently from each other, and applied to 4-bit portions of intermediate variables. Note that in order to obtain the right result at the end of the algorithm, they have to be canceled by the input encodings of the following operation (see Figure 3.2). Only the external encodings, namely the bijections applied to the input of the first operation and the output of the last one, are not canceled during the execution. They mitigate several attacks since they prevent the adversary from having access to the plaintexts and the ciphertexts, but they make the implementation not standard anymore, which is prohibitive for many use-cases. In Chow et al.'s implementation, these external encodings are linear bijections over $\mathbb{F}_2^{128}$.

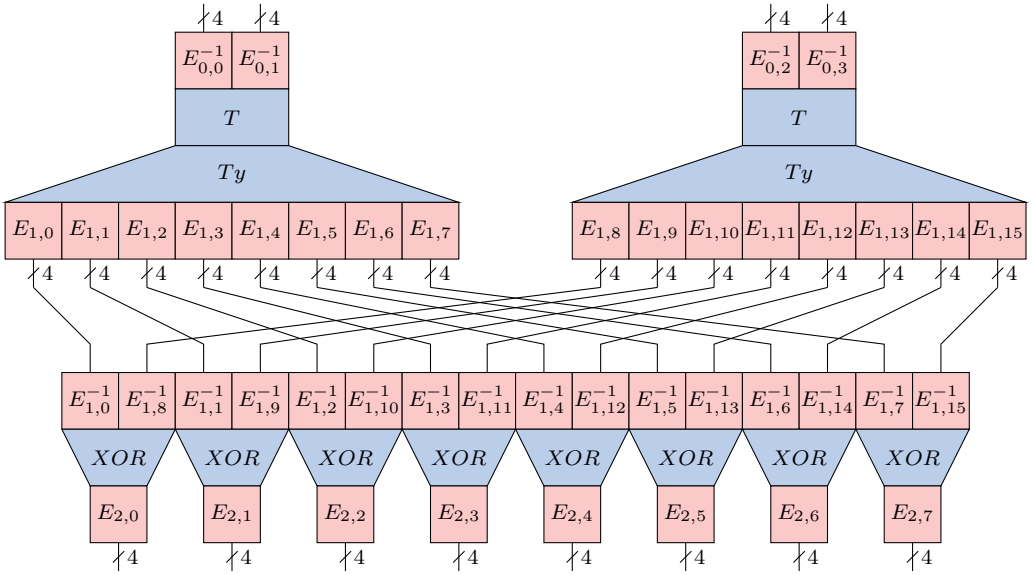


Figure 3.2: A part of an AES round protected by nibble encodings.

Adding Mixing Bijections. In addition to the small encodings that are non-linear with very high probability, Chow et al. suggest to implement bigger linear transformations called mixing bijections in order to increase the diffusion. More precisely, they propose to add :

- 8×8 mixing bijections $L_{i,j}^r$ on the input of T-boxes,
- 32×32 mixing bijections MB_i^r on the output of Ty tables.

Since these transformations are linear, they commute with the xor operations. The bijections MB_i^r can therefore be canceled by new tables that are added at the end of each round. Of course, these tables do not take 32-bit inputs; the inverse transformation takes the form of a multiplication by a matrix, which can be done similarly to the application of MixColumns to the state. As for $L_{i,j}^r$ bijections, their inverse must be applied at the end of the previous round. We refer to [CEJv03] or [Mui13] for more details.

3.3.2 Overview of Other Designs

As we will discuss in Section 3.4, Chow et al.'s implementation has been proven insecure. Consequently, several authors tried to improve it or came up with new designs. They were unfortunately all broken in the end, but we briefly discuss these implementations in this section for completeness. We refer the interested reader to [Ras20] for more details.

Bringer et al. In 2006, Bringer et al. [BCD06] introduced a white-box implementation of a variant of AES with non-standard key-dependent S-boxes. Instead of using look-up tables, each round is represented as a system of multivariate polynomials in a finite field, the structure of which is hidden by the incorporation of random polynomials and perturbation polynomials. The latter can be assimilated to well-controlled errors spread along the execution and canceled in the last round with high probability. Consequently, four carefully crafted instances of the white-box are necessary to ensure the correct result after a majority vote. This implementation was broken by De Mulder et al. in [DWP10].

Xiao and Lai. In 2009, Xiao and Lai [XL09] suggested an amelioration of Chow et al.'s design that consists in selecting 16-bit linear encodings instead of 4-bit non-linear ones. The linearity of the bijections allows to compute the xor operations directly on the encoded values without necessitating look-up tables. Furthermore, the authors thought that encodings impacting two state bytes instead of one could prevent the attacks that were designed against Chow et al.'s white-box, but their implementation was also broken a few years later by De Mulder et al. [DRP13a].

Karroumi. In 2010, Karroumi [Kar11] proposed a method to conceal the structure of the AES white-box implementation introduced by Chow et al. He suggested to use dual ciphers, which are alternative representations of a cipher, meaning that one can switch between them using a known transformation. Karroumi uses the fact that each byte of the AES state can be viewed as an element of $\mathbb{F}_{2^8}$, that can be represented as $\mathbb{F}_2[X]/P$ where P is an irreducible polynomial of degree 8 over $\mathbb{F}_2$. There are 30 such polynomials and one can use any of them to construct the field F_{2^8} and define a distinct polynomial representation of a byte. Karroumi's approach involves randomly selecting an irreducible polynomial, and thus a different AES dual cipher, for each round. However, a few years after the introduction of the design, De Mulder et al. [DRP13b] showed that an encoded dual AES round can be represented by an encoded "normal" AES round using the same key bytes. Hence, Karroumi's white-box implementation is vulnerable to the same attacks as Chow et al.'s.

Luo et al. In 2014, Luo et al. [LLY14] proposed to adapt the implementation of Chow et al. to yet another type of encodings. They took up Xiao and Lai's idea to use large linear transformations that impact several state bytes but added small non-linear encodings on top of them. This implementation is more resistant than previous ones but it can still be broken by Michiels et al.'s generic algorithm for SPN ciphers [MGH09].

Baek et al. In 2016, Baek et al. [BCH16] suggested to implement two AES encryption algorithms at once and encode the round inputs/outputs together with 256-bit linear bijections. The resulting non-standard scheme thus takes and returns modified 256-bit blocks. To reduce storage, Baek et al. use a particular class of encodings that have a sparse structure. Their implementation was broken by Derbez et al. [DFLM18], who showed how to efficiently recover linear encodings of any size applied to any SPN cipher.

3.4 White-Box Attacks

As announced in Section 3.3, all the published AES white-box designs have been proven insecure. In this section, we focus on table-based implementations as introduced by Chow et al. and present the different cryptanalysis that can be found in the literature. We also introduce a new attack based on a round-reduced AES distinguisher that is very efficient in the practical white-box scenario (Section 1.5.4) where the adversary has limited capacities.

3.4.1 BGE Attack and Extensions

Billet et al. [BGEC04] introduced the first algebraic attack on Chow et al.'s AES implementation. Their idea is not to consider every table separately, but to group all the operations performing one AES round. They represent the later with four mappings

R_j^r taking four bytes (x_0, x_1, x_2, x_3) as input and computing a column of the next state (y_0, y_1, y_2, y_3). Figure 3.3 shows one of these mappings. Here, the P_i's are the combination of the two 4-bit input encodings and the 8×8 mixing bijection applied to the corresponding T-box. Similarly, the Q_i's are the combination of an 8×8 mixing bijection and the two 4-bit output encodings applied on top. All the other internal transformations, including the 32×32 mixing bijections, are inverted during the round so they are not considered here.

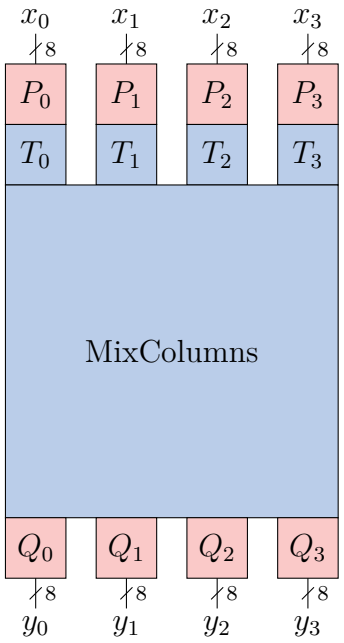


Figure 3.3: Computation of the first column of an AES round output.

Phase 1. The first step of the attack consists in recovering the mappings Q_i up to an unknown affine transformation. To do so, consider the relations between inputs and outputs of an R_j^r mapping:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} Q_0(02 \cdot T_0(P_0(x_0)) \oplus 03 \cdot T_1(P_1(x_1)) \oplus T_2(P_2(x_2)) \oplus T_3(P_3(x_3))) \\ Q_1(T_0(P_0(x_0)) \oplus 02 \cdot T_1(P_1(x_1)) \oplus 03 \cdot T_2(P_2(x_2)) \oplus T_3(P_3(x_3))) \\ Q_2(T_0(P_0(x_0)) \oplus T_1(P_1(x_1)) \oplus 02 \cdot T_2(P_2(x_2)) \oplus 03 \cdot T_3(P_3(x_3))) \\ Q_3(03 \cdot T_0(P_0(x_0)) \oplus T_1(P_1(x_1)) \oplus T_2(P_2(x_2)) \oplus 02 \cdot T_3(P_3(x_3))) \end{pmatrix}$$

By fixing x_1, x_2 and x_3 to constant values c_1, c_2 and c_3 and making x_0 vary, one can obtain the look-up tables of the bijections

$$f_{i,c_1,c_2,c_3}(x_0) = Q_i(\alpha_i \cdot T_i(P_i(x_0)) \oplus \beta_{c_1,c_2,c_3}) \ ,$$

where α_i is an entry of the MixColumns matrix and β_{c_1,c_2,c_3} is an unknown constant value. From there, one can compute

$$f_{i,c_1,c_2,c_3} \circ f^{-1}_{i,c'_1,c_2,c_3} = Q_i \circ \oplus_\beta \circ Q_i^{-1}$$

for all value of $\beta = \beta_{c_1,c_2,c_3} \oplus \beta_{c'_1,c_2,c_3}$ in $\mathbb{F}_2^8$. Billet et al. showed that this knowledge is enough to recover Q_i up to an unknown affine transformation. In other words, they introduced an efficient algorithm (later improved by Tolhuizen [Tol12]) that allows one to

find $\tilde{Q}_i = Q_i \circ A_i$ with A_i affine from the look-up tables of the 256 functions $Q_i \circ \oplus_\beta \circ Q_i^{-1}$. The inverse of $\tilde{Q}_i$ can then be applied to the corresponding round outputs so that the later are now encoded by the affine bijection

$$\tilde{Q}_i{}^{-1} \circ Q_i = A_i^{-1} \circ Q_i^{-1} \circ Q_i = A_i^{-1} \ .$$

Since the input encodings P_i are the inverse of the output encodings Q_i, it is also possible to recover them up to an affine transformation.

Phase 2. The next step is then to find the remaining affine encodings, and Billet et al. gave an efficient algorithm to do so. Note however that it is specific to AES since it uses properties of the MixColumns operation. Derbez et al. [DFLM18] later proposed a generic and efficient algorithm to recover affine encodings of any size for any SPN cipher.

Phase 3. Once the input and output encodings of one round transformation are fully recovered, the corresponding roundkey can easily be computed. Indeed, the inverses of MixColumns, ShiftRows and SubBytes operations can be applied to the round output to obtain the state after AddRoundKey. The latter, together with the input of the round, trivially leads to the corresponding subkey. The master key can then be computed by inverting the key scheduling.

All in all, the BGE attack enables the recovery of the whole secret key with a work factor of about 2^{30}. Note that the latter was then reduced to 2^{22} thanks to some optimizations [Tol12, LRD+14]. This attack was at first specific to AES but it was later generalized to other SPN ciphers [MGH09].

The Attack in Practice. Although efficient against many table-based white-box implementations, BGE might not be the simplest choice for real-world adversaries. Indeed, it is complicated to understand and implement, requires some reverse engineering to find the round outputs, and turns out impractical if the attacker cannot execute the white-box himself since it is a chosen plaintext attack that involves a lot of executions.

3.4.2 Collision Attack

Another efficient attack against Chow et al.'s implementation was introduced by Lepoint and Rivain [LR13]. Based on collision analysis, this attack is conceptually easier than the one proposed by Billet et al. for an equivalent work factor. However, contrary to the latter, it requires the knowledge of the order of the state bytes.

In the following, we will denote by f the function that maps four bytes of a plaintext to a column of the first round's output, and by

$$f' = (Q_0 \| Q_1 \| Q_2 \| Q_3) \circ f \circ (P_0 \| P_1 \| P_2 \| P_3)$$

the encoded version of f. Let f'_i be the 32-to-8-bit coordinate functions of f', with $f' = (f'_0, f'_1, f'_2, f'_3)$. The attack consists of the two following phases.

Phase 1. Let us denote by S_i for $0 \leq i \leq 3$ the functions

$$S_i : x \mapsto S(k_i^{(1)} \oplus P_i(x))$$

and note that

$$f' = (Q_0 \| Q_1 \| Q_2 \| Q_3) \circ MC \circ (S_0 \| S_1 \| S_2 \| S_3) \ .$$

The objective of the first phase of the attack is to recover these functions, starting with S_0 and S_1. To do so, Lepoint and Rivain proposed to look for collisions of the form

$$f'_0(\alpha, 0, 0, 0) = f'_0(0, \beta, 0, 0) \ ,$$

which imply that

$$Q_0(02 \cdot S_0(\alpha) \oplus 03 \cdot S_1(0) \oplus c) = Q_0(02 \cdot S_0(0) \oplus 03 \cdot S_1(\beta) \oplus c)$$

with $c = S_2(0) \oplus S_3(0)$, and thus

$$02 \cdot S_0(\alpha) \oplus 03 \cdot S_1(0) = 02 \cdot S_0(0) \oplus 03 \cdot S_1(\beta) \ . \tag{3.1}$$

Proceeding similarly with f'_1, f'_2 and f'_3, one can get 4×255 equations with 512 unknowns ($S_i(x)$ for $i \in \{0, 1\}$ and $x \in \mathbb{F}_2^8$), but unfortunately this system is not of full rank. It is possible to get down to 510 unknowns with a change of variable that consists in considering $u_i = S_0(0) \oplus S_0(i)$ and $v_i = S_1(0) \oplus S_1(i)$. Then, (3.1) can be written as

$$02 \cdot u_\alpha \oplus 03 \cdot v_\beta = 0.$$

In all their experiments, the authors found that the systems were of rank 509, so all the unknowns could be expressed in terms of one of them, say u_1. They thus proposed to do an exhaustive search on $S_0(0)$ and $S_0(1)$ before solving the system (we refer to [LR13] or [LRD+14] for more details). This directly gives the look-up table of S_0 and an additional exhaustive search on $S_1(0)$ leads to the one of S_1.

Once S_0 and S_1 have both been recovered, one can proceed similarly with S_2 and S_3, solving the systems arising from collisions of the form $f'_i(\alpha, 0, 0, 0) = f'_i(0, 0, \beta, 0)$ and $f'_i(\alpha, 0, 0, 0) = f'_i(0, 0, 0, \beta)$.

Phase 2. The second phase of the attack consists in computing the subkey of the second round. First of all, notice that phase 1 allows an attacker to recover the output encodings of the first round. Indeed, by evaluating $f'_0(\alpha, 0, 0, 0)$, one can get the value $Q_0(\psi(\alpha))$ where

$$\psi : \alpha \mapsto 02 \cdot S_0(\alpha) \oplus 03 \cdot S_1(0) \oplus S_2(0) \oplus S_3(0)$$

is a bijective function known by the attacker. The latter can therefore fully retrieve

$$Q_0(x) = f'_0(\psi^{-1}(x), 0, 0, 0)$$

by looping on the 256 input values. Of course, the other output encodings Q_i can be recovered in a similar way, allowing the decoding of the second round input.

Now, let us denote by f'' the second round of AES with $P_i^{(2)}$ (resp. $Q_i^{(2)}$) as input (resp. output) encodings. For $0 \leq i \leq 3$, we define $S_i^{(2)}$ as

$$S_i^{(2)} : x \mapsto S(k_i^{(2)} \oplus P_i^{(2)}(x)) \ .$$

For the recovery of $k_0^{(2)}$, consider the function

$$g : x \mapsto f''_0(P_0^{(2)^{-1}}(k_0^{(2)} \oplus S^{-1}(x)), 0, 0, 0) = Q_0^{(2)}(02 \cdot x \oplus c)$$

with $c = 03 \cdot S_1^{(2)}(0) \oplus S_2^{(2)}(0) \oplus S_3^{(2)}(0)$. The idea is to make a key guess, compute the look-up table of g by executing the white-box, and check if the hypothesis is correct thanks to a distinguisher. Note that with the right key guess, the algebraic degree is at most 4 by definition of the encodings proposed by Chow et al. In the opposite, an incorrect key hypothesis leads to an algebraic degree that is greater than 4 with overwhelming probability since the effect of passing through the AES S-box is not canceled. Therefore, Lepoint and Rivain proposed to compute a 4th-order derivative of g and validate the key guess if it is the null function. Of course, the other key bytes can be recovered in a similar way.

The Attack in Practice. Compared to BGE, the collision attack is conceptually simpler and requires less white-box executions. Nevertheless, it is also a chosen plaintext attack that implies some reverse engineering to find round outputs. Furthermore, the attacker needs to chose the inputs of round 2 at some point, which is not possible without lifting the code or inducing several well-controlled faults.

3.4.3 Square Attack

As discussed in Section 1.5.4, there is a gap between white-box cryptography as presented in the literature and as deployed in practice. For instance, all the published designs use external encodings even if it is not always possible in real-life scenarios for reasons of interoperability. Furthermore, in the literature, the difficulty of attacks is not always evaluated according to all parameters that have an impact in real life. For instance, the number of required white-box executions is rarely mentioned. It is thus possible that designs considered as "broken" could be usable under the actual conditions of a product's use. In particular, in contexts such as mobile payment applications where the white-box can be bound to an application with authenticating mechanisms, the attacker might not be able to execute the code himself, making chosen plaintext attacks such as BGE or the collision one impractical.

We thus think that it is interesting to study other attacks that could actually be performed by such an adversary. In this section, we present a new way to recover the secret key of a table-based implementation without external encodings. It is actually a white-box variant of the square attack [DR98] on round reduced versions of AES, and proves to be a good solution for passive attackers who cannot choose the messages to encrypt and can only observe a very limited number of executions. Before describing it, we give a remainder on the previous square attacks that were published in the literature.

Black-Box/Grey-Box Square Attacks

The square attack is a chosen plaintext attack exploiting byte-oriented structures that was first described against the Square cipher [DKR97]. As shown by Daemen and Rijmen in [DR98], it is also applicable to reduced versions of AES of up to 6 rounds. The attack is based on the analysis of the structure of δ-sets through the AES operations.

Definition 3. *A* δ-set *is a set of 256 states with the following properties:*

- *Some bytes are active: they take a different value in all of the states.*
- *The other bytes are passive: their value is constant in all of the states.*

Note that the AddRoundKey and SubBytes operations do not change the structure of a δ-set and ShiftRows only impacts the indexes of the active bytes (see Figure 3.4). However, applying MixColumns on a δ-set does not necessarily maintain the structure:

- A column consisting of passive bytes only will remain passive.
- A column with only one active byte will become entirely active.
- If we apply the MixColumns operation on a column with several active bytes, then the resulting bytes are neither active nor passive but they present an interesting property: the xor of their value in the 256 states always equals zero. This gives a distinguisher for round reduced versions of AES.

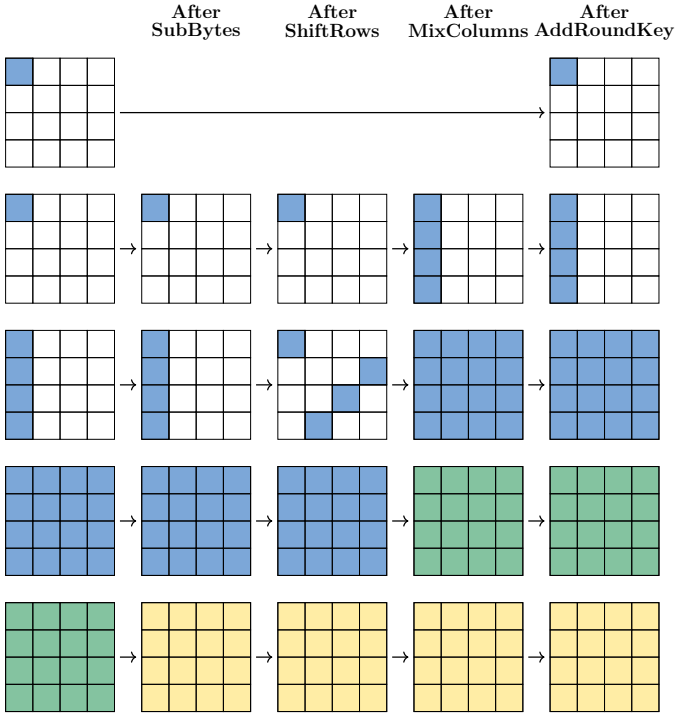


Figure 3.4: Evolution of a δ-set through the AES operations. Passive bytes are represented in white and active bytes in blue. The XOR of green bytes for all 256 states equals zero. The color yellow symbolizes the loss of any structure.

In 2004, Carlier et al. [CCDP04] generalized the Square attack and showed how to break a grey-box implementation of the full AES by mixing algebraic properties and physical observations. They use a set of 2^{32} states with 4 active bytes on the main diagonal as plaintexts for the attack (see Figure 3.5). The latter consists of two phases:

- During the observation phase, the attacker executes the targeted AES implementation and uses side-channel traces in order to distinguish a set G of 2^{16} plaintexts that lead to two passive bytes in the first column of the first round output.
- During the attack phase, a 4-byte key hypothesis is made and the first column of the first round output is computed for all plaintexts in G. If the obtained outputs form a set with two passives bytes, then the key guess is confirmed. Note that all the plaintexts do not have to be encrypted for all key guesses since as soon as a

contradiction appears (the targeted bytes are not passive), we can reject the key hypothesis.

Once the four key bytes have been recovered, the same process can be repeated with other indexes until the full round-key is recovered.

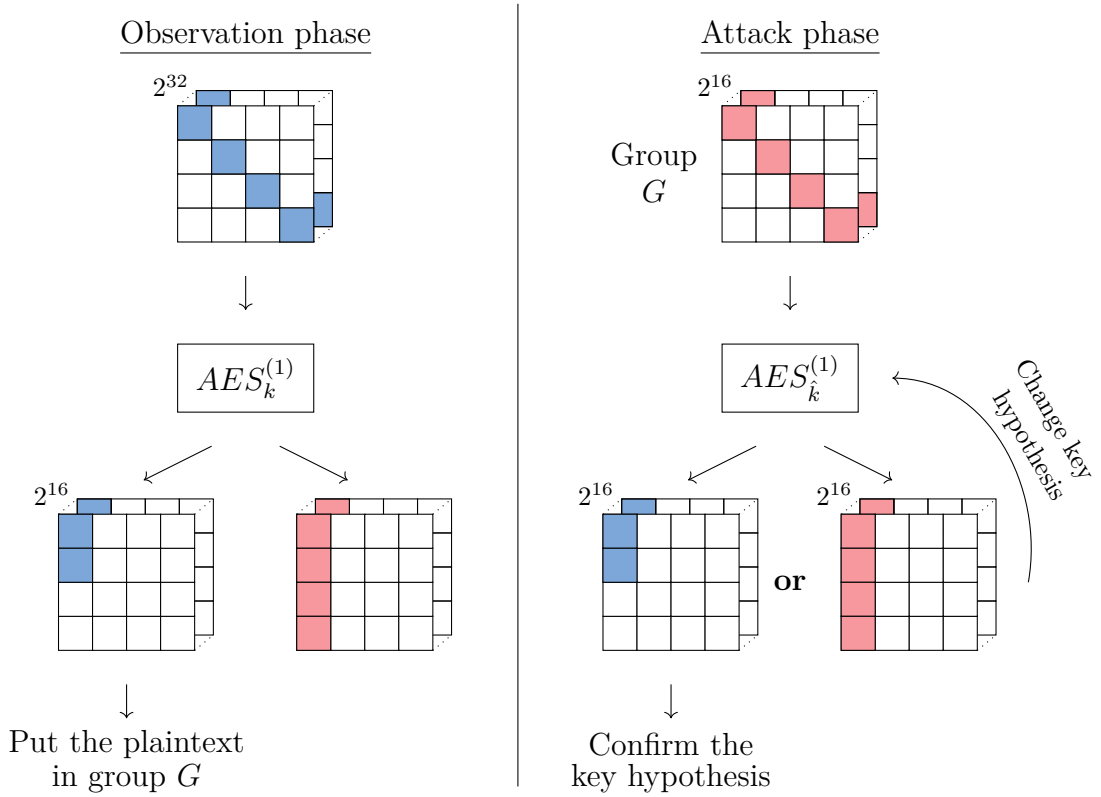


Figure 3.5: Square EM attack from [CCDP04]. Here, the bytes in white are passive, those in blue are active and those in pink vary. In the observation phase, the AES operations are performed through the execution of the target implementation while in the attack phase, they are computed based on a key guess $\hat{k}$.

A New Square Attack for the White-Box Context

The side-channel attack of [CCDP04] that we just described could directly be applied to AES white-boxes that use 4-bit or 8-bit internal encodings to protect the round outputs, as in Chow et al.'s implementation (the mixing bijections impacting 32 bits are canceled inside the round). Indeed, applying such encodings does not change the fact that some bytes are passive/active or not and only modifies their value. However, this attack would require hypotheses on four bytes of the secret key and 4×2^{32} calls to the white-box implementation, which would represent more than six months if each execution took one millisecond, so it is definitely not practical.

