Traitements sécurisés de données compressées : application à l'apprentissage automatique et aux implants connectés

Maxime Pistono

DOCTORAT BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THÈSE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPÉRIEURE
MINES-TÉLÉCOM ATLANTIQUE BRETAGNE
PAYS-DE-LA-LOIRE - IMT ATLANTIQUE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Maxime PISTONO

Traitements sécurisés de données compressées – Application à l'apprentissage automatique et aux implants connectés

Thèse présentée et soutenue à 655 Avenue du Technopôle, 29280 Plouzané, le 14 Décembre 2021
Unité de recherche : LaTIM Inserm UMR 1101
Thèse N° : 2021IMTA0259

Rapporteurs avant soutenance :

Fabrice MERIAUDEAU Professeur à l'Université de Bourgogne
François ARNAULT Maître de conférences, HDR à l'Université de Limoges

Composition du Jury :

Présidente :	Maryline LAURENT	Professeur IMT Télécom SudParis
Examinateurs :	François ARNAULT	Maître de conférences, HDR à l'Université de Limoges
	Fabrice MERIAUDEAU	Professeur à l'Université de Bourgogne
Dir. de thèse :	Gouenou COATRIEUX	Professeur à IMT Atlantique
Co-dir. de thèse :	Jean-Claude NUNES	Maître de conférences, HDR à l'Université de Rennes 1
Co-encadrant de thèse :	Reda BELLAFQIRA	Maître de conférences à IMT Atlantique

Invité :

Michel COZIC Directeur innovation à MEDECOM

Remerciements

Par avance, je tiens à m'excuser si j'oublie des personnes dans ces remerciements.

Pour commencer, je souhaiterais remercier mes directeurs Gouenou Coatrieux, Jean-Claude Nunes et Reda Bellafqira. Je tiens à remercier particulièrement Reda pour les nombreuses discussions et réflexions que l'on a pu avoir ensemble.

Je remercie également François Arnault et Fabrice Mériaudeau pour avoir accepté d'être les rapporteurs de cette thèse ainsi que Maryline Laurent d'avoir été la présidente de mon jury et Michel Cozic pour avoir participé à mon jury.

Merci également à toutes les personnes avec lesquelles j'ai pu travailler au cours de ces trois dernières années : Anass, Chloé, David, Duong, Mathieu, Mounia, Raphaël, Theo, Yan et Yutong. Je tiens également à remercier les professeurs et professeures du département qui ont toujours eu un mot gentil ou une attention particulière : Johanne Vincent, Jacques Simonin, Didier Gueriot, Pierre-Henri Conze et François Rousseau. Je remercie également Ahcène Bounceur, Elsa Dupraz et Dominique Pastor pour leur confiance durant mes différents enseignements.

Je tiens également à remercier les autres doctorants et plus maintenant avec qui nous avons partagés de très bons moments : Carlos, Xiaoyu, Oscar, Mathilde, Lucas, Damien, Reda et Maxime qui m'a tant appris. Je vous souhaite le meilleur pour la suite.

Un grand merci à Corinne Le Lann pour faciliter notre travail et plus généralement pour nos conversations toujours très plaisantes et plaines d'humour.

Merci à tous ceux avec qui j'ai eu la chance de faire mes études notamment Éric, Teddy, Yacine, Nicolas et Gaël avec qui j'ai passé de très bons moments que ce soit à jouer aux cartes ou à marcher aux calanques.

Je remercie mes amis de toujours Julien, Jouffi et Alex pour m'avoir toujours accueilli à bras ouvert à chaque fois comme si je n'étais jamais parti.

Pour conclure ces remerciements, merci à ma famille, mes parents et plus particulièrement mes deux frères Julien et Nicolas d'avoir toujours été à mes côtés dans les bons et mauvais moments et de m'avoir toujours poussé à aller plus loin. Enfin, merci à Laura sans qui je ne serais rien, c'est elle qui me soutient, me supporte, me fait rire et me pousse chaque jour à avancer et à donner le meilleur de moi-même.

TABLE DES MATIÈRES

Introduction **9**

1 Sécurisation et traitement de données médicales compressées **13**

1.1 Caractérisation des données de santé et réglementation 15

1.1.1 Spécification des données médicales 16

1.1.2 Les risques accompagnant ces informations 17

1.1.3 Les implants médicaux connectés 18

1.1.4 Obligations de sécurisation 21

1.1.5 Externalisation et réutilisation 26

1.2 Traitement automatisé et aide à la décision 26

1.2.1 Objectifs du traitement automatisé de données 27

1.2.2 Les traitements de type linéaire avec seuillage 28

1.2.3 Les avancées du *machine learning* 28

1.3 Sécurisation des traitements de données 31

1.3.1 Chiffrements rapides et peu coûteux 31

1.3.2 Traitements sur données chiffrées 32

1.4 Optimisation du stockage et transfert des données 33

1.4.1 Compression optimale de Huffman 35

1.4.2 JPEG - un standard de compression pour les images 35

1.4.3 Compression et chiffrement de données 36

1.5 Conclusion 37

2 Traitement de données concaténées chiffrées **39**

2.1 Outils de sécurisation et de traitement de données 40

2.1.1 Traitement sécurisé au travers du chiffrement additivement homomorphe 42

2.1.1.1 Définitions et premières propositions 42

2.1.1.2 Cryptosystème de Damgård–Jurik 44

2.1.2 Algorithmes rapides et peu coûteux assurant la sécurité 48

2.1.2.1 Chiffrement par flot CLCG 49
2.1.3 Conversion de chiffrement par flot et additivement homomorphe . . 50
2.1.3.1 Chiffrement par flot CLCG dans le domaine homomorphe 51
2.1.3.2 Système de conversion de chiffré 52
2.1.4 Attentes de la concaténation de données 54
2.1.4.1 Concaténation et conséquences sur les opérations homomorphes 56
2.1.4.2 Extraction des données 60
2.1.5 Assurer l'intégrité des données avec le tatouage 63
2.2 Sécurisation des communications et des traitements d'un implant connecté 68
2.2.1 Cas d'usage *Followknee* et rappel des hypothèses de sécurité 68
2.2.2 Le protocole de sécurisation et traitement complet 68
2.2.2.1 Protocole sécurisé sans concaténation 70
2.2.3 Protocole sécurisé avec concaténation 72
2.2.4 Une variante au scénario *Followknee* 76
2.3 Analyse de complexité et résultats expérimentaux 77
2.3.1 Complexité du système de traitement sécurisé 78
2.3.2 Résultats expérimentaux 79
2.4 Analyse de sécurité 80
2.4.1 *Correctness* 81
2.4.2 *Privacy* 82
2.4.3 La sécurité du CLCG 83
2.4.4 Cas d'une IHM malicieuse 85
2.5 Conclusion 86

3 Traitements de données compressées - Apprentissage automatique sur des images compressées 89
3.1 L'apprentissage automatique par réseaux de neurones 91
3.1.1 Principes élémentaires des réseaux de neurones 91
3.1.2 Les avancées proposées par les réseaux convolutifs 94
3.1.3 Apprentissage pour les algorithmes de *machine learning* 96
3.2 JPEG : le standard de compression d'images 99
3.2.1 Les bases de données partiellement décompressées 102
3.3 Résultats expérimentaux 104

3.3.1 Le choix des banques de données . . . 104
3.3.2 Les architectures NN . . . 105
3.3.2.1 Modèle avec une seule couche cachée . . . 106
3.3.2.2 Modèle à deux couches cachées . . . 108
3.3.3 Les architectures CNN . . . 109
3.3.3.1 Modèle d'Ulicny et Dahyot . . . 109
3.3.3.2 Modèle de Keras . . . 111
3.3.4 Discussion des résultats obtenus . . . 112
3.4 Conclusion . . . 113

4 Décompression partielle sécurisée de données compressées JPEG 115
4.1 Décompression de l'algorithme JPEG . . . 117
4.1.1 Algorithme de compression de Huffman . . . 117
4.1.2 L'algorithme de parcours en largeur . . . 119
4.2 Additionner et multiplier des données chiffrées : les cryptosystèmes *fully* homomorphes . . . 122
4.2.1 Le cryptosystème FHE de BFV . . . 123
4.3 Décompression sécurisée de mots de code de Huffman . . . 125
4.3.1 La décompression de Huffman . . . 125
4.3.2 Variables et opérations associées à la décompression . . . 127
4.3.2.1 Extraction de la valeur contenue dans une feuille . . . 128
4.3.2.2 Calcul du décalage . . . 130
4.3.2.3 Sauvegarde des données décodées . . . 130
4.3.2.4 Mise à jour des mémoires des valeurs booléennes . . . 131
4.3.3 Passage aux données chiffrées . . . 132
4.3.4 Scénario global . . . 134
4.4 Résultats expérimentaux . . . 135
4.4.1 Les données utilisées . . . 135
4.4.2 Les capacités de calcul utilisées . . . 136
4.5 Sécurité . . . 136
4.6 Discussion . . . 137
4.7 Extension de la méthode aux arbres de décision binaires et au RLE . . . 138
4.7.1 Les arbres de décision binaires . . . 138
4.7.2 Le Run-Length Encoding . . . 140

4.8 Conclusion 142

5 Chaîne de traitement sécurisée de données compressées 145

5.1 Classification et apprentissage sécurisé par un PHE de données concaténées 145

5.1.1 Présentation du scénario d'application et des entités impliquées . . 148

5.1.2 Sur données concaténées sécurisées 149

5.1.2.1 *Machine learning* sur données concaténées en clair 152

5.1.2.2 Sur données concaténées chiffrées 153

5.1.3 Un modèle de *machine learning* sécurisé sur données concaténées . 168

5.1.3.1 La classification sécurisée sur données concaténées 168

5.1.3.2 L'apprentissage sécurisé sur données concaténées 169

5.1.3.3 Résultats expérimentaux 170

5.1.4 Discussion des résultats 173

5.2 Classification sécurisée par un FHE de données compressées 174

5.2.1 Le cas de données compressées de Huffman 175

5.2.2 Application aux images compressées JPEG 175

5.3 Conclusion 176

Conclusion 179

Bibliographie 185

INTRODUCTION

Les avancées technologiques notamment en ce qui concerne la miniaturisation électronique ont permis l'acquisition toujours plus importante de données. Le stockage du volume important de données numériques ainsi créé a un coût important. En effet, ce dernier nécessite une infrastructure informatique adaptée. Une solution pour diminuer ces coûts consiste à mutualiser les données vers un *cloud*. Cette mutualisation a également d'autres avantages. Pour commencer, cette masse importante de données peut faire l'objet d'un traitement automatisé (*e.g. machine learning*), ainsi que d'analyse de type « *Big data* ». Elle permet également à différents acteurs de pouvoir accéder à ces informations à distance. Plus spécifiquement, dans le domaine médical, cela facilite la prise en charge du patient. Toujours dans le domaine médical, les *Implantable Medical Devices* (IMD) sont apparus grâce à ces mêmes avancées technologiques. Ces appareils permettent de fournir au patient une meilleure prise en charge. Certains d'entre eux sont dotés de capteurs ayant la capacité d'acquérir des mesures biologiques. Ces données peuvent être traitées à l'aide d'algorithmes automatisés afin de mieux suivre l'état d'un patient. Dans le même temps, il est important de noter que les données externalisées, et encore plus particulièrement celles de santé, sont des données personnelles. Or, pour l'utilisateur, une fois que ses données sont sur le *cloud*, il en perd le contrôle. Il est alors impératif d'en assurer la sécurité (*e.g.* chiffrement, tatouage). De plus, cette injonction est dictée par plusieurs réglementations nationales et internationales (*e.g.* RGPD). La sécurisation des données passe par assurer ou garantir leur : confidentialité, authenticité, intégrité et traçabilité. Dans un tel contexte, un premier enjeu est de définir des approches permettant de traiter de façon sécurisée les données externalisées (*e.g.* cryptosystèmes homomorphes). L'idée étant par exemple de pouvoir permettre à un tiers, *e.g.* le *cloud*, de traiter les données sans les déchiffrer et de retourner à l'utilisateur le résultat du traitement toujours sous forme chiffrée. À cet objectif, vient se rajouter dans le domaine de la santé une autre problématique : la compression de données. En effet, prenons pour exemple les IMD, ceux-ci sont des appareils fortement contraints en ce qui concerne leur puissance de calcul, de stockage, ... Il est alors impératif de diminuer autant que possible les coûts de communications. L'imagerie médicale en est un autre exemple. Chaque année un hôpital produit

des téraoctets de données d'imagerie. Plus précisément, ces données sont généralement stockées et transmises sous formes compressées afin d'en diminuer les coûts. Aujourd'hui, dans la majorité des cas, les traitements s'effectuent sur des données non compressées et impliquent des temps de calculs liés aux processus de décompression. Il y a donc un intérêt à pouvoir traiter les données sous une forme compacte. Ces enjeux sont au cœur de ces travaux de thèse qui visent à concilier la compression de données avec leur traitement tout en assurant leur sécurité.

