Design of white-box encryption schemes for mobile applications security

Sandra Rasoamiaramanana

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Conception de schémas de chiffrement boîte blanche pour la sécurité des applications mobiles.

THÈSE

présentée et soutenue publiquement le 12 juin 2020

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Sandra RASOAMIARAMANANA

Composition du jury

Rapporteurs :	Louis Goubin	Professeur, Université de Versailles-St-Quentin-en-Yvelines
	Caroline Fontaine	Directrice de recherche, CNRS, ENS Paris-Saclay
Examinateurs :	Véronique Cortier	Directrice de recherche, CNRS
	Brice Minaud	Chargé de recherche, Inria, ENS Paris
Directeurs de thèse :	Marine Minier	Professeure, Université de Lorraine
	Gilles Macario-Rat	Chercheur, Orange Labs
Présidente :	Véronique Cortier	Directrice de recherche, CNRS

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Acknowledgments

"*L'invention, c'est le progrès d'une pensée qui change au fur et à mesure qu'elle prend corps. C'est un processus vital, quelque chose comme la maturation d'une idée.*" - Bergson.

I'd like to start this acknowledgment section with this quote from Bergson. The purpose of research is sometimes to invent, sometimes to discover, but always to enable progress. I will not pretend to say that I have advanced science, but I have advanced my knowledge of a field and my way of approaching a problem. This was made possible by the richness of the encounters I had during the three years of my thesis.

First of all, I would like to thank Marine Minier, my thesis supervisor, who has been an unfailing supporter and from whom I have learned a great deal. I thank you for your advice, the hours of work and your involvement in your role as thesis supervisor. Secondly, I would like to thank Gilles Macario Rat who supervised me at Orange. You are admirable for your courage and intelligence, and I would probably not be here if you hadn't given me the opportunity to do an internship at Orange in the spring of 2016. I also thank my rapporteurs, Caroline Fontaine and Louis Goubin, for doing me the honour of reporting my thesis in a difficult health context. Thank you for your helpul comments which will surely help me to progress. Thanks to Véronique Cortier and Brice Minaud for having accepted to be part of my thesis jury.

Secondly, I would like to express my deep gratitude to Emmanuel Thomé and Sébastien Allard for welcoming me to their respective teams, Manu in the CARAMBA team at LORIA and Sébastien in the Systems and Devices Security team at Orange Labs. These two teams allowed me to see two worlds with each a different point of view of research. Thanks also to Nicolas Demassieux, Adam Ouorou and in particular to Jean-Philippe Wary for the exchanges rich in ideas with a special mention to Nicolas' pragmatism.

In these difficult moments, both related to the lockdown and the stress of the end of my thesis, I have a special thought for all those people who brightened my life: to Paul H for the numerous discussions and for having initiated me to climbing, to Simon M for the Aussois-Lyon trip, to my co-office at LORIA, Aude and Gabrielle, to Alicia who gave me precious advice and proofread this thesis, to my former open space neighbors, Alex who was a great help at the beginning of my thesis, Mehdi and Xiao, to Paul C and Kahina for the household scenes (just kidding).

I also remember those coffee room discussions that were sometimes polemical, often funny but never mean. And then there are those people you only meet for a short time but who remain in your memory for their personality, their humour: thanks to Mayeul, Machid, Yixin, Mélissa, Diane, Zakaria, Svyat, Simon A. I would also like to thank all the CARAMBA because thanks to them I keep very good memories of each of my visits to Nancy.

Finally, I would like to express my sincere thanks to all the people who allowed me to progress thanks to their feedback: to Paul H and Jérémie who proofread and corrected my articles, to the anonymous reviewers, to Florent Retrain for the details on image processing and to Brice Minaud for his perspective on the affine equivalence problem. I agree with Malcolm X when he says: "*If you have no critics you'll likely have no success*".

To definitively close this section, I would like to thank all my family for their support and in particular my mother. Neslon Mandela once said "*Courageous people do not fear forgiving, for the sake of peace*". So thank you Mom for your courage.

"Success is not final, failure is not fatal:
it is the courage to continue that counts."

Winston Churchill

To my mother, without whom I would be nothing.

Résumé

Aujourd'hui les équipements mobiles font partie intégrante de nos vies avec le développement des applications. En plus des téléphones intelligents, d'autres équipements comme les objets connectés peuvent avoir à manipuler des données qui doivent rester secrètes. Par exemple, l'authentification d'un objet connecté dans un réseau nécessite l'existence d'un "secret" détenu par l'objet. Dans le cas des applications mobiles, l'apparition des applications de paiement permettant un paiement sans contact à partir du téléphone ou encore les applications bancaires pose de sérieux enjeux de sécurité. Le besoin de sécurisation des applications est ainsi primordial à la fois pour un utilisateur désireux d'accéder à un service sans risquer ses biens mais aussi pour les fournisseurs de services qui y ont un intérêt financier. Ainsi, la cryptographie est utilisée pour protéger ces diverses applications mobiles. Dans ce contexte, on souhaite répondre à ce besoin avec une approche à la fois logicielle et matérielle pour sécuriser la cryptographie sur des plateformes ouvertes et exposées.

Le but de cette thèse est de vérifier la sécurité des implémentations logicielles d'algorithmes cryptographiques dans le modèle boîte blanche et de proposer des techniques permettant de renforcer cette sécurité dans un environnement mobile. Le modèle boîte blanche s'oppose au modèle boîte noire traditionnel et décrit un contexte dans lequel un attaquant contrôle l'environnement d'exécution et a accès aux implémentations logicielles. Une fois qu'une clé secrète est révélée, la sécurité du schéma de chiffrement n'est plus valable. Dans ce contexte, la dernière ligne de défense est l'implémentation elle-même : la clé secrète est cachée dans le code de manière à ce qu'elle ne puisse être ni identifiée, ni extraite. De nombreuses études ont été menées sur la cryptographie en boîte blanche et ont abouti à des propositions d'implémentations d'algorithmes standardisés tels que le DES (Data Encryption Standard) ou l'AES (Advanced Encryption Standard). Ces algorithmes intéressent particulièrement du fait de leur large déploiement. Malheureusement ces propositions ont révélé des vulnérabilités et ne garantissent pas la confidentialité de la clé secrète. Dans cette thèse, nous nous intéressons premièrement aux raisons pour lesquelles toutes les propositions d'implémentations ne permettent pas de "cacher" suffisamment la clé secrète. Nous ferons une étude détaillée des techniques utilisées ainsi que des attaques possibles. Deuxièmement, nous proposons des techniques nouvelles permettant de contrecarrer ces attaques et étudions le coût de ces techniques en termes de taille et de performance de code.

Une autre approche en cryptographie en boîte blanche est de concevoir des algorithmes dont on peut prouver la résistance à l'extraction de clé. Cette nouvelle approche implique de proposer des notions de sécurité adaptées au modèle boîte blanche. En particulier, le problème principal est de garantir que l'implémentation ne puisse être copiée et exécutée dans un autre environnement. Cette attaque appelée "code lifting" (copie de code) est équivalent à extraire la clé secrète. Une solution proposée dans la littérature est d'augmenter la taille du code afin d'augmenter la complexité en espace de l'attaque. Nous proposons une solution à ce problème en définissant un schéma de chiffrement que l'on peut implémenter en boîte blanche et qui utilise un dispositif physique appelé Physically Unclonable Function (PUF). Une PUF désigne un dispositif physique présentant des caractéristiques uniques et inclonables permettant de l'identifier. Ainsi, une PUF peut être vue comme l'empreinte (au sens biométrique) d'un appareil. Ce dispositif sera utilisé dans notre schéma comme moyen d'identifier l'environnement d'exécution d'un algorithme cryptographique et permettra de générer une clé spécifique à un appareil donné.

Abstract

Today mobile devices are an integral part of our lives with the development of applications. In addition to smart phones, which are increasingly powerful, other devices such as connected objects may have to handle data that must remain secret. For example, the authentication of a connected object in a network requires the existence of a “secret" held by the object. In the case of mobile applications, the emergence of payment applications allowing contactless payment from the telephone or banking applications poses serious security challenges. The need to secure applications is therefore essential both for users wishing to access a service without risking their goods and for service providers who have a financial interest in it. Thus, cryptography is used to protect these various mobile applications. In this context, we wish to meet this need with both a software and hardware approach to secure cryptography on open and exposed platforms.

The aim of this thesis is to verify the security of software implementations of cryptographic algorithms in the white-box model and to propose techniques to reinforce this security in a mobile environment. The white-box model or white-box attacks context is opposed to the traditional black-box model and refers to a context in which an attacker controls an execution environment and has access to software implementations of cryptographic algorithms. Once a secret key is revealed, the security of the encryption scheme is no longer valid. In this context, the last line of defense is the implementation itself: the secret key is hidden in the code so that it cannot be distinguished or extracted. Many studies have been conducted on White-Box Cryptography and have led to proposals for white-box implementations of standardized algorithms such as the DES (Data Encryption Standard) or the AES (Advanced Encryption Standard). These algorithms are of particular interest due to their wide deployment. Unfortunately these proposals have revealed vulnerabilities and do not guarantee the confidentiality of the secret key. In this thesis, we are first interested in the reasons why not all proposed implementations allow to “hide” the secret key sufficiently. We will make a detailed study of the techniques used as well as the possible attacks. Secondly, we propose new techniques to counter these attacks and study the cost of these techniques in terms of code size and performance.

Another approach in White-Box Cryptography is to design algorithms that can be proved to be resistant to key extraction. This new approach involves proposing security notions adapted to the white-box model. In particular, the main problem is to ensure that the implementation of the cryptographic algorithm cannot be copied and executed in another environment. This attack called "code lifting" (code copying) is equivalent to extracting the secret key. One solution proposed in the literature is to increase the size of the code in order to increase the space complexity of the attack. We propose a solution to this problem by defining an encryption scheme that can be implemented in a white-box and that uses a physical device called Physically Unclonable Function (PUF). A PUF refers to a physical device with unique and unclonable characteristics that can be used to identify it. Thus, a PUF can be seen as the fingerprint (in the biometric sense) of a device. The PUF will be used in our scheme as a means of identifying the execution environment of a cryptographic algorithm and will generate a key specific to a given device.

Résumé étendu

Organisation du manuscrit

Les travaux effectués pendant ma thèse portent globalement sur la cryptographie en boîte blanche et plus précisément sur ses applications pour la sécurité mobile. Ce manuscrit se décompose en deux parties. La première partie présente un état de l'art détaillé de la cryptographie en boîte blanche en décrivant toutes les approches présentées dans la littérature. Cette partie donne une idée des motivations à considérer un nouveau modèle pour définir la sécurité des algorithmes cryptographiques. A la fin de ce chapitre le lecteur pourra se convaincre de l'intérêt du domaine et aura une meilleure compréhension des différentes techniques possibles pour mettre un algorithme cryptographique en oeuvre dans le modèle boîte blanche. En particulier, la première partie permet de comprendre les approches présentées dans la seconde partie du manuscrit.

La deuxième partie regroupe, en effet, l'ensemble des contributions. Les deux premiers chapitres portent sur l'utilisation de la cryptographie en boîte blanche dans des applications mobiles (applications mettant en oeuvre la gestion des droits numériques ou l'émulation de cartes hébergées). Le dernier chapitre porte sur une implantation boîte blanche de l'AES proposée à la compétition WhibOx 2019.

Modèle boîte blanche

La cryptographie en boîte blanche a été introduite pour répondre à un problème rencontré dans le contexte de la gestion des droits numériques (Digital Rights Management) pour la distribution de contenus média : la redistribution de clés. Avec le déploiement d'Internet et la multiplication des contenus médias (vidéos, musiques, livres électroniques) à la demande, les logiciels mettant en oeuvre la gestion des droits numériques sont téléchargés sur des appareils ouverts tels que les PC ou les téléphones portables. Etant donné que l'environnement de déploiement n'est pas sous le contrôle des fournisseurs de contenus, un utilisateur malveillant peut rétroconçevoir le logiciel et retrouver la clé secrète qu'il peut distribuer illégalement.

Depuis quelques années, de nouveaux cas d'usage de la cryptographie en boîte blanche ont émergé : l'émulation de cartes hébergées (Host Card Emulation) et la signature numérique. La technologie HCE permet au système d'exploitation d'un téléphone portable de piloter la communication en champ proche (Near Field Communication). Le NFC est typiquement utilisé pour effectuer des paiements sans contact à partir de son mobile. Ainsi, grâce au HCE, une application mobile installée dans le système d'exploitation peut gérer des données sensibles. Par exemple, pour une application émulant une carte bancaire, il s'agit du Primary Account Number (PAN) et de clés cryptographiques. Alternativement, le HCE permet de stocker les données sensibles dans le "cloud" et dans ce cas, l'application mobile doit garantir la sécurité de l'authentification permettant d'accéder au "cloud".

Ces deux applications ont un point commun : l'ouverture de la plateforme d'exécution et une implantation logicielle des algorithmes cryptographiques. En effet, les systèmes d'exploitation peuvent héberger des applications malveillantes et ainsi les clés cryptographiques sont exposées.

Une solution proposée au problème de stockage sécurisé de clés cryptographiques dans des "environnements hostiles" est de cacher la clé dans le code de l'algorithme. On parle d'implantation boîte blanche car l'implantation doit résister à un attaquant en boîte blanche. Cet attaquant contrôle l'environnement d'exécution et a accès à l'implantation (par rétro-conception de l'application par exemple). Ce modèle d'attaquant caractérise le modèle boîte blanche.

L'utilisation de "boîte blanche" fait référence aux modèles boîte noire et boîte grise déjà connus. Traditionnellement, la sécurité d'un cryptosystème est évaluée dans un scénario d'attaque en boîte noire. Cela signifie que l'attaquant n'a aucun accès physique à la clé ni aux détails

internes liés à l'exécution du cryptosystème. Il peut seulement observer les entrées et les sorties qui sont constituées des messages en clair ou chiffrés.

Le modèle boîte grise a fait son apparition dans les années 90 avec le développement des cartes à puce. Dans ce modèle, l'attaquant a accès à des informations sur le fonctionnement interne du cryptosystème et qui sont dépendantes de l'implantation. Ces fuites d'information sont observées par le biais du temps d'exécution, de la consommation électrique ou des rayonnements électromagnétiques du dispositif. Ce modèle d'attaque regroupe les attaques dits "par canaux auxiliaires".

Ce chapitre décrit brièvement les attaques typiques du modèle boîte grise qui ont inspiré les attaques boîte blanche, notamment l'analyse différentielle de consommation électrique (Differential Power Analysis) qui a inspiré l'analyse différentielle de calculs (Differential Computation Analysis) et l'analyse différentielle de fautes. Trois autres attaques sont présentées : l'attaque par entropie, l'attaque par suppression de boîte S et l'attaque BGE qui sera détaillée dans le second chapitre.

Etat de l'art sur les implantations boîte blanche de l'AES

AES (Advanced Encryption Standard [101])est le standard de chiffrement approuvé par l'institut national américain des normes et des technologies (NIST) pour remplacer le précédent standard, le DES (Data Encryption Standard). Son utilisation dans de nombreuses applications (entre autres la DRM) a justifié la recherche d'une implantation boîte blanche. La première implantation a été décrite par Chow et al. [35] en 2002 et s'appuie sur des techniques de tabulation et randomisation. L'idée est de décrire l'algorithme par un réseau de tables de correspondance qui dissimulent le secret. Les tables sont randomisées afin de ne pas révéler d'information sur la clé. Cette approche suppose une clé fixée qui permet de pré-calculer les tables de correspondance. Si l'attaque par suppression de boîte S ne peut plus extraire la clé, l'attaque par entropie permet toujours de distinguer toutes les tables à forte entropie. Cela pose une question : la sécurité du cryptosystème est-elle garantie si les tables utilisées pour le chiffrement peuvent être distinguées et extraites de la mémoire? Cette question pose le second problème qui doit être abordé par la cryptographie en boîte blanche, à savoir se prémunir de la copie de code. Une solution à ce second problème peut être l'utilisation d'encodages externes (des transformations qui modifient l'entrée et la sortie de l'algorithme). Ainsi, si les données correspondant au chiffrement sont copiées, sans la connaissance des encodages externes, le chiffrement n'est pas fonctionnel.

L'article de Chow et al. ne fournit cependant aucune preuve formelle de la sécurité d'une implantation boîte blanche. L'attaque BGE [12] publiée en 2004 ne laisse plus aucun doute sur l'insécurité de l'implantation. En particulier, l'attaque montre que la structure de l'AES ainsi que les encodages utilisés pour randomiser les tables jouent un rôle décisif dans le succès de l'attaque. Il s'en est suivi plusieurs travaux sur des implantations boîte blanche de l'AES explorant tous les encodages possibles (linéaires, affines, non-linéaires) mais aucune n'a démontré une sécurité suffisante. En outre, l'attaque générique de Michiels et al. [89] montre que toute implantation boîte blanche d'un algorithme de type réseau de substitution-permutation utilisant la technique de tabulation et randomisation a une sécurité bornée.

Déclinaison des attaques en boîte grise pour le modèle boîte blanche

Par définition, un attaquant en boîte blanche a accès à l'implantation boîte blanche et peut exécuter le programme autant de fois qu'il le souhaite. Ainsi, sans connaître les détails de l'implantation, il est en mesure d'exploiter des données de calculs (des adresses mémoire ou des

données lues ou écrites) pour distinguer la clé secrète. En effet, ces données sont généralement corrélées à une partie de la clé. L'analyse différentielle de calculs est une attaque inspirée l'analyse différentielle de consommation électrique. L'attaque nécessite l'usage d'outils d'instrumentation dynamique de binaire permettant d'enregistrer des valeurs ou des adresses lors de l'exécution de l'algorithme. Ces données enregistrées sous forme de traces d'exécution sont ensuite traitées avec des outils statistiques pour déterminer la clé la plus probable.

La puissance de l'attaque a été révélée par la compétition WhibOx 2017 qui avait pour objectif l'évaluation publique d'implantations boîte blanche propriétaires. En outre, la compétition a permis de démontrer mathématiquement que les encodages utilisés ne permettent pas de masquer suffisamment les corrélations. Une seconde attaque redécouverte pendant la compétition est l'analyse différentielle de fautes. Cette attaque invasive appartenant aux attaques en boîte grise consiste à altérer la procédure de chiffrement en modifiant une ou plusieurs variables intermédiaires et en exploitant la différence entre les chiffrés correspondants. Si l'attaque est coûteuse à mettre en oeuvre sur une carte à puce, elle est bien plus abordable sur une implantation boîte blanche. En effet, elle requiert uniquement un outil d'instrumentation dynamique de binaire et l'accès au chiffré.

Nous décrivons des contre-mesures typiques aux attaques par DCA et DFA. Une contre-mesure possible à l'attaque par DCA est le masquage. Cependant, un simple masquage de la sortie de SubBytes n'est pas suffisant lorsque la cible de l'attaque est la sortie de MixColumns. Sur une implantation utilisant des tables de correspondance, la technique de masquage est limitée. Une nouvelle approche est apparue pendant la compétition et permet d'intégrer des schémas de masquage d'ordre supérieur. Il s'agit d'une implantation sous forme de circuit, c'est-à-dire que l'algorithme est décrit par des fonctions booléennes représentées par des circuits booléens. Pour ce type d'implantation, des variantes de l'attaque par DCA sont décrites.

La cryptographie en boîte blanche en théorie

L'approche consistant à décrire la cryptographie en boîte blanche comme une solution pratique pour des applications industrielles s'est révélée lacunaire. En effet, aucune preuve de sécurité n'est proposée. Une formalisation des propriétés attendues est nécessaire pour établir les preuves de sécurité (une sécurité calculatoire par réduction par exemple). Une première piste à cette formalisation est l'article de Barak et al. [7] sur l'obscurcissement (obfuscation) de programmes. La cryptographie en boîte blanche étant une technique particulière d'obscurcissement spécifique aux algorithmes cryptographiques. Si les travaux de Barak et al. ont démontré l'inexistence d'un programme d'obscurcissement générique, ils ne disent rien su l'existence d'un tel programme pour une famille spécifique, par exemple, la famille des algorithmes de chiffrement par bloc. Ces travaux ont d'ailleurs inspiré Saxena et al. [108] sur un modèle théorique de cryptographie en boîte blanche. Ils définissent la notion de White Box Property (WBP) comme étant la propriété d'un programme d'obscurcissement (obfuscateur) qui génère un programme obscurci ne donnant aucun avantage à un attaquant en boîte blanche par rapport à un attaquant en boîte noire. En d'autres termes, si un programme vérifie une certaine notion de sécurité et que l'obfuscateur vérifie la WBP alors le programme obscurci vérifie la notion de sécurité.

Bien que ces travaux ouvrent la voie à une possible formalisation de la cryptographie en boîte blanche, ils tendent à s'éloigner de son aspect pratique. Ce chapitre décrit les notions de sécurité introduites en 2013 par Delerablée et al. [43] et qui ont été repris par plusieurs travaux dont les nôtres. Les notions sont plus proches des propriétés attendues d'un programme boîte blanche. Quatre notions sont décrites : unbreakability (propriété à être incassable), incompressibility (propriété à être incompressible), traceability (propriété à être traçable) et one-wayness (propriété à

être irréversible). La première notion décrit simplement la difficulté à extraire une clé secrète d'un programme boîte blanche; la seconde décrit la difficulté à compresser le programme, c'est-à-dire à réduire les données du programme en conservant sa fonctionnalité; la troisième notion exprime la possibilité de tracer des programmes boîte blanches en particulier pour des applications de gestion des droits numériques; enfin, la dernière notion décrit la difficulté à inverser un programme boîte blanche, c'est-à-dire à trouver un programme qui exécute l'inverse de la fonction implantée. Les deux premières notions sont utilisées pour fournir une preuve de la sécurité d'algorithmes cryptographiques spécialement conçus pour le modèle boîte blanche. Nous décrivons tous les algorithmes conçus dans ce modèle, par exemple Space, SPNbox, WhiteBlock. En fonction des constructions, la notion d'incompressibilité est parfois renommée "weak white-box security" ou "space-hardness" mais l'idée derrière chaque notion est la même, notamment de s'assurer qu'il est difficile de trouver un programme utilisant moins de données que le programme boîte blanche et ayant la même fonctionnalité.

Cryptographie fondée sur une fonction physiquement inclonable

Lorsqu'un algorithme conçu pour le modèle boîte blanche est mis en oeuvre en logiciel, l'extraction de la clé secrète est par définition calculatoirement impossible. En revanche la copie de code est toujours possible malgré la propriété d'incompressibilité. Dans ce chapitre nous décrivons une nouvelle approche pour résoudre ce problème. Nous proposons une technique apparentée au "node locking" pour verrouiller un programme boîte blanche. Nous proposons d'introduire un dispositif physique appelé fonction physiquement inclonable (PUF) pour verrouiller le programme à son environnement d'exécution. L'objectif est alors de s'assurer qu'un attaquant ne peut exploiter la fonctionnalité du programme sans avoir accès à la PUF. Une PUF est caractérisée par les imperfections introduites aléatoirement dans un objet pendant le processus de fabrication et qui sont de fait impossible à reproduire. On dit que la PUF est inclonable. Une PUF est généralement caractérisée par un ensemble de défi-réponse qui décrit une fonction aléatoire. Ainsi, si un programme boîte blanche est verrouiller sur un appareil grâce à une PUF, un attaquant ne pourra exécuter le programme sur un appareil différent qui ne dispose pas de la même PUF.

Une première contribution décrite ici est la définition d'un schéma de chiffrement utilisant une PUF et la formalisation de la notion de verrouillabilité en s'appuyant sur le formalisme utilisée pour décrire les propriétés d'une PUF. La propriété de verrouillabilité se décline en deux notions de sécurité : la verrouillabilité liée à l'imprévisibilité de la PUF et la verrouillabilité liée à l'inclonabilité de la PUF. Nous démontrons alors que si la probabilité de prédire une paire valide de défi-réponse (respectivement, de cloner la PUF) est négligeable alors le schéma satisfait la première notion (respectivement la seconde notion) de verrouillabilité.

Le seconde contribution présentée dans ce manuscrit est la construction d'une fonction de hachage universelle à partir d'un algorithme de chiffrement par bloc. La fonction de hachage est presque uniforme et AXU (Almost XOR Universal) et peut être utilisée pour définir un algorithme de chiffrement paramétrable d'Even-Mansour. Cette fonction de hachage est instanciée avec un algorithme conçu pour le modèle boîte blanche, en l'occurrence SPNbox-8. Ceci permet de décrire une implantation boîte blanche du chiffrement paramétrable satisfaisant la propriété d'incassabilité. Enfin, l'implantation boîte blanche est verrouillable par une PUF.

Enfin, la dernière contribution à cette section est la recherche d'une PUF sur des appareils grand publics tels que les téléphones portables modernes. En particulier, une PUF peut être caractérisée à partir de la caméra de l'appareil. Cependant, la PUF caractérisée à partir de la caméra est une PUF faible, c'est-à-dire qu'elle n'a que peu de paires défi-réponse possibles. La caractérisation d'une PUF forte, avec un nombre élevé de paires défi-réponse, sur un téléphone

portable reste un problème ouvert.

Ces travaux seront présentés à SECRYPT 2020 (17th International Conference on Security and Cryptography).

Traçage de traîtres

Traditionnellement, dans un système de diffusion chiffrée, chaque utilisateur dispose d'une boîte de déchiffrement (une carte à puce ou un logiciel) qui embarque une clé secrète. Cette clé permet à l'utilisateur de déchiffrer une clé de contenu chiffrée qui est ensuite utilisée pour déchiffrer le contenu. Ainsi, deux données chiffrées sont diffusées à tous les utilisateurs : la clé de contenu chiffrée et le contenu chiffré. Un des problèmes majeurs qui se pose dans ce type de système est la redistribution illégale de la clé secrète. Le concept de traçage de traîtres a été introduit pour prévenir cette fraude. Elle consiste à définir une procédure de traçage qui permet d'identifier une ou plusieurs clés utilisées par un décodeur pirate. Il existe trois type de schémas de traçage de traîtres : les schémas combinatoires, les schémas à clé publique et les schémas utilisant un code.

Nous décrivons un schéma de traçage de traîtres que l'on peut implanter en boîte blanche. D'une part, on garantit que le programme boîte blanche ne révèle pas la clé secrète et d'autre part, qu'un attaquant qui a accès à plusieurs programmes boîte blanche ne peut produire un programme pirate intraçable. Dans le système que nous proposons, le déchiffrement de la clé de contenu est remplacé par la génération de la clé à l'aide d'un générateur. Ainsi, ce n'est plus la clé de contenu chiffrée qui est diffusée mais une information permettant de générer la clé. Afin de permettre le traçage, chaque utilisateur reçoit une version personnalisée du générateur de clés. Le générateur de clés d'origine (utilisé par le diffuseur) fait appel à plusieurs tables de correspondance calculées à partir d'une clé secrète et décrivant chacune une permutation. La version personnalisée du générateur est obtenue en modifiant les tables d'origine en fonction d'un mot de code appartenant à un code de Tardos et identifiant l'utilisateur. Un code de Tardos est un code binaire introduit pour laisser une marque dans les documents numériques. Il a la propriété de résister aux collisions, c'est-à-dire qu'un document contrefait embarque suffisamment d'information pour retrouver au moins un des documents ayant servi à la contrefaçon. Nous montrons qu'en fonction des paramètres choisi pour définir le système, il est possible de garantir la traçabilité des programmes boîte blanche de génération de clés.

Nos travaux ont été présentés à SecITC 2019 (International Conference on Information Technology and Communications).

Implantation boîte blanche de l'AES

Une contribution majeure de ma thèse est une implantation boîte blanche de l'AES. Malgré les nombreux échecs des différentes tentatives, ce sujet reste attractif du fait de ses implications. Dans ce chapitre, je décris les travaux qui ont abouti à la soumission d'un candidat à la compétition WhibOx 2019. Notre approche combine deux approches déjà connues à savoir l'utilisation d'un réseau de tables de correspondance et l'utilisation de systèmes de polynômes multivariés. Ainsi, la partie non-linéaire de l'algorithme (SubBytes) et AddRoundKey sont représentés par des tables de correspondance tandis que la partie linéaire constituée par MixColumns et ShiftRows est représentée par des systèmes de polynômes multivariés à coefficients dans $\mathbb{F}_2$. Afin de garantir la sécurité locale de chaque table et de se prémunir de l'attaque BGE, un encodage quadratique est utilisé. En fait, la variable sensible est masquée au moyen de polynômes quadratiques aléatoires, puis la variable masquée et le masque sont encodés avec une transformation affine. Par conséquent, une relation quadratique secrète lie chaque bit encodé et les 8 bits de la

variable sensible. La taille du masque est choisie de manière à ce que cette relation ne puisse être déterminée. L'encodage quadratique des tables permet également de se prémunir de l'attaque par DCA ciblant la sortie de SubBytes.

Les encodages quadratiques utilisés pour encoder les tables sont annulés lors de l'évaluation des systèmes décrivant ShiftRows et MixColumns. Ces systèmes sont quadratiques et leur évaluation correspond à l'application de ShiftRows et MixColumns suivi d'encodages quadratiques pour masquer chaque octet en sortie de MixColumns. Cette fois, un polynôme de permutation quadratique, le polynôme de Dobbertin, est utilisé pour calculer le masque.

Une fois la description faite, nous détaillons dans ce chapitre une attaque qui a vraisemblablement cassé le candidat soumis à la compétition. L'attaque de type de DCA exploite les collisions internes et le fait que les octets en sortie de MixColumns sont encodés séparément. Ce choix d'encoder séparément les octets est en fait dû à l'utilisation de tables pour les transformations AddRoundKey et SubBytes, ce qui limite fortement le nombre de bits que l'on peut encoder en même temps.

Nos travaux présentent une approche différente pour implanter l'AES en boîte blanche. Elle présente l'avantage de permettre l'ajout de contre-mesures à l'analyse différentielle de fautes. Cependant, une manière d'encoder les bits composant un octet séparément devrait être envisagée afin de se prémunir de l'attaque par DCA.

Contents

General introduction: White-Box Cryptography, where are we now?

"Encryption works. Properly implemented strong crypto systems are one of the few things that you can rely on. Unfortunately, endpoint security is so terrifically weak that NSA can frequently find ways around it."

Edward Snowden

The art of secret

Cryptology comes from the Greek words *kryptos* and *logia* which mean respectively "secret" and "study", thus cryptology is understood as the science of secret, or the study of how to hide information. It gathers together two complementary fields: cryptography and cryptanalysis. The Greek origin of the term cryptography suggests that it is the science of writing secret as *graphein* means "writing". More precisely, cryptography can be simply defined as the science of creating cipher systems while cryptanalysis is the science of breaking ciphers. A *cipher* is a mean of transforming an intelligible message, named *the plaintext* into an unintelligible one called *the ciphertext*. This is done under the control of a secret key. The process called *encryption* or *enciphering* is written $\mathsf{c} = E_K(\mathsf{m})$ for a plaintext m, a secret key K, and a ciphertext c. The reverse process called the *decryption* or *deciphering* is written $\mathsf{m} = D_K(\mathsf{c})$. Encryption and decryption aim at enabling a secure communication over an insecure channel between two parties (a sender and a receiver) even in the presence of an eavesdropper (the attacker). *Secure* here means *confidential*, i.e. any third party should not deduce information about the original message from an observed ciphertext. Encryption ensures the *confidentiality* of the message. Before the modern era, cryptography focused on the confidentiality of the message exchanged by spies, military or country leaders. Modern cryptography tends to prove other properties about the message, namely:

- *Integrity*: any third party should not alter the ciphertext without being detected.
- *Authentication*: a receiver should be sure when receiving a message that it was originated from the sender.
- *Non-repudiation:* when a receiver decrypts a message originated from the sender, the latter cannot deny its origin.

Cipher vs Cryptosystem.

The term cipher typically refers to the pair of encryption and decryption algorithms. These algorithms use a key which is generated by a *key generation algorithm*. A *cryptosystem* consists of the three algorithms: key generation, encryption, and decryption. In this thesis, we sometimes use the term *encryption scheme* to refer to a cryptosystem.

Classical cryptography

Popular classical ciphers are *transposition ciphers* and *substitution ciphers*. In a transposition cipher, the order of the letters in a message is simply rearranged. An example of a transposition cipher is the **scytale cipher** used in 404 before Christ by the Spartans (Figure 1). The idea of the process is to wrap a strip of paper around a staff and then write the message out in rows aligned with the staff. When the paper is taken off the staff, the order of the letters changes. Anyone with a staff of diameter equal to the original one can recover the message. Thus, the diameter of the staff is considered as a secret key known by both parties. For example, the message MESSAGEENCRYPTEDWITHSCYTALE encrypted with a staff of diameter 3 gives the ciphertext MCTERHSYSSPCATYGETEDAEWLNIE.

A substitution cipher systematically replaces a letter or a group of letters of the message with a letter or a group of letters in the ciphertext. An early substitution cipher, which is sometimes called a *shift cipher* is the **Cæsar cipher**, in which each letter of the message is replaced by the

Figure 1: A scytale. Credits: `https://fr.wikipedia.org/wiki/Scytale`.

Plaintext alphabet	A	B	C	D	E	F	G	H	I	J	K	L	M
Ciphertext alphabet	D	E	F	G	H	I	J	K	L	M	N	O	P
Plaintext alphabet	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Ciphertext alphabet	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Figure 2: The Cæsar cipher.

third letter to follow it alphabetically (in a Latin alphabet) (Figure 2). The key is the number of shift which is 3. For example, the message AVECAESAR is encrypted in DYHFDHVDU.

A generalization of the Cæsar cipher consists in shifting by some other integer value. By representing each letter by its position in the alphabet, the cipher may be viewed mathematically as $\mathsf{c} = \mathsf{m} + K \mod 26$ where $\mathsf{m}, \mathsf{c}, K \in \mathbb{Z}_{26}$. The main problem with the Cæsar cipher is the small number of possible keys, 26 which can be easily exhausted. The attack consisting in enumerating all possible keys to decrypt a ciphertext is called a *brute-force attack*. To increase the number of possible keys, a substitution cipher generally makes use of a ciphertext alphabet which is a permutation of the plaintext alphabet. The mapping between the two alphabets gives the substitution. Now, the number of possible keys is equal to the number of possible permutations on 26 letters which is $26! \approx 2^{88}$. Yet, a substitution cipher is vulnerable to a *frequency analysis*, i.e. one can use statistics of the plaintext language to recover a message without knowing the key.

The subsitution ciphers described before are said to be *mono-alphabetic*, i.e. each letter of the plaintext is always encrypted to the same letter in the ciphertext for a fixed key. This is the reason why a frequency analysis breaks the cipher since the distribution of the letter frequencies in the ciphertext space will be the same as in the plaintext language. Actually, the longer the ciphertext, the better the correlation between the two distributions.

One way to solve the problem is to take a number of substitution alphabets and encrypt each letter with a different alphabet. This is called a *poly-alphabetic susbtitution cipher*. The **Vigenère cipher** invented in 1533 by Giovan Batista Belaso, but first described by Blaise de Vigenère in "*Traité des chiffres*" in 1586 is a poly-alphabetic substitution cipher which uses the key (a short word) to change the ciphertext alphabet for each letter of the message. Since the

key is shorter than the message, it is repeated until the end of the message to form a keystream. The encryption of the message MESSAGEENCRYPTED with the key VIGENERE is as follows:

Plaintext	M	E	S	S	A	G	E	E	N	C	R	Y	P	T	E	D
Key	V	I	G	E	N	E	R	E	V	I	G	E	N	E	R	E
Ciphertext	H	M	Y	W	N	K	V	I	I	K	X	C	C	X	V	H

The Vigenère cipher was broken by Babbage using the *Kasiski test*. The cryptanalysis of the Vigenère cipher was officially published by Kasiski in 1863 about ten years after the analysis of Babbage. The principle of the cryptanalysis is to first find the key length, and then apply a frequency analysis on each set of letters of the ciphertext which corresponds to each letter of the key.

The famous **Enigma** machine used during the World War II to encrypt the communications of German military is also a poly-alphabetic cipher. However, the fact that the key was changed everyday has made the frequency analysis impractical on the cipher.

Information theoretic security

Until the modern era of cryptography, the frequency analysis was the only generic technique used to break ciphers. Thus, breaking a message without using the frequency analysis generally required the knowledge of the cipher. Since cryptography was mainly used for military applications and diplomatic communications, the details of the ciphers are most of the time kept secret. In 1883, August Kerckhoffs published in "La cryptographie militaire" six *desirata* for military cryptography. The most famous of them, known as the Kerckhoff's principle states that the security of a cryptosystem should only rely on the secrecy of the key. Thus, the details about the cipher might be a public knowledge or might fall on enemies' hands. This principle was borrowed by Shannon who reformulated it as "The enemy knows the system". Shannon was the founder of the information theory and especially, introduced the *information theoretic security* of a cipher. Also called *unconditional security* or *perfect security*, it characterizes a cryptosystem which cannot be broken whatever the computational power of the adversary. Unconditional security is at the opposite of *computational security* as a cipher is computationally secure if the best possible algorithm for breaking it requires a number of operations which is too large that an adversary with a bounded computational resources cannot carry them out. For instance, the Cæsar cipher, the substitution cipher, and the Vigenère cipher described before are not computationally secure.

Shannon proved that the **Vernam cipher** also named One-Time Pad is an unconditionally secure cipher. In term of information theory, a cryptosystem has perfect secrecy if $\Pr[P = \mathsf{m} \mid C = \mathsf{c}] = \Pr[P = \mathsf{m}]$ for all plaintexts m and all ciphertexts c. Roughly speaking, the ciphertext does not reveal information about the message. The principle of the Vernam cipher is to add (with an exclusive-or) the plaintext to a key of the same size to form the ciphertext. Each key is used once, that is why the cipher is called One-Time Pad. Shannon proved that a fundamental requirement to guarantee the perfect secrecy is that the keys are used with equal probability. The One-Time Pad has been used in the past in military and diplomatic contexts, for instance to secure the communications between the White-House and the Kremlin after the Cuban missile crisis.

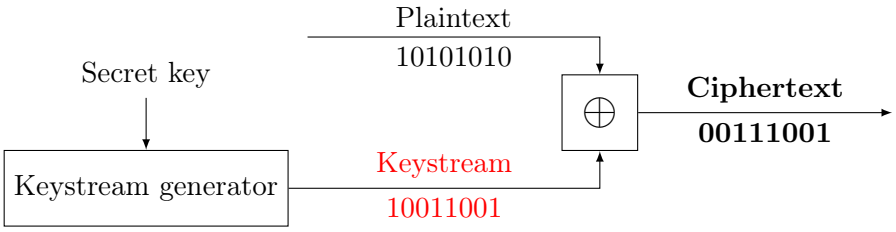


Figure 3: A simple stream cipher.

Modern cryptography: symmetric and asymmetric cryptography

The classical ciphers described before make use of a unique secret known by the two parties involved in the communication. In modern cryptography, when the key is secret and used for both encryption and decryption, the cipher is said *symmetric*. Two types of symmetric ciphers can be distinguished: *stream ciphers* which operate on one bit at a time and *block ciphers* which operate on a block of bits at a time.

In a stream cipher, the plaintext is xored with a keystream generated by a keystream generator from a short secret key. Thus, an exclusive-or of one bit of the plaintext with one bit of the keystream gives one bit of the ciphertext (Figure 3).

In a block cipher, one block of the plaintext is encrypted to get one block of the ciphertext. Popular block ciphers are the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES). The DES was developed in the early 1970s by IBM and relies on a Feistel structure (Figure 4). The DES cipher, illustrated in Figure 5, is an *iterated block cipher* with 16 rounds, i.e. a round function is repeated a number of times and includes a key addition; the so-called *round key* is derived from a unique *master key*. The block length is 64 bits, the master key length is 56 bits, and each round key has 48 bits. Until the 1990s, the DES was considered to be computationally secure. The *differential cryptanalysis* [9], the *linear cryptanalysis* [87], and the *time-memory tradeoff attack* [65] revealed that a practical cryptanalysis of the DES is feasible.

The AES is a block cipher based on the Rijndael algorithm which was the winner of the AES competition launched by the National Institute of Standards and Technology (NIST). The goal of the competition was to define a new encryption standard to replace the DES which is not considered to be computationally secure anymore. The AES is an iterated block cipher having a structure of Substitution-Permutation Network, i.e. each round function alternates a substitution transformation and a permutation. The block length is 128 bits, the key length is either 128 with 10 rounds, 192 with 12 rounds or 256 with 14 rounds. The AES algorithm is described in detail in Chapter 2 of this thesis. The AES is today standard integrated into processors, smart cards and mobile phones. The security of the AES has been assessed through cryptanalysis. So far, the AES is computationally secure.

In 1976, Diffie and Hellman [45] introduced a new paradigm which marked the advent of asymmetric cryptography or *public-key cyptography*. They proposed to use two distinct keys , a *public key* for encryption and a *private key* for decryption leading to an asymmetric cryptosystem. A public key cryptosystem generally relies on a mathematical relation between the public key and the private key, namely a message encrypted with the public key can be decrypted with the private key but the problem of determining the private key from the knowledge of the public key is mathematically hard to solve. The security of a public key cryptosystem is assessed through a proof by reduction. In this thesis, we only focus on symmetric ciphers.

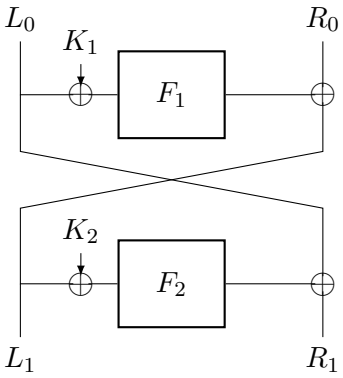


Figure 4: 2-round Feistel Cipher.

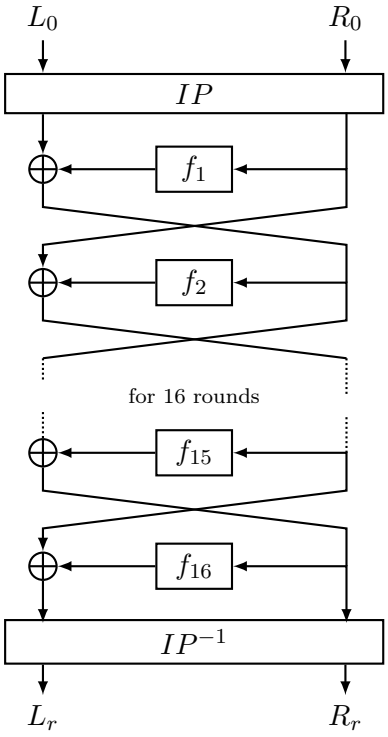


Figure 5: The DES cipher: $f_1, \ldots, f_{16}$ are the round functions, IP is an initial permutation and IP^{-1} its inverse.

Who can we trust?

"*Trust is a psychological state comprising the intention to accept a vulnerability based upon positive expectations of the intentions or behavior of another.* [105]. To place this definition in an historical context, during the World War II, when the German military leaders communicated the position of their troops using Enigma, they had accepted that this information would make them vulnerable. The strategy of the cryptanalysts of Bletchley Park to not reveal the break of Enigma was motivated by this fact: as long as the enemy trusts the machine, they are vulnerable.

When we talk about a secure communication, it is generally assumed that the parties (the end-points of the communication channel) are trustworthy, i.e. each party trusts the other to not leak the key used for the communication. This is of great importance since the security of the system relies on the secrecy of the key. In other words, the end-points of the communication channel are trusted and the cryptographic algorithms are executed in a secure environment. The adversary only intercepts the communication. Thus, the system acts as a *black-box* from the attacker's point of view. This model actually describes the use of cryptography in military contexts in which the cryptosystems are implemented in proprietary hardware.

Since the deployment of cryptography for the general public, in smart cards for instance, it appeared that the end-points of the communication channel could fall into the attacker's hands. Even worse, the hardware might leak implementation-specific information which reveals the key. The implementation attack of Paul Kocher [75] published in 1996 marked the beginning of the *gray-box model*. In a gray-box model, the security of a cryptographic algorithm does not rely only on the secrecy of the key but also on the absence of information leakage. The information leaked is called a *side-channel information* and is exploited by side-channel attacks. Examples of side-channel information are the execution time of the cryptographic algorithm or the power consumption of the smart card.

The development of computers and a variety of mobile devices like smartphones, tablets, set-top boxes, went along with cryptography-enabled applications and software implementations of cryptographic algorithms. Such open platforms are susceptible to malwares or viruses and thus should not be trusted. The key entropy attack of Shamir and Van Someren [109] and the S-box blanking attack of Kerins and Kursawe [70] have made one think to a stronger attack model. In 2002, Chow et al. introduced the *white-box model* to take into account an attacker who has full access of a software implementation of a cryptographic algorithm as well as full control over its execution environment. Even if such a model creates skepticism among academic researchers, recent attacks on industrial products, for instance the DRM Widevine L3 used by Netflix, Disney, and Amazon, support the reality of the model.

Nowadays, to trust a system, one needs to trust the cryptographers who elaborate computationally secure ciphers but also the platform in which they are implemented. White-Box Cryptography is the concern of people who only trust cryptography.

Organization of the thesis

This thesis arises the following problems:

How can we have confidence on White-Box Cryptography, is it possible to base the security brought by White-Box Cryptography on hard problems? and in a more pratical approach how can we strengthen a white-box implementation using a hardware component?

Regarding those questions, this thesis aims at *understanding the white-box paradigm, analyzing all known techniques used in White-Box Cryptography, understanding the security level*

guaranteed by each technique, and finally highlighting the outlook for White-Box Cryptography development.

For better understanding of the subject, this thesis is divided into two main parts: the first part of the thesis presents the state-of-the-art of White-Box Cryptography. It especially gives the motivations for considering cryptographic algorithms in a new theoretical model through practical use cases. Our goal is that at the end of this part the reader is convinced of the credibility of the model and clearly understands the design rationales behind each white-box implementation; the second part of the thesis gathers together our contributions. The different chapters of the two parts are summarized below.

Part I - Chapter 1: White-box model. This chapter discusses the white-box model: why the model is said to be *white-box*? And what is the objective of White-Box Cryptography. The term White-Box Cryptography constitutes the research area dedicated to the security of a software implementation of a cryptographic algorithm in the white-box model. It includes both the techniques used to propose a white-box implementation and the cryptanalytic techniques. In this chapter, we describe some real applications of White-Box Cryptography which validate the model. Even if White-Box Cryptography was originally created for Digital Rights Management and mobile agents, new use cases appear recently, namely Host Card Emulation and Digital Signature. The common point of all those applications is that a software implementation of a cryptographic algorithm may be deployed on a mobile device. However, mobile devices show two big threats: a device might be infected by a malware so it cannot be trusted and the end-user might be malicious. In that sense it constitutes an untrusted execution environment. Besides, this chapter highlights the differences between the gray-box and the white-box models. If we assume that a model is characterized by the capabilities of the attacker then the gray-box model is included in the white-box model. That means that a white-box attacker has at least the same capabilities as a gray-box attacker.

Part I - Chapter 2: White-box implementation of the AES: an endless story. This chapter gives a chronological description of published white-box implementations. Starting from the seminal work of Chow et al. which introduced the first technique used in White-Box Cryptography, the chapter goes through the improvements and the encountered problems. Further, this chapter gives the details of the white-box implementations and points out the vulnerabilities that are exploited by the white-box attacks. This chapter concludes with the promising techniques for White-Box Cryptography.

Part I - Chapter 3: Fifty shades of gray-box attacks. Since the security of white-box implementations was difficult to assess by proofs, a contest was launched in 2017 to confront anonymous white-box implementations to real-world attackers. The contest followed on from the WhibOx 2016 workshop which dealt on White-box Cryptography and obfuscation. During the workshop, generic attacks from the gray-box model, namely Differential Computation Analysis which is an adaptation of Differential Power Analysis in the white-box model and Differential Fault Analysis, were shown to be efficient on white-box implementations because they do not need an in-depth understanding of the white-box techniques. The contest validated the practicability of those attacks because all white-box implementations submitted to the competition were broken using them. This chapter describes those gray-box attacks adapted to the white-box model and explains how they can be thwarted and then improved. The competition also revealed a new technique of White-Box Cryptography, namely a circuit-based white-box

implementation. Indeed, Biryukov and Udovenko first proposed a circuit-based white-box implementation of the AES to the competition. Thus, the security of a circuit-based white-box implementation regarding DCA attacks can be enforced through masking schemes.

Part I - Chapter 4: Theory of White-Box Cryptography. White-Box Cryptography was first approached in a practical manner, i.e. how a standard cryptographic algorithm can be transformed to provide a software implementation which hides the secret key. However, such approach shows a real lack of security proof which leads to the cryptanalytic results. A theoretical model of White-Box Cryptography would consists in defining security notions which formalize the goals of an attacker and the attack scenarios. This chapter gives an overview of all theoretical works related to White-Box Cryptography and explains how the formalization helps to design provably secure encryption schemes.

Part II - Chapter 1: PUF-based cryptography. Given a white-box implementation of a cryptographic algorithm which embeds a secret key, one natural question arises: if the implementation guarantees the security of the key, what about the security of the implementation? In other words, what is the point in hiding the key in the code if the code can be copied? Indeed, anyone who copies the code is able to encrypt or decrypt without knowing the key. This attack is called a code lifting attack and can be mitigated with external encodings. Another approach for code lifting mitigation is the notion of incompressibility: a cryptographic algorithm is implemented such that an attacker cannot *compress* the white-box implementation without knowing the key. A similar notion called space-hardness ensures that a portion of the code of size smaller than a fixed bound will unlikely be functionally equivalent to the original code. In this chapter, we consider a new security notion called *lockability* which ensures the security of a white-box implementation as long as a hardware component verifies an *unclonability* property. Thus, the security of a white-box implementation regarding code lifting depends on the security of the hardware component regarding a cloning attack. The hardware component in question is called a Physically Unclonable Function (PUF).

This work was accepted for publication to the 17th International Conference on Security and Cryptography (SECRYPT 2020).

Part II - Chapter 2: Traitor tracing. In this chapter we focus on the Digital Rights Management case. A DRM generally uses a broadcast encryption scheme to enforce access control. However, a broadcast encryption scheme suffers from illegal key distribution. To discourage users to distribute their keys, a tracing procedure is associated to the scheme such that any user who shares their key can be identified. This chapter presents a broadcast encryption scheme which uses a traceable key generator. The key generator is constructed using techniques of White-Box Cryptography and verifies the notions of *unbreakability* and *incompressibility*. Any user receives a distinct version of the key generator which embeds a Tardos codeword. This way the scheme can be provided with a tracing procedure to trace traitors.

This work was published in [102] and is joint work with Gilles Macario-Rat and Marine Minier.

Part II - Chapter 3: White-box implementation of the AES. White-Box Cryptography constitutes a hot topic because it promises security without a hardware secure element. Especially, a secure white-box implementation of the Advanced Encryption Standard is of particular interest because it is a standard. In this chapter we describe a way to implement the AES in the

white-box model. Our method combines lookup tables and systems of polynomial equations. The reason of that choice is to both thwart the algebraic attack of Billet et al. and the Differential Computation Analysis. In addition, our method allows to add indelible redundancies which help against the Differential Fault Analysis. However, our white-box implementation is not secure enough because a DCA based on collisions is still possible. A collision attack exploits the fact that a collision may occur on intermediate variables and such collision reveals information about the secret key. If the collision exploited occurs in the first round, then it is relatively easy to deduce the secret key. We discuss in this chapter a way to avoid collisions in the first round on the same white-box implementation. Unfortunately, the countermeasure causes a substantial increase of the white-box size. To pave the way to a new approach, we propose a new white-box technique which only consists of systems of polynomial equations and is based on the decomposition of the S-box into functions of smaller degrees. The security of the white-box implementation regarding key extraction can be reduced to the complexity of the affine equivalence algorithm. While the generic algorithm for finding affine equivalences has a complexity exponential in the size of the affine mappings, some specialized affine equivalence algorithms have lower complexities. Thus, we looked at the relevance of these algorithms regarding our white-box technique.

The first work leads to the submission of a challenge program to the WhibOx contest 2019 [41] and is joint work with Sébastien Bardin, Gilles Macario-Rat, Marine Minier, and Mathilde Ollivier.

The second work was submitted to SSPREW-9, the 9th Software Security, Protection, and Reverse Engineering Workshop.

List of Figures

List of Tables

List of Symbols

$(a_{l-1} \dots a_1 a_0)_2$ Binary representation of length l of an integer a

$[n]$ $\{i \in \mathbb{N} \colon 1 \leqslant i \leqslant n\}$

$\mathcal{A}$ Adversary

$\mathsf{Adv}_{\mathcal{A}}(\lambda)$ Advantage of an adversary $\mathcal{A}$ according to the security parameter λ

$\mathcal{O}$ Big O notation

Θ Big and small O notation

$|S|$ Number of elements in the finite set S

Dec Symmetric decryption

$\emptyset$ Empty set

Enc Symmetric encryption

$\langle g_1, \dots, g_n \rangle$ Group generated by $g_1, \dots, g_n$

$\langle x, x \rangle$ Inner product

$\lceil x \rceil_u$ The u highest bits of x

$\leqslant_R$ Polynomial reduction

$\lfloor x \rfloor_u$ The u lowest bits of x

$\ln$ Natural logarithm

$\log_2$ Logarithm to base 2

$\mathbb{F}_q$ Finite field of $q = p^d$ elements for a prime number p

$\mathcal{K}$ Key space

$\mathcal{O}$ Oracle

$\oplus$ The exclusive-or operation

$\otimes$ The multiplication in the finite field $\mathbb{F}_{2^8}$

λ Security parameter

e_i i-th unit vector

I_n $n \times n$ identity matrix

K Master key

$K^0, \ldots, K^R$ $R + 1$ round keys

$M_n(R)$ $n \times n$ matrices over the ring R

$P(x)$ Irreducible polynomial defining a finite field

q Number of oracle queries in a security proof

$s \leftarrow S$ $s \in S$ chosen according to an implicit distribution on S

C Ciphertext space

c Ciphertext

id Identity of a user

$\mathsf{len}(a)$ Bit-length of a

M Message space

m Message

$\mathsf{span}(v_1, \ldots, v_n)$ Span of a set of vectors

$\mathsf{wt}(a)$ Hamming weight of a, i.e. the number of ones in the binary representation

Part I

From practice to theory: state-of-the-art on White-Box Cryptography

1

The White-box model

> "There's no sense in being precise when you don't even know what you're talking about."
>
> *John von Neumann*

Contents

In this chapter we investigate the origins of White-Box Cryptography. We first explain the *white-box model* paradigm and how this model materializes in practice. Especially, we give some real-world applications where the white-box model is more suitable than black-box model. Those applications show how the model addresses use cases. Besides, we recall the gray-box model specificities in order to highlight the main differences between the white-box and the gray-box models. Then, we briefly describe the white-box attacks which we consider in this thesis. The details of the attacks and their specific application to the Advanced Encryption Standard are given in Chapter 2. We explain why the gray-box model isn't enough to model the reality and why it is relevant to consider the white-box model. Finally, we discuss the fundamental property required in the white-box model, the key extraction security. Indeed, as imposed by the Kerckhoff's principle: "a cryptosystem should be secure even if everything about the system, except the key, is public knowledge". Consequently, a white-box cryptosystem should at least guarantee the confidentiality of the secret key. In this chapter, we try to answer the following question: *what is the white-box model and why considering the security of the cryptographic algorithms in this model is relevant?*

Introduction

A *model* is a simplified representation of a process. In cryptography, the standard *black-box model* refers to the *attack model* against a cryptosystem which behaves as a *black-box* in the attacker's point of view. Thus, the *black-box model* or *black-box attack model* represents any attack on a cryptosystem by an attacker who is only able to see the input and output behavior of the cryptosystem so the process modeled is an attack. Consequently, the *white-box model* represents any attack on a cryptosystem which is seen as a *white-box* by the attacker. Here white-box is chosen to be the natural opposite of black-box. That means that the attacker is not only able to see the input and output behavior of the cryptosystem but is also able to see the *internal details*. In other words, a black-box is *impenetrable* while a white-box is *open*.

The security of a cryptographic algorithm in the black-box model relies on the behavior of the algorithm. That means that an attacker should not learn information about a secret key from the algorithm execution. This model has been studied for a while and some cryptographic algorithms are considered quite secure in the black-box model.

In contrast, the security of a cryptographic algorithm in the white-box model relies on the implementation of the algorithm. That means that an attacker should not learn information about a secret key from the implementation. Note that before defining the white-box model, the *gray-box model* was defined to represent the implementation attacks, i.e. any attack that incorporates implementation-specific information. In this model, the attacker is able to see the input and output behavior of the cryptosystem but also some information correlated to the internal details. This model was adopted after the success of attacks which use the behavior of an electronic device to recover the secret key used by the embedded cryptographic algorithm. So, the gray-box model appeared because of the usage of cryptography in distributed devices like smart cards for payments or authentication. In the following, we show how the advents of new applications laid the foundations for the white-box model.

1.1 Motivations

The security of all known cryptographic algorithms has been considered in the standard cryptographic model, the black-box model, where it is assumed that the endpoints are trusted. Thus, communications between the two trusted parties are secure as long as the key used to encrypt them is kept secret. However, cryptographic algorithms may be used in software applications that are executed on an open and untrusted platform. In this case, the endpoint (the platform) cannot be considered as a trusted party and the secure storage of the key cannot be guaranteed. We describe below some examples of applications where a software implementation of a cryptographic algorithm can be used.

1.1.1 Digital Rights Management

The set of access control technologies for restricting use and distribution of a copyright protected content is called a Digital Rights Managements (DRM). DRM technologies are typically used by online digital multimedia stores (ebook, music, video, etc). In a system with DRM enforcement, a multimedia content is generally encrypted by a provider and sent to a client (an application executed on a end-user's platform). In addition to the *encrypted media content*, the client receives a *licence* and an *encrypted content key*. The task of the client is to check the user's licence and then decrypt the content key. Thus, the client should securely store a decryption key.

1.1.2 Host Card Emulation

Host Card Emulation (HCE) refers to a software architecture that provides a virtual representation of electronic cards (payment, access, transit) using only a software. HCE is used mobile devices to enable an NFC-enabled smartphone to make payments, to manage loyalty or to perform transport validation. Thanks to HCE, a mobile payment application is able to communicate with a Point of Sale (POS) in the same way as a smart card. Besides, the application may execute all the cryptographic operations traditionally executed by the smart card and so needs to store cryptographic keys.

1.1.3 Mobile Agent

Mobile agents are software which moves autonomously through a computer network and perform some computation on the host platform. A mobile agent is a specific form of mobile code. A mobile code might be executed on a malicious host platform and since the program will be run under the control of the host, it should be protected. There are several ways to protect a mobile code from its execution environment. The mobile agent can be encrypted to guarantee integrity and privacy. However, during the program execution, parts of the code are decrypted with a secret key. Consequently, this decryption should be securely executed.

1.1.4 Digital Signature

A digital signature is a mathematical scheme used to verify the authenticity of digital messages or documents. A digital signature generally uses asymmetric cryptography, i.e. a private key is used to sign the message and the corresponding public key is used to verify the signature. Thus, the confidence in the authenticity of the message strongly relies on the security of the private key. The private key can be stored in a smart card or protected using software techniques.

1.2 Gray-box model

The gray-box model defines an attack model on the implementations of cryptographic algorithms. That means that the attacker in addition to be able to observe the input/output behavior of the algorithm can also take advantage of *implementation-specific information*. These implementation-specific information are linked to the key so they leak information about the key and are called *key leakage information*. Implementation attacks can be *passive* (the attacker only observes the execution of the algorithm) or *active* (the attacker modifies the execution of the algorithm). In that sense, they tamper with the functioning of the system and are said *intrusive*. If the attacker tampers with the device where the algorithm is implemented to have access to the cryptographic implementation, the attack is said *invasive*. If he is limited to record externally available information, the attack is *non-invasive*. A side-channel attack refers to implementation attacks that are passive and non-invasive. We describe below the well-known side-channel attacks.

Timing attack. The timing attack was introduced in 1996 by Paul Kocher and is considered as the first side-channel attack. It consists in observing the variations of execution time of a cryptographic algorithm depending on the key used. Because of conditional statements the execution of an algorithm may be non-constant. Particularly, if the statements are key-dependent, the non-constant execution leaks information about the key. This kind of timing attack called *branch*

timing attack because it exploits key-dependent branch selections was described for public-key algorithms like RSA. For symmetric algorithms, Kocher presented a *cache timing attack*. This second attack exploits the time variations due to cache and Random Access Memory (RAM) readings. Consequently, it is efficient for implementation which uses lookup tables, for instance the software-based implementation of AES.

Simple Power Analysis and Differential/Correlation Power Analysis. Simple Power Analysis (SPA) and Differential Power Analysis (DPA) are side-channel attacks described by Kocher et al. in 1999. As their names suggest, they exploit the power consumption of an electronic device when performing a cryptographic operation. The idea of both attacks is to reveal the correlation between power consumption traces and the key used. In the SPA case, only a single trace is used and is correlated to the key-dependent operations while in the DPA case, a set of power consumption traces is used to allow statistical tests. A Differential Power Analysis typically makes use of the difference of means while the Correlation Power Analysis is based on the Hamming distance model and makes use of the Pearson's correlation coefficient.

Fault attack. The first fault attack was presented by Boneh et al. in 1997 and consists in introducing computational faults in the execution of a public-key algorithm, typically the RSA. Such faults are generally obtained by physically manipulating the device where the algorithm is implemented. Following this approach, Biham and Shamir described a fault attack on block ciphers called *Differential Fault Analysis*. The term "differential" refers to the fact that the attack consists in computing the difference between two ciphertexts, obtained before and after injecting a fault. Besides, the attack depends on a fault model which is based on the specificities of the algorithm and the precision of the fault injections.

1.3 Key extraction security

The purpose of White-Box Cryptography is to provide a secure software implementation of a cryptographic algorithm in a way that an attacker cannot extract a secret key even if he controls the execution environment or has access to the software implementation. This context is called the *white-box attack context* which is captured by the white-box model. So, the basic security requirement that is expected from a white-box implementation is called the *key extraction security*. Key extraction security is typically evaluated through the development of cryptanalysis techniques, that means that one can get confidence in the security of a white-box implementation after verifying that any known white-box attack fails. It is assumed that an attack fails as long as the computational complexity of the attack is out of reach. This obviously doesn't constitute a real proof of security. We will see in Chapter 4 a more theoretical approach of the security of a white-box implementation.

1.3.1 White-box attacks

A white-box attack is an attack which exploits both the execution environment and the implementation. In practice a white-box attacker can inspect the memory, analyze the implementation binary, intercept CPU calls, use debugging and analysis tools and inject faults. Besides, he can execute the program at will on chosen inputs. In this thesis, we consider the following white-box attacks.

Entropy attack. An entropy attack exploits the high entropy of a cryptographic key to distinguish the cryptographic key from the other parts of the algorithm using a memory access. It was described by Shamir and Van Someren in 1999 [109]. Actually, when looking at the memory which contains a software implementation of a cryptographic algorithm and a key, the key part is noisy while the algorithm part is more structured. In some cases this is enough to distinguish and extract the key.

A countermeasure to an entropy attack would consists in ensuring that the entropy density is uniformly distributed over the entire implementation. This can be achieved by spreading the key over the entire implementation and by introducing randomness to the algorithm.

S-box blanking. Described by Kerins and Kursawe in 2006 [70], a S-box blanking attack targets the last round of a software implementation of a block cipher which is composed of a parallel execution of a key-independent S-box followed by a key addition. The attack consists in identifying the location of the final round S-boxes within the software binary and then by setting all entries to zero. Thus, a single execution of the modified implementation reveals the last round key.

To thwart such attack, one could prevent the localization of the S-boxes using key-dependent S-boxes, S-box masking techniques or dynamically generated S-boxes, verify the integrity of the software binary before executing the block cipher or use independent round keys.

Algebraic attack. An algebraic attack generally consists in representing the block cipher as an over-defined system of multivariate equations in the bits of the plaintext, ciphertext and secret key. Thus, the secret key is the solution of the system for a set of pairs of plaintext-ciphertext. Such technique was proposed by Courtois and Pieprzik [39] in the black-box model. In this case, each equation is of high algebraic degree because of the non-linear layer of the block cipher. While the attack is unlikely successful in the black box model, it is more practical is the white-box model because the attacker can define lower-degree algebraic equations. The BGE attack of Billet et al. and its generalization by Michiels et al. as well as the collision attack of Lepoint et al. are exemple of successful algebraic attacks in the white-box model.

A countermeasure of such attack is to hide the algebraic structure of the block cipher.

Differential Fault Analysis (DFA). A Differential Fault Analysis is an attack introduced by Biham and Shamir which consists in injecting computational faults in the last rounds of the cipher and then computing the difference between a correct ciphertext and a faulted ciphertext. The computational faults are introduced by manipulating the hardware of software in a non-invasive or invasive way. It was first described in a gray-box model and the faults are injected probabilistically, which consequently implies a probabilistic attack success. In the white-box model, the faults are injected deterministically which leads to a very powerful attack.

Some techniques have been proposed in the gray-box model to prevent DFA. Those techniques generally rely on time or space redundancies to verify the integrity of the implementation.

Differential Computation Analysis (DCA). The designation *Differential Computation Analysis* was introduced by Bos et al. in 2016 to describe an adaptation of the Differential Power Analysis in the white-box model. The attack exploits the fact that an attacker may have access to *computation traces* which correspond to intermediate values or addresses of intermediate values. So, instead of correlating *power consumption traces* with a predicted key-dependent variable, a white-box attacker tries to correlate observed variables with a predicted key-dependent

variable. Thus, a DCA always consists in a prediction function which predicts the value of a key-dependent variable and a set of computations traces. In this thesis, we consider two types of DCA: simple DCA where the prediction function is simply a key-dependent transformation and a collision-based DCA where the prediction function predicts whether two values collide or not.

A typical countermeasure to DCA is *masking*. The idea is to hide the correlation that might exists between the key-dependent variable and the computation traces.

Mutual Information Analysis (MIA). A Mutual Information Analysis is a gray-box attack that can be used even if the attacker has a limited knowledge about the leakage distribution. A MIA uses the maximal mutual information between a key-dependent variable and a computation traces as a distinguisher.

Conclusion of the chapter

The white-box model is a new paradigm which appeared in the beginning of the years 2000 because the gray-box model seems to be insufficient to model the abilities of an attacker when a software implementation of a cryptographic algorithms is deployed in an open and untrusted platform. In this context called the *white-box attack context* by Chow et al., the implementation becomes the last line of defense. The term White-Box Cryptography, which should be understood as "Cryptography in the white-box model", then appears to refer to the research of secure software implementations of cryptographic algorithms. Thus, the goal of the cryptographers in White-Box Cryptography is to produce a secure software implementation of a cryptographic algorithm regarding white-box attacks. In a white-box attack, it is assumed that the attacker is executed on the same platform as the targeted software so has full control of the execution environment of the software and full access to the implementation. Consequently, a white-box attacker is more powerful compared to a gray-box attacker who has limited access to the implementation and to the execution environment. However, it seems that some white-box attacks are transposition of gray-box attacks. The reason is that a white-box attacker has access to more leakage information since he has full access of the implementation and controls the execution environment. Especially, since he has directly access to intermediate values of the algorithm, he doesn't need to manage the noisy effect of measuring physical parameters like power consumption or electromagnetic emanation. Besides, he can easily modifies the execution of the algorithm (thus, be an active attacker) while an active attack in the gray-box model only succeeds with a certain probability.

2

White-box implementation of the AES: an endless story

"I just wondered how things were put together."

Claude Shannon

Contents

In White-Box Cryptography, the main objective is to guarantee the security of a software implementation of a cryptographic algorithm in the white-box attack context. One way to have this guarantee is to implement the cryptographic algorithm with a single lookup table which associate all possible plaintexts with the corresponding ciphertexts for a fixed key. The lookup table is seen as a black-box because an attacker can only see the input and output behavior of the algorithm through the pairs of plaintext/ciphertext. This way, an attacker cannot do better than trying all possible keys to deduce the key used to compute the lookup table. However, according to the size of the input, this solution is not practical. Given that the input and output sizes are both n bits, the storage requirement for the lookup table is $2^n \times n$ bits. In the case of the AES where $n = 128$, the storage space is 4.95×10^{27} TB. This result shows that there exits a computationally secure white-box implementation of any cryptographic algorithm if the storage space is unbounded. However, a natural question may arise: *assuming a bounded storage space, is it possible to find a computationally secure white-box implementation of any cryptographic algorithm?*

This chapter answers the question in the negative in the case of the AES, i.e. all proposed white-box implementations of the AES are not computationally secure. In this chapter we try to understand the design techniques of each publication and the reasons of their insecurity. We give a chronological overview of the different implementations and the attacks that break them. This description over time provides a better understanding of how White-Box Cryptography has evolved.

The Advanced Encryption Standard (AES)

In 1997, the National Institute of Standards and Technology (NIST) started a standardization process with a competition called the "AES competition". The process was intended to define a new encryption standard for the United States government. The competition was about designing a block cipher secure enough to replace the Triple DES in the future. Among the 15 candidates, the winner *Rijndael* was designed by Vincent Rijmen and Joan Daemon. Nowadays, we call AES the block cipher based on Rijndael structure and specified in [101].

The AES is a block cipher with a 128 bit block size and variable key sizes: 128, 192 or 256 bits. The structure of the cipher is a Substitution-Permutation Network, i.e. it is constituted of several rounds which each alternates a substitution layer and a permutation layer. Depending on the key size, the number of rounds is respectively 10, 12 or 14.

In this thesis, we will focus on the 128 bits version of the AES, i.e. the key size is 128 bits and the number of rounds is 10. A key schedule algorithm enables to derive 11 round keys $K^0, \ldots, K^{10}$ from a master key K. A *round key* K^r is represented by a 4×4 array of bytes $K^r = (K^r_{i,j})_{0 \leqslant i,j \leqslant 3}$. Each round updates a *state* represented by a 4×4 array of bytes $state = (s_{i,j})_{0 \leqslant i,j \leqslant 3}$ by applying four transformations: SubBytes, ShiftRows, MixColumns and AddRoundKey. A byte is seen as an element of the field $\mathbb{F}_{2^8}$ defined as $\mathbb{F}_{2^8} \approxeq \mathbb{F}[x]/(x^8 + x^4 + x^3 + x + 1)$. Thus, a byte is represented as a polynomial of degree at most 7 and is expressed with the hexadecimal representation: 7 is represented by $x^2 + x + 1$ and written 0x7, 18 is represented by $x^4 + x$ and written 0x12 in hexadecimal. For simplicity and if it is clear with the context, the hexadecimal form is written without the "0x", i.e. we write 12 instead of 0x12 and 02 instead of 0x2.

b_0 \ b_1	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Table 2.1: The AES S-box: substitution values for the bytes $0\mathsf{x}\mathsf{b}_0\mathsf{b}_1$.

