**THÈSE**
pour obtenir le titre de
**Docteur de Sorbonne Université**
Spécialité : Informatique

Lucas BARTHELEMY

# A First Approach To Asymmetric White Box Cryptography and a Study of Permutation Polynomials Modulo $2^n$ in Obfuscation

**Directeur de thèse: Jean-Claude BAJARD**
**Co-encadrante de thèse: Marion VIDEAU**

Soutenue le XX décembre 2020 devant le **Jury composé de :**

| | | |
|---|---|---|
| *Rapporteurs :* | | |
| CARLOS AGUILAR-MELCHOR | - | Université de Toulouse, ISAE-SUPAERO. |
| LOUIS GOUBIN | - | Université de Versaille-St-Quentin-en-Yvelines (UVSQ), LMV. |
| *Examinateurs :* | | |
| CAROLINE FONTAINE | - | CNRS/Université de Paris-Saclay, LSV. |
| KARINE HEYDEMANN | - | Sorbonne Université, Paris, ALSOC LIP6. |
| GUÉNAËL RENAULT | - | École Polytechnique, Laboratoire d'Informatique (LIX). |
| *Directeur :* | | |
| JEAN-CLAUDE BAJARD | - | Sorbonne Université, IMJ-PRG. |
| *Co-Encadrante :* | | |
| MARION VIDEAU | - | Quarkslab. |

# *Acknowledgements*

There is a duality in doing a PhD. It is something *you* do, working three years on a subject unique to *you*. And yet this work would not have been possible alone. A lot of people helped making this possible, whether by providing guidance, knowledge, or simply psychological support in what have been a long process. Here, I will acknowledge the wonderful people I had the chance to work, exchange or simply have a conversation with.

First and foremost, I would like to thank the people that supervised this thesis. I thank Marion Videau and Jean Claude Bajard for their support and guidance. Marion gave a lot of her time and energy, despite her other responsibilities, helping me achieve my goal and I can't thank you enough for this herculean job. Jean Claude always took the time to hear my ramblings on asymmetric white-box whenever I came by the university. He has demonstrated great patience and insight and I am grateful for his guidance. I thank Jean-Baptiste Bedrune and Philippe Teuwen for their role in supporting me at Quarkslab. As my team leaders in the company, they helped me feel integrated to Quarksalb's activities, provided support and feedback and help me feel part of a team. I also thank Fred Raynal, who supported this thesis that happened in his company.

I would like to thank the people without whom this PhD would not have happened in the first place. First, I would like to thank Raphaël Roblin. This adventure all began when we started working together on binary permutation polynomials. Although we do not work together anymore, you will forever be my *binome* and I wish you the best for your new adventures. On the same note, I would like to thank Guénaël Renault for supporting and guiding us. You gave me and Raphaël a chance and I think I can speak for the both of us when I say you will always have our thanks, our respect and our support.

I would also like to thank Ninon Eyrolles and Adrien Guinet who became our link to Quarkslab in these early years. You gave two students the opportunity to work on a wonderful subject, with real-life application. You helped jump-start this wonderful adventure and for that you have my thanks. I would like to give additionnal thanks to Ninon Eyrolles, who also helped me during my first internship at Quarkslab (despite finishing her own PhD) and was there when I presented my first paper at a conference. You became a close friend and I wish you and GG all the happiness in the world. I would also like to thank Marion again who at the time saw something in me and gave me the opportunity to start a PhD at Quarkslab.

During those three years, I had the opportunity to engage with a lot of wonderful people. Whether they worked directly with me or simply exchanged with me over a cup of coffee, they all participated in this work. In particular, I would like to thank Jérome Courtois and Vincent Zucca with whom I shared an office (and Jean Claude) during my time at the university. In addition, I would like to thank the following people: Ninon Eyrolles, Charles Hubain, Kevin Szkudlapski, Emma Benoit, William Gouzer, Sebastien Kaczmarek, Gabrielle Viala, Thierry Doré, Alexis Challande, Matthieu Daumas, Adrien Guinet, Muriel Hoquy (a.k.a. iomumu) and Florelle Seigneur. You all contributed to this work, and you all have my thanks.

A PhD is not all about work. A lot of people, friends and family, were there along the way to give me emotional and psychological support. First I would like to thank my significant other Eugenia Delacou. You shared my life throughout this period of my life and made me the happiest duck in the world. You endured the stress, the rambling, and everything that comes with sharing a life with a PhD student. Your support means the world to me and I can only hope to give some of it back while you pursue your own PhD. You have my thanks and my love. I would also like to thank my mother Nathalie Beaufranc who gave so much to her son. There are no words to describe the amount of love and support you gave me. I would also like to thank my grandfather Yves Beaufranc. You played a key role in my life and my only regret is that you passed away a couple of month before its conclusion. I wish you are well and happy whenever you are.

Last but not least, I would like to thank my friends who helped me along the way. Many thanks to Cinesthez, Sky, Benj, Denuy, Streum, Emoren, Noutoff, Wigz, Gwaby, Kylma, Ack, Haxe, Marie, Arlynis and many more... Sometimes a drink with a friend, or a couple of games, makes a difference.

To my brothers I should have spent more time with growing up, you can do everything. I love you all!

# Contents

# Introduction (French)

La sécurité des technologies de l'information est un vaste domaine de recherche qui devient de plus en plus complexe à mesure que la technologie elle-même se complexifie. Traditionnellement, la recherche sur ce sujet se concentrait sur un modèle impliquant un attaquant à distance qui tente d'obtenir un accès aux informations présentes sur un dispositif ou un réseau, ou d'en prendre le contrôle. Cependant, les technologies modernes ont ajouté un nouveau modèle de menace à l'équation. Comment évaluer la sécurité d'un dispositif lorsque l'attaquant n'est pas à distance ? Par exemple, on peut considérer le cas du Digital Right Management (D.R.M.) où l'utilisateur final d'un produit (un lecteur, une *box* d'abonnement, etc...) est considéré comme hostile vis-à-vis des droits d'auteur. On peut également considérer l'exemple du calcul dans le cloud (*cloud computing*), où un fournisseur de *cloud computing* peut être considéré comme hostile à l'égard de questions de sécurité, telles que l'anonymat de ses utilisateurs ou la confidentialité des données stockées et potentiellement traitées sur ses serveurs. Ce nouveau contexte motive le travail réalisé au cours de cette thèse. Ce travail a porté en particulier sur deux techniques de protections, la cryptographie en boîte blanche et l'obscurcissement. Ces deux techniques peuvent être réunies dans un domaine plus large qu'est la *protection du logiciel*.

## La protection du logiciel

Le terme *protection du logiciel* désigne les techniques de protection utilisées pour prévenir les attaques effectuées par l'utilisateur final (ou un attaquant ayant les mêmes prérogatives) d'une application sur un système informatique. Ces types d'attaques sont appelés attaques *Man-At-The-End* (MATE). Bien qu'il existe un besoin croissant de protection contre ces types d'attaques (logiciels embarqués dans des appareils mobile, D.R.M. de produits livré au client, calculs dans le cloud), la protection contre les attaques MATE reste un problème difficile. En effet, il faut considérer un attaquant qui a le même accès et le même pouvoir qu'un utilisateur final légitime du logiciel, logiciel qui traite les informations sensibles que nous souhaitons protéger. Un tel attaquant peut altérer le logiciel lui-même, ce qui soulève la question de l'intégrité du code. Il peut également procéder à une rétro-ingénierie (ou ingénierie inverse, du terme technique anglais *reverse engineering*) du logiciel, ce qui lui permet de mieux comprendre son fonctionnement. Cette compréhension peut lui permettre d'accéder à des données sensibles ou peut conduire à des attaques de plus haut niveau. Dans un tel modèle d'attaque, un attaquant a accès à toutes les données traitées par le logiciel et est en mesure de modifier l'entrée de toute fonction appelée par le logiciel.

La protection du logiciel vise à résoudre le problème de la sécurité dans un modèle aussi hostile à l'aide de divers outils. Parmi ces outils, on retrouve la cryptographie en boîte blanche, l'obscurcissement, les techniques d'anti-extraction (*anti-tampering*), etc... Au cours de cette thèse, nous avons considéré deux problématiques liées à la protection du logiciel, la cryptographie en boîte blanche d'une part et l'obscurcissement d'autre part. En particulier, nous avons conçu, implémenté et proposé une première construction pour un chiffrement asymétrique en boîte blanche. Contrairement à la cryptographie symétrique en boîte blanche, le sujet de la cryptographie asymétrique en boîte blanche est très peu présent dans la littérature, malgré une utilisation dans le domaine de l'entreprise (comme l'attestent par exemple les brevets suivants : [MG09], [AAMSD18] et [HM16]). Aussi, notre proposition s'inscrit comme la première proposition disponible dans la littérature. Nous avons aussi étudié les bijections polynomiales modulo $2^n$ telles que décrites dans l'article [Riv01]. Ces bijections sont utilisées dans le cadre d'expressions mixtes arithmético-booléennes (MBA) qui sont elle-même utilisées dans des techniques d'obscurcissement, comme les techniques présentées dans [ZMGJ07], utilisant un sous-ensemble de polynômes dont les inverses sont descriptibles par une formule close. Nous proposons deux nouvelles méthodes pour inverser l'ensemble des bijections polynomiales présentées dans [Riv01]. Nous pouvons ainsi utiliser les techniques d'obscurcissement présentées dans [ZMGJ07] tout en résistant à l'attaque présentée dans [BJLS17].

Dans ce résumé, nous allons tout d'abord présenter la cryptographie et ses différents modèles d'attaques, en particulier le modèle en boîte blanche. Puis nous présenterons l'obscurcissement comme outil de protection du logiciel. Enfin, nous résumerons les travaux réalisés au cours de cette thèse et présenterons son organisation et ses contributions.

## Cryptographie

La cryptographie est la science de la protection de l'information. De manière simplifiée, elle utilise des objets mathématiques complexes pour créer un couple de fonctions. Une fonction pour transformer l'information en données protégées, et une fonction pour accéder au contenu protégé. La fonctionnalité cryptographique la plus connue et la plus ancienne est le chiffrement. Dans ce cas, le couple de fonctions est le chiffrement et le déchiffrement. Ces deux fonctions sont utilisées pour garantir qu'une donnée n'est accessible qu'à un ensemble d'entités légitimes. L'évolution historique de la cryptographie en une science de la protection de l'information a débouché sur la mise au point d'algorithmes soumis à évaluation — publique pour les standards, paramétrés par une clé dont une partie secrète est la garantie de sécurité du processus d'échange de données. Différents modèles d'attaques existent selon les données préalables connues par l'attaquant, celles qu'il peut recueillir au cours de l'attaque et sa puissance de calcul. Un cryptosystème est considéré sûr lorsqu'une entité illégitime, ne peut extraire aucune information d'un échange de données protégées cryptographiquement entre plusieurs entités légitimes.

Notons que la cryptographie s'intéresse aussi au problème de l'authentification de l'origine des données, mais également à des fonctionnalités évoluées et complexes telles que la cryptographie homomorphe, les schémas de signatures à vérifieurs désignés, le vote électronique, le partage de secrets à seuil, la cryptomonnaie et de nombreuses autres évolutions.

Nous pouvons différencier deux familles de fonctions cryptographiques :

- la cryptographie à clé secrète dite aussi symétrique, qui comprend les cryptosystèmes AES et DES par exemple,
- la cryptographie à clé publique dite aussi asymétrique, qui comprend les cryptosystèmes RSA et ECC par exemple.

La cryptographie symétrique est la plus ancienne forme de cryptographie. Dans cette configuration, deux personnes qui communiquent utilisent chacune une clé cryptographique identique à l'autre. Les deux utilisateurs peuvent utiliser cette clé pour chiffrer ou déchiffrer des messages. Le processus de chiffrement est une transformation appliquée à un message qui dépend de la valeur du message et de la clé cryptographique choisie. Le déchiffrement consiste à effectuer la transformation inverse sur un message chiffré. La connaissance de la clé cryptographique permet d'effectuer les deux transformations. La cryptographie symétrique est généralement plus rapide et plus légère en mémoire que sa contrepartie asymétrique. Elle est donc utilisée pour le traitement rapide d'une grande quantité de données ou dans des contextes contraints comme du matériel embarqué. La gestion des clés ou le partage de clé reste toutefois un problème compliqué à résoudre avec des outils cryptographiques symétriques seuls.

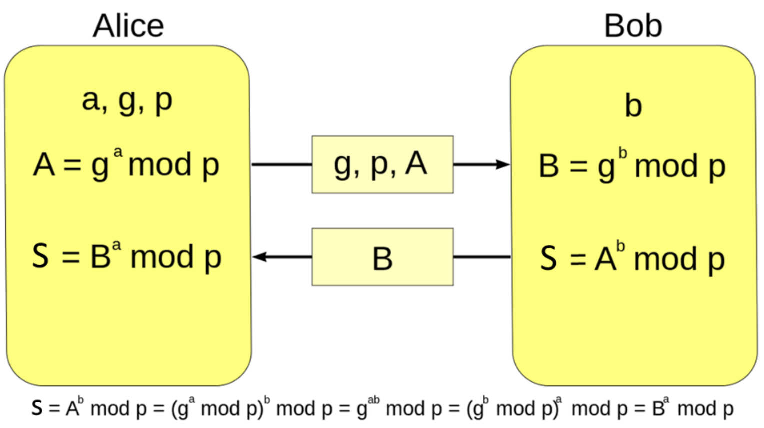
En 1976, Whitfield Diffie et Martin Hellman conceptualisent l'idée d'un cryptosystème à (bi-)clés publiques/privées. C'est-à-dire un cryptosystème pour lequel les clés de chiffrement et de déchiffrement sont différentes. De leur travail découle le protocole de négociation de clé de Diffie-Hellman (renommé Diffie-Hellman-Merkle en 2002 afin de reconnaître la contribution apportée par Ralph Merkle à la création du concept de cryptographie à clé publique). Cette technique repose sur l'utilisation d'une trappe cryptographique, une fonction qu'il est facile de calculer dans un sens (dans leur cas une exponentiation modulaire), mais dont l'opération inverse est difficile à réaliser (dans leur cas, résoudre le problème du logarithme discret).

Préalablement à la négociation de clé a lieu une étape de génération pour chaque participant d'un couple de clés : une clé publique qui sera diffusée aux autres participants et une clé privée qui devra rester un secret. Dans le cas d'une négociation entre deux participants, chaque participant combine sa clé privée avec la clé publique de l'autre afin de produire la clé négociée entre les deux parties. Cette clé négociée possède les propriétés suivantes :

- elle est unique aux deux participants. Statistiquement, aucune autre paire de participants ne peut obtenir la même valeur,
- elle ne nécessite pas aux participants d'échanger leurs clés privées,
- retrouver cette clé négociée sans avoir connaissance des clés privées des deux participants est un problème difficile.

Ce protocole permet donc à deux participants de se mettre d'accord sur une valeur commune sans jamais échanger cette valeur. Celle-ci peut alors servir à déterminer une clé de cryptographie symétrique entre les deux participants sans avoir à l'échanger dans le canal de communication.

Pour fixer les idées nous allons prendre en exemple une négociation de clé Diffie-Hellman-Merkle reposant sur le problème du logarithme discret dans le groupe multiplicatif des entiers modulo $p$, $p$ premier. Soit $g$ un générateur de ce groupe. Soit $a \in \mathbb{Z}/p\mathbb{Z}$ la valeur privée d'Alice, participant à la négociation de clé. Respectivement, soit $b$ la valeur privée de Bob, un autre participant. Supposons qu'Alice calcule la valeur $g^a \mod p$ et la diffuse sur le canal. Le problème du logarithme discret suppose qu'il est difficile de retrouver $a$ avec pour seule connaissance les valeurs $(g, p, g^a)$. Bob peut effectuer un calcul similaire pour envoyer la valeur $g^b \mod p$. Ils peuvent tout deux calculer la valeur commune $(g^a)^b \bmod p = (g^b)^a \bmod p$ sans jamais diffuser leur valeur privée. Le protocole est schématisé de la manière suivante:


Alice
Bob
a, g, p
A = g^a mod p
S = B^a mod p
g, p, A
B
b
B = g^b mod p
S = A^b mod p
S = A^b mod p = (g^a mod p)^b mod p = g^ab mod p = (g^b mod p)^a mod p = B^a mod p

En 1978, cette idée est reprise par Ron Rivest, Adi Shamir et Leonard Adleman dans leur présentation du cryptosystème asymétrique RSA qui implémente l'idée d'un chiffrement asymétrique. La sécurité de ce cryptosystème repose sur la difficulté du problème de la factorisation d'un entier produit de deux grands nombres premiers (appelé problème RSA). Soit $N = pq$ un grand entier, produit de deux entiers premiers. Soit $\Phi(N) = (p-1)(q-1)$ l'ordre du groupe $\mathbb{Z}/N\mathbb{Z}$. Chaque utilisateur dispose d'une clé de cryptographie publique, $e$, qui sera diffusée à tous les participants, et d'une clé de cryptographie privée $d$. Ces deux valeurs sont générées telles que :

$$ed = 1 \mod \Phi(N)$$

Du théorème de Fermat-Euler découle le résultat suivant:

$$x^{ed} \mod N = x \mod N \qquad \forall x \in (\mathbb{Z}/N\mathbb{Z})^\star$$

N'importe quel participant peut utiliser la clé publique d'un autre participant afin de chiffrer un message $m$ à son attention en calculant la valeur suivante :

$$c = m^e \mod N$$

Seul son destinataire dispose de la capacité de déchiffrer ce message à l'aide de sa clé privée. Pour ce faire, il calcule la valeur suivante :

$$c^d \mod N = (m^e)^d \mod N = m^{ed} \mod N = m \mod N$$

et obtient ainsi la valeur du message $m$. Notons que les exemples présentés n'ont qu'une valeur illustrative mathématique des principes décrits. Ils ne constituent pas la spécification des algorithmes implémentés en pratique qui incluent de nombreux ajouts pour traiter des cas spéciaux, des optimisations ou de protections contre diverses attaques prises en compte depuis leur invention.

La cryptographie asymétrique est plus difficile à manipuler que la cryptographie symétrique. Elle demande des calculs plus complexes (et donc qui prennent plus de temps) et peine à chiffrer/déchiffrer une large quantitée de donnée. Elle est en général utilisée pour :

- négocier des clés symétriques dans le cadre d'un protocole de négociation de clé,
- résoudre des problématiques d'authentification de participants,
- échanger des messages de petite tailles.

Récemment, des travaux effectués sur l'algorithmie quantique (algorithme pouvant être effectué par un ordinateur quantique) ont montré que de nombreux problèmes supposés difficiles pourraient être résolus à l'aide d'un ordinateur quantique [Sho97]. Bien que l'état actuel de la recherche sur l'ordinateur quantique ne permette pas encore d'effectuer ce type d'attaques, l'organisme NIST (*National Institute of Standard and Technology*) a lancé une compétition afin de trouver de nouveaux standards en matière de cryptographie asymétrique qui pourrait résister aux attaques quantiques. Parmi ces propositions, on retrouve des algorithmes cryptographiques basés sur les réseaux euclidiens.

Lorsque l'on cherche à évaluer la résistance d'un algorithme cryptographique, symétrique ou asymétrique, il nous faut caractériser l'attaquant, c'est-à-dire décrire précisément quelles sont ses capacités, ses connaissances et son but, et donc en formaliser le modèle. Nous allons maintenant définir les modèles d'attaques en boîte noire, boîte grise et boîte blanche.

## Les différents modèles d'attaques

Examinons à nouveau l'exemple simple de deux utilisateurs qui échangent des messages. Cet exemple est une configuration classique dans laquelle un attaquant est modélisé comme une entité externe qui n'a accès aux communications que lorsqu'elles sont envoyées d'un utilisateur à l'autre. Un tel modèle pour un attaquant est appelé modèle dans le contexte boîte noire. Dans un contexte boîte noire, un attaquant n'a accès qu'aux entrées/sorties d'un schéma cryptographique et voit les fonctions cryptographiques comme des oracles, transformant l'une en l'autre (il n'a pas connaissance des calculs internes effectués par les fonctions cryptographiques, d'où le terme *boîte noire*).

L'attaquant tente d'obtenir une information secrète, comme un message déchiffré ou la valeur d'une clé cryptographique, à partir d'échanges sur le canal de communication. Il peut, par exemple, rejouer des communications ou exposer un biais statistique dans le contenu des échanges chiffrés. En revanche, il ne peut faire aucune observation sur le fonctionnement interne des entités participant aux communications et ne peux pas en modifier le comportement.

La modélisation d'un attaquant dans un contexte en boîte noire est la manière la plus classique de modéliser les menaces pesant sur un système d'information dont la sécurité repose sur la cryptographie. Cependant, les applications de la vie réelle sont parfois opposées aux attaquants qui ont accès à plus de ressources que ce qui est modélisé par le contexte de la boîte noire. Par exemple, considérons notre exemple précédent dans le cas où un attaquant peut être physiquement présent à proximité d'un dispositif exécutant une des fonctions cryptographiques. Étant donné que nous vivons au 21$^{e}$ siècle, cet appareil est probablement un appareil électronique quelconque (ordinateur, téléphone, carte à puce, etc.). À courte distance, les composants électroniques ont tendance à laisser échapper toutes sortes d'informations supplémentaires, comme la consommation d'énergie ou l'émission d'ondes électromagnétiques. Ces sources d'informations supplémentaires peuvent aider un attaquant à réaliser une attaque. Par exemple, les attaques mesurant le temps d'exécution (*timing attacks*) et les attaques par analyse de puissance différentielle (DPA) [KJJ99] sont deux attaques spécifiques qui sont rendues possibles par la capacité de l'attaquant à écouter la manifestation physique d'un dispositif effectuant une opération cryptographique. En outre, la proximité physique peut permettre à l'attaquant de modifier physiquement l'état d'un dispositif pendant qu'il effectue une opération cryptographique. Par exemple, les attaques par fautes [DLV03], qui utilise l'injection d'une *faute* (modification d'une donnée en mémoire) pour obtenir de l'information sur le fonctionnement interne d'un système, sont devenues une préoccupation de plus en plus fréquente en ce qui concerne la sécurité de la cryptographie moderne. Ces types d'attaques sont communément appelés *side-channel attacks*, ou attaques par canaux auxiliaires. Un attaquant dans un tel modèle dispose de plus d'information sur les opérations de chiffrement/déchiffrement qu'un attaquant en boîte noire, sans pour autant avoir un accès total au dispositif effectuant l'opération. Pour cela, un modèle prenant en compte ces types d'attaques est généralement appelé modèle d'attaque dans un contexte en boîte grise.

Un attaquant dans un modèle en boîte grise peut interagir avec un appareil effectuant une opération cryptographique sans pour autant avoir le contrôle de l'appareil. Il peut prendre toute sortes de mesures physiques (consommation de courant, émission EM, etc. . . ) pendant le fonctionnement de l'appareil et en modifier le comportement via l'injection de faute (en modifiant le contenu d'un registre avec un laser par exemple). Il peut utiliser ces informations pour émettre des hypothèses sur les calculs internes effectués lors de l'opération cryptographique. Un attaquant en boîte grise va chercher à utiliser ces hypothèses pour l'aider à obtenir une information secrète manipulé par l'appareil, comme par exemple une clé cryptographique. Par exemple, réussir à déterminer le poids de Hamming d'une clé cryptographique peut grandement réduire l'effort nécessaire pour déterminer la valeur de cette clé.

Considérons maintenant un modèle d'attaque qui traite des attaques de type *Man-At-The-End*. Un attaquant dans un tel modèle aurait accès à beaucoup plus de ressources que le modèle classique en boîte noire, ou même le modèle en boîte grise. En effet, un tel attaquant aurait les mêmes connaissances et le même accès qu'un utilisateur légitime de l'appareil effectuant les opérations cryptographiques. Il a donc accès au contenu de la mémoire et à l'implémentation des fonctions cryptographiques. Un tel modèle hostile est appelé modèle d'attaque dans un contexte en boîte blanche.

Dans un modèle en boîte blanche, on considère un attaquant qui a un contrôle total de l'appareil effectuant les fonctions cryptographiques, et qui cherche à en extraire des clés de sécurité. L'attaquant peut lire tout contenu stocké en mémoire, manipuler le code impliqué dans l'exécution des fonctions cryptographiques, ajouter des points d'arrêt ou modifier toute variable de l'environnement pendant l'exécution, etc. . . Il peut utiliser ces capacités pour monter des attaques spécifiques au contexte en boîte blanche, comme par exemple les attaques algébriques de type BGE [BGEC05], ou adapter des attaques issues du modèle boîte grise. En effet, des techniques comme l'injection de faute sont beaucoup plus facile à réaliser dans un contexte en boîte blanche (plus besoin de modifier un bit à l'aide d'un laser quand on peut simplement changer le contenu d'un registre). C'est pour cette raison que la cryptographie en boîte blanche cherche à dissimuler la clé dans le code même de l'implémentation afin que les observations qu'un attaquant est capable de faire ne lui permette pas d'en déduire la valeur.

Notons que l'on ne cherche pas à empêcher l'attaquant d'utiliser ces fonctions cryptographiques. En effet, contrairement aux contextes en boîte noire et en boîte grise, l'attaquant en boîte blanche dispose d'un accès légitime aux fonctions cryptographiques.

Dans le second chapitre de notre manuscrit, nous discutons de l'état actuel des propositions de cryptosysèmes symétriques en boîte blanche en mettant l'accent sur les propositions en boîte blanche du cryptosystème AES. Dans le troisième chapitre, nous discutons de notre propre proposition pour la conception d'un cryptosystème asymétrique de chiffrement en boîte blanche basé sur les réseaux euclidiens.

## L'obscurcissement

L'obscurcissement est le moyen par lequel la protection des logiciels lutte contre le processus de rétro-ingénierie ( ou ingénierie inverse). Le processus d'ingénierie inverse en informatique est l'étude d'un binaire exécutable et de l'environnement sur lequel il est exécuté afin de comprendre ce qu'il fait et les détails sur la manière dont il effectue certaines opérations. Par exemple, l'ingénierie inverse peut chercher à obtenir des informations sur le fonctionnement d'un logiciel afin d'en produire une copie plus performante (en retirant par exemple certaines protections). L'ingénierie inverse peut aussi servir a extraire des données clés d'une application, comme des clés cryptographiques ou des valeurs d'initialisation de certains protocoles de sécurité.

Comme les cibles pratiques sont généralement des fichiers binaires qui ne sont pas censés être lisibles par l'homme, la rétro-ingénierie doit utiliser différents outils pour analyser et/ou simplifier la cible afin d'obtenir une meilleure compréhension de son fonctionnement. La rétro-ingénierie est généralement classée en deux types d'approches :

- l'analyse statique des éléments, dans laquelle la rétro-ingénierie considère le fichier exécutable comme une cible statique, cherchant à en extraire un maximum d'information. Ce type d'analyse est le plus courant car il ne nécessite aucune configuration ou condition préalable supplémentaire quant à la capacité de l'attaquant à manipuler l'environnement de la cible. Ce type d'analyse peut également être le seul disponible si l'attaquant ne peut (ou ne veut) pas modifier et/ou contrôler l'environnement du dispositif sur lequel le fichier exécutable est exécuté. Dans cette configuration, la rétro-ingénierie ne peut accéder qu'au code lui-même. Cela signifie, par exemple, qu'il peut chercher des constantes, mais qu'il doit prédire les valeurs des registres pendant l'exécution.

- L'analyse dynamique des éléments, dans laquelle la rétro-ingénierie inverse peut observer et/ou modifier l'exécutable pendant qu'il est exécuté dans son environnement (ou dans un environnement répliqué). Ce type d'analyse est généralement beaucoup plus puissant car il permet au rétro-concepteur d'étudier le flux de contrôle d'une application, d'accéder au contenu des registres pendant l'exécution ou même d'insérer des points d'arrêt pour étudier les différents états d'exécution. En revanche, ce type d'analyse est plus difficile à mettre en place. Il nécessite un certain contrôle de l'environnement d'exécution et de nombreux logiciels utilisent des techniques, à la fois logicielles et matérielles, visant à se prémunir de ce type de contrôle.

L'obscurcissement vise à entraver le processus de rétro-ingénierie en modifiant la cible (un binaire exécutable) afin de rendre plus difficile pour l'attaquant de comprendre ce qui est traité et comment. Une définition très théorique de l'obscurcissement est l'*obscurcissement indiscernable* dont la définition est discuté dans [BGI$^+$12] et [GR07].

Cette définition considère la notion d'indistingabilité associées à des circuits équivalent $C0$ et $C1$ de tailles similaires. L'obfuscation de $C0$ et de $C1$ doit être indistingable, signifiant que les circuits obscurcits ne doivent pas présenter de différence significative (en matière d'opérations ou d'accès mémoire par example) lors de leur execution pour une valeur d'entrée fixée. Dans la pratique, cependant, la notion d'*obscurcissement indiscernable* est extrêmement difficile à mettre en œuvre.

Voici des exemples d'outils d'obscurcissement et de ce qu'ils tentent de protéger :

- Les prédicats opaques qui remplacent une valeur constante par une expression complexe difficile à simplifier. Cette technique peut être utilisée pour ajouter de faux branchements de code. Ceci rendra l'analyse statique plus difficile comme celle-ci devra considérer du code qui n'a en réalité aucune signification vis-à-vis des opérations réellement effectuées.

- L'aplatissement du graphe de flot de contrôle qui modifie le flux de contrôle (l'ordre dans lequel les opérations sont exécutées) d'une cible exécutable en une série de blocs d'opérations dont le flux est géré par un répartisseur (programme qui va réorganiser l'ordre d'exécution des différentes opérations), lui-même obscurci. Cette technique vise à entraver l'analyse statique car elle rend plus difficile pour le rétroconcepteur de comprendre dans quel ordre les opérations sont exécutées. Dans une moindre mesure, cette technique peut également entraver l'analyse dynamique si le répartisseur est obscurci en conséquence.

- Les constantes opaques qui remplacent les valeurs constantes utilisées par l'application par des expressions complexes, des valeurs masquées ou même des valeurs chiffrées. Cette technique est utilisée pour masquer les valeurs exactes des constantes clés utilisées par l'application. Par exemple, les valeurs d'ordonnancement utilisées par le répartisseur introduites dans la technique d'aplatissement des graphes de flot de contrôle, les vecteurs d'initiation utilisés pour certains protocoles de sécurité, les valeurs constantes utilisées dans les fonctions de hachage, etc. . .

- Les techniques d'anti-extraction qui visent à supprimer la capacité de la rétro-ingénierie à extraire le code de la cible afin de le placer dans un environnement contrôlé. Bien qu'elle ne soit pas toujours considérée comme une technique d'obscurcissement par la littérature, cette technique vise à perturber l'analyse dynamique en empêchant le processus d'ingénierie inverse d'analyser la cible dans un environnement plus contrôlé.

Une technique d'obscurcissement qui peut être utilisée pour fabriquer des expressions complexes, à la fois pour les prédicats et les constantes opaques, est l'utilisation d'expressions mixtes arithmético-booléennes (MBA). Cette technique est basée sur l'hypothèse que, s'il existe de nombreuses façons efficaces de simplifier soit les expressions booléennes, soit les expressions arithmétiques, les expressions complexes qui mélangent ces deux types d'opérations sont plus difficiles à simplifier. La résistance des expressions MBA vis-à-vis de différentes méthodes et outils de simplification est étudiée dans [Eyr17]. Un outil clé nécessaire à la fabrication des expressions MBA sont les fonctions bijectives et leur inverse car les deux sont nécessaires pour appliquer l'obscurcissement (l'une est pré-calculée pendant le processus d'obscurcissement tandis que l'autre est utilisée lors de l'exécution du programme). Dans le quatrième chapitre de cette thèse nous présentons une façon de produire et d'inverser des bijections polynomiales modulo $2^n$, apportant des bijections polynomiales nouvelles par rapport à celles qui ont été précédemment présentées dans [ZMGJ07]. Ces nouveaux types de polynômes permettent une technique d'obscurcissement basée sur les MBA qui résiste à une attaque spécifique présentée dans [BJLS17].

## Organisation et contribution de la thèse

Cette thèse est organisée en trois axes précédés par un premier chapitre qui résume les différentes notions utilisées tout au long de cette thèse comme : le cryptosystème AES, les bases de la cryptographie en réseau, la décomposition RNS, la transformation NTT.

Dans le second chapitre, nous discutons de l'état de la cryptographie symétrique en boîte blanche en mettant l'accent sur les propositions d'implémentation du cryptosystème AES en boîte blanche. Nous concentrons notre attention sur les questions suivantes :

- Quelles sont les propriétés de l'AES qui facilitent la création de cryptosystèmes en boîtes blanches ?
- Quelles sont les propriétés de l'AES qui rendent ces propositions encore vulnérables aujourd'hui ?