Au cours de ces trois ans de thèse, nous avons commencé par nous intéresser dans le Chapitre 2 à la sécurisation du scénario *Followknee* impliquant un IMD (prothèse de genoux) souhaitant envoyer des données patient sur une Interface Homme-Machine (IHM, *e.g.* smartphone) non considérée comme de confiance, afin de les traiter. Dans l'idée, Cette IHM doit traiter les données et prévenir le patient ainsi que son médecin en cas d'anomalie. Ce scénario fait également intervenir un Serveur ayant pour but d'aider au traitement ainsi que de vérifier que l'IHM n'adopte pas un comportement malveillant. Plus précisément, l'IHM n'a pas le droit de connaître les données médicales du patient : cela nuirait au respect de sa vie privée. D'autre part, l'IHM utilisant des paramètres secrets (propriété intellectuelle) pour effectuer ses traitements, le Serveur (appartenant à une autre société) ne doit pas y avoir accès. La solution apportée combine deux chiffrements distincts. L'un léger et rapide, le *Combined Linear Congruential Generator* (CLCG) adapté aux contraintes de l'IMD et l'autre, celui de Damgård–Jurik (D-J) qui permet un certain nombre de traitements au niveau de l'IHM. Nous avons proposé un algorithme de conversion de chiffré (CrC) qui permet de passer de l'un à l'autre afin de ne profiter que de leurs avantages. La combinaison de ces deux chiffrements apporte une augmentation de coûts de communication qui est palliée par l'utilisation d'une technique de concaténation de données. Cette technique transmet et traite plusieurs données en une unique fois et permet ainsi de diminuer les coûts de communication. Une technique de tatouage numérique a également été proposée. Elle peut être ajoutée au protocole de traitement sécurisé afin d'assurer l'intégrité des données. Finalement, les performances de la solution proposée ont été évaluées et permettent de voir que celle-ci est utilisable en cas réel. Pour résumé, nous avons proposé un protocole original pour le scénario *Followknee*. Cette solution, efficace, s'appuie sur le nouveau protocole CrC, la concaténation de données et le tatouage de données.

Comme nous le verrons dans le Chapitre 3, nous avons étudié des techniques de traitement de données plus élaborées que sont les algorithmes de *machine learning*. Ces derniers

permettent d'automatiser des tâches complexes telles que la détection de tumeur dans une image, la classification de lésions cutanées, ... Cependant, dans le cas où l'entrée de ces modèles est constituée d'images, ces dernières doivent être décompressées. Or, comme dit plus haut les images sont presque systématiquement stockées et transmises sous formes compressées du fait de leur taille importante. En santé, le standard d'image est DICOM qui s'appuie pour la partie compression des images sur le standard JPEG. Ce dernier algorithme est aussi celui actuellement utilisé pour la quasi-totalité des images grand public. La question est : est-il possible de faire l'apprentissage et la classification à l'aide d'algorithmes de *machine learning* sur données compressées ou partiellement compressées ? Ceci nous permettrait de nous passer au moins partiellement de la phase de décompression et ainsi d'améliorer les temps de calcul. Pour répondre à cette question, nous avons étudié le processus de compression/décompression JPEG et créé, pour chacune des étapes de ce dernier, des bases de données partiellement compressées. En utilisant des modèles de *Neural Networks* (NN) et *Convolutional Neural Networks* (CNN) proposés dans la littérature scientifique, les travaux que nous avons menés montrent qu'il est possible de faire l'apprentissage et la classification de données partiellement décompressées. Cependant, lorsque les données ainsi que le modèle sont complexes, des pertes de performances apparaissent. Un compromis entre le temps gagné lors de la décompression partielle et la perte de précision est cependant possible.

La suite de nos travaux, présentée au Chapitre 4, a consisté logiquement à étudier la possibilité de décompresser partiellement des images JPEG de façon sécurisée. Pour cela, nous avons proposé une méthode de décompression de l'algorithme de Huffman, un codeur entropique essentiel de la dernière étape de la chaîne de compression JPEG. L'idée majeure en arrière-plan de cette solution a consisté à voir que la structure de décompression Huffman s'appuie sur un arbre binaire pour lequel chaque nœud représente un test à évaluer. L'évaluation de ce test de façon sécurisée nous a amené à utiliser des chiffrements permettant le traitement sécurisé de données. Ces chiffrements (nommés *fully* homomorphes) sont distincts de celui de Damgård–Jurik (additivement homomorphe) utilisé dans le scénario de l'IMD car ils proposent de sécuriser davantage d'opérations aux dépens d'une performance de calcul amoindrie. La méthode proposée permet de décoder de façon sécurisée un flux binaire composé de mot de code de Huffman. La sécurité est assurée par le fait que le Serveur ne peut pas tirer d'information des données qu'il traite. En effet, si ce dernier arrivait à connaître la longueur des mots de code, du flux binaire, ou même s'il arrivait à savoir à quelle étape de décompression l'on se trouve, cela constituerait

un problème de sécurité. Dans ces cas-là, le Serveur pourrait essayer de retrouver les mots de code utilisés. Des tests ont été effectués montrant la possibilité de l'utiliser sur des images directement codées par Huffman. Il faut cependant noter que les performances sont limitées par les algorithmes de chiffrements utilisés. Néanmoins, la méthode développée reste générale et allie faible coût de communication, sécurité des données et possibilité de traitements. Elle pourrait profiter dans le futur d'une implémentation performante d'un cryptosystème *fully* homomorphe.

Finalement, comme présenté au Chapitre 5, notre travail a consisté à réunir les outils présentés dans les chapitres ci-dessus afin de proposer des chaînes de traitements partant d'un utilisateur possédant des données personnelles compressées. Cet utilisateur les externalise vers le *cloud* qui peut à son tour soit directement, soit en faisant la décompression, les traiter à l'aide d'un algorithme de *machine learning*, le tout de façon sécurisée. Pour cela, nous proposerons une méthode utilisant le calcul multipartite sur données concaténées et chiffrées homomorphiquement qui permet de sécuriser les opérations nécessaires pour l'apprentissage et l'inférence d'un réseau NN. Ensuite, nous montrerons comment en utilisant un chiffrement *fully* homomorphe, il est possible de partiellement décompresser des images avant d'en faire la classification par un algorithme de *machine learning* de façon sécurisée. Nous montrerons également les limites actuelles de nos solutions et les solutions qu'un algorithme de chiffrement *fully* homomorphe performant pourrait apporter.

Avant de répondre à ces problématiques, définissons et présentons dans le Chapitre 1 les différents outils, notions et entités que nous utiliserons au cours de ce manuscrit.

Chapitre 1

Sécurisation et traitement de données médicales compressées

Durant cette thèse, nous avons essayé de concilier les trois propriétés suivantes :

1. **la sécurité**,
2. **le traitement**,
3. **la compression**

L'objectif étant de les appliquer à des données, plus précisément à des données médicales. Pour répondre à la question : "pourquoi avons-nous souhaité appliquer ces propriétés aux données médicales ?", il faut comprendre le contexte dans lequel ces données évoluent.

Depuis quelques années maintenant, l'outil informatique a pris une place prépondérante dans la vie de tous les jours. Plus précisément en ce qui concerne le domaine médical, les informations patient sont généralement numérisées. De plus, il est courant de pouvoir retrouver sur le site d'un laboratoire le résultat de ses analyses ou encore sur celui d'un cabinet de radiologie les images acquises. Le ministère de la Santé a lancé le dossier médical partagé (DMP) [Laf19]. L'objectif de ce dernier est de créer un dossier numérique retraçant le parcours de santé de chaque patient. L'idée consiste à faciliter aux professionnels de santé l'accès aux antécédents, résultats d'analyses, images médicales, etc. On pourrait rajouter à cela le développement de la télémédecine qui permet de faire des consultations [ODO20] voir des opérations à distance [Mar+01]. Toutes ces techniques permettent de faciliter l'organisation et la qualité des soins apportés aux patients. De plus, une fois que ces données sont externalisées elles peuvent être réutilisées afin d'améliorer les diagnostics ou bien de contribuer à la recherche médicale.

Cependant, comme nous le verrons dans la prochaine section, ces données de santé peuvent avoir un fort attrait pour certaines entités malveillantes. À ce titre, des organismes nationaux tels que la *Food and Drug Administration* (FDA) [Hea+18] aux États-Unis et des règlementations internationales comme le Règlement Général sur la Protection des

Données (RGPD) en Europe [VB17] ont été mises en place pour éviter ces dérives. Il faut alors pouvoir garantir la sécurité des données de santé depuis leur acquisition jusqu'à leur stockage final en passant par leur traitement. C'est dans ce cadre que la propriété de **sécurité** que nous souhaitons utiliser est considérée.

Comme évoqué plus haut, les données de santé sont traitées. À cela, il y a plusieurs raisons. Pour commencer, avec l'accélération des technologies telle que la miniaturisation des éléments électroniques, de nouveaux appareils médicaux ont pu émerger comme nous le verrons dans les sections qui suivent. Ces appareils ont la faculté de pouvoir surveiller de façon constante certains signes vitaux des patients. Or, il n'est pas imaginable de demander à un médecin ou spécialiste de traiter quotidiennement un flot aussi important de données. C'est pour cela que des programmes visant à automatiser certaine tâche de surveillance sont apparus. Ils permettent, entre autres, de rendre compte au patient ainsi qu'à son médecin de son état de santé actuel. De plus, cette masse de données se prête particulièrement bien à la réutilisation dans le cadre de recherches scientifiques (*big data*). C'est dans ce contexte que la propriété de **traitement** de ces données intervient. Cependant, il faut toujours garder à l'esprit que ces données sont des données sensibles. Il n'est alors pas envisageable d'en faire le traitement non sécurisé. On voit ici un premier couplage entre la **sécurité** des données et leur **traitement**. Notons qu'un point plus complet sur les traitements avec leurs contraintes et attentes sera fait plus loin dans ce chapitre.