- SubBytes substitutes each byte of the state thanks to a substitution table SB called the AES S-box (Table 2). The AES S-box is a non-linear bijective mapping from 8 bits to 8 bits. It is the composition of an affine mapping $g\colon x \mapsto A \cdot x \oplus b$ from $\mathbb{F}_2^8$ to itself with the multiplicative inversion in $\mathbb{F}_{2^8}$. For any element $x \in \mathbb{F}_{2^8} \setminus \{0\}$, its inverse x^{-1} is seen as a vector in the base field $\mathbb{F}_2$, i.e. $x^{-1} = (x_0, \ldots, x_7) \in \mathbb{F}_2^8$. The element 0 is sent to itself then the affine transformation g is applied to the vector as follows:

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \xrightarrow{g} \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

  SubBytes updates the state as follows:

$$\forall\ 0 \leqslant i, j \leqslant 3,\ s_{i,j} \leftarrow \mathsf{SB}(s_{i,j}).$$

- ShiftRows permutes all bytes of the state. The permutation corresponds to a circular shift to the left of i bytes for the i-th row for $0 \leqslant i \leqslant 3$, and can be expressed by the permutation of indexes SR:

$$\forall\ 0 \leqslant i, j \leqslant 3,\ \mathsf{SR}(i, j) = (j - i) \mod 4.$$

  ShiftRows updates the state as follows:

$$\forall\ 0 \leqslant i, j \leqslant 3,\ s_{i,j} \leftarrow s_{i,\mathsf{SR}(i,j)}.$$

- MixColumns applies a linear transformation to 4 bytes of the state in parallel. Each column of the state array is expressed as a vector of $\mathbb{F}_{2^8}^4$ and multiplied by a 4×4 matrix MC with coefficients in $\mathbb{F}_{2^8}$. The matrix is said Maximum Distance Separable as it ensures a distance at least 5 between any input-output pair and is defined as:

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

where the coefficients are in hexadecimal forms.

MixColumns updates the state as follows:

$$\forall\ 0 \leqslant j \leqslant 3,\ \begin{pmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{pmatrix} \leftarrow \mathsf{MC} \cdot \begin{pmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{pmatrix}$$

- AddRoundKey adds the r-th round key to the state array. It corresponds to an "exclusive-or" between the round key and the state array. AddRoundKey updates the state as follows:

$$\forall\ 0 \leqslant i, j \leqslant 3,\ s_{i,j} \leftarrow s_{i,j} \oplus K^r_{i,j}.$$

The encryption of a plaintext with the AES-128 block cipher is described in Algorithm 1.

Algorithm 1 The block cipher AES-128.

$state \leftarrow plaintext$
AddRoundKey$(state, K^0)$
for $r = 1 \ldots 9$ **do**
    SubBytes$(state)$
    ShiftRows$(state)$
    MixColumns$(state)$
    AddRoundKey$(state, K^r)$
end for
SubBytes$(state)$
ShiftRows$(state)$
AddRoundKey$(state, K^{10})$
$ciphertext \leftarrow state$

Standard software implementation

As specified by Rijmen and Daemon in [42], Rijndael can be efficiently implemented on a 32-bit processor (or higher word length) with four lookup tables with size $2^8 \times 32$ bits each. Thus, the implementation size is 4 kB. Those lookup tables are obtained by combining SubBytes and MixColumns. First note that the transformations AddRoundKey and ShiftRows can be swapped in the algorithm description. Since AddRoundKey can be seen as a byte addition and ShiftRows is a permutation of byte positions, adding a round key to the state then permuting the bytes of the current state is the same as permuting the bytes of the state then adding a permuted round key. That observation enables to describe the AES algorithm in an equivalent way given in Algorithm 2.

Algorithm 2 An equivalent description of the block cipher AES-128.

$state \leftarrow plaintext$
for $r = 1 \dots 9$ **do**
  $\mathsf{ShiftRows}(state)$
  $\mathsf{AddRoundKey}(state, \widehat{K}^{r-1})$ ▷ $\widehat{K}^{r-1} \leftarrow \mathsf{ShiftRows}(K^{r-1})$
  $\mathsf{SubBytes}(state)$
  $\mathsf{MixColumns}(state)$
end for
$\mathsf{ShiftRows}(state)$
$\mathsf{AddRoundKey}(state, \widehat{K^9})$
$\mathsf{SubBytes}(state)$
$\mathsf{AddRoundKey}(state, K^{10})$
$ciphertext \leftarrow state$

The composition of the two transformations SubBytes and MixColumns updates the state as follows:

$$\forall\ 0 \leqslant j \leqslant 3,\ \begin{pmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{pmatrix} \leftarrow \mathsf{MC} \cdot \begin{pmatrix} \mathsf{SB}(s_{0,j}) \\ \mathsf{SB}(s_{1,j}) \\ \mathsf{SB}(s_{2,j}) \\ \mathsf{SB}(s_{3,j}) \end{pmatrix} \text{ with,}$$

$$\mathsf{MC} \cdot \begin{pmatrix} \mathsf{SB}(s_{0,j}) \\ \mathsf{SB}(s_{1,j}) \\ \mathsf{SB}(s_{2,j}) \\ \mathsf{SB}(s_{3,j}) \end{pmatrix} = \begin{pmatrix} 02 \\ 01 \\ 01 \\ 03 \end{pmatrix} \mathsf{SB}(s_{0,j}) \oplus \begin{pmatrix} 03 \\ 02 \\ 01 \\ 01 \end{pmatrix} \mathsf{SB}(s_{1,j}) \oplus \begin{pmatrix} 01 \\ 03 \\ 02 \\ 01 \end{pmatrix} \mathsf{SB}(s_{2,j}) \oplus \begin{pmatrix} 01 \\ 01 \\ 03 \\ 02 \end{pmatrix} \mathsf{SB}(s_{3,j}).$$

Let $\mathsf{SM}_0, \mathsf{SM}_1, \mathsf{SM}_2, \mathsf{SM}_3$ be four mappings from $\mathbb{F}_{2^8}$ to $(\mathbb{F}_{2^8})^4$ defined for any $x \in \mathbb{F}_{2^8}$ as follows:

$$\mathsf{SM}_0(x) = \begin{pmatrix} 02 \\ 01 \\ 01 \\ 03 \end{pmatrix} \mathsf{SB}(x), \mathsf{SM}_1(x) = \begin{pmatrix} 03 \\ 02 \\ 01 \\ 01 \end{pmatrix} \mathsf{SB}(x), \mathsf{SM}_2(x) = \begin{pmatrix} 01 \\ 03 \\ 02 \\ 01 \end{pmatrix} \mathsf{SB}(x), \mathsf{SM}_3(x) = \begin{pmatrix} 01 \\ 01 \\ 03 \\ 02 \end{pmatrix} \mathsf{SB}(x).$$

Then, $\mathsf{SM}_0, \mathsf{SM}_1, \mathsf{SM}_2, \mathsf{SM}_3$ are implemented as 8-to-32 bits lookup tables.

Using the fact that ShiftRows is a byte transposition and implementing AddRoundKey as four parallel additions on a 32-bit word, for all $0 \leqslant j \leqslant 3$, the round transformation can be expressed as:

$$state_j \leftarrow \mathsf{SM}_0(s_{0,\mathsf{SR}(0,j)}) \oplus \mathsf{SM}_1(s_{1,\mathsf{SR}(1,j)}) \oplus \mathsf{SM}_2(s_{2,\mathsf{SR}(2,j)}) \oplus \mathsf{SM}_3(s_{3,\mathsf{SR}(3,j)}) \oplus rk_j,$$

where $state_j$ and K_j are the j-th column of respectively the state array and the round key.

This implementation is the standard software implementation of the AES used for various applications. It also inspired Chow et al. for the first white-box implementation of the AES.

2.1 Table-based white-box implementation with small non-linear encodings

The field of White-Box Cryptography was officially introduced in 2002 with two seminal papers of Chow et al. [35, 36]. The papers presented respectively a white-box implementation of the DES and a white-box implementation of the AES following the same *white-box techniques*. For both propositions, the goal was to provide a software implementation of the cryptographic algorithms which is secure in a white-box attack context.

For that, Chow et al. suggested two approaches:

- Dynamic key approach: the input of the algorithm or (and) the key is (are) encrypted.
- Fixed key approach: the key is embedded in the implementation by partial evaluation of key-dependent transformations.

2.1.1 Lookup tables specification

We describe here the application of the second approach for the AES-128 block cipher. We assume a set of 128-bit round keys $\{K^0, K^1, K^2, \ldots, K^{10}\}$ derived from a 128-bit master key K using the AES key schedule algorithm (Section 5.2 of [101]). The following method shows how to embed those round keys in the implementation. Actually, the AES algorithm (described by Algorithm 2) is implemented as a network of encoded lookup tables (LUTs)[1]. The LUTs hide partial evaluations of key-dependent transformations and are calculated by an *AES implementation generator program*. The parameters of such a program are a secret key and a random seed.

[1]The lookup tables are interconnected in a specific order and the execution of the algorithm consists in looking the LUTs following the order.

T-boxes for AddRoundKey and SubBytes. The method of Chow et al. starts with a partial evaluation of the state variables with respect to the round key. For that the transformation AddRoundKey is composed with the SubBytes to define, for $0 \leqslant i, j \leqslant 3$, a key-dependent mapping $\mathsf{T}_{i,j} \colon \mathbb{F}_2^8 \to \mathbb{F}_2^8$ as follows:

$$\forall x \in \mathbb{F}_2^8,\ \mathsf{T}_{i,j}(x) = \mathsf{SB}(x \oplus \widehat{K}_{i,j})$$

For any i, j, the mapping $\mathsf{T}_{i,j}$ is implemented as a 8-to-8 bits LUT and is called a *T-box*. Thus, for any round $1 \leqslant r \leqslant 10$ and for any $x \in \mathbb{F}_2^8$:

$$\mathsf{T}^r_{i,j}(x) = \mathsf{SB}(x \oplus \widehat{K}^{r-1}_{i,j}) \text{ for } 0 \leqslant i, j \leqslant 3 \text{ and } 1 \leqslant r \leqslant 9 \tag{2.1}$$

$$\mathsf{T}^r_{i,j}(x) = \mathsf{SB}(x \oplus \widehat{K}^{r-1}_{i,j}) \oplus K^r_{i,j} \text{ for } 0 \leqslant i, j \leqslant 3 \text{ and } r = 10 \tag{2.2}$$

$\mathsf{TMC}^r_{i,j}$ LUTs for AddRoundKey, SubBytes and MixColumns. The T-boxes are then combined with MixColumns applying the matrix partitioning method to MC which enables to compute MixColumns with four 8-to-32 bits LUTs. The composition of AddRoundKey, SubBytes and MixColumns are implemented with 16 8-to-32 bits LUTs called $\mathsf{TMC}^r_{i,j}$ for $0 \leqslant i, j \leqslant 3$, and defined as follows:

$$\mathsf{TMC}^r_{0,j}(x) = \begin{pmatrix} 02 \\ 01 \\ 01 \\ 03 \end{pmatrix} \mathsf{T}^r_{0,j}(x), \mathsf{TMC}^r_{1,j}(x) = \begin{pmatrix} 03 \\ 02 \\ 01 \\ 01 \end{pmatrix} \mathsf{T}^r_{1,j}(x),$$

$$\mathsf{TMC}^r_{2,j}(x) = \begin{pmatrix} 01 \\ 03 \\ 02 \\ 01 \end{pmatrix} \mathsf{T}^r_{2,j}(x), \mathsf{TMC}^r_{3,j}(x) = \begin{pmatrix} 01 \\ 01 \\ 03 \\ 02 \end{pmatrix} \mathsf{T}^r_{3,j}(x)$$

Let $\mathsf{MC}_0 = \begin{pmatrix} 02 \\ 01 \\ 01 \\ 03 \end{pmatrix}$, $\mathsf{MC}_1 = \begin{pmatrix} 03 \\ 02 \\ 01 \\ 01 \end{pmatrix}$, $\mathsf{MC}_2 = \begin{pmatrix} 01 \\ 03 \\ 02 \\ 01 \end{pmatrix}$ and $\mathsf{MC}_3 = \begin{pmatrix} 01 \\ 01 \\ 03 \\ 02 \end{pmatrix}$ such that $\mathsf{MC} = (\mathsf{MC}_0|\mathsf{MC}_1|\mathsf{MC}_2|\mathsf{MC}_3)$, then for any $x \in \mathbb{F}_2^8$:

$$\mathsf{TMC}^r_{i,j}(x) = \mathsf{MC}_i \cdot \mathsf{T}^r_{i,j}(x) \text{ for } 0 \leqslant i, j \leqslant 3 \tag{2.3}$$

Note that the ShiftRows transformation is implemented by shifting the state before looking at the LUTs, i.e. if $state = (s_{i,j})_{0\leqslant i,j\leqslant 3}$ is the input of the round r, the output of the round is $(\mathsf{TMC}^r_{i,\mathsf{SR}(i,j)}(s_{i,\mathsf{SR}(i,j)}))_{0\leqslant i,j\leqslant 3}$.

XOR tables. Given a four-byte vector (x_0, x_1, x_2, x_3), the linear transformation MC calculates the sum $\mathsf{TMC}^r_{0,j}(x_0) \oplus \mathsf{TMC}^r_{1,j}(x_1) \oplus \mathsf{TMC}^r_{2,j}(x_2) \oplus \mathsf{TMC}^r_{3,j}(x_3)$, i.e. an "exclusive-or" (XOR) on 32-bits words. The XOR of two 32-bits words is decomposed into 8 XOR of *nibbles* (4 bits) and the result of the XOR of two 32-bits words is the concatenation of the results of the XORs of the nibbles.

To summarize the table-based implementation so far, an equivalent representation of the AES algorithm makes call to 144 $\mathsf{TMC}^r_{i,j}$ LUTs, 864 XOR tables and 16 T-boxes (because there is no MixColumns in the last round)[2].

2.1.2 Lookup tables randomization

To avoid key extraction from the lookup tables, especially from the $\mathsf{TMC}^r_{i,j}$ which each depends on one byte of the round key, the LUTs are *randomized*, i.e. they are transformed to look independent from the key bytes. This can be achieved using random bijections called *encodings*.

Definition 1 (Encoding)**.** *Let T be a n-to-m bits mapping. Let f be a n-to-n bits random bijection and g be a m-to-m bits random bijection. The encoded version of $\overline{T}$ is a n-to-m bits mapping defined as*

$$\overline{T} = g \circ T \circ f.$$

*f is called an **input encoding** and g is called an **output encoding**.*

Definition 2 (Compatible mappings)**.** *Let T_1 a n_1-to-m_1 bits mapping and T_2 a n_2-to-m_2 bits mapping. Let $\overline{T_1} = g_1 \circ T_1 \circ f_1$ and $\overline{T_2} = g_2 \circ T_2 \circ f_2$ be two encoded versions of T_1 and T_2 respectively. T_1 and T_2 are compatible if:*

$$\overline{T_2} \circ \overline{T_1} = g_2 \circ T_2 \circ T_1 \circ f_1.$$

That means that $f_2 \circ g_1 = Id$ and so f_2 is the inverse of g_1. We denote $f_2 = (g_1)^{-1}$.

This notion of compatible mappings is fundamental to preserve the correctness of the algorithm. Actually, since the input and output encodings of successive LUT are pairwise annihilating, the *randomized implementation* is equivalent to the simple implementation. Note that the LUTs of the first round are only randomized with output encodings since the input is the plaintext. In the same way, the last LUTs are randomized with only input encodings to output the ciphertext.

Definition 3 (Concatenated encoding)**.** *Let $f_1, \ldots, f_l$ be l n-to-n bits encodings. The mapping $F = (f_1 || \ldots || f_l)$ defined as $F(x_1 || \ldots || x_l) = (f_1(x_1) || \ldots || f_l(x_l))$ is a nl-to-nl bits encoding.*

The definition of a concatenated encoding enables to define a large encoding from several small encodings. This is useful to define the output encoding of $\overline{\mathsf{TMC}}^r_{i,j}$ and the input encoding of the corresponding XOR table. Since the XOR table takes two nibbles from different $\overline{\mathsf{TMC}}^r_{i,j}$ LUTs, the input encoding of the XOR table should be the concatenation of two 4-to-4 bits encodings,

[2] $144 = 16 \times 9$, $864 = 8 \times 3 \times 4 \times 9$.

and so is the output encoding of $\overline{\mathsf{TMC}}^r_{i,j}$. Thus, a 32-to-32 bits encoding is the concatenation of 8 4-to-4 bits encodings. Consequently, the input encoding of $\overline{\mathsf{TMC}}^r_{i,j}$ is the concatenation of two 4-to-4 bits encodings and the output encoding of $\overline{\mathsf{TMC}}^r_{i,j}$ is the concatenation of 8 4-to-4 bits encodings.

Definition 4 (Mixing bijection)**.** *A n-to-n bits mixing bijection is an invertible linear transformation from $\mathbb{F}_2^n$ to itself which achieves diffusion in the implementation. It is represented by a $n \times n$ non-singular matrix over $\mathbb{F}_2$.*

Each key-dependent LUT can be seen as a small-scale block cipher with a key-addition and a substitution. So, it achieves the *confusion* requirement. Following the precept of Shannon, it should also achieve *diffusion*. This is done by the mixing bijections. The mixing bijections are randomly chosen and composed with the LUTs. The composition is done before the application of the input and output encodings. Two types of mixing bijections are defined: input mixing bijections are 8-to-8 bits linear transformations composed at the input of each T-box in rounds 2 to 10 and output mixing bijections are 32-to-32 bits linear transformations composed at the output of each $\mathsf{TMC}^r_{i,j}$ in rounds 1 to 9. Input mixing bijection is denoted $(L^r_{i,j})^{-1}$ while output mixing bijection is denoted $R^r_{i,j}$. Thus, an encoding is now composed of a non-linear part which is the concatenation of 4-bit bijections and a linear part which is composed of the mixing bijection.

Definition 5 (External encodings)**.** *Let $E_K \colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a block cipher instantiated with a secret key $K \in \mathcal{K}$. An encoded version of E_K is defined as:*

$$\overline{E_K} = G \circ E_K \circ F^{-1}$$

where $F^{-1}, G \colon \mathbb{F}_2^{128} \to \mathbb{F}_2^{128}$ are linear mixing bijections.

*F is called an **external input encoding** since it encodes the plaintext and G an **external output encoding** since it encodes the ciphertext.*

An encoded version of the cipher ensures that the plaintext and the ciphertext are not seen in clear. Indeed, for any plaintext P, $F(P)$ is encrypted with $\overline{E_K}$ and the ciphertext is obtained after decoding $G^{-1}(\overline{E_K}(F(P)))$. The external encodings protect the first and last LUTs, i.e. $\mathsf{TMC}^1_{i,j}$ and $\mathsf{T}^{10}_{i,j}$. Since the external encodings are linear, they are represented by a 128×128 matrix over $\mathbb{F}_2$. With abuse of notation, we also write F^{-1} and G for the corresponding matrices. By means of matrix partitioning, they can be partitioned as follows: $F^{-1} = (F_0^{-1} | \dots | F_{15}^{-1})$ and $G = (G_0 | \dots | G_{15})$ with $F_i^{-1}, G_i \colon \mathbb{F}_2^8 \to \mathbb{F}_2^{128}$ for $0 \leqslant i \leqslant 15$. Thus, the external encodings are included to the implementation by defining 8-to-128 bits LUTs.

To summarize, the white-box implementation of the AES is composed of five types of encoded lookup tables:

$\overline{\mathsf{Tin}}_{i,j}$ encodes the 8-to-128 bits mapping $L^1 \circ F^{-1}_{4j+i}$. That means that it removes the external input encoding F^{-1} and apply the linear mixing bijection L^1. Besides, the input encoding of the lookup table is a 8-bit non-linear bijection and the output encoding is a concatenation of 32 4-bit encodings.

$\overline{\mathsf{TMC}}^r_{i,j}$ encodes the 8-to-32 bits mapping $x \mapsto \mathsf{MC}_i \cdot \mathsf{SB}(x \oplus \widehat{K}^{r-1}_{i,j})$. The input encoding is composed of two concatenated 4-bit non-linear bijections and a linear mixing bijection $(L^r_{i,j})^{-1}$. The output encoding is composed of a linear mixing bijection R^r_j and 8 concatenated 4-bit non-linear bijections.

$\overline{\mathsf{T}}^{10}_{i,j}$ encodes the 8-to-8 bits mapping $x \mapsto \mathsf{SB}(x \oplus \widehat{K}^{9}_{i,j}) \oplus K^{10}_{i,j}$. The input encoding is composed of two concatenated 4-bits non-linear bijections and a linear mixing bijection $L^{10}_{i,j}$. The output encoding is composed of G_{4j+i} and 32 concatenated 4-bit non-linear bijections.

$\overline{\mathsf{Txor}}$ encodes the addition of two nibbles. Thus, the input encoding is the concatenation of two 4-bit non-linear bijection and the output encoding is a 4-bit non-linear bijection.

$\overline{\mathsf{Tlin}}^{r}_{i,j}$ encodes the 8-to-32 bits mapping $L^{r+1}_{i,j} \circ (R^{r}_{j})^{-1}_{i}$ given that $(R^{r}_{j})^{-1} = ((R^{r}_{j})^{-1}_{0} | \ldots | (R^{r}_{j})^{-1}_{3})$. In other words, it ensures the compatibility of the encodings between consecutive rounds, i.e. it removes the output encoding and applies the input encoding before looking to the $\overline{\mathsf{TMC}}^{r}_{i,j}$ table.

2.1.3 Security of the implementation

The security of a white-box implementation relies in the hardness to recover the instantiated secret key given the description of the algorithm and the code data. Since they are fully composed of lookup tables, the key extraction security strongly depends on the security of the lookup tables, individually as well as on the whole. The local security and the overall security defined in the following characterize respectively the hardness to recover the secret key from each lookup table independently and the hardness to recover the secret key from the composition of all the lookup tables.

Definition 6 (Local security)**.** *Let K be a k-bit secret key. For n, m two integers, let T_K be a key-dependent n-to-m bits bijective mapping and let $\overline{T} = g \circ T_K \circ f$ be its encoded version with f and g be a n-to-n bits and a m-to-m bits bijections respectively. Then $\overline{T}$ is locally secure if for any bijections f_0, g_0:* $\Pr_K \left[\overline{T} = g_0 \circ T_K \circ f_0\right] = \frac{1}{2^k}$

In other words, local security ensures that an encoded LUT doesn't leak information about the key. If the encodings f and g are uniformly chosen from the set of all bijections, then the encoded LUT is locally secure. In the case of Chow et al.'s white-box implementation, the LUTs are not locally secure. Since the encodings are concatenated encodings, they are not uniformly chosen from the set of all possible bijections, and so induced a bias.

Even if local security is a good measure of the security of LUTs taken independently, it is not sufficient to assess the security of the whole implementation. Chow et al. introduced two metrics to evaluate the robustness of their white-box implementation: *diversity* and *ambiguity*.

Definition 7 (Overall security)**.** *A white-box implementation is (D, A)-secure if it satisfies the following two metrics:*

1. *Diversity D: characterizes the number of distinct implementations given a key.*
2. *Ambiguity A: characterizes the number of candidate keys given an implementation.*

The overall security of a white-box implementation according to diversity and ambiguity only gives an estimation of the complexity of an exhaustive search. If the diversity and ambiguity parameters are sufficiently high, the cost of an exhaustive search on the white-box implementation is substantially greater than an exhaustive search on the key.

The estimation of the parameters (D, A) for Chow et al.'s white-box implementation is given in their paper [35].

2.1.4 The BGE attack

The BGE (Billet, Gilbert and Ech Chatbi) attack is an algebraic attack against Chow et al.'s white-box implementation of the AES described in [12]. The attack successfully extracts the hidden key with a work factor of at most 2^{30}. It consists in recovering the input and output encodings of the $\overline{\mathsf{TMC}}^r_{i,j}$ LUTs up to affine mappings and then recover the affine mappings. Since the encodings are determined, the round key is easily found. The BGE attack shows that even if an attacker cannot extract the key from each lookup table independently, the composition of carefully chosen lookup tables reveals information about the key.

The BGE attack exploits the fact that an attacker has access to an encoded subround of the AES by composing the tables $\overline{\mathsf{TMC}}^r_{i,j}$, $\overline{\mathsf{Tlin}}^r_{i,j}$ and $\overline{\mathsf{Txor}}$. Thus, for each byte x_i, $0 \leqslant i \leqslant 3$ of the input state of the AES subround function, the attacker has access to an encoded version $y_i = f^r_i(x_0, x_1, x_2, x_3)$ of each byte of the output state. Consequently, he is able to define four bijective mappings on $\mathbb{F}_2^8$ by setting to constant values 3 of the four variables x_0, x_1, x_2, x_3. Each mapping is determined by a lookup table which can be inverted. By composing one mapping with the inverse of another mapping, the attacker is able to construct four sets $\mathcal{S}_i$, for $0 \leqslant i \leqslant 3$, of 256 bijective mappings on $\mathbb{F}_2^8$. Each set is determined by a lookup table. Any output encoding is composed of a non-affine component and an affine component and given each set $\mathcal{S}^r_i$, the attacker is able to compute the non-affine component of the output encoding. Thus, the output encodings are made affine by composing them by the inverse of the determined non-affine component. Since, the affine parts of the output encodings of a round r are found, the affine parts of the input encoding of round $r+1$ as well.

We first give the definition of an AES subround. AddRoundKey, SubBytes and ShiftRows are byte operations, i.e. they transform the state array byte by byte. On the contrary, MixColumns operates on four bytes simultaneously: it consists of four parallel multiplication of each column of the state array seen as a vector of $\mathbb{F}^4_{256}$ with the matrix MC. Thus, an AES round consists of four subrounds in parallel where a subround is defined as follows:

Definition 8 (AES subround)**.** *For $0 \leqslant i \leqslant 3$, let $x_i, y_i \in \mathbb{F}_{256}$. The mapping $\mathsf{AES}^r_j \colon \mathbb{F}^4_{256} \to \mathbb{F}^4_{256}$ for $1 \leqslant r \leqslant 9$ and $0 \leqslant j \leqslant 3$, called an AES subround is defined by:*

$$\mathsf{AES}^r_j(x_0, x_1, x_2, x_3) = (y_0, y_1, y_2, y_3) \tag{2.4}$$

where $y_i = \mathsf{mc}_{i,0} \otimes \mathsf{SB}(x_0 \oplus \widehat{K}^{r-1}_{0,j}) \oplus \mathsf{mc}_{i,1} \otimes \mathsf{SB}(x_1 \oplus \widehat{K}^{r-1}_{1,j}) \oplus \mathsf{mc}_{i,2} \otimes \mathsf{SB}(x_2 \oplus \widehat{K}^{r-1}_{2,j}) \oplus \mathsf{mc}_{i,3} \otimes \mathsf{SB}(x_3 \oplus \widehat{K}^{r-1}_{3,j})$ and $\mathsf{mc}_{i,j}$ are the coefficients of the MixColumns matrix.

The attack is composed of the following steps:

Compute the four bijective mappings f^r_i. Let $P^r_{0,j}, P^r_{1,j}, P^r_{2,j}, P^r_{3,j}$ be the four 8-to-8 bits input encodings of an encoded subround and $Q^r_{0,j}, Q^r_{1,j}, Q^r_{2,j}, Q^r_{3,j}$ be the four 8-to-8 bits output encodings such that for any input $(x_0, x_1, x_2, x_3) \in (\mathbb{F}_2^8)^4$, the output of the encoded subround is defined by:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} Q^r_{0,j}(\mathsf{02} \otimes \mathsf{T}^r_{0,j}(P^r_{0,j}(x_0)) \oplus \mathsf{03} \otimes \mathsf{T}^r_{1,j}(P^r_{1,j}(x_1)) \oplus \mathsf{T}^r_{2,j}(P^r_{2,j}(x_2)) \oplus \mathsf{T}^r_{3,j}(P^r_{3,j}(x_3))) \\ Q^r_{1,j}(\mathsf{T}^r_{0,j}(P^r_{0,j}(x_0)) \oplus \mathsf{02} \otimes \mathsf{T}^r_{1,j}(P^r_{1,j}(x_1)) \oplus \mathsf{03} \otimes \mathsf{T}^r_{2,j}(P^r_{2,j}(x_2)) \oplus \mathsf{T}^r_{3,j}(P^r_{3,j}(x_3))) \\ Q^r_{2,j}(\mathsf{T}^r_{0,j}(P^r_{0,j}(x_0)) \oplus \mathsf{T}^r_{1,j}(P^r_{1,j}(x_1)) \oplus \mathsf{02} \otimes \mathsf{T}^r_{2,j}(P^r_{2,j}(x_2)) \oplus \mathsf{03} \otimes \mathsf{T}^r_{3,j}(P^r_{3,j}(x_3))) \\ Q^r_{3,j}(\mathsf{03} \otimes \mathsf{T}^r_{0,j}(P^r_{0,j}(x_0)) \oplus \mathsf{T}^r_{1,j}(P^r_{1,j}(x_1)) \oplus \mathsf{T}^r_{2,j}(P^r_{2,j}(x_2)) \oplus \mathsf{02} \otimes \mathsf{T}^r_{3,j}(P^r_{3,j}(x_3))) \end{pmatrix}$$

Let $c_0, c_1, c_2, c_3 \in \mathbb{F}_2^8$ be constant values, then for any $0 \leqslant i \leqslant 3$, the following mappings are bijective on $\mathbb{F}_2^8$:

$$\begin{aligned} f^r_i(x_0, c_1, c_2, c_3) &= Q^r_{i,j}(\mathsf{02} \otimes \mathsf{T}^r_{0,j}(P^r_{0,j}(x_0)) \oplus \beta^r_{i,c_1,c_2,c_3}) = f^r_{i,c_1,c_2,c_3}(x_0) \\ f^r_i(c_0, x_1, c_2, c_3) &= Q^r_{i,j}(\mathsf{03} \otimes \mathsf{T}^r_{1,j}(P^r_{1,j}(x_1)) \oplus \beta^r_{i,c_0,c_2,c_3}) = f^r_{i,c_0,c_2,c_3}(x_1) \\ f^r_i(c_0, c_1, x_2, c_3) &= Q^r_{i,j}(\mathsf{T}^r_{2,j}(P^r_{2,j}(x_2)) \oplus \beta^r_{i,c_0,c_1,c_3}) = f^r_{i,c_0,c_1,c_3}(x_2) \\ f^r_i(c_0, c_1, c_2, x_3) &= Q^r_{i,j}(\mathsf{T}^r_{3,j}(P^r_{3,j}(x_3)) \oplus \beta^r_{i,c_0,c_1,c_2}) = f^r_{i,c_0,c_1,c_2}(x_3) \end{aligned}$$

where $\beta^r_{i,c_1,c_2,c_3}, \beta^r_{i,c_0,c_2,c_3}, \beta^r_{i,c_0,c_1,c_3}, \beta^r_{i,c_0,c_1,c_2} \in \mathbb{F}_2^8$ are constants which each depends on the values of three constants.

Each mapping $f^r_{i,c_{i_1},c_{i_2},c_{i_3}}$ can be determined by taking all possible values of $x_j \in \mathbb{F}_2^8$, $j \neq i_1, i_2, i_3$ and storing the result of in a 8-to-8 bits lookup table.

Compute the set $\mathcal{S}^r_i$. Let us consider the mapping f^r_{i,c_1,c_2,c_3}. For any $x_3 \in \mathbb{F}_2^8$, the mapping $f^r_{i,c_1,c_2,x_3} = Q^r_{i,j}(\mathsf{02} \otimes \mathsf{T}^r_{0,j}(P^r_{0,j}(x_0)) \oplus \beta^r_{i,c_1,c_2,x_3})$ is bijective since $\beta^r_{i,c_1,c_2,x_3} = \mathsf{03} \otimes \mathsf{T}^r_{1,j}(P^r_{1,j}(c_1)) \oplus \mathsf{T}^r_{2,j}(P^r_{2,j}(c_2)) \oplus \mathsf{T}^r_{3,j}(P^r_{3,j}(x_3))$ takes all possible values in $\mathbb{F}_2^8$. Thus, for each $x_3 \in \mathbb{F}_2^8$, the mapping f^r_{i,c_1,c_2,x_3} is determined by a 8-to-8 bits lookup table and we obtain 256 such lookup tables. Since $f^r_{i,c_1,c_2,c_3} \circ (f^r_{i,c_1,c_2,x_3})^{-1}$ is bijective, the following set $\mathcal{S}^r_i$ is a set of 256 bijective mappings on $\mathbb{F}_2^8$:

$$\mathcal{S}^r_i = \{f^r_{i,c_1,c_2,c_3} \circ (f^r_{i,c_1,c_2,x_3})^{-1} : x_3 \in \mathbb{F}_2^8\} \tag{2.5}$$

$$\begin{aligned} f^r_{i,c_1,c_2,c_3} \circ (f^r_{i,c_1,c_2,x_3})^{-1} &= (Q^r_{i,j} \circ \oplus_{c^r_{i,j}(c_1,c_2,c_3)} \circ \otimes_{\mathsf{02}} \circ \mathsf{T}^r_{0,j} \circ P^r_{0,j}) \circ (P^r_{0,j})^{-1} \circ (\otimes_{\mathsf{02}} \circ T^r_{0,j})^{-1} \circ \\ &\quad \oplus_{c^r_{i,j}(c_1,c_2,x_3)} \circ (Q^r_{i,j})^{-1} \\ &= Q^r_{i,j} \circ \oplus_\delta \circ (Q^r_{i,j})^{-1} \end{aligned}$$

with $\delta = c^r_i(c_1, c_2, c_3) \oplus c^r_i(c_1, c_2, x_3) \in \mathbb{F}_2^8$. Finally, $\mathcal{S}^r_i = \{Q^r_{i,j} \circ \oplus_\delta \circ (Q^r_{i,j})^{-1} : \delta \in \mathbb{F}_2^8\}$.

Find the non-affine component $\widetilde{Q^r_{i,j}}$ of $Q^r_{i,j}$. For each output encoding $Q^r_{i,j}$ we want to build an approximation $\widetilde{Q^r_{i,j}}$ such that $\widetilde{Q^r_{i,j}} = Q^r_{i,j} \circ A^r_{i,j}$ where $A^r_{i,j}$ is an affine mapping, i.e. $Q^r_{i,j} = \widetilde{Q^r_{i,j}} \circ (A^r_{i,j})^{-1}$.

$\mathcal{S}^r_i$ as a vector space is a commutative group under composition.

Let the isomorphism ϕ from $(\mathcal{S}_i^r, \circ)$ to $(\mathbb{F}_2^8, \oplus)$ defined by $\phi(Q_{i,j}^r \circ \oplus_\delta \circ (Q_{i,j}^r)^{-1}) = (\delta_0, \ldots, \delta_7)$. Let $l\colon \mathbb{F}_2^8 \to \mathbb{F}_2^8$ be a linear change of base mapping the canonical vector $(e_{i,0}, \ldots, e_{i,7})$ to a basis $(b_{i,0}, \ldots, b_{i,7})$ of $\mathbb{F}_2^8$, for $1 \leqslant i \leqslant 8$. Define an isomorphism ψ from $(\mathcal{S}_i^r, \circ)$ to $(\mathbb{F}_2^8, \oplus)$ as:

$$\begin{array}{rcl} \psi\colon (\mathcal{S}_i^r, \circ) & \to & (\mathbb{F}_2^8, \oplus) \\ g = Q_{i,j}^r \circ \oplus_\delta \circ (Q_{i,j}^r)^{-1} & \mapsto & \psi(g) = l^{-1}(\phi(g)) \end{array}$$

Thus, ψ allows to recover ϕ up to an unknown linear mapping l^{-1}. To determine ψ, first select $g_1, \ldots, g_8 \in \mathcal{S}_i^r$ such that $(\phi(g_1), \ldots, \phi(g_7))$ is a basis of $\mathbb{F}_2^8$, i.e. $\phi(g_i) = b_i$ for $1 \leqslant i \leqslant 8$. $g_1, \ldots, g_8$ span the space $\mathcal{S}_i^r$ so for all $g \in \mathcal{S}_i^r$, there exists $\epsilon_1, \ldots, \epsilon_8 \in \{0, 1\}$ such that:

$$g = g_8^{\epsilon_8} \circ \ldots \circ g_1^{\epsilon_1} \tag{2.6}$$

Thus, $g_1, \ldots, g_8$ are found by gradually picking elements in $\mathcal{S}_i^r$ and checking Equation 2.6. Now, we can express ψ as follows. For any $g \in \mathcal{S}_i^r$, we have:

$$\begin{array}{rcl} \psi(g) & = & l^{-1}(\phi(g)) = l^{-1}(\phi(g)) = l^{-1}(\phi(g_8^{\epsilon_8} \circ \ldots \circ g_1^{\epsilon_1})) \\ & = & l^{-1}(\epsilon_1 \phi(g_1) \oplus \ldots \oplus \epsilon_8 \phi(g_8)) \\ & = & l^{-1}(\epsilon_1 b_1 \oplus \ldots \oplus \epsilon_8 b_8) \\ & = & \epsilon_1 l^{-1}(b_1) \oplus \ldots \oplus \epsilon_8 l^{-1}(b_8) \\ & = & \epsilon_1 e_1 \oplus \ldots \oplus \epsilon_8 e_8 \end{array}$$

Hence, ψ is entirely determined by taking all elements $g \in \mathcal{S}_i^r$ and computing $\epsilon_1 e_1 \oplus \ldots \oplus \epsilon_8 e_8$. Now let $A_{i,j}^r$ be an affine mapping defined for any $x \in \mathbb{F}_2^8$ as

$$\begin{array}{rcl} A_{i,j}^r(x) & = & l(x \oplus (Q_{i,j}^r \circ l)^{-1}(0)) \\ & = & l(x) \oplus (Q_{i,j}^r)^{-1}(0) \end{array}$$

Then $\widetilde{Q_{i,j}^r} = Q_{i,j}^r \circ A_{i,j}^r$ implies that $(\widetilde{Q_{i,j}^r})^{-1}(0) = 0$. Besides, for any $g \in \mathcal{S}_i^r$,

$$g = \widetilde{Q_{i,j}^r} \circ \oplus_{\psi(g)} \circ (\widetilde{Q_{i,j}^r})^{-1}$$

In particular,

$$\begin{array}{rcl} g(0) & = & \widetilde{Q_{i,j}^r} \circ \oplus_{\psi(g)} \circ (\widetilde{Q_{i,j}^r})^{-1}(0) \\ & = & \widetilde{Q_{i,j}^r}(\psi(g)) \end{array}$$

Hence $\widetilde{Q_{i,j}^r}$ is entirely determined.

Find the affine component $(A_{i,j}^r)^{-1}$ of $Q_{i,j}^r$. Since $\widetilde{Q_{i,j}^r}$ is determined then by composing the output of an encoded subround by $(\widetilde{Q_{i,j}^r})^{-1}$, we got from Equation 2.5:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} (A^r_{0,j})^{-1}(\mathsf{02} \otimes \mathsf{T}^r_{0,j}(P^r_{0,j}(x_0)) \oplus \mathsf{03} \otimes \mathsf{T}^r_{1,j}(P^r_{1,j}(x_1)) \oplus \mathsf{T}^r_{2,j}(P^r_{2,j}(x_2)) \oplus \mathsf{T}^r_{3,j}(P^r_{3,j}(x_3))) \\ (A^r_{1,j})^{-1}(\mathsf{T}^r_{0,j}(P^r_{0,j}(x_0)) \oplus \mathsf{02} \otimes \mathsf{T}^r_{1,j}(P^r_{1,j}(x_1)) \oplus \mathsf{03} \otimes \mathsf{T}^r_{2,j}(P^r_{2,j}(x_2)) \oplus \mathsf{T}^r_{3,j}(P^r_{3,j}(x_3))) \\ (A^r_{2,j})^{-1}(\mathsf{T}^r_{0,j}(P^r_{0,j}(x_0)) \oplus \mathsf{T}^r_{1,j}(P^r_{1,j}(x_1)) \oplus \mathsf{02} \otimes \mathsf{T}^r_{2,j}(P^r_{2,j}(x_2)) \oplus \mathsf{03} \otimes \mathsf{T}^r_{3,j}(P^r_{3,j}(x_3))) \\ (A^r_{3,j})^{-1}(\mathsf{03} \otimes \mathsf{T}^r_{0,j}(P^r_{0,j}(x_0)) \oplus \mathsf{T}^r_{1,j}(P^r_{1,j}(x_1)) \oplus \mathsf{T}^r_{2,j}(P^r_{2,j}(x_2)) \oplus \mathsf{02} \otimes \mathsf{T}^r_{3,j}(P^r_{3,j}(x_3))) \end{pmatrix}$$

with $y_i = f^r_i(x_0, x_1, x_2, x_3)$.

Besides, since $P^{r+1}_{i,j} = (Q^r_{i,j})^{-1} = A^r_{i,j} \circ (\widetilde{Q^r_{i,j}})^{-1}$ then $P^{r+1}_{i,j} \circ \widetilde{Q^r_{i,j}} = A^r_{i,j}$. So the input encoding of the round $r+1$ can be made affine.

For any pair (i, i'), there exists a unique linear mapping $L_{i,i'} : \mathbb{F}_2^8 \to \mathbb{F}_2^8$ and a unique constant $c_{i,i'} \in \mathbb{F}_2^8$ such that for all $x_0 \in \mathbb{F}_2^8$, $f^r_i(x_0, 0, 0, 0) = L_{i,i'}(f^r_{i'}(x_0, 0, 0, 0)) \oplus c_{i,i'}$ [3]. This statement is also true for all x_1, x_2, x_3. $L_{i,i'}$ and $c_{i,i'}$ can be determined by solving an overdefined system of 256×8 linear equations with 72 unknowns. Since $L_{i,i'}$ and $c_{i,i'}$ are recovered, the linear parts of $(A^r_{1,j})^{-1}$, $(A^r_{2,j})^{-1}$, and $(A^r_{3,j})^{-1}$ are computed from the linear part of $(A^r_{0,j})^{-1}$.

The next step of the attack then consists in recovering the constant parts of the affine mappings. Once the output affine encodings are determined, the affine input encodings are also determined. Consequently, the key bytes can be recovered.

2.1.5 Collision attack

Even if the BGE attack is quite efficient with a work factor of 2^{30}, Lepoint et al. [82] showed that it can be improved to obtain a better work factor of 2^{22}. In addition, they presented another attack relying on collisions of encoded variables. From a set of collisions on the output of an 8-bit bijection, the attacker constructs and solves a system where the unknowns are key-dependent 8-bit values which each depends on an 8-bit input encoding. Since the resolution of the system does not give a unique solution, a solution is checked using the algebraic degree of a boolean function. Recall that the algebraic degree of a boolean function gives a measure of how many input bits might simultaneously have an impact on the value of the output. The attack mainly exploits the fact that considering a subround of the AES, the input and output encodings has an algebraic degree of at most 4. Indeed, the encoding are constituted of the concatenation of two 4-bit bijection and a linear mixing bijection. The algebraic degree can be checked by computing the 4-th order derivative of the considered function which is equal to the null function. So, the first phase of the attack consists in removing the output encodings of the first round. Consequently, the input encodings of the second round are removed as well. Then, the second round key is determined by checking the algebraic degree of a specific mapping.

The attack enables to extract the key of Chow et al.'s white-box implementation with a complexity of 2^{22} operations.

For $0 \leqslant j \leqslant 3$, let AES^r_j denotes the j-th subround of the AES, i.e. the output of AES^r_j is the j-th column of the state array. The output of an encoded subround is given by the following mapping:

$$f^r_j = (f^r_{0,j}, f^r_{1,j}, f^r_{2,j}, f^r_{3,j}) = (Q^r_{0,j}, Q^r_{1,j}, Q^r_{2,j}, Q^r_{3,j}) \circ \mathsf{AES}^r_j \circ (P^r_{0,j}, P^r_{1,j}, P^r_{2,j}, P^r_{3,j}) \tag{2.7}$$

[3] Proposition 1 of [12].

Remove the first round's output encodings $Q^1_{i,j}$. The attack first consists in retrieving the 8-bit bijective mapping $S^1_{i,j}$ defined by $S^1_{i,j}(x) = \mathsf{SB}(\widehat{K}^0_{i,j} \oplus P^1_{i,j}(x))$ for all $x \in \mathbb{F}_2^8$. For $r = 1$, Equation 2.7 is equivalent to:

$$f^1_j = (Q^r_{0,j}, Q^r_{1,j}, Q^r_{2,j}, Q^r_{3,j}) \circ \mathsf{MC} \circ (S^1_{0,j}, S^1_{1,j}, S^1_{2,j}, S^1_{3,j}) \tag{2.8}$$

Let us consider the mapping $f^1_{0,j}$. If there exits $\alpha, \beta \in \mathbb{F}_2^8$ such that $\alpha \neq \beta$ and $f^1_{0,j}(\alpha, 0, 0, 0) = f^1_{0,j}(0, \beta, 0, 0, 0)$ then we have:

$$Q^1_{0,j}(\mathsf{02} \otimes S^1_{0,j}(\alpha) \oplus \mathsf{03} \otimes S^1_{1,j}(0) \oplus c) = Q^1_{0,j}(\mathsf{02} \otimes S^1_{0,j}(0) \oplus \mathsf{03} \otimes S^1_{1,j}(\beta) \oplus c) \tag{2.9}$$

with $c = S^1_{2,j}(0) \oplus S^1_{3,j}(0)$.
Since $Q^r_{0,j}$ is a bijective mapping, then Equation 2.9 is equivalent to:

$$\mathsf{02} \otimes (S^1_{0,j}(\alpha) \oplus S^1_{0,j}(0)) \oplus \mathsf{03} \otimes (S^1_{1,j}(0) \oplus S^1_{1,j}(\beta)) = 0 \tag{2.10}$$

Since the mappings $\alpha \mapsto f^1_{0,j}(\alpha, 0, 0, 0)$ and $\beta \mapsto f^1_{0,j}(0, \beta, 0, 0)$ are bijections of $\mathbb{F}_2^8$ then there are exactly 256 pairs (α, β)[4] that yield a collision. Since the 255 non-zero pairs satisfy Equation 2.10, then we get a system of linear equations which enables to recover $S^1_{0,j}$ and $S^1_{1,j}$.

Let $u_0, u_1, \ldots, u_{255}$ and $v_0, \ldots, v_{255}$ denote the unknowns of the system such that $u_l = S^1_{0,j}(l)$ and $v_l = S^1_{1,j}(l)$ for $l \in \mathbb{F}_2^8$. Thus, we rewrite Equation 2.10 as:

$$\mathsf{02} \otimes (u_\alpha \oplus u_0) \oplus \mathsf{03} \otimes (v_0 \oplus v_\beta) = 0 \tag{2.11}$$

By proceeding similarly for $f^1_{1,j}, f^1_{2,j}$ and $f^1_{3,j}$, we obtain a system of 4×255 equations with 2×256 unknowns. Since the system is not of full rank, let $u'_l = u_0 \oplus u_l$ and $v'_l = v_0 \oplus v_l$. Then Equation 2.10 becomes:

$$\mathsf{02} \otimes u'_\alpha \oplus \mathsf{03} \otimes v'_\beta = 0 \tag{2.12}$$

To determine $S^1_{0,j}$, we have to determine u_l for all l. We show in the following how to determine all u_l given a pair (u_0, u_1).

All unknowns of 2.12 can be expressed as a function of one unknown. Without loss of generality, let us express any u'_l as a function of u'_1, so there exists $a_l, b_l \in \mathbb{F}_2$ such that

$$u'_l = a_l \otimes u'_1 \tag{2.13}$$
$$v'_l = b_l \otimes u'_1 \tag{2.14}$$

It follows that

$$u_l = a_l \otimes (u_0 \oplus u_1) \oplus u_0 \tag{2.15}$$
$$v_l = b_l \otimes (u_0 \oplus u_1) \oplus v_0 \tag{2.16}$$

a_l and b_l can be found from Equations 2.13 and 2.14 by solving the system with $u'_1 = 1$. Then, $S^1_{0,j}$ is determined by exhaustive search on the pair (u_0, u_1). To decide if a guess (u^*_0, u^*_1) is a good pair, the following propositions are used:

[4]Including the pair $(0, 0)$.

1. The function $\mathsf{SB}^{-1} \circ S^1_{0,j}$ has an algebraic degree of at most 4
2. If $g\colon \mathbb{F}_2^8 \to \mathbb{F}_2^8$ has an algebraic degree of at most 4 then $\forall x \in \mathbb{F}_2^8$:

$$\phi(x) = \bigoplus_{\alpha=0}^{15} g(x \oplus \alpha) = 0 \tag{2.17}$$

The first proposition is due to the facts that $\mathsf{SB}^{-1} \circ S^1_{0,j} = \oplus_{\widehat{K}^0_{0,j}} \circ P^1_{0,j}$ and that the degree of $P^1_{0,j}$ is at most 4 because it is a concatenation of two 4-bit bijections.

ϕ as defined in the second proposition is a 4-th order derivative of the function g. Since the degree of g is at most 4, its 4-th order derivative is null.

According to these propositions, if (u_0^*, u_1^*) is a good pair, i.e. $u_0^* = \mathsf{SB}(\widehat{K}^0_{0,j} \oplus P^1_{0,j}(0))$ and $u_1^* = \mathsf{SB}(\widehat{K}^0_{0,j} \oplus P^1_{0,j}(1))$ then for all $x \in \mathbb{F}_2^8$

$$\bigoplus_{\alpha=0}^{15} \mathsf{SB}^{-1} \circ S^1_{0,j}(x \oplus \alpha) = 0$$

$$\bigoplus_{\alpha=0}^{15} \mathsf{SB}^{-1}(a_{x\oplus\alpha} \otimes (u_0^* \oplus u_1^*) \oplus u_0^*) = 0$$

So the right pair (u_0, u_1) is determined by checking the above sum for any $x \in \mathbb{F}_2^8$. In the contrary, if the guess for (u_0^*, u_1^*) is wrong, i.e. $u_0^* = (S^1_{0,j})^*(0)$ and $u_1^* = (S^1_{0,j})^*(1)$ such that $(S^1_{0,j})^*(i) \neq S^1_{0,j}(i)$ then $\mathsf{SB} \circ (S^1_{0,j})^*$ has an algebraic degree greater than 4.

Then, $S^1_{1,j}$ is recovered by determining all v_l. This can be done by exhausting all the possible values of v_0 and then checking by means of the 4-th order derivative as above if it is the right value, i.e. $v_0 = S^1_{1,j}(0) = \mathsf{SB}(\widehat{K}^0_{1,j} \oplus P^1_{1,j}(0))$.

Repeating the above method with collisions of the forms $f^1_j(\alpha, 0, 0, 0) = f^1_j(0, 0, \beta, 0)$ and $f^1_j(\alpha, 0, 0, 0) = f^1_j(0, 0, 0, \beta)$ enables to recover $S^1_{2,j}$ and $S^1_{3,j}$ respectively.

Extract the second round key bytes $\widehat{K}^1_{i,j}$. From Equation 2.7 we have:

$$(Q^1_{0,j}, Q^1_{1,j}, Q^1_{2,j}, Q^1_{3,j}) = f^1_j \circ (S^1_{0,j}, S^1_{1,j}, S^1_{2,j}, S^1_{3,j})^{-1} \circ \mathsf{MC}^{-1} \tag{2.18}$$

Then, since the input and encodings are pairwise annihilating, $P^2_{i,j} = (Q^1_{i,j})^{-1}$. The mapping f^2_j becomes:

$$\begin{aligned} f^2_j = (f^2_{0,j}, f^2_{1,j}, f^2_{2,j}, f^2_{3,j}) &= (Q^2_{0,j}, Q^2_{1,j}, Q^2_{2,j}, Q^2_{3,j}) \circ \mathsf{AES}^2_j & (2.19)\\ &= (Q^2_{0,j}, Q^2_{1,j}, Q^2_{2,j}, Q^2_{3,j}) \circ \mathsf{MC} \circ (S^2_{0,j}, S^2_{1,j}, S^2_{2,j}, S^2_{3,j}) & (2.20)\end{aligned}$$

with $S^2_{i,j}(x) = \mathsf{SB}(\widehat{K}^1_{i,j} \oplus x)$.

For any $0 \leqslant i, j \leqslant 3$, $x \mapsto f^2_{i,j}(\widehat{K}^1_{i,j} \oplus \mathsf{SB}^{-1}(x), 0, 0, 0)$ is a 8-bit bijective mapping and has an algebraic degree of at most 4. Consequently, the 4-th order derivative of $x \mapsto f^2_{i,j}(\widehat{K}^1_{i,j} \oplus \mathsf{SB}^{-1}(x), 0, 0, 0)$ equals the null function. So the round key byte $\widehat{K}^1_{i,j}$ is recovered by exhausting all possible value of the key byte and by checking if for any x: $\bigoplus_{\alpha=0}^{15} f^2_{i,j}(\widehat{K}^1_{i,j} \oplus \mathsf{SB}^{-1}(x \oplus \alpha), 0, 0, 0) = 0$.

2.2 System of polynomials-based white-box implementation with linear encodings

In 2006, Bringer et al. proposed a perturbed white-box implementation of a variant of the AES (an AES with secret key-dependent S-boxes). Their method was inspired by the work of Billet et al. on a *traceable block cipher* [11] for which they proposed to add *perturbations* to protect the block cipher against algebraic attacks [28]. In brief, the traceable block cipher of Billet et al. is composed of several rounds each described with a system of polynomials over a finite field. Each system is composed of a inner monomial hidden with linear transformations. The linear transformations are quite similar to the encodings defined above and are pairwise annihilating. Actually, each system is an instance of the Isomorphism of Polynomials (IP) problem. However, Faugere and Perret described in [53] an algorithm to solve the problem efficiently for Billet et al.'s instance. Ding [46] then proposed to add perturbations to the system to avoid the attack of Faugere and Perret.

The method of Bringer et al. is totally different from Chow et al.'s method since they proposed a polynomial-based implementation instead of a lookup-table-based implementation. Their idea is to represent the round function with a system of multivariate polynomials over the field $\mathbb{F}_{2^8}$ and then hide the structure with random polynomials and *perturbation polynomials*. They kept the idea of using pairwise annihilating encodings to mix the polynomials together. The perturbation polynomials spread *controlled faults* along the cipher and the faults are corrected in the last round to preserve the functionality. Actually, four instances of the cipher are executed with well-chosen perturbation polynomials such that two instances among the four agree on the same output. Hence, the correct output which is the ciphertext is decided by a majority vote.

2.2.1 System of polynomials specification

We consider here a variant of the AES with non standard S-boxes, called AEw/oS in [29], i.e. the cipher uses 160 (16 per round) key-dependent S-boxes. Thus, the secret key is composed of the master key which allows to derive 11 round keys and the S-boxes. We consider the equivalent description of the AES algorithm given by Algorithm 2, i.e. AddRoundKey becomes the first transformation of the rounds, for $1 \leqslant r \leqslant 9$ and the last round is composed of two AddRoundKey transformations. Each round of AEw/oS is represented by a system of multivariate polynomials over $\mathbb{F}_{2^8}$. Since a round updates a 16 bytes state, each polynomial has 16 variables and has degree at most 254 because of SubBytes. Note that each polynomial of the systems for $1 \leqslant r \leqslant 9$ depends on 4 variables since each byte after MixColumns depends on 4 bytes.

For any round $1 \leqslant r \leqslant 10$, $\mathscr{S}^r$ is the system of multivariate polynomials over $\mathbb{F}_{2^8}[X_0, \ldots, X_{15}]$ which is evaluated on the state:

$$state \leftarrow \mathscr{S}^r(state)$$

Each system is then extended with perturbation polynomials and random polynomials. Actually, 4 perturbation polynomials and 23 random polynomials over $\mathbb{F}_{2^8}$ are added to any system $\mathscr{S}^r$ for $1 \leqslant r \leqslant 9$. We refer to the paper of Bringer et al. for the details about the choice of the number of polynomials. For the last round, 16 *perturbation cancellation polynomials* are "xored" to the output of $\mathscr{S}^{10}$ to cancel the perturbations. Thus, any system has now 43 polynomials with 43 variables, for $2 \leqslant r \leqslant 9$, 43 polynomials with 16 variables for $r = 1$ and 16 polynomials with 43 variables for $r = 10$. The perturbed systems are finally encoded using secret bijective linear encodings f^r. The encodings are 43-to-43 bits mappings represented by 43×43 non-singular matrices over $\mathbb{F}_{2^8}$.

Perturbation polynomials. Let $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ be four chosen elements of $\mathbb{F}_{2^8}$. Let $\tilde{0}$ be a polynomial in 16 variables over $\mathbb{F}_{2^8}$ which outputs 0 for any message in a chosen subset M_1 of M and a random value for any message $\mathsf{m} \notin \mathsf{M}_1$. The perturbation polynomials $\Phi_1, \Phi_2, \Phi_3, \Phi_4$ are polynomials with 16 variables over $\mathbb{F}_{2^8}$ define, for any $\mathsf{m} \in \mathbb{F}_2^8$, by:

$$\Phi_i(\mathsf{m}) = \tilde{0}(\mathsf{m}) \oplus \alpha_i$$

$\Phi = (\Phi_1, \Phi_2, \Phi_3, \Phi_4)$ is the system of perturbation polynomials defined for fixed values α_1, α_2, α_3, α_4 in $\mathbb{F}_{2^8}$ and for any message $\mathsf{m} \in \mathsf{M}$ as:

$$\begin{aligned} \Phi(\mathsf{m}) &= (\Phi_0(\mathsf{m}), \Phi_1(\mathsf{m}), \Phi_2(\mathsf{m}), \Phi_3(\mathsf{m})) \\ &= (\tilde{0}(\mathsf{m}) \oplus \alpha_1, \tilde{0}(\mathsf{m}) \oplus \alpha_2, \tilde{0}(\mathsf{m}) \oplus \alpha_3, \tilde{0}(\mathsf{m}) \oplus \alpha_4) \end{aligned}$$

That means that $\Phi(\mathsf{m}) = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ or a random value according to the value of the plaintext.

The outputs of the perturbation polynomials are kept throughout the algorithm process via the *identity* Id and canceled in the last round with the system of polynomials O_Φ.

Perturbation cancellation polynomials. For a system of perturbations ϕ defined as above, we denote by O_Φ a system of 16 polynomials in 4 variables over $\mathbb{F}_{2^8}$ which cancel out in the input $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ and equal a random value for any other input.

Random polynomials. We denote by rp^r a system of 23 random polynomials in 43 variables (16 variables for rp^1) over $\mathbb{F}_{2^8}$. Those polynomials are used to hide the polynomials of $\mathscr{S}^r$ and Φ.

Bijective linear encodings. For $1 \leqslant i \leqslant 7$, let A_i^r be a random non-singular 5×5 matrix over $\mathbb{F}_{2^8}$ such that the inverse $(A_i^r)^{-1}$ has exactly two non-zero coefficients in $\mathbb{F}_{2^8}$ on each row, B^r be a random non-singular 8×8 matrix over $\mathbb{F}_{2^8}$ such that the inverse $(B^r)^{-1}$ has at least 7 non-zero coefficients on each row and π^r, σ^r be random permutations from $\mathbb{F}_2^{43}$ to $\mathbb{F}_2^{43}$. f^r is a non-singular 43×43 matrix over $\mathbb{F}_{2^8}$ defined as:

$$f^r = \pi^r \circ \begin{pmatrix} A_1^r & 0 & \dots & 0 & 0 \\ 0 & A_2^r & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & A_7^r & 0 \\ 0 & 0 & \dots & 0 & B^r \end{pmatrix} \circ \sigma^r$$

π^r and σ^r permute the components of the output vector of the inner matrix and the components of the input vector of f^r, respectively. So the permutation σ^r is chosen such that the A_i^r matrices mix the 16 polynomials of $\mathscr{S}^r$ with 19 polynomials of rp^r and the B^r matrix mixes the polynomials $\Phi_1, \Phi_2, \Phi_3, \Phi_4$ with the remaining polynomials of rp^r.

An encoded perturbed system $\mathscr{S}^r$ is defined as follows:

$$\begin{cases} \overline{\mathscr{S}^1} = f^1 \circ (\mathscr{S}^1, \Phi, \mathsf{rp}^1) \\ \overline{\mathscr{S}^r} = f^r \circ (\mathscr{S}^r, \mathsf{Id}, \mathsf{rp}^r) \circ (f^{r-1})^{-1} \text{ for } 2 \leqslant r \leqslant 9 \\ \overline{\mathscr{S}^{10}} = \oplus \circ (\mathscr{S}^{10}, O_\Phi) \circ (f^9)^{-1} \end{cases} \tag{2.21}$$

2.2.2 Correctness of the implementation

Due to the system of perturbation polynomials Φ, the evaluation of the encoded perturbed systems on a random message m will give an incorrect output[5] with a certain probability (according to the definition of Φ). Consequently, the white-box implementation actually consists of four correlated instances of the perturbed implementation. Each instance is constructed with a distinct (but somehow correlated to the other systems) system of perturbation polynomials. The four correlated instances are generated as follows:

- Choose two subspaces M_1 and M_2 of the message space M such that: $\mathsf{M} = \mathsf{M}_1 \cup \mathsf{M}_1^c = \mathsf{M}_2 \cup \mathsf{M}_2^c$ and $|\mathsf{M}_1| = |\mathsf{M}_2|$.
- Define four correlated polynomials $(\tilde{0}_1, \tilde{0}_1^c, \tilde{0}_2, \tilde{0}_2^c)$ such that: $\forall \mathsf{m} \in \mathsf{M}_1$,

$$\begin{cases} \tilde{0}_1(\mathsf{m}) = 0 \text{ if } \mathsf{m} \in \mathsf{M}_1 \\ \tilde{0}_1^c(\mathsf{m}) = 0 \text{ if } \mathsf{m} \in \mathsf{M}_1^c \end{cases} \tag{2.22}$$

  and

$$\begin{cases} \tilde{0}_2(\mathsf{m}) = 0 \text{ if } \mathsf{m} \in \mathsf{M}_2 \\ \tilde{0}_2^c(\mathsf{m}) = 0 \text{ if } \mathsf{m} \in \mathsf{M}_2^c \end{cases} \tag{2.23}$$

- For fixed values $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$, define four correlated perturbation polynomials Φ^1,$(\Phi^1)^c$,Φ^2, $(\Phi^2)^c$ such that: $\forall \mathsf{m} \in \mathsf{M}$

$$\Phi^1(\mathsf{m}) = (\tilde{0}_1(\mathsf{m}) \oplus \alpha_1, \tilde{0}_1(\mathsf{m}) \oplus \alpha_2, \tilde{0}_1(\mathsf{m}) \oplus \alpha_3, \tilde{0}_1(\mathsf{m}) \oplus \alpha_4) \tag{2.24}$$
$$(\Phi^1)^c(\mathsf{m}) = (\tilde{0}_1^c(\mathsf{m}) \oplus \alpha_1, \tilde{0}_1^c(\mathsf{m}) \oplus \alpha_2, \tilde{0}_1^c(\mathsf{m}) \oplus \alpha_3, \tilde{0}_1^c(\mathsf{m}) \oplus \alpha_4) \tag{2.25}$$
$$\Phi^2(\mathsf{m}) = (\tilde{0}_2(\mathsf{m}) \oplus \alpha_1, \tilde{0}_2(\mathsf{m}) \oplus \alpha_2, \tilde{0}_2(\mathsf{m}) \oplus \alpha_3, \tilde{0}_2(\mathsf{m}) \oplus \alpha_4) \tag{2.26}$$
$$(\Phi^2)^c(\mathsf{m}) = (\tilde{0}_2^c(\mathsf{m}) \oplus \alpha_1, \tilde{0}_2^c(\mathsf{m}) \oplus \alpha_2, \tilde{0}_2^c(\mathsf{m}) \oplus \alpha_3, \tilde{0}_2^c(\mathsf{m}) \oplus \alpha_4) \tag{2.27}$$

This way, for any plaintext $\mathsf{m} \in \mathsf{M}$:

- If $\mathsf{m} \in \mathsf{M}_1$ and $\mathsf{m} \in \mathsf{M}_2$ then $\Phi^1(\mathsf{m}) = \Phi^2(\mathsf{m}) = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ while $(\Phi^1)^c(\mathsf{m})$ and $(\Phi^2)^c(\mathsf{m})$ are distinct and random.
- If $\mathsf{m} \in \mathsf{M}_1$ and $\mathsf{m} \notin \mathsf{M}_2$ then $\Phi^1(\mathsf{m}) = (\Phi^2)^c(\mathsf{m}) = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ while $(\Phi^1)^c(\mathsf{m})$ and $\Phi^2(\mathsf{m})$ are distinct and random.

The same reasoning holds when exchanging the subscripts 1 and 2 in the conditions above. Thus, for any possible case, there are exactly two perturbation polynomials which output the value $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ while the two others are random and distinct. Hence a majority vote on the outputs of the four instances gives the correct ciphertext.

[5] A ciphertext different from the encryption of m with AESw/os and the instantiated key.

2.2.3 Cryptanalysis by De Mulder et al.

The white-box implementation of AEw/oS was cryptanalyzed by De Mulder et al. in [94]. The attack extracts a set of equivalent keys which comprises 160 8-bit S-boxes. The set of equivalent keys allows the attacker to construct a smaller (in terms of data storage) and invertible implementation that is functionally equivalent to the original cipher. Thus, the attack doesn't constitutes a full break since the instantiated key is not retrieved. However, the set of equivalent keys allows the attacker to invert the implementation[6]. So the implementation is not *one-way*. The notion of *one-wayness* is one of the useful properties of a white-box implementation as defined in [43]. We will see more about the theory of White-Box Cryptography in Chapter 4.

The attack comprises the following phases:

1. A setup phase consists in encrypting a randomly chosen plaintext with the four correlated instances in order to identify one of the two instances that give the correct ciphertext (we recall that the ciphertext is obtained after a majority vote on the outputs of the four instances.). Once the correct instance is identified, store the values of the plaintext, ciphertext and intermediate states.

2. The first phase analyzes the last round represented by the system $\overline{\mathscr{S}^{10}}$. The goal of this phase is to determine the first 16 rows of the input encoding $(f^9)^{-1}$. The coefficients in these rows are used to compute the input of the last round $\mathscr{S}^{10}$.

3. The second phase removes the MixColumns operation of $(f^9)^{-1}$.

4. The third phase structurally decomposes all rounds as in phases 1 and 2, i.e. the secret linear encodings are partially recovered and the MixColumns operation is removed.

5. The last phase computes an equivalent key, i.e. 160 8-bit S-box are computed.

2.3 Table-based white-box implementation with larger linear encodings

The white-box implementation of the AES proposed by Xiao and Lai in 2009 [116] follows the white-box technique of Chow et al. with some improvements. The input and outputs encodings used to encode the LUTs are now $\mathbb{F}_2$-linear and the input size of the LUTs composed of the T-boxes and part of the MixColumns transformation becomes 16 bits instead of 8 bits. Thus, the encodings apply on at least two bytes simultaneously instead of four bits in the white-box implementation of Chow et al. Since the encodings are linear transformations, the XOR operations, originally implemented as 8-to-8 bits LUTs, are now executed on encoded variables. That substantially reduces the number of LUTs and so improve the performance. Besides, ShiftRows is implemented as a matrix multiplication with a 128×128 matrix over $\mathbb{F}_2$. The total size of the white-box implementation is 20502 kB, i.e. is 27 times bigger than Chow et al.'s white-box. However, Xiao and Lai's implementation only makes 80 lookup table and actually performs as well as the conventional software implementation of the AES.

The white-box implementation of Xiao and Lai was claimed by the authors to resist the BGE attack which breaks the white-box implementation of Chow et al. with a practical complexity. The main vulnerability exploited by the BGE attack is that the attacker is able to compose the LUTs and observe the 16 encoded bytes of a round output. Xiao and Lai tried to thwart the

[6]The attacker is able to decrypt even if the white-box implementation is for encryption and vice versa.

attack by encoding two bytes at the same time. However, a generalization of the BGE attack to SPN ciphers was described by Michiels et al. [89] and applies to the white-box implementation of Xiao and Lai with an estimated work factor of 2^{49}.

2.3.1 Lookup tables specification

As in Chow et al.'s white-box implementation, AddRoundKey and SubBytes are combined to define a T-box: for any round r and $\forall x \in \mathbb{F}_2^8$

$\mathsf{T}^r_{i,j}(x) = \mathsf{SB}(x \oplus \widehat{K}^{r-1}_{i,j})$ for $0 \leqslant i, j \leqslant 3$ and $1 \leqslant r \leqslant 9$ and

$\mathsf{T}^r_{i,j}(x) = \mathsf{SB}(x \oplus \widehat{K}^{r-1}_{i,j}) \oplus K^r_{i,j}$, for $0 \leqslant i, j \leqslant 3$ and $r = 10$.

The matrix MC of MixColumns is decomposed into two 4×2 sub-matrices MC_0 and MC_1 such that the transformation:

$$\begin{pmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{pmatrix} \leftarrow \mathsf{MC} \begin{pmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{pmatrix} \text{ becomes } \begin{pmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{pmatrix} \leftarrow \mathsf{MC}_0 \begin{pmatrix} s_{0,j} \\ s_{1,j} \end{pmatrix} \oplus \mathsf{MC}_1 \begin{pmatrix} s_{2,j} \\ s_{3,j} \end{pmatrix}$$

i.e. MC_0 is composed of the first two columns of MC and MC_1 is composed of the two last columns.

For $1 \leqslant r \leqslant 9$, for $0 \leqslant i \leqslant 1$ and $0 \leqslant j \leqslant 3$, $\overline{\mathsf{TMC}}^r_{i,j}$ is then the lookup table composed of $\mathsf{T}^r_{2i,j}$ and $\mathsf{T}^r_{2i+1,j}$, MC_i and the input and output encodings $(L^r_{i,j})^{-1}$ and R^r_j:

$$\overline{\mathsf{TMC}}^r_{i,j} = R^r_j \circ \mathsf{MC}_i \circ (\mathsf{T}^r_{2i,j}, \mathsf{T}^r_{2i+1,j}) \circ (L^r_{i,j})^{-1}$$

- $(L^r_{i,j})^{-1} \colon \mathbb{F}_2^{16} \to \mathbb{F}_2^{16}$ is a linear transformation represented by a 16×16 non-singular matrix over $\mathbb{F}_2$.
- $R^r_j \colon \mathbb{F}_2^{32} \to \mathbb{F}_2^{32}$ is a linear transformation represented by a 32×32 non-singular matrix over $\mathbb{F}_2$.