Le but de cette discussion est de déterminer comment produire une proposition de cryptosystème en boîte blanche basée sur un algorithme différent sans répéter les défauts inhérents à l'AES dans un modèle d'attaque de boîte blanche. Nos conclusions à ce sujet sont que l'implémentation d'un cryptosystème via l'utilisation de tables de correspondance (*lookup tables*) reste la clé de la production d'un modèle en boîte blanche. Le cryptosystème AES est particulièrement adapté à une telle implémentation car il effectue des séries de calculs internes sur des données de petite taille. Toutefois, les les différentes propositions d'algorithmes AES en boîte blanche présentent les problèmes de sécurité suivants :

- Le cryptosystème AES est par conception extrêmement vulnérable à de nombreuses attaques par canaux auxiliaires, attaques qui peuvent être adaptées au modèle de la boîte blanche pour devenir encore plus puissantes [BHMT16] [AMR20]. Ces types d'attaques constituent toujours la majorité des attaques en boîte blanche sur les propositions de pointe.
- Les techniques d'encodage sont affaiblies car elles sont contraintes par la structure de l'AES. Étant donné la présence de transformations non-linéaires dans l'AES, les encodages appliqués aux tables de correspondances doivent être supprimés immédiatement dans la table de correspondance suivante (ou presque immédiatement dans le cas de bijections de mélange [CEJO03]). Cela ouvre la voie à un autre type d'attaques qui visent à supprimer ces encodages pour accéder à la clé secrète : les attaques algébriques [BGEC05].

Notre objectif dans le troisième chapitre est de faire une proposition qui utilise toujours la technique d'implémentation basée sur des tables de correspondance utilisée dans les propositions de boîte blanche AES, mais appliquée à un cryptosystème différent qui est, à notre connaissance, moins vulnérable aux attaques par canaux auxiliaires, et qui utilise des propriétés homomorphes latentes pour résoudre le problème des attaques algébriques. Dans ce chapitre, nous présentons notre proposition de boîte blanche asymétrique basée sur les réseaux euclidiens.

Notre proposition est basée sur la construction proposée dans les articles [GAGL15] et [AMBG+16]. L'algorithme de cryptographie présenté dans ces articles est une variante simplifiée des cryptosystèmes BGV [BGV14] et FV [FV12]. Nous avons choisi ce point de départ car cet algorithme ne s'encombre pas de questions d'homomorphie — qui nécessitent une augmentation de la taille des paramètres, ou de résistance aux attaques quantiques — qui complexifient les algorithmes, qui sont au centre d'une majeure partie de la recherche sur la cryptographie basée sur les réseaux euclidiens. Nous avons transformé cette proposition à l'aide de techniques arithmétiques, telles que la décomposition RNS [BEHZ17], la transformation NTT [LN16a] et l'algorithme de multiplication modulaire de Montgomery [BLSK98], afin de bénéficier de tables de correspondance. Ces transformations sont nécessaires pour pouvoir transformer notre algorithme initial en une série de tables de correspondance. Sans ces transformations, cette mise en table serait impossible dû à une consommation mémoire excessive. Par ailleurs, nous avons utilisé des propriétés homomorphiques latentes à notre cryptosystème pour créer des encodages homomorphes. Ce type d'encodage différent dispose de propriétés intéressantes pour résister aux attaques algébriques dans un modèle d'attaque en boîte blanche.

Nous avons décomposé notre proposition en deux segments :

- Une proposition initiale sans encodage externe respectant des limitations spécifiques (principalement sur la taille de nos table de correspondance) pour assurer la possibilité d'une utilisation pratique. Bien que cette proposition soit résistante aux attaques en boîte blanche dérivées d'attaques par canaux auxiliaires, nous montrons que cette proposition initiale est toujours vulnérable à une certaine forme d'attaques algébriques.

- Deux contre-mesures supplémentaires qui peuvent être appliquées à notre proposition initiale pour la rendre résistante aux attaques algébriques. Cependant, ces deux mesures limitent l'utilisation large en tout type de scénario de notre proposition. La première technique utilise des encodage externe, ce qui impose une configuration spécifique qui n'est pas toujours possible dans une application qui utilise la cryptographie en boîte blanche. La seconde technique, en revanche, impose un surcoût en mémoire qui dépasse notre limite initiale de taille de table d'un facteur d'au moins 3.

Nous avons séparé les deux contre-mesures de la proposition initiale car nous souhaitions :

- d'une part, une proposition générique qui ne suppose rien de la part de l'utilisateur,

- d'autre part, une proposition moins générique capable de rendre notre proposition totalement résistante au modèle en boîte blanche si l'utilisateur peut mettre en place des configurations spécifiques supplémentaires.

Notre proposition a été entièrement implémenté en python et est la première proposition publique de chiffrement asymétrique en boîte blanche. Le code de cette application sera mis à disposition des rapporteurs.

Dans le quatrième chapitre, nous présentons notre travail sur les bijections polynomiales modulo $2^n$. Les bijections polynomiales modulo $2^n$ peuvent être utilisées dans des techniques d'obscurcissement utilisant des MBA, et sont un outil puissant pour créer des constantes et prédicats opaques. Une classification de toutes les bijections polynomiales modulo $2^n$ a été présentée dans l'article [Riv01]. Cependant, cet article ne fournit pas d'algorithme pour inverser ces permutations. Ce processus d'inversion est essentiel pour l'utilisation des bijections polynomiales modulo

$2^n$ dans la création de MBA, car le polynôme et son inverse sont tous deux utilisés dans ce processus. L'article [ZMGJ07] fournit une méthode pour inverser un sous-ensemble de ces bijections, ainsi qu'une méthode pour construire les expressions MBA correspondantes. Cependant, dans l'article [BJLS17], une attaque est présentée contre les MBA qui reposent sur certaines propriétés spécifiques au sous-ensemble utilisé dans [ZMGJ07].

Nos travaux fournissent un moyen efficace basé sur la méthode de Newton pour inverser toutes les bijections polynomiales modulo $2^n$, y compris les polynômes qui ne correspondent pas aux propriétés spécifiques utilisées dans l'attaque décrite dans [BJLS17]. En outre, nous fournissons une autre méthode d'inversion des bijections polynomiales modulo $2^n$ basée sur l'interpolation de Lagrange. Cette méthode à l'avantage d'avoir une complexité déterministe ce qui permet d'avancer des arguments de sécurité plus étayés. Ces travaux ont donné lieu à deux articles diffusés dans des conférences internationales [BERR16] [BKRS18].

La cryptographie en boîte blanche et l'obscurcissement sont deux domaines différents mais intimement liés. Ils constituent tous deux des outils importants en matière de protection des logiciels. En pratique, l'obscurcissement doit être utilisé comme une mesure de protection suplémentaire au dessus d'un cryptosystème en boîte blanche. Malheureusement, elle ne suffit pas toujours à prémunir des attaques du modèle en boîte blanche comme nous avons pu le constater lors des deux challenges WhiBox, [CHE17] et [CHE19]. Ainsi, une proposition protégé par de l'obscurcissement soumise au challenge [CHE19] a été cryptanalysé [GPRW19].

## Une thèse en entreprise

Cette thèse a été réalisée dans le cadre d'une thèse CIFRE, conjointement entre le Laboratoire d'Informatique Paris 6 (LIP6) et la société Quarkslab. La société Quarkslab est une entreprise française, dont les ingénieurs travaillent dans le domaine de la sécurité informatique. La recherche est une préoccupation centrale pour Quarkslab. L'entreprise s'efforce de toujours être au fait des dernières techniques de conception, d'analyse ou d'attaque de produits de sécurité et de contribuer à l'état de l'art. Ses connaissances sont ensuite mises à contributions dans le cadre d'audits de sécurité ou lors du développement de produits logiciels de sécurité comme l'obfuscateur EPONA. Par ailleurs, cette thèse a été réalisée sous la direction de Jean-Claude Bajard, au sein de l'équipe ALMASTY. Cette équipe s'intéresse principalement à l'utilisation d'algorithmes issus de la théorie des nombres avec application à la cryptographie.

L'analyse et la conception de cryptographie en boîte blanche intéresse Quarkslab à plus d'un chef. En effet, un des piliers de l'activité d'évaluation de l'entreprise repose sur la rétro-ingénierie logicielle, la mettant en situation privilégiée pour fournir un contexte scientifique élargi de la zone purement cryptographique du sujet. L'attaque des schémas en boîte blanche fait l'objet d'un effort scientifique significatif dans l'entreprise se matérialisant dans des résultats internes mais également publics [BHMT16]. L'aspect constructif de la protection logicielle fait également partie des problématiques de Quarkslab au travers de l'obfuscateur EPONA que l'entreprise développe. Les travaux de cette thèse ont donc pris place à un carrefour entre cryptographie, rétro-ingénierie et obscurcissement, trois thèmes tous présents scientifiquement à Quarkslab.

En parallèle, mon rattachement au LIP6 et à l'équipe ALMASTY m'a permis de rester en phase avec le monde de la recherche académique. Cette équipe a pu démontrer l'utilité de nombreuses techniques issues de la théorie des nombres et de l'arithmétique des ordinateurs dans le contexte de la cryptographie asymétrique [BLSK98] [BEHZ17]. Ces techniques ont été reprises et adaptées au cours de cette thèse dans notre proposition pour une boîte blanche asymétrique basée sur les réseaux euclidiens.

Au cours de cette thèse, j'ai aussi eu l'opportunité de travailler au sein de Quarkslab sur l'audit de sécurité du logiciel Particl[1]. Particl est une plate-forme proposant un protocole décentralisé orientées *privacy* de transactions multi-cryptomonnaies. Il permet l'anonymisation de transactions en ligne par l'utilisation des techniques *bulletproof* [BBB+18] et *MLSAG* [Noe15]. Participer à cet audit m'a permis d'appliquer mes connaissances en cryptographie pour l'évaluation d'un produit logiciel implémentant des techniques avancées de l'état de l'art académique récent. Ce type d'évaluation se situe à l'intersection entre des articles académiques, qui ne sont pas des spécifications, et des implémentations de la vie réelle, ce qui en fait toute la difficulté et l'intérêt scientifique et technique. Le rapport public de cet audit est disponible en ligne [2].

[1]https://particl.io/
[2]https://blog.quarkslab.com/security-audit-of-particl-bulletproof-and-mlsag.html

# Introduction

Information technology security is a vast area of research that is becoming increasingly complex as the technology itself becomes more complex. Traditionally, research on this topic has focused on a model involving a remote attacker attempting to gain access to or control of information on a device or network. However, modern technology has added a new threat model to the equation. How to assess the security of a device when the attacker is not remote at all ? For example, one can consider the case of Digital Right Management (D.R.M.) where the end user of a product (a player, a modem user, etc. . . ) is considered hostile towards copyrights. Another example is cloud computing, where a cloud computing provider may be considered hostile to security issues, such as the anonymity of its users or the confidentiality of data stored and potentially processed on its servers. This new context motivates the work carried out in this thesis. This work focused in particular on two protection techniques, white box cryptography and obfuscation. These two techniques can be combined in a wider domain that is software protection.

## Software Protection

The term *software protection* refers to the protection techniques used to prevent attacks by the end user (or an attacker with the same prerogatives) of an application on a computer system. These types of attacks are called *Man-At-The-End* attacks. (MATE). Although there is a growing need for protection against these types of attacks (software embedded in mobile devices, D.R.M. of products delivered to the customer, cloud computing), protection against MATE attacks remains a difficult problem. Indeed, we have to consider an attacker who has the same access and power as a legitimate end user of the software, software that processes the sensitive information we want to protect. Such an attacker can alter the software itself, which raises the question of code integrity. They may also reverse engineer the software, which allows them to better understand how it works. This understanding may allow an attacker to access sensitive data or may lead to higher level attacks. In such an attack model, an attacker has access to all data processed by the software and is able to modify the input of any function called by the software.

Software protection aims to solve the problem of security in such a hostile model using various tools. Among these tools are white box cryptography, obfuscation, anti-tampering techniques, etc. . . . During this thesis, we considered two problems related to software protection, white box cryptography on the one hand and obfuscation on the other hand. In particular, we designed, implemented and proposed a first construction for asymmetric white box encryption. Contrary to symmetric white box cryptography, the subject of asymmetric white box cryptography is almost absent from the literature, despite its use by the industry (as attested for example by the following patents : [MG09], [AAMSD18] and [HM16]). Therefore, our proposal is the first available proposal in the literature. We have also studied polynomial bijections modulo $2^n$ as described in the article [Riv01]. These bijections are used in mixed arithmetic-Boolean mixed expressions (MBA) which are themselves used in obfuscation techniques, such as the techniques presented in [ZMGJ07], using a subset of polynomials whose inverses are describable by a closed form formula. This subset proved vulnerable to an attack presented in[BJLS17]. We propose two new methods to invert the original set of polynomial bijections presented in [Riv01]. We can thus use the obfuscation techniques presented in [ZMGJ07] while resisting the attack presented in [BJLS17].

In this summary, we will first present cryptography and its different attack models, in particular the white box model. Then, we will present obfuscation as a software protection tool. Finally, we will summarize the work done during this thesis and present its organization and contributions.

## Cryptography

Cryptography is the science of information protection. In a simplified way, it uses complex mathematical objects to create a couple of functions. A function to transform information into protected data, and a function to access the protected content. The best known and oldest cryptographic functionality is encryption. In this case, the pair of functions is the encryption and decryption. These two functions are used to ensure that data is only accessible to a set of legitimate entities. The historical evolution of cryptography into the science of information protection has led to the development of algorithms submitted for evaluation — public for standards, set by a key of which a secret part is the guarantee of security of the data exchange process. Various models of attacks exist according to the preliminary data known by the attacker, those that he can collect during the attack and his computing power. A cryptosystem is considered secure when an illegitimate entity cannot extract any information from an exchange of data cryptographically protected between several legitimate entities.

Note that cryptography is also interested in the problem of the authentication of a data's origin, but also in the advanced and complex functionalities such as homomorphic cryptography, signature schemes with designated verifiers, electronic vote, the sharing of secrets, cryptocurrency and many other developments.

We can differentiate two families of cryptographic functions:

- secret key cryptography, also called symmetric cryptography, which includes the AES and DES cryptosystems for example,
- public-key cryptography, also known as asymmetric cryptography, which includes the RSA and ECC cryptosystems for example.

Symmetric cryptography is the oldest form of cryptography. In this configuration, two people who communicate each use a cryptographic key identical to the one used by the other. Both users can use this key to encrypt or decrypt messages. The encryption process is a transformation applied to a message that depends on the value of the message and the chosen cryptographic key. The decryption consists in performing the reverse transformation on an encrypted message. The knowledge of the cryptographic key makes it possible to carry out both transformations. Symmetric cryptography is generally faster and lighter in memory than its asymmetrical counterpart. It is therefore used for the rapid processing of data, or in constrained contexts such as in the case of large amounts of data or embedded equipment. Key management or key sharing remains however a complicated problem to solve with tools from symmetric cryptography only.

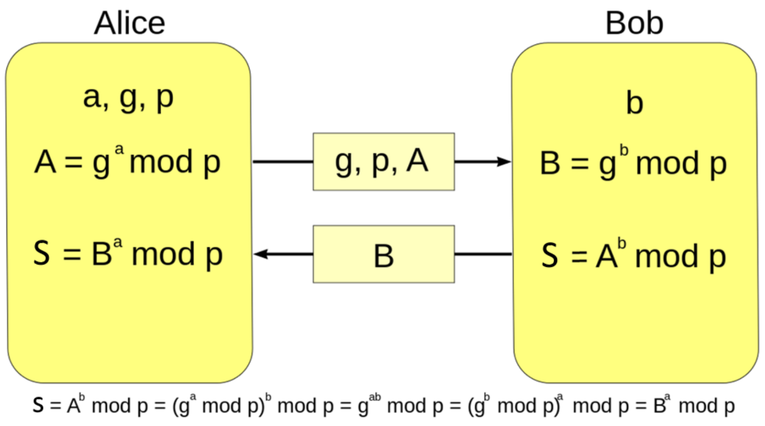
In 1976, Whitfield Diffie and Martin Hellman conceptualized the idea of a public/private (bi-)key cryptosystem. That is to say a cryptosystem for which the encryption and decryption keys are different. From their work stems the Diffie-Hellman key exchange protocol (renamed Diffie-Hellman-Merkle in 2002 to recognize Ralph Merkle's contribution to the creation of the concept of public key cryptography). This technique is based on on the use of a cryptographic trapdoor, a function that is easy to calculate in one direction (in their case a modular exponentiation), but whose reverse operation is difficult to carry out (in their case, solving the discrete logarithm problem).

Prior to the key negotiation, there is a step of generation for each participant of a pair of keys : one public key that will be distributed to other participants and a private key which will have to remain a secret. In the case of a negotiation between two participants, each participant combines their private key with the public key of the other to produce the key negotiated between the two parties. This negotiated key has the following properties:

- it is unique to both participants. Statistically, no other pair of participants can obtain the same value,
- it does not require participants to exchange their private keys,
- finding this negotiated key without knowing the private keys of both participants is a difficult problem.

This protocol therefore allows two participants to agree on a common value without ever exchanging that value. This value can then be used to determine a symmetrical cryptographic key between the two participants without having to exchange it in the communication channel.

To fix the ideas we will take as an example a Diffie-Hellman-Merkle key negotiation based on the discrete logarithm problem in the multiplicative group of integers modulo $p$, with $p$ a prime number. Let $g$ be a generator of this group. Let $a \in \mathbb{Z}/p\mathbb{Z}$ be the private value of Alice, a participant in the key negotiation. Respectively, let $b$ be the private value of Bob, another participant. Suppose Alice calculates the value $g^a \mod p$ and broadcasts it on a communication channel. The discrete logarithm problem assumes that it is difficult to find $a$ with only the values $(g, p, g^a)$ known. Bob can perform a similar calculation to send the value $g^b \mod p$. They can both calculate the common value $(g^a)^b \mod p = (g^b)^a \mod p$ without ever releasing their private value. The protocol is schematized as follows:


Alice
Bob
a, g, p
A = g^a mod p
S = B^a mod p
g, p, A
B
b
B = g^b mod p
S = A^b mod p
S = A^b mod p = (g^a mod p)^b mod p = g^ab mod p = (g^b mod p)^a mod p = B^a mod p

In 1978, this idea was taken up by Ron Rivest, Adi Shamir and Leonard Adleman in their presentation of the RSA asymmetric cryptosystem which implements the idea of asymmetric encryption. The security of this cryptosystem is based on the difficulty of the factorisation problem of an integer produced by the product of two large prime numbers (called the RSA problem). Let $N = pq$ a large integer, product of two prime integers. Let $\Phi(N) = (p-1)(q-1)$ the order of the group $\mathbb{Z}/N\mathbb{Z}$. Each user has a public cryptographic key, $e$, which will be distributed to all participants, and a private cryptographic key $d$. These two values are generated as :

$$ed = 1 \mod \Phi(N)$$

From the Fermat-Euler theorem follows the following result:

$$x^{ed} \mod N = x \mod N \qquad \forall x \in (\mathbb{Z}/N\mathbb{Z})^\star$$

Any participant can use another participant's public key to encrypt a message $m$ to their attention by calculating the following value:

$$c = m^e \mod N$$

Only the recipient has the ability to decrypt this message using his private key. To do so, it calculates the following value:

$$c^d \mod N = (m^e)^d \mod N = m^{ed} \mod N = m \mod N$$

and thus obtains the message value $m$. Note that the examples presented are only a mathematical illustration of the principles described. They do not constitute the specification of the algorithms implemented in practice, which include numerous additions to handle special cases, optimizations or protections against various attacks taken into account since their invention.

Asymmetric cryptography is more difficult to manipulate than symmetric cryptography. It requires more complex (and therefore more time-consuming) computations and struggles to encrypt/decrypt a large amount of data. It is generally used for :

- negotiate symmetrical keys within the framework of a key exchange protocol,
- solve participant authentication issues,
- exchange small messages.

Recently, work on quantum algorithms (algorithms that can be performed by a quantum computer) has shown that many supposedly difficult problems could be solved using a quantum computer [Sho97]. Although the current state of quantum computer research does not yet allow to perform such attacks, the NIST (National Institute of Standard and Technology) has launched a competition to find new standards for asymmetric cryptography that could resist quantum attacks. Among these proposals are cryptographic algorithms based on lattices.

When we try to evaluate the resistance of a cryptographic algorithm, whether symmetric or asymmetric, we need to characterize the attacker, i.e. describe precisely what his capabilities, knowledge and goals are, and thus formalize the model. We are now going to define the models of black box, grey box and white box attacks.

## Attack Models in Cryptography

Let us take a second look at the simple example of two users exchanging messages. This example is a classic configuration in which an attacker is modeled as an external entity that only has access to communications when they are sent from one user to the other. Such a model for an attacker is called a model in the black box context. In a black box context, an attacker only has access to the inputs/outputs of a cryptographic scheme and sees the cryptographic functions as oracles, transforming one into the other (he has no knowledge of the internal calculations performed by the cryptographic functions, hence the term *black box*).

The attacker tries to obtain secret information, such as a decrypted message or the value of a cryptographic key, from exchanges over the communication channel. He may, for example, replay communications or expose a statistical bias in the content of encrypted exchanges. On the other hand, it cannot make any observation on the internal functioning of the entities participating in the communications and cannot modify their behaviour.

Modeling an attacker in a black box context is the most classical way of modeling threats to an information system whose security relies on cryptography. However, real-life applications are sometimes opposed to attackers who have access to more resources than what is modeled by the black box context.

For example, consider our previous setup where an attacker may be physically present near a device performing one of the cryptographic functions. This device is probably gonna be an electronic device (computer, telephone, smart card, etc.). At close range, electronic components tend to leak all kinds of additional information, such as energy consumption or electromagnetic wave emission. These sources of additional information can help an attacker carry out an attack. For example, attacks measuring execution time (*timing attacks*) and differential power analysis (DPA) attacks [KJJ99] are two specific attacks that are made possible by the attacker's ability to listen to the physical manifestation of a device performing a cryptographic operation. In addition, physical proximity can allow the attacker to physically alter the state of a device while performing a cryptographic operation. For example, fault-based attacks, which use the injection of a *fault* (modification of data in memory) to obtain information about the internal functioning of a system, have become an increasingly common security concern in modern cryptography. These types of attacks are commonly referred to as *side-channel attacks*. An attacker in such a model has more information about the encryption/decryption operations than a black-box attacker, without having full access to the device performing the operation. For this reason, a model that takes these types of attacks into account is generally referred to as a gray box attack model.

An attacker in a gray box model can interact with a device performing a cryptographic operation without having control of the device. He can take all kinds of physical measures (power consumption, EM emission, etc. . . ) while the device is performing operations and modify its behavior via fault injection (by modifying the content of a register with a laser for example). It can use this information to make assumptions about the internal calculations performed during the cryptographic operation. A grey box attacker will try to use these hypotheses to help him obtain a secret information manipulated by the device, such as a cryptographic key. For example, being able to determine the Hamming weight of a cryptographic key can greatly reduce the effort required to determine the value of that key.

Let us now consider an attack model that deals with *Man-At-The-End* type of attacks. An attacker in such a model would have access to many more resources than the classic black box model, or even the gray box model. Indeed, such an attacker would have the same knowledge and access as a legitimate user of the device performing cryptographic operations. He would therefore have access to the contents of the memory and to the implementation of the cryptographic functions. Such a hostile model is called an attack model in a white box context.

In a white box model, we consider an attacker who has total control over the device performing cryptographic functions, and who seeks to extract security keys from it. The attacker can read any content stored in memory, manipulate the code involved in the execution of cryptographic functions, add breakpoints or modify any variable in the environment during execution, etc... He can use these capabilities to mount context-specific attacks in white box, such as algebraic BGE type attacks [BGEC05], or adapt attacks from the gray box model. Indeed, techniques such as fault injection are much easier to perform in a white box context (no need to modify a bit with a laser when you can simply change the content of a register). This is why white box cryptography seeks to hide the key in the implementation code itself so that the observations an attacker is able to make do not allow him to deduce its value.

Note that we do not try to prevent the attacker from using these cryptographic functions. Indeed, contrary to the black box and grey box contexts, the white box attacker has a legitimate access to the cryptographic functions.

In the second chapter 2 of our manuscript, we discuss the current state of symmetrical white box cryptosystem proposals with emphasis on the white box proposals of the AES cryptosystem. In the third chapter 3, we discuss our own proposal for the design of an asymmetric white box cryptosystem based on lattices.

## Obfuscation

Obfuscation is the means by which software protection combats the process of reverse engineering. Reverse engineering in computer science is the study of an executable binary and the environment on which it is run in order to understand what it does and the details of how it performs certain operations. For example, reverse engineering may seek to obtain information about how a software program works in order to produce a more efficient copy (for example, by removing certain protections). Reverse engineering can also be used to extract key data from an application, such as cryptographic keys or initialization values for certain security protocols.

Since practical targets are usually binary files that are not supposed to be human-readable, reverse engineering must use different tools to analyze and/or simplify the target in order to get a better understanding of how it works. Reverse engineering is generally classified into two types of approaches :

- static analysis, in which the reverse engineer considers the executable file as a static target, seeking to extract as much information as possible from it. This type of analysis is the most common because it does not require any additional configuration or prerequisites regarding the attacker's ability to manipulate the target's environment. This type of scan may also be the only one available if the attacker cannot (or will not) modify and/or control the environment of the device on which the executable file is run. In this configuration, reverse engineering can only access the code itself. This means, for example, that it can search for constants, but it must predict the values of the registers during execution.

- Dynamic analysis, in which the reverse engineer can observe and/or modify the executable while it is running in its environment (or in a replicated environment). This type of analysis is usually much more powerful because it allows the reverse engineer to study the control flow of an application, access the contents of registers during execution or even insert breakpoints to study different execution states. However, this type of analysis is more difficult to implement. It requires some control over the execution environment and many software applications use techniques, both software and hardware, to guard against this type of control.

Obfuscation is intended to hinder the reverse engineering process by modifying the target (an executable binary) to make it more difficult for the attacker to understand what is being processed and how. A very theoretical definition of obfuscation is *indistinguishable obfuscation*, the definition of which is discussed in [BGI+12] and [GR07]. This definition considers the notion of indististinguishability associated with $C0$ and $C1$, two equivalent circuits of similar size. The obfuscation of $C0$ and $C1$ must be indistinguishable, meaning that the obfuscated circuits must not show any significant difference (in operations or memory access for example) when executed for a fixed input value. In practice, however, the notion of *indistinguishable obfuscation* is extremely difficult to implement.

Here are some examples of obfuscation tools and what they try to protect :

- Opaque predicates which replace a constant value with a complex expression that is difficult to simplify. This technique can be used to add false branches of code. This will make the static analysis more difficult as it will have to consider code that has no meaning with respect to the actual operations performed.

- The flattening of the control flow graph that modifies the control flow (the order in which operations are executed) of an executable target into a series of blocks of operations whose flow is managed by a dispatcher (a program that will reorganize the order in which the different operations are executed), itself obfuscated. This technique aims at hindering static analysis because it makes it more difficult for the reverse engineer to understand in which order the operations are executed. To a lesser extent, this technique can also hamper dynamic analysis if the dispatcher is obfuscated accordingly.

- Opaque constants which replace the constant values used by the application with complex expressions, masked values or even encrypted values. This technique is used to mask the exact values of key constants used by the application. For example, the scheduling values used by the dispatcher introduced in the control flow graph flattening technique, the initiation vectors used for some security protocols, the constant values used in hash functions, etc. . . .

- Anti-extraction techniques which aim to remove the ability of the reverse engineer to extract code from the target in order to place it in a controlled environment. Although not always considered an obfuscation technique in the literature, this technique aims to disrupt dynamic analysis by preventing the reverse engineering process from analyzing the target in a more controlled environment.

One obfuscation technique that can be used to make complex expressions for both predicate and opaque constants is the use of Mixed Boolean-Arithmetic (MBA) expressions. This technique is based on the assumption that while there are many efficient ways to simplify either Boolean or arithmetic expressions, complex expressions that mix these two types of operations are more difficult to simplify. The resistance of MBA expressions to different simplification methods and tools is studied in [Eyr17]. A key tool needed to make MBA expressions are bijective functions and their inverse, as both are necessary to apply obfuscation (one is pre-calculated during the obfuscation process while the other is used during program execution). In the fourth chapter of this thesis we present a way to produce and invert polynomial bijections modulo $2^n$, bringing new polynomial bijections compared to those previously presented in [ZMGJ07]. These new types of polynomials allow an MBA-based obfuscation technique that resists a specific attack presented in [BJLS17].

## Organisation and Contribution of this Thesis

This thesis is organized in three axes preceded by a first chapter which summarizes the different notions used throughout this thesis such as: the AES cryptosystem, the basics of lattice-based cryptography, the RNS decomposition and the NTT transformation.

In the second chapter 2, we discuss the state of symmetric white box cryptography with emphasis on proposals for the implementation of the AES white box cryptosystem. We focus our attention on the following issues:

- What are the properties of AES that facilitate the creation of white box cryptosystems?
- What are the properties of AES that make these proposals still vulnerable today?

The purpose of this discussion is to determine how to produce a white box cryptosystem proposal based on a different algorithm without repeating the inherent flaws of AES in a white box attack model. Our conclusions on this topic are that the implementation of a cryptosystem through the use of lookup tables remains the key to the production of a white box model. The AES cryptosystem is particularly well suited to such an implementation because it performs a series of internal calculations on small data sizes. However, the different proposals of white box AES algorithms present the following security problems:

- The AES cryptosystem is by design extremely vulnerable to numerous side-channel attacks, which can be adapted to the white box model to become even more powerful [BHMT16] [AMR20]. These types of attacks still make up the majority of white box attacks on high-end proposals.
- Encoding techniques are weakened because they are constrained by the AES structure. Due to the presence of non-linear transformations in the AES, the encodings applied to the look-up tables must be removed immediately in the following look-up table (or almost immediately in the case of mixing bijections [CEJO03]). This opens the way to another type of attack that aims at removing these encodings to access the secret key: the algebraic attacks [BGEC05].

Our objective in the third chapter 3 is to make a proposal that still uses the mapping table-based implementation technique used in AES white box proposals, but applied to a different cryptosystem that is, to our knowledge, less vulnerable to side-channel attacks, and that uses latent homomorphic properties to solve the problem of algebraic attacks. In this chapter, we present our proposal for an asymmetric white box based on lattices.

Our proposal is based on the construction proposed in articles [GAGL15] and [AMBG$^+$16]. The cryptographic algorithm presented in these articles is a simplified variant of the BGV [BGV14] and FV [FV12] cryptosystems. We have chosen this starting point because this algorithm is not burdened with homomorphic issues — which require an increase in the size of the parameters, or resistance to quantum attacks — which complicate the algorithms, which are at the center of much of the research on lattice-based cryptography. We have transformed this proposal using arithmetic techniques, such as the RNS decomposition [BEHZ17], the NTT transformation [LN16a] and Montgomery's modular multiplication algorithm [BLSK98], in order to benefit from lookup tables. These transformations are necessary to be able to transform our initial algorithm into a series of lookup tables. Without these transformations, this mapping would be impossible due to excessive memory consumption. Moreover, we used homomorphic properties latent to our cryptosystem to create homomorphic encodings. This different type of encoding has interesting properties to resist algebraic attacks in a white box attack model.

We have broken our proposal down into two segments:

- An initial proposal without external encoding respecting specific limitations (mainly on the size of our lookup tables) to ensure the possibility of practical use. Although this proposal is resistant to white box attacks derived from side-channel attacks, we show that this initial proposal is still vulnerable to some form of algebraic attacks.

- Two additional countermeasures that can be applied to our initial proposal to make it resistant to algebraic attacks. However, these two measures limit the wide use in any kind of scenario of our proposal. The first technique uses external encoding, which imposes a specific configuration that is not always possible in an application that uses white box cryptography. The second technique, on the other hand, imposes an additional cost in memory that exceeds our initial table size limit by a factor of at least 3.

We separated the two countermeasures from the original proposal because we wanted:

- on the one hand, a generic proposal that assumes nothing on the part of the user,

- on the other hand, a less generic proposal capable of making our proposal totally resistant to the white box model if the user can set up additional specific configurations.

Our proposal has been fully implemented in python and is the first public proposal for asymmetric white box encryption. The code of this application will be made available.

In the fourth chapter 4, we present our work on polynomial bijections modulo $2^n$. Polynomial bijections modulo $2^n$ can be used in obfuscation techniques using MBA, and are a powerful tool to create opaque constants and predicates. A classification of all permutation polynomials modulo $2^n$ has been presented in the article [Riv01]. However, this article does not provide an algorithm to invert these permutations. This inversion process is essential for the use of polynomial bijections modulo $2^n$ in the creation of MBA, because both the polynomial and its inverse are used in this process. The article [ZMGJ07] provides a method to invert a subset of these bijections, as well as a method to construct the corresponding MBA expressions. However, in the article [BJLS17], an attack is presented against this type of MBA based on some properties specific to the subset used in [ZMGJ07].

Our work provides an efficient tool based on Newton's method to invert all permutation polynomials modulo $2^n$, including polynomials that do not match the specific properties used in the attack described in [BJLS17]. In addition, we provide another method for inverting permutation polynomials modulo $2^n$ based on Lagrange interpolation. This method has the advantage of having a deterministic complexity, which allows to put forward better security arguments. This work has led to two papers published in international conferences [BERR16] [BKRS18].

White box cryptography and obfuscation are two different but closely related fields. They are both important tools in software protection. In practice, obfuscation should be used as an additional protection measure on top of a white box cryptosystem. Unfortunately, it is not always sufficient to prevent attacks from the white box model as we have seen in the two WhiBox challenges, [CHE17] and [CHE19]. Thus, a proposal protected by obfuscation submitted to the challenge [CHE19] has been cryptanalyzed [GPRW19].