Pour revenir aux données médicales en tant que telles, il est important de voir que leurs natures sont très diverses. En effet, le médecin aura plutôt tendance à faire des rapports, un laboratoire d'analyse à fournir des résultats d'examens biologiques, un scanner des images... De plus, en ce qui concerne les images, ces dernières sont stockées et transmises de façon compressée. Sinon, la taille des images sans compression sur les supports de données serait bien trop élevée. Ceci est un premier point qui nous amène au concept de **compression**. Une autre raison qui nous a amené à étudier la **compression** est dû au fonctionnement intrinsèque des algorithmes qui permettent de sécuriser et traiter les données. Ces derniers imposent d'avoir en entrée des données de très grandes longueurs. Or, généralement les données à sécuriser et traiter sont très courtes comme, par exemple, le taux de sodium *e.g.* 140 mmol/l. Un autre type de compression est amené à être mis en place afin d'optimiser ces algorithmes en termes de temps de calcul et de coût de communication.

On constate que nos trois problématiques : **sécurité**, **traitement** et **compression**

sont interdépendantes les unes des autres. Nous essaierons tout au long de ce manuscrit de les agencer deux à deux ou toutes ensemble en fonction des scénarios étudiés.

Dans ce premier chapitre, nous commençons par revenir plus en détails sur les définitions des données de santé, les dispositifs médicaux et les réglementations légales. Ceci nous permettra d'avoir une vue globale des trois problématiques étudiées. Une deuxième partie sera consacrée aux traitements automatisés de données. On verra toutes les applications, restrictions et avancées qu'ils offrent au patient, médecin et scientifique. Ces techniques évoluent et des avancées initialement pensées hors du domaine médical ont apporté des bénéfices à ce dernier. Nous continuerons par une troisième partie qui abordera plus en détail des techniques de protections de données. On entend par là des techniques cryptographiques qui permettent, entre autre, d'assurer la confidentialité de ces dernières. Enfin, avant de conclure, nous aurons une dernière partie qui présentera la compression, ses enjeux et ses avantages.

1.1 Caractérisation des données de santé et réglementation

Avant de parler de données concernant la santé, il nous semble important d'aborder le concept de données à caractère personnel. Ce dernier, tel que définit par le parlement et le conseil de l'Union européenne [Con16] fait référence à : « toute information se rapportant à une personne physique identifiée ou identifiable ». Le parlement et le conseil considèrent alors une personne physique identifiable comme : « une personne physique qui peut être identifiée, directement ou indirectement, notamment par référence à un identifiant, tel qu'un nom, un numéro d'identification, des données de localisation, un identifiant en ligne, ou à un ou plusieurs éléments spécifiques propres à son identité physique, physiologique, génétique, psychique, économique, culturelle ou sociale ». On constate alors que ces caractéristiques sont vastes. On peut, d'une part, imaginer qu'une adresse et un âge sont suffisants pour parvenir à identifier une personne. D'autre part, une couleur de cheveux peut être une information personnelle. Pour revenir à notre sujet, l'Europe considère comme données de santé toute : « données à caractère personnel relatives à la santé physique ou mentale d'une personne physique, y compris la prestation de services de soins de santé, qui révèlent des informations sur l'état de santé de cette personne ». On peut citer des exemples tels que des résultats d'analyse biologique ou encore des antécédents médicaux. D'autres exemples sont présents sur le site de la Commission Nationale de l'In-

formatique et des Libertés (CNIL) [Lib]. Cette définition de données n'est pas universelle. Par exemple aux États-Unis le *Code of Federal Regulations* (CFR) 45 CFR 164.501 définit une information médicale comme toute information, incluant les informations orales ou bien enregistrées sur un quelconque support qui :

1. Est créé ou reçu par un prestataire de santé, une autorité de santé publique, un employeur, un assureur-vie, une école ou université, ou un centre d'échange de soins de santé.
2. Rend compte de l'état passé, présent ou futur du physique ou du mental d'un individu ; des prestations de santé d'un individu ; ou du paiement passé, présent ou futur d'une prestation de soin d'un individu.

Bien que ces réglementations diffèrent, elles restent tout de même relativement proches.

1.1.1 Spécification des données médicales

Comme nous venons de le voir, la nature des données médicales ainsi que leur provenance est très diverse. Or, ces informations sont sensibles. D'une part, elles influencent le traitement administré au patient et d'autre part elle doivent rester confidentielles afin de préserver sa vie privée. Au-delà de ces premières définitions, Dusserre dans [DDA96] propose plusieurs qualités dont doit faire preuve une donnée médicale afin d'être considérée comme valide : l'accessibilité, l'actualité, l'exhaustivité, la fiabilité, la finesse de jugement, la pertinence et la précision. Ainsi, la sécurisation des données va chercher à aider à fiabiliser la donnée, mais également à assurer la protection de la vie privée du patient.

Il est également intéressant de voir qu'il pèse sur les données médicales des risques similaires à ceux présent dans d'autres domaines. En effet, les données médicales sont, elles aussi, soumises à des problèmes dont la cause peut être accidentelle, consécutive à une négligence ou malveillance. De plus, ces différentes causes peuvent être matérielles, techniques, humaines, ... [Bou13].

En plus de ces différents points, on peut noter que les données médicales ne sont pas créées, éditées ou encore consultées par n'importe quelle personne du corps médical. En effet, seuls les laboratoires peuvent proposer des résultats d'analyses, seul un médecin pourra fournir une ordonnance, ... Il en va de même pour certains résultats issus d'images médicales, tels que scanner, IRM (Imagerie par Résonance Magnétique), ou échographie, seul un spécialiste pourra éditer ces images afin de mettre en évidence une fracture,

une tumeur, le sexe de l'enfant,... Enfin comme nous le verrons par la suite l'accès aux informations sera, lui aussi encadré.

Afin de comprendre plus concrètement les raisons qui ont poussé les institutions à adopter des lois encadrant l'usage des données médicales, regardons les menaces qui pèsent sur ces dernières. De plus, nous allons aussi aborder les conséquences qu'adviendraient si les données passaient entre des mains mal intentionnées.

1.1.2 Les risques accompagnant ces informations

On peut se poser la question de pourquoi les informations médicales doivent être protégées ? Ce sont des données sensibles à plusieurs titres. Elles participent au soin des patients, bien sûr elles ne doivent pas être modifiées sous peine de nuire à la santé du patient [Rat+17] et sont confidentielles. Par exemple, elles peuvent être source de discrimination comme dans les situations suivantes :

- Un employeur pourrait préférer entre deux personnes aux qualifications et expériences équivalentes celle ne possédant pas d'antécédents médicaux, et ce, malgré le fait que la maladie ait été guérie ou qu'elle n'influence pas la disponibilité ou l'efficacité de l'employé.
- Une compagnie d'assurance pourrait faire fluctuer les cotisations en fonction des informations médicales des clients dont elle dispose.
- Lors d'une élection, le passé médical d'un candidat pourrait influencer négativement l'électorat. On peut citer l'exemple de Nydia Velazquez [Etz00] qui durant sa campagne de 1992 a vu sa tentative de suicide exposée au grand public.

De plus, comme toutes informations personnelles, ces dernières peuvent servir à faire de la publicité ciblée, précisant encore plus le profil d'une personne. Ici, nous pouvons voir comment les informations médicales peuvent influencer la vie d'une personne voire même jouer un rôle politique.

Cependant, certaines informations médicales peuvent avoir une influence plus vaste. En effet, les résultats d'un test génétique n'engagent pas seulement la personne l'ayant fait. Par exemple, certaines maladies pour lesquelles des prédispositions génétiques existent pourraient être détectées par de tels tests. En conséquence de quoi, tous les membres d'une famille pourraient être impactées.

Pour toutes ces raisons, les acteurs de la santé pourraient tirer un net avantage financier en partageant les données médicales au détriment des patients. C'est pour cela que les données doivent être sécurisées. Des lois viennent alors encadrer les dérives associées à

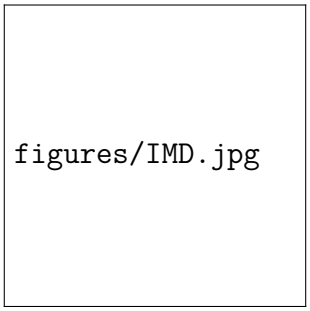


FIGURE 1.1 – Exemples d'IMD de gauche à droite : un implant cochléaire, cérébral et cardiaque.

l'usage de ces données. Ainsi l'article L1110-4-V du code de la santé publique indique que : « Le fait d'obtenir ou de tenter d'obtenir la communication de ces informations en violation du présent article est puni d'un an d'emprisonnement et de 15 000 euros d'amende. » Plus généralement l'article 226-16 dicte que : « Le fait, y compris par négligence, de procéder ou de faire procéder à des traitements de données à caractère personnel sans qu'aient été respectées les formalités préalables à leur mise en œuvre prévues par la loi est puni de cinq ans d'emprisonnement et de 300 000 euros d'amende. » Les articles 226-21 et 226-22 complètent les répressions associées aux usages malveillants de ces données.

Après avoir vu les risques qui pèsent sur ces données, regardons maintenant comment de nouveaux types d'appareils implantés permettent leurs acquisitions.

1.1.3 Les implants médicaux connectés

Les *Implantable Medical Devices* (IMD) sont des dispositifs médicaux qui sont implantés dans le corps humain. Ils servent à contrôler l'état de santé d'une personne, mais permettent aussi d'administrer un traitement. Différents types d'IMD sont présentés dans la Figure 1.1, on peut citer :

- Les stimulateurs cardiaques plus connus sous le nom de *pacemaker* [MP11].
- Les systèmes de délivrance des médicaments [PSD06] ou *Drug Delivery Systems*. Ces derniers peuvent par exemple fournir un traitement pour lutter contre le cancer [KP18], ou encore aider à contrôler le taux de glycémie [WAS03].
- Les neurostimulateurs [Kra02] peuvent par exemple effectuer une électroencéphalographie lors de phase d'épilepsie [Sal+12], qui une fois couplés à un système de délivrance de médicament, permettent de contrôler la crise.

Les IMD sont des appareils relativement récents. Le premier *pacemaker* à avoir été im-

planté à l'intérieur du corps humain date de 1958 [Lar+03]. Ces appareils étaient relativement imposants : 52.5 mm de diamètre pour 17.5 mm de largeur pour un poids total de 64.3 g [Fia88]. Depuis, ces matériels ont profité des innovations technologiques comme la miniaturisation des composants électroniques ainsi que l'augmentation de la capacité de stockage des batteries [GH91]. De nos jours, ils profitent également de fonctions sans fil [Jou13] permettant de transférer des données, voire de reprogrammer l'appareil [HD13]. Plus généralement, les IMD font partis du réseau sans fil corporel (ou *Wireless Body Area Networks*, noté WBANs) qui regroupe en plus des IMD, des appareils à l'extérieur du corps humain. Par exemple, ils permettent d'enregistrer un électroencéphalogramme [Lat+11] ou encore permettent d'effectuer diverses mesures : battements cardiaques ou température [Rus+14]. Ces entités font elles-mêmes partie du monde encore plus grand de l'internet des objets (ou IoT pour *Internet of things*) qui regroupe des milliards d'appareils [Gub+13] connectés au travers d'internet.