For the last round, the output encoding R^{10}_j represented as a 32×32 non-singular matrix over $\mathbb{F}_2$ is split into two 32×16 sub-matrices, $R^{10}_{0,j}$ and $R^{10}_{1,j}$. For $0 \leqslant i \leqslant 1$ and $0 \leqslant j \leqslant 3$, an encoded lookup table $\overline{T}^{10}_{i,j}$ is defined by:

$$\overline{T}^{10}_{i,j} = R^{10}_{i,j} \circ (\mathsf{T}^{10}_{2i,j}, \mathsf{T}^{10}_{2i+1,j}) \circ (L^{10}_{i,j})^{-1}$$

Contrary to Chow et al.'s implementation, the ShiftRows transformation is not included in the data-flow but is explicitly computed with a matrix multiplication. Indeed, ShiftRows can be seen as a linear transformation from $\mathbb{F}_2^{128}$ to itself. Thus, it can be represented as a 128×128 non-singular matrix over $\mathbb{F}_2$. To include ShiftRows into the implementation, the matrix SR of ShiftRows is composed with input and output encodings to defined a linear transformation M^r

applied before looking to the $\overline{\mathsf{TMC}}^r_{i,j}$ and $\overline{T}^{10}_{i,j}$ LUTs. M^r is represented by a 128×128 non-singular matrix over $\mathbb{F}_2$ and is defined as follows:

$$M^r = (L^r_{0,0}, L^r_{1,0}, L^r_{0,1}, L^r_{1,1}, L^r_{0,2}, L^r_{1,2}, L^r_{0,3}, L^r_{1,3}) \circ \mathsf{SR} \circ ((R^{r-1}_0)^{-1}, (R^{r-1}_1)^{-1}, (R^{r-1}_2)^{-1}, (R^{r-1}_3)^{-1})$$

where SR denotes the 128×128 non-singular matrix over $\mathbb{F}_2$ representing $\mathsf{ShiftRows}$.

External encodings. The external encodings F and G are 128-to-128 bit linear mixing bijections. Thus, they are represented by 128×128 non-singular matrices over $\mathbb{F}_2$ and are composed with the input and output encodings of the first and the last round respectively to define the corresponding transformations M^1 and M^{11}:

$$\begin{aligned} M^1 &= (L^1_{0,0}, L^1_{1,0}, L^1_{0,1}, L^1_{1,1}, L^1_{0,2}, L^1_{1,2}, L^1_{0,3}, L^1_{1,3}) \circ \mathsf{SR} \circ F^{-1} \\ M^{11} &= G \circ ((R^{10}_0)^{-1}, (R^{10}_1)^{-1}, (R^{10}_2)^{-1}, (R^{10}_3)^{-1}) \end{aligned}$$

2.3.2 Cryptanalysis by De Mulder et al.

A cryptanalysis of Xiao and Lai's white-box implementation was described in [92]. The authors used a modified version of the Linear Equivalence algorithm to extract the key from the implementation as well as the external encodings with a work factor of 2^{32}.

The attack starts with the following observation: any $\overline{\mathsf{TMC}}^1_{i,j}$ LUT is affine equivalent to $S = (\mathsf{SB}, \mathsf{SB})$. That means that there exists 16-to-16 bits affine mappings A and 16-to-32 bits affine mappings B such that $\mathsf{TMC}^1_{i,j} = B \circ S \circ A$. Since $\mathsf{TMC}^1_{i,j} = R^1_j \circ \mathsf{MC}_i \circ (\mathsf{SB}, \mathsf{SB}) \circ \oplus_{\widehat{K}^1_{2i,j} || \widehat{K}^1_{2i+1,j}} \circ (L^1_{i,j})^{-1}$ then $(A, B) = (\oplus_{\widehat{K}^1_{2i,j} || \widehat{K}^1_{2i+1,j}} \circ (L^1_{i,j})^{-1}, R^1_j \circ \mathsf{MC}_i)$ is one affine equivalence. Since B is linear, only A is affine so the problem can be reduced to finding linear equivalences by making $\overline{\mathsf{TMC}}^1_{i,j}$ key-independent. A key-independent mapping $(\overline{\mathsf{TMC}}^1_{i,j})^*$ is defined from $\overline{\mathsf{TMC}}^1_{i,j}$ as follows. Let $x_{i,j}$ such that $\overline{\mathsf{TMC}}^1_{i,j}(x_{i,j}) = 0$ then

$$(\overline{\mathsf{TMC}}^1_{i,j})^* = \overline{\mathsf{TMC}}^1_{i,j} \circ \oplus_{x_{i,j}} \tag{2.28}$$

Let $\overline{\mathsf{SB}}$ be a 8-to-8 bits S-box defined as $\overline{\mathsf{SB}} = \mathsf{SB} \circ \oplus_{\mathsf{52}}$. Then,

$$(\overline{\mathsf{TMC}}^1_{i,j})^* = R^1_j \circ \mathsf{MC}_i \circ (\overline{\mathsf{SB}}, \overline{\mathsf{SB}}) \circ (L^1_{i,j})^{-1} \tag{2.29}$$

Thus, $(\overline{\mathsf{TMC}}^1_{i,j})^*$ and $(\overline{\mathsf{SB}}, \overline{\mathsf{SB}})$ are linearly equivalent with the particular linear equivalence $(A, B) = ((L^1_{i,j})^{-1}, R^1_j \circ \mathsf{MC}_i)$. However, the Linear Equivalence algorithm of [15] finds all the linear equivalences, i.e. all linear mappings (A, B) in the form:

$$(A, B) = (A' \circ (L^1_{i,j})^{-1}, R^1_j \circ \mathsf{MC}_i \circ B') \tag{2.30}$$

where (A', B') are the self-equivalence of $(\overline{\mathsf{SB}}, \overline{\mathsf{SB}})$. Since $(\overline{\mathsf{SB}}, \overline{\mathsf{SB}})$ has 128 self-equivalences, there are 128 linear equivalences in the form (A, B). All these linear equivalences are found with a complexity in $O(2^{44})$. However, only one equivalence is needed: $(A, B) = ((L^1_{i,j})^{-1}, R^1_j \circ \mathsf{MC}_i)$. This linear equivalence is found with a lower complexity by modifying the algorithm of [15].

Build a set $S_l^{i,j}$ associated with $(L_{i,j}^1)^{-1}$. For each linear input encoding $(L_{i,j}^1)^{-1}$, we construct four sets $S_l^{i,j}$, for $0 \leqslant l \leqslant 3$, of encoded values x such that for u_0, u_1 the two 8-bit decoded values, u_1 can be expressed as a function of u_0. In other words, there exits a 8-bit bijective function $f_l^{i,j}$ such that $u_1 = f_l^{i,j}(u_0)$ and $f_l^{i,j}$ is known. The set $S_l^{i,j}$ is defined as follows:

$$S_l^{i,j} = \{x \in \mathbb{F}_2^{16} \colon (L_{i,j}^1)^{-1}(x) = (u_0, u_1) \text{ and } u_1 = f_l^{i,j}(u_0)\} \tag{2.31}$$

Let (v_0, v_1, v_2, v_3) be the output of MC_i. The relation between (u_0, u_1) and (v_0, v_1, v_2, v_3) is given by :

$$v_l = mc_{l,0}^i \otimes \overline{\mathsf{SB}}(u_0) \oplus mc_{l,1}^i(u_1) \otimes \overline{\mathsf{SB}}(u_1) \tag{2.32}$$

with $mc_{l,0}^i, mc_{l,1}^i$ be the coefficients of MC_i. Because of the output encoding R_j^1, we don't have access to the value of v_l. However, we can determine whether they are equal to 0. Indeed, if the input of M^2 is set to 0 except for the output of $\mathsf{idTMC}_{i,j}^1$, then the output bits of M^2 are all 0 except for the bits which correspond to the four bytes v_l in a linearly encoded form. Therefore, if the output of M^2 is 0, then for all $0 \leqslant l \leqslant 3$, $v_l = 0$. Now $v_l = 0$ is equivalent to:

$$mc_{l,0}^i \otimes \overline{\mathsf{SB}}(u_0) \oplus mc_{l,1}^i(u_1) \otimes \overline{\mathsf{SB}}(u_1) = 0 \quad \Leftrightarrow \tag{2.33}$$

$$mc_{l,1}^i(u_1) \otimes \overline{\mathsf{SB}}(u_1) = mc_{l,0}^i \otimes \overline{\mathsf{SB}}(u_0) \quad \Leftrightarrow \tag{2.34}$$

$$\overline{\mathsf{SB}}(u_1) = (mc_{l,1}^i(u_1))^{-1} \otimes mc_{l,0}^i \otimes \overline{\mathsf{SB}}(u_0) \quad \Leftrightarrow \tag{2.35}$$

$$u_1 = (\overline{\mathsf{SB}})^{-1}((mc_{l,1}^i(u_1))^{-1} \otimes mc_{l,0}^i \otimes \overline{\mathsf{SB}}(u_0)) \tag{2.36}$$

Hence, $f_l^{i,j} = (\overline{\mathsf{SB}})^{-1} \circ \otimes_{(mc_{l,1}^i(u_1))^{-1}} \circ \otimes_{mc_{l,0}^i} \circ \overline{\mathsf{SB}}$.

Use the set $S_l^{i,j}$ to start the LE algorithm. Let $S = S_l^{i,j}$ and $f = f_l^{i,j}$ such that $(mc_{l,0}^i, mc_{l,1}^i) = (\mathsf{01}, \mathsf{02})$ or $(\mathsf{02}, \mathsf{03})$. Let $x_0, x_0' \in S$ such that $x_0 \neq x_0'$ and $x_0, x_0' \neq 0$. By definition of S, there exists $u_0, u_0' \in \mathbb{F}_{2^8} \setminus \{0\}$ such that:

$$x_0 = L_{i,j}^1(u_0 || f(u_0))$$
$$x_0' = L_{i,j}^1(u_0' || f(u_0'))$$

Thus,

$$A(x_0) = (u_0 || f(u_0))$$
$$A(x_0') = (u_0' || f(u_0'))$$

The linear equivalence algorithm requires two initial guesses $A^*(x_0)$ and $A^*(x_0')$ which are obtained by trying all possible values of u_0^* and $u_0'^*$. For each pair of initial guesses $(x_0, A^*(x_0))$ and $(x_0', A^*(x_0'))$, the LE algorithm is executed to find the linear equivalences between $(\overline{\mathsf{SB}}, \overline{\mathsf{SB}})$ and $(\overline{\mathsf{TMC}}_{i,j}^1)^*$. If the set of linear equivalences has more than one element, the algorithm is repeated with another $x_0, x_0' \in S$. The linear equivalence $(A, B) = ((L_{i,j}^1)^{-1}, R_j^1 \circ \mathsf{MC}_i)$ is given by the intersection of the two sets.

Extract two bytes of the round key $\widehat{K}^1_{2i,j}$, $\widehat{K}^1_{2i+1,j}$. Given an input $x_{i,j} \in \mathbb{F}_2^{16}$ such that $\overline{\mathsf{TMC}}^1_{i,j}(x_{i,j}) = 0$ we get

$$\widehat{K}^1_{2i,j} || \widehat{K}^1_{2i+1,j} = (L^1_{i,j})^{-1}(x_{i,j}) \oplus (\mathsf{52}||\mathsf{52}) \tag{2.37}$$

2.4 Dual table-based white-box implementation

In 2010, Karroumi proposed a new white-box implementation of the AES [69]. The idea of Karroumi was to hide the structure of the AES to prevent a BGE attack. He used the notion of *dual cipher* which is an equivalent representation of a cipher, i.e. one can swap from one representation to the other using a known transformation. In the case of AES, Karroumi uses the fact that a byte can be seen as an element of $\mathbb{F}_{2^8} \cong \mathbb{F}_2[X]/(P)$ where P is an irreducible polynomial of degree 8 defined by $P(x) = x^8 + x^4 + x^3 + x + 1$. Thus, any byte of the AES state array has a polynomial representation according to P. Since there are 30 irreducible polynomials over $\mathbb{F}_2$ of degree 8, one can use any of these polynomials to construct the field $\mathbb{F}_{2^8}$ and so define a distinct polynomial representation of a byte. An AES cipher which uses one of these distinct polynomial representations is a dual of the conventional AES.

2.4.1 Generate a dual subround

An AES round consists of four subrounds AES^r_j for $0 \leqslant j \leqslant 3$ applied in parallel as defined in Definition 9. A dual AES subround is a subround of the AES that applies on bytes expressed in one of the 30 different representations of $\mathbb{F}_{256}$. For $1 \leqslant l \leqslant 30$, let $R_l \colon \mathbb{F}_{256} \to \mathbb{F}_{256}$ be an isomorphism mapping elements in the AES polynomial representation to elements in one of the polynomial representation of $\mathbb{F}_{256}$. According to the representation, i.e. the polynomial chosen to build $\mathbb{F}_{256}$, $\oplus_l$ and $\otimes_l$ denotes the addition and multiplication in $\mathbb{F}_{256}$ respectively. For $\alpha \in \mathbb{F}^\times_{256}$, let $m_\alpha \colon \mathbb{F}_{256} \to \mathbb{F}_{256}$ be the scalar multiplication in $\mathbb{F}_{256}$, i.e. $\forall x \in \mathbb{F}_{256}, m_\alpha(x) = \alpha x$ and for $0 \leqslant t \leqslant 7$, let $f_t \colon \mathbb{F}_{256} \to \mathbb{F}_{256}$ be the mapping defined by $\forall x \in \mathbb{F}_{256}, f_t(x) = x^{2^t}$.

A dual AES subround is given by the following definition.

Definition 9 (Dual AES subround)**.** *Let $D = \{R_l \circ m_\alpha \circ f_t : 1 \leqslant l \leqslant 30, \alpha \in \mathbb{F}^\times_{256}, 0 \leqslant t \leqslant 7\}$ for R_l, m_α, f_t defined as above. Let $\Delta_{r,j} \in D$ i.e. there exits (l, α, t) such that $\Delta_{r,j} = R_l \circ m_\alpha \circ f_t$. Let $\delta_{r,j} = R_l \circ f_t$. A dual AES subround is a mapping $\mathsf{AES}^{r,\Delta_{r,j}}_j : \mathbb{F}^4_{256} \to \mathbb{F}^4_{256}$ such that for any w_i, v_i in the polynomial representation associated with R_l, $\mathsf{AES}^{r,\Delta_{r,j}}_j(v_0, v_1, v_2, v_3) = (w_0, w_1, w_2, w_3)$ with*

$$w_i = \delta_{r,j}(\mathsf{mc}_{i,0}) \otimes_l \mathsf{SB}_{r,j}(v_0 \oplus_l \Delta_{r,j}(\widehat{K}^{r-1}_{0,j})) \oplus_l \delta_{r,j}(\mathsf{mc}_{i,1}) \otimes_l \mathsf{SB}_{r,j}(v_1 \oplus_l \Delta_{r,j}(\widehat{K}^{r-1}_{1,j})) \tag{2.38}$$

$$\oplus_l \delta_{r,j}(\mathsf{mc}_{i,2}) \otimes_l \mathsf{SB}_{r,j}(v_2 \oplus_l \Delta_{r,j}(\widehat{K}^{r-1}_{2,j})) \otimes_l \delta_{r,j}(\mathsf{mc}_{i,0}) \otimes_l \mathsf{SB}_{r,j}(v_3 \oplus_l \Delta_{r,j}(\widehat{K}^{r-1}_{3,j})) \tag{2.39}$$

where $\mathsf{SB}_{r,j} = \Delta_{r,j} \circ \mathsf{SB} \circ \Delta^{-1}_{r,j}$.

A dual AES subround allows to compute an AES subround because:

$$\mathsf{AES}^{r,\Delta_{r,j}}_j \circ (\Delta_{r,j}, \Delta_{r,j}, \Delta_{r,j}, \Delta_{r,j}) = (\Delta_{r,j}, \Delta_{r,j}, \Delta_{r,j}, \Delta_{r,j}) \circ \mathsf{AES}^r_j. \tag{2.40}$$

So $\mathsf{AES}^r_j = (\Delta^{-1}_{r,j}, \Delta^{-1}_{r,j}, \Delta^{-1}_{r,j}, \Delta^{-1}_{r,j}) \circ \mathsf{AES}^{r,\Delta_{r,j}}_j \circ (\Delta_{r,j}, \Delta_{r,j}, \Delta_{r,j}, \Delta_{r,j})$.

The details of the proof of 2.40 can be found in [91]. The equality holds if $w_i = \Delta_{r,j}(y_i)$ for $0 \leqslant i \leqslant 3$ where x_i, y_i are defined as in Definition 9. This can be obtained by replacing v_i by $\Delta_{r,j}(x_i)$ in the equality 2.38 and by using the facts that for any $a, b \in \mathbb{F}_{256}$, $\Delta_{r,j}(a) \oplus_l \Delta_{r,j}(b) = \Delta_{r,j}(a \oplus b)$ and that $\delta_{r,j}(a) \otimes_l \Delta_{r,j}(b) = \Delta_{r,j}(a \otimes b)$.

Using Definition 9, the dual AES rounds are defined by taking a random $\Delta_{r,j} \in D$ for each subround then by replacing the S-box SB by $\Delta_{r,j} \circ \mathsf{SB} \circ \Delta_{r,j}^{-1}$, a round key byte $\widehat{K}_{i,j}^{r-1}$ by $\Delta_{r,j}(\widehat{K}_{i,j}^{r-1})$ and the coefficients of the $\mathsf{MixColumns}$ matrix $\mathsf{mc}_{i,z}$ by $\delta_{r,j}(\mathsf{mc}_{i,z})$.

To maintain the functionality of the AES, the input of each dual subround is first transformed by $(\Delta_{r,j}, \Delta_{r,j}, \Delta_{r,j}, \Delta_{r,j})$ then the output is transformed by $(\Delta_{r,j}^{-1}, \Delta_{r,j}^{-1}, \Delta_{r,j}^{-1}, \Delta_{r,j}^{-1})$. This last transformation can be combined with the inverse of $\mathsf{ShiftRows}$ to define a transformation called $\mathsf{ChangeDualState}$ which applies the mapping $C_{i,j}^r \colon \mathbb{F}_{256} \to \mathbb{F}_{256}$ to each state byte between $\mathsf{ShiftRows}$ and $\mathsf{AddRoundKey}$. For any round $2 \leqslant r \leqslant 9$ and any $0 \leqslant i, j \leqslant 3$, $C_{i,j}^r$ is defined by:

$$C_{i,j}^r = \Delta_{r,j} \circ \Delta_{r,\mathsf{isr}(i,j)}^{-1}$$

For $r = 1$, $C_{i,j}^1$ simply applies $\Delta_{1,j}$.

2.4.2 White-box implementation

The white-box implementation of Karroumi is obtained using the white-box techniques of Chow et al. on the dual AES cipher. The dual AES cipher is described as a network of encoded lookup tables where the T-boxes become the *dual T-boxes* defined as follows: for $0 \leqslant i, j \leqslant 3$ and for any $x \in \mathbb{F}_{256}$

$$\mathsf{T}_{i,j}^{r,\Delta_{r,j}}(x) = \mathsf{SB}_{r,j}(C_{i,j}^r(x) \oplus_l \Delta_{r,j}(\widehat{K}_{i,j}^{r-1})) \text{ for } 1 \leqslant r \leqslant 9$$
$$\mathsf{T}_{i,j}^{10,\Delta_{10,j}}(x) = \mathsf{SB}_{10,j}(C_{i,j}^{10}(x) \oplus_l \Delta_{10,j}(\widehat{K}_{i,j}^{9})) \oplus_l \Delta_{10,j}(K_{i,j}^{10})$$

The matrix partitioning applied to the matrix MC of $\mathsf{MixColumns}$ enables to define the *dual* $\mathsf{TMC}_{i,j}^r$ and the XOR LUTs used to compute the output of $\mathsf{MixColumns}$.

Karroumi used the same types of encoding to randomize the lookup tables, i.e. the input and output encodings are either composed of a 8-bit linear mixing bijection and a concatenation of two non-linear bijections on $\mathbb{F}_2^4$ or composed of a 32-bit linear mixing bijection and a concatenation of 8 non-linear bijections on $\mathbb{F}_2^4$.

2.4.3 Cryptanalysis by De Mulder et al.

Karroumi estimated the complexity of the BGE attack against his white-box implementation to be $\mathcal{O}\left(2^{93}\right)$. However, De Mulder et al. [93] proved that Karroumi's white-box implementation is actually the same as Chow et al.'s white-box implementation. Indeed, an encoded dual subround can be represented by an encoded AES subround with the same key bytes. Thus, the BGE attack still applies on Karroumi's white-box implementation.

2.5 Table-based white-box implementation with large non-linear encodings

The white-box implementation of Xiao and Lai was pretty promising because of its performance quite similar to the standard software implementation of the AES and its resistance to the BGE

and collision attacks. However, the fact that the encodings are linear can actually be exploited to peel the encodings off the encoded LUTs, thus leaving them unprotected.

The white-box implementation of Xiao and Lai inspired the white-box implementation of Luo et al. [85] which they claimed to resist the attack described by De Mulder. The implementation is quite similar to the table-based implementation of Xiao and Lai except for the type of encodings which is non-linear. Actually, Luo et al. combined the linear encodings of Xiao and Lai with the non-linear encodings of Chow et al. Because of the use of the non-linear encodings, additional lookup tables are added. Actually, since the output encodings of the $\overline{\mathsf{TMC}}^r_{i,j}$ LUTs are non-linear, the XOR operations can no longer be computed on encoded values. Thus, the addition of two 32-bit encoded values is done with four 8-to-128 bits LUTs. Those new LUTs, called TSR by the authors embed the $\mathsf{ShiftRows}$ transformation and replace the matrix multiplication by M^r in the Xiao and Lai's implementation. Then, the 32 128-bit outputs of the TSR LUTS are merged into one 128-bit value with XOR LUTs which are either 8-to-4 bits or 12-to-4 bits.

2.5.1 Lookup tables specification

Let us recall that the white-box implementation of Xiao and Lai is composed of 8 16-to-32 bits LUTs $\overline{\mathsf{TMC}}^r_{i,j}$ for each round. Those LUTs embed the $\mathsf{AddRoundKey}$, $\mathsf{SubBytes}$ and $\mathsf{MixColumns}$ transformations and are encoded with linear encodings $(L^r_{i,j})^{-1}$ and R^r_j. Luo et al. added non-linear bijections to encode the LUTs. The non-linear encodings are concatenation of 4-bit non-linear bijections. Thus, a 16-bit input encoding is composed of four non-linear bijections on $\mathbb{F}_2^4$ and a linear mapping on $\mathbb{F}_2^{16}$. In the same manner, a 32-bit output encoding is composed of a linear mapping on $\mathbb{F}_2^{32}$ and 8 non-linear bijections on $\mathbb{F}_2^4$.

$\overline{\mathsf{TMC}}^r_{i,j}$ LUTs. For any round $2 \leqslant r \leqslant 9$, any $0 \leqslant i \leqslant 1$ and any $0 \leqslant j \leqslant 3$, the encoded LUT $\overline{\mathsf{TMC}}^r_{i,j}$ is defined by:

$$\overline{\mathsf{TMC}}^r_{i,j} = (g^r_{i,j,0}|| \dots ||g^r_{i,j,7}) \circ R^r_j \circ (\mathsf{T}^r_{2i,j}, \mathsf{T}^r_{2i+1,j}) \circ (L^r_{i,j})^{-1} \circ (f^r_{i,j,0}|| \dots ||f^r_{i,j,3}) \tag{2.41}$$

where $f^r_{i,j,l}, g^r_{i,j,l} \colon \mathbb{F}_2^4 \to \mathbb{F}_2^4$, for $0 \leqslant l \leqslant 7$, are 4-to-4 bits non-linear bijections.

$\mathsf{TSR}^r_{j,l}$ LUTs. The 8 32-bit outputs of the $\overline{\mathsf{TMC}}^r_{i,j}$ are added two by two before applying the linear mapping M^{r+1} on the whole 128-bit output. Since the output encodings are not only linear as for Xiao and Lai's implementation, the non-linear output encodings should be removed before the XOR. Thus, the XOR of two 32-bit outputs and the mapping M^{r+1} are combined to define a set of lookup tables that enables to compute the same transformation. For $1 \leqslant r \leqslant 9$, let $\mathsf{TSR}^{r+1}_{j,t}$ be the set of lookup tables that compute the XOR followed by the linear mapping M^{r+1}. The $\mathsf{TSR}^{r+1}_{j,l}$ LUTs come after the $\overline{\mathsf{TMC}}^r_{i,j}$ LUTs.

For $0 \leqslant i \leqslant 1$ and $0 \leqslant j \leqslant 3$, let $b^r_{i,j} \in \mathbb{F}_2^{32}$ be the 8 32-bit outputs of the $\overline{\mathsf{TMC}}^r_{i,j}$ and $a^r_{i,j} \in \mathbb{F}_2^{32}$ be the outputs after the linear mapping R^j. We have $b^r_{i,j} = (g^r_{i,j,0}|| \dots ||g^r_{i,j,7})(a^r_{i,j}) = (b^r_{i,j,0}, \dots, b^r_{i,j,7})$ with $b^r_{i,j,l} = g^r_{i,j,l}(a^r_{i,j,l})$ for $0 \leqslant l \leqslant 7$.

The 128-to-128 bits mapping M^{r+1} is applied on the "decoded" input $(a^r_{0,0} \oplus a^r_{1,0}, a^r_{0,1} \oplus a^r_{1,1}, a^r_{0,2} \oplus a^r_{1,2}, a^r_{0,3} \oplus a^r_{1,3})$ with $a^r_{0,j} \oplus a^r_{1,j} = (g^r_{0,j})^{-1}(b^r_{0,j}) \oplus (g^r_{1,j})^{-1}(b^r_{1,j})$. We have:

$$\begin{aligned} M^{r+1}(a^r_{0,0} \oplus a^r_{1,0}, a^r_{0,1} \oplus a^r_{1,1}, a^r_{0,2} \oplus a^r_{1,2}, a^r_{0,3} \oplus a^r_{1,3}) = M^{r+1}(a^r_{0,0}, a^r_{0,1}, a^r_{0,2}, a^r_{0,3}) \\ \oplus M^{r+1}(a^r_{1,0}, a^r_{1,1}, a^r_{1,2}, a^r_{1,3}) \end{aligned}$$

since M^{r+1} is linear.

Let $x^r = (a^r_{0,0}, a^r_{0,1}, a^r_{0,2}, a^r_{0,3})$ and $y^r = (a^r_{1,0}, a^r_{1,1}, a^r_{1,2}, a^r_{1,3})$. x^r and y^r can be written $x^r = (x^r_0, \ldots, x^r_{31}) \in (\mathbb{F}_2^4)^{32}$ and $y^r = (y^r_0, \ldots, y^r_{31}) \in (\mathbb{F}_2^4)^{32}$. Then using the matrix partitioning method, we get:

$$M^{r+1}(x^r) \oplus M^{r+1}(y^r) = \sum_{s=0}^{31} M^{r+1}_s(x^r_s \oplus y^r_s) \tag{2.42}$$

where M^{r+1}_s, for $0 \leqslant s \leqslant 31$ is 128×4 matrix over $\mathbb{F}_2$. In other words, M^{r+1} which is a 128×128 matrix over $\mathbb{F}_2$ is subdivided into 32 sub matrices M^{r+1}_s such that $M^{r+1} = (M^{r+1}_0 | \ldots | M^{r+1}_{31})$.

As a reminder

$$M^{r+1} = (L^{r+1}_{0,0}, L^{r+1}_{1,0}, L^{r+1}_{0,1}, L^{r+1}_{1,1}, L^{r+1}_{0,2}, L^{r+1}_{1,2}, L^{r+1}_{0,3}, L^{r+1}_{1,3}) \circ \mathsf{SR} \circ ((R^r_0)^{-1}, (R^r_1)^{-1}, (R^r_2)^{-1}, (R^r_3)^{-1})$$

where $L^{r+1}_{i,j}$ is a linear mapping represented by a 16×16 matrix over $\mathbb{F}_2$ so $L^{r+1} = (L^{r+1}_{0,0}, L^{r+1}_{1,0}, L^{r+1}_{0,1}, L^{r+1}_{1,1}, L^{r+1}_{0,2}, L^{r+1}_{1,2}, L^{r+1}_{0,3}, L^{r+1}_{1,3})$ is a 128×128 diagonal matrix over $\mathbb{F}_2$, $(R^r_j)^{-1}$ is a linear mapping represented by a 32×32 matrix over $\mathbb{F}_2$ and SR is the $\mathsf{ShiftRows}$ transformation represented by a 128×128 matrix over $\mathbb{F}_2$. As above the matrices $(R^r_j)^{-1}$ and SR are partitioned as follows:

$$(R^r_j)^{-1}(a^r_{i,j}) = \sum_{l=0}^{7} (R^r_j)^{-1}_l a^r_{i,j,l}$$

where $(R^r_j)^{-1}_l$ is a 32×4 matrix over $\mathbb{F}_2$, and any $c \in \mathbb{F}_2^{128}$ can be written $c = (c_0, c_1, c_2, c_3) \in (\mathbb{F}_2^{32})^4$ such that:

$$\mathsf{SR}(c) = \sum_{j=0}^{3} \mathsf{SR}_j c_j$$

where SR_j is a 128×32 matrix over $\mathbb{F}_2$.

Thus, the mapping $M^{r+1}_s \colon \mathbb{F}_2^4 \to \mathbb{F}_2^{128}$ is defined by:

$$M^{r+1}_s = L^{r+1} \circ \mathsf{SR}_j \circ (\mathsf{R}^r_j)^{-1}_l \tag{2.43}$$

with $j = \lfloor \frac{s}{8} \rfloor$ and $l = s \mod 8$. Replacing the expression of M^{r+1} in the equality 2.42, we get:

$$\begin{aligned} M^{r+1}(x^r) \oplus M^{r+1}(y^r) &= \sum_{s=0}^{31} L^{r+1} \circ \mathsf{SR}_j \circ (\mathsf{R}^r_j)^{-1}_t (x^r_s \oplus y^r_s) && (2.44) \\ &= \sum_{s=0}^{31} L^{r+1} \circ \mathsf{SR}_j ((\mathsf{R}^r_j)^{-1}_l (x^r_s) \oplus (\mathsf{R}^r_j)^{-1}_l (y^r_s)) && (2.45) \end{aligned}$$

Hence, the LUT $\mathsf{TSR}^{r+1}_{j,l}$ is defined, for any $b^r_{0,j,l}$ and $b^r_{1,j,l}$ in $\mathbb{F}_2^4$, by

$$\mathsf{TSR}_{j,l}^{r+1}(b_{0,j,l}^{r}, b_{1,j,l}^{r}) = (h_{j,l,0}^{r+1}|| \dots ||h_{j,l,31}^{r+1}) \circ L^{r+1} \circ \mathsf{SR}_j((\mathsf{R}_j^r)_l^{-1}((g_{0,j,l}^{r})^{-1}(b_{0,j,l}^{r})) \oplus (\mathsf{R}_j^r)_l^{-1}((g_{0,j,l}^{r})^{-1}(b_{1,j,l}^{r})))$$

where $h_{j,l,0}^{r+1}, \dots, h_{j,l,31}^{r+1}$ are non-linear bijections over $\mathbb{F}_2^4$ which encode the output of the lookup table. For $1 \leqslant r \leqslant 9$, there are 32 $\mathsf{TSR}_{j,l}^{r+1}$ LUTs which compute the 4 32-bit XORs followed by the linear mapping M^{r+1}.

The first $\mathsf{TSR}_{j,l}^{1}$ is a bit different and embeds the external input encoding F. Since $M^1 = (L_{0,0}^1, L_{1,0}^1, L_{0,1}^1, L_{1,1}^1, L_{0,2}^1, L_{1,2}^1, L_{0,3}^1, L_{1,3}^1) \circ \mathsf{SR} \circ F$ and using the same matrix partitioning as above to subdivide F such that $F = (F_0| \dots |F_{15})$, we get $M_s^1 = L^1 \circ \mathsf{SR} \circ F_s^{-1}$ with F_s^{-1} a 128×8 matrix over $\mathbb{F}_2$ for $0 \leqslant s \leqslant 15$. Thus, for any inputs $p_{0,j,l}, \dots, p_{1,j,l} \in \mathbb{F}_2^4$ such that $0 \leqslant l \leqslant 3$, we have

$$\mathsf{TSR}_{j,l}^{1}(p_{0,j,l}, p_{1,j,l}) = (h_{j,l,0}^{2}|| \dots ||h_{j,l,31}^{2}) \circ L^1 \circ \mathsf{SR} \circ F_s^{-1}(p_{0,j,l}, p_{1,j,l}) \tag{2.46}$$

In the same way, the last mapping M^{11} and the external output encoding G are partitioned as $M^{11} = (M_0^{11}| \dots |M_{31}^{11})$ and $G = (G_0| \dots |G_{31})$ to define the last LUTs $\mathsf{TSR}_{j,l}^{11}$:

$$M_s^{11} = G_j \circ (R_j^{10})_l^{-1} \tag{2.47}$$

for $0 \leqslant s \leqslant 31$ and $0 \leqslant l \leqslant 7$. For any $b_{0,j,l}^{10}, b_{1,j,l}^{10} \in \mathbb{F}_2^4$,

$$\mathsf{TSR}_{i,j}^{11}(b_{0,j,l}^{10}, b_{1,j,l}^{10}) = G_j((R_j^{10})_t^{-1}((g_{0,j,l}^{10})^{-1}(b_{0,j,l}^{10})) \oplus (R_j^{10})_t^{-1}((g_{1,j,l}^{10})^{-1}(b_{1,j,l}^{10})))$$

2.5.2 Cryptanalysis with Michiels et al.'s generic algorithm

The attack of Michiels et al. [89] on a table-based white-box implementation of a SPN cipher is a generalization of the BGE attack. It consists of three phases. As in the BGE attack, the first phase retrieves the input and output encodings of an encoded subround up to an affine part. In the second phase, all encodings are affine mappings, then an encoded round function is transformed into a Substitution-Affine Transformation cipher (SAT) round function. Using the SAT cipher round function, the third phase finds an affine equivalence between an encoding and the S-box layer. Finally, the round key can be extracted. In all, the third phase requires the Affine Equivalence algorithm for S-boxes and the Linear Equivalence algorithm for matrices.

The attack of Michiels et al. is generic and only requires the following properties:

- The cipher must be a Substitution-Linear Transformation cipher, i.e. it is composed of several rounds where each round is a bijective mapping which consists of a key addition, a S-box layer with parallel S-boxes and a linear layer represented by a non-singular matrix.
- The non-singular matrix must be Maximum Distance separable.
- The attacker has access to an encoded version of each output word of a round.

2.6 Table-based white-box implementation of multiple instances

Baek et al. presented in [5] an analytic toolbox for white-box implementations following the white-box techniques of Chow et al. Their goal was to provide an estimation of the security of a white-box implementation regardless of the type of encodings used. Indeed, the BGE attack requires that the input size of the S-boxes is the same as the input size of the encodings which is not the case of [116, 85]. Consequently, Baek et al. proposed a cryptanalysis method based on the Biryukov-Shamir technique [16] to remove the non-linear part of an encoding and on a specialized affine equivalence algorithm to recover the remaining affine part. According to their analysis, they suggested possible approaches to increase the security level of a white-box implementation. One of these approaches is to implement two or more instances of the cipher in one implementation and to encode those instances with large encodings. This way, multiple plaintexts are simultaneously encrypted by one white-box implementation. Since the storage size depends on the encodings size, they introduced *sparse unsplit encodings* which are large encodings with a special sparse structure. Especially, when the encodings are only affine, the linear part is represented by a sparse matrix. They used the method to propose a white-box implementation of the AES with two AES-128 instances in one. Since regarding their analysis, the complexity to recover the non-linear part of an encoding is much lower than recovering the affine part, they only used affine encodings in their white-box implementation.

For any round $1 \leqslant r \leqslant 10$, let AES^r be the r-th round function of the AES block cipher, i.e. the output of AES^r is the result of the successive applications of AddRoundKey, SubBytes, ShiftRows and MixColumns on an intermediate state. Let $A^r : \mathbb{F}_2^{256} \to \mathbb{F}_2^{256}$ be an invertible affine mapping. For $1 \leqslant r \leqslant 9$, an encoded round function $\overline{\mathsf{AES}^r}$ which hides a 128-bit round key K^{r-1} is defined by:

$$\begin{aligned} \overline{\mathsf{AES}^r} &= (A^{r+1})^{-1} \circ (\mathsf{AES}^r, \mathsf{AES}^r) \circ A^r && (2.48) \\ &= B^r \circ \underbrace{(\mathsf{SB}, \ldots, \mathsf{SB})}_{32} \circ \oplus_{K^{r-1}, K^{r-1}} \circ A^r && (2.49) \end{aligned}$$

with $B^r = (A^{r+1})^{-1} \circ \underbrace{(\mathsf{MC}, \ldots, \mathsf{MC})}_{8} \circ (\mathsf{SR}, \mathsf{SR})$ an affine mapping from $\mathbb{F}_2^{256}$ to $\mathbb{F}_2^{256}$.

If the implementation is encoded using external affine encodings $F, G : \mathbb{F}_2^{256} \to \mathbb{F}_2^{256}$ then:

$$\begin{aligned} \overline{\mathsf{AES}^0} &= (A^1)^{-1} \circ F^{-1} \text{ and} && (2.50) \\ \overline{\mathsf{AES}^{10}} &= G \circ (\mathsf{AES}^{10}, \mathsf{AES}^{10}) \circ A^{10} && (2.51) \end{aligned}$$

where AES^{10} is the last round function with no MixColumns transformation and with two AddRoundKey (corresponding to K^9 and K^{10}). Thus, $B^{10} = G \circ \oplus_{K^{10}} \oplus (\mathsf{SR}, \mathsf{SR})$.

A^r constitutes the input encoding of the round function and B^r the output encoding. The affine mapping A^r is constructed such that the encoding is said to be *sparse unsplit* as defined by the following.

Definition 10 (Sparse unsplit encoding)**.** *Let an affine encoding $A^r : \mathbb{F}_2^{256} \to \mathbb{F}_2^{256}$ of the form $\oplus_{a^r} \circ L_{A^r}$ for an invertible linear mapping $L_{A^r} : \mathbb{F}_2^{256} \to \mathbb{F}_2^{256}$ and $a^r \in \mathbb{F}_2^{256} \setminus \{0\}$. A^r is a sparse unsplit encoding if there exits 64 8×8 matrices $A^r_{i,i+1}$ over $\mathbb{F}_2$, for $0 \leqslant i \leqslant 31$, such that for any $x \in \mathbb{F}_2^{256}$:*

$$A^r(x) = \begin{pmatrix} A^r_{0,0} & A^r_{0,1} & 0 & 0 & \dots & 0 \\ 0 & A^r_{1,1} & A^r_{1,2} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ A^r_{31,0} & 0 & 0 & 0 & \dots & A^r_{31,31} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{31} \end{pmatrix} \oplus \begin{pmatrix} a^r_0 \\ a^r_1 \\ \vdots \\ a^r_{31} \end{pmatrix}$$

For the rest of this section, for $0 \leqslant i \leqslant 31$, the index $i+1$ is taken modulo 32. Since only $i+1 = 32$ becomes 0, we let $i+1$ to simplify the notation. For any $x_i, x_{i+1} \in \mathbb{F}_2^8$, $0 \leqslant i \leqslant 31$, let $\overline{A^r_i}$ be a linear mapping from $\mathbb{F}_2^{16}$ to $\mathbb{F}_2^8$ defined by:

$$\overline{A^r_i}(x_i, x_{i+1}) = \left(A^r_{i,i} | A^r_{i,i+1}\right) \begin{pmatrix} x_i \\ x_{i+1} \end{pmatrix}. \tag{2.52}$$

Even if A^{r+1} is sparse unsplit, $B^r = (A^{r+1})^{-1}$ is unlikely to be sparse. Thus, B^r can be written $B^r = \oplus_{b^r} \circ L_{B^r}$ with L_{B^r} a linear mapping and $b^r \in \mathbb{F}_2^{256}$. With abuse of notation, we also denote by L_{B^r} the matrix representing the linear mapping. L_{B^r} is partitioned into 32 256×8 matrices B^r_i such that for any $x \in \mathbb{F}_2^{256}$:

$$B^r(x) = (B^r_i | \dots | B^r_{31}) \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{31} \end{pmatrix} \oplus \begin{pmatrix} b^r_0 \\ b^r_1 \\ \vdots \\ b^r_{31} \end{pmatrix} \tag{2.53}$$

For any $0 \leqslant i \leqslant 30$, let $c^r_i \in \mathbb{F}_2^{256}$ be 31 random values. Let $c^r_{31} = b^r \oplus c^r_0 \oplus \dots c^r_{30}$. Then, for $0 \leqslant i \leqslant 31$, $\oplus_{c^r_i} \circ B^r_i$ is an affine mapping from $\mathbb{F}_2^8$ to $\mathbb{F}_2^{256}$.

Any round function can be expressed as a sum of 32 16-to-256 bits mappings T^r_i, for $0 \leqslant i \leqslant 31$ defined as follows:

$$T^r_i = \oplus_{c^r_i} \circ B^r_i \circ \mathsf{SB} \circ \oplus_{rk^{r-1}_{i'} \oplus a^r_i} \circ \overline{A^r_i} \tag{2.54}$$

with $i' = i \mod 16$.

For any $x = (x_0, \dots, x_{31}) \in \mathbb{F}_2^{256}$, we have:

$$\overline{\mathsf{AES}^r}(x_0, \dots, x_{31}) = \sum_{i=0}^{31} T^r_i(x_i, x_{i+1}) \tag{2.55}$$

For any $0 \leqslant i \leqslant 31$, let $h_i^r \colon \mathbb{F}_2^8 \to \mathbb{F}_2^{256}$ be a random mapping. A "masked mapping" $\overline{T_i^r}$ is defined as follows:

$$\forall x, y \in \mathbb{F}_2^8,\ \overline{T_i^r}(x,y) = T_i^r(x,y) \oplus h_i^r(x) \oplus h_{i+1}^r(y) \tag{2.56}$$

such that $\overline{T_i^r}(x,0) = T_i^r(x,y) \oplus h_i^r(x) \oplus h_{i+1}^r(y)$ is not affine equivalent to SB while $T_i^r(x,0) = \oplus_{c_i^r} \circ B_i^r \circ \mathsf{SB} \circ \oplus_{rk_{i'}^{r-1} \oplus a_i^r} \circ A_{i,i}^r(x)$ is.

Hence, an encoded round function $\overline{\mathsf{AES}^r}$ is defined by:

$$\overline{\mathsf{AES}^r}(x_0, \ldots, x_{31}) = \sum_{i=0}^{31} \overline{T_i^r}(x_i, x_{i+1}) \tag{2.57}$$

2.6.1 Cryptanalysis by Derbez et al.

Baek et al. estimated a security of 110 bits for their white-box implementation. However, the cryptanalysis of Derbez et al. [44] has a much lower complexity of roughly 2^{31}. Derbez et al. described a variant of the affine equivalence algorithm to construct an equivalent representation of the scheme which is invertible. Thus, they are able to decrypt any ciphertext in roughly 10×2^{30} operations. Even better, they described a key recovery attack which exploits the special structure of the affine encodings. The complexity of the attack is in $\mathcal{O}\left(2^{31}\right)$ which is quite practical.

Equivalent representation of one encoded round. Recall that an encoded round function is defined by:

$$\overline{\mathsf{AES}^r} = B^r \circ \underbrace{(\mathsf{SB}, \ldots, \mathsf{SB})}_{32} \circ \oplus_{K^{r-1}, K^{r-1}} \circ A^r$$

Let us denote by A^r the mapping $\oplus_{K^{r-1}, K^{r-1}} \circ A^r$, i.e. the round key is merged with the affine translation a^r, such that:

$$\overline{\mathsf{AES}^r} = B^r \circ \underbrace{(\mathsf{SB}, \ldots, \mathsf{SB})}_{32} \circ A^r$$

For simplicity, we drop the exponent r which indicates the round number. One can find two affine mappings A' and B' such that $\overline{\mathsf{AES}} = B' \circ (\mathsf{SB}, \ldots, \mathsf{SB}) \circ A'$.

First note that A can be decomposed into "simpler" mappings D and $\mathcal{A}$ as follows: let $\{v_1^i, \ldots, v_8^i\}$ be a basis of $\ker \overline{A_i} = \ker\left(A_{i,i}, A_{i,i+1}\right)$ which is a vector space of dimension 8 over $\mathbb{F}_2^{16}$. Then, given a basis completion $\{v_9^i, \ldots, v_{16}^i\}$, $V_i = \left(v_1^i | \ldots | v_{16}^i\right)$ is a 16×16 matrix over $\mathbb{F}_2$ which represents a linear mapping. Let 0_8 and I_8 be respectively the 8×8 0 matrix and the 8×8 identity matrix over $\mathbb{F}_2$. Then, there exists an 8×8 invertible matrix D_i such that:

$$\overline{A_i} = D_i \circ (0_8 | I_8) \circ V_i^{-1} \tag{2.58}$$

The affine mapping A can be written $A = D \circ \mathcal{A}$ where D and $\mathcal{A}$ are defined as follows: for any $x \in \mathbb{F}_2^{256}$

$$
\begin{aligned}
D(x) &= \begin{pmatrix} D_0 & 0 & \dots & 0 \\ 0 & D_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \dots & D_{31} \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{31} \end{pmatrix} \oplus \begin{pmatrix} d_0 \\ d_1 \\ \vdots \\ d_{31} \end{pmatrix} \oplus \begin{pmatrix} rk_0 \\ rk_1 \\ \vdots \\ rk_{31} \end{pmatrix} \\
&= \begin{pmatrix} \mathcal{D}_0 & 0 & \dots & 0 \\ 0 & \mathcal{D}_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \dots & \mathcal{D}_{31} \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{31} \end{pmatrix} \quad \text{where } \mathcal{D}_i = \oplus_{rk_i} \circ \oplus d_i \circ D_i, \text{ and}
\end{aligned}
$$

$$
\mathcal{A}(x) = \begin{pmatrix} (0_8|I_8) \cdot V_0^{-1} & 0 & \dots & 0 \\ 0 & (0_8|I_8) \cdot V_1^{-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \dots & (0_8|I_8) \cdot V_{31}^{-1} \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{31} \end{pmatrix}
$$

So $\mathcal{A}$ is a linear mapping and D is an affine mapping with a diagonal matrix for the linear part. A can be recovered up to an affine mapping by determining $\mathcal{A}$. Since V_i is constructed with a basis of $\ker \overline{A_i}$, $\mathcal{A}$ is determined once all kernels are computed. This can be done in approximatively 2^{23} table lookups using the lemma 1 in [44].

Once $\mathcal{A}$ is known, it can be used to define a bijective mapping which is affine equivalent to one S-box at once. We show in the following that executing the affine equivalence algorithm allows to retrieve, with complexity in $\mathcal{O}\left(2^{25}\right)$, a set of affine mappings $\mathcal{D}'_i$ which contains the $\mathcal{D}_i$.

For any $x = (x_0, \dots, x_{31}) \in \mathbb{F}_2^{256}$,

$$
\begin{aligned}
\overline{\mathsf{AES}}(x_0,\ldots,x_{31}) &= \sum_{i=0}^{31} T_i(x_i,x_{i+1}) \\
B\circ(\mathsf{SB},\ldots,\mathsf{SB})\circ A(x_0,\ldots,x_{31}) &= \sum_{i=0}^{31} T_i(x_i,x_{i+1}) \\
B\circ(\mathsf{SB},\ldots,\mathsf{SB})\circ A\circ\mathcal{A}^{-1}(0,\ldots,x_j,\ldots,0) &= \sum_{i=0}^{31} T_i\circ\mathcal{A}^{-1}(0,\ldots,x_j,\ldots,0) \\
B\circ(\mathsf{SB},\ldots,\mathsf{SB})\circ A\circ A^{-1}(0,\ldots,\mathcal{D}_j(x_j),\ldots,0) &= \sum_{i=0}^{31} T_i\circ\mathcal{A}^{-1}(0,\ldots,x_j,\ldots,0) \\
B\circ(\mathsf{SB},\ldots,\mathsf{SB})(0,\ldots,\mathcal{D}_j(x_j),\ldots,0) &= \sum_{i=0}^{31} T_i\circ\mathcal{A}^{-1}(0,\ldots,x_j,\ldots,0)
\end{aligned}
$$

Thus, only the j-th S-box is activated and the function $F\colon x_j \mapsto \sum_{i=0}^{31} T_i\circ\mathcal{A}^{-1}(0,\ldots,x_j,\ldots,0)$ is a bijection which is affine equivalent to the j-th S-box. The affine equivalence algorithm of [15] finds all affine equivalences between F and SB and so finds some $\mathcal{D}'_j$. At this stage there are 2^{11} candidates for $\mathcal{D}_j$ but this does not matter since only an equivalent representation of $\overline{\mathsf{AES}}$ is needed.

One can take 32 candidates D'_j, for $0 \leqslant j \leqslant 31$ and recover an affine mapping B' such that $\overline{\mathsf{AES}} = B'\circ(\mathsf{SB},\ldots,\mathsf{SB})\circ A'$. B' is defined as $B' = \oplus_{b'}\circ L_{B'}$. $L_{B'}$ is the linear part of B' and is represented by a 256×256 matrix over $\mathbb{F}_2$. The 256×256 matrix can be split into 32 256×8 sub matrices as follows: $L_{B'} = (B'_0|\ldots|B'_{31})$. Let $e_l = (0,\ldots,1\ldots,0)$ be l-th vector of the canonical basis of $\mathbb{F}_2^8$. For each $0 \leqslant j \leqslant 31$, the l-th column of B'_j is the difference $\Delta z^l = z^0 \oplus z^l$ between $z^0 = \sum_{i=0}^{31} T_i\circ\mathcal{A}^{-1}(0,\ldots,x_j^0,\ldots,0)$ and $z^l = \sum_{i=0}^{31} T_i\circ\mathcal{A}^{-1}(0,\ldots,x_j^l,\ldots,0))$ with $x_j^l = (\mathcal{D}'_j)^{-1}(\mathsf{SB}^{-1}(\mathsf{SB}(\mathcal{D}'_j(x_j^0))\oplus e_l))$ for a random $x_j^0 \in \mathbb{F}_2^8$. Indeed, one can easily verify that $\Delta z^l = B'_j\cdot e_l$.

The affine translation b' of B' is determined as follows: for any $x\in\mathbb{F}_2^{256}$, let $z = L_{B'}\circ(\mathsf{SB},\ldots,\mathsf{SB})\circ D'\circ\mathcal{A}(x)$ and $z' = B'\circ(\mathsf{SB},\ldots,\mathsf{SB})\circ D'\circ\mathcal{A}(x)$. Then $d' = z\oplus z'$.

Finally, an equivalent representation of one encoded round function can be found with complexity in $\mathcal{O}\left(2^{30}\right)$.

Key recovery attack. An equivalent representation enables to decrypt any ciphertext without the knowledge of the key. However, it does not constitute a full break. Derbez et al. showed that the secret key of the AES can be extracted from the white-box implementation of Baek et al. in $\mathcal{O}\left(2^{31}\right)$. For that, the mappings $\mathcal{D}'_j$ are not sufficient, one need to exactly determine the mappings $\mathcal{D}_j$. This can be done by considering successive encoded round functions and using a Meet-In-The-Middle approach. We start with the definitions of two successive round functions $\overline{\mathsf{AES}}^r$ and $\overline{\mathsf{AES}}^{r+1}$:

$$\begin{aligned}\overline{\mathsf{AES}^r} &= B^r \circ (\mathsf{SB}, \ldots, \mathsf{SB}) \circ D^r \circ \mathcal{A}^r \\ \overline{\mathsf{AES}}^{r+1} &= B^{r+1} \circ (\mathsf{SB}, \ldots, \mathsf{SB}) \circ D^{r+1} \circ \mathcal{A}^{r+1}\end{aligned}$$

So the r-th encoded round function can be expressed as:

$$\overline{\mathsf{AES}}^r = (\mathcal{A}^{r+1})^{-1} \circ (D^{r+1})^{-1} \circ (\mathsf{MC} \circ \mathsf{SR}, \mathsf{MC} \circ \mathsf{SR}) \circ (\mathsf{SB}, \ldots, \mathsf{SB}) \circ D^r \circ \mathcal{A}^r \tag{2.59}$$

Since the MixColumns transformation applies on four bytes, the attack recovers four affine mappings $\mathcal{D}_j^{r+1}$ and the corresponding $\mathcal{D}_j^r$, say $\mathcal{D}_0^{r+1}, \mathcal{D}_1^{r+1}, \mathcal{D}_2^{r+1}, \mathcal{D}_3^{r+1}$ and $\mathcal{D}_0^r, \mathcal{D}_5^r, \mathcal{D}_{10}^r, \mathcal{D}_{15}^r$.

For all m possible vectors $x^l = (x_0^l, 0, \ldots, 0)$ with $x_0^l \neq 0 \in \mathbb{F}_2^8$, $0 \leqslant l \leqslant m$, the differences Δw_0^l are computed for each candidate $\mathcal{D}_0^{r*}$ such that:

$$\Delta w_0^l = \mathsf{SB}(\mathcal{D}_0^{r*}(x_0^0)) \oplus \mathsf{SB}(\mathcal{D}_0^{r*}(x_0^l)) \tag{2.60}$$

Then, the result of the MixColumns transformation on the input $(\Delta w_0^l, 0, 0, 0)$ is denoted $(\Delta z_0^l, \Delta z_1^l, \Delta z_2^l, \Delta z_3^l)$. Each candidate $\mathcal{D}_0^{r*}$ is stored in a table and indexed by $(\Delta z_0^1, \ldots, \Delta z_0^m)$.

Then, compute the difference $\Delta y_0^l = y_0^0 \oplus y_0^l$ for $y^l = \mathcal{A}^{r+1} \circ \overline{\mathsf{AES}}^r \circ (\mathcal{A}^r)^{-1}(x^l) = (D^{r+1})^{-1} \circ (\mathsf{MC} \circ \mathsf{SR}, \mathsf{MC} \circ \mathsf{SR}) \circ (\mathsf{SB}, \ldots, \mathsf{SB}) \circ D^r(x^l)$.

For each candidate $\mathcal{D}_0^{r+1*}$ compute $\Delta^*(z_0^l) = D_0^{r+1*}(\Delta y_0^l)$. Since $\Delta y_0^l = (D^{r+1})^{-1}(\Delta z_0^l)$ then $\Delta z_0^l = D_0^{r+1}(\Delta y_0^l)$. So, if for all l, $\Delta^*(z_0^l) = \Delta z_0^l$, then the candidates for D_0^r and D_0^{r+1} are the right ones[7].

All $\mathcal{D}_j^r$ and $\mathcal{D}_j^{r+1}$ are recovered using the same method in $\mathcal{O}\left(2^{17}\right)$. So D^r and D^{r+1} are determined.

While the linear parts of D^r and D^{r+1} are the correct ones, the affine translations are not the good ones. The affine translation d^{r+1} is computed as follows: for any $x \in \mathbb{F}_2^{256}$, let $z = (\mathsf{MC} \circ \mathsf{SR}, \mathsf{MC} \circ \mathsf{SR}) \circ (\mathsf{SB}, \ldots, \mathsf{SB}) \circ D^r \circ \mathcal{A}^r(x)$ and $y = \mathcal{A}^{r+1} \circ \overline{\mathsf{AES}}^r(x)$. Then, $L_{D^{r+1}}^{-1}(z) \oplus d^{r+1} = y$ and $d^{r+1} = L_{D^{r+1}}^{-1}(z) \oplus y$.

Finally, $(D^{r+1})^{-1}(x) = L_{D^{r+1}}(x) \oplus d^{r+1} \oplus (rk^r, rk^r)$, so $(rk^r, rk^r) = (D^{r+1})^{-1}(x) \oplus L_{D^{r+1}}(x) \oplus d^{r+1}$.

[7] The linear parts of $\mathcal{D}_i^r$ and $\mathcal{D}_i^{r+1}$ are recovered.

Conclusion of the chapter

This chapter presented an overview of the published white-box AES implementations and existing attacks. After about twenty years of research, it appears that the tabulation technique is prevailing because of the structure of the AES which makes it hard to represent the cipher with low degree polynomials. So far, the white-box size of the polynomial-based white-box implementation is 568 MB. In comparison, the white-box size of the table-based white-box implementation of Chow et al. is 752 kB which is almost 800 times smaller than the polynomial-based white-box implementation of Bringer et al. However, the main disadvantage of a table-based white-box implementation is the storage requirement which exponentially grows with the input size of the lookup table. Since the table size is tightly linked to the size of the encodings, several types of encodings have been considered. We saw in this chapter that a 8-bit bijection constructed from two 4-bit bijections lead to a vulnerability to a collision attack and that a 8-bit encoding doesn't avoid the BGE attack. Other types of encodings have been proposed, namely large linear and affine encodings. However, such encodings can be recovered with the linear and affine equivalences algorithms. Even if the complexities of the algorithms are exponential in the size of the linear or affine mappings, they can be adapted to reduce the complexity and make the attacks practical. Table 2.6.1 summarizes the different white-box implementations of the AES that have been largely studied by the academic community and the published practical attacks.

This chapter described the first approach for White-Box Cryptography initiated by Chow et al. We conclude this chapter with the assessment that any white-box implementation of the AES cannot provide a long term security unless the encodings are wide enough.

Implementation / Cryptanalysis	Chow et al. (Section 2.1)	Bringer et al.(Section 2.2)	Xiao and Lai (Section 2.3)	Karroumi (Section 2.4)	Luo et al. (Section 2.5)	Baek et al. (Section 2.6)
BGE attack (Section 2.1.4)	✓	✗	✗	✓	✗	✗
Collision attack (Section 2.1.5)	✓	✗	✗	✓	✗	✗
De Mulder's attack (Setion 2.2.3)	✗	✓	✗	✗	✗	✗
Michiels et al.'s generic attack (Section 2.5.2)	✓	✗	✓	✓	✓	✓

Table 2.2: Some published white-box implementations of the AES and their attacks.

Implementation	Size	Best attack complexity
Chow et al. (Section 2.1)	752 kB	$\mathcal{O}\left(2^{22}\right)$
Bringer et al.(Section 2.2)	568 MB	$\mathcal{O}\left(2^{16}\right)$
Xiao and Lai (Section 2.3)	20 MB	$\mathcal{O}\left(2^{32}\right)$
Karroumi (Section 2.4)	752 kB	$\mathcal{O}\left(2^{22}\right)$
Luo et al. (Section 2.5)	25.8 MB	$\mathcal{O}\left(2^{49}\right)$
Baek et al. (Section 2.6)	640 MB	$\mathcal{O}\left(2^{31}\right)$

Table 2.3: Different white-box AES implementations and the corresponding best known attack complexity.

3

Fifty shades of gray-box attacks

"If there is no struggle, there is no progress."

Frederick Douglass

Contents

In this chapter, we come back to the first edition of the WhibOx competition which confirmed the idea that it is actually a hard problem to prove the security of an implementation of the AES in the white-box model. Even worse, the white-box implementations are weak in the gray-box model, in other words, side-channel attacks are successful on most white-box implementations.

The WhibOx competition aimed to confront white-box designers to unknown cryptanalytic tools. To the question: *Are there any AES implementations that have a reasonable security level in the white-box model?* the contest answered *No.* Nevertheless, the competition has enabled research to progress, in particular with regard to design choices. The goal of this chapter is to point out the design choices that led to the success of the attack during the competition and to give an idea of the best choices.

Introduction

At the end of Chapter 2, we conclude that all published white-box implementation suffer from practical attacks which generally rely on the structure of scheme, inter alia, the choice of encod-

ings and on some mathematical backgrounds. Despite this, secure white-box implementations are claimed by the industry (e.g Irdeto Cloakware's Software Protection [66], Intertrust's whiteCryption Secure Key Box, Gemalto's Software Licensing Products) and it is actually hard to evaluate the security of those solutions regarding state-of-the-art attacks because of the lack of design specifications. Besides, the white-box implementations proposed by most companies are *obfuscated* which makes them impossible to attack without reverse-engineering. Surprisingly, Bos et al. presented at CHES 2016 [26] an attack which extracts the secret key of all publicly available white-box implementations regardless of the design specification. The attack is called by the authors *Differential Computation Analysis* (DCA) and is reminiscent of the *Differential Power Analysis* (DPA) which is widely used as a tool for side-channel attacks. The attack relies on the analysis of execution traces which, contrary to DPA, do not represent power consumption but computations (e.g. data read or written, accessed memory addresses, etc). Since, in the white-box model, we assume that the attacker has access to the implementation so they are able to instrument the binary and record intermediate values or addresses. The attack reveals the fact that the way the implementation is randomized does not sufficiently conceal the existing correlation between the secret key and some intermediate variables and appears as a powerful toolbox for white-box cryptanalysis. The paper of Bos et al. opened up a new avenue for the design of secure white-box implementations.

3.1 The WhibOx competition

The first edition of the WhibOx contest was organized by the ECRYPT-CSA consortium as a Capture-The-Flag challenge of CHES 2017. It was dedicated to the academic and industrial communities working on White-Box cryptography. The principle is to submit a C source code implementing the AES encryption which embeds the secret key. The source code is made publicly available and the goal of the contest is to find the hidden secret key. Approximatively a hundred of source codes have been validated, i.e. they fulfilled the rules of the competition given in Table 3.1 and almost all secret keys have been extracted during the contest time-frame. Only one candidate remained unbroken at the end of the competition but Goubin et al. [61] successfully extracted the secret key from the white-box implementation after 28 days of reverse-engineering. Since most of the codes were obfuscated, the main cryptanalytic tools used were Differential Computation Analysis and Differential Fault Analysis (DFA) when DCA failed.

Even if the issue of the competition seems negative regarding the future of White-Box Cryptography, it helps the academic community to analyze the reasons of the DCA success. Thus, Sasdrich et al. [107] showed that the success probability of the DCA can be measured with the Walsh transform. The Walsh transform of a boolean function traditionally gives a measure of the imbalance of the function and the imbalance of a key-dependent function reveals a correlation with the secret key. Following this work, Bock et al. [19] showed that the internal encodings play a fundamental role in the success of DCA.

In addition to the emergence of new cryptanalytic tools, the first edition of the WhibOx competition promotes the appearance of a new approach for white-box implementations, namely a *circuit*-based white-box implementation of the AES. Actually, the "strongest" candidate of WhibOx 2017 named Adoring Poitras was a circuit-based white-box implementation. This approach was initiated by Biryukov and Udovenko from the University of Luxembourg and can be seen as a white-box implementation from circuit implementation. A full description of such approach can be found in Udovenko's thesis [113].

C source code	$\leqslant$ 50 MB
Compilation time	$\leqslant$ 100s
Executable binary	$\leqslant$ 20 MB
Running memory	$\leqslant$ 20 MB
Execution time	$\leqslant$ 1 s

Table 3.1: WhibOx context: requirements for a valid implementation on a reference CPU.

3.2 Differential Computation Analysis

The Differential Computation Analysis has been introduced at CHES 2016 by Bos et al. [26] as the software counterpart of the Differential Power Analysis (DPA) [76]. Instead of measuring a physical information from the device (e.g. the power consumption) during the execution, the authors proposed to exploit program's data (e.g. values or addresses) during the execution since they are available in the white-box model. These data are recorded thanks to a Dynamic Binary Instrumentation (DBI) frameworks such as Pin or Valgrind. A DBI is a method of analyzing a binary at runtime by injecting instrumentation code. The instrumentation code executes as part of the normal instruction and enables to have an insight of the program's state at different point without affecting the original computation effects. The so-called *execution traces* which likely contain information about (a part of) the key are then exploited through statistical analysis.

Remark 1. *Even if DCA was intented to mimic the principle of DPA, it was then extended to any statistical analysis on execution traces, including the Correlation Power Analysis (CPA).*

Focusing on white-box implementations of the AES, the goal of the attack is to extract the first round key which is also the master key, by recovering each byte individually. As for DPA, the principle of DCA is to determine if a predicted key-dependent value matches the traces obtained during the program's execution. So first, one needs to identify the target (key-dependent) variables. Those variables might be values, so we talk about *a value-based DCA* or addresses for an *address-based DCA*. In the case of the AES, a target variable is generally one byte after the first SubBytes because it directly follows the AddRoundKey transformation and so each state byte depends on one byte of the first round key. Thus, the attack first consists in identifying the value of this target variable for the target implementation. The attack is executed on the binary of an (obfuscated) program which contains a white-box implementation of a standard algorithm[8].

The DCA attack comprises of the following steps:

1. Execute the binary on a random input and record a trace of the whole execution, i.e. all accessed addresses and data. If the Address Space Layout Randomization[9] is enabled, it is just disabled before the record. This single record is done to identify the algorithm implemented and the address range of its instructions. Indeed, if the specification of the

[8]The DCA attack is also applicable to the DES.

[9]The Address Space Layout Randomization is a technique which randomly arranges the address space positions of data areas, e.g. the positions of the stack, heap and libraries.

algorithm is public, which is generally the case for standards like DES or AES, the algorithm is recognized thanks to patterns that are visible in a graphical representation of the trace.

2. Once the algorithm is identified and the address range of the target instructions is determined, the binary is executed several times on random inputs. At this stage, the traces consist in bytes read from the memory, bytes written on the stack, the least significant byte of memory addresses, etc.

3. The traces which are a list of bytes are serialized (converted into bits) to fit existing DPA tools.

4. A statistical analysis gives the ranking of each possible byte: the byte value ranked one is the most probable value for the key byte.

Difference of means. As for DPA, the statistical analysis performed on the execution traces might be the *difference of means*. It can be summarized as follows. Let $\mathbf{v}$ be an execution trace composed of t samples, i.e. $\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_t)$. Each sample corresponds to one bit (after the serialization) of the target value, e.g. a memory address or a stack value. Let N be the number of traces recorded at Step 2.

A selection function $\textsc{Sel}\colon \mathbb{F}_2^8 \times \mathbb{F}_2^8 \times \{0, \ldots, 7\} \to \mathbb{F}_2$ is defined as:

$$\textsc{Sel}(p, k^*, i) = s_i \text{ where } s = (s_0, \ldots, s_7) = \mathsf{SB}(p \oplus k^*)$$

For a key byte hypothesis k^*, the selection function outputs one bit, called the *target bit* of the target state byte. Then, the selection function is used to sort the collected traces into two sets A_0^i and A_1^i according to the value of the bit s_i: if the bit is 0, the trace is put into the set A_0^i, otherwise it is put into A_1^i. Let $\overline{A_0^i}$ and $\overline{A_1^i}$ be the mean of A_0^i and A_1^i respectively. The difference of means is given by:

$$\Delta^i = |\overline{A_0^i} - \overline{A_1^i}|$$

The difference of means enables to observe a correlation between the output of the selection function and the trace sample. Indeed, a *peak* in the graphical representation of Δ^i appears when the key hypothesis is correct. Let δ^i denotes the peak value in the graphical representation of Δ^i. According to the design choices, only some bits s_i might reveal information about the key. The best target bit can be chosen by observing the highest peak value among all peak values of all Δ^i, for $0 \leqslant i \leqslant 7$. In other words, if δ^{i^*}, $i^* \in \{0, \ldots, 7\}$ is the highest peak among all peak values δ^i then Δ^{i^*} is chosen as the best difference of means for a key hypothesis k^*. Once each key guess is assigned with the best difference of means and the highest peak value, the key hypothesis with the highest peak value is chosen as the best key candidate, and so is ranked 1.

Pearson's correlation coefficient. A Correlation Power Analysis (CPA) is a statistical power analysis which computes the correlation between power samples and the Hamming weight of the handled data (*Hamming weight model*) $\mathsf{wt}(D)$. Every time a bit switches from 0 to 1 or 1 to 0, some current is required to charge or discharge the data lines. Thus, the amount of current required for a data D is linked to the number of 1 in the data, which is given by the Hamming weight of D.

However, given a reference state R the *Hamming distance model* $\mathsf{wt}(R \oplus D)$ is actually more relevant and includes the Hamming weight model for $R = 0$. The idea of CPA is to assume a linear correlation between the power consumption and $\mathsf{wt}(R \oplus D)$ and to compute the corresponding correlation coefficient. It is given by the Pearson's correlation coefficient defined as follows.

Definition 11 (Pearson's correlation coefficient)**.** *Let X and Y be two random variables. Let $\mathbb{E}[X]$ and σ_X denote the expectation and the standard deviation of X respectively and let $\mathsf{Cov}(X, Y)$ denotes the covariance of X and Y. The Pearson's correlation coefficient $\rho(X, Y)$ is defined by:*

$$\begin{aligned} \rho(X,Y) &= \frac{\mathsf{Cov}(X,Y)}{\sigma_X \cdot \sigma_Y} \\ &= \frac{\mathbb{E}[XY] - \mathbb{E}[X] \cdot \mathbb{E}[Y]}{\sqrt{\mathbb{E}[X^2] - \mathbb{E}^2[X]} \cdot \sqrt{\mathbb{E}[Y^2] - \mathbb{E}^2[Y]}} \end{aligned}$$

In the context of DCA, one doesn't need to consider the Hamming distance as a leakage model since each bit of the target data can be exploited individually. Thus, a mono-bit CPA is done where the leakage model is simply the identity function.

DCA requirements. Even if the details of the implementation are not required to execute the attack, some conditions must be fulfilled for the success of a DCA attack:

(i) The program can be executed on chosen inputs several times.

(ii) Either input or output external encoding (if any) is known.

The second condition (ii) ensures that the attacker controls the input or output of the algorithm. Actually, this is essential to make a prediction on the value of a target key-dependent variable.

3.2.1 DCA attack success

The DCA attack succeeds if the right key bytes have the best ranking at the end of Step 4. This highlights a strong correlation between the predicted values and the traces. Such correlation can be measured with the Walsh transform as explained in [107].

Definition 12 (Boolean function)**.** *A boolean function f with n variables is a function from $\mathbb{F}_2^n$ to $\mathbb{F}_2$.*

- *A boolean function f has a unique representation as a multivariate polynomial with n variables over $\mathbb{F}_2$ called the Algebraic Normal Form (ANF). The ANF is given by a set of coefficients $a_u \in \mathbb{F}_2$ for $u \in \{0,1\}^n$ as:*

$$f(x) = \sum_{u \in \{0,1\}^n} a_u x^u \text{ with } x^u = \prod_{i=1}^{n} x_i^{u_i}$$

- *The algebraic degree of f, denoted by $\deg f$ is the number of variables x_i in the highest order term with a non-zero coefficient.*

- *The weight of f, denoted by $\omega(f)$ is the cardinality of the set $A = \{x \in \mathbb{F}_2^n : f(x) = 1\}$.*

Definition 13 (Boolean function imbalance)**.** *Let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ be a boolean function. We say that f is balanced if $\omega(f) = 2^{n-1}$.*

The bias or imbalance of f is defined as:

$$B(f) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)} = 2^n - 2 \cdot \omega(f)$$

It is easy to see that a boolean function f is balanced iff $B(f) = 0$.

Definition 14 (Walsh transform)**.** *Let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ be a boolean function and for any $x, \alpha \in \mathbb{F}_2^n$, let $\langle \cdot, \cdot \rangle$ be the scalar product in $\mathbb{F}_2^n$, i.e. $\langle x, \alpha \rangle = x_1 \cdot \alpha_1 \oplus \ldots \oplus x_n \cdot \alpha_n$.*

The Walsh transform of the function f is defined as:

$$\begin{array}{rcl} W_f\colon \mathbb{F}_2^n & \to & \mathbb{R} \\ \alpha & \mapsto & \displaystyle\sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus \langle x, \alpha \rangle} \end{array}$$

The Walsh transform $W_f(\alpha)$ gives the imbalance of the function $x \mapsto f(x) \oplus \langle x, \alpha \rangle$ and is used to define a m-resilient function.