We thus propose to use the additional advantages that a white-box adversary has in order to improve this attack in our specific context. Instead of using side-channel traces, the attacker may directly read the memory during the observation phase in order to extract the encoded values of the targeted bytes. Thanks to this better precision, he can form during the observation phase 256 distinct groups G_i containing the plaintexts corresponding to each output of the first round with the first byte equal to i (see Figure 3.6). In the attack phase, the adversary makes a key hypothesis and computes the outputs

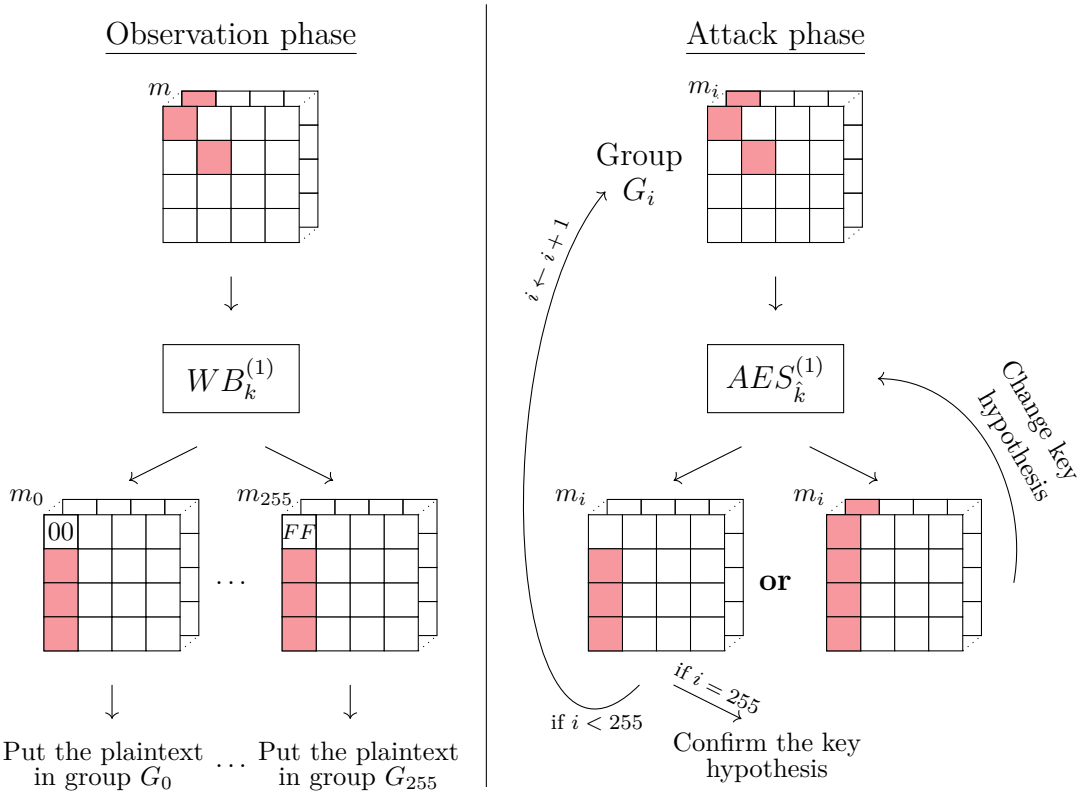


Figure 3.6: Our attack scheme. Here, the bytes in white are constant in all the states of the set while the bytes in pink may vary.

of the first round of AES (restricted to the first column) for each plaintext in G_0. Since the white-box implementation is encoded, the first byte will not necessarily be equal to 0 even if the key hypothesis is correct. Nevertheless, in that case, the first byte should remain constant. If this is verified, the attacker can test the results for the group G_1, G_2 and so on until G_{255}. If at one point the outputs do not share the same first byte, then the key hypothesis is rejected.

Note that with this method, we loose a lot of information since we do not consider the values of the three other output bytes. In order to improve this attack, one could thus create 4×256 different groups corresponding to all possible values for each byte of the first column of the output and test all of them in the attack phase. The number of AES encryptions performed in this second phase does not increase. Each plaintext is put into four different groups but it can be encrypted once for all. With this solution, more information is derived from each execution so the number of white-box calls can be reduced. The attacker can settle for a set of m plaintexts with m far smaller than 2^{32}. According to our experiments (see Figure 3.7), it is sufficient to take $m = 23$ (resp $m = 36$) to get a success probability over 99% (resp 99.99%), so the performance gain is huge. Note that this is the total number of executions needed for the recovery of the whole secret key since the attack can be performed in parallel on the four columns.

Furthermore, one can also reduce the complexity of the attack phase at the cost of a few more white-box executions. Indeed, the attacker does not necessarily need to make all the four plaintext bytes vary. If he is able to choose the messages to encrypt, he may reduce the size of the key hypotheses by setting two appropriate bytes of the plaintexts to a constant as shown in Figure 3.6. Denoting the four bytes by (x_0, x_1, x_2, x_3), with x_2

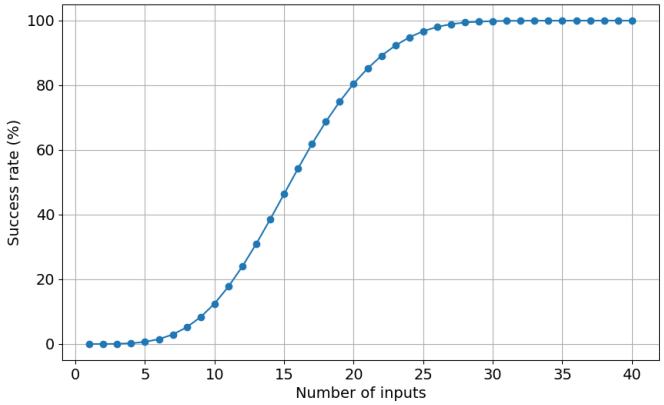


Figure 3.7: Success rate of our attack depending on the number m of inputs to the white-box. A million tests were performed for each point on a toy white-box implementation with random 8-bit encodings.

and x_3 constant, the outputs of MixColumns can be computed as:

$$\begin{aligned} s &= \mathrm{S}(x_0 \oplus k_0) \oplus \mathrm{S}(x_1 \oplus k_1) \oplus 2 \cdot \mathrm{S}(x_2 \oplus k_2) \oplus 3 \cdot \mathrm{S}(x_3 \oplus k_3) \\ &= \mathrm{S}(x_0 \oplus k_0) \oplus \mathrm{S}(x_1 \oplus k_1) \oplus c \end{aligned}$$

for some unknown constant c. They can thus be predicted, up to a constant, under an only 16-bit key hypothesis and c does not hinder the attacker since he is not interested in the value of those bytes but rather on collisions. Putting everything together, the whole secret key can be recovered with overwhelming probability with only $2 \times 36 = 72$ white-box executions on chosen plaintexts and a worst case complexity of $2 \times 36 \times 2^{16} \approx 2^{22}$ computations of the first round of AES for the attack phase.

Note nonetheless that if the attacker cannot choose the inputs of the white-box and is only able to observe the intermediate values, then restricting himself to the case where two bytes are fixed would force him to observe many executions. Indeed, only 1 plaintext over 2^{16} has the desired shape and the attack could not be run in parallel on the four columns anymore. A compromise may thus be chosen between the complexity of the observation phase and the one of the attack phase depending on the attacker's capabilities (see table 3.1).

Comparison with State-of-the-Art. Our variant of the square attack has several advantages. First, it is easy to describe and to implement compared to BGE or the collision attacks. More importantly, it requires a very small number of white-box executions and the messages to encrypt do not need to be chosen. We showed that with 36 random plaintexts, the attacker can recover the secret key with a probability close to 99.99%. In comparison, the first phases alone of BGE and the collision attack respectively involve 2^{18} and 2^{12} executions. Finally, another advantage of our attack is that it works with linear or non-linear encodings of size 8 and can be extended for even bigger ones.

Nonetheless, we realize that comparing the square attack with BGE and the collision attack is not really fair since it does not tackle external encodings. Despite our opinion on their utility in practice, we did try to break them by solving a system of equations built thanks to the square distinguisher. While we were able to retrieve a lot of information

Table 3.1: Compromises between the complexities of the two phases in the chosen and the random plaintext scenarios. Here, m is the number of inputs used to confirm a key guess.

Number of bytes varying	Scenario	Average number of executions	Maximum number of AES computations
2	Chosen Plaintexts	$2 \times m$	$2 \times m \times 2^{16}$
	Random Plaintexts	$4 \times 2 \times m \times 2^{16}$	$4 \times 2 \times m \times 2^{16}$
4	Any	m	$m \times 2^{32}$

on the encodings, the resulting attack was not competitive with other ones that were published in the literature.

A fairer comparison would thus be the square-like attack on a weakened white-box implementation without external encodings that was introduced in the original paper of Chow et al. [CEJv03]. While it uses similar techniques, it requires the observation of about 2^{13} executions before performing a brute-force on 32 bits of the key, and is therefore less efficient. Another fair comparison would be with side-channel attacks. Although the original idea is very different, we realized that our attack exploits the exact same vulnerability as the collision side-channel attack of [RW19] that we will describe in Section 3.5.1. It is thus highly similar, except that it gets the advantages and drawbacks of side-channel attacks. In particular, it is a bit less efficient but does not require a reverse engineering phase to find the round outputs.

3.5 Side-Channel Attacks: a State-of-the-Art

Compared to the attacks described in the previous section, side-channel attacks have the advantage of being fully automated: they do not require any reverse engineering step[1]. Furthermore, as mentioned in Section 2.1, the absence of inherent noise on the traces makes these attacks highly efficient and they usually require a very small number of executions on random plaintexts. They can thus be performed by adversaries with limited capabilities. However, the drawback is that to this day, no side-channel attack allows to break an implementation protected by external encodings. Indeed, the adversary has to know the inputs or the outputs of the cryptosystem in order to be able to retrieve secret information. Nevertheless, this drawback is acceptable since, as previously discussed, external encodings are rarely employed in practice for interoperability reasons. In the following, we will thus assume that the external encodings of the target white-boxes are set to the identity.

[1]There is an exception when the execution time is particularly high and the relevant portions of the code need to be selected to maintain the trace size within reasonable limits.

3.5.1 Distinguishers

The general principle of side-channel attacks has been explained in Section 2.1. In a few words, secret values are extracted from leakage traces with the help of statistical tools. In the following, we present the various criteria, known as *distinguishers*, that have been proposed in the literature to identify the correct key used in a white-box implementation from a set of hypotheses.

DCA. In 2016, Bos et al. [BHMT16] proposed to adapt the well-known Differential Power Analysis (DPA) of Kocher et al. [KJJ99] to the white-box context. The resulting attack, called Differential Computation Analysis (DCA), can be used to break a majority of the publicly available AES implementations. While the original paper focuses on differential analysis, the attack has later been extended to other statistical tools, and in particular correlation analysis [BCO04]. It then consists of the following steps:

1. Acquire a set T of computation traces of length t. Let T_i be the variable taking the values of the trace point at index i with $0 \leq i < t$.
2. Select a sensitive variable V_k that depends on a relatively small number of key bits. For example for AES, one could select the output of an Sbox in the first round, that is $V_k = \mathrm{S}(x \oplus k)$ for any byte x of the plaintext and the corresponding byte k of the first roundkey.
3. For each key guess $\hat{k}$ and each point index $0 \leq i < t$, compute $\rho_{\hat{k},i} = \mathrm{Cor}(V_{\hat{k}}, T_i)$ with Cor representing Pearson's correlation coefficient. Note that in the rest of the chapter, we will often drop the subscript and denote by $\rho^{\times}$ (resp. ρ^{*}) a score for a wrong (resp. correct) key hypothesis.
4. Validate the key hypothesis that maximizes $\rho_{\hat{k},i}$.
5. Repeat with another sensitive variable depending on other key bits until the whole roundkey is recovered. The master key can then be computed by inverting the key-scheduling.

Note that without any countermeasure, the correlation should equal 1, that is the maximum value, for the correct key guess and point index. The attack is thus successful with overwhelming probability even with a very limited number of traces.

Collisions. In 2019, Rivain and Wang [RW19] proposed to use two other distinguishers to break AES white-boxes. The first one is based on collision analysis and the attack is as follows:

1. Acquire a set of n computation traces of length t. For each pair of traces $(u^{(i_1)}, u^{(i_2)})$ with $0 \leq i_1, i_2 < n$, compute a correlation trace

$$w^{(i_1,i_2)} = (w_0^{(i_1,i_2)}, w_1^{(i_1,i_2)}, \ldots, w_{t-1}^{(i_1,i_2)}) \ ,$$

with $w_j^{(i_1,i_2)} = u_j^{(i_1)} \odot u_j^{(i_2)}$ for every $0 \leq j < t$ where the operator $\odot$ is defined as

$$a \odot b = \begin{cases} 1 \text{ if } a = b \\ 0 \text{ otherwise} \end{cases} \ .$$

2. Select a sensitive variable V_k that depends on a few key bits only. For example for AES, one could select an output byte of the first MixColumns. Note that targeting an SBox output of round 1 as proposed for DCA would not be pertinent since two different plaintext bytes cannot lead to the same output of SubBytes.

3. For each key guess $\hat{k}$ and each pair of plaintexts used to acquire traces, compute V_k and predict if it collides or not. Then, for each point index of the collision traces, compute the correlation between the sample and the collision prediction.

4. Validate the key hypothesis that maximizes the correlation.

5. Repeat with another sensitive variable depending on other key bits until the whole roundkey is recovered. The master key can then be computed by inverting the key-scheduling.

Again, in the absence of countermeasure, the correlation should equal 1 for the correct key guess and point index. As we will see in Section 3.5.2, the motivation behind such a distinguisher is that it performs better than DCA in the presence of internal encodings.

MIA. The second distinguisher investigated in [RW19] for the white-box context was actually already introduced by [GBTP08] to break hardware implementations. It is particularly interesting when the attacker only has a very limited knowledge about the dependencies between the leakage and the processed data. The attack is exactly the same as DCA, except that instead of measuring linear correlations, the attacker analyses mutual information between the trace points and the predicted values. This way, he may gain information on all dependencies, whether linear or not.

3.5.2 Encodings vs SCA

Intuitively, even if they were not introduced for this purpose, internal encodings should complicate DCA since they lower the correlation between the predicted values and the processed data. Nevertheless, Bos et al. showed in [BHMT16] that DCA allows to break several white-box instances that use this protection. At first, this was not well understood and the results were purely experimental, but other authors then tried to understand the reason behind the success of side-channel attacks on encoded implementations. In this section, we discuss the state-of-the-art on the subject.

Sasdrich et al. In 2016, Sasdrich et al. [SMG16] proposed to use white-box techniques to protect an AES implementation on a re-configurable hardware device. In particular, they used mixing bijections and random 4-bit non-linear encodings as suggested by Chow et al. The authors evaluated the vulnerability of the resulting implementation to several side-channel attacks in the grey-box context. They targeted the outputs of the first-round S-boxes and successfully retrieved the secret key, proving that Chow et al.'s encodings cannot efficiently protect an AES implementation against side-channel attacks, even in the grey-box model with noisy power traces.

Sasdrich et al. also investigated the reason behind the success of their attack and showed that the encoded outputs y_j of MixColumns are actually correlated to some unencoded S-box outputs x_i despite the internal encodings. They computed the Walsh spectrum of the functions $f(x_i) = y_j$ and argued that the high values lead to high correlation scores, explaining the success of the attack. They also stressed that selecting

encodings such that their Walsh transforms are all equal to zero may not be a viable strategy since this would lead to correlation scores so low that they could also be detected. The construction of secure encodings was left for future research.

Lee. In [Lee18], Lee continued the work of Sasdrich et al. and tried to use encodings to prevent side-channel attacks in the grey-box context. He proposed to use two complementary sets of look-up tables. More precisely, if a set of tables $\mathcal{T} = \{T_0, T_1, \ldots, T_\ell\}$ is used in a classical encoded implementation, then Lee suggested to also create a set of tables $\bar{\mathcal{T}} = \{\bar{T_0}, \bar{T_1}, \ldots, \bar{T_\ell}\}$ such that

$$y = T_i[x] \Leftrightarrow \bar{y} = \bar{T_i}[\bar{x}]$$

with $\bar{x}$ being the binary complement of x (see Figure 3.8). At the beginning of an execution, one of the two sets of look-up tables is drawn at random. If the set $\bar{\mathcal{T}}$ is chosen, then the plaintext and the ciphertext have to be complemented to get the correct result.

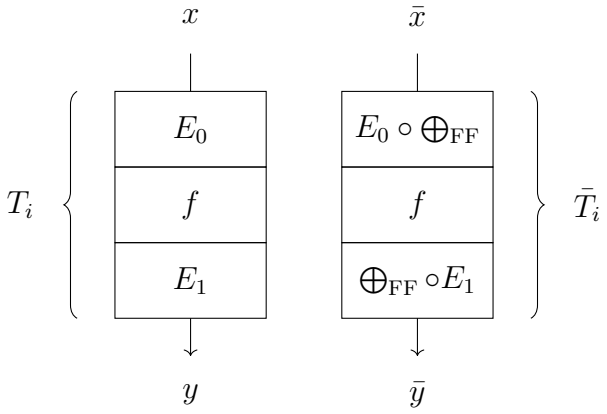


Figure 3.8: Construction of the tables T_i and $\bar{T_i}$. Here, E_0 and E_1 are two random 8-bit encodings and f is the operation to be protected. The idea can be extended to any encoding size.

The output encodings for the tables T_i being drawn at random, they are very likely to contain high values in their Walsh spectrum, leading to high correlations. Nevertheless, Lee argued that if the used set of tables varies at each execution, the trace points will sometimes negatively and sometimes positively correlate. The correlation coefficient should thus tend to 0, preventing the attacks.

Although very costly in terms of memory, Lee's solution seems to work well in the grey-box model. But what about the white-box context? Note that implementing this solution implies drawing random values during the execution of the algorithm, which is known to be challenging to secure as discussed in Section 2.1.5. This is precisely the reason why it would have been interesting to use encodings instead of masking techniques against side-channel attacks. One could argue that the number of random values to be drawn is far smaller with this solution, but still, if the attacker is able to predict or fix them, the countermeasure becomes inefficient. By observing the accessed memory addresses, the adversary may also be able to differentiate the executions where the set of tables $\bar{\mathcal{T}}$ was used instead of $\mathcal{T}$. He could then bypass the countermeasure by attacking only one half of the traces.

Alpirez Bock et al. As Sasdrich et al., Alpirez Bock et al. [ABMT18] also studied the link between internal encodings and the success of DCA even if they did not use the Walsh transform to do so. They considered two types of encodings: 4-bit non-linear and 8-bit linear ones. Note that the former are randomly selected in the set of bijections over $\mathbb{F}_2^4$ and the latter are generated via a random 8×8 invertible binary matrix that will be multiplied with the state.

Alpirez Bock et al. showed that an attack targeting an output bit of a first-round S-box protected by 8-bit linear encodings succeeds if and only if at least one of the rows of the matrix generating the encoding has a Hamming weight equal to 1. Indeed in that case, one of the sensitive bits is leaked since it is equal to one of the encoded bits. Otherwise, the linear encoding acts as a masking scheme and prevents DCA. However, the authors stress out that the secret key may still be recovered through a more sophisticated attack that consists in analyzing the correlations between the trace points and any linear combination of the predicted sensitive bits.

Regarding the non-linear 4-bit encodings, Alpirez Bock et al. proved that they lead to correlation scores that are always multiples of $\frac{1}{4}$ for the correct key guess. While a score greater or equal to 0.5 leads to a successful attack with very high probability, a correlation coefficient of 0 or 0.25 might be exceeded by other values corresponding to wrong hypotheses. Nevertheless, the authors argued that the attack can be improved by considering only key guesses leading to coefficient scores very close to a multiple of $\frac{1}{4}$ so that wrong hypothesis with a high score are quickly discarded.

Finally, Alpirez Bock et al. considered the combination of linear and non-linear encodings. They showed that depending on a given property of the matrix generating the linear encoding, the correlation score for the right key guess either converges towards 0 or is a multiple of $\frac{1}{4}$, so this scenario is quite similar to the other ones described previously.

Rivain and Wang. One year after Alpirez Bock et al.'s paper, Rivain and Wang [RW19] took over their studies and generalized some of their results. In order to compute the success probability of DCA when targeting an encoded implementation, they first analyzed the distribution of the correlation scores under good and bad key hypotheses. To do so, they placed themselves in an idealized model where the predicted variables can be expressed as $V_{\hat{k},j} = \varphi_{\hat{k},j}(X)$ with X a uniformly distributed plaintext variable, $(\varphi_{\hat{k}})_{\hat{k}\in K}$ some independent random balanced VBF from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ and $\varphi_{\hat{k},j}$ their restriction to the j^{th} output bit. For the sake of clarity, we denote by φ^* (resp. $\varphi^\times$) the functions corresponding to the good (resp. a bad) key guess.

Let i be the point index corresponding to the targeted encoded variable, that is $T_i = \epsilon_\ell \circ \varphi^*(X)$ with ϵ some unknown m-bit encoding and ϵ_ℓ its restriction to the ℓ^{th} output bit. In the idealized model, we have

$$\mathrm{Cor}(V_{\hat{k},j}, T_i) = \frac{1}{2^n} B(\epsilon_\ell \circ \varphi^* + \varphi_{\hat{k},j}) \tag{3.2}$$

where B denotes the bias of a Boolean function. Now, consider the following lemma.

Lemma 1. *Let $g : \mathbb{F}_2^n \to \mathbb{F}_2$ be a balanced Boolean function. Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a random function uniformly sampled in the set of balanced Boolean functions independently of g. We have*

$$B(f+g) = 4N - 2^n \text{ with } N \sim \widetilde{\mathcal{HG}}(n),$$

where $\widetilde{\mathcal{HG}}(n)$ is the hypergeometric distribution with parameters $(2^{n-1}, 2^n, 2^{n-1})$.

Let $Y^{\times}$ be the bias $B(\epsilon_\ell \circ \varphi^* + \varphi_j^{\times})$ and Y^* be the bias $B(\epsilon_\ell \circ \varphi^* + \varphi_j^*)$. From Lemma 1, we get that for a wrong key guess,

$$Y^{\times} = 4N - 2^n \text{ with } N \sim \widetilde{\mathcal{HG}}(n) \ . \tag{3.3}$$

Now, for the correct key hypothesis, we have

$$Y^* = B(\epsilon_\ell \circ \varphi^* + \varphi_j^*) = 2^{n-m} B(\epsilon_\ell + l_j)$$

where $l_j(x) = x_j$ (the j^{th} coordinate of x). This time, from Lemma 1, one gets

$$Y^* = 2^{n-m+2} N - 2^n \text{ with } N \sim \widetilde{\mathcal{HG}}(m) \tag{3.4}$$

Equation (3.2) implies that for DCA to succeed, the encoding ϵ protecting the sensitive variable must be such that Y^* is greater than $Y^{\times}$ for all incorrect key hypotheses, and the number of traces must be high enough to get a good accuracy in the correlation estimation. From (3.3) and (3.4), Rivain and Wang derived a formula for the probability of success of DCA. We refer the interested reader to [RW19, Proposition 2] to get the exact formula.

In the particular cases of random 4-bit encodings, the analysis is consistent with the one of Alpirez Bock et al. Contrary to what was believed at the time, the work of Rivain and Wang also showed that it is actually possible to break an AES white-box implementation using random non-linear 8-bit encodings by targeting the output of the MixColumns instead of the one of the S-box. If we focus on this cryptosystem and on encoding lengths that are small powers of 2 – which are more convenient for implementation – the conclusion of their work is roughly the following:

- An attack targeting an S-box output will succeed with very high probability if the encodings are only applied on nibbles while it will succeed with very low probability in the presence of 8-bit encodings.
- An attack targeting the output of the MixColumns will succeed with very high probability even with 8-bit encodings.

Rivain and Wang also did a similar study for the two other distinguishers that they introduce: the one based on collisions and MIA. The advantage of the first one is that encodings do not alter the success probability of the attack since encoded values collide whenever their unencoded versions do. Concerning the second one, it was precisely introduced in the grey-box context for use-cases where the leakage function is odd or complex. It is therefore totally suited for the white-box setting where only an unknown function of the sensitive values is leaked because of the encodings. Consequently, MIA and the collision attacks are expected to be more efficient than DCA, and Rivain and Wang indeed showed that the number of traces that they require is significantly lower, that is in $O(2^{\frac{m}{2}})$ instead of $O(2^{2m})$.

3.6 On the (Im)possibility of preventing DCA with Internal Encodings

Many open questions were answered in [RW19] but one remains. Indeed, the authors gave success probabilities but did not explain in which case the attacks were successful or not.

In other words, they proved that picking encodings at random is not a good strategy to protect white-box implementations against side-channel attacks, but did not discuss the idea of carefully crafting them. The question that arises from their work is thus the one of the existence of a particular class of encodings that effectively prevents DCA.

In this section, we look for such DCA-resistant encodings for white-box implementations of AES, focusing on 4-bit and 8-bit bijections for practical reasons. We show, based on Sasdrich et al.'s work, that the S-box output can be protected by random 8-bit encodings only. In particular, we argue that no 4-bit encoding can preserve the S-box output from DCA-like attacks. Regarding MixColumns, although it has been demonstrated that random 8-bit encodings are ineffective, we define a class of 8-bit encodings that actually prevent DCA. However, we also explain how to adapt the traditional attack in order to defeat even these specific bijections. We exhibit a difference of behavior of the correlation coefficient depending on the key hypothesis, allowing the attacker to identify the correct key guess, regardless of the 8-bit encodings applied. Therefore, we show that MixColumns, and thus AES, cannot be protected from DCA by encodings of practical size.

Part of the work presented in this section is currently under publication at CHES 2024 [CH24].

3.6.1 Protecting the S-box Output

In [RW19], Rivain and Wang showed that in an idealized model, DCA succeeds with probability close to 0.926 when targeting one bit of a first round's S-box output protected by 4-bit encodings, while an implementation using 8-bit encodings is broken with probability 0.0025 only. In the following, we show that it is actually impossible to find 4-bit encodings that are efficient against side-channel attacks. We also demonstrate that selecting 8-bit encodings with specific properties in order to further reduce the DCA success probability of 0.0025 is counterproductive. We indeed argue that randomly drawing 8-bit encodings is both necessary and sufficient to prevent DCA.

Why Not to Use 4-bit Encodings

Since they are less expensive than their 8-bit counterpart in terms of memory, 4-bit encodings are often used in the literature. For instance, if 8-bit non-linear encodings were used in Chow et al.'s implementation, the total memory space per round of XOR-tables would amount to $4 \cdot 2^{16} \cdot 8$ bits ≈ 262 KB instead of $8 \cdot 2^8 \cdot 4$ bits ≈ 1 KB. Finding a class of 4-bit encodings that could be used to prevent side-channel attacks targeting the S-box outputs would thus be of high interest. Unfortunately, we will now show that such encodings do not exist.

We know from [SMG16] that the success of DCA is directly linked to the Walsh spectrum of the encoding that hides the targeted value. Our study consists in computing all the possible values for the Walsh transforms of 4-bit encodings and showing that the implementation can be broken by side-channel attacks for all of them[2]. Let us start with a proposition.

Proposition 2. *Let $E^{(1)}, E^{(2)} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be two bijections with $n \geq 2$. Let $f = E^{(1)} \| E^{(2)}$ and $\{f_i\}_{0 \leq i < 2n}$ be the coordinate functions of f. For all $0 \leq i < 2n$ and for all $\omega \in \mathbb{F}_2^{2n}$ with $\mathrm{HW}(\omega) = 1$, $W_{f_i}(\omega)$ is a multiple of 2^{n+2}.*

[2]The process that we followed for the generation of the encodings with a specific Walsh spectrum that we used for our tests can be found in Appendix A.