Le statut légal d'un dispositif médical est défini par l'Union Européenne [Con90] comme : « tout instrument, appareil, équipement, logiciel, matière ou autre article, utilisé seul ou en association, ainsi que tout accessoire, y compris le logiciel destiné par le fabricant à être utilisé spécifiquement à des fins diagnostique et/ou thérapeutique, et nécessaire au bon fonctionnement de celui-ci, destiné par le fabricant à être utilisé chez l'homme à des fins :

— de diagnostic, de prévention, de contrôle, de traitement ou d'atténuation d'une maladie,
— de diagnostic, de contrôle, de traitement, d'atténuation ou de compensation d'une blessure ou d'un handicap,
— d'étude, de remplacement ou de modification de l'anatomie ou d'un processus physiologique,
— de maîtrise de la conception

et dont l'action principale voulue dans ou sur le corps humain n'est pas obtenue par des moyens pharmacologiques ou immunologiques ni par métabolisme, mais dont la fonction peut être assistée par de tels moyens ; ».

La réglementation va plus loin en définissant comme un :

— « dispositif médical actif : tout dispositif médical dépendant pour son fonctionnement d'une source d'énergie électrique ou de toute autre source d'énergie que celle générée directement par le corps humain ou la pesanteur ; »
— « dispositif médical implantable actif : tout dispositif médical actif qui est conçu

pour être implanté en totalité ou en partie, par une intervention chirurgicale ou médicale, dans le corps humain ou, par une intervention médicale, dans un orifice naturel et qui est destiné à rester après l'intervention ; »

Pour simplifier, dans ce manuscrit, nous appellerons IMD tout implant médical composé au minimum des éléments suivants :

— Une mémoire permettant de conserver les informations de santé (*e.g.* nom et âge du patient), de fonctionnement (*e.g.* paramétrage) ainsi que les informations acquises par les différents capteurs (*e.g.* pression sanguine, rythme cardiaque).
— Un microprocesseur permettant de contrôler un traitement (*e.g.* stimulation du muscle cardiaque) mais également le cas échéant d'analyser des données (*e.g.* analyse du rythme cardiaque).
— Une batterie permettant d'alimenter l'ensemble du système sur une longue durée.

Ces quelques composants sont fortement contraints, on entend par là que chacun d'entre eux doit répondre à des caractéristiques complexes. Ainsi, chaque élément implanté dans le corps humain doit être stérilisé et biocompatible [Kot+02]. La biocompatibilité est définie par l'organisation internationale de normalisation (ISO) [ISO18] comme étant : la « capacité d'un dispositif médical ou d'un matériau à produire une réponse hôte appropriée dans une application spécifique ». En particulier, le dispositif médical ne doit pas chauffer de plus de 1 à 2 °C [Wol08] afin de ne pas perturber le métabolisme humain. En effet, une augmentation trop importante de la température induit une nécrose des tissus [See+98]. En ce qui concerne les autres caractéristiques, la batterie doit avoir une longue durée de vie. En effet, il n'est pas envisageable d'opérer un patient tous les ans voir, plusieurs fois par an afin de la remplacer. Pour voir cela, il faut prendre en compte plusieurs points tels que :

— L'effet psychologique qui pèse sur le patient : il sait qu'il doit se faire réopérer fréquemment, qu'il doit alors arrêter de travailler, aller à l'hôpital, ...
— L'aspect financier résultant des coûts inhérents à une opération. Chaque opération chirurgicale nécessite du personnel qualifié et des instruments adaptés, ...
— Les risques sanitaires que sont les opérations. Certaines études [GCL91] montrent que le taux d'infection suite à une opération chirurgicale est de 6.5% chaque opération accroît de nouveau ce risque. De plus, les auteurs de [Sch+14] ont montré que le surcoût induit par une infection était en moyenne de 43%.

La batterie doit par conséquent durer le plus longtemps possible, ne chauffer que modérément et être biocompatible. La mémoire et les microprocesseurs sont affectés par ces

contraintes énergétiques. Ils doivent tous les deux utiliser une quantité d'énergie minimale pour fonctionner afin d'allonger le plus possible la durée de vie de la batterie. Leur coût peut également être un facteur limitant, empêchant par exemple, d'avoir une quantité importante de mémoire ou de faire tourner en continu des programmes énergivores. On constate ainsi que de lourdes contraintes pèsent sur chaque composant d'un IMD.

Bien que les IMD présentent de nombreuses contraintes de fiabilité, de législation, leur nombre est en constante augmentation. Cela transparaît par la constante augmentation des parts de marché qui sont attribués aux IMD :

— 72,265 millions de dollars en 2015 [Tat16],
— 90 milliards de dollars en 2018 [Sum20],
— une augmentation prévue de 7.1% par an entre 2015 et 2022 pour atteindre 116,300 millions de dollars en 2022 [Tat16],
— des estimations à 158,8 milliards de dollars vers 2026 avec une augmentation de 7.3% par an de 2019 à 2026 [RC19],

Ces parts de marché sont majoritairement détenues par une dizaine de grandes entreprises telle que Medtronic, Johnson & Johnson ou encore GE Healthcare [Com17]. Cette augmentation peut s'expliquer par le simple fait qu'ils sauvent ou simplifient la vie des patients. Par exemple, ils permettent d'offrir une meilleure surveillance des signes vitaux d'un patient que ce soit à l'hôpital [GHH16] ou bien à domicile [ZCB16 ; PBC19]. De plus, il n'existe pas toujours d'alternative à ces IMD ou alors elles sont encore plus contraignantes pour le patient : *e.g.* injection d'insuline plusieurs fois par jour, prise d'antalgiques, ... Pour cet ensemble de faits, les IMD font maintenant partie intégrante du monde médical et vont de plus en plus se démocratiser.

Après avoir parlé des IMD et plus généralement des données médicales revenons sur leur sécurisation.

1.1.4 Obligations de sécurisation

Plusieurs propriétés de sécurité des données existent :

— La **confidentialité** : propriété d'une information qui n'est ni disponible, ni divulguée aux personnes, entités ou processus non autorisés.
— La **disponibilité** : dans des conditions données d'utilisation, la disponibilité d'un équipement E est définie comme son aptitude à être en état d'accomplir une fonction requise à un instant t donné ou pendant un intervalle de temps donné.

— L'**intégrité** : garantit que le système et l'information traitée ne sont modifiés que par une action volontaire et légitime.
— La **non-répudiation** : l'information ne peut faire l'objet d'un déni de la part de son auteur.

Ces définitions sont celles proposées par l'ANSSI (Agence Nationale de la Sécurité des Systèmes d'Information) ou l'AFNOR (Association Française de NORmalisation). [AFN88] et sont similaires à celle proposées par le NIST [Ros+20]. Les données médicales doivent être manipulées suivant ces propriétés afin d'assurer leur sécurité.

Cela nous mène à parler du concept d'attaquant d'un point de vue sécurité informatique. Un attaquant (ou adversaire) est une entité dont l'objectif est de compromettre la sécurité des données. De façon plus formelle, on distingue plusieurs types d'attaquants [Gol09] :

— L'attaquant **semi-honnête**, autrement appelé honnête mais curieux, exécute de manière conventionnelle le programme qui lui est demandé. Cependant, il va essayer à partir des informations dont il dispose (entrées/sorties, résultats intermédiaires, etc), d'accéder à des informations qui lui sont normalement défendues. En d'autre terme, il va essayer de casser la confidentialité des données sans jamais modifier activement le programme.
— L'attaquant **malhonnête** ou malicieux va quant à lui librement modifier le programme, les données qu'il transmet, etc. afin une fois encore, de casser la confidentialité des données.

On constate que dans la classification des attaquants, l'attaquant honnête mais curieux a beaucoup moins de moyens que l'attaquant malhonnête pour outrepasser la sécurité des données. Ainsi, une entité cherchant à modifier des données patient serait considéré comme un adversaire malicieux. Ce type d'attaque peut se révéler très dangereux. En effet, avec des résultats faussés, un médecin pourrait inoculer à un patient un traitement inadapté et mettre en jeu sa santé.

Après avoir vu les points de sécurité qui pèsent sur tout type de données sensibles, regardons les points supplémentaires qui pèsent sur les IMD. Commençons par définir les deux types de communication d'un IMD illustrés par la figure 1.2 :

— Les communications **montantes** représentent les données qui sont envoyées depuis l'IMD vers l'extérieur du corps humain. Elles sont généralement transmises vers un appareil permettant soit à un spécialiste, soit à un programme de les analyser.
— Les communications **descendantes** représentent les données qui sont transmises

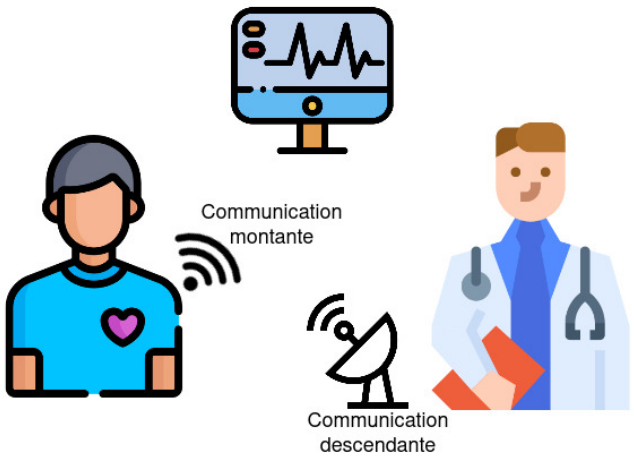


FIGURE 1.2 – Communications montantes et descendantes

vers l'IMD. Elles peuvent avoir plusieurs buts : une demande d'envoi de données, une reprogrammation, ...

Sur cette base, des contraintes supplémentaires se posent. Conformément au respect du droit à la vie privée, un patient peut ne pas souhaiter rendre public le fait qu'il soit appareillé. Ainsi, une prothèse médicale doit être **furtive**. On entend par là qu'elle ne doit pas être détectable sans autorisation préalable. Les communications montantes doivent pour ce point précis être particulièrement étudiées. En effet, en supposant que les communications montantes ne soient soumises à aucune règle, on peut penser que la prothèse transmet en continu ou périodiquement des données. Ceci permettrait à un attaquant passif d'écouter l'ensemble des communications autour de lui pour savoir si oui ou non il est en présence d'une personne portant une prothèse. Plus grave, cela pourrait permettre de tracer un individu précis afin de retrouver son domicile ou connaître ses trajets. Voici plusieurs raisons pour lesquelles les communications montantes ne doivent pas être faites en continu. Il paraît plus judicieux de demander à une personne extérieure de s'authentifier à l'IMD avant d'émettre toute information. Il est important de souligner que même un refus de connexion ne doit pas être communiqué, cela trahirait la présence de la prothèse.