Definition 15 (m-resilient function)**.** *Let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ be a boolean function and let m be an integer. If for any $\alpha \in \mathbb{F}_2^n$ such that the Hamming weight of α equals at most m we have $W_f(\alpha) = 0$ then f is called a balance m-th order correlation immune function of an m-resilient function.*

That means that the Walsh transform enables to compute the correlation between $f(x)$ and $\langle x, \alpha \rangle$ by taking $\frac{1}{2^n} W_f(\alpha)$ as the correlation coefficient.

Let us consider the white-box implementation of Chow et al. described in Section 2.1. For a plaintext byte $p \in \mathbb{F}_2^8$, let $x = \mathsf{SB}(p \oplus k)$ where $k \in \mathbb{F}_2^8$ is a first round key byte. For $1 \leqslant l \leqslant 32$, let $f_l(x)$ be a boolean function which on the input x gives the l-th bit of the output of $\overline{\mathsf{TMC}}^1_{i,j}$. Then computing the Walsh transforms of all f_l for all $\alpha \in \mathbb{F}_2^8$ allows to detect an imbalance of some functions (if any). If such an imbalance exists then the corresponding bit leaks information about the key byte. Consequently, the success of the DCA attack is due to the existence of biased key-dependent functions in the white-box implementation. The analysis of the Walsh transform enables to decide whether an encoding is a bad choice. If some of the boolean functions f_l are not m-resilient then a DCA attack will likely succeed. Nonetheless, having all boolean functions to be f_l m-resilient is not a sufficient condition to avoid DCA.

Further analysis are required to tightly estimate the success probability of the DCA attack and the number of traces which maximizes the probability. Informally, the success probability of the DCA is the probability that the right key is distinguished, i.e. has the best ranking among all possible keys, at the end of the analysis. Thus, a *DCA distinguisher* attributes a *score* to a key guess according to the value of a correlation coefficient.

Definition 16 (Boolean correlation)**.** *Let f, g be two balanced boolean functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. The correlation coefficient between f and g is defined as:*

$$\begin{array}{rcl} \mathsf{Cor}(f, g) & = & \dfrac{1}{2^n} \displaystyle\sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus g(x)} \\ & = & \dfrac{1}{2^n} B(f + g) \end{array}$$

The target variable that has been considered so far is a byte after the SubBytes transformation of the first round. This is because it depends on one byte of the first round key. Actually, any key-dependent variable might be considered (e.g. one byte after the MixColumns of the first round depends on four bytes of the first round key). To generalize, we assume a key-dependent function $f\colon \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n$. For any input $p \in \mathbb{F}_2^n$ and any key guess $k^* \in \mathbb{F}_2^n$, the selection function $\textsc{Sel}(p, k^*, i)$ outputs f_i^* which is the i-th bit of $f(p, k^*)$. Let $\overline{f}\colon \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^m$ be the encoded key-dependent function, i.e. the function which is the composition of f and a m-to-m bits bijective encoding.

Definition 17 (DCA distinguisher)**.** *Let $\mathbf{v} = (v_1, \ldots, v_t)$ be a trace obtained for an input p. For $1 \leqslant j \leqslant t$, let V_j be the boolean function which returns v_j on the input p. For a key guess k^*, the DCA distinguisher is defined as:*

$$\delta_{k^*} = \max_{1 \leqslant j \leqslant t} |\mathsf{Cor}(f_i^*, V_j)| \tag{3.1}$$

If $\mathbf{v}$ matches the target variable, i.e. $V = (V_1, \ldots, V_m) = \overline{f}$ then:

$$\begin{aligned} \max_{1 \leqslant j \leqslant m} |\mathsf{Cor}(f_i^*, V_j)| &= \max_{1 \leqslant j \leqslant m} |\mathsf{Cor}(f_i^*, \overline{f}_j)| && (3.2) \\ &= \max_{1 \leqslant j \leqslant m} \left(\frac{1}{2^n} B(f_i^* + \overline{f}_j) \right) && (3.3) \end{aligned}$$

The success probability of a DCA attack is expressed as follows.

Definition 18 (DCA success)**.** *Let k^* be the right key guess and let $k^\times$ be a wrong key guess. The probability of success of DCA is defined as:*

$$p = \Pr\left[\max_{1 \leqslant j \leqslant m} |\mathsf{Cor}(f_i^*, \overline{f}_j)| > \max_{1 \leqslant j \leqslant m, k^\times} |\mathsf{Cor}(f_i^\times, \overline{f}_j)| \right] \tag{3.4}$$

The probability that $\max_{1 \leqslant j \leqslant t} |\mathsf{Cor}(f_i^*, \overline{f}_j)| > \max_{1 \leqslant j \leqslant t, k^\times} |\mathsf{Cor}(f_i^\times, \overline{f}_j)|$ only depends on the random generation of the encoding. If this event occurs then a DCA attack will succeed providing that the number of traces is high enough.

Proposition 1. *If $n \geqslant 2m + 2$ then:*

$$p \approx 1 - \frac{\binom{2^{m-1}}{2^{m-2}}^{2m}}{\binom{2^{m-1}}{2^m}^m} \tag{3.5}$$

Proposition 1 means that if the size of the encoding m is approximatively twice smaller than the size of the input n, then the probability that DCA succeeds is quite high. For instance, when $m = 4$ and $n = 8$ as for Chow et al's white-box implementation, the probability p that $\max_{1 \leqslant j \leqslant m} |\mathsf{Cor}(f_i^*, \overline{f}_j)| > \max_{1 \leqslant j \leqslant m, k^\times} |\mathsf{Cor}(f_i^\times, \overline{f}_j)|$ equals 0.9264. In the contrary, if $n = 8$ and $m = 8$, then the probability is almost 0. Table 3.2.1 gives the probability p according to the value of n and m.

(n, m)	p
$(\mathbf{8}, \mathbf{4})$	0.9264
$(\mathbf{8}, \mathbf{5})$	0.7598
$(\mathbf{8}, \mathbf{6})$	0.3556
$(\mathbf{8}, \mathbf{7})$	0.0716
$(\mathbf{8}, \mathbf{8})$	0.0025

Table 3.2: Probability of success of the DCA attack according to the input size n and the encoding size m [104].

Proposition 2 (Trace complexity)**.** *The number of traces for a successful DCA attack is given by:*

$$N = C \left(\frac{1}{\delta_{k^*}} \right)^2 \tag{3.6}$$

where C is a small constant, and δ_{k^*} is the score for the correct key guess.

Given that $\delta_{k^*} \geqslant 2^{-m+2}$ then:

$$N = O(2^{2m}) \tag{3.7}$$

The proofs of Propositions 1 and 2 can be found in [104].

3.2.2 Countermeasures to a DCA attack

As stated before, an essential requirement for DCA is the lack of external input or output encoding. So, if external encodings are used a DCA attack can be completely avoided. However, if this is not the case, a typical countermeasure is to mask the intermediate variables such that no correlation can be observed. Following this approach, Lee et al. [81] proposed a masked white-box AES implementation which they claimed to resist a DCA attack. Starting from the white-box implementation of Chow et al., their idea is to apply a boolean masking to the lookup tables which are targeted by the DCA which are typically the $\overline{\mathsf{TMC}}$ LUTs. A random mask r is applied before encoding the output of the lookup table. So, the encoded output consists of the encoded masked key-sensitive value and the encoded mask. The mask is then removed by a XOR table. In addition to the masking scheme, the input and output encodings of some lookup tables in rounds 1 and 9 are chosen to be random 8-bit bijections instead of the concatenation of two random 4-bit bijections.

Encoded mask. For any $x \in \mathbb{F}_2^8$, $\overline{\mathsf{TMC}}^r_{i,j}(x)$ is the encoded value for $\mathsf{TMC}^r_{i,j}(x) = \mathsf{MC}_i \cdot \mathsf{SB}(x \oplus \widehat{K}^{r-1}_{i,j})$. Let us denote by y the 32-bit key-dependent value $\mathsf{MC}_i \cdot \mathsf{SB}(x \oplus \widehat{K}^{r-1}_{i,j})$. For a random mask $r = (r_0, r_1, r_2, r_3)$ where for $0 \leqslant s \leqslant 3$, r_s is uniformly drawn from $\mathbb{F}_2^8$, the masked sensitive value is given by $z = y \oplus r$. Let $g = (g_0, g_1, g_2, g_3)$ be the non-linear encoding where each g_i

is a 8-bit bijection and R the linear encoding used to encode the output of $\mathsf{TMC}^r_{i,j}$. A *masked* encoded TMC lookup table is defined by:

$$\begin{aligned}\overline{\mathsf{mTMC}}^r_{i,j}(x) &= (g \circ R, g \circ R)(z, r) \\ &= (g \circ R, g \circ R)(y \oplus r, r)\end{aligned}$$

Since the $\overline{\mathsf{TMC}}^r_{i,j}$ LUTs are followed by $\overline{\mathsf{Txor}}$ LUTs, those lookup tables are modified to remove the mask from the key-dependent value y. Recall that a $\overline{\mathsf{Txor}}$ LUT operates the XOR of two nibbles of y. If the encoding g is a 8-bit bijection which is the concatenation of two 4-bit bijections, since

$$\begin{aligned}z \oplus r &= y \Leftrightarrow \\ y_{s,t} \oplus r_{s,t} \oplus r_{s,t} &= y_{s,t} \text{ for } 0 \leqslant s \leqslant 3 \text{ and } 0 \leqslant t \leqslant 1,\end{aligned}$$

a *masked* encoded XOR table denoted $\overline{\mathsf{mTxor}}$, takes as input a nibble from the encoded masked value $g \circ R(z)$ and a nibble from the encoded mask $g \circ R(r)$.

8-bit bijection instead of two 4-bit bijections. The proposed masking technique enables to protect the lookup tables $\overline{\mathsf{TMC}}^1_{i,j}$. However, it is not sufficient since the last round lookup tables are still vulnerable as the attacker might exhaust all possible values of $\widehat{K}^9_{i,j}$ and $K^{10}_{i,j}$. More precisely, the last round T-box is defined as follows: for any $x \in \mathbb{F}_2^8$,

$$\mathsf{T}^{10}_{i,j}(x) = \mathsf{SB}(x \oplus \widehat{K}^9_{i,j}) \oplus K^{10}_{i,j}$$

Let $\overline{x}$ be the encoded input of $\overline{\mathsf{T}}^{10}_{i,j}$. Let c be one byte of a ciphertext i.e. $c = \overline{\mathsf{T}}^{10}_{i,j}(\overline{x})$ for fixed i and j. So we also have

$$c = \overline{\mathsf{T}}^{10}_{i,j}(\overline{x}) = \mathsf{SB}(x \oplus \widehat{K}^9_{i,j}) \oplus K^{10}_{i,j}, \text{ and}$$

$$x = \mathsf{SB}^{-1}(c \oplus K^{10}_{i,j}) \oplus \widehat{K}^9_{i,j}$$

Consequently, if c is not encoded, i.e. there is no external output encoding or it is known then $x = \mathsf{SB}^{-1}(c \oplus K^{10}_{i,j}) \oplus \widehat{K}^9_{i,j}$ is the target variable and the DCA attack consists in computing the correlation between $\overline{x}$ and x.

To avoid such attack, a 8-bit bijection should be used to encode x. That means that the output encodings of the XOR LUTs before the tables $\overline{\mathsf{T}}^{10}_{i,j}$ should also be 8-bit bijections. Besides, the input encodings of the lookup tables $\overline{\mathsf{Tlin}}^9_{i,j}$ are also 8-bit bijections.

In the same way, the $\overline{\mathsf{TMC}}^2_{i,j}$ tables in the second round are protected against a DCA attack which targets four bytes of the key, by using a 8-bit bijection to encode the input of $\overline{\mathsf{TMC}}^2_{i,j}$. Consequently, the output encodings of the XOR tables preceding $\mathsf{TMC}^2_{i,j}$ are also 8-bit bijections. Table 3.3 summarizes the countermeasures and their impact on the white-box size.

The white-box implementation of Lee et al. was cryptanalyzed by Rivain et al. in [104] using the variants of the DCA attack described in the following section.

Countermeasures	Size	Increase factor
First round masking	796 kB	1.56
First round masking - 8-bit bijection in round 9	2.77 MB	5.58
First round masking - 8-bit bijection in rounds 1 and 9	4.77 MB	9.59

Table 3.3: Impact of DCA countermeasures to the white-box size. Reference implementation: Chow et al.'s white-box implementation without external encodings of size 508 kB.

3.2.3 Extended Differential Computation Analysis

Regarding the result of Proposition 1, it could be tempting to choose an m-bit output encoding which verifies $n \leqslant m$ for the lookup tables $\overline{\mathsf{TMC}}^1_{i,j}$, i.e. the input size n of the tables is smaller than the encoding size. However, this countermeasure is inefficient as long as each byte after the MixColumns transformation is encoded with a small enough encoding [104].

MixColumns outputs as target variables. Let p_0, p_1, p_2, p_3 be four bytes of a plaintext and (k_0, k_1, k_2, k_3) be the corresponding first round key bytes. For $0 \leqslant i \leqslant 3$, let y_i be one byte after the MixColumns transformation defined by:

$$y_i = mc_{i,0} \otimes \mathsf{SB}(p_0 \oplus k_0) \oplus mc_{i,1} \otimes \mathsf{SB}(p_1 \oplus k_1) \oplus mc_{i,2} \otimes \mathsf{SB}(p_2 \oplus k_2) \oplus mc_{i,3} \otimes \mathsf{SB}(p_3 \oplus k_3)$$

For $i = 2$, we have

$$y_2 = \mathsf{SB}(p_0 \oplus k_0) \oplus \mathsf{SB}(p_1 \oplus k_1) \oplus \mathsf{02} \otimes \mathsf{SB}(p_2 \oplus k_2) \oplus \mathsf{03} \otimes \mathsf{SB}(p_3 \oplus k_3)$$

If p_2 and p_3 are fixed to constants, then $c = \mathsf{02} \otimes \mathsf{SB}(p_2 \oplus k_2) \oplus \mathsf{03} \otimes \mathsf{SB}(p_3 \oplus k_3)$ is a secret constant.

Let Q be a m-to-m bits encoding which encodes the byte y_2. Let X, Y be the random variables which take the values of $Q(\mathsf{SB}(p_0 \oplus k_0) \oplus \mathsf{SB}(p_1 \oplus k_1) \oplus c)$ and $Q(\mathsf{SB}(p_0 \oplus k_0) \oplus \mathsf{SB}(p_1 \oplus k_1))$, respectively. X and Y are identically distributed over the random choice of Q. Thus, for any key guess $k^* = (k_0^*, k_1^*)$ and any $p = (p_0, p_1)$, a selection function $\textsc{Sel}(p, k^*, j)$ defined as:

$$\textsc{Sel}(p, k^*, j) = y_{2,j} \text{ where } y_2 = \mathsf{SB}(p_0 \oplus k_0) \oplus \mathsf{SB}(p_1 \oplus k_1)$$

can be used to execute a DCA attack. In this case, the key search space is $(\mathbb{F}_2^8)^2$.

3.2.4 Collision correlation

So far the DCA attack exploits the correlation between a target variable and the corresponding encoded variable. When looking at the first MixColumns output, since each byte depends on four input bytes, a collision inevitably occurs. In the following, we consider one byte after the MixColumns operation in the first round as the target variable. However, the target variable might be any output of the MixColumns operation in any round. The only difference being the number of key guesses. The collision-based DCA was introduced by Rivain and Wang in [104].

Definition 19 (Set of collisions). *For any $0 \leqslant i \leqslant 3$, a set of collisions $\mathcal{S}_i$ is defined by:*

$$\mathcal{S}_i = \{(p, p') \in (\mathbb{F}_2^8)^4 \colon p \neq p', y_i = y_i'\}$$

where $y_i = mc_{i,0} \otimes \mathsf{SB}(p_0 \oplus k_0) \oplus mc_{i,1} \otimes \mathsf{SB}(p_1 \oplus k_1) \oplus mc_{i,2} \otimes \mathsf{SB}(p_2 \oplus k_2) \oplus mc_{i,3} \otimes \mathsf{SB}(p_3 \oplus k_3)$ and $y_i' = mc_{i,0} \otimes \mathsf{SB}(p_0' \oplus k_0) \oplus mc_{i,1} \otimes \mathsf{SB}(p_1' \oplus k_1) \oplus mc_{i,2} \otimes \mathsf{SB}(p_2' \oplus k_2) \oplus mc_{i,3} \otimes \mathsf{SB}(p_3' \oplus k_3)$.

Proposition 3. *For any $0 \leqslant i \leqslant 3$, let Q_i a random m-to-m bits bijection which encodes the bytes y_i. Let $\mathcal{S}_i$ be a set of collisions as defined in Definition 19. For any $(p, p') \in \mathcal{S}_i$,*

$$y_i = y_i' \Leftrightarrow Q(y_i) = Q(y_i')$$

Proposition 3 is trivial since Q is bijective.
Let $\odot$ be the binary operator defined for any $a, b \in \mathbb{F}_2^n$ as:

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

Definition 20 (Collision prediction). *Let $\mathbb{K} = (\mathbb{F}_2^8)^4$ and let $\mathbb{P} = (\mathbb{F}_2^8)^4$. For any $k \in \mathbb{K}$, a collision prediction function ϕ_k is a function from $\mathbb{P} \times \mathbb{P}$ to $\mathbb{F}_2$ defined for any $p \neq p' \in \mathbb{P}$ as:*

$$\phi_k(p, p') = f_k(p) \odot f_k(p')$$

where $f_k(p) = mc_{i,0} \otimes \mathsf{SB}(p_0 \oplus k_0) \oplus mc_{i,1} \otimes \mathsf{SB}(p_1 \oplus k_1) \oplus mc_{i,2} \otimes \mathsf{SB}(p_2 \oplus k_2) \oplus mc_{i,3} \otimes \mathsf{SB}(p_3 \oplus k_3)$.

It is easy to see that $(p, p') \in \mathcal{S}_i$ if and only if $\phi_k(p, p') = 0$.

Definition 21 (Collision computation trace). *Let p, p' be two plaintexts and let $\mathbf{v} = (v_1, \ldots, v_t)$, $\mathbf{v}' = (v_1', \ldots, v_t')$ be the corresponding computation traces. A collision computation trace $\mathbf{w} = (w_1, \ldots, w_t)$ is defined as:*

$$\mathbf{w} = \mathbf{v} \odot \mathbf{v}'$$

Remark 2. *The samples v_j for $1 \leqslant j \leqslant t$, are typically m-bit vectors instead of bits like in a classical DCA. We denote by v_{j_0} the sample which matches the target variable.*

Definition 22 (Vectorial boolean function). *Let n, m be two positive integers. A (n, m)- vectorial boolean function f is a function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ defined as:*

$$f = (f_1, \ldots, f_m)$$

where $f_1, \ldots, f_m$ are boolean functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2$.

Definition 23 (Balanced vectorial boolean function). *Let $f \colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a vectorial boolean function. f is balanced if for any $y \in \mathbb{F}_2^m$, the cardinality of the set $\{x \in \mathbb{F}_2^n \colon f(x) = y\}$ equals 2^{n-m}.*

Definition 24 (Collision distinguisher)**.** *Let $\mathbf{v}^{i_1}, \mathbf{v}^{i_2}$ be the traces obtained for a set of inputs p^{i_1}, p^{i_2} and let $\mathbf{w}^{i_1,i_2}$ be the corresponding collision computation traces. Let $W_j^{i_1,i_2}$ be the boolean function which returns the j-th component of $\mathbf{w}^{i_1,i_2}$ given the inputs p^{i_1}, p^{i_2}. The collision distinguisher for a key guess k^* is defined as:*

$$\delta_{k^*} = \max_{1 \leqslant j \leqslant t} \mathsf{Cor}(\phi^*, W_j^{i_1,i_2})$$

where $\phi^* = \phi_{k^*}$.

Proposition 4. *Let $j_0 \in [0, t]$ be the index such that $v_{j_0}^{i_1}$ and $v_{j_0}^{i_2}$ match the encoded values $\overline{f}(p^{i_1})$ and $\overline{f}(p^{i_2})$ for any inputs p^{i_1} and p^{i_2}. Let $\overline{\phi}\colon \mathbb{P} \times \mathbb{P} \to \mathbb{F}_2$ be the function defined for any input (p^{i_1}, p^{i_2}) as:*

$$\overline{\phi}(p^{i_1}, p^{i_2}) = \overline{f}(p^{i_1}) \odot \overline{f}(p^{i_2})$$

Let δ_{k^*} be the score of the key guess k^*. k^* is the right key if and only if $\delta_{k^*} = 1$.

Proof. Let $\mathcal{P} = \{(p^{i_1}, p^{i_2}) \in \mathbb{P} \times \mathbb{P}\}$.

$$\begin{aligned}
\delta_{k^*} &= \max_{1 \leqslant j_0 \leqslant t} \mathsf{Cor}(\phi^*, W_{j_0}^{i_1,i_2}) \\
&= \mathsf{Cor}(\phi^*, W_{j_0}^{i_1,i_2}) \\
&= \mathsf{Cor}(\phi^*, \overline{\phi}) \\
&= \frac{1}{|\mathcal{P}|} \sum_{(p^{i_1}, p^{i_2}) \in \mathcal{P}} (-1)^{\phi^*(p^{i_1}, p^{i_2}) \oplus \overline{\phi}(p^{i_1}, p^{i_2})}
\end{aligned}$$

Assume that k^* is the right key. Then, if (p^{i_1}, p^{i_2}) leads to a collision on f_{k^*}'s output, i.e. $f_{k^*}(p^{i_1}) = f_{k^*}(p^{i_2})$, then the pair also leads to a collision on the encoded output, i.e. $\overline{f}(p^{i_1}) = \overline{f}(p^{i_2})$. Thus, $\phi^*(p^{i_1}, p^{i_2}) = 0$ and $\overline{\phi}(p^{i_1}, p^{i_2}) = 0$. So, $(-1)^{\phi^*(p^{i_1}, p^{i_2}) \oplus \overline{\phi}(p^{i_1}, p^{i_2})} = 1$.

If no collusion occurs on f_{k^*}'s output given (p^{i_1}, p^{i_2}), then since the encoding is bijective, no collusion occurs on the encoded output. Thus, $\phi^*(p^{i_1}, p^{i_2}) = 1$ and $\overline{\phi}(p^{i_1}, p^{i_2}) = 1$. So, $(-1)^{\phi^*(p^{i_1}, p^{i_2}) \oplus \overline{\phi}(p^{i_1}, p^{i_2})} = 1$. Hence, $\delta_{k^*} = 1$.

Now, assume that $\delta_{k^*} = 1$. If k^* is not the right key then there exits a pair (p^{i_1}, p^{i_2}) such that $f_{k^*}(p^{i_1}) = f_{k^*}(p^{i_2})$ and $\overline{f}(p^{i_1}) \neq \overline{f}(p^{i_2})$. Thus, $\phi^*(p^{i_1}, p^{i_2}) = 0$ and $\overline{\phi}(p^{i_1}, p^{i_2}) = 1$. So, $(-1)^{\phi^*(p^{i_1}, p^{i_2}) \oplus \overline{\phi}(p^{i_1}, p^{i_2})} = -1$.

Hence, $\delta_{k^*} < 1$ which is a contradiction. □

The following definition expresses the probability of success of the attack, assuming a sample that matches an encoded value. The probability over all samples is given is Corollary 1.

Definition 25 (Probability of success)**.** *Let k^* denotes the correct key guess and $k^\times$ denotes a wrong key guess. The probability of success of a collision-based DCA is the probability that the value of the collision distinguisher for k^* is greater than the values of the distinguisher for all wrong key guesses.*

$$p = \Pr\left[\mathsf{Cor}(\phi^*, \overline{\phi}) > \max_{k^\times} \mathsf{Cor}(\phi^\times, \overline{\phi})\right] \tag{3.8}$$

Proposition 5. *Let N be the number of recorded traces. Assuming that for all keys $k \in \mathbb{K}$, the functions $(f_k)_k$ are mutually independent balanced vectorial boolean functions, the probability of success p of a collision-based DCA is given by:*

$$p \geqslant 1 - |\mathbb{K}| \exp\left(-\frac{(N-1)(N-2)}{2^{m+1}}\right) \tag{3.9}$$

Corollary 1. *Let N be the number of recorded traces such that each trace has t samples. The probability of full success p_{full}, i.e. the probability of success over all samples, is given by:*

$$\begin{aligned} p_{full} &= \Pr\left[\max_{1 \leqslant j \leqslant t} \mathsf{Cor}(\phi^*, W_j^{i_1,i_2}) > \max_{1 \leqslant j \leqslant t, k^\times} \mathsf{Cor}(\phi^\times, W_j^{i_1,i_2})\right] && (3.10) \\ &\geqslant 1 - t|\mathbb{K}| \exp\left(-\frac{(N-1)(N-2)}{2^{m+1}}\right) && (3.11) \end{aligned}$$

Proposition 6 (Trace complexity)**.** *Let λ be a parameter. If $N = \sqrt{2^{m+1}(\lambda \ln 10 + \ln t + \ln \mathbb{K})} + 1$ then $p_{full} \geqslant 1 - 10^{-\lambda}$. Thus, the trace complexity is:*

$$N = \Theta\left(2^{m/2}\right) \tag{3.12}$$

The proofs of Proposition 5, Corollary 1 and Proposition 6 are given in [104].

The trace complexity for a collision-based DCA is much smaller than for the classical DCA. Furthermore, the key search space $\mathbb{K} = (\mathbb{F}_2^8)^4$ can be reduced to $(\mathbb{F}_2^8)^2$ by fixing two bytes of the input p to constants.

3.3 Differential Fault Analysis

A Differential Fault Analysis is an attack originally applied to tamper-resistant device to reveal embedded secret keys. Consequently, it belongs to the category of gray box attacks. It generally consists in altering the encryption process by modifying one or several intermediate variables and exploiting the difference between a (fault-free) ciphertext and a faulty ciphertext. It appeared that DFA is very efficient on white-box implementations because an attacker can easily modify intermediate variables from the available stored data and observe computational errors.

Several papers in the literature dealt with fault analysis of the AES. See for instance [10, 17, 60]. We describe here the approach of Dusart, Letourneux and Vivolo [50] to attack the AES-128. In their fault model, they consider one fault affecting a byte between the 8-th round and the 9-th round. Specifically, it is assumed that the fault occurs between the last two MixColumns (there is no MixColumns in the last round). Then, they use the particular form of the MixColumns transformation and the S-box to write and solve a system of equations.

Let $state = \begin{pmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{pmatrix}$ be the state after ShiftRows in the 9-th round and let

$\epsilon \in \mathbb{F}_{2^8} \setminus \{0\}$ be a fault introduced on the first byte of the state. The erroneous state $\overline{state}$ is defined by:

$$\overline{state} = \begin{pmatrix} s_0 \oplus \epsilon & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{pmatrix}$$

The propagation of the error through the different transformations is as follows, were we denote by $\star$ the state bytes that are not affected by the error:

After MixColumns in round 9:

$$state \leftarrow \begin{pmatrix} 02 \otimes s_0 \oplus 03 \otimes s_1 \oplus s_2 \oplus s_3 & \star & \star & \star \\ s_0 \oplus 02 \otimes s_1 \oplus 03 \otimes s_2 \oplus s_3 & \star & \star & \star \\ s_0 \oplus s_1 \oplus 02 \otimes s_2 \oplus 03 \otimes s_3 & \star & \star & \star \\ 03 \otimes s_0 \oplus s_1 \oplus s_2 \oplus 02 \otimes s_3 & \star & \star & \star \end{pmatrix}$$

$$\overline{state} \leftarrow \begin{pmatrix} 02 \otimes (s_0 \oplus \epsilon) \oplus 03 \otimes s_1 \oplus s_2 \oplus s_3 & \star & \star & \star \\ s_0 \oplus \epsilon \oplus 02 \otimes s_1 \oplus 03 \otimes s_2 \oplus s_3 & \star & \star & \star \\ s_0 \oplus \epsilon \oplus s_1 \oplus 02 \otimes s_2 \oplus 03 \otimes s_3 & \star & \star & \star \\ 03 \otimes (s_0 \oplus \epsilon) \oplus s_1 \oplus s_2 \oplus 02 \otimes s_3 & \star & \star & \star \end{pmatrix}$$

After AddRoundKey in round 9:

$$state \leftarrow \begin{pmatrix} 02 \otimes s_0 \oplus 03 \otimes s_1 \oplus s_2 \oplus s_3 \oplus K_0^9 & \star & \star & \star \\ s_0 \oplus 02 \otimes s_1 \oplus 03 \otimes s_2 \oplus s_3 \oplus K_1^9 & \star & \star & \star \\ s_0 \oplus s_1 \oplus 02 \otimes s_2 \oplus 03 \otimes s_3 \oplus K_2^9 & \star & \star & \star \\ s_0 \oplus \epsilon \oplus s_1 \oplus 02 \otimes s_2 \oplus 03 \otimes s_3 \oplus K_3^9 & \star & \star & \star \end{pmatrix}$$

$$\overline{state} \leftarrow \begin{pmatrix} \mathsf{02} \otimes (s_0 \oplus \epsilon) \oplus \mathsf{03} \otimes s_1 \oplus s_2 \oplus s_3 \oplus K_0^9 & \star & \star & \star \\ s_0 \oplus \epsilon \oplus \mathsf{02} \otimes s_1 \oplus \mathsf{03} \otimes s_2 \oplus s_3 \oplus K_1^9 & \star & \star & \star \\ s_0 \oplus \epsilon \oplus s_1 \oplus \mathsf{02} \otimes s_2 \oplus \mathsf{03} \otimes s_3 \oplus K_2^9 & \star & \star & \star \\ \mathsf{03} \otimes (s_0 \oplus \epsilon) \oplus s_1 \oplus s_2 \oplus \mathsf{02} \otimes s_3 \oplus K_3^9 & \star & \star & \star \end{pmatrix}$$

After SubBytes in round 10**:**

$$state \leftarrow \begin{pmatrix} \mathsf{SB}(\mathsf{02} \otimes s_0 \oplus \mathsf{03} \otimes s_1 \oplus s_2 \oplus s_3 \oplus K_0^9) & \star & \star & \star \\ \mathsf{SB}(s_0 \oplus \mathsf{02} \otimes s_1 \oplus \mathsf{03} \otimes s_2 \oplus s_3 \oplus K_1^9) & \star & \star & \star \\ \mathsf{SB}(s_0 \oplus s_1 \oplus \mathsf{02} \otimes s_2 \oplus \mathsf{03} \otimes s_3 \oplus K_2^9) & \star & \star & \star \\ \mathsf{SB}(s_0 \oplus \epsilon \oplus s_1 \oplus \mathsf{02} \otimes s_2 \oplus \mathsf{03} \otimes s_3 \oplus K_3^9) & \star & \star & \star \end{pmatrix}$$

$$\overline{state} \leftarrow \begin{pmatrix} \mathsf{SB}(\mathsf{02} \otimes (s_0 \oplus \epsilon) \oplus \mathsf{03} \otimes s_1 \oplus s_2 \oplus s_3 \oplus K_0^9) & \star & \star & \star \\ \mathsf{SB}(s_0 \oplus \epsilon \oplus \mathsf{02} \otimes s_1 \oplus \mathsf{03} \otimes s_2 \oplus s_3 \oplus K_1^9) & \star & \star & \star \\ \mathsf{SB}(s_0 \oplus \epsilon \oplus s_1 \oplus \mathsf{02} \otimes s_2 \oplus \mathsf{03} \otimes s_3 \oplus K_2^9) & \star & \star & \star \\ \mathsf{SB}(\mathsf{03} \otimes (s_0 \oplus \epsilon) \oplus s_1 \oplus s_2 \oplus \mathsf{02} \otimes s_3 \oplus K_3^9) & \star & \star & \star \end{pmatrix}$$

After ShiftRows in round 10**:**

$$state \leftarrow \begin{pmatrix} \mathsf{SB}(x_0) & \star & \star & \star \\ \star & \star & \star & \mathsf{SB}(x_{13}) \\ \star & \star & \mathsf{SB}(x_{10}) & \star \\ \star & \mathsf{SB}(x_7) & \star & \star \end{pmatrix}$$

$$\text{with } \begin{cases} x_0 = \mathsf{02} \otimes s_0 \oplus \mathsf{03} \otimes s_1 \oplus s_2 \oplus s_3 \oplus K_0^9 \\ x_{13} = s_0 \oplus \mathsf{02} \otimes s_1 \oplus \mathsf{03} \otimes s_2 \oplus s_3 \oplus K_1^9 \\ x_{10} = s_0 \oplus s_1 \oplus \mathsf{02} \otimes s_2 \oplus \mathsf{03} \otimes s_3 \oplus K_2^9 \\ x_7 = s_0 \oplus \epsilon \oplus s_1 \oplus \mathsf{02} \otimes s_2 \oplus \mathsf{03} \otimes s_3 \oplus K_3^9 \end{cases}$$

$$\overline{state} \leftarrow \begin{pmatrix} \mathsf{SB}(\overline{x_0}) & \star & \star & \star \\ \star & \star & \star & \mathsf{SB}(\overline{x_{13}}) \\ \star & \star & \mathsf{SB}(\overline{x_{10}}) & \star \\ \star & \mathsf{SB}(\overline{x_7}) & \star & \star \end{pmatrix}$$

$$\text{with} \begin{cases} \overline{x_0} = \mathsf{02} \otimes (s_0 \oplus \epsilon) \oplus \mathsf{03} \otimes s_1 \oplus s_2 \oplus s_3 \oplus K_0^9 \\ \overline{x_{13}} = s_0 \oplus \epsilon \oplus \mathsf{02} \otimes s_1 \oplus \mathsf{03} \otimes s_2 \oplus s_3 \oplus K_1^9 \\ \overline{x_{10}} = s_0 \oplus \epsilon \oplus s_1 \oplus \mathsf{02} \otimes s_2 \oplus \mathsf{03} \otimes s_3 \oplus K_2^9 \\ \overline{x_7} = \mathsf{03} \otimes (s_0 \oplus \epsilon) \oplus s_1 \oplus s_2 \oplus \mathsf{02} \otimes s_3 \oplus K_3^9 \end{cases}$$

After AddRoundKey in round 10**:**

$$state \leftarrow \begin{pmatrix} \mathsf{SB}(x_0) \oplus K_0^{10} & \star & \star & \star \\ \star & \star & \star & \mathsf{SB}(x_{13}) \oplus K_{13}^{10} \\ \star & \star & \mathsf{SB}(x_{10}) \oplus K_{10}^{10} & \star \\ \star & \mathsf{SB}(x_7) \oplus K_7^{10} & \star & \star \end{pmatrix}$$

$$\overline{state} \leftarrow \begin{pmatrix} \mathsf{SB}(\overline{x_0}) \oplus K_0^{10} & \star & \star & \star \\ \star & \star & \star & \mathsf{SB}(\overline{x_{13}}) \oplus K_{13}^{10} \\ \star & \star & \mathsf{SB}(\overline{x_{10}}) \oplus K_{10}^{10} & \star \\ \star & \mathsf{SB}(\overline{x_7}) \oplus K_7^{10} & \star & \star \end{pmatrix}$$

Let C and $\overline{C}$ be the correct ciphertext and the faulty ciphertext respectively. The bytes $K_0^{10}, K_7^{10}, K_{10}^{10}$ and K_{13}^{10} of the last round key can be determined by computing the differences $C_0 \oplus \overline{C_0}$, $C_7 \oplus \overline{C_7}$, $C_{10} \oplus \overline{C_{10}}$ and $C_{13} \oplus \overline{C_{13}}$ as follows:

$$\begin{cases} C_0 \oplus \overline{C_0} &= \mathsf{SB}(x_0) \oplus \mathsf{SB}(x_0 \oplus \mathsf{02} \otimes \epsilon) \\ C_7 \oplus \overline{C_7} &= \mathsf{SB}(x_7) \oplus \mathsf{SB}(x_7 \oplus \mathsf{03} \otimes \epsilon) \\ C_{10} \oplus \overline{C_{10}} &= \mathsf{SB}(x_{10}) \oplus \mathsf{SB}(x_{10} \oplus \epsilon) \\ C_{13} \oplus \overline{C_{13}} &= \mathsf{SB}(x_{13}) \oplus \mathsf{SB}(x_{13} \oplus \epsilon) \end{cases}$$

The candidates for the value of ϵ are determined by solving the system above. It has been proved that the number of solutions of the system is at most 63 assuming that two of $\mathsf{02}^{-1}(C_0 \oplus \overline{C_0})$, $C_7 \oplus \overline{C_7}$, $\mathsf{03}^{-1}(C_{10} \oplus \overline{C_{10}})$ and $C_{13} \oplus \overline{C_{13}}$ are not equal (see [50] for the details). For each solution, there is a corresponding set of values for x_0, x_7, x_{10}, x_{13}. Besides, the following system is used to determine the candidates for the key bytes:

$$\begin{cases} C_0 &= \mathsf{SB}(x_0) \oplus K_0^{10} \\ C_7 &= \mathsf{SB}(x_7) \oplus K_7^{10} \\ C_{10} &= \mathsf{SB}(x_{10}) \oplus K_{10}^{10} \\ C_{13} &= \mathsf{SB}(x_{13}) \oplus K_{13}^{10} \end{cases} \Longrightarrow \begin{cases} K_0^{10} &= C_0 \oplus \mathsf{SB}(x_0) \\ K_7^{10} &= C_7 \oplus \mathsf{SB}(x_7) \\ K_{10}^{10} &= C_{10} \oplus \mathsf{SB}(x_{10}) \\ K_{13}^{10} &= C_{13} \oplus \mathsf{SB}(x_{13}) \end{cases}$$

Dusart et al. [50] estimated the number of faulty ciphertexts required to find four bytes of the round key to be 9. Besides, all the key bytes can be determined using 9 faulty ciphertexts by treating the errors simultaneously.

DFA attack on a white-box implementation of the AES. Given a white-box implementation of the AES, a Differential Fault Analysis consists in the following steps:

1. First, the attacker executes the algorithm on a random input and identifies the last two MixColumns of the encryption.
2. Then, the attacker identifies a variable that encodes one byte of the intermediate state between those two MixColumns and records the value of the ciphertext when executing the algorithm on a random input.
3. The attacker modifies the identified encoded variable and records the faulty ciphertext.
4. The attacker computes the difference between the two ciphertexts. If the difference is non-zero at exactly four bytes, then the attacker has a correct fault model.
5. The attacker repeats the steps 2 to 4 with another encoded variable.

Two faults on one encoded variable is generally enough to recover the last round key. If the attacker is able to clearly identify the target encoded variables, i.e. variables that encode the state bytes between the last two MixColumns, then the attacker always gets the correct fault model.

Remark 3. *As we consider a white-box model, we assume that the adversary is able to determine exactly where to inject the fault. If this is not the case, the attacker is still able to randomly inject faults. If so, the behavior of the white-box program gives information about the fault accuracy. Thus, we distinguish the following scenarios:*

- *The fault injection leads to the crash of the program or the output is unexpected: the result is not exploitable.*
- *The fault injection does not affect the output of the program: the fault has affected a useless part of the implementation.*

- *The fault injection affects all bytes of the output: the fault has affected the state but too early.*

- *The fault injection affects one byte of the output: the fault has affected the state but too late in the encryption, i.e. after the last* MixColumns*.*

- *The fault injection affects exactly four bytes of the output: the fault has affected the state between the last two* MixColumns*.*

3.3.1 Countermeasures to DFA attack

The countermeasures to DFA are typically based on space or time redundancies, e.g. using error detection or correction techniques and repeated computations. We discuss here typical countermeasures in the AES case.

Space (information) redundancy

Information redundancy techniques are based on error detecting codes. They used some check bits that are generated from the plaintext then propagated along the state and finally checked on the ciphertext. We distinguish here three types of error detecting codes: a parity code, a robust code and invariance-based code.

Parity-1 **[68].** One bit of parity is used to check the 128-bit state. The parity bit is added to the input then the parity bit of the output is checked taking into account the parity modification induced by each transformation of a round. For any input X, let $P(X)$ be the parity of X.

- A parity bit is added to the output of the S-box and corresponds to the XOR of the parity of the S-box input X and the parity of the S-box output $Y = \mathsf{SB}(X)$.

- The parity of a round key K is denoted $P(K)$.

- If Z is the output of one round then the parity of Z is defined by $P(Z) = P(X) \oplus \sum_{i=0}^{15}(P(X_i) \oplus P(Y_i)) \oplus P(K)$. The linear transformations MixColumns and ShiftRows don't change the parity of the state, so they are ignored in the calculation of the state parity.

Parity-16 **[8].** One bit of parity is used to check one byte of the state, so 16 bits are used for each state. The parity of each byte of the state is predicted after each transformation, so the parity of the ciphertext can be checked according to this prediction.

- As for the Parity-1, a parity bit is added to the output of the S-box and equals $P(X) \oplus P(Y)$ for $Y = \mathsf{SB}(X)$.

- The calculation of the parity bits after the MixColumns is more tricky. The parities of the bytes of the j-th column of the state X are defined by:

$$
\begin{aligned}
P(X_{0,j}) &= P(X_{0,j}) \oplus P(X_{2,j}) \oplus P(X_{3,j}) \oplus x_7^{0,j} \oplus x_7^{1,j} \\
P(X_{1,j}) &= P(X_{0,j}) \oplus P(X_{1,j}) \oplus P(X_{3,j}) \oplus x_7^{1,j} \oplus x_7^{2,j} \\
P(X_{2,j}) &= P(X_{0,j}) \oplus P(X_{1,j}) \oplus P(X_{2,j}) \oplus x_7^{2,j} \oplus x_7^{3,j} \\
P(X_{3,j}) &= P(X_{1,j}) \oplus P(X_{2,j}) \oplus P(X_{3,j}) \oplus x_7^{3,j} \oplus x_7^{0,j}
\end{aligned}
$$

where $x_7^{i,j}$ denotes the 7-th bit of the byte $X_{i,j}$.

Robust code [77]. A non-linear code is used to check the non-linear part of one round, i.e. the inversion in $\mathbb{F}_{2^8}$ while the linear part composed of the other transformations is checked with an 8-bit redundancy. More specifically:

- To check an error on the output Y of the non-linear part, the inverse of Y in $\mathbb{F}_{2^8}$ is computed then the last two bits of the result of $Y \otimes Y^{-1}$ is used as a redundant part: $Y \otimes Y^{-1} = \mathsf{0x01}$ if $Y \neq 0$ and $Y \otimes Y^{-1} = \mathsf{0x00}$ if $Y = 0$.
- For any column j $(X_{0,j}, X_{1,j}, X_{2,j}, X_{3,j})$ of the state matrix after the linear transformations, the 8-bit redundancy is given by $X_{0,j} \oplus X_{1,j} \oplus X_{2,j} \oplus X_{3,j}$.

Invariance-based code [63]. An invariant property of an AES round is used to compute two equivalent round functions so that the outputs can be checked. Let ARK, MC, SR, SB denote the transformations of one round of the AES and let K^r be the r-th round key. The r-th state X^r is defined as follows:

$$X^r = \mathsf{AES}(K^r, X^{r-1}) = \mathsf{ARK}(K^r, \mathsf{MC}(\mathsf{SR}(\mathsf{SB}(X^{r-1})))).$$

There exits a permutation σ such that for any round r:

$$\mathsf{AES}(K^r, X^{r-1}) = \sigma^{-1}(\mathsf{ARK}(\sigma(K^r), \mathsf{MC}(\mathsf{SR}(\mathsf{SB}(\sigma(X^{r-1})))))).$$

For a state matrix $X = \begin{pmatrix} X_{0,0} & X_{0,1} & X_{0,2} & X_{0,3} \\ X_{1,0} & X_{1,1} & X_{1,2} & X_{1,3} \\ X_{2,0} & X_{2,1} & X_{2,2} & X_{2,3} \\ X_{3,0} & X_{3,1} & X_{3,2} & X_{3,3} \end{pmatrix}$ an example of such permutation σ is defined as:

$$\sigma(X) = \begin{pmatrix} X_{0,3} & X_{0,1} & X_{0,2} & X_{0,0} \\ X_{1,3} & X_{1,1} & X_{1,2} & X_{1,0} \\ X_{2,3} & X_{2,1} & X_{2,2} & X_{2,0} \\ X_{3,3} & X_{3,1} & X_{3,2} & X_{3,0} \end{pmatrix}$$

A round output is checked by comparing $\mathsf{AES}(K^r, X^{r-1})$ and $\sigma^{-1}(\mathsf{AES}(\sigma(K^r), \sigma(X^{r-1})))$.

Time redundancy

When using time redundancy, a function is executed twice on the same input and then the results are compared.

DFA countermeasures for a white-box implementation

Time redundancy can be implemented to detect a fault. However, the comparison of the outputs of the different executions should be hidden to prevent the attacker from removing the countermeasure. Besides, if the attacker is able to see that several executions of the same algorithm is executed, they can focus on the output of only one execution. We have successfully applied the DFA to some of the candidates of the WhibOx 2019 contest by identifying the time redundancy.

Space redundancy can also be implemented in a table-based white-box implementation. First note that the *Parity*-1 technique does not detect even numbers of faults. Moreover, it is easy to guess the value of the parity bit, since there are only 2 possible values. *Parity*-16 is a bit better but the 256 possible values of the redundancy are still practical to exhaust. The *robust code* can be implemented by adding the two bits of redundancy to any output of the lookup tables calculating the inversion in $\mathbb{F}_{2^8}$. Yet, it is possible to exhaust all 4 possible values. The redundancy of the linear part is more difficult to guess since 32 bits of redundancy is used for any state. However, since 32 bits are needed to compute the 8 bits of redundancy, this will lead to an increase of the white-box size. Finally, the *invariance-based code* is not practical to implement since it consists in computing a round function in a different way, so doubling the number of lookup tables, then the 128-bit outputs of both round function need to be compared.

3.4 White-box implementation from circuit obfuscation

The approach of Biryukov and Udovenko is to use a bit-sliced implementation of the AES to integrate masking techniques. *Bitslicing* is an implementation strategy that consists in the expression of functions in terms of single bit logical operations AND, XOR, OR, NOT. In other words, a function is expressed with boolean functions which are represented with boolean circuits. Bitslicing enables fast and constant-time implementations of cryptographic algorithms which resist cache and timing attacks.

A bitslice implementation of the AES which uses the compact S-box of [30] can be found in [103].

3.4.1 Differential Computation Analysis adapted to circuit implementation

In this section, we consider a circuit-based white-box implementation, i.e. an implementation in the form of a boolean circuit. We start with the formal definition of a boolean circuit.

Definition 26 (Boolean circuit)**.** *Let n, m two integers. An n-input m-output boolean circuit C is a directed acyclic graph along with labels associated with its nodes such that:*

- *The nodes of C are partitioned into three disjoint sets: the set of input nodes, the set of constant nodes and the set of gate nodes.*

- *The set of input nodes has exactly n members labeled $X_1, \ldots, X_n$. The in-degree*[10] *of an input node is* 0.
- *Each constant node has in-degree* 0 *and is labeled* 0 *or* 1.
- *Each gate node of C is labeled by an element of the set $\{\wedge, \vee, \neg\}$. An* AND *gate is labeled $\wedge$ and has in-degree* 2*, an* OR *gate is label $\vee$ and has in-degree* 2 *and a* NOT *gate is label $\neg$ and has in-degree* 1.
- *m nodes of C are identified as output nodes of C and labeled $Y_1, \ldots, Y_m$.*

The size of C is the number of nodes and denoted $|C|$.

A boolean circuit C having n inputs and m outputs computes a function of the form:

$$\begin{array}{rcl} f\colon \{0,1\}^n & \to & \{0,1\}^m \\ (x_1, \ldots, x_n) & \mapsto & (y_1, \ldots, y_m) \end{array}$$

such that $x_1, \ldots, x_n$ are the values assigned to the input nodes and $y_1, \ldots, y_m$ are the values assigned to the output nodes.

Masking scheme.

Definition 27 (XOR masking)**.** *For any integer n, let $s \in \mathbb{F}_2^n$ be a secret variable. Let $s_1, \ldots, s_{d-1}$ be $d-1$ random variables of $\mathbb{F}_2^n$. A masked variable $s_d \in \mathbb{F}_2^n$ is defined as:*

$$s_d = s \oplus s_1 \oplus \ldots \oplus s_{d-1}.$$

$s_1, \ldots, s_d$ are called the shares of the scheme. The masking is said to be of order $d-1$ if the knowledge of $d-1$ shares or less does not enable to compute the value of s.

For any function f which operates on the secret variable s, there exits d shares $v_1, \ldots, v_d$ such that:

$$f(s) = v_1 \oplus \ldots \oplus v_d.$$

The shares $v_1, \ldots, v_d$ are computed from $s_1, \ldots, s_d$. Especially, if f is a linear function then:

$$f(s) = f(s_1 \oplus \ldots \oplus s_d) = f(s_1) \oplus \ldots \oplus f(s_d).$$

A masking scheme allows for the sound computation of a function (or evaluation of a circuit) on a masked variable. In the following, we denote by $\Diamond$ a boolean binary operator and by $C_\Diamond$ the circuit implementing the operation. The evaluation function of $C_\Diamond$ is denoted by $\mathsf{Eval}_\Diamond$.

Definition 28 (Masking scheme)**.** *A t-bit masking scheme consists of:*

- *An encoding function $e\colon \mathbb{F}_2 \times \mathbb{F}_2^r \to \mathbb{F}_2^t$*

[10] The number of egdes going into a node.

- *A decoding function* $d\colon \mathbb{F}_2^t \to \mathbb{F}_2$
- *A set of triplets* $\{(\Diamond, \mathsf{Eval}_\Diamond, C_\Diamond)\}$

For any random value $\alpha \in \mathbb{F}_2^r$*, and any* $x \in \mathbb{F}_2$*, it holds that:*

$$d(e(x, \alpha)) = x.$$

Besides, for any triplet $(\Diamond, \mathsf{Eval}_\Diamond, C_\Diamond)$*, the decoding function satisfies:*

$$d(\mathsf{Eval}_\Diamond(x_1, x_2, \alpha)) = d(x_1) \Diamond d(x_2),$$

where $x_1, x_2 \in \mathbb{F}_2$ *are the inputs of the circuit* $C_\Diamond$ *and* α *is some* r*-bit randomness. The second condition illustrates the fact that each circuit is evaluated on encoded inputs and that the decoding function enables to decode the output of the circuit for any operator* $\Diamond$*.*

The degree of a masking scheme is the algebraic degree of the decoding function. For instance, the XOR masking scheme is of degree 1, since the decoding function is the XOR which is linear.

Differential Computation Analysis. As for table-based white-box implementations, the DCA attack on a circuit-based white-box implementation consists in computing the correlation between a predictable value, i.e. a value which depends on a few bits of the key, and the values computed in some nodes. Algorithm 8.1 from [113] describes the steps of a DCA attack on a circuit-based white-box implementation.

The main challenge for a successful DCA on a masked circuit implementation is to locate the shares. This can be done in two ways:

1. With a combinatorial analysis.
2. With linear algebra.

In the first case, we talk about a *combinatorial DCA* and such an attack uses either a correlation coefficient as for a classical DCA or a time-memory trade-off. The idea is to guess the location of the shares and then check if they match the predictable value. The complexity of the correlation attack exponentially increases with the number of guessed shares. This is due to the iteration over all u-bit vectors (each bit corresponds to a share) and the computation of the correlation coefficient. The complexity might be lowered by using a table to store a tuple of vectors corresponding to the larger half of shares. Then, the tuple of vectors corresponding to the smaller half of shares is computed and one checks if the sum of the vectors with the predictable value is found in the table (see Table 3.4).

In the second case, we talk about a *Linear Algebraic DCA* since the locations of the shares are determined by the solution of a system of linear equations which involve the predictable value and the vector of some nodes values. The principle is to construct a matrix M that has as columns the vectors (in the case of the classic boolean masking) (or product of vectors in the case of a non-linear masking) and then solve the equation $M \times x = \tilde{v}$ where $\tilde{v}$ is the predictable vector.

Complexities. Table 3.4.1 gives the time complexities of each attack.

v_1	$v_1 \oplus \tilde{v}$
v_2	$v_2 \oplus \tilde{v}$
$\vdots$	$\vdots$
v_i	$v_i \oplus \tilde{v}$
$\vdots$	$\vdots$
v_j	$v_j \oplus \tilde{v}$
v_n	$v_n \oplus \tilde{v}$

Table 3.4: Time-memory trade off attack on a XOR masking scheme with $s = 2$ shares: a predictable vector $\tilde{v}$ is masked with two shares v_i, v_j, i.e. $v_i = \tilde{v} \oplus v_j$.

	Attack	**Time complexity**
Combinatorial	Correlation	$\mathcal{O}\left(\lvert C\rvert^u k 2^{2u}\right)$
	Time-Memory Trade-Off	$\mathcal{O}\left(\lvert C\rvert^{\lceil s/2\rceil} + \lvert C\rvert^{\lfloor s/2\rfloor} k\right)$
Linear algebra	Linear	$\mathcal{O}\left(\lvert C\rvert^\omega + \lvert C\rvert^2 k\right)$
	Linearization	$\mathcal{O}\left(\delta^\omega + \delta^2 k\right)$

Table 3.5: Time complexities of DCA attacks on a masked circuit: $|C|$ is the size of the circuit considered for the attack, k is the number of key candidates, u is the number of guessed shares locations, ω is the matrix multiplication exponent, d is the degree of the masking scheme, δ is the number of monomials of n-bit variables of degree at most d and s is the number of shares of the masking scheme [113].

3.4.2 Countermeasures

If the number of shares used in the masking schemes is small then the combinatorial attack would be practical. Consequently, one way to make the white-box implementation secure is to increase the number of shares while using a classic boolean masking. However, in this case, an algebraic attack is feasible and the complexity of the attack depends on the degree of the decoding function. Though, a masking scheme with a high degree decoder but small number of shares prevent the generalized algebraic attack but still not the combinatorial attack. Thus, a possible countermeasure to prevent both attacks is the use a high enough number of shares and a high degree decoder. Biryukov and Udovenko suggested to apply two type of masking schemes to the reference circuit: a non-linear masking scheme which prevents an algebraic attack and a linear masking scheme to prevent a correlation attack.

Biryukov and Udovenko described a minimalist quadratic masking scheme: the decoding function has a degree 2 and each bit is encoded with 3 bits. The quadratic masking scheme has been applied to the AES-128 reference circuit [40]. While effectively providing security against DCA correlation and algebraic attacks, the white-box implementation does not integrate any protection against a fault attack.

The quadratic masking scheme consists of the following functions:

- An encoding function $e\colon \mathbb{F}_2 \times \mathbb{F}_2^2 \to \mathbb{F}_2^3$ such that for any $x \in \mathbb{F}_2$ and any random bits r_a, r_b:

$$e(x, r_a, r_b) = (r_a, r_b, x \oplus r_a r_b).$$

- A decoding function $d\colon \mathbb{F}_2^3 \to \mathbb{F}_2$ such that for any $(a, b, c) \in \mathbb{F}_2^3$:

$$d(a, b, c) = ab \oplus c.$$

- A refresh function $\mathsf{Refresh}\colon \mathbb{F}_2^3 \times \mathbb{F}_2^3 \to \mathbb{F}_2^3$ such that for any $(a, b, c) \in \mathbb{F}_2^3$ and any random bits (r_a, r_b, r_c), $\mathsf{Refresh}$ returns (a, b, c) encoded with fresh randomness applying the following steps:

  1. Compute m_a, m_b such that $m_a = r_a(b \oplus r_c)$ and $m_b = r_b(a \oplus r_c)$.
  2. Update r_c as $r_c \leftarrow m_a \oplus m_b \oplus (r_a \oplus r_c)(r_b \oplus r_c) \oplus r_c$.
  3. Update (a, b, c) as: $a \leftarrow a \oplus r_a$, $b \leftarrow r \oplus r_b$ and $c \leftarrow c \oplus r_c$.

- An evaluation function $\mathsf{Eval}_{\mathsf{XOR}}\colon \mathbb{F}_2^3 \times \mathbb{F}_2^3 \times (\mathbb{F}_2^3)^2 \to \mathbb{F}_2^3$ for the XOR operation. For any inputs (a, b, c), (d, e, f) and any random bits $(r_a, r_b, r_c, r_d, r_e, r_f)$, $\mathsf{Eval}_{\mathsf{XOR}}$ applies the following operations:

  1. Refresh (a, b, c) and (d, e, f), i.e. (a, b, c) and (d, e, f) are encoded with fresh randomness.
  2. Compute (x, y, z) such that $x = a \oplus d$, $y = b \oplus e$ and $z = c \oplus f \oplus ae \oplus bd$.
  3. $\mathsf{Eval}_{\mathsf{XOR}}((a, b, c), (d, e, f), (r_a, r_b, r_c, r_d, r_e, r_f)) = (x, y, z)$.

- An evaluation function $\mathsf{Eval}_{\mathsf{AND}}\colon \mathbb{F}_2^3 \times \mathbb{F}_2^3 \times (\mathbb{F}_2^3)^2 \to \mathbb{F}_2^3$ for the AND operation. For any inputs (a, b, c), (d, e, f) and any random bits $(r_a, r_b, r_c, r_d, r_e, r_f)$, $\mathsf{Eval}_{\mathsf{AND}}$ applies the following operations:

  1. Refresh (a, b, c) and (d, e, f).

2. Compute m_a, m_b such that $m_a = bf \oplus r_c e$ and $m_b = ce \oplus r_f b$.
3. Compute (x, y, z) such that $x = ae \oplus rf$, $y = bd \oplus r_c$ and $z = am_a \oplus dmd \oplus r_c r_f \oplus cf$.
4. $\mathsf{Eval}_{\mathsf{AND}}((a, b, c), (d, e, f), (r_a, r_b, r_c, r_d, r_e, r_f)) = (x, y, z)$.

Conclusion of the chapter

In this chapter we have outlined the various advances in the use of White-Box Cryptography in practice. While the Chapter 2 of this thesis focused on the security of a white-box implementation regarding structural attacks like the BGE attack of Section 2.1.4 or the collision attack of Section 2.1.5, this chapter showed that designers should also be careful of side-channel attacks which are more successful in the white-box model because of the ability of the attacker to instrument the binary. Thus, a white-box implementation should integrate side-channel countermeasures like integrity check or masking scheme as conventional cryptographic softwares. Yet, this has a cost and the use of such countermeasures irremediably leads to an increase of the white-box size and a decrease of the performance. The main challenge becomes to find a trade-off between the security and the program size and/or performance. An interesting breakthrough that was highlighted in this chapter is a comprehensive study of the link between the DCA success probability and the encoding size. More precisely, this study shows that any sensitive variable computed by a key-dependent function should be encoded with an encoding which has a size at least two times bigger than the size of the input.

Another promising approach consists in using a circuit implementation of the AES as a reference implementation and apply suitable countermeasures to defeat all the variants of a DCA attack. This approach is particularly interesting for practical purposes because it enables to implement the countermeasures with reasonable white-box size. Besides, the white-box implementation is provably secure regarding a first-order DCA attack. Yet, even with different layers of obfuscation and fault attacks countermeasures, the obtained white-box implementation can be broken. For instance, three white-box implementations of the AES-128 following this approach and submitted to the second edition of the WhibOx contest have been broken after 30, 50 and 51 days. It is still insufficient for critical use cases.

4

Theory of White-Box Cryptography

"Trust, but verify."

Ronald Reagan

Contents

In this chapter, we investigate how the security in the white-box model have been formalized. The security of encryption schemes is generally assessed regarding a security notion. A security notion enables to define both the property we manage to satisfy and the attack scenarios. Examples of security notions are IND-CPA aka Indistinguishability under Chosen-Plaintext Attack or IND-CCA which means Indistinguishability under Chosen-Ciphertext Attack. Of course those notions are not necessarily relevant for encryption schemes in the white-box model. Actually, the following notions have been introduced to capture the properties expected by white-box applications in practice: *unbreakability*, *incompressibility*, *traceability* and *one-wayness*. The definitions of these security notions are needed to design and formally prove the security of an encryption scheme in the white-box model. *"We can't build on sand" - Moses Isegawa.*

Introduction

When Chow et al. introduced the paradigm of White-Box Cryptography, their objective was to guarantee the secure execution of a cryptographic algorithm when an attacker can perform memory inspection or binary instrumentation. Consequently, they proposed a white-box implementation which produces a binary containing the instantiated key but scattered throughout the file. Even if such technique allows for preventing entropy attack or S-box blanking attack, they

could not formally prove that the master key won't be deduced from the data. They only gave an estimation on the diversity and ambiguity produced by their technique. Moreover, the variety of cryptanalysis results ranging from specific algebraic attacks to generic statistical ones show the number of exploitable vulnerabilities. Also, those results highlighted the fact that the structure of the algorithm (since the specification is public) helps for the cryptanalysis. For instance, the byte-oriented structure of the AES allows for attacking independently each byte of the key and the lack of MixColumns in the last round of the AES introduces the vulnerability to Differential Fault Analysis. Thus, a novel approach for White-Box Cryptography is to design white-box provably secure encryption schemes, i.e. encryption schemes that satisfy some white-box security notions. The work of Delerablée et al. [43] nicely formalizes the notions that seem relevant in the white-box model. Since the main problem addressed by White-Box Cryptography is the secrecy of the key, the notion of *unbreakability* was introduced to measure the hardness of recovering a key from the white-box implementation. In other words, an encryption scheme is unbreakable if an instantiated key cannot be recovered from the implementation. However, in the white-box model, when the adversary has access to a white-box implementation of an encryption scheme, they might copy the entire code and profit from the encryption scheme without knowing the key. This scenario is relevant in the context of Digital Rights Management where in the client side a user key is used to decrypt several content keys each used to decrypt a block of the content. The notion of *incompressibility* was introduced to state the difficulty of finding a more compact but functionally equivalent code given a white-box implementation. The notion of *traceability* was introduced to ensure that given a set of white-box implementations, an adversary cannot produce a new implementation with embedded key which is untraceable. This property dissuades a user from redistributing a decryption software. Finally, Delerablée et al. introduced the *one-wayness* notion which expresses the difficulty of encrypting any plaintext given a white-box implementation of the decryption algorithm and vice-versa. Such a property enables to define a public key encryption scheme from a secret key one: the key-instantiated white-box implementation can be used as a public key for signature while the secret key is used with the decryption algorithm to verify the signature.

4.1 White-Box security notions

In this section, we consider an encryption scheme $\mathcal{E}$ for which a white-box compiler $C_{\mathcal{E}}$ is defined. Thus, a white-box implementation of $\mathcal{E}$ refers to the program produced by a white-box compiler.

Definition 29 (Encryption scheme)**.** *Let λ be a security parameter. Let $\mathcal{K}$, M, C be the set of keys, plaintexts and ciphertexts, respectively. A symmetric encryption scheme $\mathcal{E}$ consists of three polynomial-time algorithms* $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *such that:*

- KGen *is a probabilistic algorithm which outputs a key $K \in \mathcal{K}$ given the secret parameter λ:*

$$K \leftarrow_{\$} \mathsf{KGen}(1^{\lambda})$$

- Enc *is a probabilistic algorithm which outputs a ciphertext $\mathsf{c} \in \mathsf{C}$ given a key K and a plaintext m:*

$$\mathsf{c} \leftarrow_{\$} \mathsf{Enc}(K, \mathsf{m})$$

- Dec *is a deterministic algorithm which outputs* $m \in M$ *given* K *and a ciphertext* c*:*

$$m \leftarrow \mathsf{Dec}(K, c)$$

The encryption scheme satisfies correctness i.e. for any plaintext $m \in M$*:*

$$\Pr[\mathsf{Dec}(K, \mathsf{Enc}(K, m)) = m] = 1$$

where the probability is taken over the randomness of Enc *and* $K \leftarrow_{\$} \mathsf{KGen}(1^{\lambda})$.

When K is fixed, the encryption is deterministic and $\mathsf{Enc}(K, \cdot)$ denoted by Enc_K is a cipher.

Definition 30 (White-box compiler)**.** *Let* $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a symmetric encryption scheme. A white-box compiler (for the encryption scheme* $\mathcal{E}$*) is a probabilistic algorithm* $C_{\mathcal{E}}$ *that outputs a white-box program* P *of the cipher for a fixed key* K *and a random nonce* r*:*

$$\mathsf{P} \leftarrow_{\$} C_{\mathcal{E}}(K, r)$$

- *If* P *implements* Enc_K *then* $\forall m \in M$, $\Pr[\mathsf{P}(m) = \mathsf{Enc}(K, m)] = 1$. *We say that* P *is a white-box encryption.*
- *If* P *implements* Dec_K *then* $\forall c \in C$, $\Pr[\mathsf{P}(c) = \mathsf{Dec}(K, c)] = 1$. *We say that* P *is a white-box decryption.*

The white-box program P is typically randomized thanks to a random nonce at the input of the compiler. We admit that this is implicit and do not explicitly write it.

Before giving the formal definitions of the security notions, we first give the attack models defined in [43]:

Chosen-Plaintext Attack aka CPA The adversary chooses the plaintexts and encrypts them using the program P.

Chosen-Ciphertext Attack aka CCA The adversary chooses the plaintexts and encrypts them using the program P. Additionally, they are allowed to choose ciphertexts and interact with a decryption oracle which returns the corresponding plaintexts.

Recompilation Attack aka RCA Given a white-box program with a fixed key, the adversary interacts with a recompiling oracle and gets white-box programs compiled with the same key and distinct nonces. This attack model includes the CPA model since the adversary is allowed to encrypt some chosen plaintexts with the obtained white-box programs.

Chosen Ciphertext and Recompilation Attack aka CCA-RCA The adversary has access to both the decryption oracle and the recompiling oracle, i.e. they choose ciphertexts and get the corresponding plaintext and also obtain white-box programs which embed the same key. As before this attack model includes the CPA model.

A security notion is a formal description of the security desired from an encryption scheme. It can be described through a security game between an adversary and a challenger. We denote by $\mathcal{A}$ a Probabilistic Polynomial Time (PPT) adversary. If λ is the security parameter then the running time of $\mathcal{A}$ is polynomial in λ. Let $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}$ be a negligible function, i.e. for any polynomial function $\mathsf{poly}\colon \mathbb{N} \to \mathbb{N}$ there exists λ_0 such that $\mathsf{negl}(\lambda) < 1/\mathsf{poly}(\lambda)$ for any $\lambda > \lambda_0$.

$\mathsf{UBK\text{-}ATK}^{\mathcal{A}}_{C_{\mathcal{E}}}$

1 : $K \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$
2 : $\mathsf{P} \leftarrow\!\!\$\ C_{\mathcal{E}}(K, r)$
3 : $K' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{P})$
4 : **return** $K' = K$

Figure 4.1: The unbreakability security game: ATK is either CPA,CCA, RCA or CCA-RCA.

Advantage of the adversary. Given a security game Exp involving an adversary $\mathcal{A}$ and a white-box compiler $C_{\mathcal{E}}$, the advantage of the adversary $\mathcal{A}$ to win Exp is the probability that the experiment returns 1:

$$\mathsf{Adv}^{\mathrm{exp}}_{C_{\mathcal{E}},\mathcal{A}}(\lambda) = \Pr_{K \leftarrow\!\!\$\, \mathcal{K}}\Big[\mathsf{Exp}(1^\lambda) = 1\Big] \tag{4.1}$$

Unbreakability. The unbreakability security states the hardness for an adversary to recover an instantiated key from a white-box program given additional oracle accesses or not. It is formalized as follows:

Definition 31 (UBK-ATK security)**.** *Let $\mathcal{E}$ be an encryption scheme and $C_{\mathcal{E}}$ be the corresponding white-box compiler. The white-box compiler $C_{\mathcal{E}}$ is said to be UBK-ATK secure if all PPT adversaries $\mathcal{A}$ given an oracle access $\mathcal{O}$ have negligible advantage in the $\mathsf{UBK\text{-}ATK}^{\mathcal{A}}_{C_{\mathcal{E}}}$ game specified in Figure 4.1. We write $\mathcal{A}^{\mathcal{O}}(\mathsf{P})$ to express that the adversary is run on P.*

We say that a white-box compiler is unbreakable if the full key cannot be deduced from a white-box program by a bounded adversary. However, an unbreakable compiler in the sense of the definition always leaks information about the key since the adversary can check if the guessed key corresponds to the correct key given the white-box program and enough plaintexts.

Incompressibility. The incompressibility security states the difficulty for an adversary to compress a large white-box program. In other words, this security notion ensures that an adversary cannot find a functionally equivalent program with smaller size or at least the size of any possible functionally equivalent program is bounded. We say that a program P' is functionally equivalent to a program P if P' returns the same outputs as P on most inputs.

Definition 32 (δ-functionality)**.** *Let P be a program with a set of inputs X. For any $0 \leqslant \delta \leqslant 1$, a program P' is δ-functional if the probability that for an input $x \in X$, the output of P' is distinct from $\mathsf{P}(x)$ is smaller than δ. We write $\mathsf{P}' \equiv_\delta \mathsf{P}$.*

$$\mathsf{P}' \equiv_\delta \mathsf{P} \implies \Pr_{x \in X}\big[\mathsf{P}'(x) \neq \mathsf{P}(x)\big] \leqslant \delta \tag{4.2}$$

Definition 33 ((B,δ)-INC-ATK security)**.** *Let $\mathcal{E}$ be an encryption scheme and $C_{\mathcal{E}}$ be the corresponding white-box compiler. The white-box compiler $C_{\mathcal{E}}$ is said to be INC-ATK (B,δ)-secure if all PPT adversaries $\mathcal{A}$ given an oracle access $\mathcal{O}$ have negligible advantage in the $\mathsf{INC\text{-}ATK}^{\mathcal{A}}_{C_{\mathcal{E}}}$ game specified in Figure 4.2.*

$\mathsf{INC\text{-}ATK}^{\mathcal{A}}_{C_{\mathcal{E}}}$

1 : $K \leftarrow_{\$} \mathsf{KGen}(1^{\lambda})$
2 : $\mathsf{P} \leftarrow_{\$} C_{\mathcal{E}}(K, r), \mathsf{size}(\mathsf{P}) \gg B$
3 : $\mathsf{P}' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{P})$
4 : **if** $\mathsf{P}' \equiv_{\delta} \mathsf{P} \wedge \mathsf{size}(\mathsf{P}') < B$ **then**
5 : **return** 1
6 : **else**
7 : **return** 0

Figure 4.2: The incompressibility security game: ATK is either CPA,CCA, RCA or CCA-RCA. size(P) expresses the size of P seen as binary string.