Proof. For the sake of clarity, we will assimilate $\mathbb{F}_2^n$ to the set of integers $[\![0, 2^n - 1]\!]$ using the bijection $(x_0, x_1, \ldots, x_{m-1}) \mapsto x_0 + 2x_1 + \cdots + 2^{m-1}x_{m-1}$.
We will only prove the proposition for $0 \leq i < n$, but the demonstration is similar for $n \leq i < 2n$. Let $\mathcal{S}$ be the set:

$$\mathcal{S} = \{\{u + 2^n v \mid v \in [\![0, 2^n - 1]\!]\} \mid u \in [\![0, 2^n - 1]\!]\}.$$

By construction, $\mathcal{S}$ forms a partition of $[\![0, 2^{2n} - 1]\!]$, so for all $\omega \in \mathbb{F}_2^{2n}$, we have

$$\begin{aligned} W_{f_i}(\omega) &= \sum_{x \in \mathbb{F}_2^{2n}} (-1)^{f_i(x) + \langle x, \omega \rangle} \\ &= \sum_{S \in \mathcal{S}} \sum_{x \in S} (-1)^{f_i(x) + \langle x, \omega \rangle} \quad . \end{aligned}$$

When $0 \leq i < n$, f_i is constant over S for all $S \in \mathcal{S}$ so for any $x_S \in S$, we get:

$$W_{f_i}(\omega) = \sum_{S \in \mathcal{S}} (-1)^{f_i(x_S)} \sum_{x \in S} (-1)^{\langle x, \omega \rangle} \quad . \tag{3.5}$$

If $\omega < 2^n$ and $\mathrm{HW}(\omega) = 1$, then the function $g : S \to \mathbb{F}_2$ such that $g(x) = \langle x, \omega \rangle$ is balanced for all $S \in \mathcal{S}$, so $\sum_{x \in S} (-1)^{\langle x, \omega \rangle} = 0$ and $W_{f_i}(\omega) = 0$, which is obviously a multiple of 2^{n+2}.

If $\omega \geq 2^n$ and $\mathrm{HW}(\omega) = 1$, the function $g : S \to \mathbb{F}_2$ such that $g(x) = \langle x, \omega \rangle$ is constant for all $S \in \mathcal{S}$, so:

$$\sum_{x \in S} (-1)^{\langle x, \omega \rangle} = \#S \cdot (-1)^{\langle x_S, \omega \rangle} \quad .$$

By construction, all the sets in $\mathcal{S}$ share the same cardinality, hence (3.5) becomes:

$$W_{f_i}(\omega) = 2^n \sum_{S \in \mathcal{S}} (-1)^{f_i(x_S) + \langle x_S, \omega \rangle} \quad . \tag{3.6}$$

Furthermore, the integers $0, 1, \ldots, 2^{n-1}$ are all in different sets. Then, by choosing them as representatives of their respective set, (3.6) can be re-written as:

$$W_{f_i}(\omega) = 2^n \sum_{0 \leq x_s < 2^n} (-1)^{f_i(x_s) + \langle x_S, \omega \rangle} \quad .$$

The above sum corresponds to the Walsh transform of a coordinate function of $E^{(2)}$. Note that this function is balanced on $\mathbb{F}_2^n$ because $E^{(2)}$ is a bijection, so Proposition 1 applies and its Walsh transform is a multiple of 4. Hence, $W_{f_i}(\omega)$ is a multiple of 2^{n+2}. □

Proposition 2 states that the Walsh spectrum of a function corresponding to the concatenation of two 4-bit encodings only contains multiples of 64. Through the following proposition, this gives information on the possible correlation scores under the correct key guess when targeting one bit of an S-box output protected by 4-bit encodings.

Proposition 3. *Let $E : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be an encoding. We have:*

$$\mathrm{Cor}(E_i(x), x_j) = \frac{1}{2^n} \cdot W_{E_i}(2^j)$$

for each bit x_j of a random variable x following the uniform distribution over $\mathbb{F}_2^n$ and each coordinate function E_i of E, with $0 \leq i, j < n$.

Proof. Let $f_1, f_2 : \mathbb{F}_2^n \to \mathbb{F}_2$ be two balanced Boolean functions and x be a random variable following the uniform distribution over $\mathbb{F}_2^n$. According to [RW19, Section 2.4], we have:

$$\mathrm{Cor}(f_1(x), f_2(x)) = \frac{1}{2^n} \sum_{u \in \mathbb{F}_2^n} (-1)^{f_1(u) \oplus f_2(u)} \ . \tag{3.7}$$

Since E is a bijection, all its coordinate functions are balanced. Furthermore, the bit x_j of x can be written as the output of the function $g_j(x) = \langle x, 2^j \rangle$ which is also balanced. Hence, we get:

$$\begin{aligned} \mathrm{Cor}(E_i(x), x_j) &= \mathrm{Cor}(E_i(x), g_j(x)) \\ &= \frac{1}{2^n} \sum_{u \in \mathbb{F}_2^n} (-1)^{E_i(u) + \langle u, 2^j \rangle} \\ &= \frac{1}{2^n} \cdot W_{E_i}(2^j) \ . \end{aligned}$$

□

Therefore, if one applies 4-bit encodings on the first round's S-Box outputs, the correlation scores will always be multiples of $64/256 = 1/4$. Note that this was also shown in a different manner by Alpirez Bock et al. in [ABMT18].

Proposition 3 implies that, in order to resist DCA, one should select encodings with only small (absolute) values in their Walsh spectrum. One could then be tempted to craft encodings E such that $\mathcal{W}_1(E) = \{0\}$. This means that, for any coordinate function E_i of E and any bit x_j of the targeted intermediate value $x = \mathrm{S}(m \oplus k)$, $\mathrm{Cor}(E_i(x), x_j) = 0$. Therefore, the classical DCA would fail, but a variant that consists in searching the key hypothesis with the lowest maximum correlation score would succeed with very high probability, as we explain now.

Let $n \geq 2$ and $u \in \mathbb{F}_2^n$ with $\mathrm{HW}(u) = 1$. Let f be a random variable following the uniform distribution on the set B_n of balanced n-variable Boolean functions. A consequence of [RW19, Lemma 1] is that $W_f(u)$ can be modeled as an affine transformation of a random variable X following a hypergeometric distribution with parameters $(2^{n-1}, 2^n, 2^{n-1})$:

$$W_f(u) \sim 4X - 2^n \ . \tag{3.8}$$

Proposition 3 trivially translates (3.8) into:

$$\mathrm{Cor}(f(x), x_j) \sim \frac{4}{2^n} X - 1 \ . \tag{3.9}$$

Let $x^{(\hat{k})}$ be a predicted S-box output computed under the key hypothesis $\hat{k}$. Then, setting $y = E(x)$, we get $\mathrm{Cor}(E_i(x), x_j^{(\hat{k})}) = \mathrm{Cor}(E_i(x), g^{(\hat{k})}(E(x))) = \mathrm{Cor}(g^{(\hat{k})}(y), y_i)$ with:

$$\begin{aligned} g^{(\hat{k})} : \mathbb{F}_2^8 &\to \mathbb{F}_2 \\ z &\mapsto \mathrm{S}_j(\mathrm{S}^{-1}(E^{-1}(z)) \oplus \hat{k} \oplus k) \ . \end{aligned}$$

Thanks to the good cryptographic properties of S, we can assimilate the functions $g^{(\hat{k})}$, $\hat{k} \in K \setminus \{k\}$, to outcomes of a random variable following the uniform distribution on B_8. Then, (3.9) holds, which means that, for all $k^\times \in K \setminus \{k\}$, $\mathrm{Cor}(E_i(x), x_j^{(k^\times)})$ is an outcome of the random variable $\mathrm{Cor}(f(x), x_j)$. Therefore:

$$\mathbb{P}(\mathrm{Cor}(E_i(x), x_j^{(k^\times)}) = 0) = \mathbb{P}\left(X = \frac{2^8}{4}\right) \ .$$

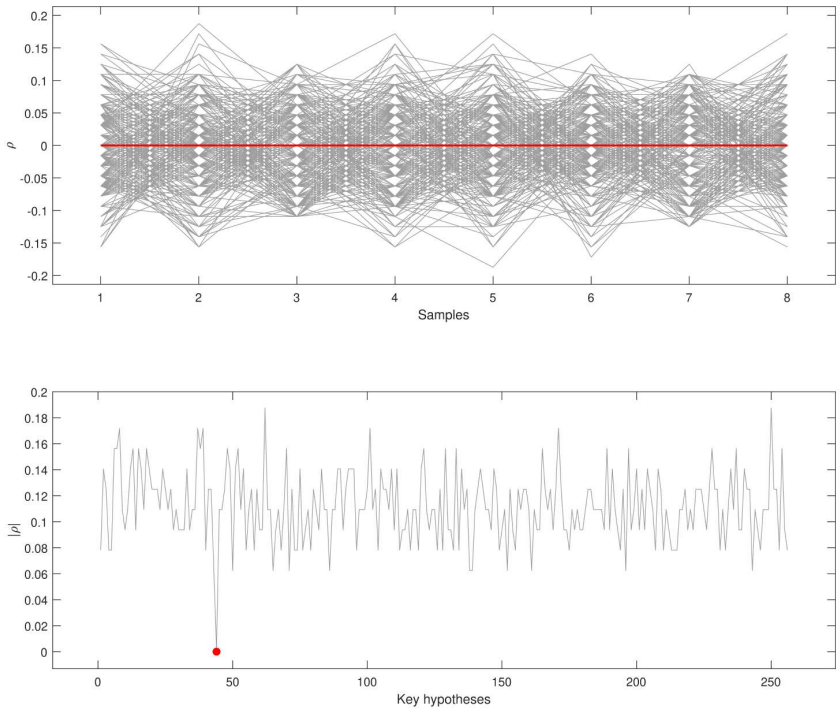


Figure 3.9: DCA targeting the most significant bit of the first round's S-box output protected by 4-bit encodings having only zero in their Walsh spectrum. Top: correlation traces for wrong (resp. good) key guesses in grey (resp. red). Bottom: highest absolute value of the correlation scores for each key guess (good hypothesis highlighted in red).

The probability for $\rho^{\times}$ to be equal to 0 for a wrong key guess $k^{\times}$ is the probability that $\mathrm{Cor}(E_i(x), x_j^{(k^{\times})}) = 0$ for all $0 \leq i < 8$. Let us assume, as in [RW19], that the coordinate functions $E_i(x)$ and $E_{i'}(x)$ are two independent and identically distributed random variables when $i \neq i'$. Then:

$$\mathbb{P}(\rho^{\times} = 0) = \mathbb{P}\left(X = \frac{2^8}{4}\right)^8.$$

Consequently, the correct key guess is the only one for which $\rho = 0$ with probability $(1 - \mathbb{P}(X = 2^6)^8)^{\#K-1} \approx 1 - 2^{-18.6}$. Therefore, the correct key guess can be identified with very high probability as the only one that never gives any correlation. We verified this experimentally (see Figure 3.9), performing a DCA targeting the output of S_7 on 2^8 simulated traces: one trace per possible value of the input byte m, each trace being computed as $E_0 \circ \mathrm{S}(m \oplus k) || \ldots || E_7 \circ \mathrm{S}(m \oplus k)$.

Since selecting encodings $E = E^{(1)} \| E^{(2)}$ such that $\mathcal{W}_1(E) = \{0\}$ does not work, one could think of crafting encodings $E^{(1)}$ and $E^{(2)}$ with only the lowest possible non-zero (absolute) value in their Walsh spectrum. The latter being equal to 64 in the case of 4-bit encodings applied to the S-box output, we would have $\mathcal{W}_1(E^{(i)}) \subseteq \{-64, 64\}$ for $i \in \{1, 2\}$. Then, if one targets the bit $x_j^{(k)}$ with $0 \leq j < 4$, the correlation peaks should equal ± 0.25 for all point $E_i(x)$ with $0 \leq i < 4$. Similarly, if one targets the bit $x_j^{(k)}$ with $4 \leq j < 8$, the correlation peaks should equal ± 0.25 for all point $E_i(x)$ with $4 \leq i < 8$. This phenomenon can be observed in Figure 3.10.

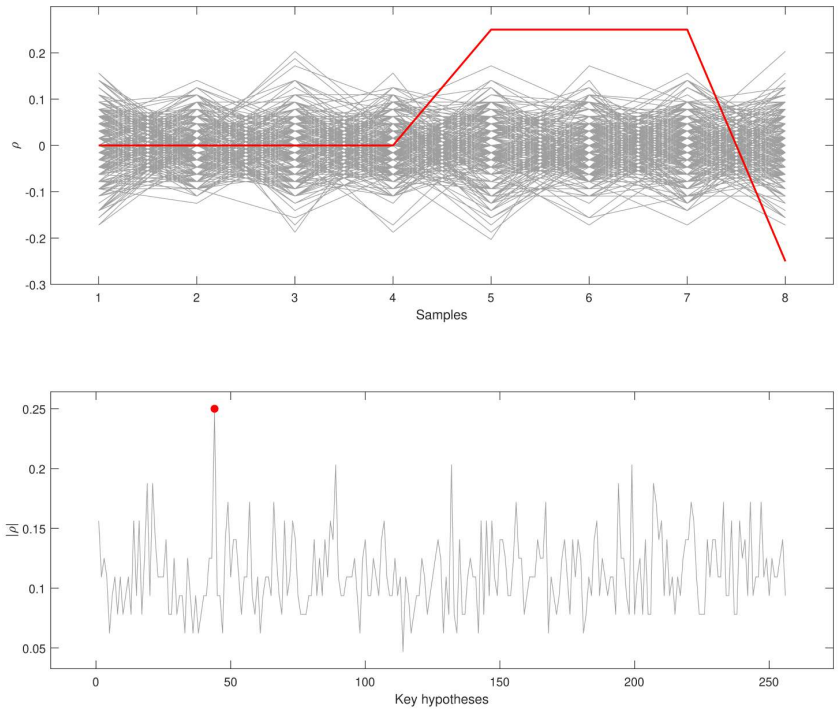


Figure 3.10: DCA targeting the most significant bit of the S-box output. Here, the Walsh spectrum of the 4-bit encodings do not contain any other value than 64 or -64. Top: correlation traces for wrong (resp. good) key guesses in grey (resp. red). Bottom: highest absolute value of the correlation scores for each key guess (good hypothesis highlighted in red).

The probability that a wrong key hypothesis exhibits a correlation score higher than 0.25 on some of the eight points equals:

$$\mathbb{P}\left(|\rho^{\times}| > \frac{1}{4}\right) = 1 - \mathbb{P}\left(\frac{2^8 - 2^6}{4} < X < \frac{2^8 + 2^6}{4}\right)^8 \approx 2^{-10.3} .$$

It would thus be very unlikely that DCA does not succeed. Furthermore, even if there are some higher peaks for some bad key guesses, the attacker could just focus on key hypotheses that give a correlation coefficient equal to ± 0.25 as noticed by Alpirez Bock et al. [ABMT18].

Moreover, an attacker searching for the least correlation score could still succeed. Indeed, as a consequence of (3.5), if one targets the bit $x_j^{(k)}$ with $0 \leq j < 4$, then the correlation scores under the correct key guess for the points $E_i(x)$ would equal 0 for all $4 \leq i < 8$. Similarly, if one targets the bit $x_j^{(k)}$ with $4 \leq j < 8$, then the correlation scores under the correct key guess for the points $E_i(x)$ would equal 0 as well for all $0 \leq i < 4$. The probability of this happening under a wrong key guess is $\mathbb{P}(\rho^{\times} = 0) = \mathbb{P}(X = 2^8/4)^4 \approx 2^{-13.3}$.

Therefore, we can conclude that a white-box implementation using 4-bit encodings to protect the S-box output can always be broken by DCA. The attacker could first perform a classical DCA and see if any key hypothesis gives very high correlations. If this is not

the case, the attacker could verify if any key hypothesis gives correlation scores equal to 0 or ± 0.25.

On 8-bit Encodings

Rivain and Wang [RW19] showed that applying random 8-bit encodings instead of two concatenated 4-bit ones decreases the success probability of DCA from 0.926 to approximately 0.0025. While this probability is notably low, it is not absolute zero. One might thus consider discarding encodings with Walsh spectrum values exceeding a certain threshold. This would prevent an excessively high correlation score for the correct key guess, but somewhat unexpectedly, it is actually counterproductive. Indeed, as we explain in the following, randomly selecting 8-bit encodings allows to decorrelate the sensitive value from the trace points, which is the best possible protection.

Let $E : \mathbb{F}_2^8 \to \mathbb{F}_2^8$ be a random encoding and E_i for $0 \leq i < 8$ be its coordinate functions. As argued previously, for all $k^\times \in K \setminus \{k\}$, $\mathrm{Cor}(E_i(x), x_j^{(k^\times)})$ is an outcome of the random variable $\mathrm{Cor}(f(x), x_j)$. Moreover, so is $\mathrm{Cor}(E_i(x), x_j^{(k^*)})$, as the random selection of E from the set of bijections in $\mathbb{F}_2^8$ enables E_i to be considered an outcome of a random variable following $\mathcal{U}(B_8)$. This implies that the correlation scores of all key hypotheses follow the same distribution. Consequently, the probability of observing a given score is independent of the key byte's value, and all hypotheses share an equal probability of obtaining the highest correlation score.

Another way to get this intuition is to realise that for any wrong key guess $k^\times$, there exists an encoding E' such that $E'(\mathrm{S}(x \oplus k^\times)) = E(\mathrm{S}(x \oplus k))$, namely

$$E'(z) = E(\mathrm{S}(\mathrm{S}^{-1}(z) \oplus k^\times \oplus k)) \ .$$

This shows that without the knowledge of the encodings, the trace points do not give any information on the correct key byte. Note that a similar argument can be given for the protection of other sensitive variables, as long as the encoded value is in bijection with the part of the plaintext involved in its computation. In other words, randomly selecting encodings having the same size as the sensitive variable that they are supposed to protect is an efficient countermeasure against side-channel attacks.

In summary, random 8-bit encodings can thus serve as a flawless solution for the protection of the S-box output. Unfortunately, we will see in the next section that securing the MixColumns output is a far greater challenge.

Remark 1. At first glance, the success rate of 0.0025 computed by Rivain and Wang might seem contradictory to our conclusion which might mistakenly suggest that the success probability should equal $1/256 \approx 0.0039$. Nevertheless, it should be noted that the success probability given by Rivain and Wang corresponds to the event where the correct key guess obtains the maximum score *alone*. It is thus normal that this probability is lower than 1/256. When considering the possibility of multiple candidates achieving the maximum score simultaneously, the success rate is actually higher than 1/256 since the events "$k^{(i)}$ has the maximum score" and "$k^{(j)}$ has the maximum score" are not disjoint.

3.6.2 Protecting the MixColumns Output

Attack Scenario

Considering what has been discussed in the previous section to protect the S-box output, a straightforward way to protect the output of the first-round's MixColumns from a side-

channel attacker would consist in evaluating MixColumns through a large look-up table that takes a 32-bit input and returns a 32-bit value encoded through a bijection of $\mathbb{F}_2^{32}$. The unaffordable drawback of this solution is the huge amount of memory that each such table would require. For this reason, tables $Ty_{i,j}^r$ are used in [CEJv03], protected by either 4-bit or 8-bit encodings. Since they contain the S-box output by definition of the MixColumns matrix, they cannot be protected against DCA by 4-bit encodings. Therefore, we only consider in the following the case of 8-bit encodings.

Although it is computationally feasible, guessing four bytes of the key to predict an output byte of the first round's MixColumns can be quite long. Nevertheless, if the attacker is able to choose the messages to encrypt, he may reduce the number of hypotheses by setting two appropriate bytes of the plaintexts to a constant. If we denote the plaintext by (m_0, m_1, m_2, m_3), with m_2 and m_3 two constants, the targeted byte s can be written as:

$$\begin{aligned} s &= \mathrm{S}(m_0 \oplus k_0) \oplus \mathrm{S}(m_1 \oplus k_1) \oplus 2 \cdot \mathrm{S}(m_2 \oplus k_2) \oplus 3 \cdot \mathrm{S}(m_3 \oplus k_3) \\ &= \mathrm{S}(m_0 \oplus k_0) \oplus \mathrm{S}(m_1 \oplus k_1) \oplus c \end{aligned}$$

for some unknown constant c. Then, the value $\mathrm{S}(m_0 \oplus k_0) \oplus \mathrm{S}(m_1 \oplus k_1)$ can be predicted under an only 16-bit key hypothesis while c does not hinder the attacker since the addition of a constant only affects the sign of the correlation scores.

In such an attack scenario, Rivain and Wang showed in [RW19] that if the encodings hiding the output of the MixColumns are random bijections over $\mathbb{F}_2^8$, the success probability of DCA when targeting one bit of s is very close to 1 – approximately 0.99995. Nevertheless, this probability is not 1, so the problem that naturally arises from this result is to describe the set of encodings that prevent these attacks.

On Encodings with a Null Walsh Spectrum

We know that in order to reduce the correlation scores between a sensitive variable and its encoded version, one needs to select an encoding with a Walsh spectrum that contains only small (absolute) values. The question of describing the set of encodings that prevent DCA then boils down to finding an appropriate bound on the maximum tolerable (absolute) value in the Walsh spectrum of the encoding.

Let us first investigate the case $\mathcal{W}_1(E) = \{0\}$. We have seen in Section 3.6.1 that when targeting an S-box output protected by an encoding with a Walsh spectrum containing only 0, the attacker may distinguish the correct key guess as the one showing the smaller correlation. Consequently, one could wonder if the same phenomenon happens when targeting a MixColumns output. We thus made an experiment. Let $f : \mathbb{F}_2^8 \times \mathbb{F}_2^8 \to \mathbb{F}_2^8$ be the function $(x_0, x_1) \mapsto \mathrm{S}(x_0 \oplus k_0) \oplus \mathrm{S}(x_1 \oplus k_1)$ and f_i with $0 \leq i < 8$ be its coordinate functions. We targeted the output of f_0 on 2^{16} simulated traces: one trace per possible value of (x_0, x_1), each trace being computed as $E_0 \circ f(x_0, x_1) || \ldots || E_7 \circ f(x_0, x_1)$. The result of this experiment is shown in Figure 3.11.

Contrary to what we observed for the S-box output in Figure 3.9, the correct key guess is far from being the only one with a correlation score at 0 when $\mathcal{W}_1(E) = \{0\}$. In fact, we count exactly 511 key hypotheses with a null score out of 2^{16}. Since the application of (3.3) gives a probability $\mathbb{P}(\rho^\times = 0) \approx 2^{-58.6}$ that an incorrect key guess produces a null score, these hypotheses cannot be a random outcome. Figure 3.12 shows a structured behaviour that strengthens this intuition. To understand what happens, we have to study in depth the behaviour of the correlation coefficient when the key guess is incorrect.

Since the functions f and $E \circ f$ are surjective and balanced, so are their coordinate functions. Thus, for any $0 \leq i, j < 8$, the correlation score between $E_j \circ f$ and f_i can be

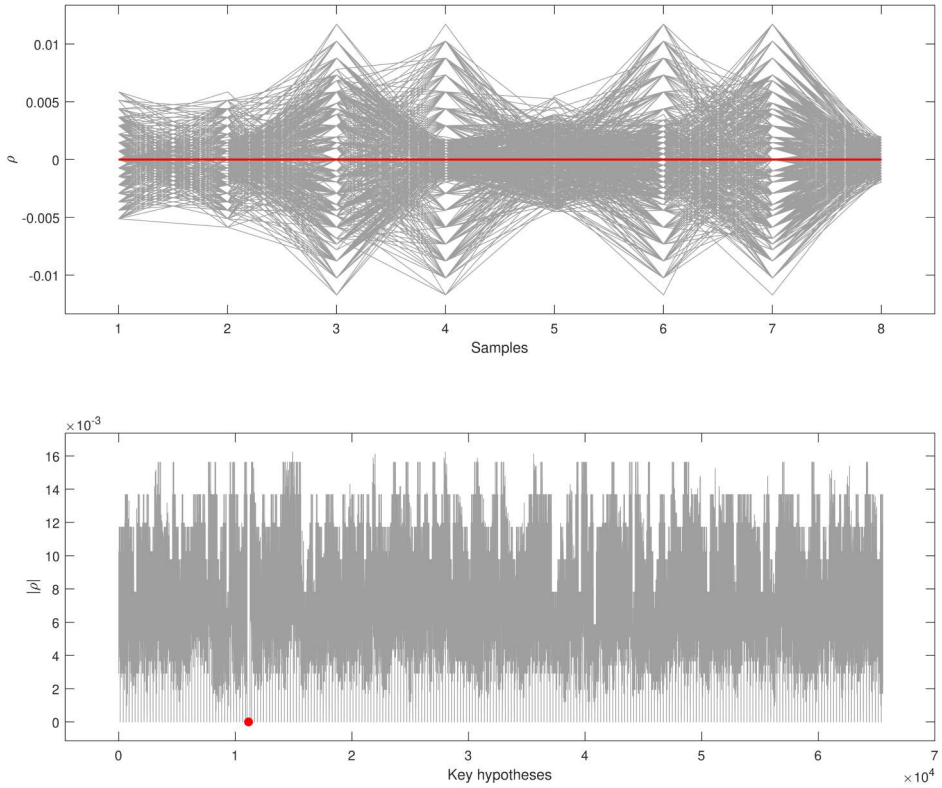


Figure 3.11: DCA targeting f_0. Here, the Walsh spectrum of the 8-bit encoding contains only 0. Top: correlation traces for wrong (resp. good) key guesses in grey (resp. red). Bottom: highest absolute value of the correlation scores for each key guess (good hypothesis highlighted in red).