Le fait d'authentifier les entités qui souhaitent entrer en contact avec l'IMD est double : il permet de conserver l'effet de furtivité pour les communications montantes, mais il permet également de s'assurer que seulement les personnes autorisées peuvent modifier ou reprogrammer l'IMD. Ainsi, dans le cas contraire, un attaquant pourrait alors faire ce qui serait considéré comme un crime "parfait". Par exemple, si l'IMD est un *pacemaker*, l'attaquant pourrait reprogrammer ce dernier de sorte qu'il envoie une décharge fatale à

son porteur tout en supprimant les données de son action. Le porteur étant déjà quelqu'un de cardiaquement affaiblie et ne présentant aucune trace de coup, coupure, poison ou autre serait alors considéré comme décédé de cause naturelle. Suite à de telles craintes, certaines personnalités telles que l'ancien vice-président des États-Unis Dick Cheney ont révélé avoir demandé à ce que la connexion sans fil de leur implant soit désactivée [PA18].

Il convient maintenant de parler des modes de fonctionnement de l'IMD. En effet, aujourd'hui, un IMD dispose en général de deux modes de fonctionnement :

— Le mode **normal** qui correspond au fonctionnement classique de l'appareil *i.e.* il envoie/réceptionne des données avec des entités qu'il aura préalablement identifiées.

— Le mode **urgence** permet quant à lui dans certaines situations d'établir une communication entre l'IMD et une entité non authentifiée. Ce mode est indispensable en cas d'accident grave menant à l'opération du porteur de l'IMD. En effet, l'accident peut avoir lieu dans un endroit géographiquement éloigné de l'hôpital d'origine du patient. Cependant, la nécessité d'intervention médicale ne doit pas être entravée par l'IMD. Ce mode permet de désactiver l'IMD ou bien de le reprogrammer afin de pouvoir opérer le patient.

Ce dernier mode très spécifique aux IMD pose de nombreux défis de sécurité. En effet, comment permettre à un médecin de désactiver une prothèse en cas d'urgence, tout en interdisant à un attaquant de pouvoir le faire ?

Pour répondre à ce scénario d'urgence, différents auteurs [Den+10] proposent de donner aux patients un mot de passe secret, inscrit sur un bracelet, un appareil électronique ou être tatoué. Cette solution présente une certaine sécurité envers les attaquants externes. En effet, l'attaquant devra accéder au bracelet afin de trouver le mot de passe. Cette solution présente néanmoins trois désavantages majeurs :

— Le dispositif contenant le mot de passe peut être perdu, cassé ou volé.

— Ces solutions imposent au malade de toujours avoir à l'esprit qu'il est sous médication. Cela a des conséquences néfastes sur l'état psychologique du patient.

— Certaines solutions telles que le tatouage posent des problèmes de sécurité, car il devient impossible de changer de mot de passe une fois celui-ci compromis.

Le tatouage possède en plus une forte connotation négative auprès du patient au vu du contexte historique auquel il fait référence (seconde guerre mondiale). Une solution proposée est d'utiliser un tatouage qui n'est visible qu'avec une lampe UV.

Il existe d'autres solutions telles que :

— Utiliser un dispositif générateur de vibration [Kim+15] couplé avec un accéléromètre implanté avec l'IMD. Les vibrations se déplaçant à une vitesse constante et en utilisant un protocole de question-réponse, l'IMD peut en déduire la proximité de l'entité. Si ce dernier se trouve à une distance prédéfinie (distance peau-IMD), un protocole d'authentification entre l'IMD et l'appareil externe se met alors en place afin d'ouvrir un canal de communication.
— Une solution similaire propose quant à elle d'utiliser des ondes ultrasoniques [Hal+08].
— Dans [LAJ11], les auteurs proposent d'utiliser le corps comme vecteur de message. L'idée est d'utiliser le corps humain pour transmettre des données. Contrairement aux communications classiques (*e.g.* ondes radio), ces communications pourraient avoir une portée limitée au corps humain. Cela a comme avantage de rendre difficile l'écoute des communications et permet d'obtenir une faible consommation électrique par rapport à celle d'une communication classique. L'article [Cha+12] de Chang *et al.* propose d'utiliser des impulsions électriques pour transférer des données dans le corps humain.
— Grâce à un électrocardiogramme, les auteurs de [Xu+11] expliquent comment parvenir à effectuer un échange de clé entre l'IMD et un appareil externe.

Toutes ces solutions supposent qu'un attaquant ne peut pas s'approcher suffisamment du patient sans éveiller ses soupçons. D'autres solutions, telles que l'alerte du patient en cas de connexion ou l'utilisation d'un bouton poussoir, existent également. Ces solutions possèdent chacune leurs avantages et inconvénients. Certains auteurs ont aussi proposé des solutions qui sécurisent les communications normales et d'urgence conjointement, voici quelques exemples :

— Dans [DFK08], les auteurs introduisent l'utilisation d'un appareil externe nommé Cloaker. Le fonctionnement est le suivant : si Cloaker est présent, l'IMD refusera toute tentative de communication avec n'importe quel autre appareil que Cloaker. Ils communiqueront alors de façon sécurisée entre eux. Dans le cas contraire, l'IMD entre dans un mode "ouvert" dans lequel il traite toutes les demandes externes.
— Un autre modèle appelé IMDGuard, présenté dans [Xu+11], utilise un appareil externe nommé Gardien. Son fonctionnement est proche de Cloaker. Quand Gardien est présent, l'IMD refuse toute communication avec d'autres appareils que Gardien. IMDGuard peut également éviter les attaques de type usurpation lors d'utilisation de brouillage en utilisant un challenge.

En s'appuyant sur un périphérique externe, ces solutions prennent en compte la contrainte

de faible puissance des IMD. Cependant, ces appareils externes ont accès en clair aux données médicales. Or, ils peuvent être hackés ou volés. Par conséquent, si un attaquant en prend le contrôle, la sécurité des données est compromise.

Maintenant que nous avons passé en revue le fonctionnement de ces appareils, regardons une autre avancée technologique qu'est l'interconnexion des appareils électroniques. Cette dernière permet une réduction des coûts de fonctionnement ainsi qu'un meilleur suivi patient ou encore de nouvelles possibilités de traitements scientifiques.

1.1.5 Externalisation et réutilisation

Afin entre autre de répondre à des problématiques de coûts de stockage élevés, mais aussi afin de faciliter la réutilisation des données, ces dernières sont externalisées. Le principe est de centraliser les données au niveau du cabinet médical, du centre de radiologie ou encore d'un hôpital. Puis, une fois celle-ci récupérées, de les renvoyer vers le *cloud*. On appelle *cloud* un réseau informatique permettant une interconnexion entre différents acteurs sur lequel plusieurs structures publiques ou privées peuvent fournir des services. Ainsi, l'externalisation des données médicales permet aux utilisateurs autorisés d'accéder aux données soit pour les consulter soit pour les étudier au travers de traitement. Dans le cadre du projet *Followknee*, qui vise le développement d'une prothèse de genou connectée, que nous prendrons comme contexte expérimental plus loin, une telle mutualisation des données est prévue. Au-delà, les initiatives de collecte et de centralisation des données de santé à des fins de réutilisation par des algorithmes sont nombreuses. On pourra, par exemple, citer le projet INSHARE [1] dont le but est de développer une plateforme « *Health Big Data* » sur le grand ouest français.

Enfin, la réutilisation des données permet, comme nous allons le voir dans la prochaine section, de traiter de façon automatique les données afin d'améliorer la prise en charge du patient et à des fins d'études scientifiques.

1.2 Traitement automatisé et aide à la décision

Comme nous avons pu le voir, les données médicales sont très diverses. De plus, de nouveaux dispositifs permettent d'accéder en continu à un nombre toujours plus important de données. Il est alors opportun de traiter deux aspects :

1. Référence projet : ANR-15-CE19-0024

1. Pourquoi ces données sont importantes dans la prise en charge d'un patient ? et donc comment les interpréter, les synthétiser, les corréler à d'autres symptômes, ... ?
2. Comment, au vu de la masse de plus en plus importante de données, arriver à les traiter en continu afin de soulager les professionnels de santé de cette tâche ?

Ainsi, nous allons commencer par voir pourquoi les médecins et autres spécialistes du corps médical se sont toujours évertués à obtenir de façon fiable le plus de données complémentaires possibles.

1.2.1 Objectifs du traitement automatisé de données

Les données permettent pour le corps médical de mieux appréhender une pathologie. Ainsi, elles peuvent être utiles en cas d'opération chirurgicale afin de cibler au mieux la zone à opérer. C'est pourquoi les professionnels de santé ont toujours cherché à augmenter la quantité et la qualité des données patient recueillies. De plus, des innovations technologiques avec principalement la miniaturisation de l'électronique permettent maintenant de traiter ces données grâce à des algorithmes. Le fait d'utiliser des algorithmes permet alors de dégager du temps au médecin. De plus, cela laisse la possibilité de traiter les données rapidement voir en temps réel afin d'aviser le patient ainsi que le corps médical le plus rapidement possible d'un changement d'état de santé. Les algorithmes ont également l'avantage d'être impartiaux ce qui n'est malheureusement pas toujours le cas chez l'être humain et par déclinaison chez le personnel médical.

Nous verrons également dans la suite que les données peuvent être traitées afin d'améliorer le diagnostic ou le traitement. Ainsi, lors d'une IRM, on pourra par exemple rechercher une masse, une échographie pourra mettre en évidence un souffle au cœur, ... On constate alors que chaque donnée permet d'analyser spécifiquement certains symptômes et que les traitements de ces données diffèrent. Des études récentes ont montré que, même pour des tâches complexe, des algorithmes arrivent aux mêmes performances qu'un praticien en première ou seconde lecture d'un examen d'imagerie [Fle+19].

Enfin, il est important de noter qu'une quantité importante de données permet d'effectuer du *big data* et ainsi pouvoir en recoupant des informations diverses arriver à trouver la cause de la survenue d'un trouble, des similitudes entre symptômes, ...

Nous allons commencer par présenter des traitements simples qui, par exemple, permettent de surveiller les constantes vitales d'un patient. Puis, nous continuerons par des

traitements plus évolués profitant d'avancées relativement récentes.

1.2.2 Les traitements de type linéaire avec seuillage

L'utilité d'un traitement de données est, par exemple, d'indiquer au patient d'aller consulter son médecin dans le cas où ces signaux sont anormaux. La détection de telles anomalies repose le plus souvent sur une opération de filtrage des signaux, qui mettra en évidence ces anomalies, et une opération de seuillage qui permettra de décider si oui ou non une anomalie est présente. On donne pour exemple le pH (potentiel Hydrogène) qui traduit l'acidité d'une solution. Il est notamment utilisé dans le contexte du projet *Followknee* pour évaluer les risques d'infection du genou suite à la pose de la prothèse et qui pourraient être à l'origine de la ré-opération du patient avec les risques que cela suppose. L'idée en filtrant la valeur du pH est d'essayer de détecter de façon précoce une infection. Dans le cas où une telle infection est détectée, le patient peut consulter en urgence son médecin qui peut lui prescrire un traitement adéquat.

Comme nous l'avons vu précédemment, d'autres traitements peuvent être intéressants tels que ceux permettant la segmentation d'images. Toutefois, on comprend que ce type de traitement ne peut pas être effectué par un simple filtrage suivi d'un seuillage. Pour ce faire, il faut utiliser des solutions de *machine learning* comme nous le présentons dans la prochaine section.