$\mathsf{OW\text{-}ATK}^{\mathcal{A}}_{C_{\mathcal{E}}}$

1 : $K \leftarrow_{\$} \mathsf{KGen}(1^{\lambda})$
2 : $\mathsf{P} \leftarrow_{\$} C_{\mathcal{E}}(K, r)$
3 : $\mathsf{m} \leftarrow_{\$} \mathsf{M}$
4 : $\mathsf{c} \leftarrow \mathsf{Enc}(K, \mathsf{m})$
5 : $\mathsf{m'} \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{P}, \mathsf{c})$
6 : **return** $\mathsf{m'} = \mathsf{m}$

Figure 4.3: The one-wayness security game: ATK is either CPA,CCA, RCA or CCA-RCA. size(P) expresses the size of P seen as binary string.

This notion might be quite strange from a theoretical perspective but is actually relevant in practice. Since a white-box model assumes that the adversary has access to the execution environment, they might copy parts or entire program's code which they know contains the key. This is for instance a real threat for digital content distribution where a redistributed decryption program enables non-subscribed users to get access to premium content. The incompressibility property then ensures that if the adversary does not transmit the whole code because of size constraints then the program will likely not be functional.

One-wayness. The one-wayness notion is quite similar to the one-wayness notion for public-key scheme: the adversary cannot recover the plaintext m of a given ciphertext c without the knowlegde of the secret key. Thus, the one-wayness security implies the unbreakability security.

Definition 34 (OW-ATK security)**.** *Let $\mathcal{E}$ be an encryption scheme and $C_{\mathcal{E}}$ be the corresponding white-box compiler. The white-box compiler $C_{\mathcal{E}}$ is said to be OW-ATK secure if all PPT adversaries $\mathcal{A}$ given an oracle access $\mathcal{O}$ have negligible advantage in the $\mathsf{OW\text{-}ATK}^{\mathcal{A}}_{C_{\mathcal{E}}}$ game specified in Figure 4.3.*

Traceability. While all previous notions are quite generic in the sense that they are not specific to a use case, the traceability notion in contrary is specific to the context of digital content distribution. For such application, a broadcast encryption scheme is used to distribute encrypted content to a set of privileged users over an insecure channel. In this case, each user is equipped with a decryption recipient containing a key which enables him to decrypt a content. Yet, when

$\mathsf{TR\text{-}ATK}^{\mathcal{A}}_{C_{\mathcal{E}}}$

1 : $K \leftarrow\!\!\$\ \mathsf{KGen}(1^{\lambda})$
2 : $\mathsf{P}_i \leftarrow\!\!\$\ C_{\mathcal{E}}(K, i)$
3 : $\mathsf{P}' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{P}_i)$
4 : $i' \leftarrow \mathcal{T}(\mathsf{P}_i)$
5 : **return** $i' = i$

Figure 4.4: The traceability security game: ATK is either CPA,CCA, RCA or CCA-RCA.

one or several users collaborate (we say that they collude) to illegally distribute pirate decryption recipient, the financial loss for providers might be substantial. To mitigate this attack, a traitor tracing scheme have been proposed: a traitor tracing scheme is a broadcast encryption scheme with a tracing procedure and so allows for accusing the users who leaked their key in a pirate recipient. When the tracing procedure enables to find at least one user from the collusion, we say that the tracing scheme is *collusion resilient*. We will give more details about traitor tracing in the second part of this thesis. For now, we define the traceability notion as the difficulty for an adversary to construct an untraceable functional program.

Definition 35 (Traceable white-box compiler)**.** *Let $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme. A traceable white-box compiler is a probabilistic algorithm $C_{\mathcal{E}}$ that outputs a white-box program P_i of a cipher given a fixed key K and an index i:*

$$\mathsf{P}_i \leftarrow\!\!\$\ C_{\mathcal{E}}(K, i)$$

such that there exits a procedure $\mathcal{T}$ which outputs the index i given P_i as input:

$$i \leftarrow \mathcal{T}(\mathsf{P}_i)$$

A traceable white-box compiler allows for specifying the white-box program, e.g. for a particular user. Compared to the first definition of a white-box compiler, the random nonce is replaced by the index i.

Definition 36 (TR-ATK security)**.** *Let $\mathcal{E}$ be an encryption scheme and $C_{\mathcal{E}}$ be the corresponding traceable white-box compiler. The white-box compiler $C_{\mathcal{E}}$ is said to be TR-ATK secure if all PPT adversaries $\mathcal{A}$ given an oracle access $\mathcal{O}$ have negligible advantage in the $\mathsf{TR\text{-}ATK}^{\mathcal{A}}_{C_{\mathcal{E}}}$ game specified in Figure 4.4.*

For the traceability security game, the recompiling attack scenario is particularly relevant in practice because the adversary might obtain several programs with different indexes. Their goal is to output a program such that the tracing procedure $\mathcal{T}$ does not output any of these indexes.

The traceability notion as defined by Delerablée et al. [43] is a bit different: they actually described a way to make a white-box program traceable by "unnoticeably modifying its functionality". If such modifications also called *perturbations* are kept secret then they can be used to trace a program. Therefore, they introduced the notions of *Perturbation-Value Hiding* and *Perturbation-Index Hiding* for a white-box compiler that supports perturbations and showed that if the white-box compiler is secure in the sense of these notions then there exists a tracing

procedure for the white-box programs. They adapted the *Private Linear Broadcast Encryption* (PLBE) scheme to describe a tracing scheme for programs generated by the white-box compiler.

We will introduce in the second part of this thesis a traitor tracing scheme which can be used in the white-box model.

4.2 ASASA-based white-box encryption

ASASA [14] refers to an encryption scheme which alternates an affine layer A and a non-linear substitution layer S. The substitution layer is typically composed of parallel secret S-boxes while the affine layer is just a secret affine mapping. An ASASA scheme can be used to define a public key scheme where the public key is the composition of the five layers and the secret key is the set of secret components. In that case, the scheme should be hard to decompose, i.e. it is unfeasible to deduce the secret components of both the affine and the substitution layers from the composition. If the substitution layer is invertible, i.e. the S-boxes are bijective, the ASASA scheme can also define a secret key scheme. Actually, the ASASA structure is an extension of the basic ASA which is used in *Multivariate Cryptography* [62] to design public key encryption schemes or signature schemes. Also, an ASA structure is the basic block of a SPN-type block cipher and for instance the AES-128 has 19 such layers.

Biryukov et al. [14] were the first to propose a white-box blok cipher based on the ASASA structure. Their idea was to use an ASASA scheme to construct a basic block of a SPN-type block cipher, which is the S-box. The S-box calculated from the ASASA component is implemented by a lookup table and the cipher consists in R iterations of two transformations: a substitution made of several S-boxes and a public linear mapping. In such construction, the secret key is composed of the secret S-boxes and the secret affine mappings of ASASA. Assuming that the ASASA scheme is hard to decompose, an implementation of the ASASA-based block cipher satisfies the unbreakability notion. Moreover, Biryukov et al. defined the notion of weak white-box security to capture the difficulty to find an equivalent representation of the cipher of a smaller size given the white-box implementation. In that sense, weak white-box security is equivalent to the notion of incompressibility. They also defined the notion of strong white-box security which expresses the difficulty to find an equivalent representation of a decryption function given a white-box implementation of the encryption. This is quite similar to the one-wayness notion.

Let us first explicitly define an ASASA scheme.

Definition 37 (m-bit ASASA scheme)**.** *Let A_1, A_2, A_3 be secret invertible affine mappings from $\mathbb{F}_2^m$ to itself and let S_1, S_2 be two substitution mappings also from $\mathbb{F}_2^m$ to itself. An ASASA scheme $E\colon \mathbb{F}_2^m \to \mathbb{F}_2^m$ is defined as:*

$$E = A_3 \circ S_2 \circ A_2 \circ S_2 \circ A_1$$

Conventionally and, following the approach of Multivariate Cryptography, an ASASA scheme is described by a system of multivariate polynomials. So the degree of the system is at most equals to $\deg S_1 \times \deg S_2$ since S_1 and S_2 are non-linear. To minimize the total degree of the system, the degree of each S-box is chosen to be 2, leading to a total degree of at most 4.

The ASASA scheme was used to construct two public key schemes using respectively, the quadratic χ function[11] and random expanding S-boxes for the substitution layer [14]. These schemes are claimed by the authors to be strongly white-box secure.

[11] Used by Keccak.

Before giving the definitions of weak and strong white-box securities, we first point out the differences of the notions used in [14] and in [43]. The security notions introduced in [43] regard a white-box compiler which outputs a program, in a very generic sense, i.e. seen as a word interpreted in the context of a programming model and an execution model. So, a white-box implementation in their sense is a fully functional program outputted by the compiler with respect to a certain function (e.g the encryption function).

In [14], the definition of a white-box implementation is less formal: given a symmetric encryption scheme with an encryption function Enc and a decryption function Dec, they define the notion of equivalent key.

Definition 38 (Equivalent key)**.** *Let $\mathcal{E}$ be a symmetric encryption scheme with a key space $\mathcal{K}$. For any key $K \in \mathcal{K}$, a set S_K is said to be a set of equivalent keys of K if for any $K' \in S_K$, there exits a function Enc' such that $\mathsf{Enc}'_{K'} = \mathsf{Enc}_K$. In other words, $\mathsf{Enc}'_{K'}$ is equivalent to the function Enc_K.*

Definition 39 (B-weak white-box security)**.** *Let $\mathcal{E}$ be a symmetric encryption scheme. For any key K, let Enc' be a white-box implementation of Enc_K. We say that Enc' is B-weakly white-box secure if it is computationally hard to find an equivalent key $K^{'}$ of length smaller than B given a full access to Enc'.*

In other words, an attacker who has access to the white-box implementation and is able to encrypt any plaintext is not able to implement the encryption function with a smaller code size. The term *computationally* is important since an unbounded attacker might try all possible values of the key until they find a match for sufficient number of plaintexts.

4.2.1 Construction of a white-box block cipher.

Let n be the plaintext/ciphertext length and let $k = 128$ be the key length, both in bits. An ASASA-based block cipher denoted ASASA-(m, t, R) instantiated with a key K consists in R rounds of two transformations: a substitution which applies t S-boxes with m-bit input and a linear mapping $L\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$. Any round r of ASASA-(m, t, R) is defined by:

$$\mathsf{Round}^{(r)} = L \circ (\mathsf{SB}_1^r, \ldots, \mathsf{SB}_t^r)$$

where $\mathsf{SB}_i^r = E_i^r$ for an m-bit ASASA scheme E_i^r. Each of the secret mappings $A_{i,1}^r$,$A_{i,2}^r$,$A_{i,3}^r$, $S_{i,1}^r$,$S_{i,2}^r$ are derived from the secret key K. Each m-bit S-box SB_i^r, i.e. the S-box of the white-box block cipher, is composed of t' m'-bit ASASA S-boxes. For instance, if $m' = 8$ and $t' = 2$, then the white-box block cipher uses $8 \times 2 = 16$ S-boxes.

The security of the block cipher, i.e. the weak white-box security, depends on the hardness to decompose a m-bit S-box. Such a decomposition is made possible by a structural attack on the ASASA S-box. As shown is [14], the maximum level of security of an ASASA scheme with m variables and m'-bit S-boxes regarding a structural attack doesn't exceed $(m - m')m'$.

Table 4.1 gives the weak white-box security of different instances of ASASA-(m, t, R).

Definition 40 (Strong white-box security)**.** *Let $\mathcal{E}$ be a symmetric encryption scheme. For any key K, let Enc' be a white-box implementation of Enc_K. We say that Enc' is strongly white-box secure if it is computationally hard to find an equivalent function Dec' of Dec_K given a full access to Enc'.*

Name	m	t	n	WB size	Security (bits)
ASASA-$(16, 4)$	16	4	64	2 MB	64
ASASA-$((18, 3), (10, 1))$	$18, 10$	$3 + 1 = 4$	64	9 MB	64
ASASA-$((24, 1), (16, 6), (8, 1))$	$24, 16, 8$	$1 + 6 + 1 = 8$	128	384 MB	64

Table 4.1: Instances of an ASASA-based block cipher: n is the plaintext/ciphertext length, m is the S-box input length, t is the number of S-boxes in a substitution layer. The security of the block cipher corresponds to the complexity of a structural attack. ASASA-$((18, 3), (10, 1))$ means that the substitution layer is composed of 3 18-bit S-boxes and one 10-bit S-box.

As we said, the strong white-box security is somehow similar to the one-wayness security notion. However, in the one-wayness notion, the attacker is only required to find the plaintext corresponding to a ciphertext given the white-box implementation of the encryption. The strong white-box security is slightly different since it requires that the attacker compute (efficiently) an equivalent decryption function with an equivalent key which allows him to decrypt all ciphertexts. Consequently, partial equivalent function is not considered in this definition of strong white-box security.

Two constructions based on the ASASA have been proposed to achieve the strong white-box security. Actually, the encryption schemes can also be used as public-key schemes.

4.2.2 Constructions of strong white-box encryptions.

The χ-scheme. The non-linear (substitution) layer consists of a quadratic function χ of length l defined for any input $(x_0, \ldots, x_{l-1})$ as:

$$\chi(x_0, \ldots, x_{l-1}) = (y_0, \ldots, y_{l-1})$$

where for any $0 \leqslant i \leqslant l-1$, $y_i = x_{i \mod l} \oplus x_{i+1 \mod l} \otimes x_{i+2 \mod l}$.

For instance, for $l = 127$ (the length of χ is always odd), the public key consists of 127 polynomials of degree 4 of 127 variables. To hide the ASASA structure, secret perturbation polynomials are added before the last affine layer. So 24 random polynomials of degree 4 of 8 variables are added. We refer to [14] for the details about the number of random polynomials that should be used. The random polynomials should remain secret to avoid an interpolation attack. Also, note that the χ function constitutes a single S-box in the substitution layer. If small S-boxes are used then the scheme is vulnerable to a structural attack.

Expanding S-boxes. The non-linear layer is composed of either 8 or 16, 16-to-32 bits S-boxes (the output size is twice bigger than the input size, that is why they are called *expanding S-boxes*). Each S-box is described by a polynomial of degree 2 over $\mathbb{F}_{16}$. So, the public key is represented by 128 polynomials of degree 4 over $\mathbb{F}_{16}$. The private key is composed of the S-boxes and the three affine transformations which operate respectively on 128, 256 and 512 bits. As for the χ-scheme, some perturbation polynomials are added. Thus, 32 perturbation polynomials over $\mathbb{F}_{16}$ of degree 4 are used to hide the ASASA structure.

The strong white-box security is estimated with the complexity of the best known attack. So, the generic attack to recover the plaintext corresponding to a ciphertext is the Grobner basis attack which has a complexity of $\mathcal{O}(q^n)$ for polynomials with n variables over $\mathbb{F}_q$. Since, the attacks on the weakened variants of both constructions do not apply on these schemes, Biryukov et al. estimated that they are strong enough. However, several attacks on both scheme have been published [90, 59, 47].

4.3 SPACE family

SPACE is a family of *white-box dedicated* block ciphers based on a Feistel structure proposed by Bogdanov and Isobe in [20]. It is "white-box dedicated" because it satisfies the unbreakability security by construction. Actually, the key extraction security in the white-box model is reduced to the problem of key recovery of the AES in the black-box model. Since so far, no efficient key recovery attack exists for the AES, it is admitted to be secure. Consequently, the probability that a black-box adversary recovers the AES key is negligible and so is the probability that a white-box adversary extracts the key from the white-box implementation.

In addition, the parameters (B, δ) for the incompressibility security can be estimated, also by construction. Bogdanov and Isobe introduced the *space-hardness* notion to measure the difficulty of compressing the white-box implementation. The space-hardness notion is equivalent to the incompressibility notion of Definition 33 applied to block ciphers. It has been defined as follows:

Definition 41 ((B, Z)-space hardness)**.** *For a key $K \in \mathbb{F}_2^k$, let $E_K \colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a block cipher. The implementation of E_K is (B, Z)-space hard if the probability that a code of size less than B encrypts (resp. decrypts) a random plaintext (resp. ciphertext) is at most 2^{-Z}.*

We see that for a block cipher E_K, if the white-box compiler that produces the implementation is (B, δ)-INC-ATK secure for any ATK $\in \{$CPA, CCA, RCA, CCA-RCA$\}$ then the implementation is (B, Z)-space hard with $Z = -\log_2(\delta)$.

Construction of SPACE. We borrow the notations in [20]. Let n be the plaintext/ciphertext length in bits, k be the key length in bits, l be the number of lines of a generalized Feistel network each of size n_a, i.e. a n-bit state is divided into l blocks of n_a bits. SPACE-(n_a, R) is an l-line target heavy generalized Feistel network with R rounds such that for any round $1 \leqslant r \leqslant R$, a n-bit state $X^r = (x_0^r, \ldots, x_{l-1}^r)$ is updated by a round function $F_{n_a}^r \colon \mathbb{F}_2^{n_a} \to \mathbb{F}_2^{n_b}$ as follows:

$$X^{r+1} = (F_{n_a}^r(x_0^r) \oplus (x_1^r || \ldots || x_{l-1}^r)) || x_0^r$$

where for any $x \in \mathbb{F}_2^{n_a}$,

$$F_{n_a}^r(x) = \lceil E_K(0_{n_b} || x) \rceil_{n_b} \oplus r$$

where $n_b = n - n_a$, 0_{n_b} is the n_b-bit string with 0 only and E_K is the AES-128, i.e. the AES block cipher with a 128-bit key.

White-box implementation of SPACE. The round function $F_{n_a}^r$ can be implemented by a lookup table T_{n_a} by computing all possible values of $\lceil E_K(0_{n_b} || x) \rceil_{n_b}$ for any $x \in \mathbb{F}_2^{n_a}$. Thus, for any $x \in \mathbb{F}_2^{n_a}$, $F_{n_a}^r(x) = \mathsf{T}_{n_a}(x) \oplus r$. For well-chosen values of n_a, it does not require too much storage space (Table 4.2).

The parameters (B, Z) for space-hardness are measured thanks to a compression attack and given in Table 4.3.

Name	n	l	n_a	R	Round function	WB size
SPACE-$(8, 300)$	128	16	8	300	$F_8^r : \mathbb{F}_2^8 \to \mathbb{F}_2^{120}$	3.84 kB
SPACE-$(16, 128)$	128	8	16	128	$F_{16}^r : \mathbb{F}_2^{16} \to \mathbb{F}_2^{112}$	918 kB
SPACE-$(24, 128)$	128	16	24	128	$F_{24}^r : \mathbb{F}_2^{24} \to \mathbb{F}_2^{104}$	218 MB
SPACE-$(32, 128)$	128	4	32	128	$F_{32}^r : \mathbb{F}_2^{32} \to \mathbb{F}_2^{96}$	51.5 GB

Table 4.2: Instantiations of SPACE.

Compression attack. The attack consists in estimating the probability p that an adversary correctly encrypts a random plaintext given a compressed version of the lookup table. It can be described as follows:

- Assume that a leakage function enables the adversary to obtain m entries of T_{n_a}, i.e. $m \leqslant 2^{n_a}$. The adversary is then able to store these entries in a lookup table T'_{n_a} of size $S = m \times n_b$ bits.
- Compute the probability p_x that a random table input x corresponds to one of the entries of T'_{n_a}. It is given by:

$$p_x = \frac{m \times n_b}{2^{n_a} \times n_b} = \frac{m}{2^{n_a}}$$

- Compute the probability p that given a random plaintext, the corresponding ciphertext can be computed with T'_{n_a}:

$$p = p_x^R = \left(\frac{m}{2^{n_a}}\right)^R$$

  since one table input is looked in T'_{n_a} for each round.

- The parameters (B, Z) can be estimated as:

$$\begin{aligned} p &\leqslant 2^{-Z} \\ \left(\frac{m}{2^{n_a}}\right)^R &\leqslant 2^{-Z} \\ \left(\frac{S}{2^{n_a} \times n_b}\right)^R &\leqslant 2^{-Z} \\ S &\leqslant 2^{-Z/R+n_a} \times n_b \end{aligned}$$

  Hence, $B = 2^{-Z/R+n_a} \times n_b$.

The compression attack allows for determining an upper bound B on the size of adversary's table. It assumes that the adversary has access to random table entries through a leakage function. Actually, the adversary might be stronger by adaptively choosing the table entries

Name	R	WB size	B	Z
SPACE-$(16, 64)$	64	918 kB	229.5 kB	128
SPACE-$(24, 64)$	64	218 MB	54.5 MB	128
SPACE-$(32, 25)$	25	51.5 GB	12.875 GB	128

Table 4.3: Parameters for (B, Z) space-hardness for different instances of SPACE.

Name	R	WB size	B	Z
SPACE-$(8, 300)$	300	3.84 kB	2.45 kB	128
SPACE-$(16, 128)$	128	918 kB	230 kB	128
SPACE-$(24, 128)$	128	218 MB	105 MB	128
SPACE-$(32, 128)$	128	51.5 GB	12.9 GB	128

Table 4.4: Parameters for (B, Z) strong space-hardness for different instances of SPACE.

while still being bounded in the number of entries they can choose. This stronger attack model is considered by Bogdanov et al. in [21]. We will detail the refined attack models for space-hardness in Section 4.4.

Bogdanov and Isobe also introduced the notion of *strong space-hardness*. This notion is merely an adaptation of the *strong white-box security* of [14] to their construction. This is also an equivalent notion for one-wayness defined in Definition 34. It is defined as follows:

Definition 42 (Strong (B, Z)-space-hardness)**.** *For a key $K \in \mathbb{F}_2^k$, let $E_K \colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a block cipher. The implementation of E_K is (B, Z) strongly space-hard if the probability that a code of size less than B enables to forge a valid plaintext/ciphertext pair is at most 2^{-Z}.*

The notion of strong space-hardness is relevant to ensure the security against forgeries in the context of message authentication codes. It is also useful if a white-box implementation is used as a "public key" and the secret key as a private key in a signature scheme. So, an adversary should not be able to produce a valid signature given only the white-box implementation.

As for the (weak) space-hardness, the parameters (B, Z) are estimated through a compression attack assuming that the adversary chooses l consecutive table entries. The obtained parameters are given in Table 4.4.

4.4 SPNbox family

SPNbox is a family of white-box dedicated block ciphers introduced in [21] with optimal performances. Compared to SPACE, a white-box implementation of SPNbox is 18 times speeder while having an equivalent security level against a key extraction attack. SPNbox is also secure in the

black-box model with a constant-time implementation. Thanks to these advantages, the SPNbox family can be used in real-world applications such as Digital Rights Management enforcement and mobile payments.

Besides, they improved the work of Bogdanov and Isobe by refining the notion of space-hardness. Actually, they defined new attack models which precise the leakage function. They introduced the following attack scenarios: Known-Space Attack (KSA), Chosen-Space Attack (CSA) and Adaptively Chosen-Space Attack (ACSA) and defined the following security goals: weak space-hardness and strong space-hardness.

We consider a white-box implementation of a block cipher which consists of a single lookup table T. The different attack models are defined as follows:

Definition 43 (Known-space attack)**.** *A known-space attack corresponds to an attack scenario where the adversary obtains q pairs of table inputs and the corresponding outputs, i.e. the adversary has access to a leakage function f which outputs a set $\{(x_i, \mathsf{T}(x_i)) : 1 \leqslant i \leqslant q\}$.*

Here, the leakage function outputs random pairs of input/output from T. This attack model is the one considered in [20]. In practice, this attack model corresponds to a limited control of the adversary over the platform, e.g. through a malware or trojan. Thus, they have a time bounded access to the platform and get a set of data which may contains table values. The known-space attack model is the weakest attack model.

Definition 44 (Chosen-space attack)**.** *A chosen-space attack corresponds to an attack scenario where the adversary obtains q pairs of table inputs and the corresponding outputs, for a set of a priori chosen inputs, i.e. the adversary has access to a leakage function f which outputs a set $\{(x_i, \mathsf{T}(x_i)) : 1 \leqslant i \leqslant q\}$ where all x_i are chosen by the adversary once and for all.*

In this model, it is assumed that the adversary is able to isolate the lookup table and choose a set of entries, e.g. if they are able to identify the data corresponding to the lookup table in memory. However, they might be limited in the amount of data they can transmit or by the timing access to the resources.

Definition 45 (Adaptively chosen-space attack)**.** *An adaptively chosen-space attack corresponds to an attack scenario where the adversary obtains q pairs of table inputs and the corresponding outputs, for a set of adaptively chosen inputs, i.e. the adversary has access to a leakage function f which outputs a set $\{(x_i, \mathsf{T}(x_i)) : 1 \leqslant i \leqslant q\}$ where the input x_{i+1} can be chosen according to the set $\{(x_1, \mathsf{T}(x_1)), \ldots, (x_i, \mathsf{T}(x_i))\}$.*

The adaptively chosen-space model corresponds to the strongest attack scenario where the adversary has a full access to the environment. In practice, the adversary may use debugger and decompiler tools to get information about the lookup table.

Construction. Let n be the plaintext/ciphertext length in bits, m be the block length in bits and t be the number of blocks such that $n = m \times t$. SPNbox-m is a Substitution-Permutation Network block cipher which encrypts a n-bit plaintext under a k-bit key. It is composed of $R = 10$ rounds such that any r-th round function applies the following transformations to any state $X = (X_0, \ldots, X_{t-1})$:

1. A Substitution $\gamma \colon \mathbb{F}_{2^m}^t \to \mathbb{F}_{2^m}^t$ which applies a S-box S to each block as follows:

$$\gamma(X) = (\mathsf{S}(X_0), \ldots, \mathsf{S}(X_{t-1}))$$

Block size m	**Matrix $\mathbf{M_m}$**	**Base field**
8	$\mathsf{had}(08, 16, 8a, 01, 70, 8d, 24, 76, a8, 91, ad, 48, 05, b5, af, f8)$	$\mathbb{F}_{2^8} = \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$
16	$\mathsf{had}(1, 3, 4, 5, 6, 8, b, 7)$	$\mathbb{F}_{2^{16}} = \mathbb{F}_2[x]/(x^{16} + x^5 + x^3 + x + 1)$
24	$\mathsf{cir}(1, 2, 5, 3, 4)$	$\mathbb{F}_{2^{24}} = \mathbb{F}_2[x]/(x^{24} + x^4 + x^3 + x + 1)$
32	$\mathsf{cir}(1, 2, 4, 6)$	$\mathbb{F}_{2^{32}} = \mathbb{F}_2[x]/(x^{32} + x^7 + x^3 + x^2 + 1)$

Table 4.5: The Maximum Distance Separable matrix M_m for different values of m.

2. A linear transformation $\theta\colon \mathbb{F}_{2^m}^t \to \mathbb{F}_{2^m}^t$ multiplies the state by a Maximum Distance Separable matrix M_m as follows:

$$\theta(X) = (X_0, \ldots, X_{t-1}) \cdot \mathsf{M}_m$$

According to the value of m, the matrix M_m is given in Table 4.5.

3. A round-dependent transformation $\sigma^r\colon \mathbb{F}_{2^m}^t \to \mathbb{F}_{2^m}^t$ adds a round-dependent constant to the state as follows:

$$\sigma^r(X) = (X_0 \oplus c_0^r, \ldots, X_{15} \oplus c_{15}^r)$$

for $c_i^r = (r-1) \cdot t + i + 1$.

The round function of SPNbox-m is similar to the round function of any SPN block cipher except for the key addition which is combined with the substitution layer here. Thus, the S-box S is key-dependent. For any key K, the S-box S is computed as follows: for any m-bit input x, encrypt x with a small-scale variant of the AES cipher using K as a master key and expanded using the SHAKE key derivation function [51]. The variant of the AES cipher, which we call *mini-AES*, is composed of R' rounds of the following transformations: SubBytes, AddRoundKey and MixColumns. The SubBytes transformation is composed of $\frac{m}{8}$ AES S-boxes and the Mixcolumns transformation applies either the identity matrix to the state (for $m = 8$) or a submatrix of MC.

One round of mini-AES updates a state $x = (x_0, \ldots, x_{m/8-1})$ as follows:

1. SubBytes applies the AES S-box to each block:

$$\mathsf{SubBytes}(x) = (\mathsf{SB}(x_0), \ldots, \mathsf{SB}(x_{m/8-1})).$$

2. MixColumns multiplies the state with a submatrix MC_m of MC:

$$\mathsf{MixColumns}(x) = \mathsf{MC}_m \cdot \begin{pmatrix} x_0 \\ \vdots \\ x_{m/8-1} \end{pmatrix}.$$

Block size m	**Number of blocks**	$\mathsf{MC_m}$	**Number of rounds R′**
8	1	I_8	64
16	2	$\begin{pmatrix} 02 & 03 \\ 01 & 02 \end{pmatrix}$	32
24	3	$\begin{pmatrix} 02 & 03 & 01 \\ 01 & 02 & 03 \\ 01 & 01 & 02 \end{pmatrix}$	20
32	4	$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$	16

Table 4.6: The parameters of the mini-AES cipher.

If $m = 8$, the MixColumns matrix is the identity matrix. For other values of m, the matrices are given in Table 4.6.

3. AddRoundKey adds the m-bit round key $K^r = (K^r_0, \ldots, K^r_{m/8-1})$ to the state:

$$\mathsf{AddRoundKey}(x) = (x_0 \oplus K^r_0, \ldots, x_{m/8-1} \oplus K^r_{m/8-1}).$$

Key schedule. The master key K is expanded to $R' + 1$ m-bit round keys using the SHAKE-128 algorithm [51].

The parameters for SPNbox are summarized in Table 4.7.

White-box implementation of SPNbox. As for SPACE, the white-box implementation of SPNbox consists of a single lookup table which corresponds to the S-box S. Thus, the white-box size as referred in Table 4.7 is the size of the S-box as a lookup table. Compared to SPACE, SPNbox requires much less storage space.

Weak space-hardness. Depending on the attack model, Bogdanov et al. gave an upper bound on the probability of success of the adversary given a space S. Thus, they estimated the probability of success of the adversary for weak space-hardness under known-space attack to be upper bounded by $\left(\frac{S}{2^m \times m}\right)^{tR}$ where S is the "known-space" size, i.e. the size of the lookup table constructed by the adversary and $T = 2^m \times m$ is the white-box size, i.e. the size of the S-box as

Name	m	t	n	Base field	Irreducible polynomial	WB size
SPNbox-8	8	16	128	$\mathbb{F}_{2^8}$	$x^8 + x^4 + x^3 + x + 1$	256 B
SPNbox-16	16	8	128	$\mathbb{F}_{2^{16}}$	$x^{16} + x^5 + x^3 + x + 1$	132 kB
SPNbox-24	24	5	120	$\mathbb{F}_{2^{24}}$	$x^{24} + x^4 + x^3 + x + 1$	50.3 MB
SPNbox-32	32	4	128	$\mathbb{F}_{2^{32}}$	$x^{32} + x^7 + x^3 + x^2 + 1$	17.2 GB

Table 4.7: The different instances of the SPNbox family.

a lookup table. Since, the plaintext given as a challenge to the adversary is randomly generated, they cannot predict the table inputs needed for the encryption. Thus, the probability of success of the adversary for weak space-hardness under chosen-space attack is the same as for WSH-KSA. In the case of weak space-hardness under adaptively chosen-space attack, the adversary is able to prepare a set of plaintext/ciphertext pairs by adaptively choosing table inputs. So the probability of success of the adversary is the probability that the challenge plaintext is among the prepared pairs. Assuming that N is the number of plaintext/ciphertext pairs computed by the adversary from the "adaptively chosen-space" of size S, the probability is upper bounded by $\frac{N}{2^{128}} + \left(1 - \frac{N}{2^{128}}\right) \cdot \left(\frac{S}{2^m \times m}\right)^{tR}$ and $N = \lceil \log_{\frac{2^m-1}{2^m}} (\frac{1-S/(2^m \times m)}{tR}) \rceil$.

Those probabilities enable to compute the lower bounds B on the required space S, i.e. the minimal space size S needed by the adversary to have a success probability greater than 2^{-Z}.

Strong space-hardness. As for the weak space-hardness, the lower bound B on the required space S is computed by evaluating the success probability of the adversary in each attack model. Although strong space-hardness cannot be ensured in the adaptively chosen-space model, the probability of success of the adversary in the known-space attack and chosen-space attack models is upper-bounded by $2^n \cdot \left(\frac{S}{2^m \times m}\right)^R$.

The parameters (B, Z) for weak and strong space hardness are given in Table 4.8.

4.5 Provable white-box primitives

SPACE was the first white-box dedicated block cipher with reasonable security regarding unbreakability and space-hardness. Especially, Bogdanov and Isobe introduced a general method to ensure that the table is indistinguishable from a random function, thus enforcing unbreakability. However, it lacks from a formal security proof about the resistance of the implementation against an adversary who tries to compress the lookup table used for the cipher. In [56], Fouque et al. tried to fill this gap by introducing a family of white-box dedicated block ciphers, in the same flavor as SPACE but with a probable security. They kept the notion of incompressibility introduced by Delerablée et al. and declined it into *weak incompressibility* and *strong incompressibility*. Weak incompressibility is then a formalization of weak space-hardness. More precisely, they defined three security notions ENC-TCOM, IND-COM and ENC-COM. ENC-TCOM is the formalization of weak incompressibility where the objective of the adversary is to *encrypt* (ENC) and the attack scenario is a *table compression* (TCOM). In the same way, IND-COM refers to the

Name	WB size	B				
		WSH-KSA	WSH-CSA	WSH-ACSA	SSH-KSA	SSH-CSA
SPNbox-8	$T = 256$ B	$T \cdot 2^{-0.80}$	$T \cdot 2^{-0.80}$	$T \cdot 2^{-0.80}$	$T \cdot 2^{-1.60}$	$T \cdot 2^{-1.60}$
SPNbox-16	$T = 132$ kB	$T \cdot 2^{-1.61}$	$T \cdot 2^{-1.61}$	-	$T \cdot 2^{-3.20}$	$T \cdot 2^{-3.20}$
SPNbox-24	$T = 50.3$ MB	$T \cdot 2^{-2.57}$	$T \cdot 2^{-2.57}$	-	$T \cdot 2^{-4.96}$	$T \cdot 2^{-4.96}$
SPNbox-32	$T = 17.2$ GB	$T \cdot 2^{-3.20}$	$T \cdot 2^{-3.20}$	-	$T \cdot 2^{-6.40}$	$T \cdot 2^{-6.40}$

Table 4.8: Parameters for weak and strong space-hardness: B gives the lower bound for space-hardness with respect to the probability 2^{-Z}.

strong incompressibility notion but here the adversary's goal is to distinguish (equivalently the security objective is the *indistinguishability* IND) and the attack scenario is a *compression* (an arbitrary implementation compression, not only a table compression). The third notion ENC-COM is the strong incompressibility notion for an encryption scheme which uses a white-box key generator. In other words, the white-box key generator outputs a key which is then used by a conventional symmetric encryption scheme. Thus, the adversary's goal is to *encrypt* with the encryption scheme under the attack scenario of *compressing* the white-box key generator.

In the following, we describe the two propositions of [56] which are called WhiteBlock and WhiteKey: WhiteBlock and WhiteKey refer respectively to a white-box block cipher and a white-box key generator.

WhiteBlock construction. Let $n = 128$ be the plaintext/ciphertext length in bits and $k = 128$ be the key length in bits. Let R be the number of rounds; each round is composed of a Feistel round function which calls t m-to-$\frac{n}{2}$ lookup tables and a permutation layer which consists in a block cipher E instantiated with a round key. More specifically, $\mathsf{WhiteBlock}(m, t, R, E)$ consists of R rounds of the following transformations:

- A round function $F\colon (\mathbb{F}_2^m)^t \to \mathbb{F}_2^{n/2}$ applies t m-to-$\frac{n}{2}$ mappings implemented as lookup tables $\mathsf{T}_0, \ldots, \mathsf{T}_{t-1}$ such that for any input $(x_0, \ldots, x_{t-1})$ where $x_i \in \mathbb{F}_2^m$:

$$F(x_0, \ldots, x_{t-1}) = \mathsf{T}_0(x_0) \oplus \mathsf{T}_1(x_1) \oplus \ldots \oplus \mathsf{T}_{t-1}(x_{t-1}).$$

  Specifically, each mapping T_i is defined for any $x \in \mathbb{F}_2^m$ as follows:

$$\mathsf{T}_i(x) = \lfloor E_K(c||x) \rfloor_{n/2}$$

  where K is the secret key of the cipher and $c \in \mathbb{F}_2^{n-m}$ is a fixed secret parameter.

- The block cipher E instantiated with a round key K^r.

The r-th round function of $\mathsf{WhiteBlock}(m, t, R, E)$ is defined for any input $(x_0, \ldots, x_{t-1}, x_t, \ldots, x_{n/m-1}) \in (\mathbb{F}_2^m)^{n/m}$ as follows:

$$\mathsf{Round}^{(r)} = E_{K^r}(F(x_0, \ldots, x_{t-1}) \oplus (x_t, \ldots, x_{n/m-1})||x_0, \ldots, x_{t-1})$$

WhiteKey construction. Let K be a k-bit master key and K^1, K^2 be two independent 128-bit keys. We denote by s_1, s_2 two seeds. For any (K^1, K^2, s_1, s_2), let m be the table input length, t be the number of table calls and $d = \lceil \sqrt{t} \rceil$. $\mathsf{WhiteKey}(m, t, E)$ is a $k = 128$ bits key generator defined by:

$$\mathsf{WhiteKey}(m, t, E) = \mathcal{H} \circ \lfloor E_{K^2}\text{-}\mathsf{CTR}(s_2) \rfloor_d \circ \lfloor E_{K^2}(s_2 + d) \rfloor_d \circ S_t \circ C_t \circ \lfloor E_{K^1}\text{-}\mathsf{CTR}(s_1) \rfloor_l$$

- E_{K^1}-CTR and E_{K^2}-CTR define the block cipher E instantiated with K^1 and K^2 respectively, used in counter mode. $\lfloor E_{k^1}\text{-}\mathsf{CTR}(s_1) \rfloor_l$ corresponds to the $l = \lceil t/8 \rceil$ first 128-bits ciphertexts obtained by encrypting s_1 with E_{K^1} in counter mode and in the same way, $\lfloor E_{K^2}(s_2 + d) \rfloor_d$ corresponds to the d first 128-bits ciphertexts of the encryption of $s_2 + d$ with E_{K^2} in counter mode.

- C_t is a function which applies a mapping $\mathsf{T} \colon \mathbb{F}_{2^m} \to \mathbb{F}_2^{128}$ to each of the m-bit blocks of the input:

$$\begin{array}{rcl} C_t \colon \mathbb{F}_2^{128l} & \to & \mathbb{F}_2^{128t} \\ (x_0, \ldots, x_{128l-1}) & \mapsto & (\mathsf{T}(a_0), \ldots, \mathsf{T}(a_{t-1})) \end{array}$$

  with $a_0 = x_0 \oplus x_1 X + \ldots + x_{m-1} X^{m-1} \mod P_m$, $a_1 = x_m \oplus x_{m+1} X + \ldots + x_{2m-1} X^{2m-1} \mod P_m$, etc. P_m is an irreducible polynomial of degree m used to construct the field $\mathbb{F}_{2^m}$.

- S_t transforms a $128t$-bit input into a $d \times d$ matrix over $\mathbb{F}_{2^{128}}$. If $t \neq d^2$, the matrix is completed with "0":

$$\begin{array}{rcl} S_t \colon \mathbb{F}_2^{128t} & \to & M_d(\mathbb{F}_{2^{128}}) \\ (x_0, \ldots, x_{128t-1}) & \mapsto & \begin{pmatrix} a_0 & a_1 & \ldots a_{d-1} \\ a_d & a_{d-1} & \ldots a_{2d-1} \\ \vdots & \vdots & \ddots \\ a_{t-1} & 0 & \ldots \quad 0 \end{pmatrix} \end{array}$$

- $\mathcal{H}$ is the following mapping:

$$\begin{array}{rcl} \mathcal{H} \colon \mathbb{F}_{2^{128}}^d \times \mathbb{F}_{2^{128}}^d \times \mathcal{M}_d(\mathbb{F}_{2^{128}}) & \to & \mathbb{F}_{2^{128}} \\ a, b, Q \mapsto \sum_{i=0}^{d-1} Q_{i,j} \otimes a_i \otimes b_j & & \end{array}$$

  $\mathcal{H}$ is an extractor introduced in [38] and enables to extract an almost uniform string with a seed of length equal to the square root of the input.

The definitions of ENC-TCOM, INC-COM and ENC-COM have been formalized with security games in [56]. We refer to the paper for details about these formalizations. To informally summarize the notions, a scheme satisfies (S, δ, λ) weak incompressibility if and only if any

Name	m	t	R	WB size	B	δ
WhiteBlock$(16, 4, 18, E)$	16	4	18	2 MB	512 kB	112
WhiteBlock$(20, 3, 23, E)$	20	3	23	$2^{4.6}$ MB	$2^{2.6}$ MB	108
WhiteBlock$(24, 2, 34, E)$	24	2	34	256 MB	64 MB	104
WhiteBlock$(28, 2, 34, E)$	28	2	34	4 GB	2 GB	100
WhiteBlock$(32, 2, 34, E)$	32	2	34	64 GB	16 GB	96

Table 4.9: Parameters for different instances of the WhiteBlock family: m is the table input length (in bits), t is the number of tables for one round, R is the number of rounds, WB size gives the total size T of the tables, B is the data known by the adversary such that $\frac{B}{T} = \frac{1}{4}$ and δ is the white-box security, i.e. the adversary is able to encrypt a portion $2^{-\delta}$ of plaintexts and any other plaintext can be encrypted with probability smaller than 2^{-128}.

adversary allowed to query up to S entries of the table can only correctly encrypt up to a proportion δ of plaintexts except with negligible probability $2^{-\lambda}$. Thus, (S, δ, λ)-ENC-TCOM is equivalent to (B, Z) weak space-hardness with $B = S \times m'$, where m' is the output length of the table, and $Z = -\log(\delta)$. Typically, $\lambda = 128$.

Tables 4.9 and 4.10 summarize the parameters of WhiteBlock and WhiteKey respectively. The minimum number of rounds required for WhiteBlock and the minimum number of table calls required for WhiteKey are calculated in the proofs for weak incompressibility.

Instantiations. Two instances of WhiteBlock have been proposed: PUPPYCIPHER and HOUND. Both rely on the AES-128 for the cipher E. Regarding WhiteKey, the instance is called COUREUR-DESBOIS. The block cipher used is also the AES-128 with a specific representation for the field $\mathbb{F}_{2^{128}} \approxeq \mathbb{F}_2[x]/(x^{128} + x^7 + x^2 + x + 1)$.

4.6 The White-box Even-Mansour family

Cho et al. presented the White-box Even Mansour (WEM) family [32] which is a family of white-box dedicated block ciphers following the Even-Mansour construction. Since, the Even-Mansour cipher is an iterated cipher constructed from public permutations and independent keys, in the WEM cipher, the key-addition is replaced by a layer of key-dependent S-boxes. Each S-box is pre-computed with the master key using a block cipher in counter mode, e.g. the AES. Thus, the unbreakability security of the cipher is obtained from the security of the underlying cipher. The goal of the authors was to design block ciphers with the same security levels as SPACE and WhiteBlock regarding space-hardness but with a better trade-off between performance and white-box security. A block cipher of the WEM family is a Substitution Permutation Network so is similar to an instance of the SPNbox family. However, the authors did not give any comparison to SPNbox. We will compare the two families regarding the security and the performance at the end of this section.

Name	**m**	**t**	**Table calls**	**WB size**	**B**	δ
$\mathsf{WhiteKey}(16, 8, E)$	16	8	57	2 MB	512 kB	112
$\mathsf{WhiteKey}(20, 6, E)$	20	6	55	16 MB	4 MB	108
$\mathsf{WhiteKey}(24, 5, E)$	24	5	53	256 MB	64 MB	104
$\mathsf{WhiteKey}(28, 4, E)$	28	4	51	4 GB	2 GB	100
$\mathsf{WhiteKey}(32, 4, E)$	32	4	49	64 GB	16 GB	96

Table 4.10: Parameters for different instances of the WhiteKey family: m is the table input length (in bits), t is the number of tables, WB size gives the total size T of the tables, B is the data known by the adversary such that $\frac{B}{T} = \frac{1}{4}$ and δ is the white-box security, i.e. the adversary is able to encrypt a portion $2^{-\delta}$ of plaintexts and any other plaintext can be encrypted with probability smaller than 2^{-128}.

Construction. Let n be the plaintext/ciphertext length in bits, m be the block length in bits and t be the number of block such that $n = m \times t$. For a k-bits master key, an instance $\mathsf{WEM}(n, m, R, E, d)$ of the WEM family consists of R rounds of the following transformations:

- A substitution operation which applies t parallel key-dependent S-boxes;
- A public permutation P^r which applies d rounds of an iterated block cipher E with a public key.

Any key-dependent S-box describes a random permutation over the range $[0, 2^m - 1]$. Since a random permutation over $[0, 2^m - 1]$ can be generated from a random sequence of bits by applying the Fisher-Yates Shuffle algorithm to the random bits, one needs to generate $2^m \times m$ random bits. Such random bits can be generated by encrypting random plaintexts with the master key and a block cipher in counter mode.

To summarize, an instance $\mathsf{WEM}(n, m, R, E, d)$ requires $t \times (R + 1)$ key-dependent S-boxes and R public permutations.

The $\mathsf{WEM}(128, 16, 2, AES, 5)$ instance. The security of WEM, regarding both unbreakability and space-hardness, tightly depends on the underlying block cipher E. As an instance, Cho et al. proposed to use the AES-128 as the underlying cipher both to generate the key-dependent S-boxes and for the public permutations P^1, P^2. Thus, for $1 \leqslant r \leqslant 2$, any public permutation P^r consists in $d = 5$ rounds of the AES instantiated with a "zero-key" such that the whole cipher is made of 10 rounds of the AES. Besides, the 24 distinct 16-bit S-boxes are generated by encrypting enough plaintexts with the AES-128 in counter mode: $\mathsf{m} = 0$ is first encrypted with $ctr = 0$, which generates 128 bits, then $\mathsf{m} = 1$ is encrypted with $ctr = 1$ which generates other 128 bits, etc.

Unbreakability. In the white-box model, the adversary has access to the full code-books of the S-boxes. Thus, they can reverse the Fisher-Yates Shuffle algorithm and get the ciphertexts

corresponding to the plaintexts $\mathsf{m}_0, \mathsf{m}_1, \ldots$. To generate $2^{16} \times 2^4 = 2^{20}$ random bits, $\frac{2^{20}}{128}$ ciphertexts are needed. So, 24×2^{13} ciphertexts are needed to generated the 24 S-boxes. An adversary who gets the code-books for all these S-boxes obtains 24×2^{13} plaintext-ciphertext pairs. This is much lower than the birthday bound of $2^{128/2}$ beyond which a ciphertext can be distinguished. Anyhow, no key recovery attack against the AES in counter mode is known so far. Consequently, $\mathsf{WEM}(128, 16, 2, AES, 5)$ is $\mathsf{UBK\text{-}ATK}$ secure as long as $\mathsf{AES\text{-}CTR}$ is secure for key recovery.

Space-hardness. The weak space-hardness of the WEM construction, as stated by Cho et al., is expressed by Theorem 1. The parameters given by the theorem actually correspond to the weak-space hardness under chosen-space attack defined in [21]. As pointed out in [21], the parameters for weak space-hardness under known-space attack and chosen-space attack are actually the same because of the random choice of the challenge plaintext.

Theorem 1. *Let $\mathsf{WEM}(n, m, R, E, d)$ be an instance of the WEM family. Let $t = \frac{n}{m}$ be the number of S-boxes in each round. Let $T = 2^m$ be the number of entries of each S-box and $S = 2^{-\alpha} \cdot T$ be the number of S-box entries known by the adversary. If $R - \frac{\log(R+1)}{\alpha} > \frac{m}{\alpha} + \frac{\log(t)}{\alpha}$*[12] *then $\mathsf{WEM}(n, m, R, E, d)$ is (weakly) $(2^{-\alpha} \cdot T, n - \log(T))$ space-hard.*

Proof. We assume that the adversary obtains a portion S of each S-box (the same portion for all S-boxes is obtained) by a priori choosing the entries. Thus, given a random plaintext P, the probability that they correctly encrypt P with the known entries is the probability that each entry lookup needed to encrypt P matches the known entries: since there are $t(R+1)$ S-boxes in total, this occurs with probability $\left(\frac{S}{T}\right)^{t(R+1)}$.

Now, assume that for a random plaintext P, the adversary only misses t entries, so the probability that they correctly guess the corresponding S-box outputs equals $(\frac{1}{2^m - 2^{m-\alpha}})^t < 2^{-t(m-1)}$. There are $\binom{t(R+1)}{t}$ possibilities for the missed t entries so the probability of success of the adversary equals $\left(\frac{S}{T}\right)^{t(R+1)-t} \cdot \binom{t(R+1)}{t}$.

We require that the probability of success of the adversary is less than 2^{-n}, i.e.

$$\begin{aligned}
\left(\frac{S}{T}\right)^{t(R+1)-t} \cdot \binom{t(R+1)}{t} &< 2^{-n} \\
2^{-\alpha t R} \cdot (t(R+1))^t &< 2^{-tm} \text{ since } \binom{t(R+1)}{t} < (t(R+1))^t \\
-\alpha t R + t\log(t) + t\log(R+1) &< -tm \\
-R + \frac{t\log(t)}{\alpha t} + \frac{t\log(R+1)}{\alpha t} &< -\frac{tm}{\alpha t} \\
R - \frac{\log(R+1)}{\alpha} &> \frac{m}{\alpha} + \frac{\log(t)}{\alpha}
\end{aligned}$$

As pointed out in [32], given T words of memory, the adversary can prepare T pairs of plaintext-ciphertext so the probability that a random plaintext belongs to the prepared set is $\frac{T}{2^n} = 2^{m-n} = 2^{\log(T)-n}$. Consequently, $Z = n - \log(T)$[13]. □

Table 4.11 gives the parameters for different instances of WEM which achieves $(2^{-\alpha} \cdot T, n - \log(T))$ weak space-hardness.

[12] Note that there is a mistake in the analysis of Cho et al. which lead to a different inequality: actually, the authors considered $8R$ S-boxes in their analysis instead of $8(R+1)$ S-boxes.

[13] Fouque et al. in [56] made the same point.

Name	m	t	n	WB size	α	R	Z
WEM$(128, 16, 12, E, 5)$	16	8	128	10.92 MB	2	12	112
WEM$(128, 24, 12, E, 5)$	24	5	120	2.64 GB	2.50	12	104
WEM$(128, 32, 12, E, 5)$	32	4	128	1.43 TB	3.14	12	96

Table 4.11: Parameters of WEM(n, m, R, E, d) such that it achieves $(2^{-\alpha} \cdot T, n - \log(T))$ weak space-hardness: T is the number of entries of any S-box, $Z = n - \log(T)$ is the security in bits, m is the block length (in bits), t is the number of blocks, n is the plaintext/ciphertext length (in bits) and R is the number of rounds.

Block size m	WEM		SPNbox	
	WB size T	WSH-CSA	WB size T	WSH-CSA
16	10.92 MB	$T \cdot 2^{-2}$	132 kB	$T \cdot 2^{-1.61}$
24	2.64 GB	$T \cdot 2^{-2.5}$	50.3 MB	$T \cdot 2^{-2.57}$
32	1.43 TB	$T \cdot 2^{-3.14}$	17.2 GB	$T \cdot 2^{-3.20}$

Table 4.12: Comparison of WEM and SPNbox in terms of WB size and weak space-hardness.

Comparison with SPNbox. WEM and SPNbox are both families of SPN block ciphers. The number of rounds for SPNbox is fixed to $R = 10$ while WEM allows for different numbers of round. The authors of [32] recommend a number of rounds $R = 12$. Regarding the white-box sizes, the instances of WEM require much more storage size than SPNbox (Table 4.12). Thus, SPNbox is more practical. Comparing the space-hardness, the 16-bit instance of WEM is a bit better than the 16-bit instance of SPNbox but for the other instances, SPNbox has a better space-hardness. With regard to the performance, the 16-bit instance of WEM requires 98.6 cycles per byte on an Intel core i7-5500 CPU at 2.40 GHz while the 16-bit instance of SPNbox requires 17.59 cycles per byte on an Intel core i7-6700 CPU at 3400 MHz. In the end, SPNbox is a best choice for both size and speed considerations.

Conclusion of the chapter

The first step towards a theoretical model of White-Box Cryptography was made by Saxena, Wyseur and Preneel in [108]. Their idea is to consider White-Box Cryptography as a special *obfuscation* where the *obfuscator* hides a particular functionality. They introduced the *white-box property* (WBP) of an obfuscator to capture the amount of useful information leaked from the obfuscation, i.e. an obfuscator satisfies the WBP if any adversary given a white-box access to an obfuscated program has no more advantage with respect to a security notion than any adversary given a black-box access to the original program. The key idea here is that the black-box security notion is actually transposed to the white-box setting. For instance, the authors considered the indistinguishability notions IND-CPA and IND-CCA2. The main disadvantage of their work is that it only highlights how existing security notions might be satisfied in the white-box setting but says nothing about how the confidentiality of the key is satisfied. The approach of Delerablée et al. is more pragmatic since it formalizes the security notions that are expected from White-Box Cryptography when considering real-world applications. Even if the notions defined in [43] are quite clear, the following works tended to redefine some of the notions, namely the incompressibility and the one-wayness notions. Thus, they were independently named weak white-box security and strong white-box security in [14]. In the same work, the authors already referred to the notion of memory-hardness. A memory-hard function is a function that costs a significant amount of memory to evaluate. The ASASA-based block cipher is memory-hard in the sense that it makes call to several lookup tables that occupy a significant amount of memory. Besides, an attacker cannot find an equivalent representation of the block cipher which requires less memory. From this emerged the notion of space-hardness which is refined into weak and space-hardness. Following the same reasoning as for memory-hardness, a weak space-hard block cipher is a function which requires a significant space to store data in order to evaluate correctly.

Even if the terms used to express the security of a block cipher in the white-box model vary, the techniques used to construct a white-box secure block cipher seem to be the same. Especially, the unbreakability security aka key extraction security is guaranteed by the construction of one or several lookup tables which embed all key bits. If the primitive used to construct the tables is known to be secure then the overall block cipher is also secure in the sense of unbreakability. Since the lookup tables generation is the main step of a white-box compiler, it is obvious that this step should be as efficient as possible.

Part II

From theory to practice: on the need of White-Box Cryptography

1

PUF-based cryptography

"Each problem that I solved became a rule, which served afterwards to solve other problems."

René Descartes

Contents

We have seen in the first part of this thesis that the key extraction security formalized by the notion of unbreakability is satisfied by a block cipher as long as the S-box (for SPN ciphers) or the round-function (for Feistel ciphers) are derived from the master key using a well-studied block cipher like the AES implemented by a lookup table. Nevertheless, this technique only mitigates a code lifting attack when the attacker has only a temporary access to the implementation or is limited in the size of the data they can transmit. For all constructions of Chapter 4, if the attacker is able to copy the entire white-box code, they can take advantage of the code functionality without any restriction. In this chapter, we consider this critical attack scenario for mobile applications and propose to bind a white-box code implementing an encryption to the device. This chapter aims to answer to the following question: *"how can we enforce the correct execution of an encryption only on a genuine mobile device?"*

1.1 Introduction

The main issue that white-box implementations suffer is code lifting. In practice, a code lifting attack relies on copying the entire code implementing a cryptographic algorithm and executing it to exploit the functionality (e.g. a decryption). This attack is similar to the attack that relies on calling a library which contains a white-box implementation to exploit the decryption functionality without any reverse engineering effort. A code lifting is sometimes observed against applications with Digital Rights Management enforcement.

The first approach to mitigate code lifting is the use of external encodings to encode the input and the output of the cryptographic algorithm [36]. Thus, the annihilating encodings should be included at other places in the application and hidden through code obfuscation to prevent an attacker from recovering the annihilating encodings. However, in light of [1] an encoded white-box implementation is still vulnerable to a Differential Fault Analysis.

Another approach relies on associating the cryptographic operation to a secure component with a method called *node locking*. The method described in [88] is to insert some predefined values to the white-box implementation, for instance the license of a DRM application such that the modification of the license leads to a modification of the white-box implementation making the decryption useless. Thus, the method enables a form of software tamper-resistance.

Our approach is somehow a node locking where we use a Physically Unclonable Function (PUF) to bind the white-box implementation to its execution environment. The goal is to prevent an attacker who has lifted the entire white-box code from taking advantage of the code functionality. A PUF might help for this because it is an unclonable hardware function based on process variation during fabrication. Since a PUF is device-specific, it has been widely studied for device identification, authentication, and key generation. This opens the study of a PUF-based cryptography which bases the security of a cryptosystem on the key generated by the PUF which is never stored on non-volatile memory. When the PUF is fully integrated in the cryptographic primitive itself, i.e. a secret key is not generated for further use but the cryptographic primitive directly exploits the behavior of the PUF, the primitive is said hardware-entangled (see Section 1.4).

We propose a construction which is between a PUF-based key generation and hardware-entangled cryptography.

We introduce the *PUF-based encryption scheme* as an encryption scheme which takes three inputs: a secret key, a plaintext and a PUF response. Thus, the PUF response adds a variability mechanism to the encryption scheme. When a PUF-based encryption scheme is provided with a white-box compiler, we talk about a *PUF-based white-box encryption scheme*. We define the

notion of *lockability*[14] to measure the security of the encryption scheme when implemented in the white-box model: lockability states the difficulty of executing a lifted white-box code outside of the dedicated environment. We propose an instantiation of a PUF-based encryption scheme using the construction of a tweakable block cipher of [37] and the block cipher SPNbox-8 [21] which satisfies unbreakability.

The encryptions of two plaintexts with the same key but two different PUF responses, are indistinguishable from the encryptions of two plaintexts with two different keys. Since the PUF-based encryption scheme will be implemented in the white-box model, we define two attack scenarios. First, we assume that the attacker has access to the PUF instance for limited period of time, their goal is to forge a valid challenge-response pair. Second, we assume that the attacker has access to the PUF for an unlimited period of time, actually the attacker is only computationally limited. Their goal is to clone the PUF instance. We define the notions of lockability under forging attack in the first scenario and lockability under cloning attack in the second scenario. Our work shows that a PUF-based encryption scheme is secure regarding both notions as far as the underlying PUF is secure in the sense of unforgeability and unclonability.

In this chapter we describe the following contributions:

- **PUF-based encryption scheme.** We define a PUF-based encryption scheme which is an encryption scheme over three spaces: the key space, the message space and a PUF range space. When a PUF-based encryption scheme is provided with a white-box compiler, we talk about a PUF-based white-box encryption scheme. We define the notion of lockability to measure the security of the encryption scheme when implemented in the white-box model: lockability states the difficulty of executing a lifted white-box code outside of the dedicated environment.
- **Proof for lockability.** Since the encryption scheme is tightly binded to a PUF, we show that lockability is achieved if the PUF is unforgeable or unclonable.
- **Instantiation.** We propose an instantiation of a PUF-based encryption scheme using the Even Mansour construction of a tweakable block cipher. Especially, we construct a family of almost uniform and XOR universal hash functions which we instantiate with SPNbox-8.
- **Image sensor-based PUF.** We instantiate the PUF component with a PUF characterized from the image sensor of a smartphone.

1.2 Preliminaries

1.2.1 Universal Hash Functions

Definition 46 (Keyed universal hash functions)**.** *Let $\mathcal{K} = \mathbb{F}_2^n$ be a set of keys and let $\mathcal{H} = \{H_K \colon (\mathbb{F}_2^n)^l \to \mathbb{F}_2^n\}_{K \in \mathcal{K}}$ be a family of keyed hash functions indexed by the set of keys $\mathcal{K}$.*

1. *$\mathcal{H}$ is universal if for any $x \neq x' \in (\mathbb{F}_2^n)^l$:*

$$\Pr_K \left[H_K(x) = H_K(x') \right] \leqslant 2^{-n}.$$

2. *$\mathcal{H}$ is ρ-universal (almost universal) if for any $x \neq x' \in (\mathbb{F}_2^n)^l$:*

$$\Pr_K \left[H_K(x) = H_K(x') \right] \leqslant \rho.$$

[14] Neologism to define the possibility to lock a cipher to an execution environment.

3. $\mathcal{H}$ *is* ρ*-XOR universal (almost XOR universal) if for any* $x \neq x' \in (\mathbb{F}_2^n)^l$ *and any* $b \in \mathbb{F}_2^n$*:*

$$\Pr_K\left[H_K(x) \oplus H_K(x') = b\right] \leqslant \rho.$$

Definition 47 (Uniformity of hash functions)**.** *Let* $\mathcal{H} = \{H_K \colon (\mathbb{F}_2^n)^l \to \mathbb{F}_2^n\}_{K \in \mathcal{K}}$ *be a family of keyed hash functions indexed by a set of keys* $\mathcal{K} = \mathbb{F}_2^n$*.*

1. $\mathcal{H}$ *is uniform if for any* $x \in (\mathbb{F}_2^n)^l$ *and for any* $y \in \mathbb{F}_2^n$*:*

$$\Pr_K[H_K(x) = y] = 2^{-n}.$$

2. $\mathcal{H}$ *is* ρ*-uniform (almost uniform) if for any* $x \in (\mathbb{F}_2^n)^l$ *and for any* $y \in \mathbb{F}_2^n$*:*

$$\Pr_K[H_K(x) = y] \leqslant \rho.$$

1.2.2 Tweakable block cipher

A tweakable block cipher is a cryptographic primitive introduced and formalized by Liskov et al. [83]. It differs from a traditional block cipher as it takes an additional input tk, called a tweak from a space $\mathcal{T}$. The tweak enables a variability in the cipher. A tweakable block cipher is, thus, generally defined as $\widetilde{E} \colon \mathcal{K} \times \mathcal{T} \times \mathsf{M} \to \mathsf{C}$. The applications of tweakable block ciphers in cryptography are various, for instance message authentication codes or authenticated encryption modes. There are three approaches for constructing a tweakable block cipher. The first one is to use a conventional block cipher to include the tweak. The security proofs then result from the security of the underlying block cipher seen as a black box. The second approach is to propose ad-hoc designs. The last method to construct a tweakable block cipher is to use the Feistel scheme to provide a provably secure construction. In this section, we present some constructions of the first approach.

Definition 48 (Tweakable block cipher)**.** *[83]*

A tweakable block cipher $\widetilde{E}$ *is defined as a function:*

$$\widetilde{E} \colon \mathbb{F}_2^k \times \mathbb{F}_2^t \times \mathbb{F}_2^n \to \mathbb{F}_2^n$$

which takes as input a key $K \in \mathbb{F}_2^k$*, a tweak* $tk \in \mathbb{F}_2^t$ *and a plaintext* $\mathsf{m} \in \mathbb{F}_2^n$ *and outputs a ciphertext* $\mathsf{c} \in \mathbb{F}_2^n$*.*

For any key $K \in \mathbb{F}_2^k$ *and any* $tk \in \mathbb{F}_2^t$*,* $\widetilde{E}_K(tk, \cdot)$ *is a permutation of* $\mathbb{F}_2^n$*.*

Liskov et al. tweakable block ciphers

Definition 49 (LRW1)**.** *Let* $E : \mathbb{F}_2^k \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ *be a block cipher. For any* $K \in \mathbb{F}_2^k$*, the mapping* $\widetilde{E}_K \colon \mathbb{F}_2^t \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ *such that for any* $tk \in \mathbb{F}_2^t$ *and any* $\mathsf{m} \in \mathbb{F}_2^n$*,* $\widetilde{E}_K(tk, \mathsf{m}) = (E_K(tk) \oplus E_K(\mathsf{m}))$ *is a tweakable block cipher.*

LRW1 is the first construction of strong tweakable block cipher proposed by Liskov et al. Authors showed that if the advantage of an adversary $\mathcal{A}$, running in time $\tau_{\mathcal{A}}$ and making q queries, to distinguish E in the random permutation model is $\mathsf{Adv}^{\mathrm{cca}}_{\mathcal{A},E}(q, \tau_{\mathcal{A}})$ then the advantage of the same adversary to distinguish $\widetilde{E}$ is lower than $\mathsf{Adv}^{\mathrm{cca}}_{\mathcal{A},E}(q, \tau_{\mathcal{A}}) + \Theta\left(q^2/2^n\right)$. Hence, $\widetilde{E}$ achieves the birthday-bound security of $2^{n/2}$ under Chosen-Ciphertext Attack. The main disadvantage of this construction is its running time which is twice of the running time of E.

The following construction was proposed by the same authors and aims to minimize the overall cost.

Definition 50 (LRW2). *Let $E : \mathbb{F}_2^k \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a block cipher. Let $\mathcal{H} = \{h \colon \mathbb{F}_2^t \to \mathbb{F}_2^n\}$ be a family of almost XOR universal hash functions. The mapping $\widetilde{E}_{K,h} \colon \mathbb{F}_2^t \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ such that for any $tk \in \mathbb{F}_2^n$ and any $\mathsf{m} \in \mathbb{F}_2^n$, $\widetilde{E}_{K,h}(tk, \mathsf{m}) = E_K(\mathsf{m} \oplus h(tk)) \oplus h(tk)$ is a tweakable block cipher.*

For this construction only one execution of the underlying block cipher is done but the length of the key is multiplied by 2. However, this construction is more efficient and achieves a security of $2^{n/2}$ as the previous one. Namely, the advantage of an adversary to distinguish $\widetilde{E}$ is lower than $\mathsf{Adv}^{\mathrm{cca}}_{\mathcal{A},E}(q, \tau_{\mathcal{A}}) + 3\rho q^2$, where ρ is the bound of the XOR universality.

The following constructions are chained version of LRW2, i.e. LRW2 with 2 rounds and a generalization CLRW with r rounds.

Definition 51 (2-LRW2). *Let $E : \mathbb{F}_2^k \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a block cipher. Let $\mathcal{H} = \{h \colon \mathbb{F}_2^t \to \mathbb{F}_2^n\}$ be a family of almost XOR universal hash functions. A 2-round LRW2 construction is a tweakable block cipher $\widetilde{E} \colon \mathbb{F}_2^k \times \mathbb{F}_2^t \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ defined for any $\mathsf{m} \in \mathbb{F}_2^n$ and any $tk \in \mathbb{F}_2^t$ as follows:*

$$\widetilde{E}_K(tk, \mathsf{m}) = E_{K_2}(E_{K_1}(\mathsf{m} \oplus h(tk)) \oplus h(tk) \oplus h'(tk)) \oplus h'(tk)$$

where $K = (K_1, K_2, h, h')$ and $h, h' \in \mathcal{H}$.

This construction requires a key of length $4n$ bits and achieves a beyond birthday-bound security of $2^{2n/3}$. The r-round CLRW (Chained LRW) construction is generalized as follows:

Definition 52 (r-CLRW). *Let $E : \mathbb{F}_2^k \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a block cipher. Let $\mathcal{H} = \{h \colon \mathbb{F}_2^t \to \mathbb{F}_2^n\}$ be a family of almost XOR universal hash functions. A r-round CLRW construction is a tweakable block cipher $\widetilde{E} \colon \mathbb{F}_2^k \times \mathbb{F}_2^t \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ defined for any $\mathsf{m} \in \mathbb{F}_2^n$ and any $tk \in \mathbb{F}_2^t$ as follows:*

$$\widetilde{E}_K(tk, \mathsf{m}) = F^r \circ F^{r-1} \circ \ldots \circ F^1(\mathsf{m})$$

where $K = (K_1, \ldots, K_r, h_1, \ldots, h_r)$, $F^r(x) = E_{K_r}(x \oplus h_r(tk)) \oplus h_r(tk)$, and $h_1, \ldots, h_r \in \mathcal{H}$.

The security proof of the generalized CLRW construction has been investigated in [79]. Authors proved that the r-round chained CLRW is secure up to $2^{\frac{rn}{r+2}}$ adversarial queries since the advantage of an adversary to distinguish $\widetilde{E}$ is lower than $r \cdot \mathsf{Adv}^{\mathrm{cca}}_{\mathcal{A},E}(q, \tau_{\mathcal{A}} + rq\tau_E) + \frac{4\sqrt{2}}{\sqrt{r+2}} q^{(r+2)/4} (2\rho)^{r/4}$.

Even if the r-round CLRW construction achieves a beyond birthday-bound security, it clearly looses efficiency because of the r calls to the block cipher E and the key length which is equal to $2rn$ bits.

Tweakable Even-Mansour construction

The tweakable Even-Mansour of Cogliati et al. [37] is constructed following the Even-Mansour construction [52], i.e. the construction of a block cipher from a set of keys and public permutations. The tweakable Even-Mansour achieves a beyond birthday-bound security with at least two rounds while the security of a r-round tweakable block cipher asymptotically tends towards 2^n.

Definition 53 (1-TEM). *A 1-round tweakable Even-Mansour cipher constructed from a public permutation P and a key K is a mapping $\widetilde{E}_K \colon \mathbb{F}_2^t \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ defined as:*

$$\widetilde{E}_K(tk, \mathsf{m}) = h_K(tk) \oplus P(h_K(tk) \oplus \mathsf{m})$$

where h_K is uniform and almost XOR universal.

Similarly, we define the 2-round and generalized r-round constructions.

Definition 54 (2-TEM). *[37] Let $\mathcal{H} = \{h\colon \mathbb{F}_2^t \to \mathbb{F}_2^n\}$ be a family of almost XOR universal hash functions. A 2-round tweakable Even-Mansour cipher constructed from the set of public permutations $P = \{P_1, P_2\}$ and the set of keys $K = \{K_1, K_2\}$ is a mapping $\widetilde{E}_K\colon \mathbb{F}_2^t \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ defined for any $m \in \mathbb{F}_2^n$ and any $tk \in \mathbb{F}_2^t$ as:*

$$\widetilde{E}_K(tk, \mathsf{m}) = F_{tk}^2 \circ F_{tk}^1(\mathsf{m})$$

where $F_{tk}^i(x) = h_{K_i}(tk) \oplus P_i(h_{K_i}(tk) \oplus x)$ for any $i \in \{1, 2\}$ and $h_{K_i} \in \mathcal{H}$.

Definition 55 (r-TEM). *Let $\mathcal{H} = \{h\colon \mathbb{F}_2^t \to \mathbb{F}_2^n\}$ be a family of almost XOR universal hash functions. A r-round tweakable Even-Mansour cipher constructed from the set $P = \{P_1, \ldots, P_r\}$ of public permutations is a mapping $\widetilde{E}_K\colon \mathbb{F}_2^t \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ defined for any $m \in \mathbb{F}_2^n$ and any $tk \in \mathbb{F}_2^t$ as:*

$$\widetilde{E}_K(tk, \mathsf{m}) = F_{tk}^r \circ \ldots \circ F_{tk}^1(\mathsf{m})$$

where $F_{tk}^i(x) = h_{K_i}(tk) \oplus P_i(h_{K_i}(tk) \oplus x)$ for any $i \in \{1, \ldots, r\}$ and $h_{K_i} \in \mathcal{H}$.