## A Thesis in Partnership with the Industry

This thesis was carried out as part of a CIFRE thesis, jointly between the *Laboratoire d'Informatique Paris 6* (LIP6) and the Quarkslab company. The Quarkslab company is a French company, whose engineers work in the field of computer security. Research is a central concern for Quarkslab. The company strives to always be up to date with the latest techniques for designing, analyzing or attacking security products and to contribute to the state of the art. This knowledge is then applied in security audits or in the development of security software products such as the EPONA obfuscator. Moreover, this thesis was carried out under the supervision of Jean-Claude Bajard, within the ALMASTY team. This team is mainly interested in the use of algorithms from number theory with application to cryptography.

The analysis and design of white box cryptography interests Quarkslab in many ways. Indeed, the reverse engineering of software security products is one of its main activities when evaluating its security, putting it in a privileged situation to provide broader scientific context than the purely cryptographic component of the subject. The attack of white box schematics is the subject of a significant scientific effort in the company materializing in results both internal and public [BHMT16]. The constructive aspect of software protection is also part of the concerns of Quarkslab through the EPONA obfuscateur that the company develops. The work of this thesis thus took place at a crossroads between cryptography, reverse engineering and obfuscation, three themes all present scientifically at Quarkslab.

At the same time, my attachment to LIP6 and the ALMASTY team allowed me to stay in touch with the world of academic research. This team was able to demonstrate the usefulness of many techniques from number theory and computer arithmetic in the context of asymmetric cryptography [BLSK98] [BEHZ17]. These techniques have been taken up and adapted in the course of this thesis in our proposal for an asymmetric white box based on lattices.

During this thesis, I also had the opportunity to work at Quarkslab on the security audit of Particl[3] software. Particl is a platform offering a decentralized protocol oriented *privacy* of multi-cryptocurrency transactions. It allows the anonymization of online transactions through the use of the bulletproof [BBB+18] and MLSAG [Noe15] techniques. Participating in this audit allowed me to apply my knowledge in cryptography for the evaluation of a software product implementing advanced techniques of the recent academic state of the art. This type of evaluation is at the intersection between academic papers, which are not specifications, and real life implementations, which makes it very difficult and of scientific and technical interest. The public report of this audit is available online [4].

[3] `https://particl.io/`

[4] `https://blog.quarkslab.com/security-audit-of-particl-bulletproof-and-mlsag.html`

# Chapter 1

# Toolbox

In this section, we will discuss various techniques and algorithms used in this thesis. For the sake of practicality, those will be presented in order of appearance in the rest of this manuscript. We will briefly give an introduction on the following topics:

- the AES cryptosystem, of which we will study state-of the art white-box proposals in chapter 3,
- the basics behind lattice-based cryptography, as our proposal for an asymmetric white-box in chapter 4 is based on such a cryptosystem,
- the Residue Number System (RNS) representation, as this representation technique will be used to allow for a fully table based implementation of our asymmetric white-box proposal in chapter 3,
- the Number Theoretic Transform (NTT) as this arithmetic technique will be used to allow for a fully table based implementation of our asymmetric white-box proposal in chapter 3,
- Montgomery's multiplication algorithm, as this modular multiplication technique will be used in chapter 3 to allow the use of both the RNS and NTT techniques in a way that differ from classical techniques commonly used in the implementation of lattice-based cryptography,
- We will discuss the challenges of using classical arithmetic techniques in the ring of integers modulo $2^n$. In particular, we will discuss Lagrange interpolation and Newton's inversion method, as those techniques will be adapted to the evaluated inversion problem encountered in chapter 4.

## 1.1 AES

The Advanced Encryption Standard, or AES, cryptosystem is a symmetric block cipher proposed in 1998 [DR98] and established in 2001 as a standard for symmetric key encryption. At its core, it encrypts a 16 byte state by performing a series of substitutions and permutation operations, one of which involves a secret key byte derived from the master key. To this date, this cryptosystem remains the standard in symmetric cryptography since its establishment. In this section, we will discuss the basic operations involved in the encryption/decryption process of the AES cryptosystem. Then we will discuss some known vulnerabilities of the AES cryptosystems against side-channel attacks, that is to say against an attacker in a grey-box model.

### 1.1.1 Basic Operations

AES performs a series of basic operations on a 16 byte state, the original state being derived from either a plaintext message (in the case of encryption) or a cyphertext (in the case of decryption). The resulting 16 byte block then becomes the new state, and the same series of substitution/permutation operations is performed again. Each cycle of those operations is referred to as a "round" of AES. By default, AES-128 performs 9 round of operations with one operation being absent from the last round. Note that the order in which we will present those operations is slightly different than the classical ordering for AES-128. This ordering was chosen to better match the ordering of operations presented in the original paper regarding AES white-box construction [CEJO03]. This paper also explains how to modify the aforementioned ordering without changing the behaviour of the AES-128 cryptosystem. For the sake of simplicity, we will only describe the encryption process of the AES cryptosystem. The decryption process simply being the application of the same basic operations in a reverse order.

#### Key Generation

The use of the AES-128 cryptosystem on a block of data requires a 128 byte key for each round performed. Those 128-byte round keys are generated from a single 128 byte master key through the use of the AES key-schedule algorithm. the first round key will match the original master key. Then a series of substitution/permutation functions, involving a set of static values *rcon* and the original Rijndael substitution box, will yield the following round keys.

Let us define the following operations:

- RotWord, a one byte circular shift of a 4 byte state,
- Subword, a substitution using the original Rijndael substitution box,

The key schedule compute the set of round keys derived from a master key $K$ in the following algorithm :

**Algorithm 1:** KeySchedule

```
input  : $K$
output: $W$
$N = 4//$ $R = 11//$ $W = [[] * 4] * 4R//$ for $i$ in range(4R) do
    if $i < N$ then
        $W[i] = K[i * 4 : i * 4 + 4]//$
    end
    if $i \geq N \&\& i = 0 \mod N$ then
        $W[i] = W[i - N] \oplus Subword(Rotword(W[i - 1])) \oplus rcon_{i/N}//$
    end
    else
        $W[i] = W[i - N] \oplus W[i - 1]//$
    end
end
return $W$
```

Once the key generation process is over, we have all round keys necessary to perform the AES cryptosystem.

#### ShiftRows

This operation simply permute the elements of our byte state with a static permutation. The following algorithm describes that process:

**Algorithm 2:** ShiftRow

```
input  : $state$
output: None
$tmp = state$
$b = [0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12, 1, 6, 11]$
for $i$ in range 16 do
    $state[i] = tmp[b[i]]$
end
```

#### AddRoundKey

This operation performs a *xor* operation between each byte of the current state and the matching round key.

**Algorithm 3:** AddRoundKey

```
input  : $state, k$
output: None
for $i$ in range(16) do

end
$state[i] = state[i] \oplus k[i]$
```

Note that this operation involves the secret key. In a white-box attack context where one seek to hide the secret key from the end-user performing the cryptographic operation, this is the operation that will require protection from the attacker.

#### SubByte

This operation performs a substitution of each byte of the state with respect to the original Rijndael substitution box.

**Algorithm 4:** SubByte

```
input  : $state, S$
output: None
for $i$ in range(16) do
    $state[i] = S[state[i]]$
end
```

#### MixColumn

This operation permute the elements of our current state by performing a product of our current state, represented as a $4 * 4$ matrix of bytes, with a static matrix in $GF(2^8)$.

**Algorithm 5:** MixColumn

```
input  : $state, M$
output: None
$state = matrice\_product(state, M)$
```

Note that this operation can be rewritten as a set of matrix/vector products followed by a reduction phase in $GF(2^8)$. This will be relevant when considering AES white-box proposals.

**Algorithm 6:** MixColumn

```
input  : $state$, $M$
output: None
$tmp = [[] * 16] * 4$
for $i$ in range(16) do
    $tmp[i] = vector\_product(state[i], M[i\%4]$
end
for $i$ in range(16) do
    $state[i] = tmp[4*(i/4)][i\%4] \oplus tmp[4*(i/4)+1][i\%4] \oplus tmp[4*(i/4)+2][i\%4] \oplus tmp[4*(i/4)+3][i\%4]$
end
```

### The AES Encryption Algorithm

Putting everything together, the following algorithm depict an encryption of a state block using AES-128:

**Algorithm 7:** AES

```
input  : $state$, $K$, $S$, $M$, $round$
output: None
$k = KeySchedule(K)$
for $r$ in range(round) do
    $ShiftRow(state)$
    $AddRoundKey(state, k[r])$
    $SubByte(state, S)$
    $MixColumn(state, M)$
end
```

## 1.1.2 Classical Attacks Against AES

While AES remains the standard regarding symmetric key encryption, it has been found vulnerable to certain type of attacks against an attacker in a grey-box or white-box model. In particular, we will detail the Differential Fault Attack [DLV03] [TH16] in a grey-box model as it is easily adaptable to the white-box model. Other attacks include Differential Power Analysis [KJJ99] which correlate power consumption during the execution of an AES implementation with its secret key, and other variants that use electromagnetic emissions. We will not discuss white-box attacks against AES white-box proposals as they will be further discussed in chapter 2. Note that the following attack do not impact the security of the AES cryptosystem in a black-box attack model. In this model, AES remains a secure cryptosystem that should be used.

### Differential Fault Attacks

Fault attacks aim to recover the secret key of an AES implementation by injecting a fault (which can be done by performing a bit-flip with the assistance of a laser) on one of the state bytes just before the last *MixColumn* operation. In this example, we will consider the Differential Fault Attack [DLV03] [TH16]. Let us consider a 16 byte state AES before and after the fault injections right before this last *MixColumn* operation:

$$\begin{Bmatrix} A & E & I & M \\ B & F & J & N \\ C & G & K & O \\ D & H & L & P \end{Bmatrix} \text{ and } \begin{Bmatrix} X & E & I & M \\ B & F & J & N \\ C & G & K & O \\ D & H & L & P \end{Bmatrix}$$

The remaining operations are the following:

- *MixColumn*
- *AddRoundKey* $k_9$
- *SubByte*
- *ShiftRow*
- *AddRoundKey* $k_{10}$

After the *MixColumn* operation, we get the following equation:

$$\begin{Bmatrix} 2A+3B+C+D & \dots & \dots & \dots \\ A+2B+3C+D & \dots & \dots & \dots \\ A+B+2C+3D & \dots & \dots & \dots \\ 3A+B+C+2D & \dots & \dots & \dots \end{Bmatrix} \text{ and } \begin{Bmatrix} 2X+3B+C+D & \dots & \dots & \dots \\ X+2B+3C+D & \dots & \dots & \dots \\ X+B+2C+3D & \dots & \dots & \dots \\ 3X+B+C+2D & \dots & \dots & \dots \end{Bmatrix}$$

After the remaining operation, we get the following states (for more details, we recommend [TH16]):

$$\begin{Bmatrix} S(2A+3B+C+D+k_{9,0})+k_{10,0} & \dots & \dots & \dots \\ \dots & \dots & \dots & S(A+2B+3C+D+k_{9,1})+k_{10,13} \\ \dots & \dots & S(A+B+2C+3D+k_{9,2})+k_{10,10} & \dots \\ \dots & S(3A+B+C+2D+k_{9,3})+k_{10,7} & \dots & \dots \end{Bmatrix}$$

and

$$\begin{Bmatrix} S(2X+3B+C+D+k_{9,0})+k_{10,0} & \dots & \dots & \dots \\ \dots & \dots & \dots & S(X+2B+3C+D+k_{9,1})+k_{10,13} \\ \dots & \dots & S(X+B+2C+3D+k_{9,2})+k_{10,10} & \dots \\ \dots & S(3X+B+C+2D+k_{9,3})+k_{10,7} & \dots & \dots \end{Bmatrix}$$

Looking at the first output byte $O_0$ (and its faulty output $O'_0$) we get the following equation:

$$O_0 = S(2A+3B+C+D+k_{9,0})+k_{10,0}$$

and

$$O'_0 = S(2X+3B+C+D+k_{9,0})+k_{10,0}$$

By computing the addition of both outputs, we get the following equation:

$$O_0 + O'_0 = S(2A+3B+C+D+k_{9,0})+k_{10,0}+S(2X+3B+C+D+k_{9,0})+k_{10,0}$$

$$O_0 + O'_0 = S(2A+3B+C+D+k_{9,0})+S(2X+3B+C+D+k_{9,0})$$

If we pose:

$$Y_0 = 2A+3B+C+D+k_{9,0}$$

$$Z = A+X$$

This becomes:

$$O_0 + O'_0 = S(Y_0) + S(2Z + Y_0)$$

By computing the same values for output bytes 7, 10 and 13, we get the following system:

$$O_7 + O'_7 = S(Y_1) + S(3Z + Y_1)$$
$$Y_1 = 3A + B + C + 2D + k_{9,3}$$
$$O_{10} + O'_{10} = S(Y_2) + S(Z + Y_2)$$
$$Y_2 = A + B + 2C + 3D + k_{9,2}$$
$$O_{13} + O'_{13} = S(Y_3) + S(Z + Y_3)$$
$$Y_3 = A + 2B + 3C + D + k_{9,1}$$

Only some values of Z can satisfy simultaneously these equations and, for each candidate value, there is a corresponding set of $Y_0$,$Y_1$,$Y_2$,$Y_3$. As we have also the following relations:

$$O_0 = S(Y_0) + k_{10,0}$$
$$O_7 = S(Y_1) + k_{10,7}$$
$$O_{10} = S(Y_2) + k_{10,10}$$
$$O_{13} = S(Y_3) + k_{10,13}$$

we get some candidates for values $k_{10,0}$, $k_{10,7}$, $k_{10,10}$ and $k_{10,13}$. The challenge now is to:

- apply a different fault to the same byte so as to increase the number of equations and reduce the number of possible values for $k_{10,0}$, $k_{10,7}$, $k_{10,10}$ and $k_{10,13}$,
- perform a similar attack on other bytes to determine the remaining key-bytes for key $k_{10}$.

Because the key schedule algorithm is fully invertible, recovery of $k_{10}$ yield all key-bytes involved in this instance of AES.

Both the injection of fault and the process of reiterating those faults on specific key-bytes can be challenging in a grey-box attack model. However, this is extremely easy to perform in a white-box attack model as the attacker has full control during execution (he can modify values, insert breakpoints, consult memory, etc. . . ).

## 1.2 Lattice-Based Cryptography

### 1.2.1 Mathematical Background

In this section we will give the basic mathematical background behind lattice-based cryptography. We will discuss the mathematical objects commonly handled for lattice-based cryptographic schemes and discuss hard problems involved with lattices. This will not be an exhaustive discussion, for more details on lattice-based cryptography, cyclotomic rings and the Closest Vector Problem/Shortest Vector Problem, we invite the reader to take a look at the following articles [LPR13] [Mic05].

### Lattices

Let $B = (v_0, v_1, \ldots, v_k)$ be a set of independent vectors in $\mathbb{Z}^n$. The lattice $L \subset R^n$ generated by this set of vectors, which we will refer as "basis", is the set of points determined by all linear equations of those vectors.

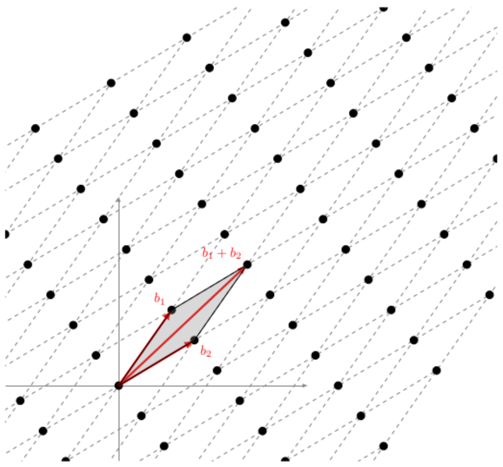
**Definition 1** *Given n linearly independent vectors $(v_0, v_1, \ldots, v_k)$ in $\mathbb{Z}^n$, the lattice generated by them is defined as:*

$$L(v_0, v_1, \ldots, v_k) = \left\{ \sum x_i v_i | x_i \in \mathbb{Z} \right\}$$

*We refer to $(v_0, \ldots, v_k)$ as a basis of the lattice. Equivalently, if we define B as the $n * k$ matrix whose columns are $(v_0, \ldots, v_k)$, then the lattice generated by B is:*

$$L(B) = L(v_0, v_1, \ldots, v_k) = \left\{ Bx | x \in Z \right\}$$

The rank of this lattice is $k$ and its dimension $n$. Bellow is an example of a lattice of dimension and rank 2:



### The Closest Vector Problem and the Shortest Vector Problem

Regarding lattices, there are two major proven hard problems that benefit lattice-based cryptography: the Closest Vector Problem (CVP) and the Shortest Vector Problem (SVP). CVP is the problem of finding the closest lattice point of a vector not contained in the lattice. On the other hand, SVP is the problem of finding a point of small norm within a lattice. Both problems are considered hard and most problems used in lattice-based cryptography can be reduced to solving either of them.

**Definition 2** *For any approximation parameter $\gamma = \gamma(n) \geq 1$, the search problem $\gamma$-CVP (Closest Vector Problem) is defined as follows: The input is a basis B for a lattice $L \subset \mathbb{R}^n$ and a vector $t \in \mathbb{R}^n$, the target. The goal is to output a vector $y \in L$ satisfying $xt - xk \leq \gamma \cdot dist(t, L)$.*

**Definition 3** *For any approximation parameter $\gamma = \gamma(n) \geq 1$, the search problem $\gamma$-SVP (Shortest Vector Problem) is defined as follows: The input is a basis B for a lattice $L \subset \mathbb{R}^n$. The goal is to output a vector $y \in L$ satisfying $y \leq \gamma \cdot \lambda_1(L)$.*

### Cyclotomic Rings and the Ring Learning With Errors

Most lattice-based cryptosystems involve elements of a cyclotomic ring:

$$\Phi_{q,n} = \mathbb{Z}[X]/(q, X^n + 1)\mathbb{Z}$$

Elements of the ring $\Phi_{q,n}$ are quite practical as they can either be seen as polynomials, allowing algorithmic techniques for fast polynomial arithmetic, or as vectors of degree $n$ with coefficients modulo $q$, allowing a reduction of problems in $\Phi_{q,n}$ to classical lattice problems like CVP or SVP. In chapter 3, we will mainly consider those elements as it ease the process of presenting the different algebraic techniques set in place to allow a fully table-based implementation of our asymmetric white-box proposal.

The Ring Learning With Error problem is a classic problem on which the security of many lattice-based cryptographic schemes are based. The main idea is given an element of a lattice that has deviated from the lattice through the addition of a small error vector, finding the original vector is considered a hard problem. Let $A_{s,\psi}$ be a sample from a RLWE distribution, with $s$ a secret vector from the cyclotomic ring $\Phi_{q,n}$ and $\psi$ an error distribution (vector with small coefficients), and let $a$ be an element of $\Phi_{q,n}$ generated uniformly at random. The sample distribution $A_{s,\psi}$ output the following:

$$(a, b = a \cdot s + e \mod q)$$

Let us define the Ring Learning With Error (RLWE) problem as follows :

**Definition 4** *Given samples (a,b) from an RLWE distribution $A_{s,\psi}$, find the value of s.*

This problem is at the core of most lattice-based cryptographic schemes. Solving this problem can be reduced to solving the approximate-SVP in an n-dimensional lattice [LPR13] and is therefore considered a hard problem.

### 1.2.2 Properties of Lattice-Based Cryptography

Lattice-based cryptography has been at the center of a lot of research given some very interesting properties. In 2009, Craig Gentry proposes a lattice-based cryptographic scheme as a Somewhat Homomorphic Encryption (SHE) scheme. He also proposes a technique called "bootstrapping" to transform his proposal into a Fully Homomorphic Encryption (FHE) scheme. An homomorphic cryptographic scheme is a cryptographic scheme that verifies the following assertion: given $E$, respectively $D$, an encryption, respectively decryption, function of a crytographic scheme $C$, and an operator $\oplus$, the cryptographic scheme C is said to be homomorphic with regards to operator $\oplus$ if the following holds:

$$D(E(m_1) \oplus E(m_2)) = m_1 \oplus m_2$$

That its to say, given two random messages $m_1$ and $m_2$, performing an operation $\oplus$ with their respective encryptions will yield, after decryption, the same result as if $\oplus$ was performed on the messages themselves.

Homomorphic properties are extremely interesting for security purposes as they allow for computation on encrypted data without the need for decryption. Applications vary from cloud computing to big data computations as it would allow computation on personal data without knowledge of the data themselves.

Not all homomorphic scheme are equals. A cryptographic scheme is referred to as a Somewhat Homomorphic Encryption scheme with regards to a complete set of operators $(\oplus, \otimes)$ if its homomorphie holds for a finite number of operations in $(\oplus, \otimes)$, meaning if too much operations are performed, decryption will no longer yield the correct result. On the other hand, a Fully Homomorphic Encryption scheme stays homomorphic regardless of the number of operations being performed on cyphertexts.

In chapter 3, we will make use of some homomorphic properties of our scheme to apply homomorphic encodings to our asymmetric white-box proposal. However, our scheme remains a Somewhat Homomorphic Encryption scheme, meaning its homomorphic capabilities will be limited by a fixed number of operations.

Another key feature of lattice-based cryptography is its theoretical ability to resist attacks performed with a quantum computer. The development of quantum computers is a major concern regarding the security of cryptography as many standards we use today are vulnerable to quantum attacks [Sho97]. In 2016, the NIST called for the proposal of cryptographic scheme that would resist the upcoming attacks [PQC16]. Many of those proposals are based on lattice problems, making it a great candidate for a future standard in asymmetric cryptography in a post-quantum world. While we do not seek to make a quantum resistant asymmetric white-box proposal, the fact that the next standard in asymmetric cryptography could be a lattice-based scheme comfort us in our choice to make a proposal for a lattice-based asymmetric scheme resistant to the white-box attack model.

## 1.3 The RNS Representation

The Residue Number System (RNS) is a method used for fast arithmetic operations. It is based on the Chinese Remainder Theorem that states the following :

**Theorem 1 (Chinese Remainder Theorem)** *Given a basis of co-prime integers $\beta = (p_0, p_1, \dots p_k)$, any integer modulo $B = \prod_{i=0}^{k} p_i$ has a unique representation in basis $\beta$. That is to say, any integer $X$ modulo $B$ has a unique representation $x = (X \mod p_0, X \mod p_1, \dots, X \mod p_k)$. Furthermore, given a set of residues $x = (x_0, x_1, \dots, x_k)$ with respect to basis $\beta$, the Chinese Remainder Theorem allows for the reconstruction of the corresponding unique value $X$ represented modulo $B$.*

The following algorithm depicts the reconstruction process:

**Algorithm 8:** CRT

```
input  : $x, \beta, l$
output: $X$
$X = 0$
$B = \prod_{i=0}^{l} \beta[i]$
for $i$ in range($l$) do
    $B_i = B/\beta[i]$
    $\tilde{B}_i = B_i^{-1} \mod \beta[i]$
    $X += (x[i] * B_i * \tilde{B}_i)$
end
return $X \mod B$
```

Another key property of the Chinese Remainder Theorem is that additions and multiplications of residues residue-wise yield, after reconstruction with the CRT algorithm, a matching result modulo $B$. Let $x = (x_0, x_1, \dots, x_k)$ and $y = (y_0, y_1, \dots, y_k)$

be two sets of residues matching integers $X$ and $Y$ modulo $B$. For addition as an example, we have the following:

$$z = (x_0 + y_0 \mod p_0, x_1 + y_1 \mod p_1, \ldots, x_k + y_k \mod p_k)$$

$$Z = CRT(z) = X + Y \mod B$$

The same result being true for multiplication.

Given computations involving integers modulo a large integer $B$, the RNS representation method simply consist in representing all values in basis $\beta = (p_0, p_1, \ldots, p_k)$, using the CRT algorithm for reconstruction. In this representation system, all computations can be performed on a set of residues, each of a smaller size than $B$. This can allow faster computation if the cost of performing the final reconstruction algorithm does not exceed the gain from performing operations on smaller integers.

## 1.4 The NTT algorithm

The Number Theoretic Transform (NTT) algorithm is a variant of the Fast Fourier Transform (FFT) on a number field that naturally possesses suitable roots of unity. The FFT algorithm represents a polynomial $R$ of degree $n$ and coefficients in $\mathbb{R}$ with the evaluation of $R$ on specific elements $(g_0, g_1, \ldots, g_n)$ for which the following holds:

$$g_i^n = 1 \qquad \forall i$$

Such elements are called roots of unity. The FFT states that such a representation is unique to a polynomial $R$, and $R$ can be easily reconstructed from its evaluated representation. Furthermore, the FFT allows for fast computation of additions and multiplications of high degree polynomials. Indeed, the result of additions of polynomials, respectively multiplications, can be computed by performing the same computations on their respective evaluations evaluation-wise. The result polynomial is then obtained by reconstructing the polynomial matching the resulting points of evaluation. Given the quadratic nature of polynomial products in particular, switching from a product of polynomials of degree $n$ to $n$ products of integers (the evaluated points) is a huge gain in performance.

However, the field $\mathbb{R}$ does not possess such roots of unity. Those can be found in the algebraic extension of $\mathbb{R}$, the set of complex numbers $\mathbb{C}$. However, despite the necessity to perform operations in an algebraic extension, this is still one of the most efficient method for polynomial operations in $\mathbb{R}[X]$.

Some number fields however, like for example the ring of integers modulo a prime integer $q$ with $(q-1) \mod n = 0$, naturally contain roots of unity. The NTT algorithm follows the same principle as the FFT except for the fact that roots of unity are natural elements of the underlying ring, as opposed to elements of an algebraic extension. The following algorithms illustrate the NTT and inverse-NTT transforms for a polynomial $R$ of degree $n$ with coefficients modulo $q$. With $w$ a set of $n^{th}$ roots of unity:

```
Algorithm 9: NTT
  input  : $R, w, q, n$
  output: $X$
  $X = init\_vector(q, n)$
  for $i$ in range($n$) do
      for $j$ in range($n$) do
          $X[i] = X[i] + R[j] * w^{i*j}$
      end
  end
```

```
Algorithm 10: invNTT
  input  : $r, w, q, n$
  output: $X$
  $X = init\_vector(q, n)$
  for $i$ in range($n$) do
      for $j$ in range($n$) do
          $X[i] = X[i] + r[j] * w^{-i*j}$
      end
      $X[i] = X[i] * n^{-1}$
  end
```

In chapter 3, we will present another version of the algorithm, using the Cooley-Tukey butterfly algorithm [CT65], providing a complexity in $O(nlogn)$, and using a negative wrapped convolution technique [LMPR08] to keep a polynomial degree of $n$ throughout our computations in NTT form.

## 1.5 Montgomery's Multiplication Algorithm

Montgomery's multiplication algorithm is a method of performing a modular multiplication modulo an integer $N$ while only performing operations modulo another integer $M$. The goal of this algorithm is to gain an advantage in complexity thanks to properties specific to operations in the ring $\mathbb{Z}/M\mathbb{Z}$. For example, this could be used to reduce integers in $\mathbb{Z}/N\mathbb{Z}$ while only using operations modulo a power of 2. Because computers store and manage data as binary fields, computing modulo a power of 2 offers a lot of advantages regarding performances.

Let us consider the problem of reducing $AB \mod N$. Let us define the value $Q$ such that $AB + Q * N \mod M = 0$. We can easily compute a value $Q$ verifying this equation in $\mathbb{Z}/M\mathbb{Z}$ by computing:

$$Q = -ABN^{-1} \mod M$$

Let $M^{-1}$ be the inverse of $M$ in $\mathbb{Z}/N\mathbb{Z}$. We can then compute the value $R$ such that:

$$R = (AB + Q * N) * M^{-1}$$

From [BLSK98], we get a proof that $R < 2N$. Moreover, $R$ satisfies the following equation:

$$R = ABM^{-1} \mod N$$

Therefore, the value of $R$ computed modulo $M$ is either the result of $ABM^{-1}$ modulo $N$ or the value of $ABM^{-1} + N$. in the second case, a simple subtraction of $N$ is enough to reduce the result modulo $N$.

However, what we seek is the result of $AB \mod N$. To obtain the correct result, the input of our algorithm needs to be in what is referred to in the literature as "Montgomery form". Let $\tilde{A} = A * M \mod N$, respectively $\tilde{B} = B * M \mod N$, be the Montgomery form of $A$, respectively $B$. By using Montgomery's multiplication algorithm on inputs $\tilde{A}$ and $\tilde{B}$, one obtain $\tilde{C} = ABM \mod N$. Another call to Montgomery's algorithm with inputs $\tilde{C}$ and 1 will simply yield the result $C = AB \mod N$. The question is then: how does this help us compute a modular multiplication if a modular multiplication must be performed on each input beforehand ? The advantages is two folds:

- in the case of an algorithm performing numerous modular multiplications involving the same variables (exponentiation for example), computing the Montgomery form of said variables only require one modular multiplication per variables, regardless of the number of products involving them,
- while the values $A$ and $B$ are variables that lies in $\mathbb{Z}/N\mathbb{Z}$, the value $M$ is static. Pre-computing all the values of $X * Y \mod N$ would cost at least $N^2$ modular products. However, one could consider the cost of pre-computing all values of $\tilde{X} = X * M \mod N$ which would only cost $N$ modular products.

The following algorithm performs Montgomery's modular multiplication:

**Algorithm 11:** Montgomery

```
input : $\tilde{A}, \tilde{B}, N, M$
output: $R$
$Ninv = N^{-1} \mod M$
$Minv = M^{-1} \mod N$
$Q = -\tilde{A}\tilde{B}Ninv \mod M$
$R = (\tilde{A}\tilde{B} + QN) * Minv \mod M$
```

Instead of $M$ being a power of 2, this algorithm can also be performed with $M$ as a product of small co-prime numbers. This allows the use of Montgomery's modular multiplication coupled with RNS decomposition. This will be used in chapter 3 to allow for a fully table based implementation of Montgomery's modular multiplication algorithm. This method is also described in the following article [BLSK98]

## 1.6 Working Modulo $2^n$

Information security is a complex research field. Whether considering the case of cryptography or obfuscation, both fields mix mathematical algebraic knowledge with computer science. This can be a challenge when one tries to apply algebraic tools commonly used in mathematics as part of a program running on a computer. In particular, a computer benefit greatly from operations being performed on integers modulo a power of 2, as numbers are stored and manipulated as binary values within registers. This is important for applications for which performance is a key concern. On the other hand, many algebraic techniques perform operations involving elements of some kind of finite field. A commonly used middle ground is the case of Galois's Field of size $2^n$ ($GF(2^n)$), an algebraic extension of $GF(2)$.

However, some research fields (obfuscation in particular), benefit from having access to a variety of mathematical objects to choose from while still being concerned with performance issues. Another set for which elements are easily represented in computer registers is the ring of integers modulo a power of 2. However, this ring is not a finite field, which may cause problems when trying to adapt classical algebraic techniques to its elements. To illustrate this, we will discuss the case of a classical algorithm: Lagrangian interpolation. We will discuss the problems encountered when trying to use this algorithm as it is depicted in the literature to elements of the ring $\mathbb{Z}/2^n\mathbb{Z}$. This algorithm will be adapted to solve a specific problem involving elements of this ring in chapter 4.

### 1.6.1 Lagrangian Interpolation

Lagrangian interpolation is an algorithm that, from a set of points, seeks to represent a function matching that set. In other words, from a set of points $((x_0, y_0), (x_1, y_1), \dots, (x_k, y_k))$, find a polynomial function $f$ verifying:

$$f(x_i) = y_i \quad \forall i$$

Classical Lagrangian interpolation algorithm finds $f$ by performing the following operation:

$$f(X) = \sum_{j=0}^{k} \Big( \prod_{0<i<k, i\neq j} \frac{X - x_i}{x_j - x_i} \Big)$$

Whether the set of points $((x_0, y_0), (x_1, y_1), \dots, (x_k, y_k))$ lies in a finite field or in the set of real numbers $\mathbb{R}$, Lagrangian interpolation is a classical algorithm for computing a value of $f$. However, let us consider the case of the ring $\mathbb{Z}/2^n\mathbb{Z}$. To compute lagrangian interpolation, we need to compute the following value:

$$\Big( \prod_{0<i<k, i\neq j} x_j - x_i \Big)^{-1}$$

Unfortunately, as soon as the degree of our interpolation exceeds 2, this value will always end up even. As the set of even integers in $\mathbb{Z}/2^n\mathbb{Z}$ is the set of non-invertible elements, this computation cannot be performed as it is. In chapter 4, we will give a method to perform a Lagrangian interpolation that avoid this impossible operation.

This is a classic case of a well known algorithm that cannot be performed on the ring $\mathbb{Z}/2^n\mathbb{Z}$ due to its unique restrictive structure (half of the elements being non-invertible elements). To solve the interpolation problem for a practical application, we need to proceed to one of the following:

- give up on the advantages from computing with integers in the ring $\mathbb{Z}/2^n\mathbb{Z}$ and choose a finite field instead,
- find a way to add more invertible elements by computing our interpolation in an algebraic extension [BERR16], which is very costly and defeat the purpose of working with elements that lie in $\mathbb{Z}/2^n\mathbb{Z}$,
- adapt Lagrangian interpolation to avoid our problematic operation [BKRS18],
- move to a different, less classical, algorithm to solve the interpolation problem.

In chapter 4, we will be confronted to a problem that could be solved with an interpolation. We discuss an adaptation to Lagrangian interpolation to the ring $\mathbb{Z}/2^n\mathbb{Z}$ and an alternative method, using an algorithm more practical when computing in $\mathbb{Z}/2^n\mathbb{Z}$ in Newton's inversion method, as another way to solve our problem.