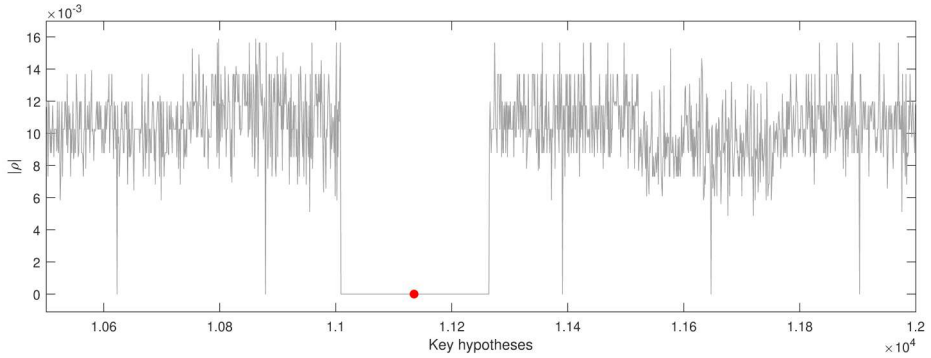


Figure 3.12: Zoom on the correct key guess of the bottom part of Figure 3.11.

calculated using (3.7), seeing f as a function from $\mathbb{F}_2^{16}$ to $\mathbb{F}_2^8$:

$$\begin{aligned}\mathrm{Cor}(E_j \circ f, f_i) &= \frac{1}{2^{16}} \sum_{x_0 \in \mathbb{F}_2^8} \sum_{x_1 \in \mathbb{F}_2^8} (-1)^{E_j \circ f(x_0,x_1) \oplus f_i(x_0,x_1)} \\ &= \frac{1}{2^{16}} \sum_{x_0 \in \mathbb{F}_2^8} \sum_{x_1 \in \mathbb{F}_2^8} (-1)^{E_j \circ f(x_0,x_1) \oplus \mathrm{S}_i(x_0 \oplus k_0) \oplus \mathrm{S}_i(x_1 \oplus k_1)} \quad .\end{aligned}$$

Among all the 2^{16} possible hypotheses for the couple (k_0, k_1), let us consider the elements from the set $\{(k_0', k_1)\}_{k_0' \in \mathbb{F}_2^8}$. Let $f' : (x_0, x_1) \mapsto S(x_0 \oplus k_0') \oplus S(x_1 \oplus k_1)$. The correlation scores between $E_j \circ f$ and f_i' can be written as:

$$\begin{aligned}\mathrm{Cor}(E_j \circ f, f_i') &= \frac{1}{2^{16}} \sum_{x_0 \in \mathbb{F}_2^8} \sum_{x_1 \in \mathbb{F}_2^8} (-1)^{E_j \circ f(x_0,x_1) \oplus \mathrm{S}_i(x_0 \oplus k_0') \oplus \mathrm{S}_i(x_1 \oplus k_1)} \\ &= \frac{1}{2^{16}} \sum_{x_0 \in \mathbb{F}_2^8} \sum_{x_1 \in \mathbb{F}_2^8} (-1)^{E_j \circ f(x_0,x_1) \oplus \mathrm{S}_i(x_0 \oplus k_0') \oplus \mathrm{S}_i(x_0 \oplus k_0) \oplus \mathrm{S}_i(x_0 \oplus k_0) \oplus \mathrm{S}_i(x_1 \oplus k_1)} \\ &= \frac{1}{2^{16}} \sum_{x_0 \in \mathbb{F}_2^8} (-1)^{\mathrm{S}_i(x_0 \oplus k_0') \oplus \mathrm{S}_i(x_0 \oplus k_0)} \sum_{x_1 \in \mathbb{F}_2^8} (-1)^{E_j \circ f(x_0,x_1) \oplus \mathrm{S}_i(x_0 \oplus k_0) \oplus \mathrm{S}_i(x_1 \oplus k_1)} \quad .\end{aligned}$$

Notice that the sum over x_1 is equal to $W_{E_j}(\omega)$ for ω such that $f_i = \langle f, \omega \rangle$. Indeed, for all $x_0 \in \mathbb{F}_2^8$, $x_1 \mapsto f(x_0, x_1)$ is a bijection so we can make the change of variable $u = f(x_0, x_1)$ and write:

$$\begin{aligned}\mathrm{Cor}(E_j \circ f, f_i') &= \frac{1}{2^{16}} \sum_{x_0 \in \mathbb{F}_2^8} (-1)^{\mathrm{S}_i(x_0 \oplus k_0') \oplus \mathrm{S}_i(x_0 \oplus k_0)} \sum_{u \in \mathbb{F}_2^8} (-1)^{E_j(u) \oplus \langle u, \omega \rangle} \\ &= W_{E_j}(\omega) \cdot \frac{1}{2^{16}} \sum_{x_0 \in \mathbb{F}_2^8} (-1)^{\mathrm{S}_i(x_0 \oplus k_0') \oplus \mathrm{S}_i(x_0 \oplus k_0)} \quad . \qquad (3.10)\end{aligned}$$

Therefore, the correlation scores obtained under a wrong key guess (k_0', k_1) are multiples of the Walsh transform of some coordinate function of the encoding. The same reasoning can be applied to key hypotheses of the form (k_0, k_1') for all $k_1' \in \mathbb{F}_2^8$. Consequently, when $\mathcal{W}_1(E) = \{0\}$, all the correlation scores for those 511 key guesses will be equal to zero.

No other key guess is expected to get a null correlation score since, as stated previously, it happens with probability $2^{-58.6}$ for a random guess. Then, when $\mathcal{W}_1(E) = \{0\}$, the correct key is revealed by the following procedure:

1. Gather in a set Z the hypotheses that get a null correlation score; there are at least 511 of them,

2. For all $z \in Z$, compute z mod 256; the value that appears the most frequently – at least 256 times – is the correct value of one key byte,

3. For all $z \in Z$, compute $\lfloor z/256 \rfloor$; the value that appears the most frequently – at least 256 times – is the correct value of the other key byte.

On Encodings with a Non-Null Walsh Spectrum

Since the implementation can be broken if the encodings have been selected such that $\mathcal{W}_1(E) = \{0\}$, one could be interested in what happens when encodings with small absolute values in their Walsh spectrum are preferred. Let $E : \mathbb{F}_2^8 \to \mathbb{F}_2^8$ be a random

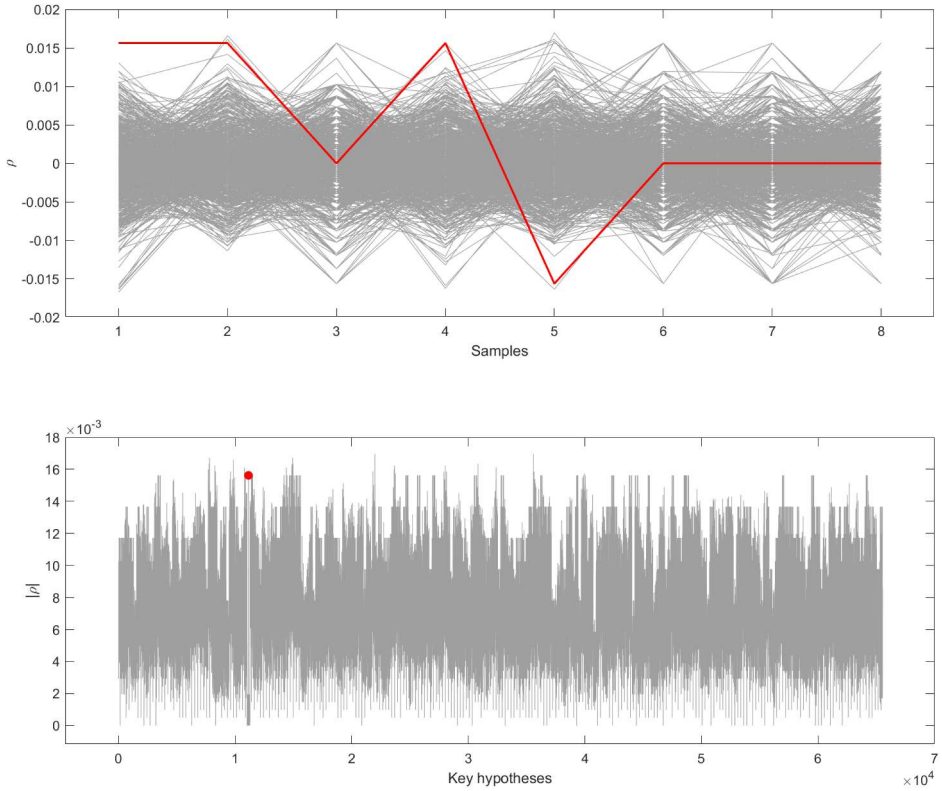


Figure 3.13: DCA targeting f_0. Here, the Walsh spectrum of the 8-bit encoding is $\{0, 4, -4\}$. Top: correlation traces for wrong (resp. good) key guesses in grey (resp. red). Bottom: highest absolute value of the correlation scores for each key guess. The good hypothesis is highlighted in red and is ranked 34^{th}.

encoding with $\mathcal{W}_1(E) \neq \{0\}$. According to Proposition 1, any non-zero value in $\mathcal{W}_1(E)$ is a multiple of 4. This implies through Proposition 3 that $|\rho^*|$ is a multiple of $1/64$. In addition, (3.3) implies that under an incorrect key guess, $\mathbb{P}(|\rho^\times| > 1/64) \approx 2^{-10.9}$. This means that even for the lowest non-zero possible absolute value of $\mathcal{W}_1(E)$, only about thirty key guesses in average show higher correlation scores than the correct one, as can be observed in Figure 3.13. Hence, the attacker could successfully recover the entire AES key by brute-force (about $2^{(16-10.9)\cdot 8} \approx 2^{40}$ remaining keys to test). Thus, even if in such a case DCA is considered by Rivain and Wang as having failed since the correct guess does not get the best correlation score, E should be regarded as insecure as soon as $\mathcal{W}_1(E)$ contains a non-zero value.

Furthermore, note that (3.10) also has an effect that defeats any value of $\mathcal{W}_1(E)$ more efficiently. One can in fact exhaustively verify that the sum over x_0 can take only ten values: 0, 256, ± 8, ± 16, ± 24, ± 32, with 256 appearing only for $k_0' = k_0$. Therefore, the distribution of the correlation scores when one of the guessed bytes is correct shows a specific behaviour that can be detected by an attacker and used to identify the correct byte value:

1. After the DCA has been performed, build 256 groups of 256 correlation traces, each group gathering the traces for which one byte of the key hypothesis is constant,

2. If, in a group, some samples from the correlation traces take exactly ten different values, then consider the corresponding constant key hypothesis as the correct one.

Figure 3.14 exhibits the particular distribution of the correlation scores when both key bytes are incorrect and when one of the two bytes is correct.

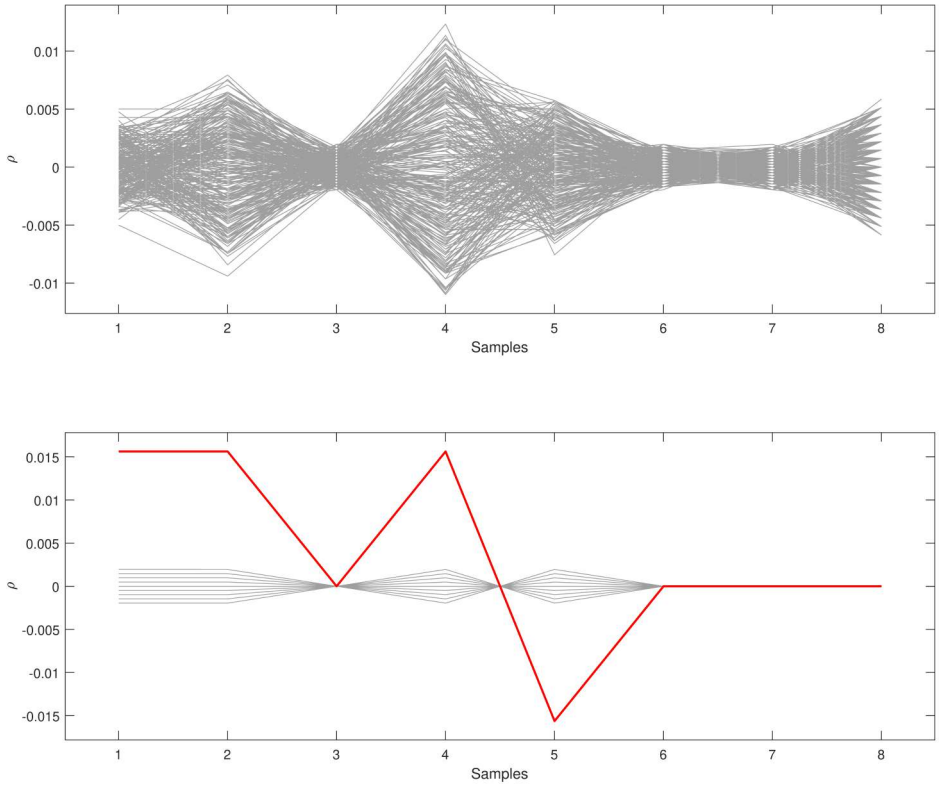


Figure 3.14: Top: correlation traces for random key guesses. Bottom: correlation traces when one of the two key guess bytes is correct. The curve highlighted in red corresponds to the two correct key bytes.

In the end, we have seen that no matter the 8-bit encodings that are selected to protect the MixColumns output, the resulting implementation can always be broken by DCA. We showed that carefully crafting encodings is as pointless as drawing them at random. Therefore, this work does rule out the existence of a particular class of encodings that could protect AES against side-channel attacks. It also highlights the challenge of establishing a general result for other cryptosystems. Indeed, while there exist encodings that prevent DCA under some conditions – met for example by the AES first round's S-box output, our attack on the MixColumns output shows that when these conditions are not satisfied, the effect of the encodings heavily depends on the operation they are supposed to protect. An interesting future work would thus be to analyze other cryptographic algorithms. Specifically, finding – or building – a cryptosystem exclusively employing operations that can be provably secured against side-channel attacks through the application of encodings would be a significant breakthrough in white-box cryptography.

3.7 Conclusion

For more than 20 years, AES white-boxes have been extensively studied. Nevertheless, many questions are still open, starting with the (im)possibility of achieving resistance against key extraction in the theoretical white-box model. Indeed, while many designs were proposed, they were all broken in the subsequent years. In practice, this does not prevent the industry to develop and sell proprietary white-box solutions for specific use-cases where potential attackers are not as powerful as in the theoretical model. For instance, if the implementation is bound to a secure application with authentication methods, the adversaries should not be able to execute the white-box themselves, which prevents them from choosing the plaintexts or performing an attack that requires a high number of executions. In this context, the main objective of the white-box designer is to thwart automated attacks and force the adversaries to go through a time-consuming reverse engineering phase to remove code obfuscation. However, even achieving this relaxed security goal is a challenge if one needs to keep the input/output behavior of the algorithm unchanged. Indeed, the traditional countermeasures employed against the automated grey-box attacks pose some new problems in the white-box context. For some time, encodings were believed to be a good alternative to prevent side-channel attacks but Rivain and Wang [RW19] showed in 2019 that small random encodings can actually be broken with very high probability by DCA when targeting the first round's MixColumns output.

In this chapter, we first went through a state-of-the-art on AES white-boxes. We discussed both designs and attacks, highlighting the advantages and drawbacks of each technique in practice. We presented a new and very efficient attack based on the square distinguisher for round-reduced AES that conveniently requires a very small number of executions. We also studied side-channel attacks and in particular the possibility of using a specific class of encodings as a countermeasure. We were able to show that carefully crafting encodings with small Walsh transforms instead of drawing them at random actually allows to prevent a classical DCA targeting the MixColumns output. However, we also illustrated how to adapt the attack and exploit the correlation traces in order to defeat these specific bijections, showing along the way that encodings cannot provide AES with efficient protection against side-channel attacks.

An interesting direction for future works could be to design a secure AES white-box for a specific real-life use-case with a well-defined weakened attacker.

Chapter 4

The Elliptic Curve Digital Signature Algorithm in the White-Box Model.

Contents

4.1 Introduction

While white-box implementations of block ciphers such as AES have been thoroughly studied, the literature on asymmetric white-boxes is still very scarce. However, there is a growing need for white-box implementations of asymmetric cryptosystems, especially the Elliptic Curve Digital Signature Algorithm (ECDSA) [DSS23] since it is used in many protocols, as for instance in major cryptocurrencies such as Bitcoin and Ethereum. Most

WBC solutions available on the market [SAS,Tha,Qua,Ird,Int,Inc,Ver,Dig] indeed propose this algorithm as a service. This is in contradiction with the lack of papers on the subject especially since the techniques employed in symmetric white-box cryptography cannot be directly applied. The first ECDSA white-box implementation [ZBJ20] was only published in 2020 and operates in a specific context only, where a semi-honest server is accessible during each signature computation. In 2021, the CHES WhibOx contest was held to encourage practical research, inviting developers to submit ECDSA white-box implementations and attackers to break the corresponding submissions. In the end, all of the 97 candidates were broken within 35 hours, suggesting the difficulty of protecting this scheme against white-box attackers. Still, the contest lead to two new publications [BBD+22, BDG+22] discussing both attacks and countermeasures.

Analyzing what happened during the 2021 WhibOx contest gives a good idea of the challenges that one has to overcome to secure ECDSA in the white-box model. This is thus how we start this chapter. At the time of the contest, we participated in the team of attackers *TheRealIdefix* and managed to break the highest number of submissions. We thus detail here all the attacks that we used and discuss their efficiency on the various candidates. We also give an overview of the design of the two best implementations.

In the subsequent part of this chapter, we study the most interesting countermeasures that we have found in patents. Indeed, as mentioned above, no public implementation of ECDSA is secured in the white-box context, but several solutions are actually sold by the industry. Their documentation provides no detail about the used techniques, so the only option left for us to gain some information on the actual design of these products was to look for patents on this subject. Of course, a patent is not comparable to a published article: some important details might be omitted and all techniques to reach overall security may not be found in the same patent. Also, the exact context and security model are usually not explained, which complicates the evaluation of the suggested countermeasures. However, these documents still give valuable information on methods implemented in the field.

Finally, in the last part of this chapter, we analyze the security of Zhou et al.'s cloud-based implementation [ZBJ20], which is the very first ECDSA white-box design to be published. We exhibit two different and very efficient fault attacks based on a common principle to re-inject some specific values from one signature execution to another. We also suggest a countermeasure that prevents both these attacks without impacting the size of the white-box.

4.2 Preliminaries

4.2.1 Elliptic Curve Cryptography

An elliptic curve over a finite field K of characteristic different from 2 and 3 is defined by an equation of the form

$$y^2 = x^3 + ax + b$$

with $a, b \in K$ such that $4a^3 + 27b^2 \neq 0$. Note that for a field with characteristic 2 or 3, the equation is somewhat more complicated.

Let $E(K)$ denote the set of curve points to which is added the *point at infinity* $\mathcal{O}$. It has an abelian group structure where:

- $\mathcal{O}$ is the neutral element.

- The opposite of a point $P = (x, y)$ is the point $-P = (x, -y)$.
- If $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ are two points such that $P \neq Q \neq \mathcal{O}$, then the point $R = P + Q$ is given by:

$$\begin{cases} x_R = \alpha^2 - x_P - x_Q \\ y_R = y_P + \alpha(x_R - x_P) \end{cases}$$

  with $\alpha = \frac{y_Q - y_P}{x_Q - x_P}$.
- If $P = (x_P, y_P)$ is such that $P \neq \mathcal{O}$ and $y_P \neq 0$, then the point $R = P + P$ is given by:

$$\begin{cases} x_R = \alpha^2 - 2x_P \\ y_R = y_P + \alpha(x_R - x_P) \end{cases}$$

  with $\alpha = \frac{3x_P^2 + a}{2y_p}$.

The operation consisting in adding a point P to itself $k \in \mathbb{N}^*$ times is called scalar multiplication and denoted as $[k]P$. It can naturally be extended to $k \in \mathbb{Z}$ by defining $[0]P = \mathcal{O}$ and $[-k]P = [k](-P)$. Scalar multiplication is a key operation for many cryptosystems based on elliptic curves. It is therefore important to find both efficient and secure algorithms, see for instance [Ver12, Chu17] for a survey.

Elliptic curve cryptography is based on the difficulty of the Discrete Logarithm Problem (DLP). Indeed, while the latter was first described in the multiplicative group of a finite field, it can be adapted for the group of points of an elliptic curve. If we consider a multiplicative group G, the problem is the following: given $g \in G$ and h an element of the subgroup of order n generated by g, find the integer x such that $0 \leq x < n$ and $h = g^x$. Similarly, the elliptic curve discrete logarithm problem consists in finding k given the points P and $[k]P$. Conveniently, the DLP seems to be harder in this scenario than in the multiplicative group of a finite field, allowing the use of smaller keys for the same security level.

4.2.2 Description of ECDSA

The *Elliptic Curve Digital Signature Algorithm* (ECDSA) [DSS23] was introduced as a variant of DSA by Vanstone in 1992. Its parameters are an elliptic curve E over $\mathbb{F}_q$ with a generator G of order n and a cryptographic hash function H. The private key d is randomly generated in $\mathbb{F}_n^*$ and the public key Q is the point resulting of the scalar multiplication between d and G, that is $Q = [d]G$. The ECDSA signature operation is described in Algorithm 4, where x_R and y_R denote the affine coordinates of the point R. The verification is as shown in Algorithm 5. Note that to prevent some attacks, the parameters of the scheme should also be checked for consistency before validating a signature.

The sensitive values manipulated in the signature algorithm are not only the secret key d, but also the nonce k since its recovery allows the computation of d from the signature (r, s) and the hash e as follows:

$$d = (ks - H(m))r^{-1} \bmod n. \tag{4.1}$$

In the black-box context, the security of ECDSA relies on the difficulty of the Elliptic Curve Discrete Logarithm Problem (ECDLP), that is on the difficulty of recovering d

Algorithm 4: ECDSA signature operation

Input : the message m
Output: the signature (r, s)

$e \leftarrow H(m)$
$k \xleftarrow{\$} [\![1, n-1]\!]$
$(x_R, y_R) \leftarrow [k]G$
$r \leftarrow x_R \bmod n$
$s \leftarrow k^{-1}(e + rd) \bmod n$
if $r = 0$ *or* $s = 0$ **:**
    Go to step 2
Return $(r,\ s)$

Algorithm 5: ECDSA verification

Input : the message m and a signature (r, s)
Output: VALID or INVALID

$e \leftarrow H(m)$
$u_1 \leftarrow es^{-1} \bmod n$
$u_2 \leftarrow rs^{-1} \bmod n$
$(x_R, y_R) \leftarrow [u_1]P + [u_2]Q$
if $x_R \equiv r \bmod n$ **:**
    Return VALID
Return INVALID

from $Q = [d]G$ or k from $R = [k]G$, and on the quality of the used randomness. Indeed, the nonce must not only remain secret but also differ at each execution of the algorithm since its value can easily be recovered from two signatures (r, s) and (r', s') of different hashed messages $e' \neq e$ using the same nonce, that is with $k' = k$. In that case, we also have $r' = r$, so the adversary may compute

$$k = (e' - e)(s' - s)^{-1} \bmod n \ . \tag{4.2}$$

Furthermore, even smaller biases in the nonce generation can be exploited via lattice-based attacks.

4.2.3 Lattice-Based Cryptography

In the remainder of this chapter, we will discuss lattice-based attacks that can be employed to recover an ECDSA secret key from signatures generated with biased nonces. To provide context for these discussions, we will first quickly review the fundamentals of lattice-based cryptography.

Geometrically, a lattice is a discrete grid of points in an n-dimensional space. A full rank lattice is typically specified by a basis B which is a $n \times n$ matrix with linearly independent row vectors and real entries. The lattice generated by B is the set of all integer linear combinations of its row vectors. It is important to note that a lattice can be generated by an infinite number of different bases.

Lattices can be used in cryptography, either to construct cryptographic schemes [BDK+18, DKL+18] or to attack others, such as RSA and (EC)DSA [DH20]. There are several lat-

tice problems, one of the most famous being the Shortest Vector Problem (SVP), which involves finding the vector with the smallest non-zero norm in a lattice given one of its bases. These problems are known to be in NP for arbitrary lattices [Ajt96, AR04] and remain hard to solve even with quantum computers, contributing to the popularity of lattice-based cryptosystems as a potential post-quantum cryptographic solution. Nevertheless, there exist algorithms such as LLL [LLL82, NS09] or BKZ [Sch87, SE94] that can be used to find approximations within a reasonable time. Lattice-based attacks, as the ones we will describe in Section 4.3.2, generally involve constructing a lattice that contains a very short vector with sensitive entries and attempting to recover it using basis reduction algorithms.

4.3 WhibOx 2021

The WhibOx contest, attached to the CHES conference, has been held three times since 2017 to encourage practical research in white-box cryptography from both the designer's and the attacker's perspectives. Each time, during several months, coders are invited to post white-box implementations and attackers to try to break them. Participants can remain anonymous and silent about any detail on their work. The first two editions in 2017 and 2019 focused on white-box implementations of AES and exhibited the community's strong interest in this subject. Some candidates survived all attacks in the second edition in 2019, showing a certain maturity for this algorithm. In 2021, organizers changed the target and decided to consider the ECDSA signature algorithm, whose white-box implementation is of substantial interest to the industry but virtually lacks scientific literature. From May 17$^{\text{th}}$ to August 22$^{\text{nd}}$ 2021, 97 candidate implementations were submitted for scrutiny by 37 (teams of) attackers. All challenges were broken within 35 hours, suggesting the difficulty of achieving a secure white-box implementation of ECDSA.

In this section, we present the attacks that we used to break the various challenges during the contest as the team *TheRealIdefix*. We consider different attack paths against ECDSA white-boxes: the ones inherited from traditional cryptanalysis, the extensions of grey-box attacks and the ones that are specific to the white-box context. We discuss the feasibility of automating each of them and provide detailed information regarding which attacks succeeded (or failed) on each candidate. We also give an overview of the winning challenge that was submitted by the team of designers *zerokey*. Although it was finally broken during the contest as all others, the employed techniques are interesting and worth mentioning.

The results presented in this section were published in a joint paper from the participants of the teams *TheRealIdefix* and *zerokey* [BBD$^+$22].

4.3.1 Contest Rules

During the 2021 WhibOx contest, designers were required to post challenges computing ECDSA signatures on the NIST P-256 curve under a hard-coded, freely chosen key, and accepting as input any 256-bit message digest $e = H(m)$. Notice that the cryptographic hash function H was excluded from the intended white-box implementation of ECDSA and the message m was also not provided. Acceptance of submitted implementations was conditioned on some requirements:

- the public key corresponding to the embedded private key, as well as a proof of knowledge of the private key, had to be provided,

- submissions had to be source code in portable C,
- linking to external libraries was forbidden, except for the GNU Multi Precision library [Gt20],
- the execution time was limited to 3 seconds, the program size to 20 MB, and the RAM usage to 20 MB as well.

There was an elaborate system with scoreboards to reward designers and attackers who managed to extract the private keys of the submissions. A challenge gained strawberries as time goes by till broken. Challenges with a higher performance score (measured in terms of execution time, code size, and RAM usage) gained strawberries faster. Eventually, the challenge with the highest number of strawberries won the competition. Accordingly, when submitting a matching private key to the system, attackers received bananas, the number of which was determined by the number of strawberries of the challenge at the time of the break. More detailed information can be found on the contest website [Whi21].

4.3.2 Automated Attacks on ECDSA White-Boxes

White-box implementations usually rely on encodings and other approaches to protect the secret values and their manipulations. Furthermore, code obfuscation techniques are often used to make the understanding of the design a time-consuming and challenging task. We thus focused on designing efficient attack methods that do not require any earlier reverse engineering step and can easily be automated.

Searching For Unprotected Sensitive Values

The most basic strategy to break white-box implementations is to look for sensitive values that are mistakenly manipulated in plain, meaning unprotected, at some time during the execution. Since the contest rules were a clear incentive for developers to use the GMP library for big number arithmetic operations, a first attempt to break the submitted challenges was then to search if sensitive values were manipulated by the GMP library. In order to do this automatically, our approach has been to hook the calls to GMP functions thanks to the so-called *`LD_PRELOAD` trick*.

Pre-loading is a feature of the dynamic linker on UNIX systems that allows loading a specific shared library before all other libraries linked to a given executable binary[1]. In our specific case, we built a shared library defining the same function as the GMP library (e.g. `mpz_mul`, `mpz_mod` or `mpz_invert`). Each of these functions simply updates a log of the given parameters before calling the real GMP function, explicitly using the dynamic linker (thanks to the `<dlfcn.h>` module) to ensure the correct execution of the white-box implementation. It is then only necessary to add our shared library to the `LD_PRELOAD` environment variable of the dynamic linker on our system before calling the ECDSA binary to have our custom functions called in place of the genuine GMP ones. The corresponding log can then be analyzed in a second step to eventually reveal the secret key if d, k or related values such as rd or $e + rd$ are found in the log. Such an approach allowed us to break 32% of the challenges.

[1] `https://man7.org/linux/man-pages/man8/ld.so.8.html`

Biased Nonces

As discussed in Section 2.1.5, white-box designers often use a PRNG applied to the algorithm's input to generate random values. This is because the adversary controls the execution environment and can manipulate the output of common sources of randomness. Despite this, some designers attempted to introduce non-determinism into their implementations during the contest. Most of these challenges used calls to the time() function from the C standard library, which could easily be replaced with a constant. Others appeared to use addresses or the values of uninitialized variables as entropy sources. For these challenges, disabling Address Space Layout Randomization (ASLR) was sufficient to obtain the same result for each execution with the same input.