The security proofs for 1-TEM and 2-TEM are demonstrated in [37]. Using the H-coefficients technique of Patarin, Cogliati et al. proved that the advantage of an adversary to distinguish 1-TEM from a random tweakable permutation under adaptive chosen-ciphertext attack is upper bounded by $q_c^2\rho + \frac{1q_cq_p}{2^n}$ where q_c is the number of queries to the construction oracle, q_p is the number of queries to each inner permutation oracle and ρ is the upper-bound on the XOR universality of $\mathcal{H}$. The advantage of a distinguisher for 2-TEM under adaptive chosen-ciphertext attack is lower than $\frac{29\sqrt{q_cq_p}}{2^n} + \rho\sqrt{q_cq_p} + 4\rho q_c^{3/2} + \frac{30q_c^{3/2}}{2^n}$.

1.3 Formalization of a Physically Unclonable Function

1.3.1 PUF terminology and measures

A Physically Unclonable Function (PUF) is a physical entity (e.g. an integrated circuit) that presents random imperfections in its structure. The imperfections result from the manufacturing process and are expected to be instance specific and unclonable. By extension, the term PUF also refers to the "expression" of these random imperfections. To characterize a PUF, the physical disorder (e.g. a current or a voltage) is generally measured under a specific condition (stimuli) and converted into a digitized form. Thus, a PUF performs a probabilistic functional operation: given a certain input (a stimuli, also called a *challenge*) it produces a measurable output (the *response*). Consequently, a PUF can be seen as a physical realization of a probabilistic mapping $\mathsf{puf}\colon \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X}$ is the domain space called the challenge set and $\mathcal{Y}$ is an output range called the response set of puf. The creation of a PUF is expressed by invoking a manufacturing process on a set of parameters param. We write $\mathsf{puf} \leftarrow\!\!\$\ \textsc{Create}(\mathsf{param})$ to signify that a PUF instance is created. Create is a probabilistic procedure that represents in practice the physical manufacturing process. The set of all instances created according to a set of parameters is called a PUF class and denoted by $\mathcal{P}$. A PUF instance can be evaluated on a challenge $x \in \mathcal{X}$ using an evaluation procedure Eval: we write $y \leftarrow \textsc{Eval}(\mathsf{puf}, x)$ or simply $y \leftarrow \mathsf{puf}(x)$ for the evaluation of puf on the challenge x. Due to the manufacturing process, the outputs of a PUF instance are generally noisy, i.e., an instance puf outputs two different responses y, y' when evaluated twice on the same challenge x. Thus, a PUF response is considered as a random variable, and information

about the distribution of the PUF responses is obtained through experiment. An experiment on a PUF class is parametrized by t, q_c, and q_r which are the number of instance evaluations on the same challenge, the number of created instances, and the number of evaluations on an instance, respectively. The experiment gives estimated descriptive statistics of the *intra-distance* δ_1, of the *instance inter-distance* δ_2 and of the PUF *inter-distance* δ_3. The *intra-distance* of a PUF class is the distance between two distinct responses of an instance evaluated on the same challenge. In the same manner, the *instance inter-distance* and the PUF *inter-distance* characterize the distance between any two responses y, y' of a PUF instance puf evaluated on two different challenges x, x' and the distance between two outputs y, y' of two distinct instances puf, $\mathsf{puf'}$ evaluated on the same challenge x, respectively. On the one hand, the intra-distance property estimates the *reproducibility* of a PUF output. Thus, low intra-distance ensures that the PUF outputs are highly reproducible. On the other hand, the PUF inter-distance estimates the *uniqueness* of an instance's output while the instance inter-distance ensures the uniqueness of a response given a challenge. We require that the intra-distance δ_1 is lower than δ_2 and δ_3 to be able to uniquely distinguish outputs from different instances and outputs from different challenges. For the rest of the paper, we assume that the PUF responses are in a binary form so we consider the Hamming distance HD.

Let $\epsilon\colon \mathbb{N} \to \mathbb{R}$ be a negligible function, i.e. for any polynomial function $\mathsf{poly}\colon \mathbb{N} \to \mathbb{N}$ there exists n_0 such that $\epsilon(n) < 1/\mathsf{poly}(n)$ for any $n > n_0$. In the following λ is a security parameter.

Definition 56 (Intra-distance)**.** *A PUF class $\mathcal{P}$ has an intra-distance equal to δ_1 if for any created PUF instance $\mathsf{puf} \leftarrow \textsc{Create}(\mathsf{param})$ evaluated t times on the same challenge $x \in \mathcal{X}$ it holds that*

$$\Pr[\max(\{\textsc{hd}(y_i, y_j)\}_{i \neq j}) \leqslant \delta_1 \mid x \in \mathcal{X},\ \{y_i \leftarrow \mathsf{puf}(x)\}_{1 \leqslant i \leqslant t}] = 1 - \epsilon(\lambda). \tag{1.1}$$

Definition 57 (Instance inter-distance)**.** *A PUF class $\mathcal{P}$ has an instance inter-distance equal to δ_2 if for any created PUF instance $\mathsf{puf} \leftarrow \textsc{Create}(\mathsf{param})$ it holds that*

$$\Pr[\min(\{\textsc{hd}(y_i, y_j)\}_{i \neq j}) \geqslant \delta_2 \mid x_i \in \mathcal{X},\ \{y_i \leftarrow \mathsf{puf}(x_i)\}_{1 \leqslant i \leqslant q_r}] = 1 - \epsilon(\lambda).$$

Definition 58 (PUF inter-distance)**.** *A PUF class $\mathcal{P}$ has an inter-distance equal to δ_3 if for any created instances $\mathsf{puf}_i \leftarrow \textsc{Create}(\mathsf{param})$ for $i \leqslant q_c$ and any challenge $x \in \mathcal{X}$ it holds that*

$$\Pr[\min(\{\textsc{hd}(y_i, y_j)\}_{i \neq j}) \geqslant \delta_3 \mid x \in \mathcal{X},\ \{y_i \leftarrow \mathsf{puf}_i(x)\}_{1 \leqslant i \leqslant q_c}] = 1 - \epsilon(\lambda).$$

Another property of PUF outputs that should be estimated is the *min-entropy*. For instance, the min-entropy of a binary string represents the number of uniform bits and measures the uncertainty one has on the output. The *conditional min-entropy* is more relevant in the case of PUF output distribution since an adversary should not deduce enough information about an unknown output given many outputs. The conditional min-entropy $\mathrm{H}_\infty(X|Y)$ of two random variables X and Y is defined as $-\log \max_{x,y} \Pr[X = x \mid Y = y]$. Then, the average min-entropy is defined as follows.

Definition 59 (Average min-entropy)**.** *For two random variables X, Z, the average min-entropy of X conditioned on Y is defined as:*

$$\tilde{\mathrm{H}}_\infty(X|Y) = -\log \mathbb{E}_{y \leftarrow Y}\left[\max_{x,y} \Pr[X = x \mid Y = y]\right]. \tag{1.2}$$

Definition 60 (Average min-entropy of PUF outputs [4]). *A PUF class $\mathcal{P}$ has an average min-entropy equal to δ_4 if for any created instance $\mathsf{puf}_i \leftarrow \textsc{Create}(\mathsf{param})$ it holds that, with $1 \leqslant i \leqslant q_c$ and $1 \leqslant j \leqslant q_r$,*

$$\Pr\left[\tilde{\mathrm{H}}_\infty(y_{i,j}|Y_{i,j}) \geqslant \delta_4 \middle| x_j \in \mathcal{X},\ Y = \{y_{i,j} \leftarrow \mathsf{puf}_i(x_j)\},\ Y_{i,j} = Y \setminus \{y_{i,j}\}\right] = 1 - \epsilon(\lambda).$$

These properties characterize the output distribution of PUF instances and give an idea of the reliability of a PUF class. A PUF class may verify some other properties, for instance *unforgeability*[15], *unclonability*, *one-wayness* and *tamper-evidence*. All possibly properties are investigated by Maes in [86]. Namely, he identified the necessary properties of a class of physical entities that show a challenge-response behaviour to be called a PUF class. We focus on the unclonability and unforgeability properties and borrow the formalization from [4][16].

Weak PUF vs Strong PUF. There are two important subtypes of PUF according to the PUF application scenario: *weak* PUF or *strong* PUF. A weak PUF is a PUF with few challenge-response pairs and sometimes only one fixed challenge. The responses are generally used to derive a secret key which is used by the embedded system as the secret input for hardware implementation of cryptographic algorithms. Thus, a weak PUF enables to store a secret key on a device and in a sense is similar to a non-volatile key storage. The main advantage of the weak PUF is that the key cannot be extracted from a memory as in the case of classical non-volatile memory such as EEPROM. On the contrary, a strong PUF has a large number of challenge-response pairs, ideally exponential in the length of the challenge. Thus, determining the complete challenge-response pairs should be computationally hard. Furthermore, it should be difficult for an attacker to predict the response of a random challenge and to deduce from previous known challenge-response pairs the response of a random challenge. Strong PUFs are typically used for system identification or authentication to prevent replay attacks.

1.3.2 Security notions

We use the formal definitions of unforgeability and unclonability that have been introduced in the paper of Armknecht et al. [4]. As in their paper, the notions are formalized with security games.

We denote by $\mathcal{A}$ a Probabilistic Polynomial Time (PPT) adversary. If λ is the security parameter then the running time of $\mathcal{A}$ is polynomial in λ. The following unforgeability game states the advantage of an adversary $\mathcal{A}$ to guess one output of a PUF without having access to the device. We assume that the challenger has access to the manufacturing process and thus can invoke the creation procedure on a set of parameters param. The adversary can only choose a set of parameters to obtain new instances from an oracle. Thus, they can get instances of one PUF class or instances of different PUF classes depending on the chosen set of parameters. We assume that there exists a recovery procedure that enables one to recover a noise-free response from noisy ones. Thus, the adversary is not required to produce the correct response since a noisy response can be corrected as long as the distance between the noisy response and the correct one is lower than the intra-distance.

[15] also called unpredictability.

[16] Authors used the term unforgeability instead of unpredictability as the goal of the adversary is to forge an output y' close enough to a valid output y given a challenge x to a PUF instance. Thus, it is not required that the adversary predicts the correct output y. Consequently, we adopt the same terminology.

Set up. The challenger selects a manufacturing process and initial parameters param. The challenger sends $(1^\lambda, \mathsf{param})$ to the adversary $\mathcal{A}$ and initializes two counters c_0, c_1.

Learning. $\mathcal{A}$ adaptively issues two types of oracle query: a creation query to create a new instance under some chosen parameters and a response query to get the output of an instance of their choice. $\mathcal{A}$ is allowed to issue at most q_c creation queries and q_r response queries.

- When $\mathcal{A}$ issues a creation query with param', the challenger creates $\mathsf{puf}_{c_0} \leftarrow \textsc{Create}(\mathsf{param})$ if $\mathsf{param}' = \mathsf{param}$ and increments c_0, or creates $\mathsf{puf}'_{c_1} \leftarrow \textsc{Create}(\mathsf{param}')$ if $\mathsf{param}' \neq \mathsf{param}$ but is a valid creation parameter and increments c_1. Otherwise the challenger responds $\perp$.
- When $\mathcal{A}$ issues a response query with (b, i, x_j), the challenger sends $y_{i,j} \leftarrow \mathsf{puf}_i(x_j)$ if $b = 0$ and $i \leqslant c_0$ or sends $y'_{i,j} \leftarrow \mathsf{puf}'_i(x_j)$ if $b = 1$ and $i \leqslant c_1$. Otherwise the challenger returns $\perp$.

Guess. $\mathcal{A}$ outputs (i^*, x^*, y^*).

The index i^* denotes the instance chosen by adversary such that y^* is probably the result of the evaluation of puf_{i^*} on x^*. We denote by $\mathsf{EUF\text{-}CICA}$ the *Existential Unforgeability under Chosen Instance and Challenge Attack*. The security of a PUF according to this security notion is given by the following definition.

Definition 61 ($\mathsf{EUF\text{-}CICA}$)**.** *Let* $\mathsf{Adv}^{\mathsf{euf\text{-}cica}}_{\mathcal{A},\mathcal{P}}(\lambda, \delta_1)$ *be the advantage of an adversary* $\mathcal{A}$ *playing the unforgeability game. We have*

$$\mathsf{Adv}^{\mathsf{euf\text{-}cica}}_{\mathcal{A},\mathcal{P}}(\lambda, \delta_1) = \Pr\left[\textsc{hd}(y^*, \mathsf{puf}_{i^*}(x^*)) \leqslant \delta_1 | x^* \in \mathcal{X} \setminus \mathcal{X}_{i^*}\right] - \frac{|B|}{|\mathcal{Y}|},$$

where puf_{i^*} *is a PUF instance that has been created by the challenger in the learning phase,* $\mathcal{X}_{i^*}$ *is the set of challenges that have been issued for the instance* i^* *and* $B = \{y \mid y_{i^*} \leftarrow \mathsf{puf}_{i^*}(x^*)$ *and* $\textsc{hd}(y_{i^*}, y) \leqslant \delta_1\}$ *is the set of outputs* y *that are at a distance at most* δ_1 *from* $\mathsf{puf}_{i^*}(x^*)$*.*

A PUF provides $(q_c, q_r, \delta_1, \epsilon)$*-*$\mathsf{EUF\text{-}CICA}$ *security if for any PPT adversary* $\mathcal{A}$ *we have*

$$\Pr\left[\mathsf{Adv}^{\mathrm{euf-cica}}_{\mathcal{A},\mathcal{P}}(\lambda, \delta_1) > 0\right] \leqslant \epsilon(\lambda).$$

The adversary wins the game if their guess y^* belongs to the set B with probability higher than randomly picking an element of B.

A stronger notion is unclonability which expresses the hardness of constructing two PUF instances that show the same input-output behavior. We describe an unclonability game in the same way as the unforgeability game:

Set up. The setup phase is the same as in the unforgeability game.

Learning. $\mathcal{A}$ issues oracle queries as in the learning phase of the unforgeability game.

Guess. $\mathcal{A}$ outputs (i^*, b, j^*).

The output (i^*, b, j^*) is interpreted as follows: i^* gives the index of a PUF instance created under the paramater param. If $b = 0$ then j^* is the index of a PUF instance created under the parameter param; otherwise j^* is the index of a PUF instance created under the parameter param'. That means that the adversary exhibits two clones either in the same PUF class or in two different classes. In other words, the unclonability should garantee that it is both hard to construct two clones using the same set of parameters and to find a set of parameters that enables the construction of clones.

We denote by UUC-CICA the *Universal Unclonability* security defined by the following.

Definition 62 (UUC-CICA)**.** *Let* $\mathsf{Adv}^{\mathrm{uuc-cica}}_{\mathcal{A},\mathcal{P}}(\lambda, \delta_1)$ *be the advantage of an adversary* $\mathcal{A}$ *playing the unclonability game. Let* puf_{i^*} *and* puf'_{j^*} *be the PUF instances which correspond to the output* (i^*, b, j^*) *of* $\mathcal{A}$*. We have*

$$\mathsf{Adv}^{\mathrm{uuc-cica}}_{\mathcal{A},\mathcal{P}}(\lambda, \delta_1) = \Pr\left[\forall x \in \mathcal{X}, \mathrm{HD}(\mathsf{puf}_{i^*}(x), \mathsf{puf}'_{j^*}(x)) \leqslant \delta_1\right].$$

A PUF provides $(q_c, q_r, \delta_1, \epsilon)$*-*UUC-CICA *if for any PPT adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{\mathrm{uuc-cica}}_{\mathcal{A},\mathcal{P}}(\lambda, \delta_1) \leqslant \epsilon(\lambda).$$

This notion of unclonability may be weakened to the notions of *target unclonability* where the adversary only issues creation queries with the parameter param and *restricted unclonability* where the adversary sends at most $q_r < |\mathcal{X}|$ response queries. In any case, the definition of the advantage of the corresponding adversary is the same. Moreover, restricted unclonability is equivalent to EUF-CICA security (Theorem 1 of [4]).

1.3.3 Construction of a fuzzy extractor for PUFs

A Physically Unclonable Function generally shows a fuzzy random behavior, i.e., it outputs a noisy and non-uniform response to a given challenge. While reproducibility of PUF responses is an obvious requirement, uniformity is an interesting property especially for key generation. To enable reproducibility and uniformity of the PUF responses, a *fuzzy extractor* is used. The concept of fuzzy extractor was introduced by Dodis et al. [49] to extract randomness from fuzzy sources. A simple way to construct a fuzzy extractor is to combine a *secure sketch* (which could be considered as an error correcting code) and a *strong extractor* (a function that extracts uniformly distributed random bits from a source with a given min-entropy). A secure sketch is a public side information that allows one to recover a string y from any noisy but close enough y'. There are different constructions of a secure sketch in the literature, most of them considering the Hamming distance, the set difference or the edit difference as particular metrics. In this chapter, for instance, we describe the code-offset and the syndrome constructions for the Hamming distance metric (see [49] for other constructions). Since the purpose of a secure sketch is to correct errors, it is typically based on a linear error-correcting code C over a space $\mathcal{F}^L$ of dimension k and minimal distance $2t + 1$[17]. If $\mathcal{F} = \mathbb{F}_2$, then C is a binary linear code which contains 2^k codewords of length L such that any $c \neq c' \in C$ differ in at least $2t + 1$ bits. C is characterized by a $k \times L$ matrix G called the generating matrix where the k rows form a basis of C as a vectorial space and by a $(L - k) \times L$ parity check matrix H satisfying $G \cdot H^T = 0$.

Definition 63 (Secure sketch)**.** *A* $(\mathcal{Y}, \mu, \mu', t)$*-secure sketch is a randomized map* $\mathsf{SS}\colon \mathcal{Y} \to \{0,1\}^*$ *such that:*

[17] The code enables to correct up to t errors.

1. $\mathrm{H}_\infty(y) \geqslant \mu$ *for all* $y \in \mathcal{Y}$.

2. *there exists a deterministic recovery function* Rec *satisfying for all* $y, y' \in \mathcal{Y}$*:*

$$\textit{if } \mathrm{HD}(y, y') \leqslant t \textit{ then } \mathsf{Rec}(y', \mathsf{SS}(y)) = y.$$

3. $\tilde{\mathrm{H}}_\infty(y|\mathsf{SS}(y)) \geqslant \mu'$.

$\mathrm{HD}(y, y')$ *is the distance between* y *and* y' *regarding the Hamming distance metric and* $\tilde{\mathrm{H}}_\infty(y|\mathsf{SS}(y))$ *is the average min-entropy of* y *knowing* $\mathsf{SS}(y)$ *(Definition 1.2).*

Even if the secure sketch leaks some information about y, the string should retain enough entropy. The entropy loss of a secure sketch is $\mu - \mu'$.

Proposition 7 (Secure-sketch from code-offset)**.** *Let* C *be a binary linear code of length* L *and dimension* k*. Let* $\mathsf{SS}\colon \mathbb{F}_2^L \to \mathbb{F}_2^L$ *be the mapping defined for any* $y \in \mathbb{F}_2^L$ *as:*

$$\mathsf{SS}(y) = y \oplus c$$

with c *uniformly picked in* C*.*

Let Rec *be a recovery procedure associated to* SS *and defined as follows:*

$$\mathsf{Rec}(y', \mathsf{SS}(y)) = \mathsf{SS}(y) \oplus C(D(y' \oplus \mathsf{SS}(y)))$$

where $C(\cdot)$ *and* $D(\cdot)$ *denote the encoding and decoding functions of the code.*

SS *is a* $(\mathbb{F}_2^L, \mu, \mu', t)$*-secure sketch with* $\mu' = \mu - (L - k)$*.*

Proposition 8 (Secure-sketch from syndrome)**.** *Let* C *be a binary linear code of length* L *and dimension* k *with a parity check matrix* H*. Let* $\mathsf{SS}\colon \mathbb{F}_2^L \to \mathbb{F}_2^L$ *be a mapping defined for any* $y \in \mathbb{F}_2^L$ *as:*

$$\mathsf{SS}(y) = y \cdot H^T$$

Let Rec *be a recovery procedure associated to* SS *and defined as follows:*

$$\mathsf{Rec}(y', \mathsf{SS}(y)) = y' \oplus e$$

with e *the unique word satisfying* $\mathsf{wt}(e) \leqslant t$ *and* $e \cdot H^T = y' \cdot H^T \oplus \mathsf{SS}(y)$*.*

SS *is a* $(\mathbb{F}_2^L, \mu, \mu', t)$*-secure sketch with* $\mu' = \mu - (L - k)$*.*

A secure-sketch constructed from a polar code is defined in Section 1.6 of this chapter.

A strong (μ, ϵ)-extractor is a function $\mathsf{Ext}\colon \mathcal{Y} \times \{0,1\}^n \to \{0,1\}^v$ such that for all Y, Z such that Y is distributed over $\mathcal{Y}$ and $\tilde{\mathrm{H}}_\infty(Y|Z) \geqslant \mu$, we get $\mathsf{SD}((\mathsf{Ext}(Y, S), (S, Z)); (U_v, (S, Z))) \leqslant \epsilon$ where SD is the statistical distance and U_v is a uniformly distributed variable of length v. S is called the seed of Ext of length n and $\mu - v$ is called the entropy loss of Ext.

Definition 64 (Strong Extractor [6])**.** *An efficient function* $\mathsf{Ext}\colon \mathcal{Y} \times \{0,1\}^n \to \{0,1\}^v$ *is a strong* (μ, ϵ)*-extractor, if for all* Y, Z *such that* Y *is distributed over* $\mathcal{Y}$ *and* $\tilde{\mathrm{H}}_\infty(Y|Z) \geqslant \mu$*, it hold that*

$$\mathsf{SD}((\mathsf{Ext}(Y, S), (S, Z)); (U_v, (S, Z))) \leqslant \epsilon$$

where $S \equiv U_n$ *is called the seed of* Ext *of length* n *and* $\mu - v$ *is called the entropy loss of* Ext*.*

A strong extractor enables to extract at most $v = \mu - 2\log(1/\epsilon) + O(1)$ uniformly distributed random bits from a source with a certain min-entropy μ. The entropy loss is at least $2\log(1/\epsilon) + O(1)$. Actually, the optimal entropy loss of $2\log(1/\epsilon)$ is attained with universal hash functions.

Our construction of a family of almost universal hash functions is actually used as a strong extractor applied to a PUF response (Section 1.5.3).

Because PUF response are noisy, we need a special case of strong extractor called fuzzy extractor which removes the noise before extracting uniformly random string.

A fuzzy extractor on input y first generates a helper string ω (called helper data) which will be made public and an extracted string R with a probabilistic procedure Gen. The helper string is used in a probabilistic reproduction procedure Rep to generate a uniform string R given a noisy y' close to y.

Definition 65 (Fuzzy extractor)**.** *A (μ, v, t, ϵ)-fuzzy extractor consists of two efficient procedures* (Gen, Rep) *such that*

- Gen *is a probabilistic procedure which on input $y \in \mathcal{Y}$ outputs a helper string $\omega \in \{0,1\}^*$ and an extracted string $R \in \{0,1\}^v$;*
- Rep *is a reproduction procedure which takes a string $y' \in \mathcal{Y}$ and a string $\omega \in \{0,1\}^*$ and outputs $R' \in \{0,1\}^v$.*

For any distribution Y of min-entropy μ, the fuzzy extractor should satisfy the correctness and security properties as follows:

1. *Correctness: If $(R, \omega) \leftarrow \text{Gen}(y)$ and $\text{hd}(y, y') \leqslant t$ then $\text{Rep}(y', \omega) = R$.*
2. *Security: If $(R, \omega) \leftarrow \text{Gen}(y)$ then $\mathsf{SD}((R, \omega), (U_v, \omega) \leqslant \epsilon$.*

A fuzzy extractor can be constructed from a secure sketch and a family $\mathcal{H}$ of universal hash functions as follows: let $H_s \leftarrow\!\!\$\, \mathcal{H}$ uniformly sampled with a seed s; define Gen as, for any $y \in \mathcal{Y}$, $\text{Gen}(y) = (R, \omega)$ with $R = H_s(y)$ and $\omega = (\mathsf{SS}(y), s)$ and define Rep as, for any y' such that the distance between y and y' is smaller than t, then $\text{Rep}(y') = H_s(\text{Rec}(y', \mathsf{SS}(y)))$.

Consequently, a fuzzy extractor for the PUF instance described in Section 1.6 can be defined from the polar code secure-sketch and the family of universal hash functions of Section 1.5.3.

1.3.4 Examples of PUFs

So far many instantiations of PUF exist. We describe here some existing PUFs in the literature. Digital intrinsic PUFs are PUFs embedded on an integrated circuit. For example, the basic arbiter PUF of [80] is a delay-based intrinsic PUF which is based on a digital race condition on two paths on a silicon chip. The paths end with an arbiter circuit which determines the winner of the race and outputs a binary value accordingly. Another example of delay-based PUF is the ring oscillator PUF of [58] which is based on the random variations on the frequencies of digital oscillating circuits. Another type of intrinsic PUF is the memory-based PUF, e.g. the SRAM (Static Random Access Memory) PUF [58]. This PUF exploits the transient behavior of an SRAM cell when it is powered up.

Recently, new kinds of PUFs available on mobile devices appeared. Examples of such PUFs is the camPUF [74] or PRNU (Photo Response Non Uniformity)-based PUF [114] which are based on the random variations of the image sensors of the mobile device, the microphone PUF [64] which exploits the silicon variability of the microphone sensor or the MEMS (Micro Electro Mechanical Systems)-based gyroscope PUF [115].

1.4 Hardware-entangled cryptography

Armknecht et al. [3] introduced a memory leakage-resilient encryption scheme using Physically Unclonable Functions. They defined a new cryptographic primitive that they called PUF-PRF as it is a Pseudo-Random Function based on a PUF. The PUF-PRF is used to generate a secret such that there is no need to digitally store a key anymore and so memory leakage is avoided. They constructed a provably secure block cipher based on a PUF-PRF using the Luby-Rackoff construction[18] and proved that the block cipher remains secure even if some information leaks[19]. In particular, the PUF-PRF provides a tamper-resilience property since any attempt to access the internals of the device will cause a change of the PUF behavior. Thus, the PUF-PRF solution is relevant for software protection or device encryption. The authors proposed to use the SRAM to instantiate the PUF-PRF.

1.5 PUF-based white-box encryption scheme

Our contribution starts with the definition of a PUF-based encryption scheme.

1.5.1 PUF-based encryption scheme

A PUF-based encryption scheme is an encryption scheme which depends on an instance puf of a PUF class. More specifically, the encryption and the decryption algorithms take three arguments, the additional argument being the PUF output.

Definition 66 (PUF-based encryption scheme (PE))**.** *Let $\mathcal{P} = \{\mathit{puf}\colon \mathcal{X} \to \mathcal{Y}\}$ be a PUF class. A PUF-based encryption scheme $\mathcal{PE}$ consists of polynomial time algorithms* $(\mathsf{KGen}, \mathrm{EVAL}, \mathsf{Enc}, \mathsf{Dec})$ *such that*

- KGen *is a probabilistic algorithm which outputs a secret key given a security parameter:*

$$K \leftarrow\!\!\$\, \mathsf{KGen}(1^\lambda)$$

- Eval *is a probabilistic algorithm which evaluates a PUF instance on a challenge x and outputs a response y:*

$$y \leftarrow\!\!\$\, \mathrm{EVAL}(\mathit{puf}, x)$$

- Enc *is a probabilistic algorithm which outputs a ciphertext $c \in C$ given a message $m \in M$, a secret key K and a PUF response y:*

$$c \leftarrow\!\!\$\, \mathsf{Enc}(K, m, y)$$

- Dec *is a deterministic algorithm which outputs a message $m \in M$ given a ciphertext $c \in C$, a secret key K and a PUF response y:*

$$m \leftarrow \mathsf{Dec}(K, c, y)$$

[18] The Luby-Rackoff construction enables to construct a pseudo-random permutation from pseudo-random functions.

[19] Authors defined a PUF fuzzy extractor which is composed of a generation procedure and a reconstruction procedure. A helper data generated by the generation procedure is computed from a noise-free PUF response and is used by the reconstruction procedure. Thus, the helper data inevitably leaks information.

A PUF-based encryption scheme satisfies correctness for any instance puf *if:*

$$\forall x \in \mathcal{X},\ y \leftarrow \text{EVAL}(\mathsf{puf}, x) \implies \forall m \in M, \Pr[\mathsf{Dec}(K, \mathsf{Enc}(K, m, y), y) = m] = 1.$$

Definition 67 (White-box compiler)**.** *Let* $\mathcal{PE} = (\mathsf{KGen}, \text{EVAL}, \mathsf{Enc}, \mathsf{Dec})$ *be a PUF-based encryption scheme. A white-box compiler for* $\mathcal{PE}$ *is a probabilistic algorithm* $C_{\mathcal{PE}}$ *that outputs a white-box program* P *for a fixed key* K *and a random nonce* r*:*

$$\mathsf{P} \leftarrow\!\!\$\, C_{\mathcal{PE}}(K, r)$$

- *If* P *implements* Enc_K *then for any instance* puf*:*

$$\forall x \in \mathcal{X},\ y \leftarrow \text{EVAL}(\mathsf{puf}, x) \implies \forall m \in M,\ \Pr[\mathsf{P}(m, y) = \mathsf{Enc}(K, m, y)] = 1$$

- *If* P *implements* Dec_K *then for any instance* puf*:*

$$\forall x \in \mathcal{X},\ y \leftarrow \text{EVAL}(\mathsf{puf}, x) \implies \forall c \in C,\ \Pr[\mathsf{P}(c, y) = \mathsf{Dec}(K, c, y)] = 1$$

The PUF-based encryption scheme and the white-box compiler form a PUF-based white-box encryption scheme (PWE).

1.5.2 Security models

We consider the white-box setting and state the security of a PWE. We recall the unbreakability notion of [43] and define the notion of lockability to ensure that a code-lifting attack will fail. We consider a white-box implementation of the encryption algorithm, i.e. the program P outputted by the white-box compiler implements Enc_K for a fixed key K. The adversary $\mathcal{A}$ is divided into two adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ who share some state information. Actually, $\mathcal{A}_2$ is run on $\mathcal{A}_1$'s output. $\mathcal{A}_1$ is allowed to make at most q_c creation queries and q_r response queries while $\mathcal{A}_2$ is allowed to make at most q decryption queries to a decryption oracle.

Unbreakability.

In the unbreakability game, we only consider the adversary $\mathcal{A}_2$ whose goal is to recover the secret key K. We consider the Chosen-Plaintext Attack (CPA) where the adversary chooses q plaintexts (and the input y) and encrypts them with the program P and the Chosen-Ciphertext Attack (CCA) where they are allowed to query a decryption oracle with q chosen ciphertexts (and input y) in addition to q chosen-plaintexts. The advantage of the adversary $\mathcal{A}_2$ in the $\mathsf{UBK\text{-}ATK}$ security game is the probability that $\mathcal{A}_2$ outputs the key K. As in [43], the white-box encryption scheme is secure in the sense of $\mathsf{UBK\text{-}ATK}$ for $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA}\}$, if the advantage of any PPT adversary is negligible. Unbreakability guarantees that a secret key cannot be recovered from a white-box implementation: the size of the white-box implementation cannot be reduced to the size of the key.

Unbreakability guarantees that a secret key cannot be recovered from a white-box implementation: the size of the white-box implementation cannot be reduced to the size of the key. Moreover, even if full break is not possible, the adversary can lift the code and decrypt without knowing the key. We define the notion of lockability to capture the difficulty of executing a lifted white-box code without a PUF instance.

Lockability.

We assume that the adversary has access to the white-box program P. Their goal is to correctly encrypt a random plaintext given a random challenge x. Thus, the adversary $\mathcal{A}_1$ plays an EUF-CICA or an UUC-CICA game while the attack model for the adversary $\mathcal{A}_2$ is either CPA or CCA.

We define two security notions, depending on the attack model used against the PUF: either a forging attack or a cloning attack.

We write LCK-FORGE for the lockability security under CPA or CCA which is described by the following security game:

Set up. The challenger selects a manufacturing process and initial parameters param. They generate $K \leftarrow\!\!\$\, \mathsf{KGen}(1^\lambda)$ uniformly at random and compile $\mathsf{P} \leftarrow\!\!\$\, C_{\mathcal{PE}}(K, r)$. They send $(1^\lambda, \mathsf{param}, \mathsf{P})$ to the adversary and initializes two counters c_0, c_1.

Learning 1. $\mathcal{A}_1$ issues creation queries and response queries. As in the unforgeability game, the challenger creates $\mathsf{puf}_{c_0} \leftarrow \mathsf{Create}(\mathsf{param})$ or $\mathsf{puf'}_{c_1} \leftarrow \mathsf{Create}(\mathsf{param'})$ when receiving creation queries and sends $y_{i,j} \leftarrow \mathsf{puf}_i(x_j)$ or $y_{i,j} \leftarrow \mathsf{puf}'_i(x_j)$ when receiving response queries.

$\mathcal{A}_1$ outputs (i^*, x^*, y^*) and sends (i^*, x^*) to the challenger. The challenger evaluates $y \leftarrow \mathsf{puf}_{i^*}(x^*)$.

Learning 2. $\mathcal{A}_2$ chooses q plaintexts and encrypts them with P and y^*. In the CCA model, $\mathcal{A}_2$ issues decryption queries with q chosen ciphertexts c_j and gets $\mathsf{m}_j \leftarrow \mathsf{Dec}(K, \mathsf{c}_j, y)$.

Challenge. The challenger draws $\mathsf{m} \leftarrow\!\!\$\, \mathsf{M} \backslash \{\mathsf{m}_j\}$ uniformly at random, computes $\mathsf{c} \leftarrow \mathsf{Enc}(K, \mathsf{m}, y)$ and sends m.

Guess. $\mathcal{A}_2$ outputs c^*.

Definition 68 (LCK-FORGE)**.** *Let $\mathcal{P}$ be a PUF class with an intra-distance δ_1. Let $\mathcal{PE}$ be the PUF-based white-box encryption scheme. Let $\mathsf{Adv}^{\mathrm{lck-forge}}_{\mathcal{A},\mathcal{PE}}(\lambda, \delta_1)$ be the advantage of an adversary playing the game above. We have*

$$\mathsf{Adv}^{\mathrm{lck-forge}}_{\mathcal{A},\mathcal{PE}}(\lambda, \delta_1) = \Pr\left[\mathsf{c}^* = \mathsf{c}\colon r = (i^*, x^*, y^*) \leftarrow \mathcal{A}_1(\mathsf{param}), \mathsf{c}^* \leftarrow \mathcal{A}_2^{\mathcal{O}}(r, \mathsf{P})\right]$$

where $\mathcal{O}$ denotes the decryption oracle in the CCA model. A PUF-based white-box encryption scheme $\mathcal{PE}$ satisfies (q_c, q_r, q, ϵ')-LCK-FORGE security if for any PPT adversary $\mathcal{A}$ it holds that:

$$\mathsf{Adv}^{\mathrm{lck-forge}}_{\mathcal{A},\mathcal{PE}}(\lambda, \delta_1) \leqslant \epsilon'(\lambda).$$

In the above security game, the adversary is required to output a PUF challenge x^* for which they can forge a valid output y^* for an instance puf_{i^*}. The second learning phase allows the adversary to interact with a decryption oracle (in the CCA model). If the attack model for $\mathcal{A}_2$ is the CPA model, then in Learning 2, $\mathcal{A}_2$ is only allowed to choose at most q plaintexts and encrypts them with P and y^*. For both models, the adversary wins if $\mathcal{A}_1$ is able to forge a valid pair (x^*, y^*) for a chosen instance i^* or if the pair returned by $\mathcal{A}_1$ is not valid but $\mathcal{A}_2$ guesses C.

Remark 4. *In the CCA model, in the learning 2, $\mathcal{A}_2$ can compute $\mathsf{c}_j \leftarrow \mathsf{P}(\mathsf{m}_j, y^*)$ and send c_j to the decryption oracle. Thus, $\mathcal{A}_2$ is able to verify if the guess of $\mathcal{A}_1$ is correct. Otherwise, $\mathcal{A}_1$ is allowed to restart the learning 1 and makes a new guess. This way the attack model is adaptive.*

However, the total number of queries made by $\mathcal{A}_1$ is still bounded by q_r and q_c and the total number of queries made by $\mathcal{A}_2$ is bounded by q. Consequently, the advantage of the adversary regarding LCK-FORGE is the same if the model is adaptive or not.

Theorem 2. *If the PUF class $\mathcal{P}$ provides $(q_c, q_r, \delta_1, \epsilon)$-EUF-CICA security then $\mathcal{PE}$ satisfies (q_c, q_r, q, ϵ')-LCK-FORGE security with*

- $\epsilon' \colon \lambda \mapsto \left(1 - \frac{1}{|\mathsf{C}|}\right) \cdot \epsilon(\lambda) + \frac{1}{|\mathsf{C}|}$ *for CPA.*
- $\epsilon' \colon \lambda \mapsto \left(1 - \frac{1}{|\mathsf{C}|-q}\right) \cdot \epsilon(\lambda) + \frac{1}{|\mathsf{C}|-q}$ *for CCA.*

Proof. Let E be the event "$\mathsf{Adv}^{\mathrm{euf-cica}}_{\mathcal{A}_1,\mathcal{P}}(\lambda, \delta_1) > 0$".

$$\mathsf{Adv}^{\mathrm{lck-forge}}_{\mathcal{A},\mathcal{PE}}(\lambda, \delta_1) = \Pr[\mathsf{c}^* = \mathsf{c} \mid E\,] \times \Pr[E] + \Pr\left[\mathsf{c}^* = \mathsf{c} \;\middle|\; \overline{E}\,\right] \times \Pr\left[\overline{E}\right].$$

Since $\mathcal{P}$ provides $(q_c, q_r, \delta_1, \epsilon)$-EUF-CICA security then $\Pr[E] \leqslant \epsilon(\lambda)$ and $\Pr[\mathsf{c}^* = \mathsf{c} \mid E\,] = 1$. This reflects the fact that once the PUF response is guessed correctly, the adversary inevitably finds the correct ciphertext. Besides, $\Pr\left[\mathsf{c}^* = \mathsf{c} \;\middle|\; \overline{E}\,\right] = \frac{1}{|\mathsf{C}|}$ in the CPA model and $\Pr\left[\mathsf{c}^* = \mathsf{c} \;\middle|\; \overline{E}\,\right] = \frac{1}{|\mathsf{C}|-q}$ in the CCA model. Hence,

$$\mathsf{Adv}^{\mathrm{lck-forge}}_{\mathcal{A},\mathcal{PE}}(\lambda, \delta_1) \leqslant \left(1 - \frac{1}{|\mathsf{C}|}\right) \cdot \epsilon(\lambda) + \frac{1}{|\mathsf{C}|}, \text{ under CPA, and}$$

$$\mathsf{Adv}^{\mathrm{lck-forge}}_{\mathcal{A},\mathcal{PE}}(\lambda, \delta_1) \leqslant \left(1 - \frac{1}{|\mathsf{C}| - q}\right) \cdot \epsilon(\lambda) + \frac{1}{|\mathsf{C}| - q}, \text{ under CCA.}$$

□

The notion of lockability under cloning attack LCK-CLONE expresses the situation in which we verify the lockability except if the adversary exhibits two distinct instances that have the same input-output behavior. We assume an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. The security game for LCK-CLONE under CPA or CCA is described as follows:

Set up. The set up is the same as in LCK-FORGE.

Learning 1. The learning phase is as in LCK-FORGE. $\mathcal{A}_1$ outputs (i^*, b, j^*).

Learning 2. $\mathcal{A}_2$ chooses q plaintexts and challenges and encrypts them with P. In the CCA model, in addition $\mathcal{A}_2$ issues decryption queries with q chosen ciphertexts c_j and challenges x_j and gets $\mathsf{m}_j \leftarrow \mathsf{Dec}(K, \mathsf{c}_j, y_j)$ where $y_j \leftarrow \mathsf{puf}_{i^*}(x_j)$.

Challenge. The challenger picks $\mathsf{m} \leftarrow\!\!\$\, \mathsf{M} \setminus \{\mathsf{m}_j\}$ and $x \leftarrow\!\!\$\, \mathcal{X} \setminus \{x_j\}$ uniformly at random, computes $C \leftarrow \mathsf{Enc}(K, \mathsf{m}, y)$ where $y \leftarrow \mathsf{puf}_{i^*}(x)$ and sends (m, x).

Guess. $\mathcal{A}_2$ outputs c^*.

Definition 69. *Let $\mathcal{P}$ be a PUF class with an intra-distance δ_1. Let $\mathcal{PE}$ be the PUF-based white-box encryption scheme. Let $\mathsf{Adv}^{\mathrm{lck-clone}}_{\mathcal{A},\mathcal{PE}}(\lambda, \delta_1)$ be the advantage of an adversary playing the game above. We have*

$$\mathsf{Adv}^{\mathrm{lck-clone}}_{\mathcal{A},\mathcal{PE}}(\lambda, \delta_1) = \Pr\left[\mathsf{c}^* = \mathsf{c} \colon r = (i^*, b, j^*) \leftarrow \mathcal{A}_1(\mathsf{param}), \mathsf{c}^* \leftarrow \mathcal{A}_2^{\mathcal{O}}(r, \mathsf{P})\right]$$

where $\mathcal{O}$ denotes the decryption oracle in the CCA model. A PUF-based white-box encryption scheme satisfies $(q_c, q_r, q, \delta_1, \epsilon')$-LCK-CLONE security if for any PPT adversary $\mathcal{A}$ it holds that:

$$\mathsf{Adv}^{\mathrm{lck-clone}}_{\mathcal{A},\mathcal{PE}}(\lambda, \delta_1) \leqslant \epsilon'(\lambda).$$

Theorem 3. *If the PUF class $\mathcal{P}$ provides $(q_r, q_c, \delta_1, \epsilon)$-UUC-CICA security then the PUF-based white-box encryption scheme satisfies $(q_c, q_r, q, \delta_1, \epsilon')$-LCK-CLONE security with*

- $\epsilon' \colon \lambda \mapsto \left(1 - \frac{1}{|\mathsf{C}|}\right) \cdot \epsilon(\lambda)^{1/(|\mathcal{X}|-q)} + \frac{1}{|\mathsf{C}|}$ *for CPA.*
- $\epsilon' \colon \lambda \mapsto \left(1 - \frac{1}{|\mathsf{C}|-q}\right) \cdot \epsilon(\lambda)^{1/(|\mathcal{X}|-q)} + \frac{1}{|\mathsf{C}|-q}$ *for CCA.*

Proof. Let E be the event "$\mathrm{HD}(\mathsf{puf}_{i^*}(x), \mathsf{puf'}_{j^*}(x)) \leqslant \delta_1$".

$$\mathsf{Adv}^{\mathrm{lck-clone}}_{\mathcal{A},\mathcal{PE}}(\lambda, \delta_1) = \Pr[\mathsf{c}^* = \mathsf{c} \mid E] \times \Pr[E] + \Pr\left[\mathsf{c}^* = \mathsf{c} \,\middle|\, \overline{E}\right] \times \Pr\left[\overline{E}\right].$$

Since, $\mathcal{P}$ provides $(q_r, q_c, \delta_1, \epsilon)$-UUC-CICA security then $\Pr[E] \leqslant \epsilon(\lambda)^{1/(|\mathcal{X}|-q)}$ and $\Pr[\mathsf{c}^* = \mathsf{c} \mid E] = 1$. $\Pr\left[\mathsf{c}^* = \mathsf{c} \,\middle|\, \overline{E}\right] = \frac{1}{|\mathsf{C}|}$ in the CPA model and $\Pr\left[\mathsf{c}^* = \mathsf{c} \,\middle|\, \overline{E}\right] = \frac{1}{|\mathsf{C}|-q}$ in the CCA model. Hence,

$$\mathsf{Adv}^{\mathrm{lck-clone}}_{\mathcal{A},\mathcal{PE}}(\lambda, \delta_1) \leqslant \left(1 - \frac{1}{|\mathsf{C}|}\right) \cdot \epsilon(\lambda)^{1/(|\mathcal{X}|-q)} + \frac{1}{|\mathsf{C}|} \text{ ,under CPA, and}$$

$$\mathsf{Adv}^{\mathrm{lck-clone}}_{\mathcal{A},\mathcal{PE}}(\lambda, \delta_1) \leqslant \left(1 - \frac{1}{|\mathsf{C}|-q}\right) \cdot \epsilon(\lambda)^{1/(|\mathcal{X}|-q)} + \frac{1}{|\mathsf{C}|-q} \text{ ,under CCA.}$$

□

1.5.3 Construction of a PUF-based block cipher

The PUF-based encryption scheme of Section 1.5.1 can be instantiated with a tweakable block cipher. The idea is to define a cipher that can be composed with a PUF instance to define a PUF-based encryption. A tweakable block cipher is a cipher that takes three inputs: the plaintext, the secret key and a public value called the tweak. We refer to [83] for more details. Thus, a tweakable block cipher matches the definition of the cipher needed by the encryption scheme of Definition 66. We use the 2 rounds Tweakable Even-Mansour (2-TEM) [37] which is secure up to approximately $2^{2n/3}$ adversary's queries in the random permutation model. This construction applies an almost XOR universal (AXU) hash function to the tweak value. Since the tweak inputs are noise-free PUF responses, they are obtained after the recovery procedure of a secure sketch (see Definition 63). Thereby, the hash function takes a noise-free but non-uniform input y and transforms it to a nearly uniform value xored with the plaintext.

PUF-based encryption scheme with 2 rounds of TEM. Let $\mathcal{P} = \{\mathsf{puf} \colon \mathcal{X} \to \mathcal{Y}\}$ be a PUF class. Let $\mathcal{H} = \{H_K \colon \mathcal{Y} \to \mathbb{F}_2^n\}_{K \in \mathcal{K}}$ be a family of ρ_1-uniform and ρ_2-AXU hash functions indexed by a set of keys $\mathcal{K}$. Let $\mathsf{Enc} \colon \mathcal{K} \times \mathcal{Y} \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a 2-TEM block cipher (see Figure 1.1). For any key $K = (K_1, K_2)$, H_{K_1}, H_{K_2} are two hash functions of the family $\mathcal{H}$ and P_1, P_2

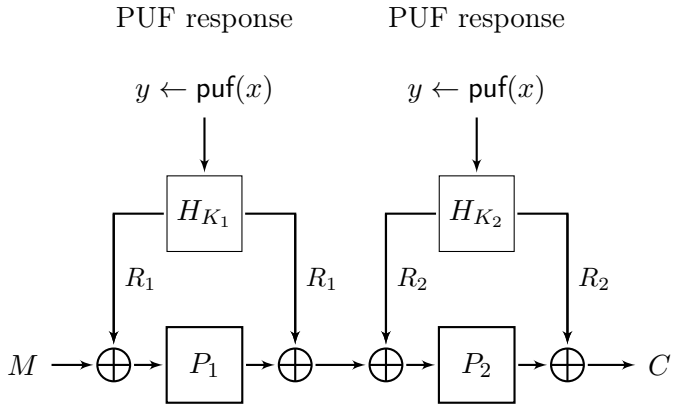


Figure 1.1: $\mathsf{TEM}(\mathcal{P})$ construction with 2 rounds and PUF responses as tweak inputs.

are two public permutations. For any plaintext $\mathsf{m} \in \mathbb{F}_2^n$, and PUF response $y \in \mathcal{Y}$ corresponding to a challenge x, the encryption function is defined as

$$\mathsf{Enc}(K, \mathsf{m}, y) = P_2(P_1(\mathsf{m} \oplus H_{K_1}(y)) \oplus H_{K_1}(y) \oplus H_{K_2}(y)) \oplus H_{K_2}(y).$$

In the same way, the decryption function is defined as

$$\mathsf{Dec}(K, \mathsf{c}, y) = P_1^{-1}(P_2^{-1}(\mathsf{c} \oplus H_{K_2}(y)) \oplus H_{K_2}(y) \oplus H_{K_1}(y)) \oplus H_{K_1}(y).$$

We denote by 2-$\mathsf{TEM}(\mathcal{P}) = (\mathbb{F}_2^{2n}, \mathbb{F}_2^n, \mathbb{F}_2^n, \mathsf{Enc}, \mathsf{Dec}, \mathcal{P})$ the constructed PE.

Keyed hash functions. We construct the following family of keyed hash functions:

Definition 70. *Let $E\colon \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a block cipher. Let $\mathcal{H} = \{H_K\colon (\mathbb{F}_2^n)^2 \to \mathbb{F}_2^n\}_{K \in \mathbb{F}_2^n}$ be a family of hash functions indexed by a set of keys, where the hash function H_K is defined as follows: for any $x = (x_1, x_2) \in (\mathbb{F}_2^n)^2$, $H_K(x) = E_K(x_1) \oplus E_{K^2}(x_2)$ where $K^2 = K \otimes K$ is the multiplication in the finite field $\mathbb{F}_{2^n}$.*

Remark 5. *The function $K \mapsto K^2$ is a permutation in $\mathbb{F}_{2^n}$, thus E_{K^2} is a permutation.*

We show that the above family of hash functions is actually almost uniform and XOR universal. First, we recall the definitions of differential characteristic probability and expected differential characteristic probability that will be used.

Definition 71. *Let $E\colon \mathbb{F}_2^n \times\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ be an iterative block cipher with r rounds and let $F_{K_i}^i$ be the i-th round function with a round key K_i derived from a master key K.*

- *A differential characteristic is a sequence of $r+1$ differences $(\Delta_0, \ldots, \Delta_r)$ where Δ_0 is the difference between two inputs and Δ_r is the output difference.*
- *For a key K, the fixed-key differential characteristic probability π_K is the probability for a pair of inputs to follow a given differential characteristic:*

$$\pi_K = \Pr_X \left[F_{K_i}^i(X) \oplus F_{K_i}^i(X \oplus \Delta_0) = \Delta_i, 1 \leqslant i \leqslant r \right].$$

- *The expected differential characteristic probability π is the differential characteristic probability averaged over all round keys:*

$$\pi = \Pr_{X,K}\left[F^i_{K_i}(X) \oplus F^i_{K_i}(X \oplus \Delta_0) = \Delta_i, 1 \leqslant i \leqslant r\right] = \mathbb{E}_K\left[\pi_K\right].$$

In the following, Π denotes an upper bound on the expected differential characteristic probability of the block cipher E introduced in Definition 70.

Theorem 4. *The family of Definition 70 is $\frac{34}{2^n}$-uniform and Π-AXU.*

Proof. We first prove the XOR universality. Let $x \neq x' \in (\mathbb{F}_2^n)^2$ and $b \in \mathbb{F}_2^n$. Without loss of generality, we assume that $x_1 \neq x'_1$. Otherwise, one simply interchanges the subscripts 1 and 2 in the following. We have

$$\begin{aligned}
& \Pr_K\left[H_K(x) \oplus H_K(x') = b\right] = \frac{|\{K \in \mathbb{F}_2^n \colon H_K(x) \oplus H_K(x') = b\}|}{2^n} \\
& \text{where } |\{K \in \mathbb{F}_2^n \colon H_K(x) \oplus H_K(x') = b\}| \\
= \; & |\{K, K^2 \in \mathbb{F}_2^n \colon E_K(x_1) \oplus E_{K^2}(x_2) \oplus E_K(x'_1) \oplus E_{K^2}(x'_2) = b\}| \\
= \; & \sum_{K^2 \in \mathbb{F}_2^n} |\{K \in \mathbb{F}_2^n \colon E_K(x_1) \oplus E_K(x'_1) = b \oplus E_{K^2}(x_2) \oplus E_{K^2}(x'_2)\}|
\end{aligned}$$

Since $|\{K \in \mathbb{F}_2^n \colon E_K(x_1) \oplus E_K(x_1 \oplus a) = b'| \leqslant \Pi \cdot 2^n$ for any $b' \in \mathbb{F}_2^n$ and $K \mapsto K^2$ is a bijection then $|\{K \in \mathbb{F}_2^n \colon H_K(x) \oplus H_K(x') = b\}| \leqslant \Pi \cdot 2^n$.

Now, we prove that the family $\mathcal{H}$ is $\frac{34}{2^n}$-uniform. Let $x \in (\mathbb{F}_2^n)^2$ and $y \in \mathbb{F}_2^n$.

We compute the cardinality of $A = \{K \in \mathcal{K} \colon H_K(x) = E_K(x_1) \oplus E_{K^2}(x_2) = y\}$ given $x = (x_1, x_2)$ and y. As the probability of picking randomly a key is equal to $1/2^n$, it means that the number of keys verifying properties of A follows a Poisson distribution with parameter $\lambda = 1$ as it could be considered as a sum of Bernoulli events with a very low probability. Thus, it matches with the modeling of the law of rare events. For $n = 128$, we obtain an upper bound equal to 34 for the cardinal of A directly applying the density of the Poisson law. Thus, $\Pr_K[H_K(x) = y] \leqslant \frac{34}{2^n}$. □

Remark 6 (Estimation of the probability when K is fixed.)**.** *In the white-box setting, the key K is fixed. Consequently, we need to estimate the XOR universality for a fixed key K. Π is an upper bound on the expected differential characteristic probability of the block cipher E over all keys. When a key K is fixed, we consider the fixed-key differential characteristic probability π_K which is the probability that an input pair $(x, x \oplus a)$ fulfills a differential characteristic $(\Delta_0 = a, \Delta_1, \ldots, \Delta_r = b)$.*

Let $N(a,b) = |\{x \colon E_K(x) \oplus E_K(x \oplus a) = b\}|$, i.e. $N(a,b)$ is the number of pairs (x, x') that fulfill the differential characteristic $(\Delta_0 = a, \Delta_1, \ldots, \Delta_r = b)$ for a fixed key K. Let q_B be the probability that all non trivial characteristics are fulfilled by at most B input pairs, i.e. $q_B = \Pr_{\substack{a,b \\ a \neq 0}}[N(a,b) \leqslant B]$. According to [18]: $q_B \geqslant 1 - \frac{\Pi^B 2^{(n-1)B}}{(B+1)!} 2^{2n}$. Hence, for a fixed key K: if $q_B \approx 1$ then $N(a,b) \leqslant B$, and $\Pr_x[E_K(x) \oplus E_K(x \oplus a) = b] \leqslant \frac{B}{2^n}$.

White-box implementation of 2-TEM($\mathcal{P}$). We use the 8-bit instantiation of SPNbox [21] as block cipher E. The description of the SPNbox family can be found in Chapter 4, Section 4.4.

Since $K = (K_1, K_2)$ and 2-TEM needs four block cipher calls, the white-box implementation makes use of four pre-computed S-boxes with K_1, K_1^2, K_2, K_2^2. The total size of the S-boxes is 1 kB.

Black-box security. Since 2-TEM is a tweakable block cipher, we evaluate the indistinguishability of the cipher in the random permutation model. In the random permutation model, we consider a distinguisher which interacts with some oracles and aims to distinguish the real world where it interacts with the tweakable block cipher and the ideal world where it interacts with a random tweakable permutation. We refer to [37] for more details about the interaction. In a nutshell, the distinguisher is allowed to make q_c queries to a construction oracle and q_p permutation queries to some inner permutation oracles. If the distinguisher makes a contruction query then the oracle returns either the output of the tweakable block cipher with a key K drawn uniformly at random (in the real world) or the output of a random tweakable permutation (in the ideal world). The following theorem gives the bound on the advantage of a chosen ciphertext distinguisher in the random permutation model.

Theorem 5. *Let $\mathcal{H}$ be a ρ_1-uniform and ρ_2-AXU family as defined in Definition 70 where $\rho_1 = \eta \cdot 2^{-n}$. In the random permutation model, the advantage of an adversary $\mathcal{A}$ making $q_c + q_p$ queries to distinguish 2-TEM($\mathcal{P}$) under Chosen-Ciphertext Attack is lower than $\frac{(24+3\eta^2+2\eta)\sqrt{q_c}q_p}{2^n} + \rho_2\eta\sqrt{q_c}q_p + 4\rho_2 q_c^{3/2} + \frac{30q_c^{3/2}}{2^n}$.*

Proof. The proof of indistinguishability is the same as provided by Cogliati et al. [37]. Using an almost uniform family instead of a uniform family, we obtain the following bounds for bad and good transcripts:

$$\Pr[T_{id} \in \Theta_{bad}] \leqslant \frac{3q_c^2 q_p \eta^2}{2^{2n}} + 2\rho_2^2 q_c^3 + \frac{\rho_2 q_c^2 q_p}{2^n} + \frac{2\sqrt{q_c}q_p\eta}{2^n} + 2\rho_2 q_c^{3/2}$$

$$\frac{\Pr[T_{re} = \tau]}{\Pr[T_{id} = \tau]} \leqslant 1 - \left(\frac{4q_c(q_p + 2q_c)^2}{2^{2n}} + \frac{14q_c^{3/2} + 4\sqrt{q_c}q_p}{2^n}\right)$$

□

White-box security. The PUF-based white-box encryption scheme satisfies the UBK-ATK security, for ATK $\in$ {CPA, CCA} because of the unbreakability of SPNbox-8. Regarding the lockability security, it is guaranteed by the unforgeability and unclonability of the PUF instance. Hence, a tight estimation of the security bounds for a specific PUF class is necessary to get the security bounds for lockability.

Remark 7. *A preliminary step to the use of a PUF is an enrollment. A PUF is an unclonable hardware function so only the owner of the device has access to it. Since the encryption scheme is symmetric, the other party should agree on the same PUF outputs to guarantee the correctness of the encryption. This phase of agreement on PUF responses is precisely the enrollment which is described in the next section.*

1.5.4 The encryption

Our scheme is designed for exchange of encrypted data between a trusted server and a client on an untrusted host. The server and the client share a secret key for the encryption: the client is given a white-box program with the hard-coded key. In addition, if a secure[20] PUF exists on the client device, it is used to reinforce the security of the white-box program. Because of the uniqueness of a PUF instance, an enrollment phase is needed to "characterize" the PUF. In other words, the trusted server stores some *information* about the PUF which are then used for the encryption. The enrollment phase consists in storing a set of challenge-response pairs corresponding to the PUF, i.e. the server evaluates the PUF on random challenges thanks to an evaluation program SECURE.EVAL executed on the client device. Such an enrollment phase is used for PUF-based authentication. During the enrollment phase, some helper data are computed to enable the client to recover the enrolled responses from noisy ones. A helper data is computed by a SECURE.SKETCH procedure and stored by the server. It is a public side information that enables one to recover a string y from any noisy but close enough y'. Assume that the server sends some encrypted data to the client and the client needs to decrypt them. The server randomly picks a pair of challenge-response and encrypts the data using the shared key and the PUF response. Then the server sends the ciphertext together with the challenge and the helper data to the client. The client evaluates the PUF instance on the challenge and gets a noisy response. Thanks to a recovery procedure REC and the helper data, the client recovers the correct response and decrypts the ciphertext. The descriptions of enrollment and encryption are respectively shown in Figures 1.2 and 1.3.

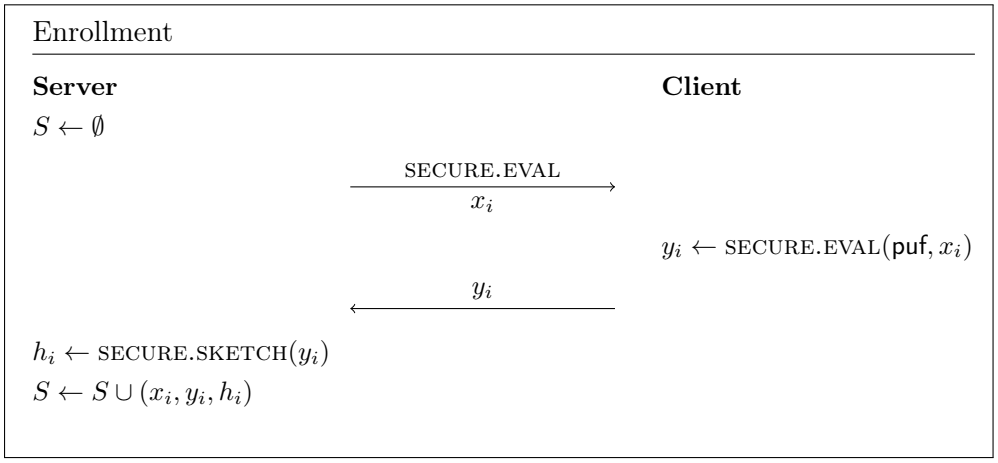


Figure 1.2: The enrollment step between a trusted server and a client: S is the set of challenge-response pairs associated to the client. puf is the PUF instance on the client device, SECURE.EVAL is a secure evaluation procedure executed by the server on the client device and SECURE.SKETCH is a procedure which enables to generate a helper data from a PUF response.

We give in the next section a possible implementation of the PUF component for mobile devices. The described PUF has the advantage to be available on modern smartphones and directly exploitable without any hardware overhead.

[20] Unpredictable and unclonable.

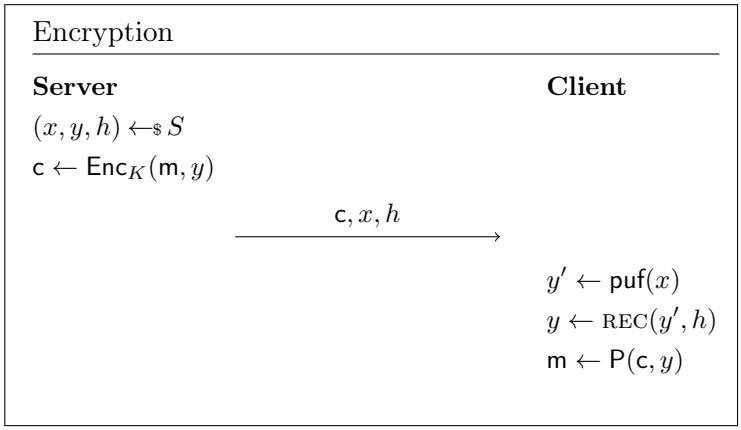


Figure 1.3: The encryption step between a trusted server and a client. m is the plaintext encrypted with a secret key K, REC is a recovery procedure which recovers a PUF response y from a noisy response y' and a helper data h and P is a white-box program implementing the decryption algorithm.

1.6 An image sensor-based PUF

In this section, we investigate the possibility of characterizing a PUF on modern smartphones. Let us recall that a PUF is based on random imperfections observed on an electronic component[21]. There are various built-in components on a mobile phone that can be exploited to characterize a PUF, namely the camera, the microphone, the radio frequency components, etc. In this thesis, we focus on the camera, namely on the image sensor of the camera.

As pointed out in [114, 74], the imperfection of the pixel array of the image sensor of a mobile device introduces a *Fixed Pattern Noise* (FPN) in the image. Such imperfection is due to impurities in the silicon wafers introduced during the manufacturing process. The imperfection is specific to each sensor and so uniquely identifies it. Thus, a sensor array can be seen as an unclonable hardware function and can be used to define a PUF. Since a PUF is characterized by a set of challenge-response pairs, it is necessary to characterize the random function.

1.6.1 Characterization of the PUF

Since the imperfection of the sensor array introduces a noise in the image, such a noise can be seen as the "expression" of the PUF or the response of the image sensor-based PUF. The Fixed Pattern Noise is composed of the Photo-Response Non-Uniformity (PRNU) [84] and the Dark Signal Non-Uniformity (DSNU) [57]. The PRNU is due to the sensibility variation between pixels, i.e. the ability of each optical sensor to convert photons to electrons. It is the dominant FPN in illuminated natural images. The DSNU is due to the variations of the dark current that passes the photodiodes even if no incident illumination arrives. Thus, the DSNU is dominant in dark images. Consequently, the characterization of the PUF first consists in extracting one of those *fingerprints*.

We choose to extract the DSNU to characterize the image sensor-based PUF. The reason is that the DSNU is less available from public images whereas the PRNU might be extracted from high-quality JPEG compressed public images. We follow the approach of [74] to extract the DSNU from t dark frames $D_1, \ldots, D_t$. The dark frames are raw images of size $H \times W$ obtained

[21] We focus on electronic PUF.

from the Camera 2 API available on Android devices. We apply the following steps to extract the DSNU:

Temporal noise removal: Calculate the pixel-wise average frame $\overline{D}$ to remove the temporal noise. Our experiment showed that at least $t = 15$ raw images are needed to remove the noise.

Noise residual extraction: Apply a Wiener filter to $\overline{D}$ then subtract the result from $\overline{D}$ to obtain the noise residual n. Since the DSNU is dominant in dark images, it is assumed that it is the only dominant noise in n.

High-frequency extraction: Apply the Discrete Cosine Transform (DCT) to n to remove the low-frequency component then extract the high-frequency (HF) component by calculating:

$$\mathsf{dsnu} = HF(\mathsf{n}) = \mathsf{IDCT}(\mathsf{DCT} \odot M)$$

where $\odot$ is the Hadamard product and M is a high-pass filtering matrix defined for a cutoff constant $c \in [0, 1]$ as:

$$M_{i,j} = \begin{cases} 1 \text{ if } i \geqslant H \cdot c \text{ and } j \geqslant W \cdot c \\ 0 \text{ otherwise} \end{cases}$$

At this stage, the size of the DSNU is around twenty megabytes. To make this exploitable on a mobile device, the size need to be reduced to a few kilobytes. Thus, the DSNU, seen as a vector of a subspace of $\mathbb{R}^n$ is projected to a subspace of smaller dimension m. Especially, the *compressed* DSNU is quantized to get a m-bit binary string.

1.6.2 Dimensionality reduction

The adaptive random projection of [114] is used to reduce the size of the DSNU.

Definition 72 (Random projection)**.** *Let $m, n \in \mathbb{N}$ such that $m < n$. For any $u \in \mathbb{R}^n$, a projection $v \in \mathbb{R}^m$ of u is defined by:*

$$v = \phi \cdot u^t$$

where ϕ is a $m \times n$ matrix called a sensing matrix.

The Johnson Lindenstrauss lemma [67] states that for a sensing matrix with Gaussian independent and identically distributed (i.i.d.) coefficients, the distances of the vectors are nearly preserved by the projection.

Definition 73 (Binary random projection)**.** *Let $m, n \in \mathbb{N}$ such that $m < n$. For any $u \in \mathbb{R}^n$. Let ϕ be a Gaussian independent identically distributed sensing matrix. A binary projection $v \in \{0, 1\}^m$ of u is defined by:*

$$v = sgn(\phi \cdot u^t)$$

where for any vector $a = (a_0, \ldots, a_{m-1}) \in \mathbb{R}^m$: $sgn(a) = (sgn(a_0), \ldots, sgn(a_{m-1}))$ and

$$sgn(a_i) = \begin{cases} 1 \text{ if } a_i > 0 \\ 0 \text{ otherwise} \end{cases}$$

Adaptive binary random projection. The adaptive random projection described in [114] enables a better binary representation[22] of any DSNU highly correlated with a reference DSNU, i.e. any DSNU extracted from the same device under the same conditions. Besides, the sensing matrix is defined as a circulant matrix with i.i.d. Gaussian coefficients and so, the matrix multiplication is performed efficiently using a Fast Fourier Transform. Recall that $\mathsf{dsnu} \in \mathbb{R}^n$ is the extracted high-dimensional DSNU. The projected vector $w \in \{0,1\}^m$, called the *DSNU fingerprint* is calculated as follows:

- For a parameter m' such that $n > m' > m$, the vector $v' = \phi \cdot \mathsf{dsnu}$ is computed, for ϕ a $m' \times n$ circulant matrix.
- The m components of v' with the largest magnitude are identified and constitute a vector $v'' \in \mathbb{R}^m$. The positions of these components are retained.
- The DSNU fingerprint is defined by $w = sgn(v'') \in \{0,1\}^m$.

The adaptive random projection ensures that correlated DSNU, i.e. DSNU extracted from the same device at different times but under the same conditions are mapped into "near" binary vectors. Actually, when the correlation coefficient tends to 1, the Hamming distance between the binary vectors tends to 0.

1.6.3 Error correction

Two DSNU extracted from the same device under the same conditions are only similar in average, i.e. the distributions of pixel values are the same. The values of the pixel vary from a DSNU to another and the binary vectors obtained after the projection will differ in several bits. As for any PUF instances, an error correcting code is used to correct the bit errors. Valsesia et al. [114] proposed to use a *polar code* first introduced in [2] for the PRNU. We followed the same approach for the DSNU.

Definition 74 (Polar code)**.** *A polar code $C(m,k)$ of length m and dimension k is a k-dimensional linear subspace of $\mathbb{F}_2^m$. For $k' = \log_2(m)$, let $B^{\oplus k'}$ be the k'-fold Kronecker product of the kernel matrix $B^{\oplus 1} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ i.e. $B^{\oplus k'} = \begin{pmatrix} B^{\oplus k'-1} & 0_{k'-1} \\ B^{\oplus k'-1} & B^{\oplus k'-1} \end{pmatrix}$ for any $k' > 2$. Let A be the $k \times m$ matrix such that for any $x \in \mathbb{F}_2^k$, the result of $A \cdot x$ is constituted of k bits of x and $m-k$ bits set to 0.*

The generating matrix G of the code C is the $k \times m$ matrix defined by:

$$G = A \cdot B^{\oplus k'}.$$

The encoding function is an injective linear application which associates a codeword $c \in C$ to any element $x \in \mathbb{F}_2^k$ as follows:

$$c = x \cdot G.$$

Let C be a polar code of length m and dimension k. Let w be a reference DSNU fingerprint. The error correcting code is used to define a secure sketch:

[22] than a simple binary random projection.

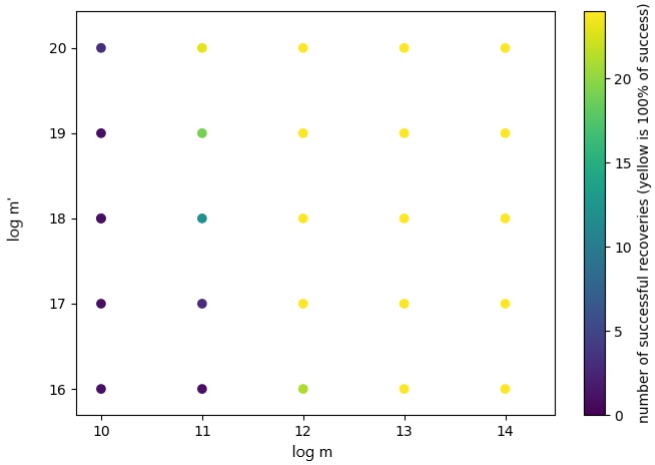


Figure 1.4: Probability of success of the decoding algorithm of a Polar code according to the values of m' and m. Device: Samsung Galaxy S7, number of DSNU = 100 with an average ISO and exposure time.

- A random k-bit vector x is generated and encoded using the polar code: the m-bit codeword c is defined by

$$c = x \cdot G.$$

- A secure sketch s is defined as:

$$s = w \oplus c.$$

- Given a noisy fingerprint w' such that $\text{HD}(w, w') \leqslant \delta_1$ (δ_1 is the intra-distance) and the secure sketch s, a noisy codeword c' is calculated:

$$c' = s \oplus w'.$$

  Since $w' = w \oplus e$ then $c' = s \oplus w \oplus e = c \oplus e$.

- The decoding algorithm first recovers $c = c' \oplus e$, then retrieves $x = G^{-1} \cdot c$.