# Chapter 2

# Symmetric White-Box Cryptography

In 2003, Chow, Eisen, Johnson and C. Van Oorschot published a design [CEJO03] for an AES based Symmetric White-Box. This design is still used today as a starting point for many AES white-box designs. The main idea is to replace all computations involving the secret key (whether in the case of encryption or in the case of decryption) by a set of lookup tables performing said computation, hiding the key in a vast amount of data. The problem then becomes protecting those lookup tables from key-extraction. That is to say modifying the lookup tables so that, given their input/output, it is impossible to decide which key was used to create them. While this design is flawed and vulnerable to many side-channel attacks, it is still the perfect starting point to consider how modern white-box constructs are built and why they struggle to resist motivated white-box adversaries.

This chapter will discuss the initial design proposed in [CEJO03] in order to explain why AES is still the main candidate for state of the art white-box cryptography while it is *by design* very weak to white-box attacks. Then we will discuss what properties makes it so vulnerable. Understanding what makes AES both great and bad for white-box cryptography is key to propose a legitimate alternative. While our work on that subject focused on a proposal based on an existing cryptosystem other than AES as an alternative for white-box cryptography, those same properties could help build a new cryptosystem tailored for white-box cryptography.

## 2.1 The Initial Construct From Chow, Eisen, Johnson and C. Van Oorschot

In this section, we will discuss the basic techniques involved with the AES white-box proposal presented in [CEJO03], for a general overview of the AES cryptosystem, the reader can refer to section 1.1 of the toolbox chapter.

### 2.1.1 Transforming AES into a Set of Lookup Tables

As we mentioned in our introductory section 1.1, AES is composed of four basic operations performed in succession over a certain amount of rounds (here 11 rounds for AES-128). In particular, *AddRoundKey* is the one operation involving data derived from the secret key $s_k$ (a.k.a. the round keys). During the round cycle of AES-128 operations, this particular operation is performed once for each byte of the current state. It involves each time a different secret byte, depending on which state byte is involved and the current round. All secret bytes are derived from the secret key and form together the round keys of AES-128. The recollection of a complete set of those secret key bytes, that is to say all secret bytes involved in a specific round, allows for the recovery of the full secret key. It is therefore the asset we need to protect in a white-box context.

Let us consider a secret byte $k_i^j$ involved within the *AddRoundKey* operation performed on the $i^{th}$ round and $j^{th}$ byte of the current state. We need to perform the following operation without leaking the secret byte $k_i^j$ to a white-box attacker: $a \oplus k_i^j$, where $a$ is a variable byte representing the current state byte. Let us consider $T_{k_i^j}$, a lookup table built specifically for that operation:

$$T_{k_i^j}[a] = a \oplus k_i^j \quad \forall a \in \mathbf{Z}/2^8\mathbf{Z}$$

By replacing all instances of the *AddRoundKey* operation by such a lookup table, it is ensured that secret bytes $k_i^j$ will not clearly appear in the code during encryption/decryption.

Unfortunately, because lookup tables $T_{k_i^j}$ only depend on one byte of the secret, there is only 128 different tables. While this offers some protection in a grey-box setting, an opponent in a white-box setting would have no problem retrieving the secret key of such a design. Because an attacker in a white-box setting has full access to any data used during decryption, it is very easy for such an attacker to reconstruct those lookup tables. Then, this same attacker can recover the secret key by brute-forcing all possible tables for each possible secret key byte before comparing those tables with the ones present in the code. Therefore, additional countermeasures are required to prevent key-extraction from those lookup tables. In [CEJO03], encodings are proposed as such a countermeasure. However, in order to make better use of those encodings, the AES scheme needs to be further modified first.

Let us recall AES 4 operations cycle :

- *AddRoundKey*
- *SubByte*
- *ShiftRows*
- *MixColumn*

Consider that both the *AddRoundKey* and *SubByte* operations only take one of the state byte as input. Therefore, by performing the *ShiftRows* operation on each round key, we can change the order of operations without changing their outcome :

- *ShiftRows*
- *AddRoundKey*
- *SubByte*
- *MixColumn*

Then, consider the lookup tables $T_{k_i^j}$, used in place of the *AddRoundKey* operations, and the substitution boxes, used in the *SubByte* operations. Both substitution tables take inputs of the same size. Therefore we can easily fuse those tables into one lookup table:

$$T'_{k_i^j}[a] = SubByte(a \oplus k_i^j) \quad \forall a \in \mathbf{Z}/2^8\mathbf{Z}$$

Now, consider the *MixColumn* operation. As explained in [CEJO03], for any four input bytes $x_0$, $x_1$, $x_2$ and $x_3$, this operation can be rewritten as the following equation:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = x_0 \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus x_1 \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus x_2 \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus x_3 \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix}$$

This equation can be separated into two components:

- ⋆ a product of each state byte by a constant vector depending solely on the current number of round and state byte, resulting in a 4 bytes output for each byte state (expansion),
- ⋆ a xor operation combining a 4 bytes set into one output byte (reduction).

As with the *SubByte* operations, we can include the first part of this computation (expansion) within our lookup table. For example, for the first byte of the current state:

$$TTy_{k_i^1}[a] = T'_{k_i^1}[a] \cdot \begin{bmatrix} 02 & 01 & 01 & 03 \end{bmatrix}^T$$

$$TTy_{k_i^1}[a] = [T'_{k_i^1}[a] \cdot 02, T_{k_i^1}[a], T_{k_i^1}[a], T_{k_i^1}[a] \cdot 03] \quad \forall a \in \mathbf{Z}/2^8\mathbf{Z}$$

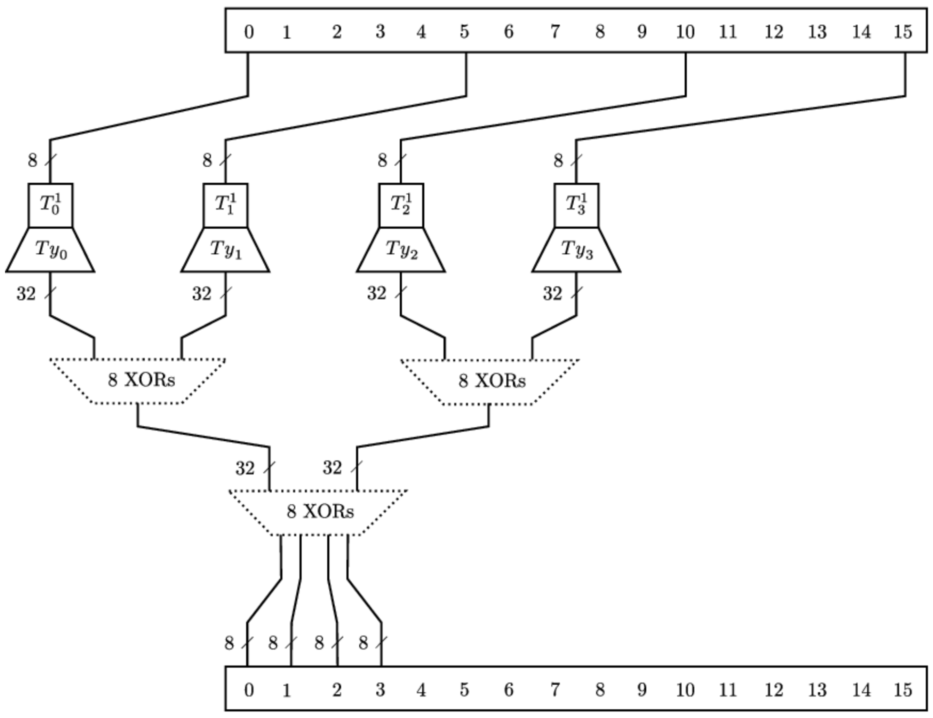
Note that the result of such a lookup table no longer is one byte of data, but a tuple of 4 bytes. Also note that on the last round of AES-128, the *MixColumn* operation will not be performed. Therefore, the last round of the proposed design will simply hold a lookup table performing the *AddRoundKey* and *SubByte* operations. We will refer to those tables performing the *AddRoundKey*, *SubByte* and the first step of the *MixColumn* operation as *TTy* lookup tables while lookup tables solely performing the *AddRoundKey* and *SubByte* operations will be referred as *T* lookup tables.

Finally, to finish our *MixColumn* operation, a set of lookup tables (referenced as *xor-tables* in [CEJO03]) perform the second part of this computation (reduction). Each of those lookup tables will take as input two of the output bytes of two distinct *TTy* lookup tables and output their xored value. To reduce the memory cost overhead, the design proposed in [CEJO03] uses a collection of 8 to 4-bit *xor-tables*. For simplicity, let us simply consider 16 to 8-bit lookup tables. For any two bytes $x$ and $y$ as inputs:

$$XOR_{i,j}[x][y] = x \oplus y$$

In order to perform all xor operations required within one round of AES-128 to finish the *MixColumn* operation, we will need 48 such lookup tables for each round of AES (i.e. 432 *xor-tables* in total). Note that each xor operation will be replaced with its *own* lookup table. This is important to support our next layer of protection : encodings.

As a result, we obtain the following "network" of lookup tables:



### 2.1.2 Protecting the Lookup Tables with Parasitic Encodings

To prevent an attacker in a white-box setting from extracting a secret key from our set of lookup tables, the article [CEJO03] proposes the use of non-linear encodings applied on each input/output of the lookup tables. The idea is that each output encoding applied to a specific lookup table will met its counterpart (i.e. an inverse encoding) applied to the input of the next lookup table. That way, we can ensure the entire design still produces correct encryption/decryption of our state.

Let $f$ and $g$ be non-linear bijective functions over a ring of $R$ $2^8$ elements (note that this could be any ring, not only $GF(2^8)$). let $f^{-1}$ and $g^{-1}$ be their respective inverse bijective function so that:

$$f^{-1}(f(x)) = x \quad \forall x \in R$$

Let $T_0$, $T_1$ and $T_2$ be a set of successive lookup tables, meaning for any input $x$, we expect our design to compute the value $y$ such that:

$$y = T_2(T_1(T_0(x)))$$

Let $f$ be an encoding applied to the output of lookup table $T_0$ and $f^{-1}$ be its inverse encoding applied to the input of lookup table $T_1$. Respectively, let $g$ be applied to the output of lookup table $T_1$ and $g^{-1}$ be applied to the input of lookup table $T_2$. The resulting lookup tables being:

$$T_0'(x) = f(T_0(x))$$
$$T_1'(x) = g(T_1(f^{-1}(x)))$$
$$T_2'(x) = T_2(g^{-1}(x))$$

It holds that:

$$T_2'(T_1'(T_0'(x))) = T_2(T_1(T_0(x))) = y$$

However, the content of lookup tables $T_0$, $T_1$ and $T_2$ are now obfuscated by encodings $f$ and $g$.

Such input/output encodings are applied to all lookup tables of the desgin proposed in [CEJO03], with the exception of the following lookup tables:

- ⋆ the set of TTy lookup tables of the first round of AES do not hold an input encoding (as no computation precedes it),
- ⋆ the set of $T$ lookup tables of the last round of AES do not hold an output encoding (as the result yield the final state of decryption).

To an attacker in a white-box setting, there is no longer 256 possible TTy lookup tables, as those tables now depends on both a secret key byte and input/output encoding functions. Therefore, as long as separating the encoding function and the basic AES operation from the lookup table is hard, extracting the secret key byte embedded in a specific lookup table should be hard, even in a white-box setting.

However, this is not the case for all lookup tables of our current design. In particular, the first and last set are each missing part of our parasitic encodings, respectively an input encoding for our first set of lookup tables and an output encoding for our last set of lookup tables. Moreover, while the *SubByte* operation of AES-128 is a non-linear transformation, all other operations performed in a round of AES is a linear transform of its state bytes. This discrepancy between linear and non-linear transform can yield to efficient methods to separate the encoding functions from the main AES operation computed within a specific lookup table. In the article [CEJO03], the author suggests the use of both external and linear encodings to strengthen the design against motivated attacker in a white-box setting.

### 2.1.3 Strengthening the Design with Linear and External Encodings

#### Linear Encodings

To increase the resistance of their design against an attacker in the white-box setting, the authors of [CEJO03] define additional countermeasures, including linear encodings (denoted *Mixing Bijections* in article [CEJO03]). The idea is that while parasitic encodings provide confusion as to the content of our lookup tables, they are limited as they need to be removed between tables to ensure our design still produces a correct encryption/decryption of a given input. Whereas linear encodings will have the ability to mix with some of the basic operations performed within our lookup tables before they need to be removed. The application of these encodings to the input/output of our lookup tables should add a measure of diffusion to this design.

We will present a simplified design to give the reader an intuition on how such encodings can be built and used before briefly covering how they are used in the proposal of article [CEJO03].

Let $le$ be such a linear encoding. For simplicity, let us consider a bijective function that simply consist in adding a constant value $\delta$ to our given state byte:

$$le(x) = x \oplus \delta \quad \forall x \in GF(2^8)$$

Let us consider the application of four linear encodings $le_0$, $le_1$, $le_2$ and $le_3$ to the first output byte of four TTy lookup tables so that each output affected will be used together in pair in the following set of xor-tables:

$$\begin{aligned}
TTy'_{k_i^0}[a] &= le_0(TTy_{k_i^0}[a]) = TTy_{k_i^5}[a] \oplus \delta_0 && \forall a \in GF(2^8)\\
TTy'_{k_i^5}[a] &= le_1(TTy_{k_i^5}[a]) = TTy_{k_i^5}[a] \oplus \delta_1 && \forall a \in GF(2^8)\\
TTy'_{k_i^{10}}[a] &= le_2(TTy_{k_i^{10}}[a]) = TTy_{k_i^{10}}[a] \oplus \delta_2 && \forall a \in GF(2^8)\\
TTy'_{k_i^{15}}[a] &= le_3(TTy_{k_i^{15}}[a]) = TTy_{k_i^{15}}[a] \oplus \delta_3 && \forall a \in GF(2^8)
\end{aligned}$$

By following the network of *xor-tables* involved in the second part of the round, it holds that the resulting state byte 0 of round $i+1$ (denoted $sb_{i+1}^0$ in the following equation) will now verify:

$$\bar{sb}_{i+1}^0 = sb_{i+1}^0 \oplus \delta_0 \oplus \delta_1 \oplus \delta_2 \oplus \delta_3$$

This holds because linear encodings $le$ and *xor-tables* both perform a simple xor operation on our byte state. More complex linear operations are possible (for example, affine functions with an identical multiplicative constant for all $le_i$ involved together) are possible as long as linear encodings have specific linear properties with our *xor-tables*. In particular, the following equation must hold:

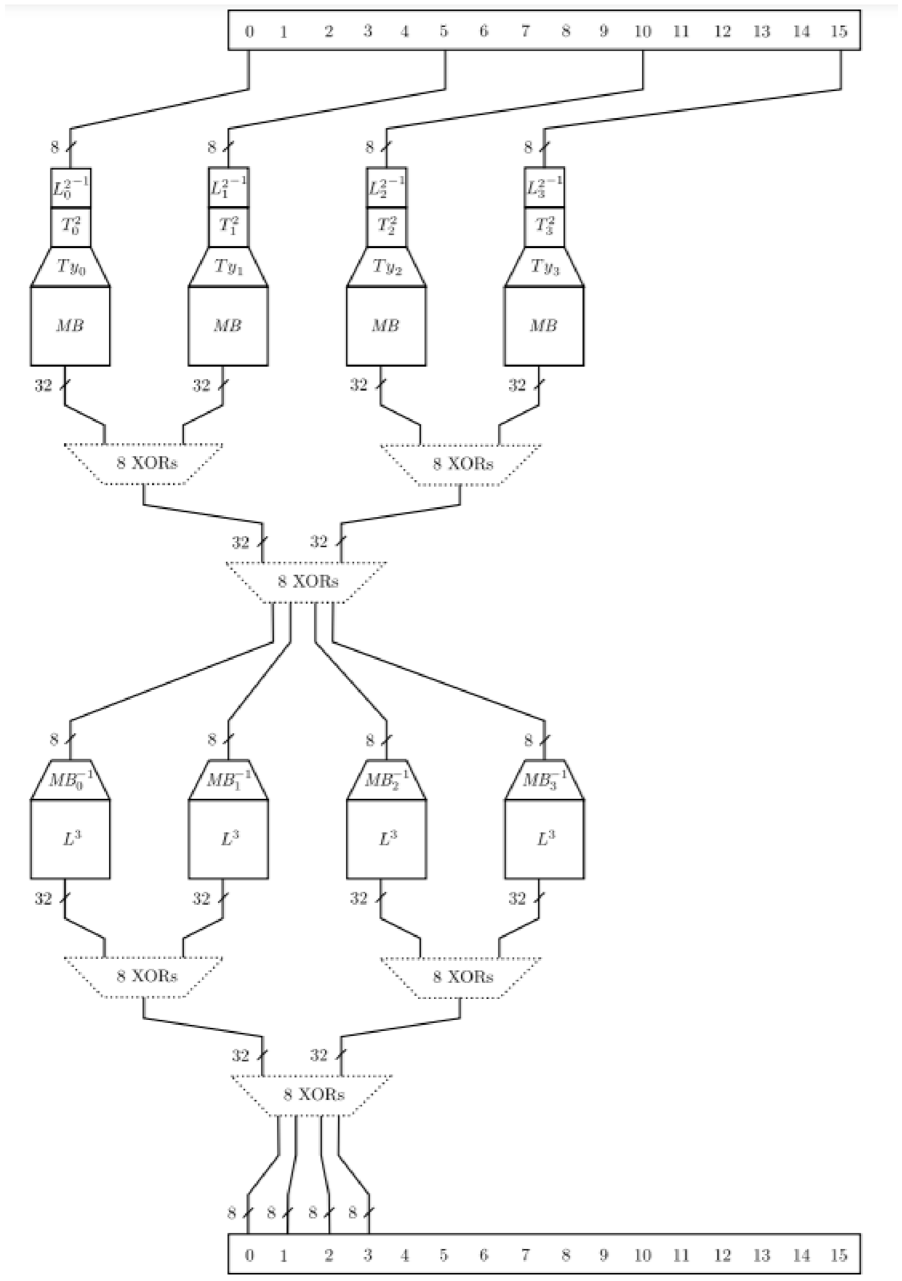
$$le_0(x) \oplus le_1(y) = le_0(le_1(x \oplus y)) = le_1(le_0(x \oplus y))$$

Finally linear encodings needs to be removed before the next non-linear operation performed in an AES round, i.e. as input encoding of the next TTy lookup table (as the *SubByte* operation embedded in those tables is non-linear). Let $le_{0-3}^{-1}$ be the inverse linear encoding of the combination of our four linear encodings $le_0$, $le_1$, $le_2$ and $le_3$:

$$\begin{aligned}
le_{0-3}^{-1}(x) &= x \oplus \delta_0 \oplus \delta_1 \oplus \delta_2 \oplus \delta_3\\
TTy'_{k_{i+1}^0}[x] &= TTy_{k_{i+1}^0}[le_{0-3}^{-1}(x)]\\
&= TTy_{k_{i+1}^0}[x \oplus \delta_0 \oplus \delta_1 \oplus \delta_2 \oplus \delta_3]
\end{aligned}$$

In the design presented in [CEJO03], linear encodings (or *mixing bijections*) applied to the TTy lookup tables are generated in a different manner. Each TTy lookup tables has a 32-bit to 32-bit linear encoding applied to its output. This linear encoding is represented by a $32 * 32$ invertible matrix over $GF(2)$. Then, additional lookup tables are generated, those new lookup tables perform the removal of the 32-bit linear encoding and apply a new linear encoding composed of 4 8-bit to 8-bit linear encodings, resulting in a similar 32-bit output. Each 8-bit linear encoding has its inverse encoding applied to the input of the next *TTy* lookup table to the matching state byte (with respect to the ShiftRow operation). This allows for more complex linear encodings as output of TTy lookup table that allow for a better diffusion of linear encodings.

The final design is represented by the following schematic :



### External Encodings

In article [CEJO03], the author discuss the use of external encodings as an additional layer of defense against an attacker in a white-box context. Unlike other types of encodings, external encodings are applied to data before (in the case of input external encodings) or after (in the case of output external encodings) they are processed by the white-box. This means the white-box now manages *encoded* inputs and outputs. The idea is to have the first range of lookup tables remove the input external encoding, and the last range of lookup tables apply the output external encoding. The encoded output is then decoded within a *secure* part of the software (through obfuscation or hardware protection for example).

Such a protection does not increase the size of the design. However, the security provided by output external encodings assume the ability to perform the decoding operations out of the reach of the white-box attacker. Moreover, input external encodings need to be applied to the data by a trusted party, as knowledge of those external encodings would nullify their added security. In practice, external encodings cannot always be used, and the security they provide depends a lot on the context they are used in.

## 2.2 Limitations of the Initial Construct

Unfortunately, this initial construct is flawed and cannot resist a motivated attacker in a white-box setting. Since the original proposal presented in [CEJO03], a lot of attacks were designed to break similar constructs in a white-box setting. What followed was a frantic game of cat and mouse between new designs made to resist the latest attack on white-box AES proposals and new attacks breaking those designs. A few examples of designs/attacks can be found in the following literature:

- for designs, articles [BCD06], [XL09] and [Kar11] are three examples of AES white-box proposals,
- for attacks, the articles [BGEC05], [BHMT16] and [LR13] proposes various attacks, each breaking at least one of the aforementioned proposals.

While some attacks are originally derived from grey-box model attacks (side-channel attacks) and adapted to the white-box model (fault attacks, Differential Computational Attacks or DCA, ...) as illustrated by article [BHMT16], some attacks find new ways to exploit the white-box setting to extract a secret key from a white-box design, like the BGE attack presented in [BGEC05] and [LR13].

This section will discuss some basic attacks made against AES white-box designs. Those attacks not only easily break the initial construct proposed in [CEJO03], they can be used to break many state of the art designs. In this section we will discuss those attacks and, in particular, their relation with the AES cryptosystem itself. While AES has some interesting properties that favorises a table-based implementation, which is at the center of most state of the art white-box techniques, its relation with side-channel attacks and the lack of security gained from available encoding techniques causes some serious security concern. These facts usually makes obfuscation the last line of defense for many state of the art designs. While obfuscation is a strong tool in a white-box setting, it unfortunately does not fix some inherent problems tied to the underlying design. Obfuscated designs are still vulnerable to attackers in a white-box models as illustrated by the result of both editions of the WhibOx challenges [CHE17] and [CHE19], aswell as the following article [GPRW19].

### 2.2.1 Differential Computational Attack

A well known side-channel attack on AES designs is the Differential Power Analysis (DPA) [KJJ99]. At a hardware level, a device performing a task will always consume some amount of power and produce electromagnetic (EM) radiation as charges are applied or removed from transistor gates. As different tasks will utilise hardware resources in different ways, power consumption and EM radiation intensity will vary. That variation can be significant enough to yield information on the computation being performed by the device. In particular, a correlation can be made between the secret key of a classic AES implementation and its power consumption/EM radiation.

In 2015, the article [BHMT16] proposes an adaptation of this technique to the white-box setting, the Differential Computational Attack or DCA. The main idea is to expose a correlation between the value of data stored in memory (by using measures like the hamming weight or the access pattern of said data), a.k.a. the lookup tables, and the secret key bytes used in a specific implementation of a white-box design. Because an attacker in a white-box model has full access to the content of both the RAM and the memory of the device hosting the white-box design, and because it has this level of access whether the white-box is running or not, gathering data to expose this correlation (which would be akin to gathering traces for a side-channel attack) is very easy to do. The only thing standing in the way of the attacker becomes its understanding of how the data is stored, which can be tampered with through obfuscation, and its knowledge of how to exploit this information to mount an attack, which is detailed in article [BHMT16].

### 2.2.2 Fault Attacks

Another well known side-channel attack on AES implementations is the fault attack or Differential Fault Analysis (DFA). Let us consider an implementation of AES performing the decryption of a specific cyphertext with regards to a specific secret key. The idea of this attack is that if an attacker is able to change the result of a state byte used in the ante-penultimate round of an AES implementation (for example by switching one of its bits through the use of a laser), the results obtained with and without the fault can be used together to gain information on some of the key bytes used by the implementation.

Specifically, a fault on one of the state bytes right before its use in the last round will yield a fault on exactly four bytes of the final result. By representing this last round as a system of equations involving unknown intermediate state bytes from the previous round and secret key bytes from the last two rounds, it is possible to combine both systems (the one with a fault and the one without it) to exhibit a constraint on the involved key bytes in the form of an equation. Once this operation is repeated for different faults on different state bytes, the number of constraints grows until only one set of secret key bytes for the last round of AES is possible. Once this is recovered, reversing the algorithm generating round keys from the original 16 byte key yield said key to the attacker. For further information on fault attacks on classical AES designs, article [DLV03] explain this attack in detail.

In a white-box setting, this attack is even more devastating as the attacker has full access to the content of any register being used by the device the cryptographic function is being computed on. Where an attacker in a grey box setting needs to carefully plan fault injections by carefully planning a bit-flip on the right register at the exact moment it stores a state byte of the ante-penultimate round of AES *and* manage to reproduce that operation on multiple occasions, an attacker in a white-box setting needs only to modify the lookup table corresponding to a specific key byte in the last round of AES. Because the attacker has full access to all static values that make up the lookup tables used by the application, changing one of those values is extremely easy to do in comparison to the grey-box approach.

The following article [AMR20] discuss how to mount such an attack against an AES white-box proposal. In addition, the following blog-posts [TH16] and [Her20] also discusses this attack in a more practical approach.

Both this attack and the DCA attack exposes one of the main drawbacks of using AES as a starting point for white-box designs. We know for a fact that AES is *by design* vulnerable to side-channel attacks. We also know that these attacks are easier to perform and extremely potent in a white-box setting. While our proposal presented in chapter (cite chapter 3) may have its own vulnerabilities to side-channel attacks that are yet to be revealed, focusing all public research on white-box proposals on a cryptosystem plagued with such vulnerabilities in a grey-box model does not seem to be working out.

### 2.2.3 BGE Attacks

The next attack we will discuss was introduced in the article [BGEC05]. The idea behind this attack is that while one lookup table alone may not yield much information on a secret key byte of the underlying design, the relationship between those tables may be used to exhibit constraints on the secret key bytes and encodings embedded in the various lookup tables that constitute the design. Later on, this technique has been made more efficient in the following article [LR13].

In particular, the BGE attack exposes that the combination of both linear and non-linear encodings can be rewritten as a series of composed affine functions. By representing the transformations occurring to a state of an AES white-box design within a round of operations as a mathematical system of equations, an attacker may attempt to solve this system by removing the affine part of the equation and brute-forcing the remaining unknown variables. Once the system is solved, the key can be extracted from the white-box design. The theoretical complexity of the BGE attack is considered to be $2^{30}$ operations (for AES-128) as discussed in [BGEC05], while a more practical analysis exposes a complexity of $2^{22}$ operations for a similar white box[LR13].

### 2.2.4 On the Identification of AES White-Box

Another concern for white-box AES designs is the ability for a motivated attacker to identify the different steps performed by the AES algorithm. AES, with its 9 rounds of computation and 16 byte states, is by design extremely recognisable when used within a software. This can be seen as a security concerned for white-box implementation. Indeed, many attacks on white-box AES designs revolve around the ability to target a specific round or a specific intermediate state byte and being able to read or modify any value being used in that moment.

In response, many designs from recent WhibOx challenges (which confront white-box designs provided by the community against attackers) tried to impede classic white-box attacks by adding layers of obfuscation meant to hide the overall AES sequence of operations. In the example of DFA, an attacker unable to correctly identify a register containing a state byte of the ante-penultimate round would be unable to mount his attack.

Unfortunately, obfuscation is yet to be proven perfect against a reverse engineering process that possess both enough time and resources. This is shown by [GPRW19] which details the reverse engineering process that preceded their break of a state of the art white-box design.

## 2.3 Conclusion on Symmetric White-Box

In this section we presented the initial design for an AES white-box implementation presented in [CEJO03]. We also presented many attacks that can be used to break this initial design ([BHMT16], [AMR20], [BGEC05]), attacks that can be used to break state of the art white-box designs as well ( [BCD06], [XL09], [Kar11], or proposals from [CHE17] and [CHE19]). We can separate those attacks in two categories:

- attacks that are derived from classical side-channel attacks like DPA and DFA,
- attacks that rely on the unique structure of our white-box design, that is to say successively encoded lookup tables, like the BGE attack.

Sadly, the only response we have found to those very efficient attacks is to add extra layers of obfuscation to delay the attack. Unfortunately, a motivated attacker will eventually find a way to move past this layer of obfuscation and proceed to break those white-box designs [GPRW19].

Therefore, instead of proposing yet another AES white-box design, this thesis aims to propose something different: what if we decided to part ways with AES and propose a new design based on some other cryptographic primitive ? Indeed, not all cryptosystems are as vulnerable to side-channel attacks as AES, and different cryptosystems may yield interesting properties we could use for encoding techniques, like homomorphic properties or a different set of linear operations. While AES is accepted as a standard by the industry and is considered secure in a black-box setting (even with regards to a potential quantum computer [Rob17]), AES may not be the perfect candidate for a white-box setting.

The next chapter 3 will propose a white-box design derived from a lattice-based cryptosystem. While research on side-channel attacks on lattice-based cryptographic schemes is still quite recent [PPM17] [BDK+20], proposed state of the art countermeasures seems to hold in a grey-box setting. In addition, this design make use of small homomorphic properties specific to lattice-based cryptography to propose a new type of encodings: homomorphic encodings. Unlike regular encodings, those encodings are applied to encrypted messages while decoding only occurs on their decrypted form. This helps break the classical structure of encodings in white-box designs that consist of applying encodings to a set of lookup tables before decoding in the next lookup table.

# Chapter 3

# Asymmetric White-Box Cryptography

## 3.1 Asymmetric White-Box Cryptography

### 3.1.1 Asymmetric White-Box in the Literature

Unlike its symmetric counterpart, asymmetric white-box is almost absent from public papers in white-box cryptography. While there is a lot of work on protecting asymmetric designs from a grey-box context (for example, [PPM17] and [BDK$^+$20] regarding lattice-based cryptography) and a lot of research on obfuscating asymmetric schemes, which can be used to delay attacks in a white-box setting, straight forward papers on "asymmetric white-box designs" are almost inexistent.

However, this does not mean asymmetric white-box is not a studied topic. Indeed, the presence of various patents on the subject indicates that asymmetric white-box are, if not used, at least researched by the industry. For example, patents [MG09], [AAMSD18] and [HM16] all discuss possible means to implement a white-box version of the RSA cryptosystem. While [MG09] and [HM16] give very little information on how this is done (patent [MG09] does not even name RSA, but mentions a "white-box implementation" performing an exponentiation), patent [AAMSD18] discuss the use of homomorphic properties of the RSA cryptosystem to help protect against white-box attacks. In truth, asymmetric white-box designs can be found in the wild. Unfortunately they are often embedded in heavily obfuscated proprietary code, which can make their study somewhat impractical. Because of the absence of a well-researched basic reference design, such as the AES white-box design of [CEJO03], designing an asymmetric white-box design often requires one to start from scratch. This makes the resistance of those designs vary. A team of developers well versed in obfuscation and/or white-box cryptography may, with enough time, produce an interesting design. However, a less fortunate set up may lead to a design vulnerable to attacks from the white-box model.

### 3.1.2 The Use For Asymmetric White-Box Cryptography

As far as the use for asymmetric white-box is concerned, because research on asymmetric white-box is scarce and its usage rarely documented, determining a classic use case for it is challenging. This is exacerbated by the fact that implementing a symmetric cryptographic scheme to resist the white-box attack model creates an asymmetry. This creates confusion as to the relation between symmetric white-box cryptography and regular asymmetric cryptography, what they protect and when to use them.

Let us consider the fact that turning a symmetric scheme into a white-box creates an asymmetry. Indeed, encodings applied to a white-box performing the encryption operation and encodings applied to one performing decryption are different. Then, the security of a white-box design revolves around the assumption that those encodings are unknown. Meaning, in a white-box setting, we can produce two different circuits, one performing encryption while the other performs decryption. With the assumption that the underlying secret key cannot be extracted from those circuits, knowledge of one circuit does not give any meaningful information to build its counterpart, resulting in an asymmetry. From this observation of a parallel between symmetric white-box and asymmetric cryptography, we raise the following questions (which we will answer one at a time):

- If the goal of white-boxing a symmetric scheme is to get an asymmetry, why is this not equivalent to use an asymmetric scheme? Could an asymmetric scheme solve the problem of using a symmetric scheme in a white-box context ?
- If we already have a means to have a symmetric white-box, which means an asymmetry, why is it necessary to white-box an asymmetric scheme? Why not replace the asymmetric scheme by a white-box symmetric scheme, which creates an asymmetry ?
- What is the use for the attacker to find the key of a cryptosystem if he can already use the device and perform the cryptographic operation?

**If the goal of white-boxing a symmetric scheme is to get an asymmetry, why is this not equivalent to use an asymmetric scheme? Could an asymmetric scheme solve the problem of using a symmetric scheme in a white-box context ?**

In the case of a symmetric scheme designed to resist the white-box model, the goal is to prevent the extraction of the secret key from the software by an attacker in a white-box model. Moreover, the goal is to prevent this attacker from using this key to perform an unsupervised cryptographic function. For example, in the case of AES, extracting the cryptographic key from a symmetric white-box performing decryption allows the attacker to perform the inverse operation, meaning the encryption of data relative to a legitimate secret key.