1.2.3 Les avancées du *machine learning*

Le *machine learning* [Ver+20] (ou apprentissage automatique) regroupe un ensemble de techniques et d'algorithmes permettant à un ordinateur d'"apprendre" à effectuer une tâche à partir d'un ensemble de données transmis au modèle de *machine learning*. Voici quelques objectifs que ces algorithmes cherchent à compléter [Kwo+19] :

- **La classification** permet d'attribuer une classe (ou un label) à des données d'entrée. Un exemple d'une telle application est de donner en entrée des images à l'algorithme qui devra définir si elles représentent un chien ou un chat.
- **Le partitionnement de données** (ou *data clustering*) permet de diviser un ensemble de données en différents groupes ayant des caractéristiques communes. Ce type d'algorithme peut, par exemple, servir à segmenter une image médicale dans le but d'identifier des régions d'intérêt.
- **La détection d'anomalie** est capable d'extraire des données celles qui diffèrent

sensiblement des autres. Un tel algorithme a notamment un intérêt dans le cas de recherche de fraude bancaire.

— **Le *feature learning*** pour lequel l'objectif est de trouver une représentation adaptée des données. Ce type de procédé peut être utilisé sur les données en amont d'algorithme de classification pour, par exemple, augmenter ses performances.

— **La réduction de la dimensionnalité** consiste à réduire la dimension des données d'entrée. Ce type d'algorithme permet de seulement sélectionner les données pertinentes.

Cette liste bien que non exhaustive représente une majorité des problématiques associées au *machine learning*. Dans la suite de cette thèse, nous nous intéresserons principalement à la classification.

Afin d'atteindre leurs objectifs, ces algorithmes passent par deux phases :

1. **L'apprentissage** ou l'entraînement durant lequel l'algorithme apprend à réaliser la tâche qui lui est attribuée en s'appuyant sur un grand nombre d'exemples.
2. **L'inférence**, aussi appelée phase de mise en production, consiste à utiliser un modèle entraîné et lui donner de nouvelles données à traiter.

En ce qui concerne l'apprentissage, il existe également plusieurs types :

— L'apprentissage **supervisé** utilise une base de données d'apprentissage pour laquelle le résultat de la tâche à effectuer est connu à l'avance. Le modèle utilise alors cette base de données pour "apprendre" à réaliser sa tâche.

— L'apprentissage **non supervisé** quant à lui ne possède pas de base de données pour laquelle il connaît le résultat de la tâche à effectuer. Le modèle doit alors apprendre seul à distinguer les structures sous-jacentes lui permettant d'effectuer sa tâche.

— L'apprentissage **semi-supervisé** possède une petite base de données pour laquelle il connaît le résultat de la tâche à effectuer. Son objectif est alors d'allier les deux modes d'apprentissage **supervisé** et **non supervisé**.

— L'apprentissage **par renforcement** consiste à récompenser (réussite de la tâche) ou pénaliser (échec de la tâche) le modèle au cours de son utilisation.

Ces quelques types sont les plus communs. Nous nous attarderons principalement sur l'apprentissage supervisé.

Maintenant que nous avons présenté les objectifs et modes de fonctionnement de ces algorithmes, regardons quelques exemples de tels modèles :

— La **méthode des k plus proches voisins** [FH89] consiste à attribuer une classe à une donnée en fonction des k données d'apprentissage qui lui sont le plus proche.
— Les **réseaux de neurones artificiels** [Wu+15] ont un fonctionnement inspiré des neurones biologiques. Ils prennent en entrée des données qui passent au travers de couches de neurones pour finalement fournir une réponse à la tâche qui leur a été affectée.
— Les **réseaux neuronaux convolutifs** [FM82] sont inspirés des neurones du cortex visuel. Le fonctionnement interne diffère de celui des réseaux de neurones artificiels de par le fait qu'ils utilisent des convolutions. Cependant, la finalité reste identique.
— Les **auto-encodeurs** [HS06] ont pour but d'apprendre une représentation des données d'entrée, généralement dans un espace de dimension inférieure.
— Les **arbres de décision** [SL91] sont composés d'une série de branches connectées par des nœuds et terminées par des feuilles qui correspondent chacune à une réponse sur la tâche à effectuer.

Dans cette thèse, nous nous intéresserons principalement aux réseaux de neurones artificiels (notés NN pour *Neural Networks*) et aux réseaux neuronaux convolutifs (notés CNN pour *Convolutional Neural Networks*) sur lesquels nous revenons plus longuement dans la suite de ce manuscrit. Nous reviendrons également brièvement sur les arbres de décision.

Il est important de noter que certaines de ces méthodes, en particulier, celles des NN et CNN permettent de traiter de façon rapide et efficace les données. Dès la fin des années 80, plusieurs auteurs [LeC+89 ; Den+89] ont montré comment il était possible à l'aide d'un réseau de neurones de lire des codes postaux écrits à la main. Par la suite, le perceptron multicouche ou *multilayer perceptron* a permis dès la fin des années 90 d'améliorer certains résultats obtenus par des techniques classiques d'analyse de données [GD98]. En 2012, le concours ImageNet dont le but consiste à attribuer a des images leur classe correspondante a été remporté par un algorithme de *machine learning* [KSH12]. Plus récemment dans le domaine médical, certains auteurs [Fle+19] ont proposé des méthodes de *machine learning* dont le résultat de la classification était dans certains cas plus performant qu'une classification faite par un spécialiste [Fle+19].

Ce sont donc ces algorithmes que nous avons cherchés à sécuriser. Nous ne sommes pas les premiers à nous intéresser à la sécurisation de tels traitements et de leurs données, différents outils et mécanismes ont été proposés : Chiffrement *fully* homomorphe [Cha+17], tatouage [Nag+18], ... Nous les décrivons ci-après pour mieux positionner l'originalité de nos travaux présentés dans les différents chapitres de ce manuscrit.

1.3 Sécurisation des traitements de données

Comme nous l'avons montré les données médicales sont sensibles. De plus, nous souhaitons pouvoir les traiter afin, entre autres, d'aider le médecin dans sa prise de décision. En conséquence de quoi, dans cette partie, nous allons introduire la notion de chiffrement qui permet d'assurer la confidentialité des données. Plus simplement, ce type d'algorithme rend inintelligible les données pour toute personne n'étant pas autorisée à y accéder (en particulier les attaquants). Parmi les algorithmes les plus connus, on retrouve le *Data Encryption Standard* (DES) [Dav78] ou encore l'*Advanced Encryption Standard* (AES) [DR01] qui sont deux algorithmes de chiffrements symétriques (autrement appelé à clé secrète). Cela signifie que les clés nécessaires pour effectuer le chiffrement et le déchiffrement sont identiques. Ces algorithmes ont l'avantage d'être peu coûteux, cependant, ils ne permettent pas le traitement de données chiffrées. D'autres cryptosystème permettent d'assurer la confidentialité des données tout en permettant leur traitement au détriment de coûts de calcul et de communications supplémentaires. Comme nous le verrons plus loin, un de nos objectifs a été d'allier ces différents types de chiffrement afin de profiter des avantages de chaque chiffrement tout en s'affranchissant de leurs contraintes. Pour ce faire, commençons par parler d'un type de chiffrement particulièrement peu coûteux.

1.3.1 Chiffrements rapides et peu coûteux

Comme précisé en introduction de cette partie, les chiffrements à clé secrète font généralement partie des cryptosystèmes les plus rapides et moins coûteux permettant la sécurisation des données. Parmi ces algorithmes, on retrouve deux sous-types de chiffrements :

— Les chiffrements pas blocs tels que Prince [Bor+12], PRESENT [Bog+07] ou encore HIGHT [Hon+06]. Ces derniers, comme leurs noms l'indiquent, chiffrent un message bloc par bloc. Pour ce faire, ils découpent le message d'entrée en blocs de même longueur comprise entre 32 et 512 bits. Ces blocs sont ensuite chiffrés les uns après les autres.

— Les chiffrements à flots tels que Snow [EJ03] Grain [HJM07], ou bien Salsa20 [Ber05]. Contrairement au chiffrement par blocs, ces derniers chiffrent les messages à la volée, c'est-à-dire qu'ils n'ont pas besoin de les découper en blocs.

Une synthèse d'un certain nombre de protocoles de chiffrement standardisés par les acteurs du secteur a été proposée par Biryukov et Perrin dans [BP17].

On note que ces chiffrements ne souffrent généralement pas d'expansion de chiffré, *i.e.* ratio de l'augmentation de la taille du message chiffré par rapport au message en clair. Ainsi, un facteur d'expansion de 2 signifie que le chiffré est de taille deux fois supérieure au message en clair. Plus ce facteur est important, plus les coûts de communication augmentent.

Ces algorithmes permettent d'assurer la sécurité des données, mais ne permettent pas leur traitement sécurisé. On entend par traitement sécurisé toute opération que l'on pourrait effectuer sur une donnée chiffrée dont l'incidence sur la donnée en clair est connue. La seule solution pour traiter les données avec ce type de chiffrement est de les déchiffrer, de faire le traitement puis de les re-chiffrer. Or, une fois les données disponibles en clair, elles deviennent vulnérables. Une solution pour palier à ce problème consiste à utiliser des chiffrements dits homomorphes que l'on présente dans la section suivante.

1.3.2 Traitements sur données chiffrées

Le fait de vouloir faire un traitement sécurisé n'est pas nouveau. En 1982 déjà, Yao a posé le problème des millionnaires [Yao82]. Cette expérience de pensée est constituée de deux protagonistes Alice et Bob qui sont millionnaires, leur but est de savoir lequel d'entre eux est le plus riche sans pour autant que l'autre ne connaisse la fortune dont il dispose. Comme nous le verrons plus en détails, des solutions ont été apportées à ce problème en 1986 avec les *garbled circuits* [Yao86]. Cependant, ce type de solution n'est pas satisfaisant en termes de coûts de calcul et de communication. Plusieurs autres solutions ont été présentées jusqu'à l'arrivée des chiffrements homomorphes. Ces cryptosystèmes sont asymétriques ou à clé publique. Cela signifie qu'une clé est réservée au chiffrement et une autre au déchiffrement. On distinguera trois types principaux de chiffrements homomorphes :

- Les **additifs** qui permettent à partir de deux messages chiffrés de calculer le chiffré de leur somme sans jamais devoir les déchiffrer ni compromettre leur sécurité.
- Les **multiplicatifs** qui à l'instar de l'additif permettent cette fois-ci la multiplication sous les mêmes conditions.
- Les ***fully*** qui permettent les deux opérations à la fois.

Les additifs et multiplicatifs sont appelés *Partially Homomorphic Encryptions* (PHE) et les *fully* sont abrégés FHE. Chaque type de chiffrement possède ses avantages et inconvénients : plus ou moins rapide, permettent ou non d'effectuer les opérations dans le domaine chiffré de façon illimitée, ... Cependant, ils ont tous au moins deux caractéristiques

en commun. Pour commencer, ces algorithmes sont beaucoup plus coûteux en termes de calcul que les algorithmes de chiffrement symétrique. De plus, ils souffrent d'une expansion de chiffré avec, pour conséquence, l'augmentation des coûts de communication. Il est intéressant de voir que pour des raisons de sécurité, l'espace des données en clair avant chiffrement doit être grand. Du fait que la taille du chiffré est indexée sur celle du clair, une perte en termes de coûts de communication se produit du fait que tout l'espace du clair n'est pas utilisé. Ces problématiques occuperont une part importante de cette thèse. Ainsi, nous étudierons comment utiliser les différents types de chiffrement qui existent pour ne profiter que de leurs avantages (*e.g.* rapidité, faible coût de communication, possibilité de traitement) sans leurs inconvénients.