Let us now assume that all the potential sources of entropy, but the input of the white-box, have been fixed. The nonce k is then computed as a function of the hash, i.e. $k = f(e)$. If the function f is not carefully selected, it could happen that the k_i's generated from different e_i's are not uniformly random. In the worst case, we have collisions such that different hash values e_0 and e_1 produce the same nonce ($k_0 = k_1$). If such a collision occurs, one can recover the nonce with equation (4.2) and then the private key with (4.1). Furthermore, this can be efficiently detected by looking at the r part of the signature. To efficiently browse a subset of hash values in search of collisions, we limited ourselves to hash values with a Hamming weight equal to 1 or 2. We thus considered 32 896 hash values and were able to break 60% of the challenges with this technique.

In those cases where we did not find any collision, we looked for smaller biases in the nonce generation and used well-known lattice-based attacks derived from [NS02] and [FGR13]. These attacks may allow one to recover an ECDSA private key with the knowledge of only a few bits of the nonces for several signatures. Basically, the idea is to construct a lattice containing a short vector composed of sensitive values and to recover it via a basis reduction algorithm. In our case, we computed (r_i, s_i) with $s_i = k_i^{-1}(e_i + r_i d) \bmod n$ for $0 \leq i < 1000$ and $e_i = i$. We selected some subsets of m signatures to build our lattices and separately performed the attack on all of them.

Let us first assume that in a subset of m signatures, the l most-significant bits of the nonces are all at zero. Let $K = 2^{256-l}$, we have $k_i < K$ for $0 \leq i < m$. Note that in our experiments, we often chose $m = 70$ and $l = 6$. For any signature (r_i, s_i), one can write

$$k_i = s_i^{-1} s_m r_i r_m^{-1} k_m - s_i^{-1} r_i e_m r_m^{-1} + s_i^{-1} e_i \bmod n \ .$$

If we denote $t_i = s_i^{-1} s_m r_i r_m^{-1}$ and $u_i = -s_i^{-1} r_i e_m r_m^{-1} + s_i^{-1} e_i$, then the above equation can be written as

$$k_i = t_i k_m + u_i \bmod n \ .$$

We thus constructed the following lattice basis:

$$B = \begin{bmatrix} n & & & & & \\ & n & & & & \\ & & \ddots & & & \\ & & & n & & \\ t_1 & t_2 & \cdots & t_{m-1} & 1 & \\ u_1 & u_2 & \cdots & u_{m-1} & 0 & K \end{bmatrix} .$$

The vector $(k_1, k_2, \ldots, k_m, K)$ is in this lattice by construction and it is expected to be one of the shortest ones since $k_i < 2^{256-l}$ and its norm is smaller than the approximated

norm of the shortest vector for a random lattice that can be computed using the Gaussian heuristic [Ajt96]. Running algorithms such as LLL or BKZ on this lattice may thus allow an attacker to recover the nonces and compute the private key from equation (4.1).

Note that the attack can be adapted for other values and other positions of the known bits (see for example [DH20]). In particular, we also tried it with all the bits set to 1 and with the knowledge of the least-significant bits instead of the most-significant ones. Finally, Lattice-based attacks can also be applied when the nonce k is the product of a small random κ by another (large) constant scalar t. This is an interesting choice from an implementation point of view since taking $k = \kappa t$ allows to perform the scalar multiplication at a reduced cost as $R = [\kappa]T = [k]G$ with $T = [t]G$ a precomputed value and without directly manipulating the nonce.

All in all, our attacks exploiting biases in the nonce generation allowed us to break 72% of the challenges.

Side-Channel Attacks

The most well-known side-channel attack against ECDSA implementations is probably the Simple Power Analysis (SPA) targeting the nonce [Cor99]. It consists in the analysis of a single trace obtained during the scalar multiplication and is made possible by the use of non-regular algorithms as the *double-and-add* (see Algorithm 6) which is the elliptic curve equivalent of the *square-and-multiply* presented in Section 2.1. Here, if the adversary is able to distinguish the doublings from the additions, then he may recover the nonce bit by bit and use it to compute the secret key as shown in equation (4.1).

Algorithm 6: Scalar multiplication – the *double-and-add* algorithm

Input : a scalar $k = (k_0, k_1, \ldots, k_\ell)_2$ and a point P
Output: the point kP

$Q \leftarrow \mathcal{O}$
for $i \in [\![0, \ell]\!]$ **:**
    $Q \leftarrow [2]Q$
    if $k_i = 1$ **:**
        $Q \leftarrow Q + P$
Return Q

Another possibility to break an ECDSA implementation is to make assumptions on the computation of rd and try to attack this value by analyzing multiple traces. For instance, if the multiplication is performed on 8-bit words, the attacker could guess a byte of d and multiply it with a byte of r. He could then use the procedure described in Section 2.1 and use a score function on the hypothesized sensitive values and the trace points in order to distinguish the good key hypothesis. Note that there exists a variant of the attack that targets the addition between e and rd rather than the multiplication.

In theory, side-channel attacks are particularly devastating since they can be fully automated and do not require any earlier reverse engineering step. In practice, they are quite difficult to apply in the white-box context because of the size of the traces, in particular for cryptosystems such as ECDSA that have a relatively long execution time. Indeed, if the whole white-box execution were to be recorded, each trace would easily reach several gigabytes. Iterating over dozens of traces for the attack would thus be overwhelming in time and memory. A time-consuming step of reverse engineering allowing

to select a smaller window of the implementation before the attack is then required, which is why we did not use this technique to break the challenges of the WhibOx contest.

Fault Injections

Several fault attacks can be performed on ECDSA in the grey-box context, and they are obviously also a potential threat in the white-box context. For example, the attacker can force the use of a weak elliptic curve during the scalar multiplication by disturbing the curve parameters [BMM00] in order to solve the DLP easily. Alternatively, he can force the use of biased nonces, for instance by sticking a word of k at zero during several executions. The corresponding signatures can then be used to obtain information on the key using lattice-based algorithms. Finally, modifying one byte of d during the computation of rd may also allow one to recover information on the key, as shown in [GK04].

In addition, the white-box model offers new possibilities [PSS+17, ABF+18, DGH21] arising from the fact the scheme is, or can be made, deterministic. When the algorithm is used twice on the same message, the same nonce k is derived. The attacker may thus obtain a correct signature for a given hash e, and an erroneous one by modifying a second execution on the same input. To break the challenges of the WhibOx contest, we mainly disturbed the computation of the first part of the signature r, obtaining faulty results $\tilde{r}$ and $\tilde{s} = k^{-1}(e + \tilde{r}d) \bmod n$. This allowed us to create and solve the following system in k and d:

$$\begin{cases} s = k^{-1}(e + rd) \\ \tilde{s} = k^{-1}(e + \tilde{r}d) \end{cases}.$$

It is also possible to disturb other values, as the hash e after the computation of the nonce. However, one would need to know the faulty value $\tilde{e}$ to solve the obtained system

$$\begin{cases} s = k^{-1}(e + rd) \\ \tilde{s} = k^{-1}(\tilde{e} + rd) \end{cases}.$$

If the attacker can manage to induce a precise fault on say, a byte of the targeted value, then he may make a hypothesis on $\tilde{e}$ and perform a brute-force attack. Interestingly, we do not have this problem with the attack on r since the faulty value is just given to the attacker as part of the output. Furthermore, the attack surface to disturb r is huge: the fault may happen anywhere during the scalar multiplication. This is why we only altered this variable in the context of the WhibOx competition. It indeed proved to be an efficient attack since it allowed us to break 75% of the challenges.

Note that after the contest, two other teams called bananaramadama and auguste also joined to publish a paper on their attacks and countermeasures [BDG+22]. In particular, they used other more sophisticated fault attacks targeting different variables. These can only be performed if the attacker manages to obtain the same faulty value twice in two different executions on two different hash values, which is difficult but not impossible in the white-box context where the adversary can directly modify the binary and the faults are easily reproducible. For completeness, we briefly present these attacks in the following.

The first attack is called the collision fault attack. Its advantage compared to the simple one described above is that it can overcome some countermeasures. In particular, it may allow the attacker to recover the secret key without any brute force even if the faulty value is not known. For instance, suppose that the same faulty value $\tilde{e}$ can be

induced in two executions. This could be realized for example by skipping the addition of e during the computation of s (special case $\tilde{e} = 0$). The attacker could then obtain

$$\begin{cases} s_1 = k_1^{-1}(e_1 + r_1 d) \\ s_2 = k_2^{-1}(e_2 + r_2 d) \\ \tilde{s_1} = k_1^{-1}(\tilde{e} + r_1 d) \\ \tilde{s_2} = k_2^{-1}(\tilde{e} + r_2 d) \end{cases}$$

and solve this system of 4 equations in k_1, k_2, d and $\tilde{e}$. Note that a similar key recovery can be achieved with a fault induced on r, rd, $e + rd$, k, k^{-1} or d.

The second attack is called the differential collision fault attack. It is a variant of the first one, where the faulty value is the original one plus a constant. In other words, if the fault is induced on e, it is assumed that $\tilde{e} = e + cst$. One can thus obtain and solve the following system in k_1, k_2, d and cst to retrieve the secret key:

$$\begin{cases} s_1 = k_1^{-1}(e_1 + r_1 d) \\ s_2 = k_2^{-1}(e_2 + r_2 d) \\ \tilde{s_1} = k_1^{-1}(e_1 + r_1 d + cst) \\ \tilde{s_2} = k_2^{-1}(e_2 + r_2 d + cst) \end{cases} .$$

Once again, this attack can be adapted to faults induced on r, rd, $e + rd$, k, k^{-1} or d.

Attacks Results

We give in Appendix B the specific vulnerabilities of each of the 97 submitted challenges as well as the corresponding private keys. The success rates of each attack methods are presented in Table 4.1. We observe that lattice and fault attacks are very efficient. Collision attacks also give good results.

Table 4.1: Success rate of each attack on the 97 challenges.

Attack type	Percentage of broken challenges
Hooking	32%
Bad nonce	
- Collision	60%
- Lattice	72%
Fault Injection	75%

However, we noticed that many challenges had a low level of security, some were even plain implementations. In order to focus on the *strongest* candidates, we thus excluded 30 challenges[2] where the nonce and/or the private key were manipulated in plain. Table 4.2 illustrates the efficiency of the attacks presented in Section 4.3.2 on the remaining 67 challenges. We observe that hooking gives no significant result anymore, collision and lattice attacks become less efficient, and fault injection is still the most powerful attack.

[2]The challenges 3, 4, 8, 10, 11, 32, 45, 54, 55, 57, 85, 97, 114, 135, 136, 139, 153, 157, 174, 185, 187, 231, 235, 267, 274, 299, 307, 320, 321, and 323 are considered as weak.

Table 4.2: Success rate of each attack on the 67 strongest challenges.

Attack type	Percentage of broken challenges
Hooking	1%
Bad nonce	
- Collision	49%
- Lattice	61%
Fault Injection	69%

Among the 67 strongest challenges, Challenges 226 and 227 are the winning ones. In the next section, we give an overview of their design.

4.3.3 Best Candidates

Design overview

The two winning challenges of the WhibOx contest were both submitted by *zerokey* and implemented with the same methodology. They only differ in some additional countermeasures. Challenge 227, the one that obtained the highest number of strawberries, is the lightweight variant. Challenge 226, the one that acheived second place in the contest and stood unbroken for the longest time, is the hardened but heavier variant. Their design was inspired from the white-box implicit framework of [RVP22].

Definition 4. *Let $F : \mathbb{F}_q^l \to \mathbb{F}_q^{l'}$. A function $T : \mathbb{F}_q^{l+l'} \to \mathbb{F}_q^{l''}$ is called an* implicit function *of F if for all $u_1, u_2, \ldots, u_l$ and $v_1, v_2, \ldots, v_{l'}$ in $\mathbb{F}_q$, it satisfies*

$$T(u_1, u_2, \ldots, u_l, v_1, v_2, \ldots, v_{l'}) = 0 \Leftrightarrow F(u_1, u_2, \ldots, u_l) = (v_1, v_2, \ldots, v_{l'}) \ .$$

In that case, T is said to be quasilinear if for any $(u_1, u_2, \ldots, u_l) \in \mathbb{F}_q^l$, the function $(v_1, v_2, \ldots, v_{l'}) \mapsto T(u_1, u_2, \ldots, u_l, v_1, v_2, \ldots, v_{l'})$ is affine over $\mathbb{F}_q$.

The idea is to represent the operations of the scheme by quasilinear implicit functions and to protect them with large affine bijections acting as encodings. The quasilinear property allows the implicit evaluation of a function F in a point $(u_1, u_2, \ldots, u_l)$ by solving the affine system $T(u_1, u_2, \ldots, u_l, v_1, v_2, \ldots, v_{l'}) = 0$ for the variables $v_1, v_2, \ldots, v_{l'}$. Furthermore, the representation of the implicit functions as systems of multivariate polynomials allows applying countermeasures from multivariate public-key cryptosystems. For instance, in Challenge 226, additional variables and components preserving the input-output behavior of the underlying encoded functions are added in the equations. Another layer of obfuscation consists in multiplying each term with some random polynomials in the input variables. Interestingly, this operation preserves the quasilinear property and does not change the solution set. We refer the reader to [BBD+22] for more information on how to obtain the implicit implementation in the specific case of ECDSA.

Attacks

Note that these challenges were specifically built for the WhibOx contest, where the attackers did not know the design details. Once fully reverse-engineered, they are easy to

break. Nevertheless, reverse-engineering is not mandatory as challenges 226 and 227 were both broken by automated attacks during the contest. Indeed, we broke Challenge 227 with hooking techniques[3] during the contest and lattice-based and fault attacks afterwards. Challenge 226 resisted our automated attack tools at first, but was then proven sensitive to fault attacks [BDG+22].

In the end, no candidate white-box resisted all the automated attacks, and no challenge survived more than two days, showing that securing ECDSA in the white-box model really is a challenging problem. Nevertheless, as mentioned earlier, some companies already sell ECDSA white-boxes. In the next section, we will explore their patents and discuss some of the employed countermeasures.

4.4 Patent Overview

In the last decade, several companies registered patents exposing different techniques to protect white-box ECDSA implementations. Of course, a patent is not comparable to a published article. It may omit crucial details and often lacks explicit explanations of the context and security models, complicating the assessment of the proposed methods. Furthermore, all the techniques used for achieving overall security might not be disclosed within a single patent. Despite these limitations, patents remain valuable sources of information on implemented methods in the field. After a careful research, we found a dozen of such documents and studied them in detail before selecting the six most interesting ones in our context [Boc20, CP18, GV20, SEMM15, RH18, SPC18].

In this section, we expose the main ideas to counteract the different attacks discussed in Section 4.3.2. We distinguish the countermeasures that are meant to prevent active attacks, where the adversary actively alters the execution of the white-box, from the ones that tackle passive attacks as SCA. This survey was published in collaboration with Emmanuelle Dottax and Christophe Giraud in [DGH21].

4.4.1 Countermeasures Against Passive Attacks

As discussed previously, the traditional protection against passive attacks in the white-box model is the use of look-up tables protected by random encodings. Since the needed memory space increases exponentially with the number of input bits of the tables, the implementation is usually broken into a network of small tables acting on a restricted number of bits, typically 4 or 8. Nevertheless, this technique is not well suited for asymmetric white-box cryptography that usually relies on complex operations on a large number of bits such as the scalar multiplication. This explains why, in the patents that we found, encodings and look-up tables are often replaced by masking techniques or homomorphic encryption. In the following, we show how these techniques are used to protect ECDSA white-box implementations and we expose their advantages and drawbacks.

Masking

The two main use-cases for masking techniques are the scalar multiplication and the inversion of k since they are very complex operations.

[3]The nonce and the key did not appear in the log, but we found another sensitive variable, that is the nonce multiplied by a constant. We will see how this leads to the recovery of the key in Section 4.4.1

For instance in [SEMM15], the base point G and the secret key d are both multiplicatively masked by a random value t selected offline. The point $T = [t^{-1}]G$ is used for the scalar multiplication instead of G, which implies that the nonce is in fact $\kappa = kt^{-1}$ rather than k. The modified key $\delta = dt \bmod n$ is used instead of d to maintain the consistency of the signature. The computation of s is then performed as shown in Algorithm 7.

Algorithm 7: ECDSA signature [SEMM15]

Input : the message m
Output: the signature (r, s)

`// Precomputations:`

1 $t \xleftarrow{\$} [\![1, n-1]\!]$
2 $T \leftarrow [t^{-1}]G$
3 $\delta \leftarrow dt$

`// Signature:`

4 $e \leftarrow H(m)$
5 $k \xleftarrow{\$} [\![1, n-1]\!]$
6 $(x_R, y_R) \leftarrow [k]T$
7 $r \leftarrow x_R \bmod n$
8 $u \leftarrow te + \delta r \bmod n$
9 $s \leftarrow k^{-1}u \bmod n = (\kappa)^{-1}(e + rd) \bmod n$ with $\kappa = kt^{-1} \bmod n$
10 **if** $r = 0$ *or* $s = 0$ **:**
11 Go to step 5
12 Return (r, s)

The point is that an adversary searching for the nonce or the secret key in the memory would not find these values anymore. Nevertheless, notice that since both δ and k are manipulated in plain, finding t may lead the attacker to a trivial key recovery. The authors of the patent thus propose to protect t by computing step 8 of the algorithm using a network of look-up tables, as usually done for symmetric white-box cryptography. Nevertheless, even if the adversary is unable to recover t during that step, he may still compute it together with the secret key by solving the following system of equations

$$\begin{cases} k_1 s_1 = t(e_1 + r_1 d) \\ k_2 s_2 = t(e_2 + r_2 d) \end{cases}$$

with (r_1, s_1) and (r_2, s_2) two different signatures. This method can add some complexity to the attacker's task but does not prevent key recovery. Therefore, it must be implemented alongside other countermeasures.

Another possibility is to use an additive mask for the scalar multiplication. In such a case, $v = k + t$ has to be computed from e in a secure way with t being a random mask that can be generated offline [SPC18] or derived from e [RH18]. The point R can then be computed as $R = [k + t]G - T$ with $T = [t]G$ potentially precomputed. However, the nonce also needs to be inverted for the computation of s and this is not easily feasible from $(k + t)^{-1}$. A common technique to overcome this problem is to compute k from e again, but this time, masking it multiplicatively [SPC18, RH18, Boc20]. The result is then inverted in plain and multiplied again by the mask to give k^{-1}. An example of such an approach is given in [SPC18] where the constant t and the corresponding point

$T = [t]G$ are selected offline together with two symmetric encryption keys sk and sk'. Let us denote by Enc_{sk} an encryption function with key sk. The signature algorithm is depicted in Algorithm 8.

Algorithm 8: ECDSA signature [SPC18]

Input : the message m
Output: the signature (r, s)

`// Precomputations:`

1 $t \xleftarrow{\$} [\![1, n-1]\!]$
2 $T \leftarrow [t]G$
3 $(sk, sk') \xleftarrow{\$} [0, 2^{64}-1]^2$

`// Signature:`

4 $e \leftarrow H(m)$
5 $v \leftarrow Enc_{sk}(e) + t = k + t$
6 $R \leftarrow [v]G - T = [k]G$
7 $r \leftarrow x_R \bmod n$
8 $w \leftarrow Enc_{sk}(e) \times Enc_{sk'}(e) = kk'$
9 $s \leftarrow Enc_{sk'}(e)w^{-1}(e + rd) = k^{-1}(e + rd)$
10 **if** $w^{-1}Enc_{sk}(e)Enc_{sk'}(e) \neq 1$ **:**
11 Fault detected, abort or return random value
12 Return (r, s)

Once again, the mask t has to be protected or the nonce, and thus the key, could be computed from v. Moreover, even the knowledge of v alone might be enough to break the scheme. Indeed, since t is fixed, the attacker could guess its highest bits and obtain the ones of k from v. If this is done for several signatures, a lattice-based attack could lead to successful key recovery as shown in Section 4.3.2.

To conclude, masking prevents an attacker from directly reading the nonce and the secret key in memory or from performing a classical side-channel attack. However, the technique often seems easy to bypass when the masks are fixed and selected offline. Making them vary randomly would be a good solution, but once again, this is difficult to achieve in the white-box context. Furthermore, one has to be careful with the implementation so that the masks cannot be recovered. In some cases, it may be difficult to get rid of them without revealing any secret information. Masking is thus often combined with encodings [SEMM15, SPC18, RH18]. Nevertheless, the latter implies the use of a large amount of memory and all the published white-box implementations relying on this technique have been broken. Some patents thus suggest another idea to protect data against passive attacks: the use of homomorphic encryption. While it seems to be the safest solution, we will see that it also has some drawbacks.

Homomorphic Encryption

Homomorphic encryption (see for instance [ABC+15]) allows to perform some computations directly on encrypted data without having to decrypt it first. The supported operations depend on the encryption scheme, it can be both addition and multiplication or only one of them.

In the white-box context, sensitive data can be encrypted offline and never appear in plain. For instance, the authors of [GV20] use a pool of pre-encrypted randoms to compute the nonce, and none of these values is ever decrypted. Scalars $(k_{i,0}, k_{i,1})$ are drawn offline for $1 \leq i \leq b$, with b being the maximum bit length of a message, and the corresponding $(R_{i,0}, R_{i,1}) = ([k_{i,0}]G, [k_{i,1}]G)$ are computed. The scalars $(k_{i,0}, k_{i,1})$ are then encrypted, resulting in couples $(c_{i,0}, c_{i,1})$ that are stored on the device together with the generated points $(R_{i,0}, R_{i,1})$. The first step of the signature is the computation of $R = [k]G$ and c, the encrypted value of the nonce. Let us denote the i-th bit of the message by m_i. The computations are as follows:

$$c = \sum_{i=1}^{b} c_{i,m_i} \text{ and } R = \sum_{i=1}^{b} R_{i,m_i} \ .$$

The way the second part of the signature is computed from c, r and e is not detailed. It is only mentioned that the encrypted value of s is obtained and not decrypted before being sent with r to the verifier, so the verification algorithm is not standard. This is the main drawback of using homomorphic encryption: one has to choose between using a decryption white-box or not respecting the standard by returning an encrypted value.

In [Boc20], the first option is chosen. Paillier's homomorphic encryption [Pai99], denoted by Enc, is used to protect sensitive data during computations and a decryption white-box allows the recovery of s in plain at the end of the algorithm. This white-box may be very costly and not easy to build. Furthermore, it induces a security issue: an attacker could apply it on encrypted sensitive variables to recover their plain values. Another protection layer should thus be added. The inventors decided to mask the variables before their encryption. For instance, the value $Enc(dt)$ is used instead of $Enc(d)$ for the computation of s, with t a random mask that will be removed later on. This way, an adversary trying to apply the decryption white-box to $Enc(dt)$ would only recover the masked value of the key. As we have seen before, this may increase the security but is probably not enough to prevent key recovery.

4.4.2 Countermeasures Against Active Attacks

We showed in Section 4.3.2 that a white-box implementation of ECDSA is particularly vulnerable to fault attacks. The usual countermeasures implemented for the grey-box model rely on redundancy and consistency checking. These techniques can also be used in the white-box context even if they are by essence memory consuming. An alternative is to link several values by encoding them together so that disturbing one of them also modifies the others with a high probability. The last known option is to precompute the sensitive variables offline and embed them inside a white-box.

Redundancy and Checks of Consistency

A first fault attack that can be performed on ECDSA white-boxes consists in forcing the use of the same nonce twice. Of course, the fault can be induced directly on k, but if the computations are deterministic, the same result can also be obtained by signing twice the same message and altering the hash e after the computation of the nonce during the second execution. An idea to reinforce the security against this attack is exposed in [SPC18] (see Algorithm 8). The nonce k is derived from e three times across the execution, in steps 5, 8 and 10, and its consistency is checked at the end of the algorithm. Hence, a fault

on k or e has to be induced several times to obtain an exploitable faulty output, which drastically complicates the attack.

Another protection presented in [SPC18] consists in storing a list of all messages that have been signed. This way, the attack on e cannot be performed since the white-box would detect that the hash has already been used. However, this protection is not efficient against an adversary having full control over the execution environment. Indeed, the latter could erase parts of the list or restore the previous state of the white-box to be able to perform his attack.

The fault on the first part of the signature presented in Section 4.3.2 seems even more difficult to prevent. Indeed, this value necessarily appears in plain if the scalar multiplication is not protected, which makes it is easy to find and alter. Moreover, r is returned by the algorithm so the attacker always knows the result of the fault he induced. In [SPC18], the inventors try to check the consistency of R without entirely computing it twice, which could be quite costly, especially if the scalar multiplication is protected against passive attacks. They suggest two techniques: the secure delegation of calculus to a third party [CDKS15] and the technique of "small moduli". The latter is not described in the patent but we assume that it is a reference to Shamir's method exposed in [Sha99, Joy19]. The idea would consist in computing $R = (x_R \bmod \rho n, y_R \bmod \rho n)$ with the coordinates in $\mathbb{Z}/\rho n\mathbb{Z}$ instead of $\mathbb{Z}/n\mathbb{Z}$ and then another time in $\mathbb{Z}/\rho\mathbb{Z}$ for ρ a small random. Subsequently, one could check the integrity of R by computing ($x_R \bmod \rho, y_R \bmod \rho$) and by comparing it with the second computation. This technique is usually used in the grey-box context as a countermeasure against fault attacks. However in the white-box context, the adversary may be able to control the fault he induces on the first computation of R in $\mathbb{Z}/\rho n\mathbb{Z}$. He could thus add a multiple of ρ to its coordinates and the resulting faulty point would pass the verification. Hence, this technique prevents the use of random faults on R but it could still be bypassed. Of course, the attacker could also try to skip the consistency check, but the same problem arises for any redundancy-based countermeasure. As for delegation of calculus to a third party, it may not be possible in some contexts.

Joint Encodings

The idea of joint encodings is to link several values together. For instance, if the encodings used in the scheme are bijections applied to bytes, one can jointly encode variables a and b by applying the bijections on a nibble of a concatenated with a nibble of b. This way, it shall be more difficult for an adversary to modify a without disturbing b. In particular, joint encodings can prevent re-injections that consist in storing a variable obtained during an execution to force its value in a subsequent one.

Joint encodings can for instance be used as described in [RH18]. Let us denote by $\overline{(x, y)}$ the joint encoding of (x, y) and let $\alpha \geq 2$ be a security parameter. First, a function f and several functions g_i (for $0 \leq i \leq 2\alpha - 1$) are selected offline. Then, during the execution, two sets A and B are generated:

$$A = \{\overline{(f(e), g_i(e))}\}_{0 \leq i \leq 2\alpha-1} = \{\overline{(k, e_i)}\}_{0 \leq i \leq 2\alpha-1},$$
$$B = \{\overline{(g_{2j}(e), g_{2j+1}(e))}\}_{0 \leq j \leq \alpha-1} = \{\overline{(e_{2j}, e_{2j+1})}\}_{0 \leq j \leq \alpha-1}.$$

The set A is used to compute an encoded version of the variable $a = k + \sum_{i=0}^{2\alpha-1} c_i e_i$, with c_i being constants drawn offline. The point $[a]G = R + [\sum_{i=0}^{2\alpha-1} c_i e_i]G$ is then computed and we would like to extract R from it. To do so, the point $[\sum_{i=0}^{2\alpha-1} c_i e_i]G$ is constructed

from the set B and is subtracted from $[a]G$. The computation of s is performed similarly to the scalar multiplication: two distinct sets containing jointly encoded values depending on the nonce k and the hash e are used separately to compute two variables which, once added together, give the value of s.