In the case of an asymmetric scheme performing, for example, the decryption of data, hiding the cryptographic secret key in a white-box context does not prevent an attacker from performing the inverse operation. Indeed, this operation is bound to the public key. This key is supposedly known to anyone, including the attacker.

While symmetric white-box present some asymmetry in their design, this example shows their usage is not equivalent to that of an asymmetric scheme. Symmetric white-box schemes use their asymmetry to ensure that possession of the white-box software does not allow the computation of cryptographic operations outside the software's scope, even from an attacker with the same privileges as a legitimate user. Asymmetric cryptography uses its asymmetry to ensure only the owner of a private key may perform all cryptographic operations relative to its pair of cryptographic keys.

**If we already have a means to have a symmetric white-box, which means an asymmetry, why is it necessary to white-box an asymmetric scheme? Why not replace the asymmetric scheme by a white-box symmetric scheme, which creates an asymmetry?**

In theory, we could imagine the replacement of asymmetric schemes with symmetric white-box schemes. While the asymmetry present in asymmetric schemes does not address some of the security concerns symmetric white-box schemes try to address, the asymmetry present in symmetric white-box schemes could address the security concerns usually answered by asymmetric cryptography.

Consider two symmetric white-box, a white-box performing encryption and a white-box performing decryption. Let us assume that those white-boxes are resistant to an attacker in a white-box model. This means possession of the white-box performing encryption does not allow a user to perform the decryption operation, and vice-versa. Therefore, in theory, we could simulate an asymmetric scheme by using one white-box to perform operations that usually involve the public-key, and its white-box counterpart performing operations associated to the secret key.

In the case of encryption, we could use such a set up in place of an asymmetric scheme, which would yield some protection in a white-box context (considering the symmetric white-box is protected against white-box attacks). However, practical use of symmetric encryption white-box schemes in place of a regular asymmetric encryption scheme is impeded by two problems:

- state of the art symmetric encryption white-box schemes are still vulnerable to the white-box attack model,
- a software may need to communicate with respect to certain protocols, imposing the use of a specific asymmetric scheme.

Let us consider the first problem. The current state of symmetric white-box is extremely vulnerable to the white-box model, due to underlying design vulnerabilities to side-channels and algebraic attacks (standards in symmetric cryptography were not built with the white-box model in mind). Switching to a different type of cryptosystem, which was designed with a different attacker model in mind, can make it harder for an attacker in a white-box model to use classical white-box attacks (for example, consider fault attacks that are very specific to AES design). A practical set-up vulnerable to those types of attacks may consider the overhead cost of asymmetric cryptography worth the potential increase in security (as long as a secure design for asymmetric white-box exist, which is the main topic of this chapter).

Let us now consider the second problem. In a practical world, a software may need to communicate sensitive data with parties which are not under the developer's control. While we could argue that an authentication protocol could use the AES cryptosystem to encrypt data specific to one user (or some kind of hash of those data) to compute a signature, then communicate a white-box designed for its specific secret key as a public key to verify said signature, this may not be possible in a practical setting. In a practical setting, a user may need to communicate with a third party that uses a specific protocol using a specific asymmetric scheme for its signature algorithm. In this case, protecting the secret key used by that asymmetric algorithm from a legitimate user (or someone with its level of access) would require an asymmetric white-box scheme specific to this asymmetric scheme (so that the signed output is compatible with the verification algorithm used by the third party).

This second problem is of particular interest as it is the cause for many practical need for asymmetric white-box cryptography. Indeed, a software may be obligated to use a specific asymmetric cryptosystem to accommodate a specific protocol while still needing some protection in a white-box setting.

**What is the use for the attacker to find the key if he can already use the device and get the decryption?**

The misconception here is that a black-box implementation aims to protect the secret key of a cryptosystem to prevent an attacker from performing a decryption or signing operation, while a white-box only aims at protecting the secret key. This can be used, for example, alongside techniques that prevent code-lifting to ensure decryption can only occur on a selected device. This can also be used to prevent an attacker from providing a faster (usually cheaper) version of the same software. However, the end-user *does* have a legitimate access to the result of the cryptographic operation and will always have access to it in the end.

### 3.1.3 Our Contribution on the Topic of Asymmetric White-Box Cryptography

In this chapter, we present a public proposal for an asymmetric white-box design. It is based on a simple lattice-based scheme presented in [GAGL15] and [AMBG+16]. We transformed this design into a fully table-based scheme, using various algorithmic tools like the NTT transform and the RNS decomposition, which will be detailed in section 3.3. Those lookup tables are then protected by various encoding techniques which make use of the latent homomorphic properties of our base scheme. At this point of our construction, we propose the use of external encodings at this point to protect against algebraic attacks. However, we also discuss how the use of external encodings is extremely restrictive for a practical use of our asymmetric white-box proposal. Finally, we propose a different design, the 3-table based design, to fully protect against a white-box attackers.

Note that lattice-based cryptography is a topic which, while it is being researched by a wide community of researchers for its potential resilience to attacks specific to post-quantum technology, has seen less work regarding its resilience to side-channel attacks compared to AES. However, recent works [OSPG16] [BDK+20] [PPM17] have considered the security of lattice-based scheme in a grey-box model, and proposed countermeasures seems reliable so far. Furthermore, some of those countermeasures present similarities with the white-box techniques proposed later in this chapter.

Our proposal was fully implemented in Python. We detail how we implemented our design in section 3.5.

## 3.2 The Lattice-Based Scheme

This section will introduce the lattice-based scheme from which our proposal originated. We will then adapt this lattice-based scheme to resist an attacker in a white-box model. It is a variant of the BGV [BGV14] cryptosystem influenced by the design presented in both [AMBG+16] and [GAGL15] (in french). Those design were themselves drawn from the cryptosystem presented in [BV11]. In the following sections, we will refer to this cryptosystem as the *BV11-SHE* cryptosystem. This cryptosystem was chosen for its simplicity, while maintaining some key properties of lattice-based cryptography. Some changes were made to the parameters as well for security concerns. In particular, in our implementation of our white-box proposal (presented in section 3.5), we doubled the recommended security level to be more in par with modern lattice-based proposals.

One of the main properties that this scheme presents, that will be later used when transforming it to resist the white-box model, are homomorphic properties. Let $m_1$ and $m_2$ be a couple of plaintexts and $c_1$ and $c_2$ be the corresponding cyphertexts when using an homomorphic encryption scheme. An encryption scheme is said to be homomorphic to an operator $\otimes$ if the following relation holds:

$$Decrypt(c_1 \otimes c_2) = m_1 \otimes m_2$$

While other cryptosystems may present their own level of homomorphic capabilities (the RSA cryptosystem for example has some multiplicative homomorphic properties) and while lattice-based cryptography is, as of today, not a standard of the industry, lattice-based cryptography homomorphic properties are easier to exploit for encoding purposes, making it the easier choice to us for an asymmetric white-box proposal exploiting such properties. In addition, lattice-based cryptography is at the heart of several designs proposed to the NIST contest for the next standard in post-quantum cryptography like saber [DKSRV18] and CRYSTALS-Kyber [BDK+18] .

### 3.2.1 Notations for Lattice-Based cryptography

Our cryptosystem involves elements of the cyclotomic ring:

$$R_q = \frac{(\mathbf{Z}/q\mathbf{Z})[X]}{X^n + 1}$$

with parameters $q$, a big prime integer (15 bits in our case), and $n$ a power of 2.

The choice of parameters $q$ and $n$ set the security level of our scheme. Increasing the size of parameter $q$ allows for increased homomorphic properties, but result in a less convoluted lattice which lowers the difficulty of solving the underlying RLWE problem (as defined in the following section **??**). This requires an increase of parameter $n$ to keep a similar security level. In our case, we set our security parameter $n = 1024$ and chose $q$ to be a prime integer of size $\approx 2^{15}$. This ensure a minimum amount of homomorphic properties while providing a security level that is about twice the one proposed in article [AMBG+16]. The following article [ABBK17] discuss the choice of parameters for more recent lattice-based cryptosystems. While not directly considering our version of a lattice-based encryption scheme, their choice of parameter sizes are close to our own.

Elements of $R_q$ can be seen as polynomials of degree $n-1$, with coefficients modulo $q$, modulo $X^n+1$. The product modulo $R_q$ of two elements $a$ and $s$ of $R_q$ will be denoted $a \cdot s$ while the addition of two elements, denoted $a+s$, will simply consist of a coefficient-wise addition of those elements. The notation $a[i]$ denotes the coefficient of degree $i$ of the element $a$.

For simplicity, a plaintext represented by an element of $R_q$ for which all coefficients are set to 0 will simply be denoted 0. In a similar way, 1 will refer to a plaintext represented by an element of $R_q$ for which all coefficients are set to 0 except for the coefficient of degree 0 set to 1, and $X^i$ will denote an element for which all coefficient are set to 0 except the coefficient of degree $i$ set to 1. Those notations will be used in the section regarding homomorphic encodings.

### 3.2.2 Our Lattice-Based Scheme

This section briefly presents the three main functions of our scheme. Those are once again more detailed in [GAGL15] and [AMBG+16]. However, this scheme is extremely generic and white-box techniques discussed in the rest of this article should be applicable to other cryptosystems like BGV [BGV14] or FV [FV12]. The two functions *GaussianPoly* and *UniformPoly* generate random elements of $R_q$ with, respectively, Gaussian and uniform distribution. the public and secret keys are, respectively, denoted by $pk$ and $sk$. While $\alpha$ and $m$ are, respectively, a ciphertext and a plaintext.

**Algorithm 12:** KeyGen

```
input : k == security_level
output: $(pk, sk)$

$sk$ = GaussianPoly($n, q, k$)
$pka$ = UniformPoly($n, q, k$)
$pkb = pka \cdot sk + 2\cdot$ GaussianPoly($n, q, k$)
$pk$ = $(pka, pkb)$
```

**Algorithm 13:** Encrypt

```
input : $pk, m$
output: $\alpha = (\alpha_1, \alpha_2)$

$u$ = GaussianPoly($n, q, k$)
$e_1$ = 2·GaussianPoly($n, q, k$)
$e_2$ = 2·GaussianPoly($n, q, k$)
$\alpha = (pka \cdot u + e_1, pkb \cdot u + e_2 + m)$
```

**Algorithm 14:** Decrypt

```
input : $\alpha = (\alpha_1, \alpha_2), sk$
output: $m$

$m = \alpha_2 - \alpha_1 \cdot sk$
for each coefficient of m: do
    $m[i] = (m[i] < q/2)?\ m[i]\%2\!:\ (1 - m[i])\%2$
end
```

Cyphertexts are pairs of elements of $R_q$, while a plaintext $m$ is a binary element of $R_q$, that is to say a polynomial with coefficients in $\{0, 1\}$. During decryption, the computation:

$$m = \alpha_2 - \alpha_1 \cdot sk$$

involves the secret key $s_k$. It is that computation that we will need to hide in a white-box context as it involves the secret key.

## 3.3 Transforming our Scheme into Lookup Tables

To transform our scheme so that it resists an attacker in a white-box model, we will transform its critical computation $\alpha_2 - \alpha_1 \cdot sk$ into a set of lookup tables. We seek to achieve such a transformation without reducing the values of the parameters of our ring $n$ and $q$ as they are both tied to the overall security of our scheme.

#### NTT

Similar to a Fast Fourier Transform (FFT), the Number Theoretic Transform (NTT) [LN16a] switches from a coefficient representation of polynomials into an evaluated representation in $n^{th}$ roots of unity. Once in such a representation, computations can be performed on each evaluated value value-wise. Let $\omega$ be an $n^{th}$ root of unity in $R_q$, the NTT procedure maps a polynomial $\alpha_1$ in $R_q$ with coefficients $\alpha_1[0] \dots \alpha_1[n-1]$ to a set of n linear combinations of n distinct powers of $\omega$ as follows:

$$\beta_1[k] = \alpha_1[0] \times \omega^{0k} + \alpha_1[1] \times \omega^{1k} + \alpha_1[2] \times \omega^{2k} + \dots + \alpha_1[n-1] \times \omega^{(n-1)k}$$

$$\forall k, 0 \leq k < n$$

Let $\beta_1$, $\beta_2$ and $\gamma$ be, respectively, the NTT representation of $\alpha_1$, $\alpha_2$ and $sk$. We can then perform our key computation value-wise:

$$\delta[k] = \beta_2[k] - \beta_1[k] \times \gamma[k]$$

and finally compute the inverse NTT transformation to recover $m$.

This technique allows us to perform $n$ addition/product of integers in $\mathbb{Z}/q\mathbb{Z}$ instead of one addition/product of polynomials in $R_q$. This is a first step toward a table-based implementation but is not sufficient on its own as $n$ lookup tables the size of $q$, while feasible in theory, would be unusable in practice.

### RNS

Our computation $m = \alpha_2 - \alpha_1 \cdot sk$ is now, with the use of the NTT, a series of products of big integers. Another technique allows for further decomposition of those computations into operations involving smaller integers: Montgomery's multiplication algorithm in a Residue Number System (RNS) [BLSK98].

Let $\beta = \{p_0, p_1, \dots, p_k\}$ be a basis composed of small co-prime numbers $p_i$. Let $p = \prod_{i=0}^{k} p_i$, in a Residue Number System (RNS) any integer in $\mathbb{Z}/p\mathbb{Z}$ is represented in basis $\beta$ (that is to say by its residues modulo each $p_i$). Any number represented in basis $\beta$ can be translated back into an element of $\mathbb{Z}/p\mathbb{Z}$ via the use of the Chinese Remainder Theorem (CRT). In the RNS representation, additions and multiplications can be performed residue-wise, as a reconstruction via CRT will still produce a correct result modulo $p$.

Through the use of NTT, our scheme now performs $n$ subtractions and multiplications in parallel on $\mathbb{Z}/q\mathbb{Z}$, $q$ being a prime number of size at least the size of $n$ (otherwise NTT would not have been possible). To use an RNS to further decompose our computations, we need to switch to another representation in $\mathbb{Z}/p\mathbb{Z}$ where $p = \prod_{i=0}^{k} p_i$. Fortunately, the well-known Montgomery's multiplication algorithm allows us to switch from $\mathbb{Z}/q\mathbb{Z}$ to $\mathbb{Z}/p\mathbb{Z}$ to perform our computations. Once in $\mathbb{Z}/p\mathbb{Z}$ representation, we move to an RNS representation and perform our computations residue-wise (an example of such a combination of RNS and Montgomery's multiplication algorithm can be found in the following article [BLSK98]).

### The Resulting Scheme

After both transformations (NTT and RNS), our scheme has become a series of products involving extremely small integers. For our chosen $q$, $4-bit$ integers are enough to ensure a practical transformation into lookup tables . Let $\alpha_1^{i,j}$, $\alpha_2^{i,j}$ and $sk^{i,j}$ be the $j^{th}$ residue in RNS of the $i^{th}$ evaluation of respectively $\alpha_1$, $\alpha_2$ and $sk$ in NTT/RNS decomposition form. we can now perform:

$$m^{i,j} = \alpha_2^{i,j} - \alpha_1^{i,j} \cdot sk^{i,j} \forall i, j$$

and then use the Chinese Remainder Theorem and the inverse NTT operation to retrieve the element $m$. Those elements are small enough so that those computation can be replaced by a series of lookup tables:

$$m^{i,j} = S_{sk}^{i,j}[\alpha_2^{i,j}][\alpha_1^{i,j}]$$

With both use of NTT and RNS, such lookup tables are possible in practice. For example, with our chosen parameters, all lookup tables weight is no more than 20 MB. Moreover, doubling our security parameter $n$ only doubles the number of lookup tables (and therefore their size).

Note that many lattice-based schemes (BGV [BGV14], FV [FV12]) use RNS *before* the NTT. This use of both techniques allows for faster polynomial products in those schemes [BEHZ17]. However, the minimum size of two integers involved in a product, when performing both techniques in that order, is bound by the size of $n$ (otherwise, NTT would not be possible for the lack of $n^{th}$ roots of unity), which is too much for a practical transformation of our computations in lookup tables. However, both approaches are similar.

It is possible to lower the cost of our lookup tables by chaining RNS decompositions (using Montgomery's algorithm each time to move to another basis composed of smaller primes). However, this comes at a cost of noticeable computation time overhead (as reconstruction requires more and more use of the CRT). Therefore, this technique would only be of interest for embedded software that struggles with memory consumption overhead.

Computations involving our secret key are now fully table-based. However, the same problem present in Chow's AES white-box is emerging. Because those tables only depends on the value of the residues of $sk$, it is possible to brute force those values until all $sk^{i,j}$ are recovered. Therefore, we now need a technique to hide those residues: encodings.

### 3.3.1 Homomorphic Encodings

Unlike the AES white-box proposal in [CEJO03], our scheme does not present itself as a big network of lookup tables, but rather in a set of lookup tables in parallel. Therefore, linear and non-linear encodings do not provide a way to prevent the extraction of $sk$'s residues.

However, unlike AES, our scheme present small homomorphic properties. Those open the way to a new type of encodings: homomorphic encodings. We will not discuss in detail the proofs of those homomorphic properties, as there is already a lot of research on that topic available (for example see BGV [BGV14] and FV [FV12] schemes) and it is not the main subject of this manuscript. But we will discuss the extent and implication of those properties.

Let $m_1$ and $m_2$ be two plaintexts and $c_1$ and $c_2$ their respective ciphers for private key $sk$. An encryption scheme is homomorphic for an operator $\otimes$ if and only if:

$$Decrypt(c_1 \otimes c_2, sk) = m_1 \otimes m_2$$

lattice-based scheme of the BGV/FV family are not fully homomorphic. They are homomorphic for a specific amount of additions and multiplications, depending on their parameters. If too many operations are performed on cyphertexts, decryption will fail to produce the correct plaintext. The homomorphic properties (the amount of additions and/or multiplication that can be performed before it fails) heavily depends on the size of $q$. With our parameters, we can easily perform several homomorphic additions, but at most two homomorphic products.

### Zero Encodings

Let $z = (z_1, z_2)$ be a static encryption of the plaintext 0, we can use homomorphic properties to modify our lookup tables:

$$S'^{i,j}_{sk}[\alpha_2^{i,j}][\alpha_1^{i,j}] = \alpha_2^{i,j} - \alpha_1^{i,j} \cdot sk^{i,j} + z_2^{i,j} - z_1^{i,j} \cdot sk^{i,j}$$

where $z_1^{i,j}$ and $z_2^{i,j}$ are, respectively, the residues of $z_1$ and $z_2$ in NTT/RNS form. Because $z$ is an encryption of 0, homomorphic properties will ensure that:

$$m'[i] = m[i] + 0$$

Therefore masking our lookup tables.

Note that by design, there are many different cyphertexts resulting from an encryption of 0 (This is due to IND-CCA properties of lattice-based schemes). Therefore an attacker in a white-box setting would not only have to understand that an encryption of 0 has been used to encode our tables, but *which one*.

### One Encodings

A similar technique consists in using a static encryption of the plaintext 1, and applying a multiplication of its decryption function instead of an addition. Because of the small amount of multiplicative homomorphic properties, this encoding can only be performed twice while still guaranteeing correct decryption. In our proposition, we recommend using only one of this type of encoding as, while it adds confusion, we will need one more multiplication for the last type of encodings. Let $u = (u1, u2)$ be an encryption of 1, we can modify our lookup tables:

$$S'^{i,j}_{sk}[\alpha_2^{i,j}][\alpha_1^{i,j}] = (\alpha_2^{i,j} - \alpha_1^{i,j} \cdot sk^{i,j}) \cdot (u_2^{i,j} - u_1^{i,j} \cdot sk^{i,j})$$

Both techniques can and should be used together to provide a better masking of our lookup tables.

### Masking Encodings

Let *mask* be a binary mask of size $n$, we can use the encryption method to encrypt each bit of this mask into one static cipher $\beta = (\beta_1, \beta_2)$. If we now modify our lookup tables in a way similar to the additive zero masking:

$$S'^{i,j}_{sk}[\alpha_2^{i,j}][\alpha_1^{i,j}] = \alpha_2^{i,j} - \alpha_1^{i,j} \cdot sk^{i,j} + \beta_2^{i,j} - \beta_1^{i,j} \cdot sk^{i,j}$$

This time, once decryption is finished, we get the following relation:

$$m'[i] = m[i] \oplus mask[i]$$

The user can then get knowledge of *mask* and remove it from the resulting plaintext, as knowledge of *mask* does not allow for the removal of the encoding provided by $\beta$ on the lookup tables.

### Rotating Encodings

Let $r = (r_1, r_2)$ be the encoding of the plaintext $X^{index}$. Because our elements lie in $R_q$, the computation:

$$m \cdot X^{index}$$

will simply result modulo 2 in a rotation of the coefficients of our plaintext. Modifying our lookup tables in the following manner:

$$S'^{i,j}_{sk}[\alpha_2^{i,j}][\alpha_1^{i,j}] = (\alpha_2^{i,j} - \alpha_1^{i,j} \cdot sk^{i,j}) \cdot (r_2^{i,j} - r_1^{i,j} \cdot sk^{i,j})$$

will yield after decryption a plaintext whose coefficients have been rotated. The user can then get knowledge of $X^{n-index}$ and perform the inverse rotation:

$$m' \cdot X^{n-index}$$

as only the knowledge of its encrypted form allows for the removing of this encoding from the lookup tables.

As explained in the section covering zero-encodings, the use of multiplicative homomorphic encodings is however limited due to the small amount of homomorphic properties our scheme possess.

## 3.3.2 On the Security and Limitations of our Design in its Current Form

First, we would like to mention the relation between the security level of our scheme and the memory consumption overhead of its white-box design. Usually, in lattice-based cryptography, the security level depends on the number of dimensions $n$. In our proposal, doubling the size of $n$ only doubles the number of lookup tables, which in turn doubles the size of our white-box design.

We propose a design for an asymmetric white-box based on a lattice-based cryptographic algorithm. All computations involving the secret key *sk* occurring during decryption are replaced with lookup tables. A series of homomorphic encodings are then applied to prevent trivial key extraction from those lookup tables. As a result of those encodings, an extra step of decoding is required to reverse the masking and rotation of the plaintext.

While using various combination of homomorphic encodings in our white-box design adds a layer of security against white-box attacks, extraction of the secret key is still possible. Mixing our key computation with static encryption of various values (zero, masks, rotations) increases the complexity of the computation being performed within our lookup tables. However, the resulting table still represents the result of a linear equation. While we increase the number of values unknown to the attacker (as opposed to a non-encoded version of the scheme where *sk* is the only unknown), an attacker will eventually be able to solve such a system. Indeed, an attacker not only has full knowledge of our lookup tables, he also has full control over their inputs/outputs. Those tables can be seen as oracles providing as many equations as the attacker may need to recover all encodings, making key extraction trivial.

To prevent key extraction, we need one of two countermeasures: non-linear encodings to prevent the solvability of our systems, or to be able to perform the final step of our decryption process, the reduction modulo 2, within our lookup tables. We propose two methods to achieve each countermeasures. Those will be discussed in the following section. However, they both present severe restrictions to be used in practice.

## 3.4 Strengthening the Design Against White-Box Attacks

In this section, we will cover two additional countermeasures that can be used to improve the resilience of our scheme against white-box attacks. Unlike the different transformations applied to our scheme so far, those countermeasures are either specific to a particular setting of white-box cryptography or present considerable overhead to both memory consumption and computation time. However, both those techniques provide a gain in security in a white-box setting.

### 3.4.1 External Encodings

In article [CEJO03], external encodings are mentioned as encodings applied to encrypted messages before they are sent to the party utilizing the white-box. In the case of a white-box performing decryption, for example, a message is encrypted and then encoded before it is sent to the party performing decryption. This means that the white-box used by said party needs to be constructed with those encodings in mind. In the case of AES, the first set of lookup tables therefore contains an input encoding reversing the effect of those external encodings.

Using the same principle, we propose the use of external encodings, in particular input external encodings, to solve the problem of vulnerability against algebraic attacks in a white-box context. However, we will weight this added security against the fact that external encodings are not particularly suited to the problem of asymmetric white-box cryptography.

Indeed, in symmetric white-box cryptography, we usually consider the case of a trusted party setting up a white-box specific to a secret key, and an untrusted party using said white-box to perform a cryptographic operation on data sent by the trusted party. From this point of view, it is not far fetched to consider the ability of the trusted party to apply specific external encodings to data before it is sent to the untrusted party.

However, in the case of asymmetric white-box cryptography, this may not be the case. Consider the case of a key exchange protocol for example, in which two party each send data encrypted by the other party's public key. Input external encodings would need to be applied to data before it is exchanged between the two parties, and collusion between the two could nullify the security added from the use of external encodings. Moreover, external encodings would need to be applied after the data is encrypted with the untrusted party public key, meaning only trusted party may use the untrusted party public key to perform the key exchange protocol. In other words, external encodings in a key-exchange protocol may only be used if one of the parties is trusted, which is far from the classical use of such protocols. In other words, the use of external encodings may only occur in a specific set-up that is not generic to all asymmetric white-box uses.

More specifically for our decryption algorithm, the use of external encodings as we implemented them can only provide security if the attacker has no knowledge of its own *public key*. Meaning he cannot forge his own encrypted messages. This goes against many application for asymmetric key white-box cryptography.

Let us now consider the use of external encodings in our scheme. Because such encodings will be applied and removed by different parties, external encodings do not need any linear or homomorphic properties. Let $G$ (respectively $H$) be a set of non-linear encodings over Galois' field $GF_{2^4}$ (one encoding for each residue derived from the NTT transform and the RNS decomposition). Let the tuple $(\alpha_1, \alpha_2)$ be an encryption of a message $m$. Let $\alpha_1^{i,j}$ (respectively $\alpha_2^{i,j}$) be the set of residues obtained after using an NTT transform and an RNS decomposition on $\alpha_1$ (respectively $\alpha_2$) on $\alpha_1$ using the exact same root of unity $\omega$ (for the NTT transform) and primed basis $\beta$ (for the RNS decomposition) as those used in our white-box performing decryption. The encrypting party can apply external encodings by computing:

$$\bar{\alpha_1}^{i,j} = G_{i,j}(\alpha_1^{i,j})$$

respectively:

$$\bar{\alpha_2}^{i,j} == H_{i,j}(\alpha_2^{i,j})$$

Before computing the new values $\bar{\alpha_1}$ and $\bar{\alpha_2}$ by using the CRT and inverse NTT transform on the resulting residues.

As far as our white-box is concerned, decoding is pre-computed within lookup tables. Let $G_{i,j}^{-1}$ (respectively $H_{i,j}^{-1}$) be the inverse encodings of $G_{i,j}$ (respectively of $H_{i,j}$):

$$S_{sk}^{i,j}[\bar{\alpha_2}^{i,j}][\bar{\alpha_1}^{i,j}] = H_{i,j}^{-1}(\bar{\alpha_2}^{i,j}) - G_{i,j}^{-1}(\bar{\alpha_1}^{i,j}) \cdot sk^{i,j}$$

This transformation adds a layer of resistance against white-box attacks as the encodings of both sets $G$ and $H$ do not need to be linear functions, as opposed to the homomorphic encodings covered in the previous section. Therefore, our lookup tables cannot be seen by an attacker as a solvable linear system. However,This new type of encodings is only usable in a specific context that may not be accessible to all applications.

### 3.4.2 A 3-table scheme

This next countermeasure aims to include the last part of our decryption process, the reduction modulo 2, in our lookup tables. Reducing the result of our lookup tables modulo 2 would effectively erase most of the information accessible to an attacker in a white-box context. However, this operation cannot be performed while our lookup tables use as input residues derived from an NTT transform coupled to an RNS decomposition. To solve this issue, we propose an alternative representation of our table-based scheme.

Let $S_{sk}$ be a set of lookup tables using a combination of homomorphic encodings:

- a rotating encoding $r = (r_1, r_2)$,
- an encoding of 0 $z = (z_1, z_2)$.
- an encoding of 1 $u = (u_1, u_2)$

For each key residue $sk^{i,j}$, the table $S_{sk}^{i,j}$ pre-computes the following equation:

$$\begin{aligned} S_{sk}^{i,j}[\alpha_2^{i,j}][\alpha_1^{i,j}] =& ((\alpha_2^{i,j} - \alpha_1^{i,j} \cdot sk^{i,j}) \cdot (r_2^{i,j} - r_1^{i,j} \cdot sk^{i,j}) + \\ & (z_2^{i,j} - z_1^{i,j} \cdot sk^{i,j})) \cdot (u_2^{i,j} - u_1^{i,j} \cdot sk^{i,j}) \end{aligned}$$

By simply developing this equation, we get:

$$\begin{aligned}S_{sk}^{i,j}[\alpha_2^{i,j}][\alpha_1^{i,j}] = &-\alpha_1^{i,j} \cdot r_1^{i,j} \cdot u_1^{i,j} \cdot (sk^{i,j})^3 + \\ &(\alpha_1^{i,j} \cdot r_1^{i,j} \cdot u_2^{i,j} + \alpha_2^{i,j} \cdot r_1^{i,j} \cdot u_1^{i,j} + \\ &\alpha_1^{i,j} \cdot r_2^{i,j} \cdot u_1^{i,j} + z_1^{i,j} \cdot u_1^{i,j}) \cdot (s_k^{i,j})^2 - \\ &(\alpha_2^{i,j} \cdot r_1^{i,j} \cdot u_2^{i,j} + \alpha_1^{i,j} \cdot r_2^{i,j} \cdot u_2^{i,j} + \\ &z_1^{i,j} \cdot u_2^{i,j} + \alpha_2^{i,j} \cdot r_2^{i,j} \cdot u_1^{i,j} + z_2^{i,j} \cdot u_1^{i,j}) \cdot (sk^{i,j}) + \\ &\alpha_2^{i,j} \cdot r_2^{i,j} \cdot u_2^{i,j} + z_2^{i,j} \cdot u_2^{i,j}\end{aligned}$$

Let us define two random elements $v \in R_q$ and $w \in R_q$ that will serve as masking for the following design. Let us then define two set of lookup tables $V_{sk}$ and $W_{sk}$ performing the following operation for all residues:

$$\begin{aligned}V_{sk}^{i,j}[\alpha_2^{i,j}][\alpha_1^{i,j}] = &-\alpha_1^{i,j} \cdot r_1^{i,j} \cdot u_1^{i,j} \cdot (sk^{i,j})^3 + \\ &(\alpha_2^{i,j} \cdot r_1^{i,j} \cdot u_1^{i,j} + z_1^{i,j} \cdot u_1^{i,j}) \cdot (s_k^{i,j})^2 - \\ &(\alpha_2^{i,j} \cdot r_1^{i,j} \cdot u_2^{i,j} + \alpha_1^{i,j} \cdot r_2^{i,j} \cdot u_2^{i,j} + z_1^{i,j} \cdot u_2^{i,j}) \cdot (sk^{i,j}) + \\ &z_2^{i,j} \cdot u_2^{i,j} + v^{i,j}\end{aligned}$$

$$\begin{aligned}W_{sk}^{i,j}[\alpha_2^{i,j}][\alpha_1^{i,j}] = &(\alpha_1^{i,j} \cdot r_1^{i,j} \cdot u_2^{i,j} + \alpha_1^{i,j} \cdot r_2^{i,j} \cdot u_1^{i,j}) \cdot (s_k^{i,j})^2 - \\ &(\alpha_2^{i,j} \cdot r_2^{i,j} \cdot u_1^{i,j} + z_2^{i,j} \cdot u_1^{i,j}) \cdot (sk^{i,j}) + \\ &\alpha_2^{i,j} \cdot r_2^{i,j} \cdot u_2^{i,j} + w^{i,j}\end{aligned}$$

such that for all residues $(\alpha_2^{i,j}, \alpha_1^{i,j})$:

$$V_{sk}^{i,j}[\alpha_2^{i,j}][\alpha_1^{i,j}] + W_{sk}^{i,j}[\alpha_2^{i,j}][\alpha_1^{i,j}] - v^{i,j} - w^{i,j} = S_{sk}^{i,j}[\alpha_2^{i,j}][\alpha_1^{i,j}]$$

The last computation required to perform decryption is an addition of the result of both tables and the removal of encoding values $v$ and $w$. Note that unlike the computations represented by $V_{sk}$ and $W_{sk}$, this last part of the decryption process is free of products. Let $a^{i,j}$ and $b^{i,j}$ be examples of residues obtained through the use of, respectively, lookup tables $V_{sk}^{i,j}$ and $W_{sk}^{i,j}$ such that:

$$\begin{aligned}a^{i,j} &= V_{sk}^{i,j}[\alpha_2^{i,j}][\alpha_1^{i,j}] \\ b^{i,j} &= W_{sk}^{i,j}[\alpha_2^{i,j}][\alpha_1^{i,j}]\end{aligned}$$

Let $a$ (respectively $b$) be an element of $R_q$ reconstructed from the residues $a^{i,j}$ (respectively $b^{i,j}$). Let $T$ be a set of $n$ lookup tables performing the following operation:

$$T_i[a[i], b[i]] = (m[i] = a[i] + b[i] - v[i] - w[i]) \mod 2$$

Such a lookup table $T$ not only computes the addition of our two intermediate elements and the removal of masking values $v$ and $w$, it also performs the last step of the decryption process that is the reduction modulo 2. However, such a table far exceed the size of our previous lookup tables $V$ and $W$. As an example, our implementation manages to produce a set of tables $T$ of size approximately 200MB. Therefore, a full set of tables $V$, $W$ and $T$ weight around 240MB, increasing the size of our white-box design by a factor 10.