Determining the values of m and m'. The value m is the length (in bits) of the fingerprint, i.e. the dimension of the vector space when the fingerprint is seen as a vector. m' is the dimension of an intermediate projected space and correspond to the number of components calculated with the sensing matrix. The parameter m' determines for a sample of N extracted DSNU, the conservation of the Hamming distances after the (intermediate) projection according to the Johnson-Lindenstrauss lemma. If m' is too small then the Hamming distance between the

Brand	Model	Number
Samsung	Galaxy S7	2
Nokia	7 Plus	1
LG	H870	2

Table 1.1: List of devices used in the experiments.

Number of DSNU N	Fingerprint length m	Code dimension
100	2^{12}	256

Table 1.2: Parameters used in the experiments to estimate the values of m' and m.

binary vectors coming from different DSNU will be similar to the Hamming distance between the binary vectors obtained from correlated DSNU. An experimentation on two Samsung Galaxy S7, one Nokia 7 Plus and two LG H870 for a serie of $N = 100$ DSNU shows a 100% true positive and 100% true negative for $m = 2^{12}$ and $m' = 2^{18}$ (Figure 1.4).

1.6.4 Description of the weak PUF

Let us recall that a PUF is characterized by a challenge-response behavior. A challenge is generally a signal which enables the stimulation of the physical device while the response is actually the digital value obtained by measuring the physical expression of the PUF.

The *weak* PUF with a single challenge-response pair is characterized as follows:

Challenge: a set of predefined parameters (e.g. fixed ISO sensitivity and exposure time).

Measured response: a m-bit fingerprint.

The list of devices used in the experiments as well as the chosen parameters are given in Tables 1.1 and 1.2.

The described weak PUF might not be robust enough in the white-box model. Although the measured response is unpredictable (there are $2^m = 2^{2^{15}}$ possibilities) and the sensor imperfection is physically unclonable, if the DSNU is estimated by an attacker who has a temporary access to the device, then the unpredictability property is not guaranteed anymore. In order to satisfy the security notions of Section 1.3.2, we investigate the possibility of defining a *strong* PUF from the image sensor.

1.6.5 On the possibility of characterizing image sensor-based strong PUF

To characterize a strong PUF, one needs to enlarge the set of challenges. So, we have explored the parameters that can influence the noise induced by the sensor and de facto affect the PUF responses. In particular, we focused on two parameters which are easily controllable on a mobile device: the exposure time and the ISO sensitivity.

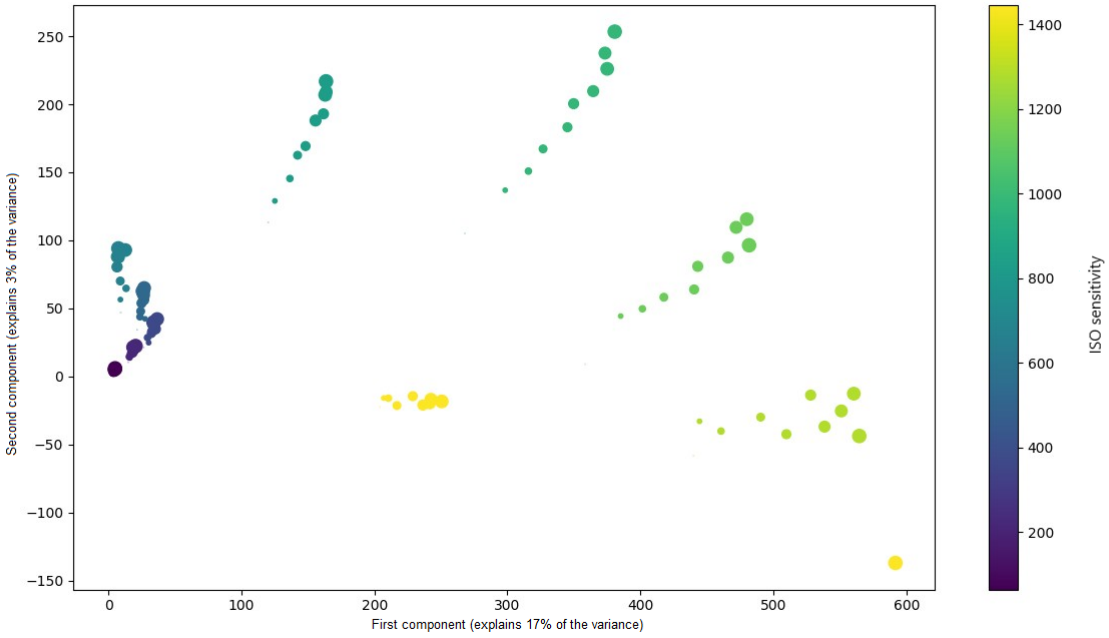


Figure 1.5: Variance of the pixel's value depending on the number of RAW images averaged using the Principal Component Analysis. The size of the point is proportional to the exposure time. Device: Samsung Galaxy S7.

The exposure time. The exposure time is the integration time of the signal by the sensor. The longer the exposure time, the more the number of electrons measured by the photodiodes. In the case of dark signal, there is no variation of the input signal since it is set to zero when the sensor is hidden from exterior light. So increasing exposure time only lead to a longer measurement of the same dark current. Actually, the DSNU probably depends linearly on the exposure time. Hence, this linear dependence makes the exposure time unusable for a strong PUF.

Sensitivity ISO. The ISO sensitivity measures the sensitivity of the sensor to the light. The sensitivity is the number of photons required to change the grayscale. It depends on three factors: the active surface of the photodiode, the quantum efficiency of the photodetector and the reflexion coefficient of the material.

A principal component analysis (PCA) keeping only the two main components of the DSNU for different ISO values shows that the ISO sensitivity has a more significant influence on the DSNU variance. The projection of the DSNU on the first PCA base vector shows that the DSNU strongly depends on the ISO value. The first axis explains between 15% and 20% of the variance of the DSNU for high values of the ISO (Figure 1.5). The correlation to a reference DSNU with low sensitivity remains almost constant and low on the whole sensitivity range. On the opposite, the correlation to a reference DSNU with high sensitivity strongly depends on the ISO values for values greater than 800 (Figure 1.6). However, experiments also show that "near" ISO values imply a high correlation between the corresponding DSNU and so fingerprints with a

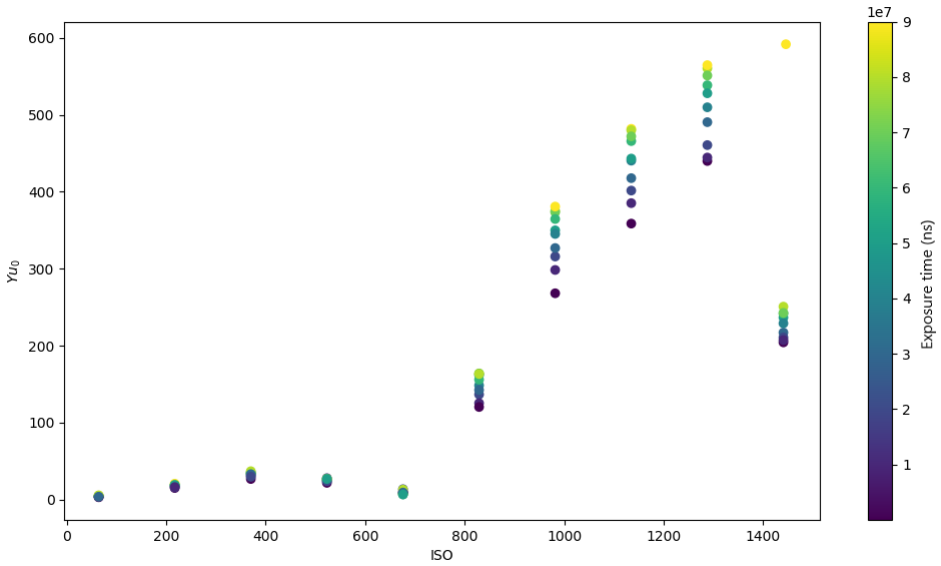


Figure 1.6: Projection of DSNU data on the first PCA axis which explains 17% of the variance function of the ISO sensitivity. Device: Samsung Galaxy S7.

small Hamming distance. The main problem with this analysis is that ISO values are considered to be "near" a reference value if they belong to a wide range of ISO values around the reference. Thus, the whole range of ISO values can only be partitioned into five ranges that each defines an equivalence class. The equivalence classes have the following property: any ISO value of the corresponding range determines highly correlated DSNU and so leads to the same fingerprint (up to some error bits). Thus, the five equivalence classes correspond to five challenge-response pairs. Hence, the characterized PUF is still a weak PUF

We conclude that neither the exposure time nor the ISO sensitivity are exploitable to characterize a strong PUF.

Conclusion of the chapter

In this chapter, we have proposed a method to integrate a hardware component in the definition of an encryption scheme to lock the encryption to a device. The unclonability of a PUF makes it a good candidate for our purpose. Typical applications of PUFs found in the literature are system identification, authentication and secure key storage. In the last case, the PUF is combined with a cryptographic algorithm making a PUF-based cryptography. We have enriched the field by proposing a PUF-based encryption scheme that can be implemented in the white-box context. The principle is that the PUF, as an unclonable hardware component, reinforces the white-box implementation of a cryptographic algorithm against code lifting. We formalize our goal with the notion of lockability. According to the attack scenario, we defined the lockability under forging attack and the lockability under cloning attack. We proved that the lockability security is implied by the unforgeability or unclonability of the underlying PUF instance. The PUF-based encryption scheme can be instantiated with a tweakable block cipher taking a PUF response as the tweak input. We chose the Even-Mansour tweakable block cipher but any construction of a tweakable block cipher suits as long as it allows for a secure white-box implementation. For this particular construction, we constructed a class of hash functions from block ciphers which we proved to be almost uniform and XOR universal. The main interest of such class of hash functions is that any white-box dedicated block cipher can be used and so a secure white-box implementation of any hash function of the family exists. Four instances of the block cipher SPNbox-8 are sufficient to define the hash function, thus the total white-box size is only 2^{13} bits. Since the critical part of the scheme is the PUF instance, it is necessary to use a PUF with strong properties. In particular, according to the use case, a strong PUF is often mandatory (for critical applications like mobile payment).

As a proof-of-concept, we have studied the possibility of characterizing a PUF on a mobile device. Actually, we looked for "available" PUF instead of "implemented" PUF such as the SRAM PUF. We focused on the image sensor which shows random variations in its structure and is available on any modern smartphone. The main issue is to measure such variations to make them usable by an application. Following previous works on image sensor-based PUF, we measure the Fixed Pattern Noise introduced by the sensor in an image and consider it as the PUF response. We successfully characterized a weak PUF by measuring the Dark Signal Non Uniformity from raw images. However, we have concluded that defining a strong PUF using the same method is unreachable. Anyway, the constructed weak PUF can be used to lock white-box implementations for non-critical applications (e.g. access control or ticketing using Host Card Emulation).

2

Traitor tracing

"I just invent, then wait until man comes around to needing what I've invented."

R. Buckminster Fuller

Contents

In this chapter we address the problem of traceability of white-box programs. As explained in [43], such a property enables to define a traitor tracing scheme which is collusion-resilient. In particular, the traitor tracing scheme described in [43] implements a *Private Linear Broadcast Encryption* scheme. We have investigated the possibility of implementing the scheme with the state-of-the-art white-box block cipher, SPNbox. The idea is to modify some values of the lookup table in the last round such that the output of the cipher is "incorrect" for a small set of inputs. We conclude that in this case, the approach has two main drawbacks: first, the number of users of the system is limited and second, a Chosen-Ciphertext Attack enables to recover the "hidden perturbations" with a practical complexity. Consequently, we propose a different approach: a traitor tracing system which uses a user-specific key generator. All key generators enable to generate a common key which ensures the correctness of the system. This chapter aims at studying *the link between the traceability property of a white-box program and the traitor tracing problem.*

2.1 Introduction

A broadcast encryption scheme is a cryptographic method used to distribute encrypted data over an insecure channel[23]. Such a scheme is used, e.g., for digital content distribution, and each user of the system generally owns a decryption box to decrypt the data. To allow the users to decrypt the data, a broadcast information is sent and each user is provided with a secret key that enables him to decrypt the encrypted data. The decryption box can be a tamper-resistant device such as a smart-card or a software on a personal computer or a smart-phone. Because tamper-resistant devices are hard and expensive to produce, software decryption boxes are more and more spread. For both cases, the main issue encountered is the illegal redistribution of the decryption key. In the case of a software decryption box, key extraction is within the reach of a white-box attacker. Illegal redistribution of decryption keys may then lead to a substantial financial loss and one solution to this threat is traitor tracing. A traitor tracing scheme is a broadcast encryption provided with a tracing procedure. In a traitor tracing system, each user's decryption box is provided with a distinct secret key which uniquely identifies him. Sometimes, some misbehaved users called the traitors collude to forge and redistribute a decryption key. The system is collusion-resilient if the tracing procedure retrieves at least one those traitors.

The problem of traitor tracing was first introduced in [33] by Chor, Fiat and Naor who proposed a combinatorial scheme. Since the problem statement numerous works proposed collusion-resilient schemes which can be classified into three classes: combinatorial scheme, public key scheme (first introduced in [78]) and, collusion-secure code-based scheme. Kiayias and Yung [72] first introduced collusion-secure code-based scheme to solve the problem of traitor tracing and many papers followed the trend [23, 54]. One advantage of such schemes is that the ratio of the ciphertexts and the plaintexts is constant. In particular, Billet and Phan introduced in 2008 a collusion-resilient traitor tracing scheme based on a Tardos code [13]. A Tardos code is a binary code that was introduced by G. Tardos in 2003 for watermarking digital documents [112]. Each document is identified with a mark which is a codeword embedded within the document and if some malicious users collude to produce a pirate version of the document, an accusation algorithm retrieves at least one of the misbehaved users. Billet and Phan harnessed the fact that traitor tracing and collusion-secure codes may share some properties. Actually, both have a collusion strategy that is constrained by an assumption called a "marking assumption". Simply put, the marking assumption states that a coalition of users are only able to identify the positions where their codewords differ. Our traitor tracing scheme belongs to the class of collusion-secure code-based scheme and uses a Tardos code. Especially, we describe a symmetric Tardos code as introduced by Skoric et al. [110]. The main difference with Billet and Phan's work is that the bits of the codeword are hidden. Consequently the collusion strategy that we describe is completely different and is not constrained by the standard marking assumption. In particular, the proof of the collusion resistance does not directly depend on the collusion resistance of the code. Besides, the code length is shorter than the "optimal" length suggested by G. Tardos.

Solving the problem of traitor tracing in the white-box model was suggested by Delerablée et al. in [43]. They presented the desired "white-box security notions", i.e. the security notions that should be proved to state the security of an encryption scheme in the white-box model. Specifically, the problem of traitor tracing is rephrased by the traceability property of a white-box program. The authors stated that "a program can be made traceable by unnoticeably modifying its functionality", i.e. if the set of inputs for which the modified program unusually behaves is negligible compared to the whole set of inputs then the program can be traced using

[23] In particular used by the Advanced Access Content System (AACS) standard

these inputs. In the same paper, Delerablée et al. defined an encryption scheme that is collusion-resilient using a primitive first introduced in [24] called a Private Linear Broadcast Encryption (PLBE) scheme. Yet they did not instantiate the encryption scheme in the white-box model and we are not aware if it can be practically instantiated with known white-box implementations.

Billet and Gilbert proposed in [11] a "traceable block cipher" that can be used in a broadcasting system. Their scheme is non combinatorial and relies on the hardness of the Isomorphism of Polynomials (IP) problem. Actually, each user of the broadcasting system receives a distinct description of the block cipher as systems of multivariate polynomials. Even if the description of the block cipher differs from a user to another, the evaluation of the block cipher gives the same result for all users which ensures correctness. Unfortunately, the scheme is unlikely a weak instance of the IP problem and can be solved efficiently [53].

In this chapter we describe a traitor tracing system with a key generation function, i.e. instead of using a decryption algorithm to decrypt a content key, we use the key generator. We use a Tardos code to encode the identity of each user of the system. The codewords are then used to compute a user-specific key generator. In order to derive distinct user-specific key generators that correctly "hide" the codewords, we use techniques of WBC to construct the key generator. We describe a collusion strategy that is completely different from the strategy associated to a Tardos code. Indeed we show that either the collusion retrieves the (untraceable) broadcaster's key generator or it exposes the identity of one traitor.

2.2 Traitor tracing

The concept of traitor tracing was introduced by Chor, Fiat and Naor [33] to study the problem of identifying a set of dishonest or corrupted users who contribute to forge a pirate decoder in a broadcasting system. Thus, a traitor tracing scheme is an encryption scheme that includes a tracing algorithm used to identify the corrupted or colluding users. As an encryption scheme, the traitor tracing scheme guarantees the confidentiality of the messages, i.e. any message sent by the sender is hidden from every unauthorized user who does not have access to a decryption key. Besides, if the tracing algorithm enables to retrieve at least one user among a set of c colluding users who contribute to a pirate decoder, then the traitor tracing scheme is said to be c-collusion resilient or simply c-resilient.

Definition 75 (Traitor tracing scheme (TT))**.** *A traitor tracing scheme consists of four polynomial time algorithms (*Setup, Enc, Dec, Trace*) defined as follows:*

- $\mathsf{Setup}(1^\lambda, l)$ *where* λ *is the security parameter and* l *is the number of users, generates an encryption key* mk*, a tracing key* tk*, and* l *user secret keys* $\mathsf{sk}_1, \ldots, \mathsf{sk}_l$*:*

$$\{\mathsf{mk}, \mathsf{tk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_l\} \leftarrow_\$ \mathsf{Setup}(1^\lambda, l).$$

- $\mathsf{Enc}(\mathsf{mk}, m)$ *takes as input the encryption key* mk *and a message* m *and outputs a ciphertext* c*:*

$$c \leftarrow \mathsf{Enc}(\mathsf{mk}, m).$$

- $\mathsf{Dec}(\mathsf{sk}_i, c)$ *takes as input a user decryption key* sk_i *and a ciphertext* c *and outputs a message* m*:*

$$m \leftarrow \mathsf{Dec}(\mathsf{sk}_i, c).$$

- $\mathit{Trace}^D(\mathit{tk})$ *takes as input the tracing key, and given an oracle access to a decoder* D*, outputs a user identity* $i \in [l]$*:*

$$i \leftarrow \mathit{Trace}^D(\mathit{tk})$$

The traitor tracing scheme is correct if for all $l \in \mathbb{N}$, $\{\mathit{mk}, \mathit{tk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_l\} \leftarrow_{\$} \mathit{Setup}(1^\lambda, l)$*, and for any* $i \in [l]$*:*

$$\mathsf{Dec}(\mathsf{sk}_i, \mathsf{Enc}(\mathit{mk}, m)) = m$$

with overwhelming probability.

A TT scheme can be:

- A *trace-and-revoke scheme* if the tracing authority is able to revoke users (who have been corrupted or have coerced to forge a pirate decoder). After a revocation, the revoked users are unable to decrypt a ciphertext.
- A *secret key* TT scheme if both the encryption key mk and the tracing key tk are kept secret.
- A *public key* TT scheme if the encryption key mk is public (in this case it is generally denoted by pk) but the tracing key tk is kept secret.
- A *publicly traceable* TT scheme if both the encryption key mk and the tracing key tk are public.

Theorem 6 (Collusion resilience)**.** *Let* $\mathcal{T}$ *be a TT scheme defined for* l *users. Let* $S \subseteq [l]$ *be a set of colluding users, called the traitors such that* $|S| = c$ *for any* $c \geqslant 1$*. Let* D *be a pirate decoder forged by the set* S*. The TT scheme* $\mathcal{T}$ *is* c*-collusion resilient if:*

$$S' \leftarrow \mathit{Trace}^D(\mathit{tk}) \text{ such that } S' \subseteq S \text{ and } S' \neq \emptyset \tag{2.1}$$

2.2.1 Combinatorial traitor tracing schemes

The first traitor tracing scheme introduced by Chor et al. in [33] is a *combinatorial scheme*. In such class of TT schemes, any user is given a subset of symmetric keys. The tracing algorithm then consists in analyzing the set of keys embedded in the pirate decoder. This is called a *white-box tracing* as opposed to a *black-box tracing* where the tracing algorithm has only an oracle access to the pirate decoder.

[34] and [96] belong to this class of TT schemes.

2.2.2 Algebraic traitor tracing schemes

This class gathers public-key schemes and non-combinatorial schemes. In a public-key TT scheme, the Setup algorithm generates a public encryption key pk, several private decryption keys, and a tracing key tk. The security of the schemes generally relies on the DDH (Decisional Diffie-Hellman) assumption.

The public-key TT scheme of [22] uses a linear space tracing code and relies on the difficulty of computing a discrete logarithm in a group of prime order.

In [72] the TT scheme is white-box traceable and uses bilinear maps. The TT scheme of [31] also makes use of bilinear maps but is the first publicly traceable TT scheme.

The Private Linear Broadcast Encryption scheme was introduced by Boneh et al. in [24] to construct a fully collusion-resilient TT scheme, i.e. the tracing algorithm retrieves at least one of the traitors even if all users of the system collude. The construction of the PLBE scheme is based on pairings on composite order group. The scheme was improved in [25] to provide a public traceability and a revocation scheme.

The TT scheme of Billet and Gilbert [11] is a non-combinatorial TT scheme which relies on the Isomorphism of Polynomials problem.

2.2.3 Code-based traitor tracing schemes

When a code (generally a binary code) is used to generate the decryption keys, the TT scheme is a *code-based* TT scheme. The copyrighted symmetric scheme of [95] belongs to this class and uses a set of "special ciphertexts" to trace a pirate decoder.

The first use of collusion-secure fingerprinting codes was initiated by [73]. The same authors further improved the scheme using watermarks in [71].

In [99], the code-based TT scheme uses a different type of codes called *Identifiable Parent Property* codes. The TT scheme is publicly traceable.

Billet and Phan constructed a TT scheme with constant-size ciphertexts exploiting the collusion-resilience property of a Tardos code which is used as a fingerprinting code to protect digital documents [111].

The TT scheme of Boneh and Naor [23] also uses collusion-secure fingerprinting codes. Besides, the collusion resilience relies on both the IND-CPA security of the public key scheme and the collusion resilience of the code.

2.2.4 Tardos code

A Tardos code [111, 110] is a binary code used for fingerprinting digital document. The code is composed of l codewords, where l is the number of users who receive a copy of the document with a unique embedded codeword. If some dishonest users collude to create a pirate version of the document then the distributing entity executes an accusation algorithm which outputs a non-empty set of users or $\emptyset$. The code is collusion-resilient if the accusation algorithm outputs a subset of the set of colluding users. Tardos proved that for at most c colluding users a code of length $L = 100c^2\lceil\log(1/\epsilon)\rceil$ provides c-collusion resilience with a probability of failure strictly less than ϵ. Besides, Tardos proved that the length is optimal regardless of the collusion strategy. The code is probabilistic i.e. each codeword is constructed from a sequence of probabilities. A c-collusion ϵ-resilient binary code $\mathscr{C}[L, c, \epsilon]$ of length L is defined as follows:

- For $\tau = 1/(300c)$, let $0 < \tau' < \pi/4$ s.t. $\sin^2\tau' = \tau$. Let $r_i \in [\tau', \pi/2 - \tau']$ some random values. Define the probability p_i as $p_i = \sin^2 r_i$. The p_i are independent, identically

distributed (i.i.d.) random variables from $[\tau, 1-\tau]$.

- A codeword $\omega = \omega_0 \dots \omega_{L-1}$ is constructed s.t. $Pr[\omega_i = 1] = p_i$.

This way l codewords $\omega_1, \dots, \omega_l$ are constructed and represented as a $l \times L$ matrix M where for $u \in [1, l]$, each entry $\omega_{u,i}$ is independent:

$$M = \begin{pmatrix} \omega_{1,0} & \dots & \omega_{1,L-1} \\ \omega_{2,0} & \dots & \omega_{2,L-1} \\ \vdots & \ddots & \vdots \\ \omega_{l,0} & \dots & \omega_{l,L-1} \end{pmatrix}$$

Each of these codewords is embedded in the user's document and allows to identify the owner of the document. The collusion-resistance of the code depends on an accusation algorithm and a "marking condition". In a nutshell, the marking condition ensures that a collusion of users $\mathcal{T}$ which compare their codewords are only able to modify the bits where their codewords differ, i.e. if $y \in \{0,1\}^L$ is the binary word forged by the collusion then y is such that $\omega_{u,i} = b$ implies $y_i = b$ for $u \in \mathcal{T}$ and $i \in [0, L-1]$. The accusation algorithm relies on a $l \times l$ matrix U defined as follows:

$$\forall u \in [l], i \in [0, L-1],\ U_{u,i} = \begin{cases} \sqrt{\frac{1-p_i}{p_i}} \text{ if } \omega_{u,i} = 1 \\ -\sqrt{\frac{p_i}{1-p_i}} \text{ if } \omega_{u,i} = 0 \end{cases}$$

Given a forged codeword y, the distributing entity is able to retrieve a subset of the set of colluding users $\mathcal{T}$ by computing an accusation sum $\sum_{i=0}^{L-1} y_i U_{u,i}$ for each user u. For any bit number i, if $y_i U_{u,i}$ is positive then $y_i = \omega_{u,i}$ and the accusation algorithm tends to accuse the user u. Otherwise, if $y_i U_{u,i}$ is negative that means that $y_i \neq \omega_{u,i}$ and the user u is considered as an innocent. This way, the accusation algorithm determines how much a user can be considered guilty. If $\sum_{i=0}^{L-1} y_i U_{u,i} > Z$ where $Z = 20c\lceil \log(1/\epsilon) \rceil$ is a fixed threshold then u is accused. Thus, the accusation algorithm does not need to compute the accusation sums for all users of the system but outputs the index of a user as soon as the corresponding sum is sufficiently high.

The work of Tardos aimed to provide the optimal code length that enables collusion resistance regardless of the collusion strategy. Besides, the code length does not depend on the number of users of the system but only on the number of colluding users tolerated which makes it practical for a large system. In his work, Tardos proved that:

(i) For an arbitrary user $u \in [l]$ and a coalition $\mathcal{T} \subseteq [l] \setminus \{u\}$ the probability that u is wrongly accused is less than ϵ.

(ii) For a coalition $\mathcal{T} \subseteq [l]$ of size c, the probability that no users of the coalition is accused is less than $\epsilon^{c/4}$.

In other words, the probability of false-positive when all other users collude is at most ϵ and the probability of non-detection for a coalition of size at most c is at most $\epsilon^{c/4}$.

We give some details about the Tardos code to understand the intuition behind the construction of the code. Actually, we do not use the same accusation algorithm since the marking condition does not hold in our case. We only take advantage of the probabilistic generation of the code.

2.3 White-box traitor-tracing

2.3.1 Broadcasting encrypted information

One method to secure digital content distribution is broadcast encryption. Such a method is used, e.g., for pay TV systems or distribution of encrypted musics, videos and documents over the Internet. Each user owns a decryption box (smart-card or software) and needs a decryption key to enjoy digital contents. Generally, the system is subject to subscription or digital management licenses before the key is delivered or updated. To avoid key distribution (a legitimate user shares his key to allow someone to decrypt content without subscribing), a unique personal decryption key is given to each user and unambiguously identifies him. Given any user key or a key forged by a subset of users, a tracing procedure retrieves the identity of the user (or at least one traitor when the key has been forged by a collusion). This way, the broadcast encryption scheme becomes a traitor tracing scheme. We first introduce our definition of a broadcast encryption scheme.

Definition 76 (Broadcast encryption (BE))**.** *A BE scheme is composed of three algorithms:*

- $\mathsf{Setup}(l,K)$ *takes as input the number of users* l *and a randomly drawn secret key* K *and outputs a key generator* KG *and* l *derived key generators* $KG_1, \ldots, KG_l$*:*

$$\{KG, KG_1, \ldots KG_l\} \leftarrow \mathsf{Setup}(l, K)$$

- $\mathsf{Enc}(EB,C)$ *takes as input an enabling block* EB *and a content* C *and outputs a pair* (EB, CB) *s.t.* CB *is the result of the encryption of a content* C *under a secret* CW *called a control word and generated by* KG *from the input* EB*.* CB *is called the cipher block:*

$$(EB, CB) \leftarrow \mathsf{Enc}(EB, C)$$

- $\mathsf{Dec}(u,EB,CB)$ *takes as input an index* u *and the broadcast information* (EB, CB) *and outputs* C_u *which is the result of the decryption of* CB *under a control word* $CW_u = KG_u(EB)$*:*

$$C_u \leftarrow \mathsf{Dec}(u, EB, CB)$$

The broadcast encryption scheme should be correct, i.e. for any user $u \in [l]$*:*

$$\text{if } \{KG, KG_1, \ldots, KG_l\} \leftarrow \mathsf{Setup}(l, K) \text{ and } (EB, CB) \leftarrow \mathsf{Enc}(EB, C) \text{ then}$$
$$C_u = C \text{ for } C_u \leftarrow \mathsf{Dec}(u, EB, CB).$$

The correctness property guarantees that all users of the system can recover the content C even if the key generators are all distinct. Here, the correctness of the BE relies on the specificity of the key generators and the way the enabling block EB is chosen. Actually, EB is randomly drawn from a subset of the input domain. We will give the details of the constructions of the key generator and of the enabling block respectively in Sections 2.3.2 and 2.3.3. For now, we describe our system as follows. Let l be the number of users. The setup algorithm outputs a key generator

KG and l user-specific key generators $KG_1, \ldots, KG_l$. The broadcaster chooses an enabling block EB computes CW using KG and computes CB using a block cipher E instantiated with the key CW. Any user u is given a decryption box provided with a key generator KG_u and the decryption algorithm E^{-1}. Given an enabling block EB, u computes the control word CW and then decrypts the cipher block CB into C. Thus, the broadcaster and the users of the system each generates the control word separately but both agree on the same control word. Figure 2.1 illustrates the difference between the decryption boxes of two users.

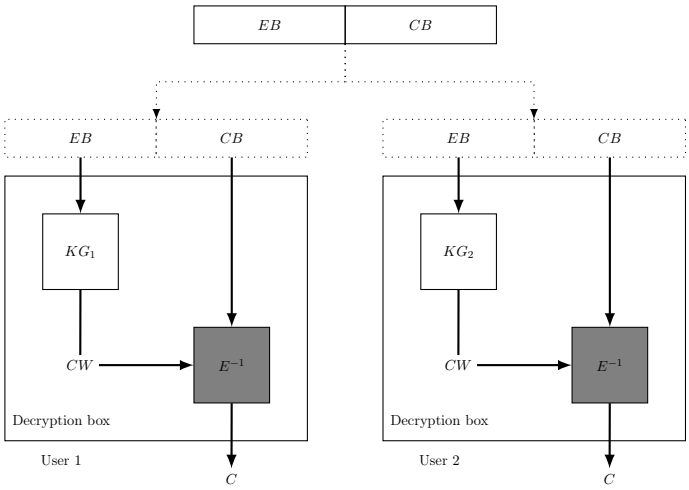


Figure 2.1: The system with user-specific key generators.

2.3.2 Description of the key generator

As we said before, the correctness of the scheme depends on the key generator. Thus, we introduce in this section our construction which is at the heart of the system. Before going into details, let us give the ideas behind the construction. We construct a function which outputs a N-bit key[24]. As the key should be random i.e. indistinguishable from a N-bit random sequence, the function should output any key with probability $\frac{1}{2^N}$. Let m, t be such that $N = mt$ and $f_1, \ldots, f_t$ be t permutation functions over $\mathbb{F}_{2^m}$. Then, each output of the function $f = (f_1, \ldots, f_t)$ given an input $(x_1, \ldots, x_t) \in \mathbb{F}_{2^m}^t$ appears with probability $(\frac{1}{2^m})^t = \frac{1}{2^N}$. Thus, the key generator makes call to t permutation functions. Besides, we want to derive user-specific description of the key generator. We chose to implement the permutation functions with m-bit lookup tables, so we used white-box techniques[25]. This way, the key generator makes use of t m-bit lookup tables which describe t distinct permutations over $\mathbb{F}_{2^m}$. As we will see in Section 2.3.3, the lookup tables will "embed" a Tardos codeword and become user-specific. For each lookup table, we use the AES-128 in counter mode to generate $2^m \cdot m$ pseudo-random bits and then use the Fisher-Yates shuffle algorithm [55] to generate a m-to-m bits pseudo-random permutation[26].

Let $f \colon \mathbb{F}_2^{128} \times (\mathbb{F}_2^m)^t \to (\mathbb{F}_2^m)^t$ be a function defined for any input (K, X) with $X = (x_1, \ldots, x_t)$

[24]"Key" here refers to the control word CW.

[25]A popular technique used in White-Box Cryptography to hide a part of a secret key is to compute some lookup tables that are key-dependent and then make table calls to execute the algorithm. Actually, our construction is a white-box implementation of a key generation algorithm.

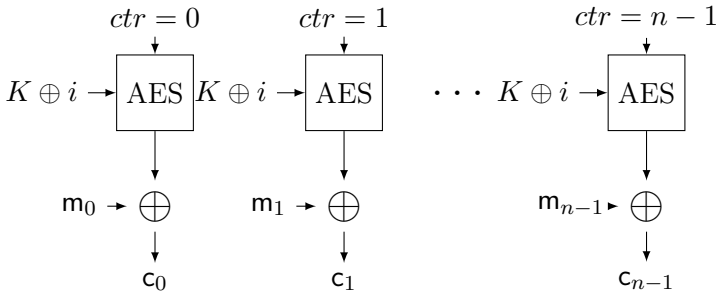
[26]We chose this method as it is an elegant way to generate a permutation from a random binary sequence. Other methods exist and can be used for this construction.

as:

$$f(K, X) = (T_{K\oplus 1}(x_1), \ldots, T_{K\oplus t}(x_t)),$$

where $T_{K\oplus 1}, \ldots, T_{K\oplus t}$ are lookup tables that each describes a random permutation. Each lookup table $T_{K\oplus i}$ for $1 \leqslant i \leqslant t$ maps m bits to m bits and can be seen as a random permutation over the range $\{0, 1, \ldots, 2^m - 1\}$. The tables are computed using the AES-128 in counter mode with the key $K \oplus i$ and with some plaintexts $\mathsf{m}_i = i$. The generated pseudo-random sequence of $2^m \cdot m$ bits is used to compute an array of 2^m m-bit values using the Fisher-Yates shuffle algorithm. We illustrate the tables generation in Figure 2.2.

Figure 2.2: Generation of $n = \frac{2^m \cdot m}{128}$ ciphertexts before the Fisher-Yates shuffle algorithm.



Since the key K is fixed and the tables are precomputed, we write $T_1, \ldots, T_t$ for $T_{K\oplus 1}, \ldots, T_{K\oplus t}$ and f_K in place of $f(K, \cdot)$.

Definition 77 (Key generator)**.** *For $d \geqslant 2$, the key generator $KG\colon (\mathbb{F}_2^{mt})^{2^d} \to \mathbb{F}_2^{mt}$ is defined for any $(X_1, \ldots, X_{2^d})$ s.t $X_k = (x_{k,1}, \ldots, x_{k,t})$ as*

$$KG(X_1, \ldots, X_{2^d}) \quad = \quad \left(\sum_{k=1}^{2^d} T_1(x_{k,1}), \ldots, \sum_{k=1}^{2^d} T_t(x_{k,t}) \right) = \sum_{k=1}^{2^d} f_K(X_k) \tag{2.2}$$

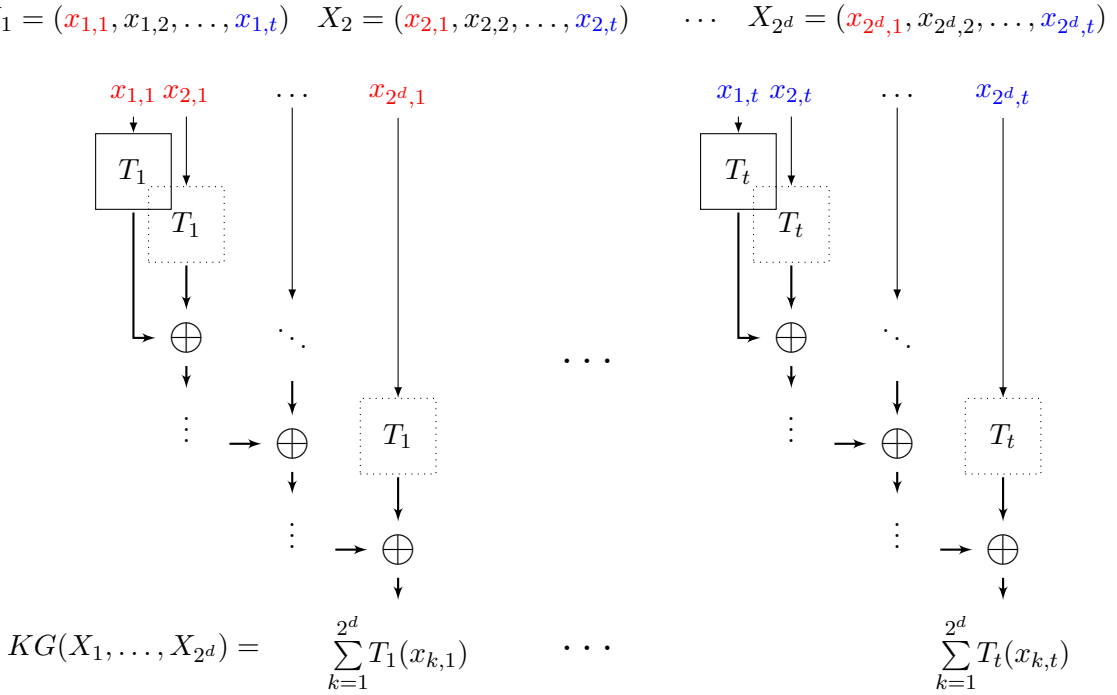
Remark 8. *As we see in Equation (2.2), the output of the key generator is a vector of t "sums". Each "sum" is the result of the XOR of 2^d permutations. Thus, the value of the sum can be distinguished from random with $\mathcal{O}(2^m)$ input/output values. As concluded in [98] the XOR of some permutations allows to generate a random function from random permutations. Hence, the output of the key generator is random.*

Remark 9. *The key generator is a white-box dedicated algorithm and provides the unbreakability security [43], i.e. the master key K cannot be recovered by an adversary having access to the lookup tables. Indeed, the key extraction security reduces to the security of AES-128 in counter mode.*

2.3.3 User-specific key generator

The key generator KG described in Section 2.3.2 is the broadcaster's generator. We describe in this section how the users' key generators are derived from KG. This is where the Tardos code comes in. We "encode" the identity of each user using a Tardos code. The user-specific key

Figure 2.3: The key generator.



generator is then computed according to the codeword associated to the user. For a user u, the key generator KG_u makes use of lookup tables $T_1^{(u)}, \ldots, T_t^{(u)}$ derived from the original lookup tables $T_1, \ldots, T_t$ by modifying all the tables values. However, this modification is done such that the user's key generator outputs the same output as the broadcaster's one on a specified set of inputs.

User's identification. Let a system with l users. Each user u is represented as a binary codeword of a c-collusion ϵ-resilient Tardos code $\mathscr{C}[L, c, \epsilon]$, i.e. $u = (u_0, \ldots, u_{L-1})$ s.t $Pr[u_i = 1] = p_i$ for a sequence of probabilities p_i outputted by a distribution $\mathcal{P}$. The users of the system are represented by a matrix with l rows and L columns. The main idea of the personalization is to use all bits of each user codeword to compute all the tables values. First, the codeword is divided into t sub-words of length L/t and each sub-word is used for each table. The parameters (L, d, m, t) of the scheme verify $L = td2^{\lceil m/d \rceil}$ where d is called the "dimension" of the system. d is a parameter chosen to ensure the collusion resistance of the system. We will see in Section 2.3.6 how it should be chosen.

User's perturbation function. Now that each user of the system is identified with a codeword, we define a user's perturbation function that depends on that codeword and enables to compute the user's tables accordingly.

Definition 78 (User's perturbation function)**.** *Let L, t, d, m be defined as above. Let $e_0, \ldots, e_{L-1}$ be L distinct non-zero m-bit values randomly drawn from $\mathbb{F}_2^m \setminus \{0\}$. For $d \geqslant 2$, let A be a d-dimensional array with $v = \frac{L}{td}$ cells which randomly contains the elements of $\mathbb{F}_2^m$. Let* index *be a function that takes $x \in \mathbb{F}_2^m$ as input and returns a d-uple $(\mathbf{i_1^{(x)}}, \ldots, \mathbf{i_d^{(x)}})$ of indices.* index *gives the coordinates of x in the d-dimensional array A, i.e. $0 \leqslant \mathbf{i_j^{(x)}} \leqslant v - 1$ for any $1 \leqslant j \leqslant d$.*

For any user u associated to a codeword $(u_0, \ldots, u_{L-1})$ and any $1 \leqslant i \leqslant t$, a perturbation

function is defined as

$$\forall x \in \mathbb{F}_2^m \ m_u(i,x) = \sum_{j=1}^{d} (u_{\mathbf{i_j^{(x)}}+jv+\frac{(i-1)L}{t}})(e_{\mathbf{i_j^{(x)}}+jv+\frac{(i-1)L}{t}}).$$

The values $e_0, \ldots, e_{L-1}$ are random secret values that will be used to transform the permutation implemented by T_i into a pseudo-random function. For any $x \in \mathbb{F}_2^m$, a mask value $m_u(i,x)$ is computed as a sum of at most d secret values among $e_0, \ldots, e_{L-1}$, i.e. if the bit u_* is equal to 1 then e_* contributes to the mask value, otherwise e_* does not contribute.

User's lookup table. For $1 \leqslant i \leqslant t$, the i-th table $T_i^{(u)}$ is computed from the original table T_i as follows: $\forall x \in \mathbb{F}_2^m \ T_i^{(u)}(x) = T_i(x) + m_u(i,x)$. m_u is a "perturbation function" used to mask the value of $T_i(x)$ for any x. Since $m_u(i,x)$ is random then $T_i^{(u)}$ implements a pseudo-random function.

Definition 79 (User-specific key generator)**.** *The user-specific key generator is then defined for any* $(X_1, \ldots, X_{2^d})$ *as*

$$KG_u(X_1, \ldots, X_{2^d}) = \left(\sum_{k=1}^{2^d} T_1^{(u)}(x_{k,1}), \ldots, \sum_{k=1}^{2^d} T_t^{(u)}(x_{k,t}) \right) \tag{2.3}$$

The idea behind the construction can be summarized as follows. For any user, each table value is a masked (perturbed) version of the broadcaster table value. The perturbation added to each table value is random and depends on the user codeword. However, the perturbation function is defined such that the sum $\sum_{k=1}^{2^d} m_u(i, x_k)$ equals 0 for some inputs $(x_1, \ldots, x_{2^d})$ and this way guarantees the correctness of the user's key generator. This is stated by the following lemma.

Lemma 1. *For any user* u*,* KG_u *is correct iff for any* $(X_1, \ldots, X_{2^d}) \in (\mathbb{F}_2^{mt})^{2^d}$*,* $\sum_{k=1}^{2^d} m_u(i, x_{k,i}) = 0$ *for all* $1 \leqslant i \leqslant t$*.*

Proof. Let $(X_1, \ldots, X_{2^d}) \in (\mathbb{F}_2^{mt})^{2^d}$.

$$\begin{aligned}
\left(\sum_{k=1}^{2^d} T_1^{(u)}(x_{k,1}), \ldots, \sum_{k=1}^{2^d} T_t^{(u)}(x_{k,t}) \right) &= \left(\sum_{k=1}^{2^d} T_1(x_{k,1}), \ldots, \sum_{k=1}^{2^d} T_t(x_{k,t}) \right) \\
\Leftrightarrow \ \forall i \ \sum_{k=1}^{2^d} T_i^{(u)}(x_{k,i}) &= \sum_{k=1}^{2^d} T_i(x_{k,i}) \Leftrightarrow \\
\forall i \ \sum_{k=1}^{2^d} (T_i(x_{k,i}) + m_u(i, x_{k,i})) &= \sum_{k=1}^{2^d} T_i(x_{k,i}) \Leftrightarrow \\
\forall i \ \sum_{k=1}^{2^d} (T_i(x_{k,i}) + m_u(i, x_{k,i})) - \sum_{k=1}^{2^d} T_i(x_{k,i}) &= 0 \Leftrightarrow \ \forall i \ \sum_{k=1}^{2^d} m_u(i, x_{k,i}) = 0
\end{aligned}$$

□

Definition 80 (Partial correctness)**.** *For any* u*,* KG_u *is partially correct if there exists a set* $I \subset (\mathbb{F}_2^{mt})^{2^d}$ *such that for any* $(X_1, \ldots, X_{2^d}) \in I$*,* $KG_u(X_1, \ldots, X_{2^d}) = KG(X_1, \ldots, X_{2^d})$*.*

We say that KG_u *is* I*-correct and that* I *is a perturbation canceler.*

The following proposition gives a sufficient condition on the set I to enable the the partial correctness.

Proposition 9. *Let $d \geqslant 2$. For any $1 \leqslant i \leqslant t$, let a set $I_i \subset (\mathbb{F}_2^{mt})^{2^d}$ such that for any $(x_{1,i}, \ldots, x_{2^d,i}) \in I_i$, for any $1 \leqslant j \leqslant d$ there exists a partition $\mathcal{P}^j_{1,i}, \ldots, \mathcal{P}^j_{2^{d-1},i}$ of the set $\{x_{1,i}, \ldots, x_{2^d,i}\}$ defined as follows*

$$\begin{cases} |\mathcal{P}^j_{k,i}| = 2 \\ \mathbf{i}_\mathbf{j}^{(\mathbf{x})} = \mathbf{i}_\mathbf{j}^{(\mathbf{x}')} \text{ for } x, x' \in \mathcal{P}^j_{k,i} \end{cases} \quad \text{and if } x \in \mathcal{P}^j_{k,i} \text{ and } y \in \mathcal{P}^j_{k,i'} \text{ then } \mathbf{i}_\mathbf{j}^{(\mathbf{x})} \neq \mathbf{i}_\mathbf{j}^{(\mathbf{y})}.$$

Then, for any u, P_u is I-correct for $I = I_1 \times \ldots \times I_t$.

Proof. We prove that for any $(x_{1,i}, \ldots, x_{2^d,i}) \in I_i$, $\sum_{k=1}^{2^d} m_u(i, x_{k,i}) = 0$.

For any i, let $(x_{1,i}, \ldots, x_{2^d,i}) \in I_i$. Then,

$$\begin{aligned} \sum_{k=1}^{2^d} m_u(i, x_{k,i}) &= \sum_{k=1}^{2^d} \left(\sum_{j=1}^{d} (u_{\mathbf{i}_\mathbf{j}^{(\mathbf{x_{k,i}})} + jv + \frac{(i-1)L}{t}})(e_{\mathbf{i}_\mathbf{j}^{(\mathbf{x_{k,i}})} + jv + \frac{(i-1)L}{t}}) \right) \\ &= \sum_{j=1}^{d} \left(\sum_{k=1}^{2^{d-1}} \sum_{x \in \mathcal{P}^j_{k,i}} (u_{\mathbf{i}_\mathbf{j}^{(\mathbf{x})} + jv + \frac{(i-1)L}{t}})(e_{\mathbf{i}_\mathbf{j}^{(\mathbf{x})} + jv + \frac{(i-1)L}{t}}) \right) \end{aligned}$$

Since for any $x, x' \in \mathcal{P}^j_{k,i}$, $\mathbf{i}_\mathbf{j}^{(\mathbf{x})} = \mathbf{i}_\mathbf{j}^{(\mathbf{x}')}$ then $u_{\mathbf{i}_\mathbf{j}^{(\mathbf{x})} + jv + \frac{(i-1)L}{t}} = u_{\mathbf{i}_\mathbf{j}^{(\mathbf{x}')} + jv + \frac{(i-1)L}{t}}$ and $e_{\mathbf{i}_\mathbf{j}^{(\mathbf{x})} + jv + \frac{(i-1)L}{t}} = e_{\mathbf{i}_\mathbf{j}^{(\mathbf{x}')} + jv + \frac{(i-1)L}{t}}$.

Thus, for any i, j, k, $\sum\limits_{x \in \mathcal{P}^j_{k,i}} (u_{\mathbf{i}_\mathbf{j}^{(\mathbf{x})} + jv + \frac{(i-1)L}{t}})(e_{\mathbf{i}_\mathbf{j}^{(\mathbf{x})} + jv + \frac{(i-1)L}{t}}) = 0$.

Hence, $\sum_{k=1}^{2^d} m_u(i, x_{k,i}) = 0$ for all i. □

Example 1. *Let $d = 2$ and $m = 4$. Then A is a 4×4 matrix containing all the elements of $\mathbb{F}_2^4$ (which we represent with *). Without loss of generality, let $i = 1$. For simplicity, we write (x_1, x_2, x_3, x_4) instead of $(x_{1,1}, x_{2,1}, x_{3,1}, x_{4,1})$. Assume that the matrix A is defined as follows*

$$A = \begin{pmatrix} \star & \star & \star & \star \\ \star & x_1 & \star & x_2 \\ \star & \star & \star & \star \\ \star & x_3 & \star & x_4 \end{pmatrix}$$

Since $\begin{cases} \mathbf{i}_\mathbf{1}^{(\mathbf{x_1})} = \mathbf{i}_\mathbf{1}^{(\mathbf{x_2})} = 1 \\ \mathbf{i}_\mathbf{1}^{(\mathbf{x_3})} = \mathbf{i}_\mathbf{1}^{(\mathbf{x_4})} = 3 \end{cases}$ *and* $\begin{cases} \mathbf{i}_\mathbf{2}^{(\mathbf{x_1})} = \mathbf{i}_\mathbf{2}^{(\mathbf{x_3})} = 3 \\ \mathbf{i}_\mathbf{2}^{(\mathbf{x_2})} = \mathbf{i}_\mathbf{2}^{(\mathbf{x_4})} = 1 \end{cases}$ *, we get the following partitions of $\{x_1, x_2, x_3, x_4\}$:*

$$\begin{aligned} \mathcal{P}^1_1 &= \{x_1, x_2\} \\ \mathcal{P}^1_2 &= \{x_3, x_4\} \\ \mathcal{P}^2_1 &= \{x_1, x_3\} \\ \mathcal{P}^2_2 &= \{x_2, x_4\} \end{aligned}$$

Then, (x_1, x_2, x_3, x_4) *belongs to the set* I_1.

Proposition 10. *Let* $d \geqslant 2$. *Let* I *be the set as defined in Proposition 9. Then for any* $1 \leqslant i \leqslant t$*:*

$$|I_i| \geqslant \prod_{k=1}^{2^{d-1}} (2^m - (k-1)d2^{(d-1)m/d}).$$

Proof.

- $d = 2$. Let us count all 4-uple $(x_{1,i}, x_{2,i}, x_{3,i}, x_{4,i}) \in (\mathbb{F}_2^m)^4$ s.t. there exits two partitions as defined in Proposition 9: $x_{1,i}$ can be any element of $\mathbb{F}_2^m$ so we have 2^m possible choices; $x_{2,i}$ can be chosen s.t. $\mathbf{i_1^{(x_{2,i})}} \neq \mathbf{i_1^{(x_{1,i})}}$ and $\mathbf{i_2^{(x_{2,i})}} \neq \mathbf{i_2^{(x_{1,i})}}$, thus we have $2^m - 2 \cdot 2^{m/2}$ possible choices. Since we get $x_{1,i}$ and $x_{2,i}$, without loss of generality we can set $x_{1,i} \in \mathcal{P}_1^1, \mathcal{P}_1^2$ and $x_{2,i} \in \mathcal{P}_2^1, \mathcal{P}_2^2$ then $x_{3,i}$ and $x_{4,i}$ are fixed as the second elements of each set.

- $d = 3$. We count all 8-uple $(x_{1,i}, \ldots, x_{8,i}) \in (\mathbb{F}_2^m)^8$. Following the same reasoning as above: there are 2^m possible choices for $x_{1,i}$, $2^m - 3 \cdot (2^{m/3})^2$ possible choices for $x_{2,i}$, at most $2^m - 6 \cdot (2^{m/3})^2$ possible choices for $x_{3,i}$ and at most $2^m - 9 \cdot (2^{m/3})^2$ possible choices for $x_{4,i}$. Each of these elements is set as the first element of each set $\mathcal{P}_k^j$, for $1 \leqslant j \leqslant 3$ and $1 \leqslant k \leqslant 4$, then $x_{5,i}, x_{6,i}, x_{7,i}, x_{8,i}$ are fixed as the second elements.

The same reasoning is used for any value of d, i.e. choose one element for each set $\mathcal{P}_k^j$, for $1 \leqslant j \leqslant d$ and $1 \leqslant k \leqslant 2^{d-1}$, then the 2^{d-1} remaining coordinates are fixed to be the second elements of each set. □

2.3.4 Broadcast information generation and decryption procedure

The broadcaster needs to generate a broadcast information (EB, CB) that enables all users of the system to recover a content C. Since each user has a distinct description of the key generator, we define the enabling block EB with a special structure. Actually, we take EB in the perturbation-canceler set I, thus the key generator removes all perturbations added to the user-specific description and enables each user to recover CW and the content C.

Let $E\colon \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a block cipher, i.e. E is the encryption function and E^{-1} is the decryption function. The system consists of the following steps:

1. The broadcaster randomly picks an enabling block EB in the perturbation-canceler set I, then generates a control word CW using KG: $CW = KG(EB)$.
2. The broadcaster encrypts a content block C using a block cipher E instantiated with the secret key CW. He computes the cipher block $CB = E_{CW}(C)$.
3. The broadcaster sends (EB, CB) to all users of the system.
4. Any user u generates the control word CW using his key generator KG_u: $CW = KG_u(EB)$ and decrypts the cipher block to obtain the content C: $C = E_{CW}^{-1}(CB)$.

2.3.5 The tracing algorithm

A tracing algorithm is executed by the tracing entity to identify the owner of a decryption box. We assume that the tracing procedure is executed in a white-box way, i.e. the tracing entity has access to the pirate decryption box and executes the tracing procedure on the key generator, namely on the set of lookup tables. The pirate decryption box embeds a key generator with

forged tables $T_i^{u^*}$ so the tracing procedure takes as input a set of pirate tables $T_i^{u^*}$ and outputs a user index u^*. For all $1 \leqslant i \leqslant t$, the tracing entity executes the following steps:

1. For any $x \in \mathbb{F}_2^m$ look at the value $T_i^{u^*}(x)$ and compare with the 2^d possible values $T_i(x) + \sum_{j=1}^{d}(u^*_{\mathbf{i_j^{(x)}}+jv+\frac{(i-1)L}{t}})(e_{\mathbf{i_j^{(x)}}+jv+\frac{(i-1)L}{t}})$.
2. Deduce the values of the bits $u^*_{\mathbf{i_j^{(x)}}+jv+\frac{(i-1)L}{t}}$ for all $1 \leqslant j \leqslant d$.
3. When $(u_0^*, \ldots, u_{L-1}^*)$ is fully determined, search the user in the matrix M.

The key generator is traceable in the sense that the identity of its owner can be retrieved by the tracing entity. We assume that even if some traitors collude either they expose the identity of one of them or they cannot create a functional pirate decryption box. Thus, the tracing procedure only needs to reconstitute the codeword associated to a key generator and then search in the matrix of all codewords. The details of the collusion strategy are described in the proof of Theorem 7.

2.3.6 Collusion resistance

The advantage of the broadcast encryption scheme described is Section 2.3.4 is the traceability of the key generators. This property enables to find the owner of an illegally redistributed decryption box. Besides, we prove that even if several dishonest users collude, they cannot produce an untraceable and functional key generator. We first define the notion of traceability against an arbitrary collusion.

Definition 81 (Traceability game against an arbitrary collusion)**.** *Let an adversary $\mathcal{A}$ and a challenger $\mathcal{B}$ playing the following game:*

1. *$\mathcal{B}$ randomly picks a key $K \in \mathcal{K}$ and a set of secret parameters* param[27]*.*
2. *$\mathcal{B}$ computes the key generator KG as in Section 2.3.2.*
3. *$\mathcal{A}$ chooses a set $\mathcal{T} = \{u_1, \ldots, u_c\} \subseteq [l]$ of traitors and queries the corresponding key generators with q chosen inputs $(X_1, \ldots, X_{2^d})$.*
4. *If $(X_1, \ldots, X_{2^d})$ is valid, i.e. $(X_1, \ldots, X_{2^d}) \in I$, then $\mathcal{B}$ outputs the corresponding tables outputs. Otherwise, $\mathcal{B}$ outputs $\perp$.*
5. *$\mathcal{A}$ outputs a key generator KG^*.*
6. *If KG^* is I-correct, $\mathcal{B}$ runs the tracing procedure* Trace *on KG^* and outputs a set $S \subseteq [l]$.*
7. *$\mathcal{A}$ wins if KG^* is I-correct and $S = \emptyset$ or $S \nsubseteq \mathcal{T}$.*

We say that a (L, d, m, t)-key generator satisfies (q, ϵ)-traceability against a c-collusion if the probability that $\mathcal{A}$ wins is less than ϵ.

Theorem 7. *Consider our construction of a (L, m, d, t)-key generator where L is the length of a Tardos code, m is the table input length, d is the dimension and t is the number of tables. We claim that if the adversary makes at most $q \leqslant |I_i|$ queries with q_v valid queries then no coalition of users less than $\frac{d}{2} + 1$ can produce a functional pirate key generator with probability greater than $\left(\frac{1}{2^m}\right)^{t(|I_i|-q_v)}$.*

[27] param are the parameters of the Tardos code and the random m-bit values $e_1, \ldots, e_{L-1}$.

Proof. We consider two cases: $c = 1$ and $c > 1$ and for both cases we describe two possible strategies that make the pirate key generator untraceable. First, the adversary records the outputs $(\sum_{i=1}^{2^d} T_1(x_{k,1}), \ldots, \sum_{i=1}^{2^d} T_t(x_{k,t}))$ for all valid inputs for later use and, second, the adversary recovers the original table values $T_i(x)$.

- $c = 1$. We assume that the adversary is a single user. They adopt the first strategy, i.e. record the outputs $(\sum_{i=1}^{2^d} T_1(x_{k,1}), \ldots, \sum_{i=1}^{2^d} T_t(x_{k,t}))$ for all valid inputs.

  Let q, q_v be respectively the number of input queries made by the adversary such that $q \leqslant |I_i|$ and the number of valid queries. One query can be seen as t simultaneous queries to the tables. For a query $\mathbf{q}$, we write $\mathbf{q_1}, \ldots, \mathbf{q_t}$ for the corresponding table queries. We say that a query $\mathbf{q}$ is valid iff all of the queries $\mathbf{q_1}, \ldots, \mathbf{q_t}$ are valid. If the query is valid, then the adversary gets the output $(\sum_{i=1}^{2^d} T_1(x_{k,1}), \ldots, \sum_{i=1}^{2^d} T_t(x_{k,t}))$ corresponding to an input $(\mathbf{q_1}, \ldots, \mathbf{q_t})$. Otherwise, they do not gain any information. Thus, if the adversary makes q_v valid queries then they get q_v values $\sum_{i=1}^{2^d} T_i(x_{k,i})$ for each table. If $q_v < |I_i|$ then the adversary needs to guess the sum values for the $|I_i| - q_v$ remaining table inputs for each table. Since for any i, $(x_{1,i}, \ldots, x_{2^d,i}) \mapsto \sum_{i=1}^{2^d} T_i(x_{k,i})$ is a pseudo-random function, the probability that the adversary produces a functional pirate key generator, i.e. that the adversary guesses the unknown sum values is equal to $\left(\frac{1}{2^m}\right)^{t(|I_i|-q_v)}$. To ensure that the probability is negligible, it is sufficient to show that $|I_i| - q_v \geqslant 1$.

  Let X be the random variable that gives the number of valid queries among q. X follow the hypergeometric distribution with parameter $(q, p = \frac{|I_i|}{2^{m2^d}}, 2^{m2^d})$. Indeed, we calculate the probability of successes in q draws (q queries) without replacement from a population of size 2^{m2^d}. $p2^{m2^d}$ corresponds to the number of valid inputs (for one table), i.e. $|I_i|$. Thus, the probability of the event $\{X = q_v\}$ is given by: $\mathbb{P}(q_v) = \frac{\binom{|I_i|}{q_v}\binom{2^{m2^d}-|I_i|}{q-q_v}}{\binom{2^{m2^d}}{q}}$. If $q = |I_i|$ then $\mathbb{P}(|I_i|) = \frac{1}{\binom{2^{m2^d}}{q}}$. Hence, $q_v < |I_i|$ and $|I_i| - q_v \geqslant 1$.

  Now assume that the adversary adopts the second strategy, i.e. recovers all the table values $T_i(x)$. For simplicity, let us consider one table $T(x)$ (i.e. we omit the index i since the same technique can be applied independently to all tables).

  For a valid tuple $(x_1, \ldots, x_{2^d})$, the adversary knows the table outputs. Let s be the index of the adversary, for $1 \leqslant k \leqslant 2^d$, $1 \leqslant j \leqslant d$, we denote by $\alpha_{s,k,j}$ the unknown $u_{s,\mathbf{i_j}^{(\mathbf{x_k})}+jv+\frac{(i-1)L}{t}}$ and by $\beta_{k,j}$ the unknown $e_{s,\mathbf{i_j}^{(\mathbf{x_k})}+jv+\frac{(i-1)L}{t}}$ for a fixed i. The adversary constructs the following system of equations:

$$\begin{cases} T^{(u_s)}(x_1) = T(x_1) + \sum_{j=1}^{d} \alpha_{s,1,j} \cdot \beta_{1,j} \\ T^{(u_s)}(x_2) = T(x_2) + \sum_{j=1}^{d} \alpha_{s,2,j} \cdot \beta_{2,j} \\ \vdots \\ T^{(u_s)}(x_{2^d}) = T(x_{2^d}) + \sum_{j=1}^{d} \alpha_{s,2^d,j} \cdot \beta_{2^d,j} \end{cases}$$

Since $\alpha_{s,k,j} \in \{0,1\}$, the adversary can construct 2^{d2^d} systems of 2^d equations where there are at most $2^m + d2^m$ unknowns in the case where all the $\alpha_{s,k,j}$ are equal to 1. With $q_v < |I_i|$ valid inputs, each system is composed of at most 2^m equations with $2^m(\frac{d}{2}+1)$ unknowns[28].

The adversary can solve the correct system iff $2^m \geqslant 2^m(\frac{d}{2}+1) \Leftrightarrow \frac{d}{2}+1 \leqslant 1$ which is impossible for $d \geqslant 2$.

- $c > 1$. We assume that the adversary is a collusion of c users. Let $q_v < |I_i|$ be the number of valid queries. As above the adversary can store the sum values and guesses the other sum values. Hence, the probability to produce a functional pirate key generator is equal to $\left(\frac{1}{2^m}\right)^{t(|I_i|-q_v)}$.

  Besides, the adversary may leverage the coalition to retrieve the original values $T_i(x)$ for all i, x and thus reduce the storage size, this is the second strategy.

  For a tuple $(x_1, \ldots, x_{2^d})$, the adversary knows the table outputs of all members of $\mathcal{T}$. For $1 \leqslant s \leqslant c$, $1 \leqslant k \leqslant 2^d$, $1 \leqslant j \leqslant d$, we denote by $\alpha_{s,k,j}$ the unknown $u_{s,\mathbf{i_j}^{(\mathbf{x_k})}+jv+\frac{(i-1)L}{t}}$ and by $\beta_{k,j}$ the unknown $e_{s,\mathbf{i_j}^{(\mathbf{x_k})}+jv+\frac{(i-1)L}{t}}$ for a fixed i. The adversary constructs the following system of equations:

$$\begin{cases} T^{(u_1)}(x_1) = T(x_1) + \sum_{j=1}^{d} \alpha_{1,1,j} \cdot \beta_{1,j} \\ T^{(u_1)}(x_2) = T(x_2) + \sum_{j=1}^{d} \alpha_{1,2,j} \cdot \beta_{2,j} \\ \vdots \\ T^{(u_1)}(x_{2^d}) = T(x_{2^d}) + \sum_{j=1}^{d} \alpha_{1,2^d,j} \cdot \beta_{2^d,j} \\ \vdots \\ T^{(u_c)}(x_1) = T(x_1) + \sum_{j=0}^{d} \alpha_{c,1,j} \cdot \beta_{1,j} \\ \vdots \\ T^{(u_c)}(x_{2^d}) = T(x_{2^d}) + \sum_{j=0}^{d} \alpha_{c,2^d,j} \cdot \beta_{2^d,j} \end{cases}$$

As above, the adversary can construct 2^{cd2^d} systems of $2^d c$ equations where there are at most $2^m + d2^m$ unknowns and in average $2^m + \frac{d}{2}2^m$ unknowns. With $q_v < |I_i|$ valid inputs, each system is composed of at most $2^m c$ equations with $2^m(\frac{d}{2}+1)$ unknowns.

The adversary can solve the correct system iff $2^m c \geqslant 2^m(\frac{d}{2}+1) \Leftrightarrow c \geqslant \frac{d}{2}+1$.

□

Examples of parameters. Some parameters for implementation are given in Table 2.3.6.The block cipher E is the AES-128 and the control word length is 128 bits.

[28] Only one of these systems is correct and since there are approximatively as many as "1" values as "0" values in a codeword, the correct system has in average $2^m + \frac{d}{2}2^m$ unknowns.

t	d	m	L	c	WB size	Upper bound on q	Size of EB (bits)
8	2	16	2^{12}	1	2 MB	2^{63}	512
8	3	16	2^{10}	2	2 MB	2^{63}	1024
4	2	32	2^{19}	1	64 GB	2^{127}	512
4	3	32	2^{15}	2	64 GB	2^{127}	1024
4	4	32	2^{12}	2	64 GB	2^{127}	2048

Table 2.1: Examples of parameters: m is the table input length, t is the number of tables, d is the dimension, L is the code length and is defined as $L = td2^{\lceil m/d \rceil}$ and c is the collusion size. WB size is the total size of the key generator.

Conclusion of the chapter

In this chapter we have focused on the traceability property of white-box programs. The notion is tightly linked to the problem of traitor tracing since the purpose of a traceable white-box program is to identify any redistributed program or even a rogue program. Such a feature was introduced to mitigate code lifting in the context of Digital Rights Management. The main idea is that a white-box program should embed a *fingerprint* which is hardly removable by pirates. The traitor tracing scheme of [43] is a *Private Linear Broadcast Encryption* scheme adapted to the white-box model. However, the scheme cannot be instantiated with state-of-the-art white-box dedicated block ciphers because a PLBE scheme makes use of bilinear maps of composite order.

Our work aims at defining a traitor tracing system with white-box techniques, i.e. we use the popular tabulation technique to construct several distinct key generators from a unique secret key. Thus, the broadcaster can efficiently generate control words with a compact description of the key generator while the users use their incompressible key generators. Besides, the master key cannot be extracted from the lookup tables used by the key generators. Our scheme relies on a special code, the Tardos code, to personalize the key generators. In other words, each key generator embeds a Tardos codeword used to trace the white-box program in the future. Even if each user receives a personalized key generator the special structure of the enabling information enables all users to generate the correct control word as the broadcaster.

While several classes of traitor tracing schemes exist, ranging from combinatorial schemes to code-based schemes and public-key schemes, in the white-box context it seems that a code-based scheme is more suitable.

3

White-box implementation of the AES

> "No amount of experimentation can ever prove me right, a single experiment can prove me wrong."
>
> Albert Einstein

Contents

In this chapter, we describe a white-box implementation of the AES. It is mainly composed of lookup tables which merge the key addition within the substitution layer and of systems of multivariate polynomials that are evaluated to obtain the result of the permutation layer. Our white-box implementation resists the first-order Differential Computation Analysis and the Differential Fault Analysis. This chapter is based on a joint work with CEA List that leads to a candidate to the WhibOx 2019 competition. Our candidate arrived eighth among around a hundred of candidates.

Introduction

In Chapter 3, we introduced the WhibOx contest and especially the attacks that have been successful on the white-box implementations during the first edition. Recall that the WhibOx contest is a public contest first organized in 2016 to gather industrial and academic people who are interested in White-Box Cryptography. The goal of the competition was somehow to get a

confidence in the security of some potentially commercialized white-box implementations. The competition revealed two attacks which are actually not new because they belong to side-channel attacks but it turns out that they can be adapted to white-box implementations, namely Differential Computation Analysis (DCA) [27] and Differential Fault Analysis (DFA) [100, 106]. Those attacks were efficient on plenty of different white-box implementations, e.g. [35, 69, 116] because they do not require knowledge of the designs and are somehow generic. In particular, the Differential Fault Analysis exploits a characteristic of the AES which is the lack of MixColumns in the last round. Thus, without an appropriate countermeasure to DFA, any white-box implementation of the AES can be broken using this attack. The goal fixed upon the second edition of the contest launched in March 2019 was to propose implementations that at least resist the first edition's attacks and show good performance. The competition targeted both coders and attackers, i.e. one could either post a challenge program or break others programs. The rules of the competition are mainly the same as the first edition: a challenge program should be a C source-code of a white-box implementation of AES-128 under a freely chosen key. The code size should not exceed 50 MB and the execution time should be less than one second. In contrary to the first edition, the challenges should also resist inversion, i.e. it should be difficult to find the plaintext that corresponds to a ciphertext without the knowledge of the key: this is the one-wayness security not reached so far.

In [104], Rivain and Wang gave a theoretical analysis of DCA success and propose two DCA-like attacks which they called collision attack and mutual information analysis. As the original DCA, the attacks do not required knowledge of the white-box designs but are more efficient in term of trace complexity. Actually, the analysis of Rivain and Wang shows that the size of the internal encodings is determining for the success probability.

We took into account the analysis in [104] and proposed a white-box implementation that fulfills the requirements of the competition and resists the first-order DCA and the simple DFA. Even if we were aware of the collision attack of [104], we could not find a countermeasure that fits in the size constraint of the competition. However, we stress that the attack can be made impractical at a cost of the increase of the white-box size. Yet, for the competition we used code obfuscation to make the attacker's task harder. Since a DCA attack, as well as a collision attack require to identify a target variable that can be correlated to a key-dependent variable, these can be made more difficult with some techniques of code obfuscation.