Joint encodings are quite costly, but they can prevent several fault attacks, and in particular the one that consists in storing the value of k during an execution in order to re-inject it in another one and obtain two signatures using the same nonce. Indeed, an adversary attempting to fix k would have to also fix the e_i's in the set A, which would make them inconsistent with those in the set B and lead to an unexploitable faulty output. If the adversary also tried to re-inject the values of the set B, the resulting signature would be correct but the signed hash would be the same as the previous one, so he would not learn any new information.

Note however that joint encodings are not flawless: the adversary could for instance manage to introduce a bias in the nonce without modifying the e_i's, in particular if some part of the encoded value does not depend on them.

Offline r Computations

Since it is both difficult and costly to implement efficient protections against fault attacks on k or r, the authors of [CP18] propose to precompute theses values and embed them inside the white-box. Offline, several nonces k_i are drawn at random and the corresponding r_i are computed. One-time usage white-boxes WB_i embedding k_i^{-1} and $k_i^{-1}r_id$ are then created and couples (r_i, WB_i) are stored in the device. To sign a message, a couple is selected and the hash e is given as input to WB_i, which only performs encoded modular operations to compute $s_i = k_i^{-1}(e + r_id) \bmod n$.

To prevent an adversary from using twice the same white-box on different messages, which would lead him to two signatures using the same nonce, it is proposed to erase the couple (r_i, WB_i) once it is used. However, this protection is inefficient against an attacker having full control over the execution environment. Indeed, such an adversary could extract a couple (r_i, WB_i) and use it as many times as he wants. Finding a countermeasure seems to be a difficult task since a white-box cannot count on secure non-volatile memory to prevent its use on two different inputs. Hence, this patent does not seem to be applicable in every context. Furthermore, it implies to store many white-boxes, and thus use a lot of memory, or to have a regular connection to a server providing new tables.

Table 4.3 summarizes the countermeasures presented in this section. They all prevent some specific attacks but present limitations in terms of performances and/or security. While simple automatic attacks may be avoided with a limited cost, an adversary having full knowledge on the implementation would probably be able to recover the key despite a combination of countermeasures. More work on the protection of ECDSA in the white-box model is thus needed.

4.5 Cloud-Assisted ECDSA White-Box

As mentioned at the beginning of this chapter, there is actually one published ECDSA white-box design [ZBJ20], even though it cannot be used in every context. In that scheme, the signature is computed with the assistance of a cloud server that helps to enhance the security by selecting some random values and by performing some of the ECDSA

Table 4.3: Recap on the countermeasures and their main drawbacks.

Attack type	Countermeasures	Restrictions
Passive analysis	Encoding	Not suited for all operations, all propositions broken.
	Masking	No reliable source of randomness, not always secure with fixed masks.
	Homomorphic encryption	Implies implementing a decryption white-box or outputting encrypted signatures.
Active analysis	Redundancy and checks of consistency	Can be bypassed with two or more faults.
	Joint encodings	Difficult to apply to some values without a prohibitive cost, may not prevent all fault attacks.
	Offline r computations	Difficult to prevent two uses of the same nonce.

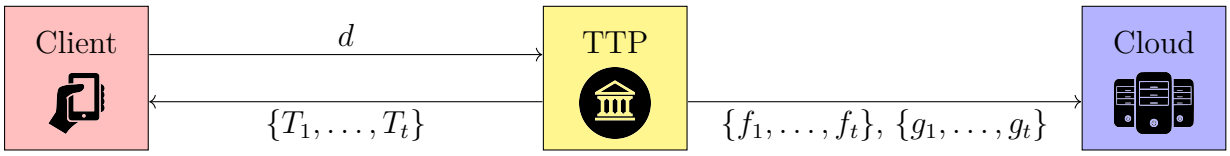


Figure 4.1: The ECDSA white-box implementation of [ZBJ20]: initialization.

operations. In this section, we analyze the security of this white-box design. We exhibit two different and very efficient fault attacks based on a common principle to re-inject some specific values from one signature execution to another. We also suggest a countermeasure that prevents both these attacks without impacting the size of the white-box. Only the performances on the server side are slightly impacted.

The results presented here were published in collaboration with Christophe Giraud in [GH23].

4.5.1 Zhou et al.'s Construction

In 2020, Zhou et al. published the very first public ECDSA white-box scheme [ZBJ20]. In this proposal, the signature is partly computed on the client device and partly on a cloud server. The latter is considered semi-honest, meaning that it may try to recover information on the sensitive variables but stores the data without tempering with it and honestly executes every operation. The key is fully contained on the client's side, preventing the server to learn any information about it, and is protected against a white-box attacker by the use of encoded look-up tables. As previously mentioned, this method is not well suited for operations involving a large number of bits, such as those performed during an ECDSA signature. Here, the size of the tables is kept relatively small thanks to the use of the Residue Number System (RNS).

The Residue Number System

The *Residue Number System* (RNS) was introduced in 1959 by Garner [Gar59]. The idea is to represent an integer x by its values modulo several pairwise coprime integers $\{p_1, \ldots, p_t\}$, that is $x = (x_1, \ldots, x_t)$ with $x_i = x \bmod p_i$. Note that x is uniquely represented only if we add the condition $0 \leq x < P$ with $P = \prod_{i=1}^{t} p_i$. Additions, subtractions and multiplications can then be performed on the small values x_i in order to improve their efficiency and the final result can be retrieved using the Chinese Remainder Theorem (CRT). This means that for $\odot \in \{+, -, \times\}$, $x = (x_1, \ldots, x_t)$ and $y = (y_1, \ldots, y_t)$, the integer $z = x \odot y$ can be represented by the $z_i = x_i \odot y_i \bmod p_i$ for all $1 \leq i \leq t$. Furthermore, if $0 \leq z < P$, its value can be retrieved as

$$z = \sum_{i=1}^{t} P_i (z_i P_i^{-1} \bmod p_i) \bmod P,$$

where $P_i = P/p_i$.

Initialization

As usually in white-box cryptography, an initialization phase is needed to compute the tables containing the key. It is considered to be performed in a secure environment by a

Trusted Third Party (TTP), see Figure 4.1. After receiving the key d from the client, the TTP will follow these steps:

1. Select a basis for the RNS $\beta = \{p_1, \dots p_t\}$ of pairwise co-prime integers. Some ECDSA operations are going to be performed modulo $P = \prod_{i=1}^{t} p_i$ instead of modulo n, and if some reductions occur, then the resulting signature is not valid. Hence, the basis of the RNS has to be such that P is sufficiently big, and in our case, $P > n^2 + n$.

2. Select random permutations f_i and g_i on $\mathbb{Z}_{p_i}$ for $1 \leq i \leq t$.

3. Construct the tables

$$T_i(j,k) = f_i^{-1}(j) + g_i^{-1}(k)d_i \bmod p_i$$

with $d_i = d \bmod p_i$ for $1 \leq i \leq t$ and $1 \leq j, k \leq p_i$.

4. Securely send the tables $\{T_1, \dots, T_t\}$ to the client and the permutations $\{f_1, \dots, f_t\}$ and $\{g_1, \dots, g_t\}$ to the server.

5. Publish the RNS basis $\beta = \{p_1, \dots p_t\}$.

Computing a Signature

Once the initialization phase is over, the client and the cloud server can compute a signature together. As shown in Figure 4.2, the client starts by sending the hash e of the message and a point $R_1 = [k_1]G$, k_1 being a random value corresponding to its share of the nonce k. The server can then compute the first part of the signature r, that is the x-coordinate of the point $R_2 = [k_2]R_1 = [k_1k_2]G$ with k_2 its own share of the nonce. It also computes two values $u = k_2^{-1}e \bmod n$ and $v = k_2^{-1}r \bmod n$ and represents them in the RNS basis as $(u_1, \dots, u_t)$ and $(v_1, \dots, v_t)$ with $u_i = u \bmod p_i$ and $v_i = v \bmod p_i$. Since these values would allow an attacker to recover the private key, they cannot be sent in plain to the client, so they are first obfuscated by the functions f_i and g_i received from the TTP. The server's response to the client thus consists of the values r, $(\overline{u_1}, \dots, \overline{u_t})$ and $(\overline{v_1}, \dots, \overline{v_t})$ with $\overline{u_i} = f_i(u_i)$ and $\overline{v_i} = g_i(v_i)$ for $1 \leq i \leq t$. These values are then given as inputs to the tables T_i stored on the client's side, yielding

$$\begin{aligned} w_i &= T_i(\overline{u_i}, \overline{v_i}) \\ &= u_i + v_i d_i \bmod p_i. \end{aligned}$$

Using the CRT, the client can reconstruct $w = u + vd \bmod P$. Since the basis of the RNS was chosen such that $P > n^2 + n > w$, there is no reduction in the computation of w, and $w = u + vd$. Therefore, the second part of the signature can be computed as

$$\begin{aligned} s &= k_1^{-1} w \bmod n \\ &= k^{-1}(e + rd) \bmod n \end{aligned}$$

with $k = k_1k_2$.

The authors of [ZBJ20] argue the security of their scheme against key recovery attacks by explaining that an attacker knowing only the values of the intermediate variables on the client side cannot learn any sensitive information. Similarly, an attacker that is only in the cloud cannot recover the private key.

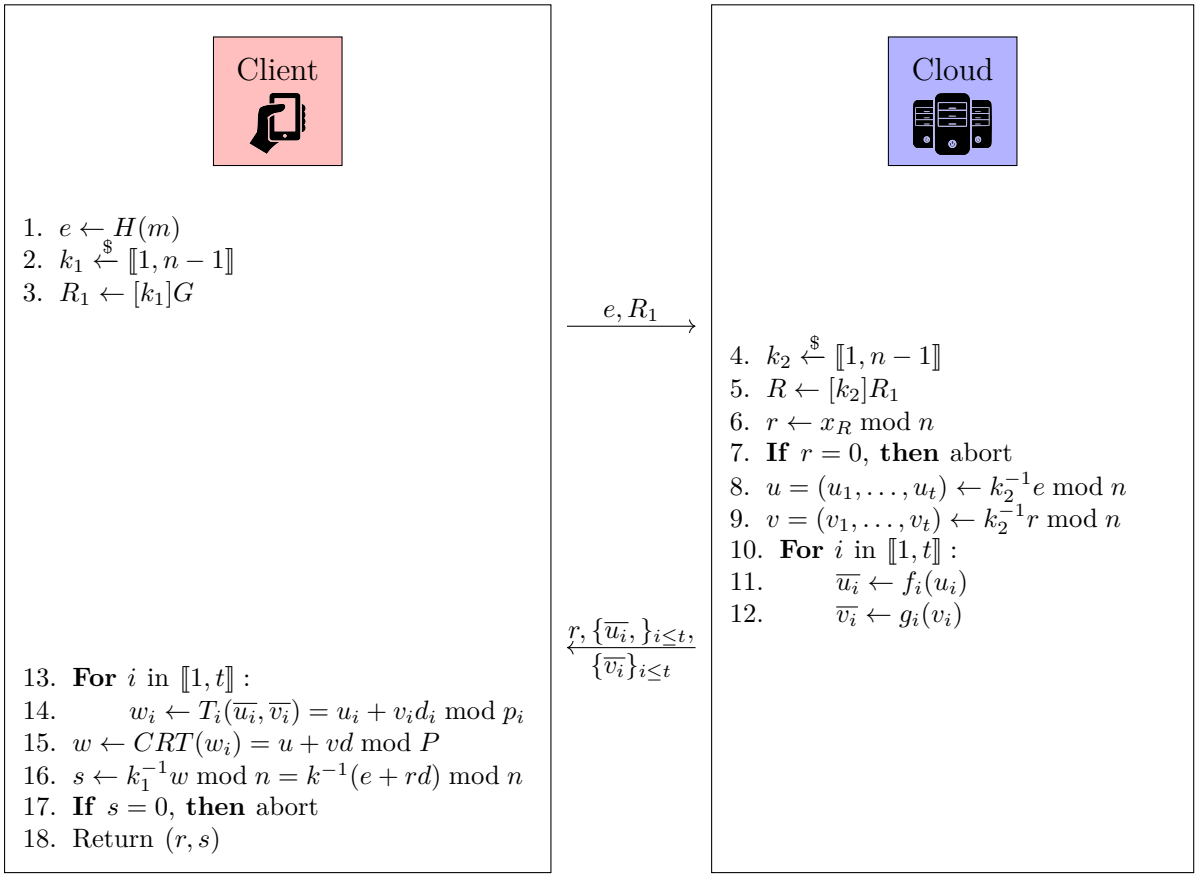


Figure 4.2: The ECDSA white-box implementation of [ZBJ20]: signature

Resisting Code-Lifting Attacks

Zhou et al. also propose a slight modification of their scheme that allows to resist simple code-lifting attacks, that consist in stealing the code and copying it on another device in order to use it to encrypt chosen plaintexts. The idea is to bind the implementation to a specific device by using its unique identification information I. The tables are then constructed as

$$T_i(j,k) = f_i^{-1}(j) + g_i^{-1}(k)d_i + I_i \bmod p_i$$

with $I_i = I \bmod p_i$. We then get $w = u + vd + I \bmod n$ and the second part of the signature is computed as

$$s = k_1^{-1}(w - I) \bmod n.$$

Of course, the value I should not be stored in plain in the code but rather extracted from the device for each signature.

Despite the apparent robustness of this solution, we will show in the next section that this white-box implementation can actually be broken if the attacker uses fault attacks.

4.5.2 Two Fault Attacks

Both our fault attacks require the adversary to observe the computation of three signatures and fault two of them. The first attack is more efficient but it requires to force the value of the hash to zero, which could be detected by the server. On the contrary, the second one is impossible for the server to detect. For the sake of simplicity, let us first describe the attacks on the version of Zhou et al.'s scheme described in Figure 4.2 that is not protected against code-lifting.

The Null Hash Attack

A first idea to break the protocol is based on the fact that when the hash is null, that is $e = 0$, we have $u_i = 0$ for all $1 \leq i \leq t$, so one can obtain $\overline{u_i} = f_i(0)$ when observing the exchange between the server and the client. This attack requires the adversary to ask for three signatures. In the following, we denote by $x^{(j)}$ the value of x obtained during the j^{th} execution for $1 \leq j \leq 3$. For example, $e^{(2)}$ is the hash value of the second execution. Here are the steps that an attacker has to follow in order to retrieve the private key:

1. During a first execution, fault the hash value in order to obtain $e^{(1)} = 0$ and store $\overline{u_i^{(1)}}$ for all $1 \leq i \leq t$. The fault can either be done during Step 1 of the protocol, see Figure 4.2, or during the hash transmission from the client to the server;

2. During a second execution, store the values $w^{(2)} = u^{(2)} + v^{(2)}d$ and $\overline{v_i}^{(2)}$ for all $1 \leq i \leq t$;

3. During a third execution, re-inject the values $\overline{u_i^{(1)}}$ and $\overline{v_i^{(2)}}$ in order to obtain $w^{(3)} = v^{(2)}d \bmod P$;

4. Compute $k_2^{(2)} = e^{(2)}(w^{(2)} - w^{(3)})^{-1} \bmod n$ and recover d as shown in Equation (4.1).

This attack is very efficient but it has a small drawback: the server could detect that the hash value was forced at zero. We now discuss another attack that cannot be detected by the server.

The Greatest Common Divisor Attack

Another solution for the adversary to recover the private key is to fix the values u_i during several executions so that the differences of the $w^{(j)}$ give multiples of d. The adversary may then recover the private key by computing a greatest common divisor (gcd). In the following, we denote by $a \wedge b$ the gcd of two variables a and b. The attack consists of the following steps:

1. During a first execution, store the values $w^{(1)} = u^{(1)} + v^{(1)}d \bmod P$ and $\overline{u_i^{(1)}}$ for all $1 \leq i \leq t$;

2. During a second execution, change $\overline{u_i^{(2)}}$ into $\overline{u_i^{(1)}}$ for $1 \leq i \leq t$ in order to obtain $w^{(2)} = u^{(1)} + v^{(2)}d \bmod P$;

3. During a third execution, change $\overline{u_i^{(3)}}$ into $\overline{u_i^{(1)}}$ for $1 \leq i \leq t$ in order to obtain $w^{(3)} = u^{(1)} + v^{(3)}d \bmod P$;

4. Compute $a = w^{(1)} - w^{(2)} = (v^{(1)} - v^{(2)})d \bmod P$;

5. Compute $b = w^{(1)} - w^{(3)} = (v^{(1)} - v^{(3)})d \bmod P$;

6. Compute $D = a \wedge b = \alpha \times d$ with $\alpha = (v^{(1)} - v^{(2)}) \wedge (v^{(1)} - v^{(3)})$;

7. Brute force the value of α.

Note that we are able to compute the gcd of a and b because there is no reduction modulo P in the computations of $w^{(1)}$, $w^{(2)}$, and $w^{(3)}$. This is due to the fact that P is greater that $n^2 + n$. Furthermore, the last step of the attack is possible only if the value of α is relatively small. If we model $v^{(1)} - v^{(2)}$ and $v^{(1)} - v^{(3)}$ as two independent random variables in $\{1, \ldots, n\}$, the probability of their gcd being large is very low. Indeed, it is shown in [DE04] that if x and y are two random variables in $\{1, \ldots, n\}$, then for all α in $\{1, \ldots, n\}$ we have

$$\mathbb{P}(x \wedge y = \alpha) \approx \frac{6}{\pi^2 \alpha^2}. \tag{4.3}$$

Especially, we deduce from Equation (4.3) that the probability of our two variables being co-primes, that is $\alpha = 1$, approaches 0.61. Hence, our attack theoretically succeeds with probability 0.61^4 without the brute-force step. Also from Equation (4.3), one can compute that we reach a 99% success rate[4] if we test all the α from 1 to 62.

As mentioned previously, the authors of [ZBJ20] also propose a slightly modified version of their scheme where an identification information I is added in the tables in order to resist code-lifting attacks. Interestingly, both our null hash and greatest common divisor attacks also apply to this version without any modification since in both cases we subtract two values of w obtained during two different executions, thus getting rid of I.

Our work proves once again the difficulty of securing an ECDSA white-box against fault attacks, even with the help of a cloud server.

[4]Note that to increase these success rates, one can also use other multiples of the key d. For instance, we have $b - a = (v^{(2)} - v^{(3)})d$. In the sixth step of the attack, the computation of D is then replaced by $D' = (a \wedge b) \wedge (b - a) = \alpha' d$ with $\alpha' = ((v^{(1)} - v^{(2)}) \wedge (v^{(1)} - v^{(3)})) \wedge (v^{(2)} - v^{(3)}) \leq \alpha$.

4.5.3 Experiments

We implemented both the null hash attack and the greatest common divisor attack in Python with the parameters given as example in [ZBJ20, Table 2]. In particular,

$$\begin{aligned}\beta = \{&97, 113, 127, 131, 137, 139, 149, 151,\\ &157, 163, 167, 173, 179, 181, 191, 193,\\ &197, 199, 211, 223, 227, 229, 233, 239,\\ &241, 251, 257, 263, 269, 271, 277, 281,\\ &283, 293, 307, 311, 313, 317, 331, 337,\\ &347, 349, 353, 359, 367, 373, 379, 383,\\ &389, 397, 401, 409, 419, 421, 431, 433,\\ &439, 443, 449, 457, 461, 463, 467, 479\}\end{aligned}$$

and

$$n = 0\text{x}800000000000000000000000000000016288\text{AEDF18093517F7A3E3CFF023CBFF} \ .$$

While the null hash attack always succeeds, the greatest common divisor attack is possible only if α is small. To confirm that this happens with a very high probability, we attacked 50 000 different white-box instances (with different private keys), stored the values of α that we encountered and computed accordingly the probabilities to recover the private key depending on the effort we set for the brute force on α (see Figure 4.3).

If one fixes the limit for the brute force at 5 (resp. 58), he has a probability close to 0.9 (resp. 0.99) to recover the private key. Hence, the attack is very efficient. These results are coherent with the theoretical results of [DE04].

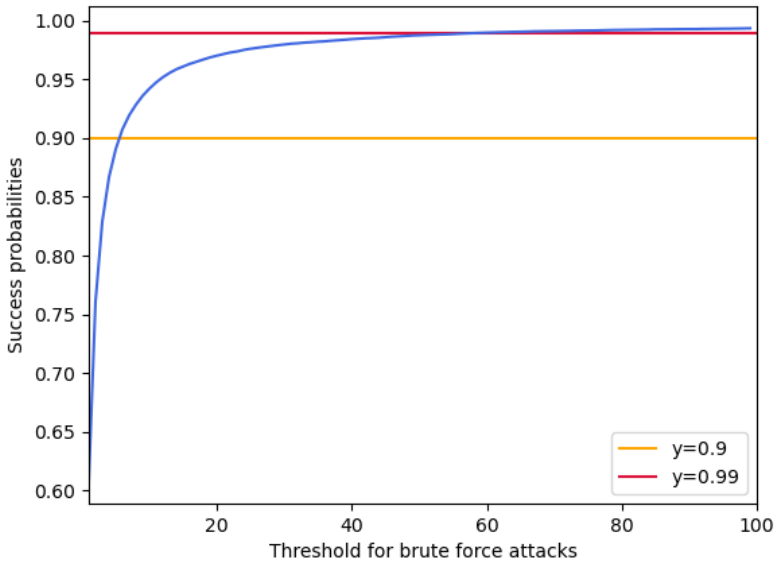


Figure 4.3: Probabilities of success for the greatest common divisor attack depending on the limit set for the brute force on α. These are computed over 50 000 executions.

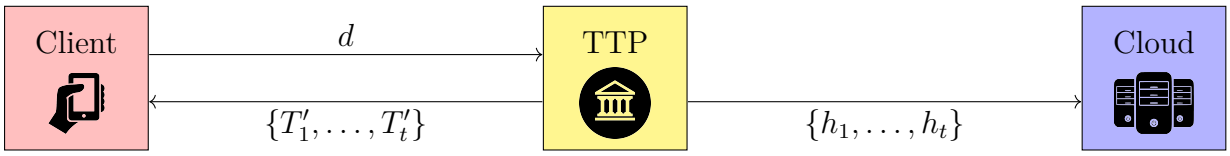


Figure 4.4: Modified initialization.

4.5.4 Proposition of Countermeasure

The null hash attack presented in Section 4.5.2 requires the adversary to set the hash value to 0 during an execution. This could be detected by the cloud but a signature for $e = 0$ must be supported [DSS23]. In order not to disclose the values $\overline{u_i}$ that are necessary for the attack, one can store in advance some valid signatures for $e = 0$ on the client side before making the server abort the protocol when a null hash is given. However, such a solution cannot be implemented against the greatest common divisor attack since the hash does not have to take any specific value. We thus propose a better solution, that actually prevents both attacks.

The null hash attack and the greatest divisor attack both rely on the fact that it is possible to fault the values $\overline{u_i}$ without perturbing $\overline{v_i}$ and vice-versa for all $1 \leq i \leq t$. A countermeasure against these attacks is thus to bind them together, following the principle of joint encodings. We note ℓ_i the bit size of p_i, that is $\ell_i = \log(p_i)$. Let q_i be such that $q_i = 2^{2\ell_i} - 1$ for all $1 \leq i \leq t$. Also, let $x^{[k]}$ be the k most significant bits of x and $x_{[k]}$ be the k least significant bits of x. We propose the following slight modifications of the scheme (see Figure 4.4 and Figure 4.5):

- Instead of two sets of permutations $\{f_1, \ldots, f_t\}$ and $\{g_1, \ldots, g_t\}$ with each of the f_i, g_i on $\{1, \ldots, p_i\}$, the TTP generates a unique set of bigger permutations $\{h_1, \ldots, h_t\}$ with h_i on $\{1, \ldots, q_i\}$.
- The TTP then constructs the tables T'_i as

$$T'_i(j) = h_i^{-1}(j)_{[\ell_i]} + h_i^{-1}(j)^{[\ell_i]} d_i \bmod p_i \ .$$

- Instead of $\{\overline{u_1}, \ldots, \overline{u_t}\}$ and $\{\overline{v_1}, \ldots, \overline{v_t}\}$, the server computes and sends $\{\overline{x_1}, \ldots, \overline{x_t}\}$ with $\overline{x_i} = h_i(u_i \| v_i)$ for $1 \leq i \leq t$.
- The client computes $w_i = T'_i(\overline{x_i}) = u_i + v_i d_i \bmod p_i$ for $1 \leq i \leq t$.

With these modifications, it is not possible anymore to select a value for u_i without impacting v_i and the attacks presented in Section 4.5.2 do not work.

This countermeasure comes with a small overhead. In the TTP's side, bigger permutations have to be selected, which induces an overhead in both time complexity and memory consumption. The same is observed for the server that has to store a permutation twice as big. However, the interesting thing is that nothing changes on the client side since the size of the tables remains the same. Indeed, instead of taking two inputs of ℓ_i bits, the tables takes one input of $2\ell_i$ bits which does not make a difference.

4.6 Conclusion

In this chapter, we presented a variety of simple automated attacks against ECDSA white-boxes and studied their efficiency on the challenges from the WhibOx 2021 contest. We

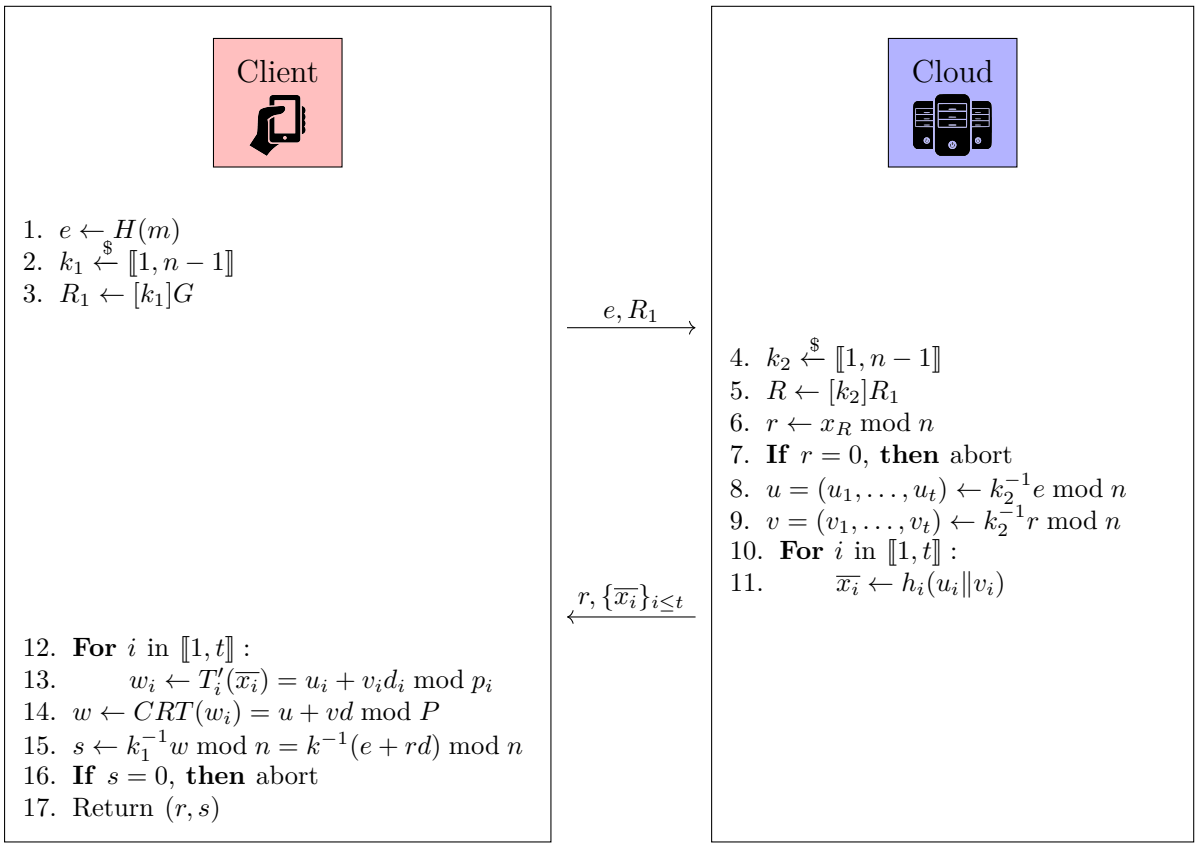


Figure 4.5: Modifed ECDSA signature operation.

then discussed various countermeasures that we found in patents, ranging from classical ones as masking or encodings to more sophisticated solutions as the use of homomorphic encryption. We highlighted the advantages and limitations of each countermeasure. Finally, we showed how to break the only published design with two fault attacks that consist in re-injecting intermediate values from an execution to another.