Let us now consider the security of this design against an attacker in a white-box model. Table $T$ does not yield any information to the attacker, as the modulo 2 operation precomputed within it destroys exploitable data. However, tables $V$ and $W$ can be used together to retrieve the secret key residues $sk^{i,j}$. Indeed, tables $V$ and $W$, when combined, both represent linear systems that can be solved in a similar manner as the lookup table $S$ of our initial design.

The silver lining comes from the fact that the distribution of computation between $V$ and $W$ can be randomised from one white-box implementation to the other. Furthermore, from an attacker point of view, both tables are indistinguishable from each other (they have the same inputs and produce a result of similar entropy), opening the door for various obfuscation techniques impeding the effort of an attacker trying to isolate one table from the other.

To achieve security against an attacker in a white-box model motivated enough to break obfuscation techniques, we would need to incorporate the modulo 2 operation in one unique table. Unfortunately, this would require about 10 To of data, which is far from practical for use in the industry.

In conclusion, the 3-table design is an interesting attempt at achieving protection against an attacker in a white-box model by including the modulo 2 operation to our lookup tables, as it opens the way to data obfuscation techniques. However, it does not definitely solve the problem of asymmetric white-box cryptography.

## 3.5 Implemention of an Asymmetric White-box

In this section, we will discuss how we implemented the generation of a practical asymmetric white-box proposal, following the initial design described in section 3.3. The implementation of the generation of our fully table-based white-box initial design can be divided in two parts:

- the creation of the white-box data, in our case the lookup tables that precompute our target computation for each residue given by our NTT/RNS decompositions and apply homomorphic encodings,
- the executable binary that will perform the decryption process by transforming input data accordingly to our NTT/RNS decomposition, fetch the corresponding data for each residue and finalize the decryption process (removal of noise and homomorphic encodings).

As far as our implementation goes, an end-user of our design should have on his/her device an executable file and a data file. For obfuscation purposes, a real-life white-box implementation may present only one file combining the two, with obfuscation techniques impeding the process of mapping what is data against what is executed code. Before presenting how we implemented both parts of the process, we will detail our implementation of the NTT and RNS modules as they are both involved in each part of the implementation.

### 3.5.1 Setting Up the NTT and RNS Modules and the Montgomery Multiplication Algorithm

#### The NTT Module

In this section, we will discuss how we implemented our Number Theoretic Transform (NTT) module. This section will not cover the mathematical basics of the NTT technique, as this subject is discussed in chapter 1. Various efficient libraries implement the NTT technique, especially when considering lattice-based cryptography. In particular, if one seeks to implement our proposal with more optimized tools, we would recommend the library NFLlib as an efficient library providing various modules for NTT-based lattice cryptography.

However, as far as our work is concerned, we made the choice to implement our own NTT module. This decision was driven by various factors. Mainly, we wanted to maintain a level of control and knowledge of our NTT module, whether for an easier integration to our white-box modules or for better control of the resulting executable binary, in case it needed obfuscation. Another point in favor of a module of our own, as opposed to an external library, is that a lot of external libraries implementing modules for lattice-based cryptography tend to perform the RNS decomposition *before* the NTT transform. As explained in section 3.3, we need our scheme to perform those operations in the reverse order. This resulted in many implementation problems with some external libraries, forcing us to write code with the sole purpose of managing input/outputs to match our implementation. With our own implementation of the module, this became unnecessary.

Our implementation is based on the Cooley-Tukey butterfly algorithm [CT65], providing a complexity in $O(nlogn)$, using a negative wrapped convolution technique [LMPR08] to keep a vector size of $n$ throughout our computations. A similar algorithm was implemented and presented in the following article [LN16b]. Note that the article [LN16b] mentions that their implementation of the Cooley-Tukey butterfly algorithm requires as input a lookup table holding powers of a root of unity $\omega$ in bit-reversed order. Because our ntt module is also used in the executable binary provided to the end-user of our scheme, we did not wish to add more pre-computed data to the equation. Therefore, the computation of powers of $\omega$ in bit-reversed orders is done within our NTT module. Algorithms for the forward and inverse NTT are presented in the following:

**Algorithm 15:** NTT

```
input  : $vector, root, n, q$
output: None

$levels = n.bit\_length() - 1$
$tmp = 1$
$powtable = []$
for $i$ in range($n$) do
    $vector[i] = vector[i] * tmp \mod q$
    if not $i \mod 2$ then
        $powtable.append(tmp)$
    end
    $tmp = tmp * root \mod q$
end
for $i$ in range($n$) do
    $j = reverse(i, levels)$
    if $j > i$ then
        vector[i], vector[j] = vector[j], vector[i]
    end
end
$size = 2$
while $size <= n$ do
    $halfsize = size/2$
    $tablestep = n//size$
    for $i$ in range(0, $n$, $size$) do
        $k = 0$
        for $j$ in range($i$, $i + halfsize$) do
            $l = j + halfsize$
            $left = vector[j]$
            $right = vector[l] * powtable[k]$
            $vector[j] = (left + right) \mod q$
            $vector[l] = (left - right) \mod q$
            $k+ = tablestep$
        end
    end
end
```

**Algorithm 16:** InvNTT

```
input  : $vector, unroot$
output: None
$ninv = n^{-1} \mod q$
$levels = n.bit\_length() - 1$
$tmp = 1$
$powtable = []$
$powtable2 = []$
for $i$ in range(n) do
    $vector[i] = vector[i] * tmp \mod q$
    if not $i \mod 2$ then
        $powtable.append(tmp)$
    end
    $powtable2.append(tmp)$ $tmp = tmp * unroot \mod q$
end
for $i$ in range(n) do
    $j = reverse(i, levels)$
    if $j > i$ then
        vector[i], vector[j] = vector[j], vector[i]
    end
end
$size = 2$
while $size <= n$ do
    $halfsize = size/2$
    $tablestep = n//size$
    for $i$ in range(0, n, size) do
        $k = 0$
        for $j$ in range(i, i + halfsize) do
            $l = j + halfsize$
            $left = vector[j]$
            $right = vector[l] * powtable[k]$
            $vector[j] = (left + right) \mod q$
            $vector[l] = (left - right) \mod q$
            $k+ = tablestep$
        end
    end
end
for $i$ in range(n) do
    $vector[i] = vector[i] * ninv * powtable2[i] \mod q$
end
```

### The RNS Module

The RNS module is pretty straightforward, it consists in simple modular reductions to reduce an integer into our RNS basis, and a Chinese Remainder Theorem to go back from our decomposed representation. Both algorithms are detailed in the following:

**Algorithm 17:** RNS

```
input  : $x$, $base$, $length$
output: None
$res = []$
for $i$ in range(length) do
    $res.append(x \mod base[i])$
end
return $res$
```

**Algorithm 18:** CRT

```
input  : $x$, $base$, $length$
output: None
$res = 0$
$B = \prod base$
for $i$ in range(length) do
    $B_i = \frac{B}{base[i]}$
    $res = res + (x[i] * B_i * (B_i^{-1} \mod base[i]))$
end
return $res$
```

### Montgomery's Multiplication Algorithm

In this section, we will discuss Montgomery's multiplication algorithm. More specifically, we will discuss which part of the algorithm is being processed by the executable binary provided to the end-user of our white-box design, and which part is being pre-computed within our set of lookup tables.

The following pseudo-code illustrate a simple example of an implementation of Montgomery's algorithm for modular products:

**Algorithm 19:** Montgomery

```
input  : $x, y, \beta_1, \beta_2, q, k$
output: $r$
$x_1 = RNS(x, \beta_1, k)$
$y_1 = RNS(y, \beta_1, k)$
$x_2 = RNS(x, \beta_2, k)$
$y_2 = RNS(y, \beta_2, k)$
$M_1 = \prod \beta_1$
$M_2 = \prod \beta_2$
$N = q$
$N_1 = RNS(N, \beta_1, k)$
$N_2 = RNS(N, \beta_2, k)$
$Ninv = RNS(N^{-1} \mod M_1, \beta_1, k)$
$Minv = RNS(M_1^{-1} \mod M_2, \beta_2, k)$
for $j$ in range(k) do
    $z_1[j] = ((x[j] * y[j]) * (-Ninv[j])) \mod \beta_1[j]$
end
$z_2 = shift\_modulus(z_1, \beta_1, \beta_2)$ for $jinrange(k)$ do
    $r = (((x_2[j] * y_2[j]) + (z_2[j] * N_2[j])) * Minv[j]) \mod \beta_2[j]$
end
return $r = r * M_1 \mod N$
```

This algorithm is going to be adapted to our white-box design in the following way:

- during the generation procedure, computations of $z_1$ and $r$ are going to be pre-computed within lookup tables, they are also going to be modified to fit our target decryption function and include homomorphic encodings computations,
- in the binary executable file, computations of $z_1$ and $r$ are going to be replaced by a fetch of data from the lookup tables,
- the pre-computation of $z_1$ and $r$ are gonna be modified to include homomorphic encodings.

The pre-computation leading to the manufacture of our lookup tables will be presented in the following section 3.5.2, while the final decrypt function will be presented in section 3.5.3.

### 3.5.2 Generation of the White-Box Data

The first step in generating a white-box matching our design is to generate the set of encoded lookup tables that will be used to pre-compute our target computation involving the secret key. In classical AES white-box designs, this process can be presented in two steps:

- creation of the set of lookup tables matching the different operations composing the AES cryptosystem,
- generation and application of the various encodings to the network of lookup tables.

In our implementation of an asymmetric white-box design, the generation process needs to be done a bit differently. Because homomorphic encodings are mixed with elements of our secret key, and because computation is done with respect to Montgomery's algorithm, the application of our encodings should not be done independently of the generation of lookup tables process. Therefore, we need to generate our homomorphic encodings first (and decompose them in a set of residues with our NTT and RNS modules) before we generate our lookup tables. The process of generating our lookup tables will then have to pre-compute our target computation and the application of our encodings in one function.

### Generating Homomorphic Encodings

To generation our homomorphic encodings, we first need to generate static plaintext values and encrypt them to obtain static pairs of encrypted vectors that will be used as homomorphic encodings. The plaintext values need to be stored temporarily as they will be of use when generating the executable binary in the removal of homomorphic encodings process. homomorphic encoding vectors are generated through the following functions:

- The encryption functions as described in 3.2.

**Algorithm 20:** Encrypt

```
input  : $pk, m$
output: $\alpha = (\alpha_1, \alpha_2)$
$u$ = GaussianPoly($n, q, k$)
$e_1$ = 2·GaussianPoly($n, q, k$)
$e_2$ = 2·GaussianPoly($n, q, k$)
$\alpha = (pka \cdot u + e_1, pkb \cdot u + e_2 + m)$
```

- The generation of a "zero" homomorphic encoding vector, i.e. an encrypted vector of the plaintext value "0".
- The generation of a "one" homomorphic encoding vector, i.e. an encrypted vector of the plaintext value "1".
- The generation of a "rotate" homomorphic encoding vector, i.e. an encrypted vector of the plaintext value "$X^r$" for a static value of $r$.
- The generation of a "mask" homomorphic encoding vector, i.e. an encrypted vector of the plaintext variable *mask*.

Once all homomorphic encoding vectors are generated, they need to be processed into a set of residues with our RNS and NTT modules. For details on those modules, refer to sections 3.3 and 3.3. For each homomorphic encoding, we should have two sets of residues matching both components output by the encoding process. Once all residues have been computed, the generation process can move on to the generation of lookup tables.

**Algorithm 21:** Generate Homomorphic Encodings

```
input  : $pk$, $mask$, $r$, $n$
output: $\alpha^0$, $\alpha^1$, $\alpha^r$, $\alpha^{mask}$

$m = [0]^{*n}$
$\alpha^0 = Encrypt(pk, m)$
$m[0] = 1$
$\alpha^1 = Encrypt(pk, m)$
$m[0] = 0$
$m[r] = 1$
$\alpha^r = Encrypt(pk, m)$
for $i$ in range($n$) do
    $m[i] = mask[i]$
end
$\alpha^{mask} = Encrypt(pk, m)$
return $\alpha^0$, $\alpha^1$, $\alpha^r$, $\alpha^{mask}$
```

### Setting Up the Lookup Tables

In order to pre-compute our lookup tables, we need to create an entry for each residue/each coefficient of the ntt construct, create a lookup table that process the target function with respect to Montgomery's algorithm. Bear in mind we need 2 tables for each residue, one for each basis used in Montgomery's algorithm.

**Algorithm 22:** Generate Lookup Tables

```
input  : $sk, q, n, \beta_1, \beta_2, k$
output:
$M_1 = \prod \beta_1$
$M_2 = \prod \beta_2$
$N = q$        %% change in variable to better match Montgomery's literature
$sk = ntt(sk, root, n, q)$
$N_1 = RNS(N, \beta_1, k)$
$Ninv = RNS(N^{-1} \mod M_1, \beta_1, k)$
$Minv = RNS(M_1^{-1} \mod M_2, \beta_2, k)$
for $i$ in range(n) do
    $z\_1 = RNS(sk[i], \beta_1, k)$
    $z\_2 = RNS(sk[i], \beta_2, k)$
    for $jinrange(k)$ do
        for $x$ in range($\beta_1[j]$) do
            for $yinrange(\beta_1[j])$ do
                $Q = ((x * z\_1[j] + y) * (-Ninv[j])) \mod \beta_1[j]$
                $store\_data(x, y, i, j, Q, 0)$[a]
            end
        end
        for $x$ in range($\beta_2[j]$) do
            for $yinrange(\beta_2[j])$ do
                $R = ((x * z\_2[j] + y) * Minv[j]) \mod \beta_2[j]$
                $store\_data(x, y, i, j, R, 1)$[b]
            end
        end
    end
end
```

[a]Last argument is a simple flag to differentiate the pre-computation of Q from R
[b]Last argument is a simple flag to differentiate the pre-computation of R from Q

At this point, the homomorphic encodings need to be added to the generation process. Let us take for example the addition of a masking homomorphic encoding 3.3.1. Let $\alpha^{mask} = (\alpha_1^{mask}, \alpha_2^{mask})$ be a valid encryption of the static binary vector *mask*. Let $\alpha_1^{mask}[i,j]$ be a residue corresponding to the $i^{th}$ coefficient represented in the $j^{th}$ basis module of $\beta_1$ of the vector $\alpha_1^{mask}$. Respectively, let $\alpha_2^{mask}[i,j]$ be the matching residue of the vector $\alpha_2^{mask}$. To include the homomorphic addition of our mask to the pre-computation of Montgomery's multiplication algorithm, the computation of $Q$ becomes:

$$Q = ((x * z\_1[j] + y + \alpha_2^{mask}[i,j] * z\_1[j] + \alpha_1^{mask}[i,j]) * (-Ninv[j])) \mod \beta_1[j]$$

Respectively, let $\gamma_2^{mask}[i,j]$ and $\gamma_2^{mask}[i,j]$ be similar residues for basis $\beta_2$, the computation of $R$ becomes:

$$R = ((x * z\_1[j] + y + \gamma_2^{mask}[i,j] * z\_1[j] + \gamma_1^{mask}[i,j]) * Minv[j]) \mod \beta_2[j]$$

The method $store\_data()$ will store the results of our lookup tables. For the sake of simplicity, we chose to store our lookup tables in a different file from the executable binary. For practical application, $store\_data()$ should store our lookup tables within our binary executable file, and the data access process should be obfuscated accordingly. In our implementation, we added an extra argument to our function *store_data*() as a flag to differentiate the pre-computations of the residues of $Q$ and $R$.

### 3.5.3 The White-Box Executable Binary

The white-box executable binary, that is to say the binary that will be executed by the end user to decrypt encrypted content, will hold the following content :

- various static values specific to the instance in which public and private cryptographic keys were generated, like $n$ and $q$,
- various static values specific to both the NTT and RNS transforms, in particular, the $n^{th}$ root of unity and its inverse, and the chosen basis, $\beta_1$ and $\beta_2$, for RNS decompositions,
- various static values representing the plaintext values involved in the manufacture of the homomorphic encodings during the generation process,
- a decryption function performing the decryption algorithm on an encrypted message. The function transforms input encrypted messages with the NTT and RNS modules, then performs the decryption algorithm on each residue with respect to Montgomery's multiplication algorithm. Finally, the final steps of the decryption process (removal of noise and homomorphic encodings) are performed,
- a function fetching data from the lookup table for a specific residue.

The following algorithm depicts the aforementioned decryption function:

**Algorithm 23:** Decrypt

```
input  : $\alpha, n, q, root, unroot, \beta_1, \beta_2$, k
output: $m$

$\alpha_0 = NTT(\alpha[0], root)$
$\alpha_1 = NTT(\alpha[1], root)$
$Q = init\_vector()$
$R = init\_vector()$
$M_1 = \prod \beta_1$
$M_2 = \prod \beta_2$
$N_1 = RNS(q, \beta_1, k)$
$N_2 = RNS(q, \beta_2, k)$
$Ninv = RNS(N^{-1} \mod M_1, \beta_1, k)$
$Minv = RNS(M_1^{-1} \mod M_2, \beta_2, k)$
for i in range(n) do
    $x_1 = RNS(\alpha_0[i], \beta_1, k)$
    $y_1 = RNS(\alpha_1[i], \beta_1, k)$
    $x_2 = RNS(\alpha_0[i], \beta_2, k)$
    $y_2 = RNS(\alpha_1[i], \beta_2, k)$
    $Q_1 = RNS(Q, \beta_1, k)$
    $Q_2 = RNS(Q, \beta_2, k)$
    $R_1 = RNS(R, \beta_1, k)$
    $R_2 = RNS(R, \beta_2, k)$
    for j in range(k) do
        $Q_1[j] = fetch\_data(i, j, x_1[j], y_1[j], 0)$
    end
    $Q = CRT(Q_1, \beta_1, k)$
    $Q_2 = RNS(Q, \beta_2, k)$
    for j in range(k) do
        $R_2[j] = (Q_2[j] * N_2[j] * Minv[j]) \mod \beta_2[j]$
        $R_2[j] = R_2[j] + fetch\_data(i, j, x_2[j], y_2[j], 1)$
    end
    $R[i] = CRT(R_2, \beta_2, k)$
    $R[i] = R[i] * M \mod q$
end
$R = invNTT(R, unroot)$
$m = init\_vector()$
$mask = fetch\_data\_mask()$
for i in range(n) do
    if $m[i] > q/2$ then
        $m[i] = (1 - m[i] \mod 2) \oplus mask[i]$
    end
    else
        $m[i] = (m[i] \mod 2) \oplus mask[i]$
    end
end
$rotate = fetch\_data\_rotate()$
$m = rotate\_vector(m, rotate)$
```

This algorithm performs the decryption operation while the key remains embeded within our lookup tables. Those tables are protected by various homomorphic encodings. Note that the presence of the static variable *mask* and the *rotate_vector* method assumes both types of homomorphic encodings have been used during the generation process. If some encodings are not used during the generation process, the *Decrypt* algorithm should be changed accordingly.

### 3.5.4 On the Obfuscation Process and Additional Modules

#### Obfuscation of our Initial Design

Our proposal was fully implemented in Python and was designed to serve as a proof of concept regarding the feasibility of an asymmetric white-box design. Because of this, our implementation as it is does not make use of obfuscation techniques to better protect against an attacker in a white-box setting. In this section, we will list a few basic obfuscation techniques that should at least be considered for practical use of an asymmetric white-box. Those techniques would probably require a second implementation in a low-level programming language, as many obfuscators [BCGM19] perform operations at an intermediate level during the compilation process.

- Our lookup tables should be stored within our executable binary file. The functions *store_data* and *fetch_data* should work in concert accordingly. *fetch_data* should be equipped with a scheme to fetch the relevant data from randomized locations within the binary file while *store_data* should place the corresponding data accordingly.
- Said data placement scheme should use obfuscation techniques, such as the opaque constant technique for example, to impede a reverse engineer from rebuilding our lookup tables.
- If possible (with respect to potential memory consumption limitations), lookup tables should be duplicated and additional fake dependencies should be added to the fetching process to further impede the reverse engineer.

Those basic techniques do not increase the theoretical security of our initial design against an attacker in a white-box model. However, from a practical point of view, obfuscation has been proven an effective tool to slow down attacks against white-box constructs [SW08]. However, it has yet to solve the white-box problem altogether [GPRW19].

#### Implementation of External Encodings and the 3-Table scheme

In this section, we discuss the implementation of our additional countermeasures designed to defeat an attacker in a white-box setting. More precisely, the use of external encodings and our 3-tabled scheme.

Regarding external encodings, their implementation should be rather straightforward. Those simply consist in encodings applied by the party sending encrypted messages which should be removed by the executable binary while performing the decryption process. In the case of a masking encoding for example, a dynamic encryption of a mask should be homomorphically added to the encrypted data before being sent to the end-user with the underlying mask. Said mask simply needs to be processed by the binary executable file and removed during the final step of decoding. Practical usage should ensure that masks are randomly generated for each message sent to the end-user.

Regarding the 3-table scheme presented in section 3.4.2, this technique requires an overhaul rewrite of both the generation process and the *Decrypt* algorithm. Mainly, the table generation process will have to pre-compute:

- residues from the computation of $V$, using Montgomery's multiplication algorithm,
- residues from the computation of $W$, using Montgomery's multiplication algorithm,
- coefficients of vector $T$.

This last lookup table $T$ will have a dimension that far exceeds that of $V$ and $W$ as it does not benefit from the RNS representation. However, this overhead will be balanced by the fact that it only outputs one bit as a result. Note that this table does not make use of Montgomery's multiplication algorithm as it simply performs an addition of coefficients in a non-RNS representation.

## 3.6 Conclusion on Asymmetric White-Box

In this chapter, we presented our own proposal for an Asymmetric white-box scheme. We started from a straightforward lattice-based cryptographic scheme presented in [AMBG+16] and [GAGL15] and transformed it into a fully table based implementation. We then used latent homomorphic properties of our scheme to add homomorphic encodings to prevent key-extraction in a white-box attack model.

The resulting scheme was fully implemented in Python and managed to hold a limitation in size of 20MB. This threshold, chosen according to a known AES white-box design competition (WhibOx) as part of the CHES conference [CHE17] [CHE19], is meant to justify that our design is somewhat practical in real-life applications. However, we mentioned that the implementation of our design can probably be improved by simply switching to a low-level language like C or C++. In addition, we discussed the use of obfuscation techniques as well as the implementation of additional countermeasures to provide more security against an attacker in a white-box model.

Those additional countermeasures were discussed in section 3.4 and provide security against algebraic attacks. However, each of them comes with its specific drawback. The first technique imposes restrictions on the type of protocol that would benefit from our white-box implementation, but retain the same limit in memory consumption overhead as the initial construct. The second technique does not assume any restriction on the protocol used but triples the memory consumption overhead, making it less practical than our original construct.

This proposal is the first public proposal for an asymmetric white-box design. We hope that this work will promote public research on a less known field of white-box cryptographic study. Field that is already being used by the industry despite the lack of well-defined design guidelines.

# Chapter 4

# Polynomial Bijections Modulo $2^n$ and their use in Obfuscation

The following chapter describes our work on polynomial bijections in the ring $\mathbb{Z}_{2^n} = \mathbb{Z}/2^n\mathbb{Z}$ and their use in obfuscation. From that work, we published two articles, one was published and presented at the international workshop on software protection SPRO (Software PROtection) [BERR16], while the second was published and presented at the international conference ICMS (International Congress on Mathematical Software) [BKRS18].

## 4.1 Obfuscation and Binary Permutation Polynomials

### 4.1.1 Obfuscation and its Use in White-Box Cryptography

As cryptography is designed to resist cryptanalysis, obfuscation is a research field designed to impede the reverse engineering process. In practice, an adversary in a white-box model does not usually have a complete understanding of its target. In the case of software protection, this adversary is usually confronted to an executable file that is being run on a device that he controls. This type of material is usually written in assembly code that is not meant to be understandable to the human eye (as compilation tends to rewrite the code for performances). The reverse engineering process aims to understand the various operations performed by a software in order to, for example, gain knowledge of sensitive information manipulated by that software. As opposed to it, the purpose of obfuscation is to modify a program so that, while it still performs all the desired operations, it is hard to humanly comprehend what the program does and how it does it.

While white-box cryptography aims to protect a software from leaking cryptographic keys from an attacker who has total control over the device the software is being run on, obfuscation aims to modify how the software computes various operations in order to prevent the attacker from understanding what the software is doing. While strictly speaking both field have different objectives, they both address in their own way the problem that is the white-box model. In practice, both field are usually entwined in the process of protecting a software in a white-box model. Practical white-box implementations often make use of obfuscation as an additional layer of defense against an attacker in a white-box model.

### 4.1.2 Mixed Boolean Arithmetics Based Obfuscation

In the context of obfuscation, Mixed Boolean Arithmetics expressions (MBA) refers to a computation mixing arithmetic computations over the ring of integers modulo $2^n$ and Boolean expressions. Obfuscating an arithmetic expression (respectively a

Boolean expression) is a challenging endeavour. Indeed, the simplification of arithmetic (respectively Boolean) expressions is a well researched topic and many automatic solvers already exist. On the other hand, simplifying expressions mixing both arithmetic and Boolean operators proves far more complex.

For example, the expression E written as For example, the following expression E is an example of an obfuscated expression using MBA:

$$E = (x \oplus y) + 2 * (x \wedge y)$$

which is equivalent to the expression:

$$E = x + y$$

Recent work using pattern matching techniques have been conducted [Eyr17] and yielded promising results to break MBA obfuscation techniques, but MBA based obfuscation is still a very strong tool for obfuscation.

MBA expressions are also presented as part of an obfuscation technique in article [ZMGJ07]. In particular, this article utilises binary permutation polynomials as part of its arithmetic computations over the ring of integers modulo $2^n$. To use those elements, one must solve the inversion problem of binary permutation polynomials in the ring $\mathbb{Z}_{2^n}$. Solving this problem was the main contribution of both our articles [BERR16] [BKRS18].

### 4.1.3 Binary Permutation Polynomials and Mixed Boolean Arithmetics Based Obfuscation

In this section, we will give a definition of binary permutation polynomials (polynomials with coefficients in an integer ring modulo a power of 2) as they are defined in Rivest's article [Riv01]. We will then discuss their use in Mixed Boolean Arithmetics (MBA) based obfuscation.

#### Binary Permutation Polynomials

Permutation polynomials in the context of Galois' fields are very well studied in particular for their applications in cryptography. The study of binary polynomials (polynomials with coefficients in an integer ring modulo a power of 2) is less extensive, but it has been shown that they are important for computer security. As discussed in [ZMGJ07], a straightforward application of binary permutation polynomials is obfuscation, in particular MBA based obfuscation.

In the sequel, we will be mainly interested in bijective functions of $\mathbb{Z}_{2^n} = \mathbb{Z}/2^n\mathbb{Z}$ the integer ring modulo $2^n$. More precisely, we will consider objects as defined in the following:

**Definition 5** *Let $A$ be a finite ring. A map $f : A \to A$ is said to be a* polynomial function *if there exists a polynomial $P$ in $A[x]$ such that $\forall a \in A : f(a) = P(a)$. If, moreover $f$ is bijective, this function and the polynomial $P$ are said to be a* permutation polynomial *of $A$. When $A = \mathbb{Z}_{2^n}$ we will call them* binary permutation polynomial.

In [Riv01] Rivest provides the following result which characterizes binary permutation polynomials.

**Theorem 2 ([Riv01])** *Let $A = \mathbb{Z}_{2^n}$ with $n \geq 2$ and let $P(x) = a_0 + a_1 x + \cdots + a_d x^d$ be a polynomial with integral coefficients. The polynomial $P$ represents a function $f$ of $A$ which* *is a binary permutation polynomial if and only if $a_1$ is odd, $(a_2 + a_4 + a_6 + \dots)$ is even, and $(a_3 + a_5 + a_7 + \dots)$ is even.*

It is worth to note that the function $f$ in the preceding theorem can be represented by many different polynomials. More precisely, we have the following results.

**Proposition 1** *Let $A$ be a finite ring. The ring $\mathcal{F}(A)$ of polynomial functions on $A$ is isomorphic to the quotient ring $A[X]/I$ where $I$ is the polynomial ideal defined by $I = \{P \in A[x] : \forall a \in A, P(a) = 0\}$. In particular, the group $\mathcal{P}(A)$ (for the composition) of permutation polynomials of $A$ is a subset of this quotient ring.*

In the case where $A$ is a finite field, the structure of $\mathcal{F}(A)$ can be identified (since every function is polynomial) [MN85], but in the case of a finite ring it is far from easy. When $A$ is a finite ring, there are many papers providing theoretical results on $\mathcal{F}(A)$ or $\mathcal{P}(A)$ (e.g. structure, isomorphism, cardinality) but only few reports on algorithmic aspects. In the rest of this chapter, we will address the problem of efficiently computing a polynomial representing the inverse of a given binary permutation polynomial and consider some applications to obfuscation.

In comparison with finite field permutation polynomials, the binary polynomials allow fast computation of bijective functions, since they can be directly implemented with low level arithmetic operations on computers. Moreover their use adds diversity to obfuscation techniques. The last point is of primary concern for obfuscation. Indeed, obfuscation usually does not rely on one overwhelming method, but on an aggregation of several layers of different techniques that aim to prevent automated attacks. As such, we propose a tool, in the form of an inversion algorithm, to allow the use of new classes of polynomials for the obfuscation technique described in [ZMGJ07] that are resistant to the attacks defined in [BJLS17].

### On the Use of Binary Permutation Polynomials in Mixed Boolean Arithmetics Expressions

In [ZMGJ07], Zhou et al. present a constant obfuscation based on two components: *Mixed Boolean Arithmetics* (MBA) expressions and binary permutation polynomials. They present their own subset of binary permutation polynomials for which they provide a closed-form formula for inversion. For a given degree $m > 0$, they define the following subset of $\mathcal{P}(\mathbb{Z}_{2^n})$:

$$P_m(\mathbb{Z}_{2^n}) = \left\{ \sum_{i=0}^{m} a_i x^i \mid a_1 \text{ is odd and } a_i^2 = 0 \text{ for } i > 1 \right\}$$

and prove the following theorem.

**Theorem 3 ([ZMGJ07])** *The set $P_m(\mathbb{Z}_{2^n})$ is a permutation group under the functional composition operator $\circ$. The inverse of an element $f(x) = \sum_{i=0}^{m} a_i x^i \in P_m(\mathbb{Z}_{2^n})$ is given by $g(x) = \sum_{j=0}^{m} b_j x^j$ where each coefficient can be computed by: $b_m = -a_1^{-m-1} a_m$, $b_1 = a^{-1} - a_1^{-1} \sum_{j=2}^{m} j a_0^{j-1} A_j$, $b_0 = -\sum_{j=1}^{m} a_0^j b_j$ and*

$$b_k = -a_1^{-k-1} - a_1^{-1} \sum_{j=k+1}^{m} \binom{n}{r} a_0^{j-k} A_j, m - 1 \geq k \geq 2$$

*where $A_m = -a_1^{-m} a_m$, and $A_k$ are recursively defined by*

$$A_k = -a_1^{-k} a_k - \sum_{j=k+1}^{m} \binom{j}{k} a_0^{j-k} A_j, \quad \textit{for } 2 \leq k < m.$$

From this set, Zhou et al. propose an obfuscation technique based on the composition of two binary permutation polynomials $P(x)$ and $Q(x) = P^{-1}(x)$, and an $m$-variable MBA identity $E = 0$ with multiple non-zero terms. For a certain constant $K \in \mathbb{Z}_{2^n}$, one has $K = Q(P(K)) = Q(E + P(K))$. Expanding the expression yields an obfuscated expression containing $m$ variables and yet having always $K$ as an output value. Such an expression can be used in opaque constant obfuscation of the value $K$ or be part of an opaque predicate obfuscation technique.

While this work provides an interesting way of using binary permutation polynomials in obfuscation, it presents two drawbacks: firstly, only a small subset of $\mathcal{P}(\mathbb{Z}_{2^n})$ is used and secondly, public work describing attacks of this technique [BJLS17] was later published.

#### Attack of MBA Obfuscation

In [BJLS17], Biondi et al. describe three approaches to attack the MBA-based constant obfuscation: algebraic simplification, SMT solver-based deobfuscation, and synthesis-based deobfuscation. The algebraic simplification method is of first interest for us, since it is strongly based on the form (see Theorem 3) of the set of polynomials defined by Zhou et al. As stated by the authors of the attack, the recovering of both polynomials $P$ and $Q$ is based on the fact that the coefficients of $Q(x) = \sum_{j=0}^{m} b_j x^j$ are all even except for $b_1$. It is indeed stressed in [BJLS17] that, should polynomials from another family that the one of Theorem 3 be used to obfuscate, the reduction attack presented would fail.

Our work aimed to propose efficient methods to invert all permutation polynomials with respect to the original definition provided by Rivest [Riv01], expanding the family of polynomials presented in Theorem 3. With this definition of permutation polynomials, many permutation polynomials resistant to the attack described in [BJLS17] may be use with the technique described in [ZMGJ07].