3.1 Equivalent description of the encryption algorithm

State array. The two-dimensional array of bytes *state* is seen as a one-dimensional array of 16 bytes as follows:

$$state = \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \text{ becomes}$$

$$state = (s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0}, s_{0,1}, s_{1,1}, s_{2,1}, s_{3,1}, s_{0,2}, s_{1,2}, s_{2,2}, s_{3,2}, s_{0,3}, s_{1,3}, s_{2,3}, s_{3,3})$$

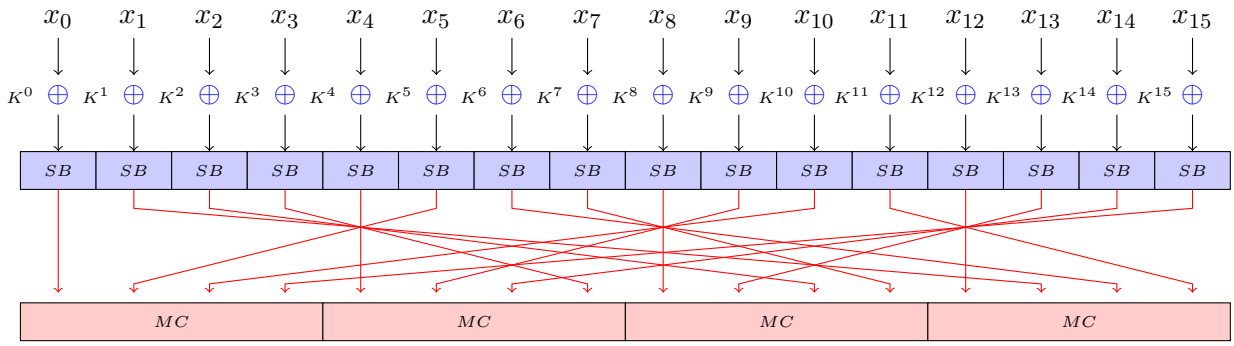


Figure 3.1: Equivalent description of one round of AES: in blue, the KeyTransform operation, in red, the Evaluate operation.

such that for any $0 \leqslant i, j \leqslant 3$: $s_{i+4j} \leftarrow s_{i,j}$.

One round of AES is composed of four basic operations: AddRoundKey, SubBytes, ShiftRows and MixColumns. We gather together AddRoundKey and SubBytes as well as ShiftRows and MixColumns to define two new transformations which we call respectively KeyTransform and Evaluate. As the name suggests, KeyTransform is a key-dependent transformation and is non-linear and Evaluate can be seen as a linear mapping, because MixColumns is $\mathbb{F}_{2^8}$-linear. The encryption algorithm of AES is then described in Algorithm 3 with the two new transformations. Figure 3.1 illustrates one round of the equivalent description.

Algorithm 3 Equivalent description of the encryption algorithm of AES-128.

$state \leftarrow plaintext$
for $r = 1 \dots 9$ **do**
  KeyTransform$(state, r-1)$
  Evaluate$(state)$
end for
KeyTransform$(state, 9)$
ShiftRows$(state)$
$ciphertext \leftarrow state$

KeyTransform makes call to key-dependent lookup tables pre-computed from the round keys. On the contrary, Evaluate does not depend on the secret key. We use the term "evaluate" to refer to the evaluation of systems of multivariate polynomials on the state variable. The details of the lookup tables calculation and polynomials coefficients calculation are given in the next section. The last two AddRoundKey operations, i.e. AddRoundKey$(state, K^9)$ and AddRoundKey$(state, K^{10})$ are combined together with the last SubBytes to define KeyTransform$(state, 9)$. KeyTransform$(state, 9)$ makes call to lookup tables that depend on both K^9 and K^{10}. The reason is that we exchange the positions of ShiftRows and AddRoundKey in the last round to compose AddRoundKey, SubBytes, and AddRoundKey. We can make the exchange if we apply the right permutation to the last round key K^{10}, i.e. the inverse of ShiftRows. Indeed, since ShiftRows is a linear transformation, ShiftRows$(state)$ followed by AddRoundKey$(state, K^{10})$ gives the same result as AddRoundKey$(state, \widehat{K}^{10})$ followed by ShiftRows$(state)$ where $\widehat{K}^{10}$ is the result of applying InvShiftRows to the round key K^{10}.

3.2 White-box implementation

Recall that a white-box implementation is an implementation of a cryptographic algorithm for fixed secret parameters. It is characterized by the stored data containing the secrets which will be used by the algorithm during the execution. Let K be a 128-bit secret key. After executing the key expansion algorithm, the secret parameters are the round keys $K^0, K^1, \ldots, K^{10}$ and some random encodings.

3.2.1 Quadratic encoding

An encoding enables to guarantee the "local security" in the sense of Chow et al. [35]. However, it has to be chosen carefully to not leak information about the secret key, information which is further exploited by some differential analysis [27, 107, 104]. The size of the encoding should be strictly higher than the input size of the key-dependent function. We introduce the definitions of two types of quadratic encodings which we further use.

Definition 82 (Quadratic polynomial)**.** *A quadratic polynomial over $\mathbb{F}_{2^n}$ is a polynomial of the form:*

$$\sum_{0 \leqslant i,j \leqslant n-1} a_{i,j} X^{2^i+2^j},\ a_{i,j} \in \mathbb{F}_{2^n}.$$

Definition 83 (Quadratic multivariate polynomial)**.** *A quadratic multivariate polynomial with n variables over $\mathbb{F}_2$ is of the form:*

$$\sum_{0 \leqslant i < j \leqslant n-1} \alpha_{i,j} x_i x_j + \sum_{0 \leqslant i \leqslant n-1} \beta_i x_i + \gamma,\ \alpha_{i,j}, \beta_i, \gamma \in \mathbb{F}_2.$$

A quadratic polynomial $P \in \mathbb{F}_{2^n}[X]$ can be seen as a vector of n multivariate quadratic polynomials with n variables.

Definition 84 (Permutation polynomial)**.** *A permutation polynomial $P \in \mathbb{F}_{2^n}[X]$ is a polynomial which permutes $\mathbb{F}_{2^n}$ under evaluation, i.e. the mapping $f\colon \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ such that for any $x \in \mathbb{F}_{2^n}$: $f(x) = P(x)$ is a bijection.*

Definition 85 (Dobbertin polynomial [48])**.** *Let m an integer and $n = 2m + 1$. The Dobbertin polynomial over $\mathbb{F}_{2^n}$ is a quadratic permutation polynomial of the form:*

$$X^{2^{m+1}+1} + X^3 + X.$$

Example 2.

- *$m = 2, n = 5$: $X^9 + X^3 + X$ is a Dobbertin polynomial over $\mathbb{F}_{2^5}$.*
- *$m = 3, n = 7$: $X^{17} + X^3 + X$ is a Dobbertin polynomial over $\mathbb{F}_{2^7}$.*
- *$m = 4, n = 9$: $X^{33} + X^3 + X$ is a Dobbertin polynomial over $\mathbb{F}_{2^9}$.*

Definition 86 (Type-1 quadratic encoding)**.** *Let $m > 8$. For any key $k \in \mathbb{F}_2^8$, let $\phi_k\colon \mathbb{F}_2^8 \to \mathbb{F}_2^8$ be a key-dependent function. For any $x \in \mathbb{F}_2^8$, let $y = \phi_k(x)$ be a sensitive variable depending on k. A type-1 quadratic encoding ϵ of size m is a mapping from $\mathbb{F}_2^8$ to $\mathbb{F}_2^m$ defined as*

$$\epsilon(y) = A(y \oplus q(\sigma(x)) || \sigma(x)) \tag{3.1}$$

where A is an invertible affine mapping from $\mathbb{F}_2^m$ to itself, q is a quadratic function from $\mathbb{F}_2^8$ to itself and σ is a bijection of $\mathbb{F}_2^8$.

Definition 87 (Concatenation of encodings)**.** *Let $\epsilon_1, \ldots, \epsilon_t$ be t quadratic encodings. The concatenation $\epsilon_0 || \ldots || \epsilon_{t-1}$ is a quadratic encoding from $(\mathbb{F}_2^8)^t$ to $(\mathbb{F}_2^m)^t$ defined as*

$$\forall (x_0, \ldots, x_{t-1}) \in (\mathbb{F}_2^8)^t,\ (\epsilon_0 || \ldots || \epsilon_{t-1})(x_0, \ldots, x_{t-1}) = (\epsilon_0(x_0), \ldots, \epsilon_{t-1}(x_{t-1})).$$

Definition 88 (Type-2 quadratic encoding)**.** *For any key $k = (k_0, \ldots, k_{j-1}) \in \mathbb{F}_2^{8j}$, let $\psi_k \colon \mathbb{F}_2^{8j} \to \mathbb{F}_2^8$ be a key-dependent function. For any $x = (x_0, \ldots, x_{j-1}) \in \mathbb{F}_2^{8j}$, let $z = \psi_k(x)$ be a sensitive variable depending on k. Let $\overline{y} = (\epsilon_0 || \ldots || \epsilon_{j-1})(\phi_{k_0}(x_0), \ldots, \phi_{k_{j-1}}(x_{j-1}))$ be an encoded variable. A type-2 quadratic encoding ζ of size m' is a mapping from $\mathbb{F}_2^{8j}$ to $\mathbb{F}_2^{m'}$ defined as*

$$\zeta(z) = A(z \oplus q(l(\overline{y})) || \sigma(l(\overline{y}))) \tag{3.2}$$

where A is an invertible affine mapping from $\mathbb{F}_2^{m'}$ to itself, l is a linear mapping from $\mathbb{F}_2^{m'j}$ to $\mathbb{F}_2^{m'-8}$, q is a quadratic function from $\mathbb{F}_2^{m'-8}$ to $\mathbb{F}_2^8$ and σ is a quadratic bijection of $\mathbb{F}_2^{m'-8}$ described by the Dobbertin permutation polynomial over $\mathbb{F}_{2^{m'-8}}[X]$.

For both types of encoding, A and q are secret. σ is secret for type-1 and public for type-2. l is secret for type-2. Consequently, it is hard to compute y or z knowing respectively $\epsilon(y)$ and $\zeta(z)$. However, when the secret mappings are known, it is easy to "invert" the encodings. We define below the notion of "inverse encoding".

Proposition 11 (Inverse encoding)**.**

- *Type-1: Let A and q be secret mappings and ϵ be a type-1 encoding. For any encoded variable $\overline{y} = \epsilon(y)$, the mapping $\overline{y} \mapsto \lceil A^{-1}(\overline{y}) \rceil_8 \oplus q(\lfloor A^{-1}(\overline{y}) \rfloor_8)$ is the inverse of ϵ.*
- *Type-2: Let A, q, l be secret mappings and ζ be a type-2 encoding. For any encoded variable $\overline{z} = \zeta(z)$, the mapping $\overline{z} \mapsto \lceil A^{-1}(\overline{z}) \rceil_8 \oplus q(\sigma^{-1}(\lfloor A^{-1}(\overline{z}) \rfloor_{m'-8}))$ is the inverse of ϵ.*

The inverse encoding of $\epsilon_0 || \ldots || \epsilon_{t-1}$ is the concatenation of the inverse encodings $\epsilon_0^{-1} || \ldots || \epsilon_{t-1}^{-1}$.

We respectively denote by ϵ^{-1} and ζ^{-1} the inverse encodings of ϵ and ζ.

The encodings are used in the next sections to mask the lookup tables values and the systems of polynomials used by respectively KeyTransform and Evaluate. In other words, type-1 encodings are used as output encodings for KeyTransform and type-2 encodings are used as output encodings for Evaluate. Since the two transformations come one after the other, type-1 encodings become input encodings for Evaluate and type-2 encodings become input encodings for KeyTransform.

3.2.2 Lookup table calculation

For each round, the KeyTransform operation makes call to 16 lookup tables, i.e. each table depends on one byte of the round key. The lookup tables called the *T-boxes* are defined as follows: for any $x \in \mathbb{F}_2^8$

$$\begin{aligned} \mathsf{T}_i^r[x] &= \mathsf{SB}(x \oplus K_i^{r-1}) \text{ for } 0 \leqslant i \leqslant 15 \text{ and } 1 \leqslant r \leqslant 9 \\ \mathsf{T}_i^{10}[x] &= \mathsf{SB}(x \oplus K_i^9) \oplus \widehat{K}_i^{10} \text{ for } 0 \leqslant i \leqslant 15 \end{aligned}$$

The values of the table are encoded such that the tables do not leak key information. To mask a byte of a sensitive variable, m bits are stored. Thus, the encoded T-boxes are defined as follows:

$$\begin{aligned}
\overline{\mathsf{T}_i^1}[x] &= \epsilon_i^1(\mathsf{T}_i^1[x]) \text{ for any } x \in \mathbb{F}_2^8 \\
\overline{\mathsf{T}_i^r}[x] &= \epsilon_i^r(\mathsf{T}_i^r[(\zeta_i^{r-1})^{-1}(x)]) \text{ for any } x \in \mathbb{F}_2^m \text{ and for } 2 \leqslant r \leqslant 9 \\
\overline{\mathsf{T}_i^{10}}[x] &= \mathsf{T}_i^{10}[(\zeta_i^9)^{-1}(x)] \text{ for any } x \in \mathbb{F}_2^m
\end{aligned}$$

An encoded T-box maps m bits to m bits except for $r = 1$ for which the T-box maps 8 bits to m bits and for $r = 10$ for which the T-box maps m bits to 8 bits.

For each round r, $1 \leqslant r \leqslant 9$, Algorithm 4 computes 16 lookup tables.

Algorithm 4 Lookup tables calculation for a fixed round key K^{r-1} and secret encodings $(\epsilon_0^r || \dots || \epsilon_{15}^r)$ and $((\zeta_0^{r-1})^{-1} || \dots || (\zeta_{15}^{r-1})^{-1})$.

$\overline{\mathsf{T}_0^r} \leftarrow [\,], \dots, \overline{\mathsf{T}_{15}^r} \leftarrow [\,]$
for $x = 0 \dots 2^m - 1$ **do**
  $state \leftarrow ((\zeta_0^{r-1})^{-1} || \dots || (\zeta_{15}^{r-1})^{-1})(x)$
  AddRoundKey$(state, K^r)$
  SubBytes$(state)$
  $state \leftarrow (\epsilon_0^r || \dots || \epsilon_{15}^r)(state)$
  for $i = 0 \dots 15$ **do**
    $\overline{\mathsf{T}_i^r}[x] \leftarrow s_i$
  end for
end for

Algorithms 5 and 6 compute each 16 lookup tables respectively for $r = 1$ and $r = 10$.

Algorithm 5 Lookup tables calculation for a fixed round key K^0 and secret encodings $(\epsilon_0^1 || \dots || \epsilon_{15}^1)$.

$\overline{\mathsf{T}_0^1} \leftarrow [\,], \dots, \overline{\mathsf{T}_{15}^1} \leftarrow [\,]$
for $x = 0 \dots 2^8 - 1$ **do**
  AddRoundKey$(state, K^0)$
  SubBytes$(state)$
  $state \leftarrow (\epsilon_0^1 || \dots || \epsilon_{15}^1)(state)$
  for $i = 0 \dots 15$ **do**
    $\overline{\mathsf{T}_i^1}[x] \leftarrow s_i$
  end for
end for

3.2.3 Description of a system of multivariate polynomials

By definition, Evaluate corresponds to the application of ShiftRows and MixColumns to the intermediate state. It is described by systems of quadratic multivariate polynomials with coefficients in $\mathbb{F}_2$ that are evaluated on the intermediate state. Recall that for any $(x_0, x_1, x_2, x_3) \in (\mathbb{F}_2^8)^4$, MixColumns applies the following mapping: $(x_0, x_1, x_2, x_3) \mapsto \mathsf{MC} \cdot (x_0, x_1, x_2, x_3)^\top$ to each column of the state. Thus, MixColumns can be expressed with 4 multivariate polynomials $p_i \in \mathbb{F}_2^8[X_0, X_1, X_2, X_3]$ for $0 \leqslant i \leqslant 3$ defined as $p_i(x_0, x_1, x_2, x_3) = \sum_{j=0}^{3} mc_{i,j} \otimes x_j$. Each polynomial is $\mathbb{F}_{2^8}$-linear and is isomorph to a linear application of $\mathbb{F}_2^{32}$ (by defining an isomorphism from

Algorithm 6 Lookup tables calculation for the round keys K^9, K^{10} and secret encodings $((\zeta_0^9)^{-1}||\dots||(\zeta_{15}^9)^{-1})$.

$\overline{\mathsf{T}_0^{10}} \leftarrow [\,], \dots, \overline{\mathsf{T}_{15}^{10}} \leftarrow [\,]$
for $x = 0 \dots 2^m - 1$ **do**
  $state \leftarrow ((\zeta_0^9)^{-1}||\dots||(\zeta_{15}^9)^{-1})(x)$
  $\mathsf{AddRoundKey}(state, K^9)$
  $\mathsf{SubBytes}(state)$
  $\mathsf{AddRoundKey}(state, \widehat{K}^{10})$
  for $i = 0 \dots 15$ **do**
    $\overline{\mathsf{T}_i^{10}}[x] \leftarrow s_i$
  end for
end for

$(\mathbb{F}_{2^8})^4$ to $\mathbb{F}_2^{32}$). Thereby, MixColumns can be described with a system of 4×8 linear polynomials $p_{i,j} \in \mathbb{F}_2[X_0, \dots, X_{31}]$, for $0 \leqslant i \leqslant 3$, $0 \leqslant j \leqslant 7$ as follows:

$$\mathscr{S} = \begin{cases} p_{0,0}(x_0, \dots, x_{31}) \\ \vdots \\ p_{0,7}(x_0, \dots, x_{31}) \\ p_{1,0}(x_0, \dots, x_{31}) \\ \vdots \\ p_{1,7}(x_0, \dots, x_{31}) \\ p_{2,0}(x_0, \dots, x_{31}) \\ \vdots \\ p_{2,7}(x_0, \dots, x_{31}) \\ p_{3,0}(x_0, \dots, x_{31}) \\ \vdots \\ p_{3,7}(x_0, \dots, x_{31}) \end{cases} \tag{3.3}$$

The result of ShiftRows followed by MixColumns is the result of the evaluation of $\mathscr{S}$ on four 32-bit input variables where each 32-bit input corresponds to 4 bytes of the state after the permutation ShiftRows. In other words, $\mathscr{S}$ corresponds to the MixColumns transformation on one column of the state.

For any round $1 \leqslant r \leqslant 9$, an encoded system $\overline{\mathscr{S}^r} = \overline{\mathscr{S}_0^r}||\overline{\mathscr{S}_1^r}||\overline{\mathscr{S}_2^r}||\overline{\mathscr{S}_3^r}$ is evaluated on an encoded input and is defined as

$$\overline{\mathscr{S}^r} = (\zeta_0^r||\dots||\zeta_{15}^r) \circ (\mathscr{S}||\mathscr{S}||\mathscr{S}||\mathscr{S}) \circ ((\epsilon_0^{r-1})^{-1}||\dots||(\epsilon_{15}^{r-1})^{-1}) \tag{3.4}$$

For any $0 \leqslant i' \leqslant 3$, $\overline{\mathscr{S}_{i'}^r}$ is a system of $4m'$ quadratic polynomials with $4m$ variables and with coefficients in $\mathbb{F}_2$. The degree of the polynomials is 2 because of the quadratic encodings. For $0 \leqslant i \leqslant 3$ and $0 \leqslant j \leqslant m'$, let $\overline{p_{i,j}^r}$ be a polynomial of $\overline{\mathscr{S}_{i'}^r}$. Then,

$$\overline{p_{i,j}^r}(x_0, \dots, x_{4m-1}) = \sum_{u<v} \alpha_{u,v}^r x_u x_v + \sum_{u=0}^{4m-1} \beta_u^r x_u + \gamma^r \text{ with } \alpha_{u,v}^r, \beta_u^r, \gamma^r \in \mathbb{F}_2. \tag{3.5}$$

Round	Type-1 encoding size m	Type-2 encoding size m′	Number of LUTs	Number of polynomials	Data size
1	24	17	16	16×17	166.53 kB
2	24	13	16	16×13	6.12 MB
3 to 7	13	13	16	16×13	1.20 MB
8	4×13	13	16	$16 \times 4 \times 4$	2.98 MB
9	24	17	16×4	16	3.6 MB
10	-	-	16	-	2 MB
				Total	16.1 MB

Table 3.1: Parameters and size of data to be stored: the size of a n-to-m bits lookup table is $2^n \times m$ bits and the size of a quadratic multivariate polynomial with n variables is $\frac{n(n+1)}{2} + 1$ bits.

Determining coefficients. For any r, i, j, notice that $p^r_{i,j}(0, \ldots, 0) = \gamma^r$ and that $p^r_{i,j}(1, \ldots, 0) = \beta^r_0 \oplus \gamma^r$. Thus, $\beta^r_0 = p^r_{i,j}(1, \ldots, 0) \oplus p^r_{i,j}(0, \ldots, 0)$. In the same way, $p^r_{i,j}(1, 1, \ldots, 0) = \alpha^r_{0,1} \oplus \beta^r_0 \oplus \beta^r_1 \oplus \gamma^r$. Thus, $\alpha^r_{0,1} = p^r_{i,j}(1, 1, \ldots, 0) \oplus p^r_{i,j}(1, \ldots, 0) \oplus p^r_{i,j}(0, 1, \ldots, 0) \oplus p^r_{i,j}(0, \ldots, 0)$.

More generally, for any u, v:

$$\begin{cases} \beta^r_u = p^r_{i,j}(0, \ldots, x_u, \ldots, 0) \oplus p^r_{i,j}(0, \ldots, 0) \text{ where } x_u = 1 \\ \alpha^r_{u,v} = p^r_{i,j}(0, \ldots, x_u, \ldots, x_v, \ldots, 0) \oplus p^r_{i,j}(0, \ldots, x_u, \ldots, 0) \oplus p^r_{i,j}(0, \ldots, x_v, \ldots, 0) \oplus p^r_{i,j}(0, \ldots, 0) \\ \text{where } x_u = x_v = 1 \end{cases}$$

3.3 Security analysis

3.3.1 Differential Computation Analysis

A Differential Computation Analysis (DCA) is a white-box attack inspired from Differential Power Analysis (DPA). As DPA, it exploits some information leakage about the secret key. In particular DCA exploits computation traces like, for instance, memory accesses. It is particularly efficient on white-box implementations since an attacker only needs to have access to the plaintext and/or ciphertext. Thus, the attacker does not need to know the implementation details and only has to identify a key-dependent operation where an information leaks. In the case of AES, the key-dependent operation targeted is $\mathsf{SB}(x \oplus K^0_i)$. Since the operation depends on only one byte of the secret key, the attacker can easily exhaust all possible values of the key byte. Thus, since the attacker controls the input value x, they are able to predict the value of $\mathsf{SB}(x \oplus k^*)$ for any key guess k^*. The attacker then tries to correlate their *prediction* with the corresponding

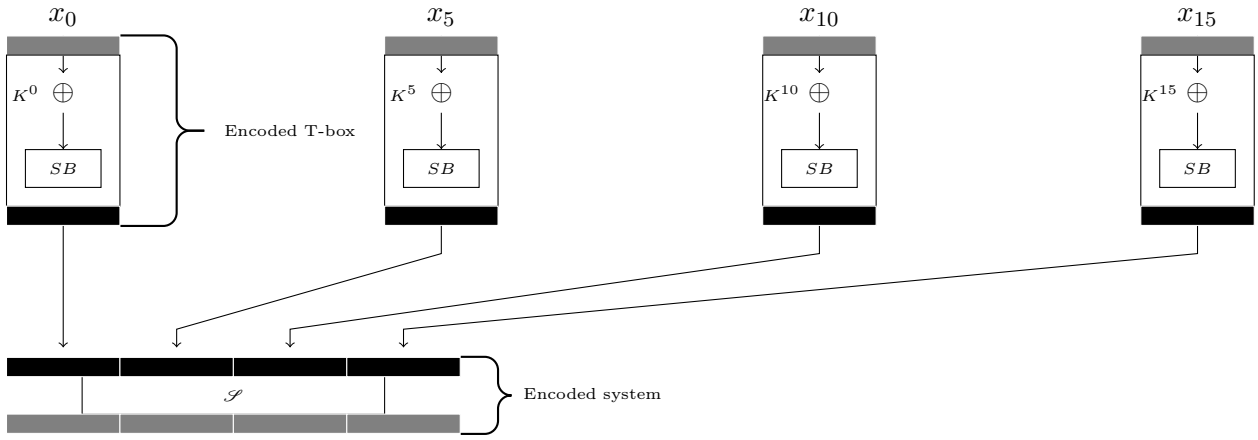


Figure 3.2: One encoded round.

encoded value. Actually, if the encoded value leaks information about the secret key then the attacker obtains a correlation peak for a right key guess.

A DCA attack generally consists in defining a distinguisher based on correlation scores. The DCA distinguisher for a key guess k^* is defined as the maximal absolute value of the correlation between the i-th bit of the predicted sensitive value and each trace sample. Thus, for DCA to succeed, the correlation score for a right key guess should be strictly greater than all other scores. For a key-dependent function with an n-bit input, this event happens when $n \geqslant 2m + 2$.

The key-dependent transformation KeyTransform does not leak any information about the secret key because the size of the encoding is strictly greater than the size of the sensitive variable so there is no leakage from each T-box.

In the same way, the composition of KeyTransform and Evaluate does not leak information about the secret key. Actually, if we consider one byte after the first round (after AddRoundKey, SubBytes, ShiftRows and MixColumns), we define a key-dependent function $\psi_k \colon (\mathbb{F}_2^8)^4 \to \mathbb{F}_2^8$ as follows: for any $k = (k_0, k_1, k_2, k_3) \in (\mathbb{F}_2^8)^4$ and any $x = (x_0, x_1, x_2, x_3) \in (\mathbb{F}_2^8)^4$

$$\psi_k(x) = \sum_{j=0}^{4} mc_{i',j} \otimes \phi_{k_j}(x_j) \text{ where } \phi_{k_j}(x_j) = \mathsf{SB}(x_j \oplus k_j).$$

ψ_k is a non-injection but we have $32 < 2m + 2$ and so a DCA attack will probably fail.

3.3.2 Collision attack

A collision attack is a DCA-like attack that exploits collision on sensitive values. As a traditional DCA attack, it requires recording computation traces and consists in defining a distinguisher based on correlation scores. Actually, the attack is based on the following property: *if a pair of encoded variables collides then the same collision occurs on the corresponding sensitive variables.*

A collision attack generally targets the output of the first MixColumns since each byte of the output depends on 4 bytes of the plaintext. So, there will necessary be collisions.

Let ψ_k be the key-dependent function that gives the i'-th byte after one round of the transformations AddRoundKey, SubBytes, ShiftRows and MixColumns. For any input $x = (x_0, x_1, x_2, x_3) \in (\mathbb{F}_2^8)^4$, an encoded variable is defined as $\overline{z} = \zeta_{i'}^1 \circ \psi_k(x) = \zeta_{i'}^1(z)$.

Proposition 12. *For any $x \neq x'$, $\overline{z} = \overline{z'}$ iff $\psi_k(x) = \psi_k(x')$.*

Proof. It is trivial that if $\psi_k(x) = \psi_k(x')$ then $\zeta^1_{i'} \circ \psi_k(x) = \zeta^1_{i'} \circ \psi_k(x')$.

For any $x \neq x'$, if $\overline{z} = \overline{z'}$, then $A(z \oplus q(l(\overline{y}))||\sigma(l(\overline{y}))) = A(z \oplus q(l(\overline{y'}))||\sigma(l(\overline{y'})))$ where $\overline{y} = (\epsilon^1_0|| \dots ||\epsilon^1_3)(\phi_{k_0}(x_0), \dots, \phi_{k_3}(x_3))$ and $\overline{y'} = (\epsilon^1_0|| \dots ||\epsilon^1_3)(\phi_{k_0}(x'_0), \dots, \phi_{k_3}(x'_3))$.

$$\begin{aligned} A(z \oplus q(l(\overline{y}))||\sigma(l(\overline{y}))) &= A(z \oplus q(l(\overline{y'}))||\sigma(l(\overline{y'}))) && (3.6)\\ \implies L_A(z \oplus q(l(\overline{y}))||\sigma(l(\overline{y}))) &= L_A(z' \oplus q(l(\overline{y'}))||\sigma(l(\overline{y'}))) && (3.7)\\ \implies L_A((z \oplus q(l(\overline{y}))||\sigma(l(\overline{y}))) &\oplus (z' \oplus q(l(\overline{y'}))||\sigma(l(\overline{y'}))) = 0 && (3.8) \end{aligned}$$

$$\implies \begin{cases} z \oplus q(l(\overline{y})) = z' \oplus q(l(\overline{y'})) \\ \sigma(l(\overline{y})) = \sigma(l(\overline{y'})) \end{cases} \implies \begin{cases} z \oplus q(l(\overline{y})) = z' \oplus q(l(\overline{y'})) \\ l(\overline{y}) = l(\overline{y'}) \end{cases} \implies \begin{cases} z = z' \\ q(l(\overline{y})) = q(l(\overline{y'})) \end{cases} \quad (3.9)$$

□

Let $\mathcal{S}$ be a set of pairs of inputs (x, x') and N be the number of corresponding traces, i.e. $N = |\mathcal{S}|$. We assume that each sample matches an encoded value. For a key guess k^*, we have:

$$\frac{1}{N} \sum_{(x,x') \in \mathcal{S}} (-1)^{\zeta^1_{i'}(z) \odot \zeta^1_{i'}(z') \oplus \psi_{k^*}(x) \odot \psi_{k^*}(x')} = 1 \text{ iff } k^* = k.$$

Complexity. The complexity of the collision attack is characterized by the trace complexity, i.e. the number of traces needed to distinguish the right key guess and the number of operations needed to compute the correlation scores.

According to Proposition 5, the probability of success p of a collision-based DCA for N traces is such that : $p \geqslant 1 - |\mathbb{K}| \exp -\frac{(N-1)(N-2)}{2^{m+1}}$, where $|\mathbb{K}|$ is the key search space. For a probability of success greater than $1 - 2^{-\lambda}$, the number of traces N is such that:

$$N \leqslant \sqrt{2^{m+1}(\lambda \ln 10 + \ln |\mathbb{K}|)}.$$

For $m = 17$, $\mathbb{K} = (\mathbb{F}_2^8)^4$, and $\lambda = 64$, we get $N \leqslant 6667 < 2^{13}$.

The number of operations needed to compute the correlation scores is at most $2^{32} \times 2^{13} = 2^{45}$.

We conclude that we cannot avoid the collision-based DCA attack, except if the encoding size is increased.

3.4 Our countermeasure to DFA

The idea here is to find a suitable DFA countermeasure that does not substantially increase the size of data stored. We describe a technique that is both time and space redundancies. It belongs to time redundancy because a function is executed several times on the same input and also belongs to space redundancy because we add redundancy to some encoded values. Actually, the redundancy corresponds to the same byte of the state encoded with different encodings.

Since a DFA generally consists in modifying a variable between the last two MixColumns, we add redundancy to the lookup tables called by KeyTransform in the 8-th round. For each sensitive variable, four encoded versions of the variable are stored in the lookup tables. The transformation Evaluate that follows is also modified to check that the sensitive variables obtained after applying the inverse encodings are the same. Then, the outputs of Evaluate and the following KeyTransform are also redundant and checked by Evaluate in the 9-th round.

KeyTransform in the 8-th round. Let T_i^8 be the lookup tables called by KeyTransform in the 8-th round:

$$\mathsf{T}_i^8[x] = \mathsf{SB}(x \oplus K_i^7).$$

The encoded lookup tables become:

$$\overline{\mathsf{T}_i^8}[x] = (\epsilon_{i,0}^8||\epsilon_{i,1}^8||\epsilon_{i,2}^8||\epsilon_{i,3}^8)(\mathsf{T}_i^8[(\zeta_i^7)^{-1}(x)])$$

where $\epsilon_{i,i'}^8$ are 8-to-m bit encodings. Thus, $\overline{\mathsf{T}_i^8}[x]$ maps m bits to $4m$ bits.

Evaluate now evaluates an encoded system $\overline{\mathscr{S}^8} = \overline{\mathscr{S}_0^8}||\overline{\mathscr{S}_1^8}||\overline{\mathscr{S}_2^8}||\overline{\mathscr{S}_3^8}$ where each $\overline{\mathscr{S}_{i'}^8}$ is a system of $16m$ quadratic polynomials with $16m$ variables and with coefficients in $\mathbb{F}_2$.

MixColumns with an error detection. Given a vector of 4 bytes (x_0, x_1, x_2, x_3), the 4 bytes output (y_0, y_1, y_2, y_3) of MixColumns is defined as:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 02 \otimes x_0 \oplus 03 \otimes x_1 \oplus x_2 \oplus x_3 \\ x_0 \oplus 02 \otimes x_1 \oplus 03 \otimes x_2 \oplus x_3 \\ x_0 \oplus x_1 \oplus 02 \otimes x_2 \oplus 03 \otimes x_3 \\ 03 \otimes x_0 \oplus x_1 \oplus x_2 \oplus 02 \otimes x_3 \end{pmatrix}$$

Let $(c_0^0, \ldots, c_{15}^0)$, $(c_0^1, \ldots, c_{15}^1)$, $(c_0^2, \ldots, c_{15}^2)$, $(c_0^3, \ldots, c_{15}^3)$ be four 16-byte vectors such that for any $0 \leqslant i' \leqslant 3$ and any $0 \leqslant j \leqslant 3$:

$$\sum_{i=4j}^{4j+3} c_i^{i'} = \begin{cases} 1 \text{ if } j = i' \\ 0 \text{ otherwise} \end{cases}$$

The 4×16 matrix M such that $M = (M_0|M_1|M_2|M_3)$ with

$$M_{i'} = \begin{pmatrix} mc_{0,i'} \otimes \sum_{i=0}^{3} c_i^{i'} & mc_{0,i'} \otimes \sum_{i=4}^{7} c_i^{i'} & mc_{0,i'} \otimes \sum_{i=8}^{11} c_i^{i'} & mc_{0,i'} \otimes \sum_{i=12}^{15} c_i^{i'} \\ mc_{1,i'} \otimes \sum_{i=0}^{3} c_i^{i'} & mc_{1,i'} \otimes \sum_{i=4}^{7} c_i^{i'} & mc_{1,i'} \otimes \sum_{i=8}^{11} c_i^{i'} & mc_{1,i'} \otimes \sum_{i=12}^{15} c_i^{i'} \\ mc_{2,i'} \otimes \sum_{i=0}^{3} c_i^{i'} & mc_{2,i'} \otimes \sum_{i=4}^{7} c_i^{i'} & mc_{2,i'} \otimes \sum_{i=8}^{11} c_i^{i'} & mc_{2,i'} \otimes \sum_{i=12}^{15} c_i^{i'} \\ mc_{3,i'} \otimes \sum_{i=0}^{3} c_i^{i'} & mc_{3,i'} \otimes \sum_{i=4}^{7} c_i^{i'} & mc_{3,i'} \otimes \sum_{i=8}^{11} c_i^{i'} & mc_{3,i'} \otimes \sum_{i=12}^{15} c_i^{i'} \end{pmatrix}$$

is a MixColumns control matrix, i.e. a matrix that calculates the MixColumns transformation if and only if an error is not detected.

Given a redundant vector $(x_0, \ldots, x_{15})$ such that $\begin{cases} x_0 = x_1 = x_2 = x_3 \\ x_4 = x_5 = x_6 = x_7 \\ x_8 = x_9 = x_{10} = x_{11} \\ x_{12} = x_{13} = x_{14} = x_{15} \end{cases}$, we have

$$M \cdot \begin{pmatrix} x_0 \\ \vdots \\ x_{15} \end{pmatrix} = \begin{pmatrix} 02 \otimes x_0 \oplus 03 \otimes x_1 \oplus x_2 \oplus x_3 \\ x_0 \oplus 02 \otimes x_1 \oplus 03 \otimes x_2 \oplus x_3 \\ x_0 \oplus x_1 \oplus 02 \otimes x_2 \oplus 03 \otimes x_3 \\ 03 \otimes x_0 \oplus x_1 \oplus x_2 \oplus 02 \otimes x_3 \end{pmatrix}$$

Evaluate in the 8-th round. The transformation $(x_0, \ldots, x_{15}) \mapsto M \cdot (x_0, \ldots, x_{15})^\top$ is $\mathbb{F}_{2^8}$-linear in 16 variables and can be described by a system $\mathscr{S}$ of 16×8 linear polynomials $p_{i,j} \in \mathbb{F}_2[X_1, \ldots, X_{128}]$.

The 8-th round encoded system is defined as $\overline{\mathscr{S}^8} = \overline{\mathscr{S}_0^8} || \overline{\mathscr{S}_1^8} || \overline{\mathscr{S}_2^8} || \overline{\mathscr{S}_3^8}$ where for any $0 \leqslant i' \leqslant 3$:

$$\overline{\mathscr{S}_{i'}^8} = (\zeta_{4i',0}^8 || \ldots || \ldots || \zeta_{4i'+3,3}^8) \circ \mathscr{S} \circ ((\epsilon_{i_0,0}^8)^{-1} || \ldots || (\epsilon_{i_3,3}^8)^{-1})$$

with $i_0 = \mathsf{SR}(4i')$, $i_1 = \mathsf{SR}(4i'+1)$, $i_2 = \mathsf{SR}(4i'+2)$, $i_3 = \mathsf{SR}(4i'+3)$.

Since the output of Evaluate in the 8-th round is redundant, the KeyTransform transformation of the 9-th round should take it into account. Consequently, the number of lookup tables in the 9-th round are increased fourfold. Each of the four copies calculates the same sensitive value, namely $\mathsf{SB}(x \oplus K_i^8)$ but the input and output encodings are distinct. So, an attacker cannot determinate from the encoded form if some lookup tables calculate the same sensitive variable or not.

KeyTransform in the 9-th round. The encoded lookup tables called by KeyTransform in the 9-th round are defined as:

$$\overline{\mathsf{T}_{i,i'}^9}[x] = (\epsilon_{i,0}^9 || \epsilon_{i,1}^9 || \epsilon_{i,2}^9 || \epsilon_{i,3}^9)(\mathsf{T}_i^9[(\zeta_{i,i'}^8)^{-1}(x)])$$

for $0 \leqslant i \leqslant 15$, $0 \leqslant i' \leqslant 3$.

Evaluate in the 9-th round. The redundancy is removed by the encoded system in the 9-th round, i.e. the sensitive variables are not quadrupled when encoded. The encoded system of the 9-th round is defined as $\overline{\mathscr{S}^9} = \overline{\mathscr{S}_0^9} || \overline{\mathscr{S}_1^9} || \overline{\mathscr{S}_2^9} || \overline{\mathscr{S}_3^9}$ where:

$$\overline{\mathscr{S}_{i'}^9} = (\zeta_{4i'}^9 || \ldots || \zeta_{4i'+3}^9) \circ \mathscr{S} \circ ((\epsilon_{i_0,0}^9)^{-1} || \ldots || (\epsilon_{i_0,3}^9)^{-1} || \ldots || (\epsilon_{i_3,0}^9)^{-1} || \ldots || (\epsilon_{i_3,3}^9)^{-1}).$$

To summarize, our white-box implementation of the AES-128 comprises two structures: encoded key-dependent lookup tables and encoded systems of quadratic multivariate polynomials.

The systems of multivariate polynomials are implemented by lookup tables which store the coefficients of the polynomials and an *evaluation function* which evaluates the result of the systems given an input and with respect to the coefficients. Our white-box implementation integrates a DFA countermeasure which makes the DFA described in Chapter 3 of the first part of this thesis unsuccessful.

3.5 Submission to the WhibOx competition

We submitted our white-box implementation of the AES to the WhibOx context Edition 2. The competition requires a C source code of the AES-128 of size at most 50 MB. The size of the executable is limited to 20 MB and the execution time must be less than 1 second. Our code is an obfuscated version of the described white-box implementation. The size of the source code and the executable are approximatively 44 MB and 17 MB respectively. The execution time is around 40 ms. The source code can be downloaded at `https://whibox.cyber-crypt.com/show/candidate/93.c`.

Our candidate named brave_swanson has been broken in 13 days with a final score of 373.21 which puts us in eighth place in the contest rankings. The score takes into account both the performance and the number of days passed. The performance is measured in terms of average running time, code size and memory consumption. The rankings can be found at `https://whibox-contest.github.io/2019/`.

Conclusion of the chapter

In response to the WhibOx contest, we proposed a new approach for a white-box implementation of the AES. Our method aims to prevent state-of-the-art attacks like the BGE attack [12] or the collision attack based on the algebraic degree [82]. Yet, our main concern remained the Differential Computation Analysis and the Differential Fault Analysis. While the output of the first SubBytes operation can be protected from a DCA by increasing the size of the encodings, the output of first MixColumns is still vulnerable to a DCA-like attack exploiting internal collisions. One solution to this issue would be to increase the size of the encodings such that it is sufficiently greater than the input size, namely greater than 32. However, such a wide encoding results in a wide lookup table in the second round and would then be impractical. Another approach would be to use a structure that makes an encoded round output dependent of all bits of the input. Of course this cannot be achieved with lookup tables. Thus, if the system of multivariate polynomials used to compute the permutation layer of the AES is extended to the substitution layer, then any DCA-like attack will fail. Nevertheless, such approach arises several issues. First, the high algebraic degree of the AES S-box makes it hard to represent the substitution layer with multivariate polynomials without substantially increasing the size of the data to be stored. This approach of representing one round of the AES with multivariate polynomials over $\mathbb{F}_{2^8}$ has already been investigated in [29] and resulted in a 568 MB white-box implementation. In order to reduce the number of coefficients, we suggest to decompose the AES S-box into low degree functions. [97] showed that the AES S-box can be decomposed into quadratic and cubic functions. In the light of their work, the possibility to decompose the AES S-box with only quadratic functions remains open. Second, since the substitution layer is made of several small S-boxes, an encoded system might be vulnerable to a structural attack like [44].

Conclusion: White-Box Cryptography, we are here

The main purpose of Cryptography is to provide confidentiality, integrity, authentication, and non-repudiation of exchanged data in many applications. Initially dedicated to military applications, modern uses of Cryptography include electronic commerce, chip-based payment cards, digital currencies, Digital Rights Management, etc. While cryptographic algorithms are generally designed to be executed in a secure environment, the spread of software applications boosted by the breakthrough of the computer and mobile devices tends to invalidate the traditional black-box model. In practice, cryptographic algorithms might be implemented in hardware or software and executed on an untrusted open platform. The White-Box Attack Context also known as the white-box model is characterized by an attacker having a full access of the software implementation of a cryptographic algorithm and a full control over its execution environment. The definition of this new model leads to the emergence of White-Box Cryptography which should be understood as "the science of creating systems in the white-box model".

In this thesis, we studied the opportunities offered by White-Box Cryptography. We have investigated three approaches. First, in light of recent works we studied the possibility of designing a secure white-box implementation of the AES. Second, we studied the more theoretical approach for White-Box Cryptography. Finally, we proposed an application-oriented approach which focuses on specific use cases of White-Box Cryptography.

White-box implementation of the AES.

In Chapter 2 of the first part of this thesis, we gave an overview of state-of-the-art white-box implementations of the AES with the corresponding published attacks. We have seen that the type of encodings as well as some specificities of the AES are exploited to extract the secret key from the white-box implementation. For instance, the BGE attack exploits two main vulnerabilities: first, the attacker has access to an encoded version of each byte of the state, and they know that the encoding is a fixed small bijection; second, the attacker has access to encoded subrounds which allow them to define bijective mappings induced by the specification of the AES. In the same flavour, the collision algebraic attack exploits the same encoded subrounds as the BGE attack and the type of encodings, namely a 8-bit bijection composed of two 4-bit bijections.

The cryptanalysis of the polynomial-based white-box implementation mainly exploits the characteristics of the secret linear encodings between successive rounds. Even if the encodings are wide, the matrix have a sparse structure which leaks information. Moreover, the specificity of the AES is used in the cryptanalysis to construct an equivalent implementation of the algorithm. Even if the use of large encodings is recommended to thwart the BGE attack, the large linear encodings can be retrieved with a variant of the Linear Equivalence algorithm. In addition, the attack exploits the internal structure of the AES. As pointed out in [91], the use of other types of encodings does not provide a much better security since the Affine Equivalence algorithm retrieves large affine encodings and the BGE generalization of [89] can be used in the case of large non-affine encodings. In particular, a variant of the Affine Equivalence algorithm which exploits the fact that the AES substitution layer is composed of small S-boxes, results to a lower attack complexity compared to a generic algorithm to solve the Affine Equivalence.

In addition to these structural attacks, we have seen in Chapter 3 that the WhibOx competition brings other vulnerabilities out. Actually, the published white-box implementations are not secure even if the attacker's abilities are weakened. Without any knowledge of the design specification an attacker who can instrument the binary is able to inject fault and thus applies a Differential Fault Analysis or can observe data during the execution which can be exploited for Differential Computation Analysis. While applying masking techniques to table-based white-box implementations seems to be limited and inefficient because of several adaptations of the DCA,

a circuit-based white-box implementation arises to better integrate DCA countermeasures. The circuit-based white-box implementation of the AES has a provable security against a first-order DCA with a reasonable size and performance. However, such an approach of considering only the weakened attack model is still insufficient against a strong white-box attacker.

In Chapter 3 of the second part of the thesis, we proposed a white-box implementation of the AES considering both structural and automated (DCA, DFA) attacks. The white-box implementation relies on randomized key-dependent lookup tables and systems of quadratic multivariate polynomials. We chose the encodings such that the success probability of the DCA is small. Additionally, we integrated a DFA countermeasure relying on data redundancy to the lookup tables and systems of polynomials. The main issue of our implementation is the possibility for the attacker to observe encoded bytes namely after the MixColumns. The size of the encoding as well as the structure of the AES remain the obstacles to a secure white-box implementation.

Main conclusion of Chapter 2 of the 1st part and Chapter 3 of the 2nd part and future works. *At first sight, a secure white-box implementation of the AES promises real advantages: an implementation resisting side-channel attacks with the easy deployment and update of a software. Yet, it appears to be a hard task as a white-box implementation has a size overhead and a performance slowdown with a poor security guarantee, so far. Consequently, it is legitimate to ask ourselves if such an approach is sustainable. We think that this research direction is not completely vain. Thus, we suggest to investigate a polynomial-based white-box implementation relying on a S-box decomposition. The motivation for this is that it would provide the one-wayness property which enables to transform a symmetric encryption scheme into a public-key scheme. Additionally, it would solve an open problem which is the decomposition of the AES S-box into quadratic permutations. If such a decomposition exists then it could be interesting to analyze how this can be used together with quadratic encodings to get a polynomial-based white-box implementation. Yet, one needs to be careful with the structural attacks applied on ASASA schemes as these attacks might be adapted to the white-box implementation.*

Theoretical approach for White-Box Cryptography.

The first theoretical model for White-Box Cryptography was inspired by theoretical models for code obfuscation and relies on the White-Box Property. However, the formalization does not fit well the properties that are expected from WBC in practice. We described in Chapter 4 of the first part of the thesis the security notions for the white-box model. The notions get close well to the needs of real-world applications, namely unbreakability ensures that a secret key cannot be deduced from the implementation, incompressibility guarantees that the implementation cannot be efficiently compressed by the attacker, and thus forcing them to lift the entire white-box code; traceability ensures that a redistributed software leaks the identity of a corrupted user, and one wayness guarantees that a white-box attacker cannot turn a decryption feature into an encryption feature and conversely. New families of block ciphers have been designed and proved to satisfy some of the security notions by construction. The main approach for providing unbreakability is to reduce the advantage of a white-box adversary to the advantage of a black-box adversary. The security reduction holds if the key-dependent component of the block cipher is constructed from a well-studied block cipher, typically the AES. The key-dependent component is implemented by a lookup table such that the adversary has access to a limited number of plaintext-ciphertext pairs of the underlying cipher. The table construction is also the basis for incompressibility. For instance, a lookup table implementing a pseudo-random permutation implies that any table

value is looked with the same probability. Thus, an attacker cannot compress the table without loosing a part of the encryption functionality.

We have described the different families of white-box dedicated block ciphers published so far to get an idea of their efficiency and security levels.

Main conclusion of the Chapter 4 of the 1st part and future research. *This approach for White-Box Cryptography is promising since it allows for the emergence of new families of block ciphers which can be implemented both in black-box and white-box. Thanks to the use of the AES as an underlying block cipher and parallelization, some implementations are quite efficient while bringing a size overhead due to the lookup table. Another direction for a theoretical approach for WBC is to involve building blocks that are originally designed for asymmetric schemes.*

Application-oriented approach.

In the second part of this thesis, we focused on two use cases of White-Box Cryptography, namely the use of cryptographic algorithms by mobile applications and for Digital Rights Management. When a cryptographic algorithm is executed by an application on the untrusted operating system of a mobile device, the secret key is exposed to malicious applications. A library with the white-box implementation is developed to enable an application to perform a cryptographic operation without revealing the key. Yet, if the library is not protected, it can be copied and executed by another application on another device. We proposed in Chapter 1 of the 2nd part an approach for device-binding using an unclonable hardware function named a Physically Unclonable Function (PUF). The concept of PUF have been studied for years to provide device identification, authentication and key storage.

We first assumed the existence of a PUF on a mobile device (plenty of works about PUFs actually suggest that the assumption is realistic). We defined a security notion which relies on the security of the PUF. The main idea is that the PUF is combined with an encryption scheme such that the unclonability of the PUF implies the security of the encryption scheme regarding the security notion. We named it *lockability* to refer to the node-locking technique which is used for software protection.

The PUF-based encryption scheme can be instantiated with the Even-Mansour construction of a tweakable block cipher. The advantage of the construction is that the AES block cipher can be used for the permutation part. In a nutshell, the so-called PUF-based block cipher generates a secret key from a PUF response by applying a keyed hash function and this secret key can be used to encrypt a message. Thus, a key is dynamically generated for any PUF response. Another advantage of our construction is that the keyed hash function can be constructed with a white-box dedicated block cipher and implemented in the white-box model.

The second contribution of the chapter is the characterization of a PUF on a mobile device. We chose to exploit the random variations of the image sensor as the unclonable physical characteristics. Since a PUF is characterized by a challenge-response behavior, we chose a set of predefined parameters as a challenge, namely a fixed exposure time and a fixed ISO sensitivity. The measured response of the PUF corresponds to a compressed quantized Dark Signal Non Uniformity which is a Fixed Pattern Noise dominant in dark images. The defined PUF is a weak PUF, i.e. a PUF with a small set of challenges.

The main inconvenient with the image sensor-based PUF is that it cannot be turned into a strong PUF (a PUF with an exponential number of challenges) which is mandatory to provide lockability.

A second contribution of this thesis is a traitor tracing system which relies on white-box techniques and a Tardos code. The problem of traitor tracing is linked to White-Box Cryptography since it provides, in theory, a solution to make a white-box program traceable. Such a feature is relevant for Digital Rights Management but existing traitor tracing schemes are not suitable for the white-box model. In Chapter 2 of the 2nd part, we proposed a traitor tracing system which relies on a key generator. The key generator makes call to key-dependent lookup tables and is unbreakable. Each user of the system receives a personalized key-generator which embeds a Tardos codeword. To the best of our knowledge, this is the first traitor tracing scheme implemented in the white-box model.

Main conclusion of Chapters 1 and 2 of the 2nd part and further research. *According to the use case, White-Box Cryptography should not be used as a stand alone protection technique. Namely, modern devices offer security features such as a Trusted Execution Environment which can be used together with WBC to protect critical applications or assets.*

Bibliography

[1] Alessandro Amadori, Wil Michiels, and Peter Roelse. A DFA attack on white-box implementations of AES with external encodings. In *Selected Areas in Cryptography - SAC 2019 - 26th International Conference*, volume 11959 of *Lecture Notes in Computer Science*. Springer, 2019.

[2] Erdal Arikan. Channel polarization: A method for constructing capacity-achieving codes. In *2008 IEEE International Symposium on Information Theory, ISIT 2008*, pages 1173–1177. IEEE, 2008.

[3] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, Berk Sunar, and Pim Tuyls. Memory leakage-resilient encryption based on physically unclonable functions. In *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security*, volume 5912 of *Lecture Notes in Computer Science*, pages 685–702. Springer, 2009.

[4] Frederik Armknecht, Daisuke Moriyama, Ahmad-Reza Sadeghi, and Moti Yung. Towards a unified security model for physically unclonable functions. In *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016*, volume 9610 of *Lecture Notes in Computer Science*, pages 271–287. Springer, 2016.

[5] Chung Hun Baek, Jung Hee Cheon, and Hyunsook Hong. White-box aes implementation revisited. *Journal of Communications and Networks*, 18(3):273–287, 2016.

[6] Boaz Barak, Yevgeniy Dodis, Hugo Krawczyk, Olivier Pereira, Krzysztof Pietrzak, François-Xavier Standaert, and Yu Yu. Leftover hash lemma, revisited. In *Advances in Cryptology - CRYPTO 2011, 31st Annual Cryptology Conference*, volume 6841 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011.

[7] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2001.

[8] Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri. Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. *IEEE Trans. Computers*, 52(4):492–505, 2003.

[9] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.

[10] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Conference*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[11] Olivier Billet and Henri Gilbert. A traceable block cipher. In *Advances in Cryptology - ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 331–346. Springer, 2003.

[12] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a white box AES implementation. In *Selected Areas in Cryptography- SAC 2004*, volume 3357 of *Lecture Notes in Computer Science*, pages 227–240. Springer, 2004.

[13] Olivier Billet and Duong Hieu Phan. Efficient traitor tracing from collusion secure codes. In *International Conference on Information Theoretic Security*, pages 171–182. Springer, 2008.

[14] Alex Biryukov, Charles Bouillaguet, and Dmitry Khovratovich. Cryptographic schemes based on the ASASA structure: Black-box, white-box, and public-key (extended abstract). In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security*, volume 8873 of *Lecture Notes in Computer Science*, pages 63–84. Springer, 2014.

[15] Alex Biryukov, Christophe De Canniere, An Braeken, and Bart Preneel. A toolbox for cryptanalysis: linear and affine equivalence algorithms. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 33–50. Springer, 2003.

[16] Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques*, volume 2045 of *Lecture Notes in Computer Science*, pages 394–405. Springer, 2001.

[17] Johannes Blömer and Jean-Pierre Seifert. Fault based cryptanalysis of the advanced encryption standard (AES). In *Financial Cryptography, 7th International Conference, FC 2003*, volume 2742 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 2003.

[18] Céline Blondeau, Andrey Bogdanov, and Gregor Leander. Bounds in shallows and in miseries. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference*, volume 8042 of *Lecture Notes in Computer Science*, pages 204–221. Springer, 2013.

[19] 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 *Applied Cryptography and Network Security - 16th International Conference ACNS 2018*, volume 10892 of *Lecture Notes in Computer Science*, pages 103–120. Springer, 2018.

[20] Andrey Bogdanov and Takanori Isobe. White-box cryptography revisited: Space-hard ciphers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1058–1069. ACM, 2015.

[21] Andrey Bogdanov, Takanori Isobe, and Elmar Tischhauser. Towards practical whitebox cryptography: Optimizing efficiency and space hardness. In *Advances in Cryptology - ASIACRYPT 2016, 22nd INternational Conference on the Theory and Application of Cryptology and Information Security*, volume 10031 of *Lecture Notes in Computer Science*, pages 126–158, 2016.

[22] Dan Boneh and Matthew K. Franklin. An efficient public key traitor tracing scheme. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference*, volume 1666 of *Lecture Notes in Computer Science*, pages 338–353. Springer, 1999.

[23] Dan Boneh and Moni Naor. Traitor tracing with constant size ciphertext. In *Proceedings of the 15th ACM conference on Computer and Communications Security*, pages 501–510. ACM, 2008.

[24] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 4004 of *Lecture Notes in Computer Science*, pages 573–592. Springer, 2006.

[25] Dan Boneh and Brent Waters. A fully collusion resistant broadcast, trace, and revoke systems. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006*, pages 211–220. ACM, 2006.

[26] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. In *Cryptographic Hardware and Embedded Systems - CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 215–236. Springer, 2016.

[27] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. In *Cryptographic Hardware and Embedded Systems - CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 215–236. Springer, 2016.

[28] Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. Perturbing and protecting a traceable block cipher. In *IFIP International Conference on Communications and Multimedia Security*, pages 109–119. Springer, 2006.

[29] Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. White box cryptography: Another attempt. *IACR Cryptology ePrint Archive*, 2006(2006):468, 2006.

[30] David Canright. A very compact s-box for AES. In *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2005.

[31] Hervé Chabanne, Duong Hieu Phan, and David Pointcheval. Public traceability in traitor tracing schemes. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 3494 of *Lecture Notes in Computer Science*, pages 542–558. Springer, 2005.

[32] 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 *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017*, volume 10159 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2017.

[33] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *Annual International Cryptology Conference*, pages 257–270. Springer, 1994.

[34] Benny Chor, Amos Fiat, Moni Naor, and Benny Pinkas. Tracing traitors. *IEEE Trans. Inf. Theory*, 46(3):893–910, 2000.

[35] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In *Selected Areas in Cryptography - SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 250–270. Springer, 2002.

[36] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box DES implementation for DRM applications. In *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.

[37] Benoît Cogliati, Rodolphe Lampe, and Yannick Seurin. Tweaking even-mansour ciphers. In *Advances in Cryptology - CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 189–208. Springer Berlin Heidelberg, 2015.

[38] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference*, volume 6841 of *Lecture Notes in Computer Science*, pages 487–504. Springer, 2011.

[39] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer, 2002.

[40] Cryptolux. Github repository. `https://github.com/cryptolu/whitebox`.

[41] CYBERCRYPT. The whibox contest edition 2. `https://www.cyber-crypt.com/whibox-contest/`.

[42] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[43] Cécile Delerablée, Tancrède Lepoint, Pascal Paillier, and Matthieu Rivain. White-box security notions for symmetric encryption schemes. In *Selected Areas in Cryptography - SAC 2013*, volume 8282 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 2013.

[44] Patrick Derbez, Pierre-Alain Fouque, Baptiste Lambin, and Brice Minaud. On recovering affine encodings in white-box implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):121–149, 2018.

[45] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

[46] Jintai Ding. A new variant of the matsumoto-imai cryptosystem through perturbation. In *International Workshop on Public Key Cryptography*, pages 305–318. Springer, 2004.

[47] Itai Dinur, Orr Dunkelman, Thorsten Kranz, and Gregor Leander. Decomposing the ASASA block cipher construction. *IACR Cryptology ePrint Archive*, 2015:507, 2015.

[48] Hans Dobbertin. Uniformly representable permutation polynomials. In *Sequences and their Applications - Proceedings of SETA 2001*, Discrete Mathematics and Theoretical Computer Science, pages 1–22. Springer, 2001.

[49] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[50] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. Differential fault analysis on AES. In *Applied Cryptography and Network Security, First International Conference ACNS 2003*, volume 2846 of *Lecture Notes in Computer Science*, pages 293–306. Springer, 2003.

[51] Morris J Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. Technical report, National Institute of Standards and Technology, 2015. Federal Information Processing Standards Publication 202.

[52] Shimon Even and Yishay Mansour. A construction of a cipher from s single pseudorandom permutation. *J. Cryptology*, 10(3):151–162, 1997.

[53] Jean-Charles Faugere and Ludovic Perret. Polynomial equivalence problems: Algorithmic and theoretical aspects. In *Advances in Cryptology - EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 30–47. Springer, Berlin, Heidelberg, 2006.

[54] Nelly Fazio, Antonio Nicolosi, and Duong Hieu Phan. Traitor tracing with optimal transmission rate. In *International Conference on Information Security*, pages 71–88. Springer, 2007.

[55] Ronal Aylmer Fisher and Frank Yates. Statistical tables for biological, agricultural and medical research. *Statistical tables for biological, agricultural and medical research*, (6th ed), 1963.

[56] Pierre-Alain Fouque, Pierre Karpman, Paul Kirchner, and Brice Minaud. Efficient and provable white-box primitives. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security*, volume 10031 of *Lecture Notes in Computer Science*, pages 159–188, 2016.

[57] Abbas El Gamal and Helmy Eltoukhy. Cmos image sensors. *IEEE Circuits and Devices Magazine*, 21(3):6–20, 2005.

[58] Blaise Gassend, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. Silicon physical random functions. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 148–160, 2002.

[59] Henri Gilbert, Jérôme Plût, and Joana Treger. Key-recovery attack on the ASASA cryptosystem with expanding s-boxes. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference*, volume 9215 of *Lecture Notes in Computer Science*, pages 475–490. Springer, 2015.

[60] Christophe Giraud. DFA on AES. In *Advanced Encryption Standard - AES, 4th International Conference, AES 2004*, volume 3373 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2004.

[61] Louis Goubin, Pascal Paillier, Matthieu Rivain, and Junwei Wang. How to reveal the secrets of an obscure white-box implementation. *IACR Cryptology ePrint Archive*, 2018:98, 2018.

[62] Louis Goubin, Jacques Patarin, and Bo- Yin Yang. Multivariate cryptography. In *Encyclopedia of Cryptography and Security, 2nd Ed*, pages 824–828. Springer, 2011.

[63] Xiaofei Guo and Ramesh Karri. Invariance-based concurrent error detection for advanced encryption standard. In *The 49th Annual Design Automation Conference 2012, DAC '12*, pages 573–578. ACM, 2012.

[64] Yunxi Guo and Akhilesh Tyagi. Voice-based user-device physical unclonable functions for mobile device authentication. *Journal of Hardware and Systems Security*, 1(1):18–37, 2017.

[65] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory*, 26(4):401–406, 1980.

[66] Irdeto. Cloackware software protection. `https://www.irdeto.com/cloakware-by-irdeto`.

[67] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(1):189–206, 1984.

[68] Ramesh Karri, Grigori Kuznetsov, and Michael Gössel. Parity-based concurrent error detection of substitution-permutation network block ciphers. In *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop*, volume 2779 of *Lecture Notes in Computer Science*, pages 113–124, 2003.

[69] Mohamed Karroumi. Protecting white-box AES with dual ciphers. In *Information Security and Cryptology - ICISC 2010*, volume 6829 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 2010.

[70] Tim Kerins and Klaus Kursawe. A cautionary note on weak implementations of block ciphers. In *1st Benelux Workshop on Information and System Security (WISSec 2006)*, volume 12, 2006.

[71] Aggelos Kiayias and Moti Yung. On crafty pirates and foxy tracers. In *Security and Privacy in Digital Rights Managements, ACM CCS-8 Workshop DRM 2001*, volume 2320 of *Lecture Notes in Computer Science*, pages 22–39. Springer, 2001.

[72] Aggelos Kiayias and Moti Yung. Breaking and repairing asymmetric public-key traitor tracing. In *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002*, volume 2696 of *Lecture Notes in Computer Science*, pages 32–50. Springer, 2002.

[73] Aggelos Kiayias and Moti Yung. Traitor tracing with constant transmission rate. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 450–465. Springer, 2002.

[74] Younghyun Kim and Yongwoo Lee. Campuf: physically unclonable function based on CMOS image sensor fixed pattern noise. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018*, pages 66:1–66:6, 2018.

[75] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[76] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[77] Konrad J. Kulikowski, Mark G. Karpovsky, and Alexander Taubin. Fault attack resistant cryptographic hardware with uniform error detection. In *Fault Diagnosis and Tolerance in Cryptography, Third International Workshop, FDTC 2006*, volume 4236 of *Lecture Notes in Computer Science*, pages 185–195. Springer, 2006.

[78] Kaoru Kurosawa and Yvo Desmedt. Optimum traitor tracing and asymmetric schemes. In *International Conference on the Theory and Application*, pages 145–157. Springer, 1998.

[79] Rodolphe Lampe and Yannick Seurin. Tweakable blockciphers with asymptotically optimal security. In *Fast Software Encryption - 20th International Workshop, FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2013.

[80] Jae W Lee, Daihyun Lim, Blaise Gassend, G Edward Suh, Marten Van Dijk, and Srinivas Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. In *2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No. 04CH37525)*, pages 176–179. IEEE, 2004.

[81] Seungkwang Lee, Taesung Kim, and Yousung Kang. A masked white-box cryptographic implementation for protecting against differential computation analysis. *IEEE Trans. Information Forensics and Security*, 13(10):2602–2615, 2018.

[82] Tancrède Lepoint, Matthieu Rivain, Yoni De Mulder, Peter Roelse, and Bart Preneel. Two attacks on a white-box aes implementation. In *International Conference on Selected Areas in Cryptography*, pages 265–285. Springer, 2013.

[83] Moses D. Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable block ciphers. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.

[84] Jan Lukás, Jessica J. Fridrich, and Miroslav Goljan. Determining digital image origin using sensor imperfections. In *Electronic Imaging: Image and Video Communications and Processing*, volume 6, 2005.

[85] Rui Luo, Xuejia Lai, and Rong You. A new attempt of white-box AES implementation. In *Proceedings IEEE International Conference on Security, Pattern Analysis and Cybernetics, SPAC 2014*, pages 423–429. IEEE, 2014.

[86] Roel Maes. *Physically Unclonable Functions: Constructions, Properties and Applications (Fysisch onkloonbare functies: constructies, eigenschappen en toepassingen).* PhD thesis, Katholieke Universiteit Leuven, Belgium, https://lirias.kuleuven.be/handle/123456789/353455, 2012.

[87] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Applications of Cryptographic Techniques*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.

[88] Wil Michiels and Paul Gorissen. Mechanism for software tamper resistance: an application of white-box cryptography. In *Proceedings of the Seventh ACM Workshop on Digital Rights Management*, pages 82–89. ACM, 2007.

[89] Wil Michiels, Paul Gorissen, and Henk DL Hollmann. Cryptanalysis of a generic class of white-box implementations. In *Selected Areas in Cryptography, 15th International Workshop, SAC*, volume 5381 of *Lecture Notes in Computer Science*, pages 414–428. Springer, 2008.

[90] Brice Minaud, Patrick Derbez, Pierre - Alain Fouque, and Pierre Karpman. Key-recovery attacks on ASASA. In *Advances in Cryptology - ASIACRYPT 2015*, volume 9453 of *Lecture Notes in Computer Science*, pages 3–27. Springer, 2015.

[91] Yoni De Mulder. *White-Box Cryptography: Analysis of White-Box AES Implementation (White-Box Cryptografie: Analyse van White-Box AES implementaties)*. PhD thesis, Katholieke Universiteit Leuven, Belgium, https://lirias.kuleuven.be/handle/123456789/430072, 2014.

[92] Yoni De Mulder, Peter Roelse, and Bart Preneel. Cryptanalysis of the xiao - lai white-box AES implementation. In *Selected Areas in Cryptography - SAC 2012*, volume 7707 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2012.

[93] Yoni De Mulder, Peter Roelse, and Bart Preneel. Revisiting the BGE attack on a white-box AES implementation. *IACR Cryptology ePrint Archive*, 2013:450, 2013.

[94] Yoni De Mulder, Brecht Wyseur, and Bart Preneel. Cryptanalysis of a perturbated white-box AES implementation. In *Progress in Cryptology - INDOCRYPT 2010*, volume 6498 of *Lecture Notes in Computer Science*, pages 292–310. Springer, 2010.

[95] David Naccache, Adi Shamir, and Julien P. Stern. How to copyright a function? In *Public Key Cryptography, Second International Workshop on Practice and Theory in PUblic Key Cryptography, PKC '99*, volume 1560 of *Lecture Notes in Computer Science*, pages 188–196. Springer, 1999.

[96] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference*, volume 2139 of *Lecture Notes in Computer Science*, pages 41–62. Springer, 2001.

[97] Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Decomposition of permutations in a finite field. *Cryptography and Communications*, 11(3):379–384, 2019.

[98] Jacques Patarin. Generic attacks for the xor of k random permutations. In *Applied Cryptography and Network Security - 11th International Conference ACNS 2013*, volume 7954 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2013.

[99] Duong Hieu Phan, Reihaneh Safavi - Naini, and Dongvu Tonien. Generic construction of hybrid public key traitor tracing with full-public-traceability. In *Automata, Languages and Programming, 33rd International Colloquium ICALP 2006*, volume 4052 of *Lecture Notes in Computer Science*, pages 264–275. Springer, 2006.

[100] Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against spn structures, with application to the aes and khazad. In *International Workshop on Cryptographic Hardware and Embedded Systems*, Lecture Notes on Computer Science, pages 77–88. Springer, 2003.

[101] NIST FIPS Pub. 197: Advanced encryption standard (aes). *Federal information processing standards publication*, 197(441), 2001. Available from https://www.nist.gov/publications/advanced-encryption-standard-aes.

[102] Sandra Rasoamiaramanana, Gilles Macario-Rat, and Marine Minier. White-box traitor tracing from probabilistic tardos codes. In *Innovative Security Solutions for Information Technology and Communications, 12th International Conference, SecITC 2019*, volume 12001 of *Lecture Notes in Computer Science*, 2019.

[103] Chester Rebeiro, A. David Selvakumar, and A. S. L. Devi. Bitslice implementation of AES. In *Cryptology and Network Security, 5th International Conference, CANS*, volume 4301 of *Lecture Notes in Computer Science*, pages 203–212. Springer, 2006.

[104] Matthieu Rivain and Junwei Wang. Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):225–255, 2019.

[105] Denise M Rousseau, Sim B Sitkin, Ronald S Burt, and Colin Camerer. Not so different after all: A cross-discipline view of trust. *Academy of management review*, 23(3):393–404, 1998.

[106] Eloi Sanfelix, Cristofaro Mune, and Job de Haas. Unboxing the white-box. In *Black Hat EU 2015*. 2015.

[107] Pascal Sasdrich, Amir Moradi, and Tim Güneysu. White-box cryptography in the gray box - - A hardware implementation and its side channels -. In *Fast Software Encryption - FSE 2016*, volume 9783 of *Lecture Notes in Computer Science*, pages 185–203. Springer, 2016.

[108] Amitabh Saxena, Brecht Wyseur, and Bart Preneel. Towards security notions for white-box cryptography. In *Information Security, 12th International Conference, ISC 2009*, volume 5735 of *Lecture Notes in Computer Science*, pages 49–58. Springer, 2009.

[109] Adi Shamir and Nicko van Someren. Playing "hide and seek" with stored keys. In *Financial Cryptography, Third International Conference, FC '99*, volume 1648 of *Lecture Notes in Computer Science*, pages 118–124. Springer, 1999.

[110] Boris Skoric, Stefan Katzenbeisser, and Mehmet Utku Celik. Symmetric tardos fingerprinting codes for arbitrary alphabet sizes. *Des. Codes Cryptogr.*, 46(2):137–166, 2008.

[111] Gaábor Tardos. Optimal probabilistic fingerprint codes. *Journal of the ACM*, 55(2):10, 2008.

[112] Gábor Tardos. Optimal probabilistic fingerprint codes. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 116–125. ACM, 2003.

[113] Aleksei Nikolaevich Udovenko. *Design and Cryptanalysis of Symmetric-Key Algorithms in Black and White-box Models*. phdthesis, University of Luxembourg, Luxembourg, 2019.

[114] Diego Valsesia, Giulio Coluccia, Tiziano Bianchi, and Enrico Magli. User authentication via prnu-based physical unclonable functions. *IEEE Transactions on Information Forensics and Security*, 12(8):1941–1956, 2017.

[115] Oliver Willers, Christopher Huth, Jorge Guajardo, and Helmut Seidel. Mems gyroscopes as physical unclonable functions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 591–602, 2016.

[116] Yaying Xiao and Xuejia Lai. A secure implementation of white-box aes. In *Computer Science and its Applications, 2009. CSA'09. 2nd International Conference on*, pages 1–6. IEEE, 2009.