Our work shows that securing ECDSA in the white-box model is an even bigger challenge than the one faced for symmetric cryptography. Indeed, several new issues are encountered. First, the usual encoding techniques are not well suited for arithmetic operations on big numbers, so new solutions have to be invented and deployed. Second, the security of ECDSA is heavily based on the possibility of drawing random values during its execution, which is difficult in the white-box context. In an attempt to prevent attackers from fixing them, designers often chose to derive the random values from the only reliable source of randomness available, that is the input of the white-box. This solution has the effect of making the scheme deterministic, which, in the case of ECDSA, opens many efficient attack paths.

In particular, fault attacks can be devastating against deterministic ECDSA white-boxes. The attack surface is huge as a lot of variables can be altered in order to retrieve the secret key. Moreover, the white-box context makes the attacker extremely powerful and may enable him to chose the value of the induced fault or to alter several variables in a single execution. Usual countermeasures relying on consistency checks can therefore be bypassed more easily. An interesting future work would thus be to find infective countermeasures for ECDSA that even a white-box attacker could not defeat.

Chapter 5

Conclusion

Despite an extensive study for some algorithms such as AES, the existence of secure white-boxes for standard schemes is still an open problem as every published design is currently broken. This has led companies to implement proprietary solutions tailored to their specific use cases. These implementations are intended to prevent automated attacks and use code obfuscation as a first barrier to force the adversary to go through a time-consuming reverse-engineering process before attempting to identify vulnerabilities. Nevertheless, even preventing automated attacks can be challenging, particularly so when one wants to preserve the program's normal input/output behavior for interoperability purposes. Indeed, there exists a large variety of threats, including side-channel and fault attacks inherited from the grey-box model which become even more powerful in the white-box context.

For some use-cases, it may be possible to implement additional security features through the application itself. For instance, one could add authentication mechanisms or regularly update the secret key so that the number of executions observable by the adversary is limited. Consequently, the most dangerous attacks in practice are the ones that can be automated and require a reduced number of white-box executions on random inputs. We believe that studying such attacks alongside practical countermeasures is of significant interest, making it the primary focus of this thesis.

In this work, we concentrated on two standard and widely-used cryptosystems: AES and ECDSA. After providing a brief overview of the current state-of-the-art on AES white-box implementations, we proposed a new square-like attack that advantageously requires very few executions on random plaintexts. We then studied side-channel attacks and in particular the protection provided by internal encodings. While it was of common knowledge that an AES implementation protected by small random bijections could be broken by a Differential Computation Analysis (DCA) with high probability, the question of the existence of a particular class of encodings that could actually prevent the attack was still open. In this thesis, we answered it negatively. We showed that the correlation between predicted values and trace points actually depends on the Walsh spectrum of the used encodings, and although selecting bijections with only small Walsh transforms could prevent a classical DCA, it is still possible to break the implementation with a variant of the attack. Internal encodings are therefore insufficient for the protection of AES white-boxes against side-channel attacks, whether drawn at random or carefully crafted.

We then turned our attention to ECDSA and highlighted the inherent additional complexities of the scheme. Although very little studied to this day, ECDSA white-boxes are crucial for the industry and solutions are already deployed on the field. This contradiction motivated the 2021 and 2024 WhibOx contests, affiliated to the CHES

workshops, which were organized to encourage practical research from both designers' and attackers' perspectives. In this thesis, we described several efficient attacks and in particular those executed by the team who broke the highest number of challenges during the 2021 edition, TheRealIdefix, which we were part of. We showed that fault attacks are particularly devastating when the ECDSA white-box is deterministic, as most of the ones implemented during the contest. Additionally, we provided an overview of the design of the two winning candidates. While theoretically interesting, the countermeasures employed were insufficient to thwart all automated attacks.

In the subsequent part of the chapter, we thus studied the only other source of information that is publicly available on standard ECDSA white-boxes: the patents that were filled by some companies. Of course, patents may omit crucial details on the implementation or the security model, but they remain valuable sources of information on implemented methods in the field. We thus exposed in this thesis the main ideas to counteract the various attacks that can be applied to ECDSA white-boxes, emphasizing their advantages and limitations in terms of performances and security. We investigated existing countermeasures against both passive and active automated attacks and showed once again that it seems difficult to forestall all threats without a prohibitive memory cost.

Finally, our last contribution is the first attack on the only published ECDSA white-box design from Zhou et al. Since it requires the help of a semi-honest cloud server during each signature computation, this scheme cannot be used in every context but should be easier to secure. The key, fully contained on the client's side, is protected with encoded look-up tables which size is kept relatively small thanks to the use of the Residue Number System (RNS). We showed that it can be efficiently recovered by an attacker inducing faults into the computations. More precisely, we presented two different attacks that involve re-injecting specific values stored during a previous execution in a new one. What differentiates them is that the first attack can be detected by the server while the other cannot but is a little less efficient. We also proposed a countermeasure based on joint encodings that prevents both attacks and advantageously does not impact the performances on the client's side nor the input/output behavior.

To conclude, this thesis aimed to advance the research on practical white-box cryptography by studying attacks that could be performed by real-life adversaries on AES or ECDSA implementations, and countermeasures that are not in contradiction with industrial needs. Specifically, the attacks we discussed require very few white-box executions and most can be automated and/or applied on random inputs. The countermeasures we considered are practical in terms of performances and do not alter the input/output behavior of the algorithms. We also provided new insights into the protection offered by internal encodings, one of the most widely deployed countermeasure, against DCA, one of the most efficient and well-known attacks in white-box cryptography.

Future research directions

New countermeasures againt DCA. Since we show that no class of small encodings could efficiently prevent side-channel attacks on AES white-boxes, it would be of high interest to find other countermeasures that do not require drawing random variables during the execution of the algorithm and that could, potentially in combination with encodings, make DCA fail.

Practical implementation of AES. We argue in this thesis that even though there is currently no public implementation of AES that is secure in the theoretical white-box model, some of them can be used to achieve practical security in some contexts. We believe that it would be interesting to develop a design that intends to protect the secret key against a very specific weakened attacker. For instance, one could consider that the adversary cannot execute the code himself and thus has a bounded number of executions to observe and/or alter. Preferably, such a design should not rely on external encodings or any other countermeasure that modifies the input/output behavior of the algorithm.

WhibOx 2024 and ECDSA white-boxes. We have seen that for several reasons, protecting ECDSA in the white-box model seems particularly challenging. More research on this subject is therefore needed to achieve even practical security against automated attacks. With the WhibOx contest being held once again in 2024 and still targeting ECDSA, a first step would be to try to submit an implementation that is immune to all the attacks mentioned in this thesis and could stay unbroken for a long period of time.

Other (post-quantum) standards. Finally, another research direction would be to study the white-boxability of other standard asymmetric schemes. In particular, it would be of high interest to find a secure design for a post-quantum algorithm such as the Module-Lattice-Based Digital Signature Standard (ML-DSA, [ML-23]).

Bibliography

[AABM20] Estuardo Alpirez Bock, Alessandro Amadori, Chris Brzuska, and Wil Michiels. On the security goals of white-box cryptography. *IACR TCHES*, 2020(2):327–357, 2020. doi:10.13154/tches.v2020.i2.327-357.

[ABC+15] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A. Reuter, and Martin Strand. A guide to fully homomorphic encryption. Cryptology ePrint Archive, Report 2015/1192, 2015. https://eprint.iacr.org/2015/1192.

[ABF+18] Christopher Ambrose, Joppe W. Bos, Björn Fay, Marc Joye, Manfred Lochter, and Bruce Murray. Differential attacks on deterministic signatures. In Nigel P. Smart, editor, *CT-RSA 2018*, volume 10808 of *LNCS*, pages 339–353. Springer, April 2018. doi:10.1007/978-3-319-76953-0_18.

[ABMT18] Estuardo Alpirez Bock, Chris Brzuska, Wil Michiels, and Alexander Treff. On the ineffectiveness of internal encodings - revisiting the DCA attack on white-box cryptography. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18International Conference on Applied Cryptography and Network Security*, volume 10892 of *LNCS*, pages 103–120. Springer, July 2018. doi:10.1007/978-3-319-93387-0_6.

[AES23] Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce, May 2023. Update of a document published in 2001. doi:10.6028/NIST.FIPS.197-upd1.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996. doi:10.1145/237814.237838.

[AR04] Dorit Aharonov and Oded Regev. Lattice problems in NP cap coNP. In *45th FOCS*, pages 362–371. IEEE Computer Society Press, October 2004. doi:10.1109/FOCS.2004.35.

[Bar20] Lucas Barthelemy. Toward an asymmetric white-box proposal. Cryptology ePrint Archive, Report 2020/893, 2020. https://eprint.iacr.org/2020/893.

[BBD+22] Guillaume Barbu, Ward Beullens, Emmanuelle Dottax, Christophe Giraud, Agathe Houzelot, Chaoyun Li, Mohammad Mahzoun, Adrián Ranea, and Jianrui Xie. ECDSA white-box implementations: Attacks and designs from CHES 2021 challenge. *IACR TCHES*, 2022(4):527–552, 2022. doi:10.46586/tches.v2022.i4.527-552.

[BCD06] Julien Bringer, Herve Chabanne, and Emmanuelle Dottax. White box cryptography: Another attempt. Cryptology ePrint Archive, Report 2006/468, 2006. https://eprint.iacr.org/2006/468.

[BCH16] Chung Hun Baek, Jung Hee Cheon, and Hyunsook Hong. White-box AES implementation revisited. *Journal of Communications and Networks*, 18(3):273–287, 2016. doi:10.1109/JCN.2016.000043.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, August 2004. doi:10.1007/978-3-540-28632-5_2.

[BCZ18] Luk Bettale, Jean-Sébastien Coron, and Rina Zeitoun. Improved high-order conversion from Boolean to arithmetic masking. *IACR TCHES*, 2018(2):22–45, 2018. doi:10.13154/tches.v2018.i2.22-45.

[BDG+22] Sven Bauer, Hermann Drexler, Max Gebhardt, Dominik Klein, Friederike Laus, and Johannes Mittmann. Attacks against white-box ECDSA and discussion of countermeasures – A report on the WhibOx contest 2021. *IACR TCHES*, 2022(4):25–55, 2022. doi:10.46586/tches.v2022.i4.25-55.

[BDK+18] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-kyber: a CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367. IEEE, 2018. doi:10.1109/EUROSP.2018.00032.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 37–51. Springer, May 1997. doi:10.1007/3-540-69053-0_4.

[BECN+04] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. Cryptology ePrint Archive, Report 2004/100, 2004. https://eprint.iacr.org/2004/100.

[BG13] Alberto Battistello and Christophe Giraud. Fault analysis of infective AES computations. In Wieland Fischer and Jörn-Marc Schmidt, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 101–107. IEEE Computer Society, 2013. doi:10.1109/FDTC.2013.12.

[BGEC04] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a white box AES implementation. In Helena Handschuh and Anwar Hasan, editors, *SAC 2004*, volume 3357 of *LNCS*, pages 227–240. Springer, August 2004. doi:10.1007/978-3-540-30564-4_16.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, August 2001. doi:10.1007/3-540-44647-8_1.

[BHMT16] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 215–236. Springer, August 2016. doi:10.1007/978-3-662-53140-2_11.

[BI15] Andrey Bogdanov and Takanori Isobe. White-box cryptography revisited: Space-hard ciphers. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1058–1069. ACM Press, October 2015. doi:10.1145/2810103.2813699.

[BIT16] Andrey Bogdanov, Takanori Isobe, and Elmar Tischhauser. Towards practical whitebox cryptography: Optimizing efficiency and space hardness. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 126–158. Springer, December 2016. doi:10.1007/978-3-662-53887-6_5.

[BLU23] Alex Biryukov, Baptiste Lambin, and Aleksei Udovenko. Cryptanalysis of ARX-based white-box implementations. *IACR TCHES*, 2023(3):97–135, 2023. doi:10.46586/tches.v2023.i3.97-135.

[BMM00] Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential fault attacks on elliptic curve cryptosystems. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 131–146. Springer, August 2000. doi:10.1007/3-540-44598-6_8.

[Boc20] Markus Bockes. White-box ECC implementation. Patent WO2020192968A1, 2020.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 513–525. Springer, August 1997. doi:10.1007/BFb0052259.

[CCD+17] Jihoon Cho, Kyu Young Choi, Itai Dinur, Orr Dunkelman, Nathan Keller, Dukjae Moon, and Aviya Veidberg. WEM: A new family of white-box block ciphers based on the Even-Mansour construction. In Helena Handschuh, editor, *CT-RSA 2017*, volume 10159 of *LNCS*, pages 293–308. Springer, February 2017. doi:10.1007/978-3-319-52153-4_17.

[CCDP04] Vincent Carlier, Hervé Chabanne, Emmanuelle Dottax, and Hervé Pelletier. Electromagnetic side channels of an FPGA implementation of AES. Cryptology ePrint Archive, Report 2004/145, 2004. https://eprint.iacr.org/2004/145.

[CCJ03] Benoît Chevallier-Mames, Mathieu Ciet, and Marc Joye. Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. Cryptology ePrint Archive, Report 2003/237, 2003. https://eprint.iacr.org/2003/237.

[CDKS15] Bren Cavallo, Giovanni Di Crescenzo, Delaram Kahrobaei, and Vladimir Shpilrain. Efficient and secure delegation of group exponentiation to a single server. Cryptology ePrint Archive, Report 2015/206, 2015. https://eprint.iacr.org/2015/206.

[CEJv02] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box DES implementation for DRM applications. In Joan Feigenbaum, editor, *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop*, volume 2696 of *LNCS*, pages 1–15. Springer, 2002. doi:10.1007/978-3-540-44993-5_1.

[CEJv03] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 250–270. Springer, August 2003. doi:10.1007/3-540-36492-7_17.

[CGP+12] Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-order masking schemes for S-boxes. In Anne Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 366–384. Springer, March 2012. doi:10.1007/978-3-642-34047-5_21.

[CH23] Arnaud Casteigts and Agathe Houzelot. Vers une cryptographie en boîte blanche ? *Interstices*, 2023. https://interstices.info/vers-une-cryptographie-en-boite-blanche.

[CH24] Laurent Castelnovi and Agathe Houzelot. On the (im)possibility of preventing differential computation analysis with internal encodings. Accepted for publication in *TCHES*, 2024(3), 2024.

[Chu17] Chitchanok Chuengsatiansup. *Optimizing curve-based cryptography*. PhD thesis, Technische Universiteit Eindhoven, 2017. https://research.tue.nl/en/publications/optimizing-curve-based-cryptography.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 398–412. Springer, August 1999. doi:10.1007/3-540-48405-1_26.

[Cla07] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 181–194. Springer, September 2007. doi:10.1007/978-3-540-74735-2_13.

[Cor99] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Çetin Kaya Koç and Christof Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 292–302. Springer, August 1999. doi:10.1007/3-540-48059-5_25.

[CP18] Herve Chabanne and Emmanuel Prouff. Method for electronic signing of a document with a predetermined secret key. Patent FR3063857A1, 2018.

[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 2007*, volume 4727 of *LNCS*, pages 28–44. Springer, September 2007. doi:10.1007/978-3-540-74735-2_3.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, August 2003. doi:10.1007/3-540-36400-5_3.

[CT03] Jean-Sébastien Coron and Alexei Tchulkine. A new algorithm for switching from arithmetic to Boolean masking. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 89–97. Springer, September 2003. doi:10.1007/978-3-540-45238-6_8.

[DE04] Persi Diaconis and Paul Erdös. On the distribution of the greatest common divisor. *Lecture Notes-Monograph Series*, 45:56–61, 2004. doi:10.1214/lnms/1196285379.

[Deb12] Blandine Debraize. Efficient and provably secure methods for switching from arithmetic to Boolean masking. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 107–121. Springer, September 2012. doi:10.1007/978-3-642-33027-8_7.

[DEK+18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: Exploiting ineffective fault inductions on symmetric cryptography. *IACR TCHES*, 2018(3):547–572, 2018. doi:10.13154/tches.v2018.i3.547-572.

[DFLM18] Patrick Derbez, Pierre-Alain Fouque, Baptiste Lambin, and Brice Minaud. On recovering affine encodings in white-box implementations. *IACR TCHES*, 2018(3):121–149, 2018. doi:10.13154/tches.v2018.i3.121-149.

[DGH21] Emmanuelle Dottax, Christophe Giraud, and Agathe Houzelot. White-box ECDSA: Challenges and existing solutions. In Shivam Bhasin and Fabrizio De Santis, editors, *COSADE 2021*, volume 12910 of *LNCS*, pages 184–201. Springer, October 2021. doi:10.1007/978-3-030-89915-8_9.

[DGH22] Emmanuelle Dottax, Christophe Giraud, and Agathe Houzelot. Procédé de signature cryptographique d'une donnée, dispositif électronique et programme d'ordinateur associés. Patent FR3133251A1, 2022.

[DH20] Gabrielle De Micheli and Nadia Heninger. Recovering cryptographic keys from partial information, by example. Cryptology ePrint Archive, Report 2020/1506, 2020. https://eprint.iacr.org/2020/1506.

[Dig] Digital.ai. Application protection. https://digital.ai/application-security/key-data-protection/.

[DKL+18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR TCHES*, 2018(1):238–268, 2018. doi:10.13154/tches.v2018.i1.238-268.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 149–165. Springer, January 1997. doi:10.1007/BFb0052343.

[DLPR14] Cécile Delerablée, Tancrède Lepoint, Pascal Paillier, and Matthieu Rivain. White-box security notions for symmetric encryption schemes. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 247–264. Springer, August 2014. doi:10.1007/978-3-662-43414-7_13.

[DR98] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael, 1998. NIST AES proposal. https://csrc.nist.rip/encryption/aes/rijndael/Rijndael.pdf.

[DRP13a] Yoni De Mulder, Peter Roelse, and Bart Preneel. Cryptanalysis of the Xiao-Lai white-box AES implementation. In Lars R. Knudsen and Huapeng Wu, editors, *SAC 2012*, volume 7707 of *LNCS*, pages 34–49. Springer, August 2013. doi:10.1007/978-3-642-35999-6_3.

[DRP13b] Yoni De Mulder, Peter Roelse, and Bart Preneel. Revisiting the BGE attack on a white-box AES implementation. Cryptology ePrint Archive, Report 2013/450, 2013. https://eprint.iacr.org/2013/450.

[DSS23] Digital Signature Standard (DSS). National Institute of Standards and Technology, NIST FIPS PUB 186-5, U.S. Department of Commerce, February 2023. Update of a document published in 1998. doi:10.6028/NIST.FIPS.186-5.

[DWP10] Yoni De Mulder, Brecht Wyseur, and Bart Preneel. Cryptanalysis of a perturbated white-box AES implementation. In Guang Gong and Kishan Chand Gupta, editors, *INDOCRYPT 2010*, volume 6498 of *LNCS*, pages 292–310. Springer, December 2010. doi:10.1007/978-3-642-17401-8_21.

[FGR13] Jean-Charles Faugère, Christopher Goyet, and Guénaël Renault. Attacking (EC)DSA given only an implicit hint. In Lars R. Knudsen and Huapeng Wu, editors, *SAC 2012*, volume 7707 of *LNCS*, pages 252–274. Springer, August 2013. doi:10.1007/978-3-642-35999-6_17.

[FKKM16] Pierre-Alain Fouque, Pierre Karpman, Paul Kirchner, and Brice Minaud. Efficient and provable white-box primitives. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 159–188. Springer, December 2016. doi:10.1007/978-3-662-53887-6_6.

[Gar59] Harvey L Garner. The residue number system. *IRE Transactions on Electronic Computers*, 8(2):140–147, 1959. doi:10.1109/TEC.1959.5219515.

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES 2008*, volume 5154 of *LNCS*, pages 426–442. Springer, August 2008. doi:10.1007/978-3-540-85053-3_27.

[GG22] Pierre Galissant and Louis Goubin. Resisting key-extraction and code-compression: a secure implementation of the HFE signature scheme in the white-box model. Cryptology ePrint Archive, Report 2022/138, 2022. https://eprint.iacr.org/2022/138.

[GH23] Christophe Giraud and Agathe Houzelot. Fault attacks on a cloud-assisted ECDSA white-box based on the residue number system. In *Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 72–80. IEEE, 2023. doi:10.1109/FDTC60478.2023.00016.

[GK04] Christophe Giraud and Erik Woodward Knudsen. Fault attacks on signature schemes. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP 04*, volume 3108 of *LNCS*, pages 478–491. Springer, July 2004. doi:10.1007/978-3-540-27800-9_41.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer, May 2001. doi:10.1007/3-540-44709-1_21.

[GMQ07] Louis Goubin, Jean-Michel Masereel, and Michaël Quisquater. Cryptanalysis of white box DES implementations. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *SAC 2007*, volume 4876 of *LNCS*, pages 278–295. Springer, August 2007. doi:10.1007/978-3-540-77360-3_18.

[Gou01] Louis Goubin. A sound method for switching between Boolean and arithmetic masking. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 3–15. Springer, May 2001. doi:10.1007/3-540-44709-1_2.

[GPQ11] Laurie Genelle, Emmanuel Prouff, and Michaël Quisquater. Thwarting higher-order side channel analysis with additive and multiplicative maskings. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 240–255. Springer, September / October 2011. doi:10.1007/978-3-642-23951-9_16.

[GST12] Benedikt Gierlichs, Jörn-Marc Schmidt, and Michael Tunstall. Infective computation and dummy rounds: Fault protection for block ciphers without check-before-output. In Alejandro Hevia and Gregory Neven, editors, *LATINCRYPT 2012*, volume 7533 of *LNCS*, pages 305–321. Springer, October 2012. doi:10.1007/978-3-642-33481-8_17.

[Gt20] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.2.1 edition, 2020. http://gmplib.org/.

[GV20] Aline Gouget and Jan Vacek. Method for generating a digital signature of an input message. Patent EP3709561A1, 2020.

[HB15] Máté Horváth and Levente Buttyán. The birth of cryptographic obfuscation–a survey. Cryptology ePrint Archive, Paper 2015/412, 2015. https://eprint.iacr.org/2015/412.

[HGD+11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293–302, December 2011. doi:10.1007/s13389-011-0023-x.

[Ide] Idemia. Digital car keys. https://www.idemia.com/wp-content/uploads/2021/06/digital-car-keys-idemia-brochure-202106.pdf.

[Inc] PACE Anti-Piracy Inc. White-box works. https://www.paceap.com/white-box_cryptography.html.

[Int] Intertrust. Whitecryption secure key box. https://www.intertrust.com/products/application-protection/secure-key-box/.

[Ird] Irdeto. Cloakware. https://irdeto.com/whitebox-cryptography/.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, August 2003. doi:10.1007/978-3-540-45146-4_27.

[Joy07] Marc Joye. Highly regular right-to-left algorithms for scalar multiplication. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 135–147. Springer, September 2007. doi:10.1007/978-3-540-74735-2_10.

[Joy19] Marc Joye. Protecting ECC against fault attacks: The ring extension method revisited. Cryptology ePrint Archive, Report 2019/495, 2019. https://eprint.iacr.org/2019/495.

[JT09] Marc Joye and Michael Tunstall. Exponent recoding and regular exponentiation algorithms. In Bart Preneel, editor, *AFRICACRYPT 09*, volume 5580 of *LNCS*, pages 334–349. Springer, June 2009. doi:10.1007/978-3-642-02384-2_21.

[Kar11] Mohamed Karroumi. Protecting white-box AES with dual ciphers. In Kyung Hyune Rhee and DaeHun Nyang, editors, *ICISC 10*, volume 6829 of *LNCS*, pages 278–291. Springer, December 2011. doi:10.1007/978-3-642-24209-0_19.

[Ker83] Auguste Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, pages 5–38, 1883.

[KI21] Yuji Koike and Takanori Isobe. Yoroi: Updatable whitebox cryptography. *IACR TCHES*, 2021(4):587–617, 2021. doi:10.46586/tches.v2021.i4.587-617.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, August 1999. doi:10.1007/3-540-48405-1_25.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, August 1996. doi:10.1007/3-540-68697-5_9.

[Lee18] Seungkwang Lee. A white-box cryptographic implementation for protecting against power analysis. *IEICE Transactions on Information and Systems*, 101(1):249–252, 2018. doi:10.1587/TRANSINF.2017EDL8186.

[LLL82] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261:515–534, 1982. doi:10.1007/BF01457454.

[LLY14] Rui Luo, Xuejia Lai, and Rong You. A new attempt of white-box AES implementation. In *IEEE International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*, pages 423–429. IEEE, 2014. doi:10.1109/SPAC.2014.6982727.

[LR13] Tancrède Lepoint and Matthieu Rivain. Another nail in the coffin of white-box AES implementations. Cryptology ePrint Archive, Report 2013/455, 2013. https://eprint.iacr.org/2013/455.

[LRD+14] Tancrède Lepoint, Matthieu Rivain, Yoni De Mulder, Peter Roelse, and Bart Preneel. Two attacks on a white-box AES implementation. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 265–285. Springer, August 2014. doi:10.1007/978-3-662-43414-7_14.

[LRH+22] Jun Liu, Vincent Rijmen, Yupu Hu, Jie Chen, and Baocang Wang. WARX: efficient white-box block cipher based on ARX primitives and random MDS matrix. *Science China Information Sciences*, 65(3):1–15, 2022. doi:10.1007/S11432-020-3105-1.