### 4.1.4 On the Inversion Problem Modulo $2^n$

In order to prevent the attack described in [BJLS17] against MBA obfuscation technique detailed in [ZMGJ07], we need to have access to binary permutation polynomials that do not belong in the following subset:

$$P_m(\mathbb{Z}_{2^n}) = \left\{ \sum_{i=0}^{m} a_i x^i \mid a_1 \text{ is odd and } a_i^2 = 0 \text{ for } i > 1 \right\}$$

In particular, we need binary permutation polynomials, as described by Rivest in [Riv01], for which $a_i^2 \neq 0$ for $i > 1$. The problem at hand is that the inversion method proposed in [ZMGJ07] is a closed form formula specific to this subset of binary permutation polynomials. Therefore, another algorithm for permutation polynomials is required.

### A First Attempt Using Lagrange Interpolation

In [BERR16], we first described our attempt to solve the inversion problem by using Lagrange interpolation. Lagrange interpolation algorithm, given a set of $d$ points and their respective evaluation by a function $f$, yields a polynomial of degree at most $d$ that approximate the behaviour of function $f$.

The idea was quite simple, given $P$ a binary permutation polynomial, we can generate two sets of elements, the set of points $x = (x_0, x_1, \ldots, x_d)$ and their evaluation $y = (y_0, y_1, \ldots, y_d)$ by the polynomial $P$. That is to say:

$$y_i = P(x_i) \text{ for } 0 \le i \le d$$

Let $Q$ be our target binary permutation polynomial which permutation is inverse to $P$. It holds that:

$$x_i = Q(y_i) \text{ for } 0 \le i \le d$$

Therefore, by switching our two sets of points $x$ and $y$, it should be possible to interpolate the inverse polynomial $Q$. However, working modulo $2^n$ makes the task more challenging.

In order to perform a classical Lagrange interpolation, one must compute the following value:

$$Q = \sum_{j=0}^{d} x_j \prod_{i \neq j} \frac{(X - y_i)}{(y_j - y_i)} \tag{4.1}$$

However, the case of binary permutation polynomials, the Expression (4.1) is not always computable, as half of the elements in $\mathbb{Z}_{2^n}$ are zero-divisors. By considering the fraction's denominator, we need to compute the inverse of the following value:

$$z = \prod_{0 \le i < j \le d_{inv}} (y_j - y_i) \tag{4.2}$$

Thus, the computation of the Lagrange interpolation is possible if and only if $z$ is inversible.

However, it can be shown that if the $y_i$ are in the ring $\mathbb{Z}_{2^n}$, then as soon as $d \ge 2$, the product $\prod_{0 \le i < j \le d}(y_j - y_i)$ is not invertible. Therefore, Lagrange interpolation is not applicable as usual on this particular ring.

Thus, to solve the inversion problem, we need to either:

- adapt Lagrange interpolation to computations in the ring $\mathbb{Z}_{2^n}$,
- use another algorithm better suited to the ring $\mathbb{Z}_{2^n}$ to solve the inversion problem.

Our first intuition was to adapt Lagrange interpolation to the ring $\mathbb{Z}_{2^n}$ by using a ring extension to solve the issue of inverting the value $z$. The idea was that Lagrange interpolation was prevented by the fact that our ring contained to many zero-divisors (i.e. elements that do not possess an inverse). By using a ring extension on the ring $\mathbb{Z}_{2^n}$, we showed in article [BERR16] that it was possible to increase the ratio of invertible elements, allowing the computation of Lagrange interpolation. However, this method had two major drawbacks:

- its complexity was far superior to a regular interpolation,
- while it successfully invert more polynomials than the subset presented in [ZMGJ07], it still failed to invert all permutation polynomials as defined in

[Riv01]. In particular, binary permutation polynomials with odd coefficients, which happen to be the best candidates to resist the attack described in [BJLS17].

In the following section, we will define two method to solve the inversion problem in the ring $\mathbb{Z}_{2^n}$. One adapts Newton's method for the computation of compositional inverses as presented in [Bos10] and [BZ10] to our problem of inverting binary permutation polynomials in $\mathbb{Z}_{2^n}$. The other is a new variation of Lagrange interpolation specifically designed to avoid the computation of $z$.

## 4.2 Inverting all Polynomial Bijections Modulo $2^n$

In this section, we will use a polynomial ideal to give a new definition of permutation polynomials. This definition is equivalent to the one described by Rivest [Riv01] rather than the one described in [ZMGJ07]. We will then propose a simple method for inverting binary permutation polynomials based on Newton's method (details on newton's method for polynomial inverse can be found in [Bos10] and [BZ10]). In this section, we will not go over our initial attempt at Lagrange interpolation based inversion method described in [BERR16], instead we will focus on Newton's method for permutation polynomials inversion, as Lagrange interpolation will be covered in section 4.3.

### 4.2.1 The Zero Ideal of Binary Permutation Polynomials

#### On Polynomial Ideals

Understanding the inversion problem for binary permutation polynomials requires to focus on the problem of *polynomial simplification*, meaning that for a given polynomial $f$, one asks for a polynomial $g$ (if it exists) verifying:

$$g(x) = f(x) \mod 2^n \quad \forall x \in \mathbb{Z}_{2^n} \text{ with } \deg(g) < deg(f)$$

In [SM10], a method for polynomial simplification using a polynomial ideal is presented: in this section, we adapt their method to our case. An ideal of $\mathbb{Z}_{2^n}[x]$ generated by the finite sequence of polynomials $\{F_0, F_1, \ldots, F_s\}$ is the set of all its $\mathbb{Z}_{2^n}[x]$-linear combinations:

$$\langle F_0, \ldots, F_s \rangle = \{\sum_{i=0}^{s} G_i F_i : G_i \in \mathbb{Z}_{2^n}[x]\}$$

#### The Zero Ideal

The method presented in [SM10] is defined in the ring of polynomials with coefficients in $\mathbb{Z}^*_{2^n}$, the ring of inversible elements in $\mathbb{Z}_{2^n}$. The authors define an ideal $I^*$ generated by a finite set $\{P^*_1, \ldots, P^*_{d^*_{n+1}}\}$ of polynomials which evaluate globally to 0 as a function taking values in $\mathbb{Z}^*_{2^n}$. Thus the ideal $I^*$ verifies:

$$P \in I^* \iff P(x) = 0 \mod 2^n \qquad \forall x \in \mathbb{Z}^*_{2^n}$$

The authors define this set of polynomials in the following way (we choose here to use positive roots):

$$P^*_i(x) = 2^{n-i-t_i} \prod_{j=0}^{i} (x - 2j - 1)$$

where $t_i$ is the largest integer $\ell$ such that $2\ell$ divides $i!$. The integer $d_n^*$ is defined as the largest integer $i$ such that $n - i - t_i$ is positive. In particular, $P_0^* = 2^n$ and $P^*_{d^*_{n+1}} = (x-1)(x-3)\cdots(x-2d_n^*+1)$.

Let us recall that, in a field, for a variable $a$ to be root of a polynomial, $(x-a)$ must divide that polynomial. However, in a general ring, zero-divisors provide an alternative way of producing 0.

When $x$ is an odd integer, the value $\prod_{j=1}^{i}(x-2j-1)$ is a product of $i$ consecutive even numbers. Therefore, as explained in [SM10], it is divisible by $2^i i!$ and so it is divisible by $2^{i+ti}$. Multiplying this polynomial by $2^{n-i-t_i}$ results in $2^n$ which is equal to zero in $\mathbb{Z}_{2^n}$.

From The $P_i^*$ generating the ideal $I^*$ on $\mathbb{Z}^*_{2^n}$, we deduce the set of polynomials $P_i$ generating the ideal $I$ in $\mathbb{Z}_{2^n}[x]$ of all the polynomials evaluating to 0 for any value of $x$ in $\mathbb{Z}_{2^n}$, i.e. $I$ is the *zero ideal* of $\mathbb{Z}_{2^n}[x]$ (see Proposition 1). The definition of $P_i$ is derived from $P_i^*$, by considering even roots, so that if $x$ is even, it still induces a product of $i$ consecutive even numbers:

$$P_i = 2^{n-i-t_i}\prod_{j=0}^{2i-1}(x-j)$$

Note that $P_0$ is equal to $2^n$ and the degree of $P_i$ is increasing with $i$. The index of the polynomial $P_i$ with highest degree is defined as follows. Let $i_{max}$ be the greatest integer $\ell$ such that $n-(\ell-1)-t_{\ell-1} > 0$. Two cases are possible, i.e $n - i_{max} - t_{i_{max}}$ equals 0 or is negative, but the polynomial $P_{i_{max}}$ is always defined as

$$P_{i_{max}} = \prod_{j=0}^{2i_{max}-1}(x-j)$$

**Polynomial Simplification**

The generating set $\{P_o, \ldots, P_{i_{max}}\}$ of the zero ideal $I$ provides an algorithmic method to compute in $\mathbb{Z}_{2^n}[x]$ modulo $I$. In particular, for a given $f$, this let us compute a polynomial $g$ of smallest degree in the class $\{f+h \;:\; h \in I\}$ of $f$ modulo $I$. Such a computation is called *polynomial simplification*. As stated in Proposition 1, any polynomial $g$ in the class of $f$ will define the same function on $\mathbb{Z}_{2^n}$, thus the simplification procedure let us define a binary permutation polynomial with the smallest polynomial as possible.

We now describe this simplification method. First, note that for all $i$, the polynomial $P_i$ is the product of a monic polynomial (with a leading coefficient equals to 1) and a constant equal to a power of 2 (which we will refer as $exp_i$ in the sequel). Moreover, the more $i$ increases (therefore the degree of $P_i$), the smaller $exp_i$ gets. From these remarks, one can simplify a given permutation polynomial $f$ by using the following procedure. If $exp_i$ divides the leading coefficient $a_d$ of $f(x) = \sum_{i=0}^{d} a_i x^i$ and $deg(P_i) < d$, let us define $g$ the permutation interpreted as the polynomial:

$$g(x) = f(x) - (\frac{a_d}{exp_i})(x^{d-deg(P_i)})P_i$$

Then $g$ is in the same class as $f$ modulo $I$ and $deg(g) < d$ (i.e $g$ is a reduction of $f$ modulo $I$).

The process continues with the new polynomial $g$ while the conditions on the leading coefficient and the degree are true. At the end, one obtains the reduction of $f$ modulo $I$ with the smallest degree.

To resume the previous discussion, the permutation polynomial set $\mathcal{P}(\mathbb{Z}_{2^n})$ is now seen as a quotient sub-group of $\mathbb{Z}_{2^n}[X]/I$ as described in (label section). For each class in $\mathcal{P}(\mathbb{Z}_{2^n})$, one can choose, as a representative, a polynomial of degree smaller than the ones of the $P_i$. Hence, these representatives have a degree smaller than $d_n = 2 \times i_{max}$ the degree of $P_{i_{max}}$ and thus, the set $\mathcal{P}(\mathbb{Z}_{2^n})$ is finite.

We then deduce the following central result.

**Theorem 4** *The ideal $I = \langle P_0, \dots, P_{i_{max}} \rangle$ verifies*

$$\textit{for } f, g \in \mathbb{Z}_{2^n}[X] \textit{ and } h \in I$$
$$f = g - h \iff f(x) = g(x) \mod 2^n \quad \forall x \in \mathbb{Z}_{2^n}$$

*Moreover any binary permutation polynomial can be equally represented by a polynomial of degree at most $d_n = Deg(P_{i_{max}})$.*

### Identity Polynomials in $\mathbb{Z}_{2^n}$

Simplification is not the only result induced by the zero ideal. Let us define the set $I$:

$$I = \{x + h(x)\} \quad \forall h \in I$$

This set defines a class of polynomials which are, as functions, equivalent to $X$. Therefore, for a given polynomial $f$, if there exists $g$ such as $g \circ f \in I$, $g$ is equivalent to $f^{-1}$.

This is an important result as it hints the reason why standard Lagrange interpolation struggles to invert some polynomials. Lagrange interpolation searches for a polynomial verifying $g(f(x)) = x \mod 2^n$, but as we shown, there are other choices producing a compositional inverse. That is to say, $x$ (the polynomial) is not the only permutation polynomial defining $X$ (the function). This means that:

$$g(f(x)) = x \mod 2^n \quad \forall x \in \mathbb{Z}_{2^n} \not\Leftrightarrow g \circ f = X$$

Let $g$ be a polynomial such as $g(f(x)) = x + h(x)$ with $h \in I$, we witnessed, through experimentation, that replacing $x$ in the Lagrange formula by $x + h(x)$ produced a valid $g$. The problem with such an approach is that, unlike $\mathcal{P}(\mathbb{Z}_{2^n})$, $I$ is not finite. Therefore predicting $h$ before interpolation did not seem reliable in practice.

However, other inversion methods do not present such restrictions. Newton's method for compositional inversion is one of them.

## 4.2.2 Newton's Method for Inversion of Binary Permutation Polynomials

In this section we present an adaption for the binary permutation polynomials of the Newton's method for the computation of the compositional inverse as presented in [Bos10] and [BZ10].

### Iterative Inversion in Fields

Newton's method (also known as Newton-Raphson method) aims to find successively better approximations of the roots of a function $f$ on some set $E$. It is based on the assumption that, near a point $x_0$ in the vicinity of a root, a function's behavior is almost that of its tangent line:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) \quad \forall x \in E$$

Therefore, one can approximate a better root by finding $x$ such that:

$$\begin{aligned} f(x_0) + f'(x_0)(x - x_0) &= 0 \\ x &= x_0 - \frac{f(x_0)}{f'(x_0)} \end{aligned}$$

As long as $f$ is derivable and that its derivative evaluation in some $x_i$ is invertible, one can define the following iteration:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Where $x_{i+1}$ is a better root approximation than $x_i$. Therefore, if a root exists, the algorithm should converge to it (they are some exceptions in which the algorithm is "trapped" in a cycle, but we never encountered this problem while testing this algorithm in $\mathbb{Z}_{2^n}$).

This method can be adapted for other purposes, for example:

$$x_{i+1} = x_i - \frac{f(x_i) - 7}{f'(x_i)}$$

should converge to $x$ such as $f(x) = 7$.

The formula linked to our problematic is that of compositional inverse:

$$g_{i+1} = g_i - \frac{f \circ (g_i - X)}{f' \circ g_i}$$

The main problem with this formula is the arithmetic inversion of $f'(g_i)$. But it is possible to use the following simplification:

$$\begin{aligned} f \circ g &= X \\ (f \circ g)' = (f' \circ g) \times g' &= 1 \\ g' &= \frac{1}{f' \circ g} \\ g_{i+1} &= g_i - g_i' \times (f \circ (g_i - X)) \end{aligned}$$

This method was defined on a field, and in the following section we address the subject of its use in $\mathbb{Z}_{2^n}$.

### The Case of Binary Permutation Polynomials

In order to assess the correctness of the method in $\mathbb{Z}_{2^n}$, we use the two properties for Newton's method to correctly compute a compositional inverse (as presented in [Bos10] and [BZ10]:

$$\begin{aligned} f(0) &= 0 \\ f'(0) &\text{ is invertible} \end{aligned}$$

The second condition is respected as $f'(0) = a_1$ which is always invertible, regardless of the polynomial we study.

In order to check for the first property, let us consider the polynomial:

$$r(x) = \sum_{i=1}^{d} a_i x^i$$

we have $f = r + a_0$, meaning $f$ produces the same permutation as $r$ with an additional shift. Therefore, if we invert $r$, we have the following equality:

$$\begin{aligned} r^{-1} \circ f = f \circ r^{-1} = a_0 + r \circ r^{-1} &= a_0 + X \\ (r^{-1} \circ f) - a_0 &= X \end{aligned}$$

The polynomial $r(x)$ having no constant coefficient, this means $r^{-1} - a_0$ is inverse of $f$. However, $r(0) = 0$, thus $r$ is invertible through Newton's method (in practice, Newton's method in $\mathbb{Z}_{2^n}$ converges while $f(0) \neq 0$ most of the time).

Therefore, if there exists $g$ such as $g \circ f = X$, Newton's method should converge to $g$. We tried to use this method on the set of polynomials which were invertible through Lagrange interpolation and the method succeeded every time.

However, the full set of permutation polynomials as characterized in [Riv01] still remains out of reach for Newton's method as it is. But, as mentioned in Section 4.2.1, this results from the fact that:

$$g(f(x)) = x \mod 2^n \quad \forall x \in \mathbb{Z}_{2^n} \;\not\Longleftrightarrow\; g \circ f = x$$

Let us take a second look at the formula presented in the previous section:

$$g_{i+1} = g_i - \frac{f \circ (g_i - X)}{f' \circ g_i}$$

The problem is that $x$ is not the only polynomial equivalent to the identity function $X$ modulo the zero ideal $I$. Thus if $g(f(x)) = x + h(x)$ with $h \in I$ not equal to 0, this formula would become:

$$g_{i+1} = g_i - \frac{f \circ (g_i - x - h(x))}{f' \circ g_i}$$

However, we will see in the next section an alternative way of computing on reduced polynomials, which does not assume the knowledge of $h$.

#### Inverting all Polynomials Defined by Rivest

One can compute Newton's method in the ring of reduced polynomials $\mathcal{F}(\mathbb{Z}_{2^n})$ where:

$$g(f(x)) = x \mod 2^n \quad \forall x \in \mathbb{Z}_{2^n} \iff g(f) = x$$

Using Newton's method in the ring of reduced polynomials consists in reducing modulo $I$ the result of each iteration (see Section 4.2.1). Then, during the computation, if a polynomial $g$ equivalent to $x$ appears it would be reduced to $x$. Thus, the problem emphasized above is now solved.

We tried to inverse all the permutation polynomials as characterized in [Riv01]. Not only does it succeed in inverting all the polynomials, but the fact that the degrees of the reduced polynomials are bounded prevents unnecessary increase of the size of the objects during composition and multiplication.

As far as computation goes, each iteration requires a composition and a multiplication (along with some additions and subtractions). In theory, Newton's method converge to a solution with quadratic speed (see [Bos10] or [BZ10]). Meaning to inverse a polynomial, the number of iterations required would be squared in the degree of its inverse. This degree being bounded by $n$, we get at most $\sqrt{n}$ iterations.

Considering a quadratic cost for polynomial multiplication and composition, an overall complexity of $O(d_n^2 \times \sqrt{n})$ is achieved, where $d_n$ is a bound on the degree of a reduced polynomial (see Section 4.2.1).

Note that the cost of polynomial reduction does not change this complexity estimation. Actually, if we consider consider the product of two polynomials of degree $d_{max}$, the resulting polynomial before reduction is then of degree $2 \times d_{max}$. Reducing that polynomial to a polynomial of degree $d_{max}$ requires $d_{max}$ polynomial subtractions, and therefore presents a complexity of $O(d_{max}^2)$. On the same principle, further reduction from this polynomial of degree $d_{max}$ to a polynomial of smaller degree will at most be quadratic in the degree, which is the same complexity as the other computations performed (multiplications and compositions) during each iterations of Newton's inversion.

**On the degree of the inverse**

An interesting result which could follow this study is the relation between the degree of a permutation polynomial and the degree of its minimal inverse (let us recall that in contrary to the set defined in [ZMGJ07], the degree of a permutation polynomial is not necessary the same as its inverse).

Because the polynomial set defined in [ZMGJ07] does not present such a problem, our hypothesis is that this difference in degree is tight to the difference between this set and the full characterization given in [Riv01].

Permutation polynomials presented in [ZMGJ07] are polynomials $\sum_i a_i x^i$ such that:

$$2^{\frac{n}{2}} | a_i \qquad \forall i \geq 2$$

Therefore each coefficient $a_i$ with $i \geq 2$ has a very high 2-adic valuation.

The set of all the permutation polynomials as characterized by Rivest does not present such a restriction: it includes polynomials with coefficients of small 2-adic valuation. In the opposite, those polynomials results in high degree differences between a polynomial and its inverse. We leave these observations as an open problem that will be studied in a future work.

## 4.3 Updated Lagrange Interpolation for Binary Permutation Polynomials

This section covers our following work on a Lagrange interpolation method for inverting binary permutation polynomials, this work has been presented in the following article [BKRS18]. While Newton's method for inverting binary permutation polynomials is an effective algorithm, we present in this section a new technique based on Lagrange interpolation with two important properties:

- In a designer point of view, it is very important to measure the strength of any obfuscation technique based on binary permutation polynomials. The interpolation algorithm analyzed in this paper is proven to have a fixed complexity. This provides a more precise framework when measuring attack complexities with regard to computational overhead of using a binary permutation polynomial.
- From the reverse engineering point of view, this algorithm enables inversion techniques in a black-box context (i.e. when an encoding function is given as an evaluation function only). This is of importance when considering the reliance of encodings based on binary permutation polynomials since this algorithm can retrieve the explicit function through interpolation.

In addition, our version of Lagrange interpolation allows a better understanding on how to use binary permutation polynomials. This should prove useful for future work on the subject.

### 4.3.1 A second look on Binary Permutation Polynomials

Let us recall the characterization of binary permutation polynomials in terms of the coefficients of the polynomial as described in the article [Riv01]. Namely, a polynomial $P(x) = a_0 + \cdots + a_m x^m \in \mathbb{Z}[x]$ induces a permutation on $\mathbb{Z}_{2^n}$ if and only if (i) $a_1$ is odd, (ii) the sum $a_3 + a_5 + a_7 \ldots$ is even, and (iii) the sum $a_2 + a_4 + a_6 + \ldots$ is even.

The criterion is stated and proved in this form by Rivest [Riv01], but he points out that it also follows easily from the following more general criterion: $P(x)$ induces a permutation on $\mathbb{Z}_{p^n}$, where $p$ is a prime and $n \geq 2$, if and only if (i) $P(x)$ induces a permutation on $\mathbb{Z}_p$ and (ii) $P'(a) \not\equiv 0 \pmod{p}$ for $a \in \mathbb{Z}$. The last criterion is stated in the work of Mullen and Stevens [MS84], who consider it a direct corollary of Theorem 123. in the book by Hardy and Wright [HW95].

From this criterion, we get the following corollary, crucial for our purposes.

**Corollary 1** *Let $P(x) = a_0 + \cdots + a_m x^m \in \mathbb{Z}[x]$ induce a permutation on $\mathbb{Z}_{2^n}$. For all $a, b \in R$, with $a \neq b$, the Newton quotient $k_{a,b} = \frac{P(a)-P(b)}{a-b}$ is an odd integer.*

**Proof 1** *Indeed, $k_{a,b} = a_1 A_1 + a_2 A_2 + \cdots + a_m A_m$, where $A_1 = 1$ and, for $i = 2, 3, \ldots, m$, $A_i = a^{i-1} + a^{i-2}b + \cdots + ab^{i-2} + b^{i-1}$. If both $a$ and $b$ are even then, modulo 2, $k_{a,b} \equiv a_1$, if they have different parity then $k_{a,b} \equiv a_1 + a_2 + \cdots + a_m$, and if they are both odd, $k_{a,b} \equiv a_1 + a_3 + a_5 + \ldots$. Thus, $k_{a,b}$ is always odd.*

This will be important to prove the invertibility of some elements in our updated Lagrange interpolation algorithm presented in the next section.

Additionally, let us define the following theorem:

**Theorem 5** *Let $P(x) \in \mathbb{Z}[x]$ be a polynomial that induces a permutation on $\mathbb{Z}_{2^n}$, given by its sequence of values $P(0), \ldots, P(d_n)$ in $\mathbb{Z}_{2^n}$. There exists an algorithm of time complexity $O(n^2)$ that determines the sequence of coefficients $b_0, b_1, \ldots, b_{d_n}$ of the unique reduced polynomial $Q(x)$ that induces the inverse permutation to $P(x)$ on $\mathbb{Z}_{2^n}$.*

### 4.3.2 Lagrange Interpolation : Solving the associated linear system

Going back to our initial problem of the inversion of permutation polynomials, let us fix $n$, and to simplify notation, set $d = d_n$.

For $i = 0, \ldots, d$, set $x_i = P(i)$ and $y_i = Q(x_i) = i$. We need to solve, over $R_n$, the linear system of equations

$$V[x_0, x_1, \ldots, x_d](b_0, b_1, \ldots, b_d)^T = (y_0, y_1, \ldots, y_d)^T,$$

where $V = V[x_0, x_1, \ldots, x_d] = [v_{i,j}]_{(d+1)\times(d+1)}$ is the $(d+1) \times (d+1)$ Vandermonde matrix in which $v_{i,j} = x_i^j$.

We will use the following two results by Oruç and Phillips.

**Theorem 6 (Oruç-Phillips 2000 [OP00])** *Let $x_0, x_1, \ldots, x_m$ be distinct. An explicit LDU decomposition of the Vandermonde matrix $V = V[x_0, x_1, \ldots, x_m]$ is given by $V = LDU$, where $D$ is the diagonal matrix*

$$Diag(1,\ x_1 - x_0,\ (x_2 - x_1)(x_2 - x_0), \ldots,\ (x_m - x_{m-1})(x_m - x_{m-2}) \ldots (x_m - x_0)),$$

*$L$ is the lower triangular matrix $L = [\ell_{i,j}]$ given by*

$$\ell_{i,j} = \prod_{t=0}^{j-1} \frac{x_i - x_{j-1-t}}{x_j - x_{j-1-t}}, \qquad 0 \le j \le i \le m,$$

*and $U$ is the upper triangular matrix $U = [u_{i,j}]$ given by*

$$u_{i,j} = \tau_{j-i}(x_0, \ldots, x_i) \qquad 0 \le i \le j \le m,$$

*with the understanding that empty products are equal to 1 (thus all diagonal entries in both $L$ and $U$ are equal to 1), and $\tau_r(x_0, \ldots, x_i)$ is the complete symmetric function evaluated at $x_0, \ldots, x_i$, that is,*

$$\tau_r(x_0, \ldots, x_i) = \sum_{\lambda_0 + \lambda_1 + \cdots + \lambda_i = r} x_0^{\lambda_0} x_1^{\lambda_1} \ldots x_i^{\lambda_i}.$$

For $m = 4$, we have

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & \frac{x_2 - x_0}{x_1 - x_0} & 1 & 0 & 0 \\ 1 & \frac{x_3 - x_0}{x_1 - x_0} & \frac{(x_3 - x_1)(x_3 - x_0)}{(x_2 - x_1)(x_2 - x_0)} & 1 & 0 \\ 1 & \frac{x_4 - x_0}{x_1 - x_0} & \frac{(x_4 - x_1)(x_4 - x_0)}{(x_2 - x_1)(x_2 - x_0)} & \frac{(x_4 - x_2)(x_4 - x_1)(x_4 - x_0)}{(x_3 - x_2)(x_3 - x_1)(x_3 - x_0)} & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} 1 & x_0 & x_0^2 & x_0^3 & x_0^4 \\ 0 & 1 & x_0 + x_1 & x_0^2 + x_0 x_1 + x_1^2 & x_0^3 + x_0^2 x_1 + x_0 x_1^2 + x_1^3 \\ 0 & 0 & 1 & x_0 + x_1 + x_2 & x_0^2 + x_0 x_1 + x_1^2 + x_0 x_2 + x_1 x_2 + x_2^2 \\ 0 & 0 & 0 & 1 & x_0 + x_1 + x_2 + x_3 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The entries of $U$ can be obtained recursively, by $u_{0,j} = x_0^j$, $u_{i,i} = 1$, and

$$u_{i,j} = u_{i-1,j-1} + u_{i,j-1} \cdot x_i, \qquad \text{for } 1 \le i < j. \tag{4.3}$$

**Theorem 7 (Oruç-Phillips 2000 [OP00])** *Let $x_0, x_1, \ldots, x_m$ be distinct. The matrix $L$ from the explicit LDU decomposition of the Vandermonde matrix $V = V[x_0, x_1, \ldots, x_m]$*

*given in Theorem 6 decomposes as the product*

$$L = L^{(1)} L^{(2)} \dots L^{(m)}$$

*of subdiagonal* $(m+1) \times (m+1)$ *matrices* $L^{(k)} = [\ell^{(k)}_{i,j}]$ *with 1s on the diagonal and the subdiagonal entries given, for* $j = 0, \dots, m-1$, *by*

$$\ell^{(k)}_{j+1,j} = \begin{cases} 0, & 0 \le j < m-k, \\ \displaystyle\prod_{t=0}^{j-(m-k)-1} \frac{x_{j+1} - x_{j-t}}{x_j - x_{j-1-t}}, & m-k \le j \le m. \end{cases}$$

For $m = 4$, the following table provides the subdiagonal entries:

| $j$ | $\ell^{(1)}_{j+1,j}$ | $\ell^{(2)}_{j+1,j}$ | $\ell^{(3)}_{j+1,j}$ | $\ell^{(4)}_{j+1,j}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | $\frac{x_2-x_1}{x_1-x_0}$ |
| 2 | 0 | 1 | $\frac{x_3-x_2}{x_2-x_1}$ | $\frac{(x_3-x_2)(x_3-x_1)}{(x_2-x_1)(x_2-x_0)}$ |
| 3 | 1 | $\frac{x_4-x_3}{x_3-x_2}$ | $\frac{(x_4-x_3)(x_4-x_2)}{(x_3-x_2)(x_3-x_1)}$ | $\frac{(x_4-x_3)(x_4-x_2)(x_4-x_1)}{(x_3-x_2)(x_3-x_1)(x_3-x_0)}$ |

The subdiagonal entries can be calculated recursively as follows. For fixed $j$ and $k = m - j$, we have $\ell^{(k)}_{j+1,j} = 1$, and for $k > m - j + 1$,

$$\ell^{(k)}_{j+1,j} = \ell^{(k-1)}_{j+1,j} \cdot \frac{x_{j+1} - x_{m-k+1}}{x_j - x_{m-k}}. \tag{4.4}$$

Going back to our situation, we see that the entries of $U$ and $D$ are integers and, as such, are well defined over $\mathbb{Z}_{2^n}$. The entries of $L$ and $L^{(k)}$ are not necessarily integers, but they are still well defined over $\mathbb{Z}_{2^n}$.

**Proposition 2** *Let* $P(x) \in \mathbb{Z}_{2^n}[x]$ *induce a permutation on* $\mathbb{Z}_{2^n}$.
*(a) Each entry of* $L$ *is has odd denominator in its simplest form.*
*(b) Each entry of* $L^{(k)}$, *for* $k = 1, \dots, d$ *has odd numerator and odd denominator in its simplest form.*

**Proof 2** *(a) By Corollary 1, we have, for* $0 \le j \le i \le d$,

$$\ell_{i,j} = \prod_{t=0}^{j-1} \frac{x_i - x_{j-1-t}}{x_j - x_{j-1-t}} = \prod_{t=0}^{j-1} \frac{P(y_i) - P(y_{j-1-t})}{P(y_j) - P(y_{j-1-t})} = \prod_{t=0}^{j-1} \frac{(y_i - y_{j-1-t})k_{i,j-1-t}}{(y_j - y_{j-1-t})k_{j,j-1-t}}$$
$$= \prod_{t=0}^{j-1} \frac{(i-j+1+t)k_{i,j-1-t}}{(1+t)k_{j,j-1-t}} = \binom{i}{j} \prod_{t=0}^{j-1} \frac{k_{i,j-1-t}}{k_{j,j-1-t}},$$

*where each* $k_{*,*}$ *is an odd integer.*
*(b) Let* $d - k \le j \le d$ *and set* $s = j - (d-k) - 1$. *By Corollary 1, we have,*

$$\ell^{(k)}_{j+1,j} = \prod_{t=0}^{s} \frac{x_{j+1} - x_{j-t}}{x_j - x_{j-1-t}} = \prod_{t=0}^{s} \frac{P(y_{j+1}) - P(y_{j-t})}{P(y_j) - P(y_{j-1-t})} = \prod_{t=0}^{s} \frac{(y_{j+1} - y_{j-t})k_{j+1,j-t}}{(y_j - y_{j-1-t})k_{j,j-1-t}}$$
$$= \prod_{t=0}^{s} \frac{(1+t)k_{j+1,j-t}}{(1+t)k_{j,j-1-t}} = \prod_{t=0}^{s} \frac{k_{j+1,j-t}}{k_{j,j-1-t}},$$

*where each $k_{*,*}$ is an odd integer.*

We are ready to prove the main result.

**Proof 3 (Proof of Theorem 5)** *Recall that, in our situation, $m = d = d_n < n + \log_2 n$ and we are solving the system $LDU\mathbf{b} = \mathbf{y}$. The recursive formulas (4.3) and (4.4) show that the entries of $U$ and the subdiagonal entries in all $L^{(k)}$, $k = 1, \dots, d$, can be calculated in $O(n^2)$ steps. The diagonal entries of $D$ can also be calculated recursively in $O(n^2)$ steps. The inverse of $L^{(k)}$ is obtained by simply changing the sign in all subdiagonal entries. Therefore, we can calculate $\mathbf{y}' = L^{-1}\mathbf{y} = {L^{(m)}}^{-1}{L^{(m-1)}}^{-1} \dots {L^{(1)}}^{-1}\mathbf{y}$ in $O(n^2)$ steps.*

*We then solve the system $DU\mathbf{b} = \mathbf{y}'$ by backward substitution in $O(n^2)$ steps. Note that the i-entry of $D$ has the form $2^{t_i}f_i$, where $f_i$ is odd. Because of our constraints on the coefficients of reduced polynomials, we are seeking only for solutions for $b_i$ in the range $0 \le b_i < 2^{n-t_i}$, and a solution exists and is unique in this range. More precisely, once $b_{i+1}, \dots, b_d$ are substituted in, we need to solve for $b_i$ from an equation of the form $2^{t_i}f_ib_i = g_i \pmod{2^n}$, for some odd $f_i$ and some $g_i \in \mathbb{Z}_{2^n}$. We already know that a solution exists, so it must be that $g_i = 2^{t_i}g_i'$ for some $g_i' \in Rn$. After canceling the term $2^{t_i}$ we solve for $b_i$ from $f_ib_i = g_i' \pmod{2^{n-t_i}}$ by inverting $f_i$, and thus produce the unique solution in the range $0 \le b_i < 2^{n-t_i}$.*

### 4.3.3 Another solution

If we are not interested in producing the coefficients of the inverse polynomial $Q(x)$, but rather just in calculating the values of $Q(x)$ at various points, a slightly different algorithm exists and we outline it here.