Dans la partie suivante, nous abordons la troisième propriété centrale de cette thèse, à savoir la compression des données qui, comme nous l'avons vu, permet de réduire les coûts de stockage. Cependant, les traitements qu'on leur fait subir imposent de les décompresser d'où la nécessité de voir comment allier ces différentes problématiques.

1.4 Optimisation du stockage et transfert des données

La compression permet de réduire l'espace de stockage des données. Elle est particulièrement intéressante dans le cadre des images. Un rapide calcul prenant en compte une image en couleur composée de trois canaux RVB (rouge, vert, bleu) en 4k soit 3 840 $\times$ 2 160 pixels nécessite presque 25 Mo de stockage avant compression. Dans le cas d'une vidéo de même résolution, cadencée à 24 images par seconde, il faut un disque d'un To pour en stocker 30 minutes. Grâce à la compression un tel disque pourrait contenir plusieurs dizaines de films. La compression prend alors tout son sens. De plus, comme précisé à la Section 1, le domaine médical utilise lui aussi un grand nombre d'images issues de scanner, IRM, ... Il est alors intéressant de constater que les traitements que nous avons présentés à la Section 1.2 fonctionnent sur des données non compressées. Or, nous venons de voir que pour des raisons de stockage, mais également de coût de communication, les données sont compressées. C'est pourquoi nous souhaitons investiguer plus en détail les phases de la compression ainsi que les types de traitement pour étudier comment les allier. Ceci permettrait alors soit de ne pas décompresser les données avant d'en effectuer le traitement soit de seulement les décompresser partiellement. Les coûts et temps de calcul s'en verraient alors diminués.

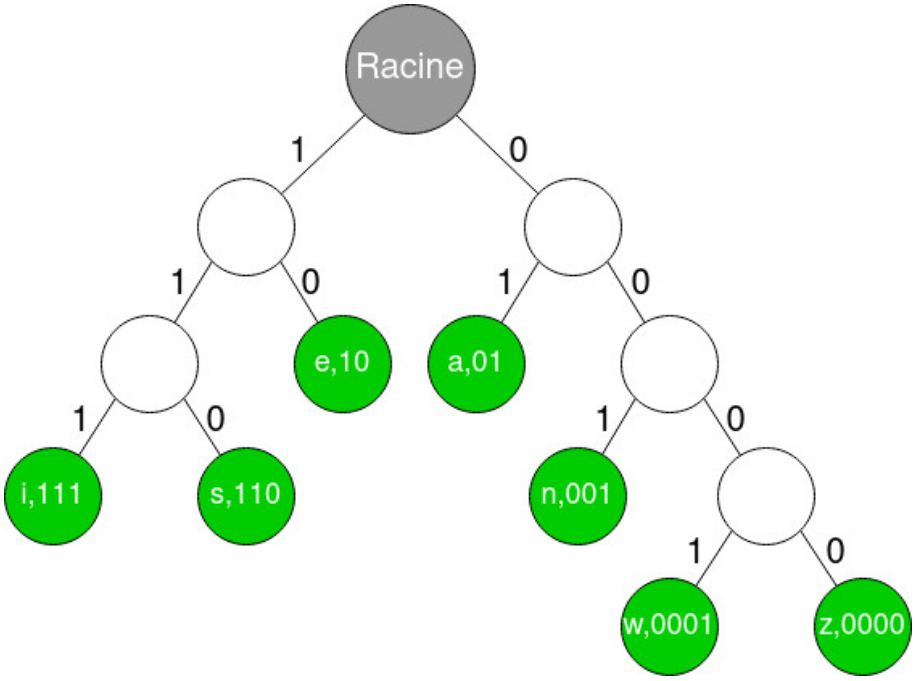


FIGURE 1.3 – Exemple d'arbre de Huffman.

Il existe deux grands types de compression :

— La compression **sans perte** qui permet de reconstruire à l'identique l'image originale après décompression. Ce type d'algorithme permet de s'assurer qu'aucun artéfact n'apparaîtra sur l'image résultante de la décompression. Il est cependant moins performant en termes de taux de compression. On peut citer le codage de Huffman ou encore *Portable Network Graphics* (PNG) qui appartient à cette classe.
— La compression **avec perte** qui permet généralement d'atteindre des taux de compression supérieurs au détriment de la qualité. En effet, ce type d'algorithme ne préserve pas l'image originale. Autrement dit, l'image reconstruite différera de l'originale ce qui engendrera une perte de qualité. Ces algorithmes, par exemple MPEG, MP3, JPEG ou encore JPEG2000 essaient généralement de limiter au maximum la perte de qualité.

Dans la suite, nous commencerons par nous intéresser au codage de Huffman, un codeur entropique sans perte, avant de présenter succinctement les principes de la compression JPEG.

1.4.1 Compression optimale de Huffman

Le codage de Huffman a été présenté en 1952 par David A. Huffman [Huf52]. C'est un algorithme de compression sans perte qui est intégré à de nombreux standards de compression très répandus tels que JPEG [Wal92] ou MP3 [Kar99]. De façon synthétique, ce codage s'appuie sur la connaissance de la probabilité d'apparition des symboles. À partir de ces probabilités, il attribue à chaque symbole à compresser un mot de code dont la longueur dépend de la fréquence d'apparition du symbole. Ainsi, il attribue à un symbole apparaissant fréquemment un mot de code de faible longueur et inversement pour un symbole peu fréquent. De cette manière, la longueur moyenne des mots codes tend vers l'entropie qui, comme définie par Shannon [Dud13], établit le nombre de bits optimal permettant de coder les symboles d'une source d'information. À noter que les mots de code attribués ne sont jamais racine d'autre mot de code. Ceci permet de représenter le codage de Huffman par un arbre comme proposé à la Figure 1.3. Dans cet exemple fictif, la fréquence des lettres représente leur fréquence d'apparition dans la langue française : "e" 12.1%, "a" 7.11%, "i" 6.59%, ... On constate que le "e" et le "a" qui sont les deux lettres les plus communes ont les deux mots de code les plus courts "10" et "01". À l'inverse, le "w" et le "z" qui sont peu fréquents ont eux les deux mots de code les plus longs "0001" et "0000".

Nous reviendrons plus en détail sur cet algorithme et la façon d'attribuer les mots de code aux symboles plus loin dans ce manuscrit. Cependant, il est important de retenir que cet algorithme de compression apparaît dans l'algorithme de compression d'image le plus répandu à savoir JPEG.

1.4.2 JPEG - un standard de compression pour les images

Le standard JPEG, présenté en 1992, est le standard qui reste le plus utilisé actuellement [Hud+18]. Son fonctionnement schématique est le suivant :

1. Il divise les images en blocs de taille 8×8.
2. Il applique la DCT (*Discrete cosinus transform*) sur chacun des blocs.
3. Les coefficients DCT d'un bloc sont quantifiés indépendamment les uns des autres.
4. Les coefficients quantifiés sont compressés sans perte en utilisant le codage de Huffman par exemple.

Cet algorithme a l'avantage d'être rapide et permet de moduler le taux de compression

de l'image en influant sur le facteur de qualité qui entre en jeu lors de la phase de quantification.

JPEG est un des standards de compression d'image accepté par le standard Digital [MDG08] *Digital Imaging and COmmunications in Medicine* (DICOM), le standard de référence en imagerie médicale. DICOM n'est pas qu'un simple standard de stockage d'images, il stipule également comment des systèmes d'imagerie médicale compatibles DICOM peuvent échanger des images ou proposer des services pour manipuler les images (*e.g.* affichage sur des stations de diagnostic). On peut tout de même noter qu'il existe d'autres algorithmes de compression d'images tels que JPEG 2000 [TM02] ou PNG [Bou97] dont l'utilisation est plus minoritaire ou spécifique.

Notre objectif sera alors de voir s'il est possible de concilier la compression JPEG avec du *machine learning* afin d'éviter la phase de décompression et ainsi se prémunir de coûts de calculs superflus au détriment d'autres critères. Nous étudierons également s'il est possible de sécuriser à l'aide de fonction de chiffrement homomorphe la décompression JPEG afin d'avoir une chaîne permettant de :

1. Compresser les données.
2. Chiffrer les données puis les externaliser.
3. Récupérer les données chiffrées sur le *cloud* puis les décompresser de façon sécurisée pour les traiter toujours de façon sécurisée.

Avant de passer à la conclusion, il est intéressant de revenir sur un point évoqué à la Section 1.3.2, à savoir que la taille des données à chiffrer doit être importante pour des raisons de sécurité.

1.4.3 Compression et chiffrement de données

Dans la Section 1.3.2, nous avons exprimé le fait que les algorithmes de chiffrement homomorphe demandent en entrée des données de taille importante. Or, les données sur lesquelles nous travaillons ont généralement une faible longueur. L'idée, qui sera expliquée de façon formelle plus loin dans ce manuscrit, est de concaténer les données les unes à la suite des autres comme représenté schématiquement Figure 1.4 tout en préservant la capacité de traiter les données. Comme nous le verrons dans sa section dédiée, cette technique permet de gagner en termes de coût de communication. Cependant, elle limite les possibilités de traitement sécurisé que l'on peut effectuer sur les données. Cette technique reste tout de même très avantageuse. Nous verrons également plus loin dans ce manuscrit

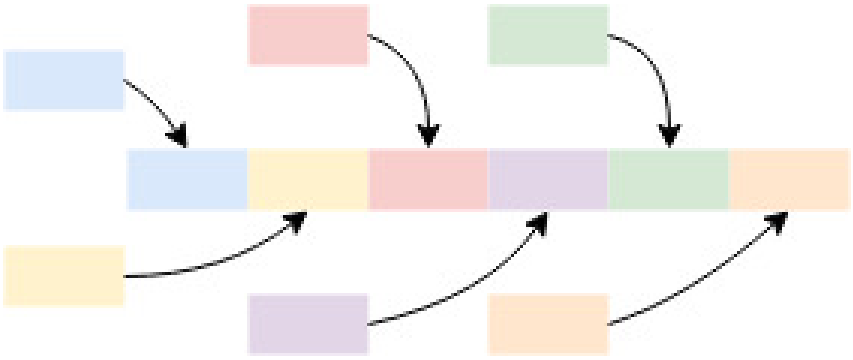


FIGURE 1.4 – Représentation schématique de la concaténation.

que les chiffrements *fully* homomorphes (FHE) permettent, eux aussi, de faire du SIMD pour *Simple Instruction Multiple Data* qui permet alors d'appliquer en une seule fois une même opération sur plusieurs données.

Dans cette partie, nous avons parlé des différents types de compression que nous allons appliquer à nos données. Passons maintenant à la conclusion de ce chapitre dans laquelle nous revenons sur notre contexte et les enjeux que l'on souhaite traiter dans ce manuscrit.