[LRT12] Victor Lomné, Thomas Roche, and Adrian Thillard. On the need of randomness in fault attack countermeasures–application to AES. In Guido Bertoni and Benedikt Gierlichs, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 85–94. IEEE Computer Society, 2012. doi:10.1109/FDTC.2012.19.

[Mes00] Thomas S. Messerges. Using second-order power analysis to attack DPA resistant software. In Çetin Kaya Koç and Christof Paar, editors, *CHES 2000*, volume 1965 of *LNCS*, pages 238–251. Springer, August 2000. doi:10.1007/3-540-44499-8_19.

[MGH09] Wil Michiels, Paul Gorissen, and Henk D. L. Hollmann. Cryptanalysis of a generic class of white-box implementations. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *SAC 2008*, volume 5381 of *LNCS*, pages 414–428. Springer, August 2009. doi:10.1007/978-3-642-04159-4_27.

[ML-23] Module-Lattice-Based Digital Signature Standard. National Institute of Standards and Technology, NIST FIPS 204 (draft), U.S. Department of Commerce, August 2023. doi:10.6028/NIST.FIPS.204.ipd.

[Mui13] James A. Muir. A tutorial on white-box AES. Cryptology ePrint Archive, Report 2013/104, 2013. https://eprint.iacr.org/2013/104.

[NS02] Phong Q. Nguyen and Igor Shparlinski. The insecurity of the digital signature algorithm with partially known nonces. *Journal of Cryptology*, 15(3):151–176, June 2002. doi:10.1007/s00145-002-0021-3.

[NS09] Phong Q. Nguyen and Damien Stehlé. An LLL Algorithm with Quadratic Complexity. *SIAM Journal on Computing*, 39(3):874–903, 2009. doi:10.1137/070705702.

[OMHT06] Elisabeth Oswald, Stefan Mangard, Christoph Herbst, and Stefan Tillich. Practical second-order DPA attacks for masked smart card implementations of block ciphers. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 192–207. Springer, February 2006. doi:10.1007/11605805_13.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, May 1999. doi:10.1007/3-540-48910-X_16.

[Pat96] Jacques Patarin. Hidden fields equations (HFE) and isomorphisms of polynomials (IP): Two new families of asymmetric algorithms. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96, International conference on the theory and applications of cryptographic techniques*, volume 1070 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 1996. doi:10.1007/3-540-68339-9_4.

[Pay] PayCore. Secure mobile payment white-box cryptography. https://paycore.com/secure-mobile-payment-white-box-cryptography/.

[PQ03] Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 77–88. Springer, September 2003. doi:10.1007/978-3-540-45238-6_7.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, May 2013. doi:10.1007/978-3-642-38348-9_9.

[PSS+17] Damian Poddebniak, Juraj Somorovsky, Sebastian Schinzel, Manfred Lochter, and Paul Rösler. Attacking deterministic signature schemes using fault attacks. Cryptology ePrint Archive, Report 2017/1014, 2017. https://eprint.iacr.org/2017/1014.

[Qua] Quarkslab. Quarks keys protect. https://quarkslab.com/quarks-appshield-keys-protect/.

[Ras20] Sandra Rasoamiaramanana. *Design of white-box encryption schemes for mobile applications security.* PhD thesis, Université de Lorraine, 2020. https://hal.univ-lorraine.fr/tel-02949394.

[RH18] Ronald Rietman and Sebastiaan Jacobus De Hoogh. Elliptic curve point multiplication device and method in a white-box context. Patent US20200119918A1, 2018.

[Riv09] Matthieu Rivain. *On the Physical Security of Cryptographic Implementations*. PhD thesis, Université du Luxembourg, 2009. https://www.matthieurivain.com/files/phd-thesis.pdf.

[Riv11] Matthieu Rivain. Fast and regular algorithms for scalar multiplication over elliptic curves. Cryptology ePrint Archive, Report 2011/338, 2011. https://eprint.iacr.org/2011/338.

[RVP22] Adrián Ranea, Joachim Vandersmissen, and Bart Preneel. Implicit white-box implementations: White-boxing ARX ciphers. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 33–63. Springer, August 2022. doi:10.1007/978-3-031-15802-5_2.

[RW19] Matthieu Rivain and Junwei Wang. Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. *IACR TCHES*, 2019(2):225–255, 2019. doi:10.13154/tches.v2019.i2.225-255.

[SA03] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 2–12. Springer, August 2003. doi:10.1007/3-540-36400-5_2.

[SAS] CryptoExperts SAS. White-box cryptography. https://www.cryptoexperts.com/technologies/white-box/.

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical computer science*, 53:201–224, 1987. doi:10.1016/0304-3975(87)90064-8.

[SE94] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66:181–199, 1994. doi:10.1007/BF01581144.

[SEMM15] Jiayuan Sui, Philip Allan Eisen, James Muir, and Daniel Elie Murdock. System and method for protecting cryptographic assets from a white-box attack. Patent CA2792787C, 2015.

[Sha99] Adi Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks. Patent US5991415A, 1999.

[SMG16] Pascal Sasdrich, Amir Moradi, and Tim Güneysu. White-box cryptography in the gray box - - A hardware implementation and its side channels -. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 185–203. Springer, March 2016. doi:10.1007/978-3-662-52993-5_10.

[SPC18] Victor Servant, Emmanuel Prouff, and Herve Chabanne. Method for electronic signature of a document with a predetermined secret key. Patent FR3066845B1, 2018.

[SWP09] Amitabh Saxena, Brecht Wyseur, and Bart Preneel. Towards security notions for white-box cryptography. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, editors, *ISC 2009*, volume 5735 of

LNCS, pages 49–58. Springer, September 2009. doi:10.1007/978-3-642-04474-8_4.

[Tha] Thales. Sentinel portfolio. https://cpl.thalesgroup.com/software-monetization/white-box-cryptography.

[Tol12] Ludo Tolhuizen. Improved cryptanalysis of an AES implementation. In Raymond N.J. Veldhuis, Luuk J. Spreeuwers, Jasper Goseling, and Xiaoying Shao, editors, *33rd WIC Symposium on Information Theory in the Benelux*, pages 68–71. Werkgemeenschap voor Informatie- en Communicatietheorie (WIC), 2012. https://ris.utwente.nl/ws/portalfiles/portal/5130869/Proceedings_33rd_WIC_symposium_2012.pdf.

[Ver] Verimatrix. Whitebox designer. https://www.verimatrix.com/products/whitebox/.

[Ver12] Vincent Verneuil. *Elliptic curve cryptography and security of embedded devices*. PhD thesis, Université de Bordeaux, 2012. https://theses.hal.science/tel-00733004.

[VMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 740–757. Springer, December 2012. doi:10.1007/978-3-642-34961-4_44.

[VRP22] Joachim Vandersmissen, Adrián Ranea, and Bart Preneel. A white-box speck implementation using self-equivalence encodings. In Giuseppe Ateniese and Daniele Venturi, editors, *ACNS 22International Conference on Applied Cryptography and Network Security*, volume 13269 of *LNCS*, pages 771–791. Springer, June 2022. doi:10.1007/978-3-031-09234-3_38.

[Wee05] Hoeteck Wee. On obfuscating point functions. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 523–532. ACM Press, May 2005. doi:10.1145/1060590.1060669.

[Whi17] CHES 2017 Challenge - WhibOx Contest, 2017. https://whibox.io/contests/2017/.

[Whi19] CHES 2019 Challenge - WhibOx Contest, 2019. https://whibox.io/contests/2019/.

[Whi21] CHES 2021 Challenge - WhibOx Contest, 2021. https://whibox.io/contests/2021/.

[Whi24] CHES 2024 Challenge - WhibOx Contest, 2024. https://whibox.io/contests/2024/.

[WMGP07] Brecht Wyseur, Wil Michiels, Paul Gorissen, and Bart Preneel. Cryptanalysis of white-box DES implementations with arbitrary external encodings. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *SAC 2007*, volume 4876 of *LNCS*, pages 264–277. Springer, August 2007. doi:10.1007/978-3-540-77360-3_17.

[XL09] Yaying Xiao and Xuejia Lai. A secure implementation of white-box AES. In *2nd International Conference on Computer Science and its Applications (CSA)*, pages 1–6. IEEE, 2009. doi:10.1109/CSA.2009.5404239.

[YJ00] Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on computers*, 49(9):967–970, 2000. doi:10.1109/12.869328.

[YKLM02] Sung-Ming Yen, Seungjoo Kim, Seongan Lim, and Sang-Jae Moon. RSA speedup with residue number system immune against hardware fault cryptanalysis. In Kwangjo Kim, editor, *ICISC 01*, volume 2288 of *LNCS*, pages 397–413. Springer, December 2002. doi:10.1007/3-540-45861-1_30.

[YKM06] Sung-Ming Yen, Dongryeol Kim, and SangJae Moon. Cryptanalysis of two protocols for RSA with CRT based on fault infection. In Luca Breveglieri, Israel Koren, David Naccache, and Jean-Pierre Seifert, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography*, volume 4236 of *Lecture Notes in Computer Science*, pages 53–61. Springer, 2006. doi:10.1007/11889700_5.

[ZBJ20] Jie Zhou, Jian Bai, and Meng Shan Jiang. White-box implementation of ECDSA based on the cloud plus side mode. *Security and communication networks*, pages 1–10, 2020. doi:10.1155/2020/8881116.

Appendix A

Generating Encodings with Bounded Walsh Transforms

Randomly drawing bijections is an inefficient method for obtaining encodings with only small values in their Walsh spectrum, particularly so when the encoding size increases (see Figure A.1). Therefore, we needed to develop an algorithm to generate permutations with bounded Walsh transforms for testing purposes. In the following section, we discuss the techniques we employed.

Let us represent an n-bit encoding by a table of $2^n \times n$ bits, where the i^{th} row is the binary representation of the encoded value of i, as in Figure A.2. In order to generate a random encoding with bounded Walsh transforms, we construct its table one column at a time. The general idea is to construct a tree for each column, with each node corresponding to a bit being set to 0 or 1. As we traverse each branch of the tree, we check the validity of the partially completed column. Based on the outcome, we either prune the branch and backtrack to the previous node or continue forward by randomly deciding another bit.

We now need to explain how to check the validity of a column. Let us start with the generation of encodings with a null Walsh spectrum, which are the most challenging to obtain. We aim for each column, representing a bit of the encoded values, to be

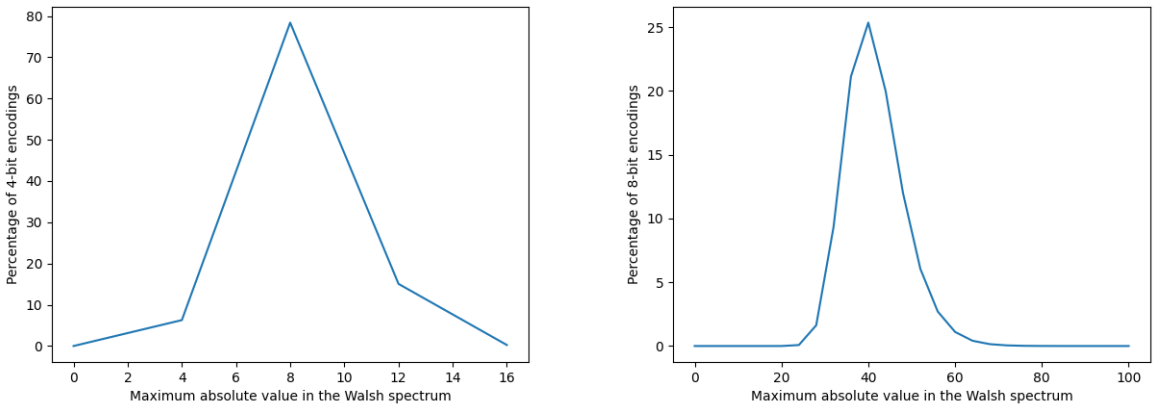


Figure A.1: Percentage of n-bit encodings having a given maximum absolute value in their Walsh spectrum over 10 000 000 random draws. The left (resp. right) figure corresponds to $n = 4$ (resp. $n = 8$). For the 8-bit case, we did not get any encoding with a maximum absolute Walsh transform less than 16.

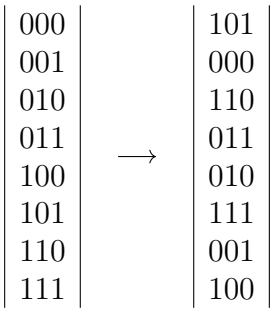


Figure A.2: Representation of the 3-bit encoding $\{5, 0, 6, 3, 2, 7, 1, 4\}$, with plain values on the left and encoded ones on the right. Note that this encoding has a null Walsh spectrum.

uncorrelated to each column of the identity permutation, representing the bits of the plain values. In other words, once the tree has been fully traversed, for each column x_i of the identity permutation and each column y_j of our n-bit encoding, we want to get $HW(x_i \cdot y_j) = 2^{n-2}$, where $0 \leq i, j < n$, and $\cdot$ denotes the coefficient-wise logical and. This requirement also applies to the columns x'_i of the identity permutation's complement. At each node of the tree for column y_j, we thus compare $HW(x_i \cdot y_j)$ and $HW(x'_i \cdot y_j)$ to 2^{n-2} for all $0 \leq i < n$. If any Hamming weight exceeds the threshold, then we prune the branch. If they are all strictly below the limit, then we randomly select another bit and follow the corresponding branch. Finally, if equality is achieved, we set the necessary bits to keep the column's validity before continuing the process.

Once the first column has been fully generated, an additional challenge arises for the subsequent ones. Since we want the encoding to be a permutation, each row must be the binary representation of a distinct value. Consequently, a combination of valid columns does not necessarily form a valid encoding. For instance, selecting the same column twice obviously leads to a dead end. Therefore, when generating a column, we take the previous ones into account and ensure that each partial row appears the correct number of times. For example, when generating the second column of a 3-bit encoding, one has to ensure that the partial rows 00, 01, 10 and 11 appear exactly twice. Experimentally, we found that when this requirement is satisfied, a number $k < n$ of generated columns seems to always be extensible to a full permutation with a null Walsh spectrum, so we never need to backtrack and modify previously completed columns.

The process for generating encodings with small non-zero Walsh transforms is very similar, the only difference being slightly modified correlation constraints with different values allowed for the Hamming weight of the columns.

Appendix B

WhibOx 2021 Attacks Summary Table

For each challenge submitted to the WhibOx 2021 contest, we show in Table B.1 which attacks were successful and give the value of the corresponding key. During the contest, we broke 92 out of 97 challenges. The 5 remaining ones have been broken a posteriori.

Table B.1: Vulnerabilities of the various challenges.

Challenge	*Hooking*	*Collision*	*Fault*	*Lattice*	*Key*
3	✓	✓	✓	✓	45189C81EADEE03202BFA06EAA15831789F0C76575508A563E1A739CA37B87BE
4	✓	✓	✓	✓	22BEF7AC4C31B2B98227D95B5EB49AF23343004CF2713FED48BEC3B5B7C3D24D
8	✓	✓	✓	✓	F484955872415A32B1B5B731EA1A8C729458055C17DC5FE9C57BCB39D1A40BFE
10	✓	✓	✓	✓	32D67733DF0D0257DA78E92752494CFD5112E303BA1413388126EA33BB60AEFC
11	✓	✓	✓	✓	E7F3287D91B528D78BF19D5E62828C845E1A4027A3E1F988B62B7407EBF5CF38
12		✓		✓	773F0C0FFACB531F50FAE0987D2B8972FE1B9231BBF46859F475BAFB45257FED
13		✓		✓	034332A23341538143FDB88F314FD942501FF8B6BA6A14D5013F1FC0984924BE
15		✓		✓	3F77C51259E1C8CC48217A66998CCF3212A17120B0FCA09163E300576DFCD9E7
16		✓			23773F0BECFACB534250FAE0987D2B8969D1AFD7EF942F148746DC73A3C6B39A
32	✓	✓		✓	32D67733DF0D0257DA78E92752494FFD5112E303BA14133FF126EA33BB60AEFC
33			✓		CD9540B70C2F92B2894594CABC4E724203A615B9144C459714758BC3CAA12242
34		✓	✓	✓	70253E6587D04D7A9A30A1461A80FCD235B28FBFC11FE8534CDFCE0A341C9257
36			✓		10D7EF92F06DF6EB94F2F344085DAD51D3A550E24A4569922460F579CB5DF11A
38			✓		70C3A9F11773C8DD795FD7942B5DB448FDFA5D12E6EC387691A19B6E523AE6AE
42			✓		1BEDDC1DD79F8856BF2E1FD66EB194073D60FEC658C5D0E2C8BAE02DC72ADF65
44			✓		B519BB44EC5BF3380CB2DF555F39ED836CDBF4961E43A66C218FADB211BF468C
45	✓	✓	✓	✓	32D67733DF3D0257DA78E92752494FFD5112E303BA14133FF126EA33BB60AEFC
50			✓		7A7AA97370B1EE16D64C71C7C5BC8C9F9456FBEA603780883399D89DA43F8A15
54	✓	✓	✓	✓	32D67733DF3D0257DA78E92752494FFD5112E22222222222F126EA33F6E49790
55	✓	✓	✓	✓	0049859485984958495 4E92752494FFD5112E2222222EE22F126EA33F6E49790
57	✓	✓	✓	✓	7D1BBD475A8EB5AF7DDB238CD8A67F86B601E0EA101C04036849B31F96CA6083
58		✓	✓	✓	BD3026C700A75B5970807802E2B47C2A892DF85E3CE57366D335EEBABCAAE255
61		✓	✓	✓	F4DDC95A88146CF52DEC752E737F8E3FB16AE4F6B7E726068946F3B0BA0C8E95
62		✓	✓	✓	0A99EB20F9DE4DD7607288B8B766F6217FE5D2CE6DDD51C6159941066AF192ED
66		✓	✓	✓	8836AC84AA148440A20628810CA65EB038BB625841275CC11590D8F5BC7F1BAC

Challenge	*Hooking*	*Collision*	*Fault*	*Lattice*	*Key*
70		✓	✓	✓	21A35C57E23B2D23ADDA19EA30325F1B532DA645489E29E47A13E92CA1F6670C
71		✓	✓	✓	588BEED930355AF54EEBAFAA46A7D26DA378A36EF5CD15D1F876D753A395F8AF
72		✓	✓	✓	B7A9B0F7661FC9A1DEC001F2C2C9EAE08748AEB187E1247726663E3DD1AB36BF
73		✓	✓	✓	4DAA29CBD634F28137499B9557104FDD36D4D4EFDFE87EFC0D8BD03555F8497F
74		✓	✓	✓	12691AAC55A079F529FE81205DF775EF297A14CA81499BF0857643E694CF8816
76		✓	✓	✓	F5178EEC7A9779E13CE01B35C8264BF32C094B172051CA32156DC61485718318
77		✓	✓	✓	A0543814F86D1C4AF6A08094CD0246F606F7E76CEE47EC052B62328038146D93
78		✓	✓	✓	511128DCBF369E985B99D07CC1668A2D28F4BA535CF7AC7926D4C5F696C3D35F
79		✓	✓	✓	595AD4C8A0EB2FDA798BC01D322F4C5ED098A2E749004B2B54FD815215F46686
80		✓	✓	✓	8E938EA9BE9E51A28DFD30BD6EDB9D6765C1272B8F7048CE81021194759C3E52
81		✓	✓	✓	F134975C5A989635F1D9FA7469C848A953622E9DA1BED7E12455DCD2AFA070BE
84		✓	✓	✓	36A990B9F35B79934FB25C64681DE3A83FC178DC2383C585FFCFDDD7C1F6C2B7
85	✓	✓	✓	✓	AB700D75274336FD26A1FE49D400ACEAE89F0FDBFE4BDE9A70373CA693003CA8
87		✓	✓	✓	9A4D4A94A1FE0FA1C559764C85D06496BD752498E0B5A2459624211013B9A088
89		✓		✓	C80682FCB2D78B2515A70A70D17C47A8512E24A127E797C073566D54586B9482
94		✓		✓	A04B6199A1DFE39EF35F6302454D71C872771A2F02A27AB5EC8130DA226F6F90
96		✓		✓	AFAAABE59B2EBB4FE15274E4EB5D1999C0554CC2D498BC92C59A3F6CD8FE2BC0
97	✓	✓	✓	✓	0754CA8EA936675EC3F64782A14E1A75B3D357044D4B2C434C6011279D17E829
100			✓		7F58EDB783C1F3FA7FF424CF7F5DF6D4BCDCF18D8A98CE4559EC22EB17030578
101			✓		25D31D3AFF5773799ECF43DEC1882B8F05D9231697BDDA5482DE05B14FB8A63B
103		✓		✓	CC977E0748722D615B845C1B10EA554B69DFCA640440CA5C468BBEF84B8C0442
104		✓	✓	✓	638C9DFBF9F376CBB3E3B01DF27960EC53A689D2FF4DFF23D97EE5351ED4A3D0
105		✓	✓	✓	D29E9D130016D930BF830BCAD071BC6503F877FB207922A9E495CF71A79631FE
107		✓	✓	✓	4E420B6AA9E9F07F19CF7ED97497871C1223BC2A68E83716575C235DE6D63E17
108			✓		60609404F0B9086D3A995AF0680D048724CF2B1AF2B33CEA8DD4AF4B62A5DDBB
114	✓		✓	✓	0000000000000000000000000000000000000000000000000000000000000005
127			✓		1144D82B9568581405D10CF8B219FF7E94E4559E0832B06056F1F87D43C75777
135	✓	✓	✓	✓	0C2A5692FE1A7F9B8EE7EB4A7CD59CD62BCE33576B3123CECBB6406837BF51F5
136	✓		✓	✓	0C2A5692FE1A7F9B8EE7EB4A7CD59CD62BCE33476B3123CECBB6406837BF51F4
139	✓		✓	✓	000000000000000000000000000000004319055358E8617B0C46353D039CDAA9
153	✓			✓	9C29EDDAEF2C2B4452052B668B83BE6365004278068884FA1AC3F6D0622875C3
157	✓	✓	✓	✓	F04DBFD1147F9D43747538C1C9256DD2BC20562F9D92B83E9AFA751299B160A4
165		✓	✓	✓	84DAF8B6620FC6669BF1EE264D1B214A4FBECACEADDFDC0DCBC89CF4B6E3232B
166	✓		✓		C746740A4A6BCBD462D9041023A0FEF5CCF0328FF80D9C50132682030D77D33C
172		✓	✓	✓	285E57F7BDDAAA6201D8870A0B9B168C7A5D8200085F62504EE3EBFCC11EF150
174	✓	✓	✓	✓	9C29EDDAEF2C2B4452052B668B83BE6365004278068884FA1AC3F6D0622875EC
185	✓	✓	✓	✓	7729EDDAEF2C2B4452052B668B83BE6365004278068884FA1AC3F6D0622875EC
187	✓	✓	✓	✓	7779EDDAEF2C2B4452052B668B83BE6365004278068884FA1AC3F6D0622875EC
192			✓		09302BDFA5313312B9A665316F7E9365DCC57DA7E21FD8612CDCD553BABB51FE
193			✓		E0FE06BE0684455EDD2F5134A3AE8B9F6852561C821672FA16606986233BF811
209				✓	6E3A09F8EC613B8A524F7608CB80B2D3C510E27506AD84FA14C3B6D018E659F7
212			✓		D663E156F036F11D4E73CC0EC09A952DEAED316947DF73EB28467EC623C5740D

Challenge	*Hooking*	*Collision*	*Fault*	*Lattice*	*Key*
226[1]					6F1D9093F3D5AE7C5F133659295914C9AF22E54B4ADE38CA421CA9BBD3D48A50
227	✓			✓	ADA6C6A1049825989811C9495D83681A68C67AB5E8EBDDC126CEE77056A7BB27
228				✓	EA7BA345EB9D99F54261D01AE6319B184769E5745621706D77018E0DB46DDAFA
231	✓	✓	✓	✓	8ADE24EE6413C6E408784DBB4D81D04F33238AB503CBE35C77400517EE5ABC96
235	✓	✓	✓	✓	00000000000000000000000000000000000000000000000000D0FACADE0DEFACED
251		✓	✓	✓	DDE098A74086ECBB4DBA1848511BEA924145D1A9ED2EC9E64E0C5934BAAC97AE
253		✓		✓	B22DB44C9E66D567B3B2CBB3C720309D1EEAD38717017F5E79F05274F289A52C
256			✓		F1662664E7E303740C0CA3927F9870A789978DAE95892302E73C85E3993B4CC9
261			✓	✓	3266C9F6379DFDAE4AA763E8E6BA94526504CA364C482306829D4BF1E97BFF92
262			✓		A0F00DCAA5DAB169FD4DFE2186BCBCBD22631AB68BFEFF1FC19306174EAF8970
264				✓	D0EE17829A397C18074EA3888057AE815B5336773F9668E6CE4464D4B2B05F1F
267	✓	✓	✓	✓	C17536B60BCF94326A9C8CA17E0FC4EDBD76822532B350E8237CA2D8CF9C74B0
274	✓	✓	✓	✓	0080ECD2A00080ECD2A00080ECD2A00080ECD2A00080ECD2A00080ECD2A00080
283			✓		79FE8D884DC2F7440824DE79C9F7C513C2B4549631D343523C73CB8F85983A4F
299	✓	✓	✓	✓	3A0F803A874CD5B826023F2073FF200371D399E76E66B05E1241AA787B0564D6
304			✓		EE8942A527CA1A58B8A8EA369441CB8518836DDB98F6380B8008B6053BC8182C
305			✓		311EA92FBCDD3C6A29D269589A9E71F13A231FFEC85FF36B398967EC9934805E
307	✓		✓		FA3FCDE70679E7E44391F7157E2B5822F5B9B9C93ADD95C2BA90FF4B95C8A6BB
308			✓	✓	84CCCAA904CB397F41A36FF9E05D4EB6C58B8E203E02373C465B6C3F03280C82
314				✓	7E045DB89DD77BD6B2EAF23172A89A656B5084748642DB82BBAE931E737560C2
320	✓	✓		✓	D235C2B1D089F158A0AE4E7799C2DCA9985E3D44C8F243BAD8B5E1A4EB647E1B
321	✓	✓	✓	✓	BA15757E1B0DB122F349C0C50C97071A4CFFF4FD2875B4A092FBDD985E8595DE
323	✓	✓	✓	✓	C7491BBC530FFA9DDCF3E7D732536FACF04239693D549C50DDAD41931A6244C2
325			✓	✓	6902CD65AE124A45B9DD16BAEFD26D9CFFB5C291DC1E256D9CCE17BE3CF11775
327				✓	2BC6F2467C7F8DFA164EDC68DDCF65E795B8A2153182565481D8D6878D80EA81
328				✓	37170CF851A89AAFD3511234BE2B96C89B783A44D7A6C22E9A150872809F7CDF
335			✓		EC90CC12DC70E3C5A7D47B6083A988F3F6C6B2B63EB0D8991F84B19E21ACC061
336[1]					D2E2AE325946DDADC9A67A2DFE8EE74065D8D39968707F5D818D7B62910894EA
345			✓		3266C9F6378DFDAE4AA763E9166B131E6514CA364C482306829D4BF1E97BFF92
346			✓		5EE43950837D0ABA419FE5B586D1A7AA44DDAAC6327DADC3133F18A850211B9F

[1]This challenge has not been broken by our automatic tools so we have used reverse engineering techniques.