Let $\mathcal{U}_n$ be the ring of units of the ring $\mathbb{Z}_{2^n}$. Without loss of generality we may assume that $P(x)$ separately permutes $\mathcal{U}_n$, the odds, and its complement, the evens (if it does not, we may replace $P(x)$ by $P(x) + 1$).

The ideal $I'$ of integer polynomials that induce the zero function on $\mathcal{U}_n$ is described in section 4.2.1. It is generated by $P_i(x) = 2^{n-i-t_i}\prod_{j=0}^{i-1}(x - (2j+1))$, for $i = 0, 1, \dots, d_n'$, and $P_{d_n'+1}(x) = \prod_{j=0}^{d_n'}(x - (2j+1))$, where $d_n'$ is the largest integer $i$ such that $n - i - t_i > 0$. Every integer polynomial that permutes $\mathcal{U}_n$ has a unique representative modulo $I'$, which is a polynomial of degree at most $d_n'$ with the $i$-coefficient in the range from 0 to $2^{n-i-t_i} - 1$. The maximum degree $d_n'$ is approximately half of $d_n$. We may calculate, by using the same approach as above (the Vandermonde matrix will have dimension $(d_n' + 1) \times (d_n' + 1)$ and the interpolation is preformed for $x_i = P(2i + 1)$, $i = 0, \dots, d_n'$) the coefficients of the unique reduced polynomial $\overline{Q}(x)$ modulo $I'$ that inverts the values of $P(x)$ on $\mathcal{U}_n$ (and not necessarily on its complement).

By a similar approach, the coefficients of another polynomial, $\overline{\overline{Q}}(x)$, of degree at most $d_n'$ that inverts the values of $P(x)$ on the complement of $\mathcal{U}_n$ may be calculated. The two polynomials $\overline{Q}(x)$ and $\overline{\overline{Q}}(x)$ may then be used to calculate the values of $Q(x)$ (use the former for odd $x$ and the latter for even). Since the degrees of $\overline{Q}(x)$ and $\overline{\overline{Q}}(x)$ are, in general, smaller than the degree of $Q(x)$, this approach may be faster if we need to calculate many values of $Q(x)$.

### 4.3.4 Interpolation of the inverse polynomial over $\mathbb{Z}_{p^n}$

Fix a prime $p$ and $n > 1$.

We claim that the same inversion technique works equally well for permutation polynomials over the ring $\mathbb{Z}_{p^n}$.

The ideal of integer polynomials that induce the zero function on $\mathbb{Z}_{p^n}$ is generated by the polynomials

$$P_i(x) = p^{n-t_{p,i}} \prod_{j=0}^{i-1}(x-j) \text{ for } i = 0,1,\dots,d_{p,n}, \text{ and } P_{d_{p,n}+1}(x) = \prod_{j=0}^{d_{p,n}}(x-j).$$

where, for $i \geq 0$, $t_{p,i}$ is the largest integer $\ell$ such that $p^\ell$ divides $\ell!$, and $d_{p,n}$ is the the largest integer $i$ such that $n - t_{p,i} > 0$. Each integer polynomial is equivalent, as a function over $\mathbb{Z}_{p^n}$, to a unique reduced polynomial, that is, polynomial $b_0 + b_1 x + \cdots + b_{d_{p,n}} x^{d_{p,n}}$ of degree at most $d_{p,n}$, such that, for $i = 0, \dots, d_{p,n}$, we have $0 \leq b_i < p^{n-t_{p,i}}$ (see [MS84, Theorem 2.1]).

We prove an analog of Corollary 1.

**Proposition 3** *Let $P(x) = a_0 + \cdots + a_m x^m \in \mathbb{Z}[x]$ induce a permutation on $\mathbb{Z}_{p^n}$. For all $a, b \in \mathbb{Z}$, with $a \neq b$, the Newton quotient $k_{a,b} = \frac{P(a)-P(b)}{a-b}$ is an integer that is not divisible by $p$.*

**Proof 4** *Since $P(x)$ induces a permutation on $\mathbb{Z}_{p^n}$, it induces a permutation on $\mathbb{Z}_p$ and $P'(a) \not\equiv 0 \pmod{p}$, for all $a$.*

*We work modulo $p$.*

*Let $a \neq b$ and, moreover, $a - b \not\equiv 0$. If we assume $k_{a,b} \equiv 0$, then $P(a) - P(b) \equiv (a-b)k_{a,b} \equiv 0$, a contradiction, since $P(x)$ permutes $\mathbb{Z}_p$.*

*Let $a \neq b$, but $a \equiv b$. Then, for $i \geq 2$, we have $A_i = a^{i-1} + a^{i-2}b + \cdots + ab^{i-2} + b^{i-1} \equiv ia^{i-1}$ and $k_{a,b} \equiv a_1 + 2a_2 a + \cdots + m a_m a^{m-1} \equiv P'(a) \not\equiv 0$.*

The rest of the proof is exactly the same as in the binary case, except, of course, that the analog of Proposition 2 should state that, in their simplest form, all denominators of the entries in $L$ are integers not divisible by $p$, and all numerators and denominators of the entries in $L^{(k)}$, $k = 1, \dots, d_{p,n}$, are integers not divisible by $p$. Thus, $L$ and $L^{(k)}$ are well defined over $\mathbb{Z}_{p^n}$.

Thus we may state a more general version of our main result.

**Theorem 8** *Let $p$ be a prime and $P(x) \in \mathbb{Z}[x]$ a polynomial that induces a permutation on $\mathbb{Z}_{p^n}$, given by its sequence of values $P(a), P(a+1), \dots, P(a+d_{p,n})$ in $\mathbb{Z}_{p^n}$ for some $a \in \mathbb{Z}_{p^n}$ (not necessarily 0). There exists an algorithm of time complexity $O(n^2)$ that determines the sequence of coefficients $b_0, b_1, \dots, b_{d_{p,n}}$ of the unique reduced polynomial $Q(x)$ that induces the inverse permutation to $P(x)$ on $\mathbb{Z}_{p^n}$.*

### 4.3.5 Lagrange Interpolation Method and Reverse Engineering of Binary Permutation Polynomials Based Obfuscation

In addition to being better suited for complexity measurement, Lagrange interpolation hold another distinct advantage compared to Newton's method for an attacker. When considering Newton's method for computing inverse permutation polynomial, one must keep in mind that the algorithm takes the forward permutation polynomial as input. This means an attacker that seeks to obtain the inverse permutation polynomial must first seek to extract the forward permutation polynomial from its MBA form. As it happens, the attack presented in [BJLS17] does just that. However, we exposed a method to inverse a new set of permutation polynomials, including permutation polynomials that resist this attack. Therefore an attacker would need to come up with a new attack to break the MBA obfuscation before being able to use Newton's method to compute the inverse polynomial.

On the other hand, Lagrange interpolation only require points and their respective image after permutation. Meaning an attacker may consider the MBA obfuscation as a black box providing inputs/outputs that can be used to interpolate. However, in order to properly use our updated Lagrange interpolation for polynomial elements of the ring $\mathbb{Z}_{2^n}$, an attacker would need to find consecutive images. In other words, a set of points $(x_0, x_1, \ldots, x_k)$ such that:

$$y_i = f(x_i)$$

$$y_{i+1} = y_i + 1$$

Considering the ring $\mathbb{Z}_{2^n}$ holds $2^n$ elements, Let us consider the probability of finding $k$ elements upholding this property. Let us assume the attacker has no knowledge of which permutation polynomial was used in the process and regard the MBA as a black-box. Randomly selecting inputs $(x_0, x_1, \ldots, x_k)$ would yield a favorable outcome with the following probability:

$$P = (\frac{k}{2^n})(\frac{k-1}{2^n})(\ldots)(\frac{1}{2^n})$$

While this might be doable for certain values of $n$ and $k$, this also assume the MBA always output the result of the forward permutation polynomials for all input values. This could be tampered with Boolean formula that output non-zero values for specific outputs.

In conclusion, Lagrange interpolation could serve as a tool for inverting permutation polynomials based MBA. However, the complexity of the attack and the fact that obfuscated expressions could be tailor made to prevent a suitable set of input/output for Lagrange interpolation makes this obfuscation an effective tool against the reverse engineering process.

## 4.4 Applications in Obfuscation

In this section, we assess the use of our contributions in the field of program obfuscation, whereas by using permutation polynomial inversion, or identity polynomials as presented in Section 4.2.1.

The cost of obfuscation can be determined either during obfuscation time or during execution. We focus more on the latter, since obfuscation usually has limitations regarding the performance overhead it produces, but more rarely on the time needed to actually obfuscate the input program.

### 4.4.1 Using Permutation Polynomial Inversion

The obfuscation technique presented in [ZMGJ07] aims at hiding a constant $K$ with a permutation polynomial $f$, its inverse $f^{-1}$ and an MBA expression $E$ equivalent to zero, with the following process:

$$K = f^{-1}(E + f(K))$$

Thus what will appear in the obfuscated code is a polynomial in the variables of $E$. Obviously, this method could be used with any non-trivial expression $E$ equivalent to zero, not only MBA expressions.

Our results regarding binary permutation polynomial inversion allow any designer to use any of these permutations for this type of obfuscation. This offer both more diversity in the obfuscation technique, and resilience to algebraic attacks such as presented in [BJLS17].

For example, the binary permutation polynomial:

$$f = 195907858x^3 + 727318528x^2 + 3506639707x + 6132886$$

could be used to obfuscate a constant $K$. However 195907858 is only divisible by 2 while 727318528 is divisible by $2^{16}$. Therefore, when trying to simplify $f$ with the technique presented in [BJLS17], the factor $2^{16}$ is supposed to divide both coefficients $a_3$ and $a_2$, and in fact, it only divides $a_3$. This polynomial thus provides a counter example to the attack proposed in [BJLS17].

More generally, $f$ and $f^{-1}$ can be used to encode variables or constants. For constants, this means computing the value $f(K)$ during obfuscation, storing this value in the program instead of $K$, and using $f^{-1}(f(K))$ during execution. In the same manner, one can replace every writing occurrences of a variable $x$ with $f(x)$ and all reading instructions of $f(x)$ with $f^{-1}(f(x))$. This process is quite common in obfuscation as it prevents the value of $x$ from appearing in memory, but is more costly since it requires two evaluations of a permutation polynomial ($f$ and $f^{-1}$) compared to the encoding of a constant ($f(K)$ is computed during obfuscation). This also means that during the obfuscation process, encoding a variable requires to check for the degrees of both $f$ and $f^{-1}$ for performances issues, while constant encoding only requires a small degree for $f^{-1}$.

Regarding encoding of variables or constants, the use of permutation polynomials does not provide much more security compared to affine functions, as much as it provides diversity. Indeed, we showed in Section 4.2.2 that inverting permutation polynomials is not a difficult problem, as Newton's method efficiently invert all binary permutation polynomials. Nevertheless, encoding with such polynomials contributes to the diversity of the obfuscation technique, thus increasing the difficulty to detect it by reverse engineering.

### 4.4.2 Using Polynomials of the Zero Ideal $I$

From the zero ideal $I$ we defined in Section 4.2.1, we can use polynomials in $I$ to design two obfuscation techniques. Let us consider a polynomial $h \in I$, then $h(x) = 0 \mod 2^n \quad \forall x \in \mathbb{Z}_{2^n}$. Therefore $h$ can be used to produce an *opaque* zero, i.e. a non-trivial null expression. This could be used for example as an opaque predicate (see [CTL97]), or in replacement of the MBA expression in Zhou et al. [ZMGJ07] obfuscation:

$$K = f^{-1}(h(x) + f(K))$$

Another use for the polynomial of this ideal is to obfuscate arithmetic expressions, namely polynomials in $\mathbb{Z}_{2^n}[x]$. Let us consider a polynomial $g(x)$ to obfuscate, one could write it $(h(x)+1) \times g(x)$ and expand the product to hide the details of $g$. It is also possible to add dependencies to other variables of the obfuscated program by using $h(z)$ for some other variable $z$ of the context: adding dependencies increases the resilience of the obfuscation technique by making the analysis of the program more complex.

On the other hand, once one example of the obfuscation scheme has been reversed, its weakness is that the analyst only needs to identify the variable $z$ not influencing the output of the computation, and remove any term in $z$. Keeping the same input for $h$ and $g$ (e.g. $(h(x)+1) \times g(x)$) removes this weakness, but quotienting by the ideal will remove $h(x)$.

## 4.5 Conclusion

In this section, we presented two different techniques for the inverse computation of elements of the set of permutation polynomials as described by [Riv01]. Those polynomials can be used in MBA obfuscation techniques as presented in article [ZMGJ07]. In addition, some elements of this set are resistant to current state of the art technique for their reverse engineering presented in [BJLS17].

One of those technique is based on Newton's method for compositional inverse. This method is extremely efficient in practice but, due to the fact that it is not constant time, is less suitable for complexity analysis. The other technique is based on Lagrange interpolation. This method is less efficient in practice but has the merit of being constant time, making it a better candidate when trying to discuss the complexity of our obfuscation techniques.

In addition, Lagrange interpolation based technique could be used by an attacker to try to invert MBA obfuscated expressions. However we discussed such a technique would be costly for an attacker and MBA expressions could be designed to impede this type of attacks.

As a field of research working against reverse engineering, obfuscation benefit from having more variety in the techniques at its disposal. Security against a reverse engineer is hard to quantify, but the more techniques we can use in our obfuscations, and the more diverse those techniques are, the harder it will be to automate the reverse engineering process. Our work brought new tools to develop MBA with polynomials of the ring $\mathbb{Z}/2^n\mathbb{Z}$ as opposed to more classical elements of Galois' Field $GF(2^n)$.

# Conclusion

Over the years, it has become more and more apparent that software security needs to address the question of security against a legitimate end-user. Indeed, many applications, including for example IoT or mobile devices, have to consider the software environment hostile. The term "software protection" refers to techniques that aim to defend against attackers that are positioned in such a setting, attacks sometimes referred to as man-at-the-end attacks (MATE). In particular, software protection needs to consider protection of data such as cryptographic keys, constants or even the control flow of a software against an attacker in a white-box context. We can distinguish two fields of research with regards to software protection in a white-box context: white-box cryptography, which aims to protect cryptographic keys against an attacker in a white-box context, and obfuscation, which aims to impede the reverse engineering process (that is to say protect data that can help an attacker understanding how a software operate, such as the control flow graph of a software for example). Yet, in practice, both fields are extremely close in their application and are usually used in conjunction with each other.

The question of protection in a white-box setting is a subject far more recent than the classical black-box setting. While security in a more classical black-box setting have strong and well understood tools, security in a white-box setting proves more challenging. Indeed, tools we need and already use, like white-box cryptography for the protection of cryptographic keys, are still undergoing heavy research and struggle to resist the white-box context (as an example, see the results of both edition of the WhiBox challenge [CHE17] [CHE19]). Our work focused on two of those tools:

- asymmetric white-box cryptography, for the protection of cryptographic keys of an implementation of an asymmetric primitive in a white-box setting,
- obfuscation based on Mixed Boolean Arithmetic (MBA) expressions involving polynomial bijections modulo $2^n$, which can be used, for example, to protect a software's constants (opaque constant techniques) in a white-box setting.

Because of the lack of public research on the subject of asymmetric white-box, we began by considering the problem of its symmetric counterpart 2 with two questions in mind:

- What are the key techniques used in symmetric white-box and are they compatible with an asymmetric white-box context ?
- What properties make state of the art symmetric white-box vulnerable to the white-box context and can we break from those when considering an asymmetric scheme ?

In chapter 2, we discussed those questions with regards to a classical white-box design [CEJO03] and concluded with the following remarks: On the one hand, the AES cryptosystem presents some interesting properties with the ability to switch to a fully table based implementation, giving space for both encoding techniques and obfuscation of data techniques to protect the resulting set of lookup tables. On the other hand, AES white-box designs are still to this day quite vulnerable to white-box attacks due to its vulnerability to two types of attacks. AES is by design vulnerable to side-channel attacks in a grey-box context and those attacks can be adapted to the white-box context to become even more potent [BHMT16]. In addition, current encoding techniques are still vulnerable to algebraic attacks like the BGE attack [BGEC05]. While AES white-box designs can still hold some resilience to the white-box context when paired with state of the art obfuscation techniques, those techniques are often not enough to impede a motivated attacker in a white-box context [GPRW19]. The question we raise in the following chapter 3 is: can we follow the same logic used in AES white-box designs (implementing a fully table based design protected by encoding techniques) with regards to other cryptosystems that are easier to protect against side-channel attacks and that possess interesting properties allowing for new and more resilient encoding techniques.

Then, we gave our own proposal for an asymmetric white-box design 3 based on lattice-based cryptography that is fully table based and benefit from homomorphic properties that enable homomorphic encoding techniques which helps resist the white-box context. The initial model we propose in 3 was fully implemented in python and uphold a strict limit in size of 20MB. That limit was chosen with regards to the limitation imposed during the WhiBox challenge to justify that our proposal could have a practical usage in a real-life situation. We discussed the resilience of our design in a white-box context. While our design present some interesting properties with regard to the protection against side-channel attacks or BGE type attacks, thanks to its use of homomorphic encodings, we concluded that in its original form, our design is still vulnerable to some types of algebraic white-box attacks. We then presented different techniques that can be applied to our initial design to prevent those type of attacks, and discussed the drawbacks of those techniques as far as practical usage of our proposal is concerned. In conclusion, we presented a first public proposal for an asymmetric white-box design. That proposal is composed of an initial design that is usable in practice but still retain some vulnerabilities in a white-box context, and a more advanced design that does not present those vulnerabilities but is not suited for all practical use of an asymmetric white-box.

Our future work on that subject will be three-folds:

- improving the advanced techniques presented in 3 to ensure a secure design that remains practical,
- consider potential new types of attacks specific to lattice-based cryptography in a white-box context,
- consider other asymmetric cryptosystems for white-box applications, or consider a design for a cryptosystem tailored to the white-box context.

Finally, We discussed our work on polynomial bijections modulo $2^n$ and their use in MBA based obfuscation 4. Like white-box cryptography, obfuscation is one technique used in software protection to provide security against an attacker in a white-box context. In 2007, an obfuscation technique coupling MBA obfuscation and polynomial bijections modulo $2^n$ was proposed in [ZMGJ07]. Such an obfuscation technique requires knowledge of both a polynomial bijection and its inverse. Unfortunately, the only mean of inverting such a bijection in this article was a closed form formula that only worked on a restricted subset of polynomial bijections modulo $2^n$. An attack exploiting this restriction was presented in [BJLS17]. In our work, we gave new inversion techniques for polynomial bijections modulo $2^n$ that successfully invert all permutation polynomials originally described in [Riv01]. Those inversion techniques allow for a use of all polynomial bijections modulo $2^n$ in MBA based obfuscation, including polynomials that resist the known state of the art attack against such constructs. We described two inversion techniques: A recursive inversion technique based on Newton's method that is extremely effective in practice, but which does not have a fixed complexity and is not applicable on an obfuscated expression without extracting first the underlying permutation polynomial, making it hard to measure the complexity of potential attacks. An inversion technique based on Lagrangian interpolation that has a fixed complexity and which only requires a set of inputs/outputs from an obfuscated expression to be performed, making it ideal to consider the complexity of our obfuscation. We concluded that MBA based obfuscation using polynomial bijections modulo $2^n$ can be a strong tool for data obfuscation if the underlying permutation polynomials are chosen with care.

As far as future works are concerned, we believe our work on polynomial bijections modulo $2^n$ can be adapted to consider polynomial bijections modulo $p^n$, with $p$ a prime number. While the usage of such polynomials may prove less practical for obfuscation purposes, we think this subject could be interesting from a mathematical perspective as the literature on such objects is particularly scarce.

# Bibliography

[AAMSD18] Lex Aaron Anderson (Auckland) et CA) Alexander Medvinsky (San Diego, CA)and Rafie Shamsaasef (San Diego : Protecting The INPUT/OUTPUT Of Modular Encoded White-Box RSA, July 2018. U.S. patent number 20180198613.

[ABBK17] Nabil Alkeilani Alkadri, J. Buchmann, Rachid El Bansarkhani et J. Krämer : A Framework to Select Parameters for Lattice-Based Cryptography. *IACR Cryptol. ePrint Arch.*, 2017:615, 2017.

[AMBG$^+$16] Carlos Aguilar-Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian et Tancrède Lepoint : NFLlib: NTT-Based Fast Lattice Library. *In* Kazue Sako, éditeur : *Topics in Cryptology - CT-RSA 2016*, pages 341–356, Cham, 2016. Springer International Publishing.

[AMR20] Alessandro Amadori, Wil Michiels et Peter Roelse : A DFA Attack on White-Box Implementations of AES with External Encodings. *In* Kenneth G. Paterson et Douglas Stebila, éditeurs : *Selected Areas in Cryptography – SAC 2019*, pages 591–617, Cham, 2020. Springer International Publishing.

[BBB$^+$18] B. Bunz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille et G. Maxwell : Bulletproofs: Short Proofs for Confidential Transactions and More. *In 2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, Los Alamitos, CA, USA, may 2018. IEEE Computer Society.

[BCD06] Julien Bringer, Herve Chabanne et Emmanuelle Dottax : White Box Cryptography: Another Attempt. *IACR Cryptology ePrint Archive*, 2006:468, 01 2006.

[BCGM19] Pierrick Brunet, Béatrice Creusillet, Adrien Guinet et Juan Martinez : Epona and the Obfuscation Paradox: Transparent for Users and Developers, a Pain for Reversers. pages 41–52, 11 2019.

[BDK$^+$18] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler et D. Stehle : Crystals - kyber: A cca-secure module-lattice-based kem. *In 2018 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 353–367, 2018.

[BDK$^+$20] Michiel Van Beirendonck, Jan-Pieter D'Anvers, Angshuman Karmakar, Josep Balasch et Ingrid Verbauwhede : A Side-Channel Resistant Implementation of SABER. Cryptology ePrint Archive, Report 2020/733, 2020. https://eprint.iacr.org/2020/733.

[BEHZ17] Jean-Claude Bajard, Julien Eynard, M. Anwar Hasan et Vincent Zucca : A Full RNS Variant of FV Like Somewhat Homomorphic

Encryption Schemes. *In* Roberto AVANZI et Howard HEYS, éditeurs : *Selected Areas in Cryptography – SAC 2016*, pages 423–442, Cham, 2017. Springer International Publishing.

[BERR16] Lucas BARHELEMY, Ninon EYROLLES, Guenaël RENAULT et Raphaël ROBLIN : Binary Permutation Polynomial Inversion and Application to Obfuscation Techniques. *In Proceedings of the 2016 ACM Workshop on Software PROtection*, SPRO '16, page 51–59, New York, NY, USA, 2016. Association for Computing Machinery.

[BGEC05] Olivier BILLET, Henri GILBERT et Charaf ECH-CHATBI : Cryptanalysis of a White Box AES Implementation. *In* Helena HANDSCHUH et M. Anwar HASAN, éditeurs : *Selected Areas in Cryptography*, pages 227–240, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[BGI$^+$12] Boaz BARAK, Oded GOLDREICH, Russell IMPAGLIAZZO, Steven RUDICH, Amit SAHAI, Salil VADHAN et Ke YANG : On the (Im)Possibility of Obfuscating Programs. *J. ACM*, 59(2), mai 2012.

[BGV14] Zvika BRAKERSKI, Craig GENTRY et Vinod VAIKUNTANATHAN : (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Trans. Comput. Theory*, 6(3), juillet 2014.

[BHMT16] Joppe W. BOS, Charles HUBAIN, Wil MICHIELS et Philippe TEUWEN : Differential Computation Analysis: Hiding Your White-Box Designs is Not Enough. *In* Benedikt GIERLICHS et Axel Y. POSCHMANN, éditeurs : *Cryptographic Hardware and Embedded Systems – CHES 2016*, pages 215–236, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[BJLS17] Fabrizio BIONDI, Sébastien JOSSE, Axel LEGAY et Thomas SIRVENT : Effectiveness of Synthesis in Concolic Deobfuscation. *Computers & Security*, 70:500–515, sep 2017.

[BKRS18] Lucas BARTHÉLÉMY, Delaram KAHROBAEI, Guenaël RENAULT et Zoran SUNIĆ : Quadratic time algorithm for inversion of binary permutation polynomials. ICMS 2018 - South Bend (USA) Notre Dame, 2018.

[BLSK98] Jean-Claude BAJARD, Didier LAURENT-STÉPHANE et Peter KORNERUP : An RNS Montgomery modular multiplication algorithm. *Computers, IEEE Transactions on*, 47:766 – 776, 08 1998.

[Bos10] Alin BOSTAN : Algorithmes rapides pour les polynômes, séries formelles et matrices. *In Actes des Journées Nationales de Calcul Formel*, Les cours du CIRM, tome 1, numéro 2, pages 75–262, Luminy, France, 2010.

[BV11] Zvika BRAKERSKI et Vinod VAIKUNTANATHAN : Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. volume 6841, pages 505–524, 08 2011.

[BZ10] R.P. BRENT et P. ZIMMERMANN : *Modern Computer Arithmetic*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2010.

[CEJO03] Stanley CHOW, Philip A. EISEN, Harold JOHNSON et Paul C. van OORSCHOT : White-Box Cryptography and an AES Implementation. *In Revised Papers from the 9th Annual International Workshop on Selected Areas in Cryptography*, SAC '02, pages 250–270, London, UK, UK, 2003. Springer-Verlag.

[CHE17] CHES 2017 Capture the Flag Challenge, 2017.

[CHE19] CHES 2019 Capture the Flag Challenge, 2019.

[CT65] James W. COOLEY et John W. TUKEY : An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comput.*, 19:297–301, 1965.

[CTL97] Christian COLLBERG, Clark THOMBORSON et Douglas LOW : A Taxonomy of Obfuscating Transformations. Rapport technique 148, University of Auckland, 1997.

[DKSRV18] Jan-Pieter D'ANVERS, Angshuman KARMAKAR, Sujoy SINHA ROY et Frederik VERCAUTEREN : Saber: Module-LWR Based Key Exchange, CPA-Secure Encryption and CCA-Secure KEM". *In* Antoine JOUX, Abderrahmane NITAJ et Tajjeeddine RACHIDI, éditeurs : *Progress in Cryptology – AFRICACRYPT 2018*, pages 282–305, Cham, 2018. Springer International Publishing.

[DLV03] Pierre DUSART, Gilles LETOURNEUX et Olivier VIVOLO : Differential Fault Analysis on A.E.S. *In* Jianying ZHOU, Moti YUNG et Yongfei HAN, éditeurs : *Applied Cryptography and Network Security*, pages 293–306, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[DR98] Joan DAEMEN et Vincent RIJMEN : The block cipher rijndael. volume 1820, pages 277–284, 01 1998.

[Eyr17] Ninon EYROLLES : *Obfuscation par expressions mixtes arithmético-booléennes : reconstruction, analyse et outils de simplification*. Thèse de doctorat, 2017. Thèse de doctorat dirigée par Goubin, Louis et Videau, Marion Informatique Université Paris-Saclay (ComUE) 2017.

[FV12] Junfeng FAN et F. VERCAUTEREN : Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.

[GAGL15] Adrien GUINET, Carlos AGUILAR, Serge GUELTON et Tancrède LEPOINT : Quatre millions d'échanges de clés par secondes. SSTIC Archive, 2015, 2015.

[GPRW19] Louis GOUBIN, Pascal PAILLIER, Matthieu RIVAIN et Junwei WANG : How to Reveal the Secrets of an Obscure White-Box Implementation. Journal of Cryptographic Engineering, 10,49-66(2020), 2019.

[GR07] Shafi GOLDWASSER et Guy N. ROTHBLUM : On Best-Possible Obfuscation. *In* Salil P. VADHAN, éditeur : *Theory of Cryptography*, pages 194–213, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[Her20] Paul HERNAULT : Introduction to Whiteboxes and Collision-Based Attacks With QBDI, 8 2020.

[HM16] Jan HOOGERBRUGGE et Wil MICHIELS : Protecting The INPUT/OUTPUT Of Modular Encoded White-Box RSA, November 2016. U.S. patent number 20160328543.

[HW95] Godfrey HARDY et Edward WRIGHT : *An Introduction to Theory of Numbers*. 01 1995.

[Kar11] Mohamed KARROUMI : Protecting White-Box AES with Dual Ciphers. *In* Kyung-Hyune RHEE et DaeHun NYANG, éditeurs : *Information Security and Cryptology - ICISC 2010*, pages 278–291, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[KJJ99] Paul KOCHER, Joshua JAFFE et Benjamin JUN : Differential power analysis. *In* Michael WIENER, éditeur : *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[LMPR08] Vadim LYUBASHEVSKY, Daniele MICCIANCIO, Chris PEIKERT et Alon ROSEN : SWIFFT: A Modest Proposal for FFT Hashing. *In* Kaisa NYBERG, éditeur : *Fast Software Encryption*, pages 54–72, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[LN16a] Patrick LONGA et Michael NAEHRIG : Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography. *In* Sara FORESTI et Giuseppe PERSIANO, éditeurs : *Cryptology and Network Security*, pages 124–139, Cham, 2016. Springer International Publishing.

[LN16b] Patrick LONGA et Michael NAEHRIG : Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography. Cryptology ePrint Archive, Report 2016/504, 2016. https://eprint.iacr.org/2016/504.

[LPR13] Vadim LYUBASHEVSKY, Chris PEIKERT et Oded REGEV : On Ideal Lattices and Learning with Errors over Rings. *J. ACM*, 60(6), novembre 2013.

[LR13] Tancrède LEPOINT et Matthieu RIVAIN : Another Nail in the Coffin of White-Box AES Implementations. Cryptology ePrint Archive, Report 2013/455, 2013. https://eprint.iacr.org/2013/455.

[MG09] Wilhelmus P.A.J. MICHIELS et Paulus M.H.M.A. GORISSEN : White-box implementation, February 2009. U.S. patent number 20110150213A1.

[Mic05] Daniele MICCIANCIO : *Closest Vector Problem*, pages 79–80. Springer US, Boston, MA, 2005.

[MN85] Gary L. MULLEN et Harald NIEDERREITER : The structure of a group of permutation polynomials. *J. Austral. Math. Soc. Ser. A*, 38(2):164–170, 1985.

[MS84] G. MULLEN et H. STEVENS : Polynomial functions (mod m). *Acta Mathematica Hungarica*, 44(3-4):237–241, septembre 1984.

[Noe15] Shen NOETHER : Ring Signature Confidential Transactions for Monero. Cryptology ePrint Archive, Report 2015/1098, 2015.

[OP00] Halil ORUÇ et George PHILLIPS : Explicit factorization of the Vandermonde matrix. *Linear Algebra and its Applications*, 315:113–123, 08 2000.

[OSPG16] Tobias ODER, Tobias SCHNEIDER, Thomas PÖPPELMANN et Tim GÜNEYSU : Practical CCA2-Secure and Masked Ring-LWE Implementation. Cryptology ePrint Archive, Report 2016/1109, 2016. https://eprint.iacr.org/2016/1109.

[PPM17] Robert PRIMAS, Peter PESSL et Stefan MANGARD : Single-Trace Side-Channel Attacks on Masked Lattice-Based Encryption. Cryptology ePrint Archive, Report 2017/594, 2017. https://eprint.iacr.org/2017/594.

[PQC16] Announcing Request for Nominations for Public-Key Post-Quantum Cryptographic Algorithms, 12 2016.

[Riv01] Ronald L. RIVEST : Permutation Polynomials Modulo 2w. *Finite Fields and Their Applications*, 7(2):287–292, avril 2001.

[Rob17] Thomas G. ROBERTAZZI : *AES and Quantum Cryptography*, pages 129–140. Springer International Publishing, Cham, 2017.

[Sho97] Peter W. SHOR : Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484–1509, 10 1997.

[SM10] Zoran Sunic SMILE MARKOVSKI, Danilo Gilgoroski : Polynomial functions on the units of $Z_{2^n}$. *Quasigroups and Related Systems*, 18:59–82, 2010.

[SW08] Amitabh SAXENA et Brecht WYSEUR : On white-box cryptography and obfuscation. 06 2008.

[TH16] Philippe TEUWEN et Charles HUBAIN : Differential Fault Analysis on White-box AES Implementations, 12 2016.

[XL09] Yaying XIAO et Xuejia LAI : A secure implementation of White-Box AES. 12 2009.

[ZMGJ07] Yongxin ZHOU, Alec MAIN, Yuan Xiang GU et Harold JOHNSON : Information Hiding in Software with Mixed Boolean-Arithmetic Transforms. *In 8th International Workshop in Information Security Applications (WISA'07)*, pages 61–75, 2007.