1.5 Conclusion

Tout au long de ce chapitre, nous avons essayé de situer le contexte d'étude de cette thèse. Pour commencer, nous avons défini et caractérisé les données, et plus particulièrement, celles de santé. À savoir, comment elles sont obtenues et dans quel but. Cela nous a mené à étudier les IMD qui sont une innovation médicale permettant d'améliorer le traitement et le suivi d'un patient. Nous avons précisé que les données personnelles sont des données sensibles et qu'à ce titre leur usage est réglementé par des lois nationales et internationales, avec des sanctions pénales et financières importantes (cf. RGPD). En effet, nous avons montré qu'un attaquant mal intentionné pourrait tirer plusieurs bénéfices en ayant la connaissance de ses données. Il est alors apparu que les données médicales font partie des données personnelles et bénéficient des mêmes droits. Par la suite, il est apparu que pour des raisons de réduction des coûts de maintenance informatique, les données sont de plus en plus externalisées. Cette externalisation permet également à des acteurs physiquement éloignés de pouvoir accéder rapidement aux informations patient ce qui facilite la prise en charge. De plus, le fait de réunir un grand nombre de données permet aux scientifiques de faire des analyses *big data* afin de pouvoir identifier la cause d'une

maladie ou d'éventuels effets indésirables d'un traitement par exemple. Ce constat nous a alors poussé à nous intéresser au traitement que l'on peut envisager sur de telles données. Nous avons en particulier présenté deux types de traitements. Les premiers portent sur le filtrage et le seuillage de données. Ce sont là des opérations essentielles en traitement du signal, notamment pour la détection d'anomalies. Le second type de traitement concerne les algorithmes d'apprentissage automatique qui sont d'importance pour le développement de système d'aide au diagnostic fondée sur la réutilisation de données externalisées.

Cependant, il est primordial de se rappeler que les données médicales sont sensibles. C'est pourquoi nous nous sommes intéressés dans la suite aux possibilités existantes en ce qui concerne la sécurisation des données et leurs traitements associés. Nous avons alors distingué plusieurs types de chiffrement. Des chiffrements légers, par bloc, qui permettent ou non de traiter les données une fois celle-ci chiffrée, ... Un enjeu ici est de voir comment il est possible de profiter des avantages de chaque type de chiffrement tout en s'affranchissant de leurs inconvénients. Dans ce sens, on souhaite pouvoir traiter les données une fois chiffrées, tout en diminuant autant que possible les coûts de communication et de calcul. Ceci est particulièrement vrai dans le contexte des IMD ou les ressources disponibles sont très limitées. Pour finir, un dernier constat sur les données ainsi que sur les traitements est apparu : les données, pour des raisons de stockage, sont en général disponibles sous forme compressée. Or, les traitements effectués fonctionnent sur des données non compressées. Pouvoir traiter les données directement sous leur forme compressée donne l'espoir de pouvoir gagner en temps de calcul : il ne serait plus nécessaire de décompresser les données. C'est une question largement ouverte d'autant plus si on souhaite réaliser ces opérations de manière sécurisée. Nous avons également vu que l'utilisation d'algorithmes de chiffrement homomorphes implique de travailler sur des espaces de messages ou de données en clair de très grande taille (cf. Section 1.3.1). Les données de santé étant de bien petite taille par rapport à ces contraintes, il est tout naturel de s'interroger sur des stratégies de "compression" de données et exploiter l'espace des données en clair avant chiffrement d'un cryptosystème homomorphe tout en préservant les capacités de traitement de ce cryptosystème. Ces sont là les questions sur lesquelles nous avons travaillé ces trois dernières années.

Commençons par étudier le cas de la sécurisation d'un traitement avec des données issues d'un IMD.

Chapitre 2

Traitement de données concaténées chiffrées

Dans ce chapitre, nous nous intéressons à la sécurisation d'opérations relatives au projet *Followknee*. Ce projet consiste à implanter chirurgicalement une prothèse de genou à des patients dont l'articulation n'est plus ou peu fonctionnelle. Un tel dysfonctionnement peut intervenir dans des cas d'arthrose, de polyarthrite rhumatoïde, de surpoids ou encore d'accident [VI91]. Dans ce contexte, *Followknee* est une prothèse de genou qui rentre dans la catégorie de *Implantable Medical Devices* (IMD) [Kha+14b]. Les IMD, définis dans la Section 1.1.3, doivent :

— remplir leur rôle médical ;
— être biocompatibles, peu contraignants, volumineux et visibles ;
— avoir une durée de vie conséquente et être fiables.

On souhaite, grâce à une connexion sans fil, pouvoir communiquer puis traiter des données issues de cette prothèse. Cependant, une telle communication doit être extrêmement sécurisée, comme discuté dans la Section 1.1.4.

Comme présentées lors de la Section 1.1.4, des solutions existent. Cependant, elles reposent sur des appareils externes qui sont une source de vulnérabilité. De plus, ces appareils utilisent des solutions basées sur la cryptographie symétrique qui ne permet pas de faire un traitement sur les données. C'est pourquoi dans ce qui suit nous allons présenter une solution qui permet de sécuriser à la fois les communications et les traitements.

Le scénario, considéré dans le projet *Followknee* et que nous avons étudié dans cette thèse, est présenté dans la Figure 2.1. Il se compose de trois entités : un IMD, une IHM (Interface Homme-Machine) ici un appareil mobile comme un smartphone ou une tablette [Thi+14]) et un Serveur. Chaque entité a un rôle bien défini :

— L'IMD collecte et transmet les données de santé jusqu'à l'IHM.
— L'IHM traite les données en collaboration avec le Serveur.
— Le Serveur stocke les données dans le but de les utiliser plus tard, par exemple,

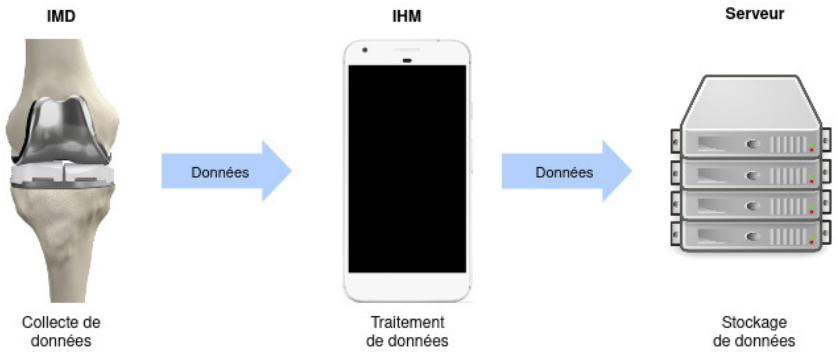


FIGURE 2.1 – Scénario avec IMD, IHM et un Serveur à sécuriser.

pour du *Big Data* [Bou+17].

L'IHM au travers de son traitement sécurisé essaye de savoir si le patient est en danger ou non. Nous allons supposer que l'IMD envoie des données de santé par exemple des données de pH (Potentiel Hydrogène). Toutefois, ces données peuvent révéler un problème de santé tel qu'une infection et ainsi éviter une nouvelle opération avec les risques associés c.f. Section 1.1.3. L'IHM peut permettre un diagnostic précoce d'infection. Ainsi, le patient peut aller consulter rapidement un médecin qui pourra lui fournir un traitement adéquat. Dans ce contexte, nous ne voulons pas que l'IHM ait accès aux données de santé. Puisque par sa nature, un smartphone est l'élément le plus faible en termes de sécurité dans la chaîne de transmission des données, dans le sens où il peut être volé ou hacké.

Dans ce chapitre, nous commencerons par présenter les solutions de sécurité que nous utiliserons pour sécuriser le projet *Followknee*. Par la suite, nous montrerons comment passer d'un cryptosystème léger à un cryptosystème permettant de sécuriser des traitements afin d'optimiser les coûts de calculs et de communication. Nous présenterons en plus un protocole permettant d'assurer l'intégrité des données. Une fois tous ces outils présentés, nous les appliquerons à notre scénario afin de le sécuriser. Nous finirons par établir la sécurité de notre solution après avoir étudié ses performances.

2.1 Outils de sécurisation et de traitement de données

L'objectif est de sécuriser le scénario présenté Figure 2.1. Ainsi, il faut sécuriser les communications qui partent de l'IMD jusqu'à l'IHM ainsi que celle partant de l'IHM jusqu'au Serveur. De plus, on souhaite que l'IHM puisse faire un traitement sur les données.

Ici, nous allons vouloir effectuer un filtrage suivi d'un seuillage. Pour ce faire, prenons M données $\{d_i\}_{i=0,\dots,M-1}$ que l'on cherche à traiter, $\{w_i\}_{i=0,\dots,M-1}$ les poids des filtres et s le seuil on cherche alors à calculer :

$$A = \sum_{i=0}^{M-1} (w_i d_i) - s \tag{2.1}$$

Ce type de traitement permet, par exemple, d'étudier si sur un temps donné, la moyenne des valeurs est en dessus ou en dessous d'un seuil.

Dans notre scénario, l'IMD qui collecte les données est considéré comme une entité de confiance. Ensuite, l'IMD les transmet de manière sécurisée à l'IHM. Cette dernière n'est pas considérée de confiance : elle est considérée comme un adversaire honnête mais curieux. L'IHM doit tout de même être capable d'effectuer un filtrage suivi d'un seuillage avec les données fournies par l'IMD, et cela, de manière sécurisée. L'IHM ne doit pas accéder aux données de l'IMD en clair. Comme nous allons le voir, une solution efficace et relativement peu coûteuse pour cela est d'utiliser un chiffrement à clé publique additivement homomorphe. Cependant, bien que les coûts de ces algorithmes soient maîtrisés, ils restent bien trop importants pour l'IMD. Pour pallier ce problème, nous allons présenter une solution permettant d'utiliser conjointement un chiffrement léger, qui assure la sécurité lors du transfert de l'IMD à l'IHM, et d'un chiffrement additivement homomorphe permettant à l'IHM de traiter de façon sécurisée ces données. Enfin, le Serveur dans notre scénario est aussi considéré comme un adversaire honnête mais curieux et a un triple rôle :

1. Il est celui qui va stocker les données de santé transmises par l'IMD au travers de l'IHM.

2. Il va aider l'IHM à obtenir le résultat du traitement sécurisé.

3. Il va pouvoir dans un contexte plus avancé où l'IHM est considérée comme malicieuse, vérifier si cette dernière effectue correctement ou non les traitements.

Maintenant que nous avons vu le rôle de chaque entité, regardons les solutions présentes dans la littérature pour calculer de façon sécurisée une comparaison. En effet, un seuillage n'est rien d'autre qu'une comparaison de deux valeurs entre elles.

2.1.1 Traitement sécurisé au travers du chiffrement additivement homomorphe

2.1.1.1 Définitions et premières propositions

Regardons plus concrètement ces algorithmes que nous avions brièvement décrits dans la Section 1.3.2. L'idée derrière les *garbled circuits* de Yao [Yao86] est de décrire par un circuit booléen à deux entrées l'opération à sécuriser. Alice chiffre et brouille le circuit booléen avant de l'envoyer à Bob. Puis, grâce à l'utilisation d'un algorithme d'*oblivious transfer* [Rab81] (ou transfert inconscient), Bob récupère son entrée chiffrée auprès d'Alice. Grâce à cela et quelques autres communications, Bob peut récupérer de façon sécurisée la sortie du circuit binaire qu'il communique à Alice. Ainsi, les deux connaissent l'évaluation de leur problème, mais pas les données d'entrée de chaque participant. Le problème est que cette solution est onéreuse en termes de coût de calcul et d'envoi. D'autres solutions plus récentes [Cac+00] proposent également de tirer parti des *garbled circuit* ainsi que de l'*oblivious transfer* mais restent relativement coûteuses en termes de complexité de communication.

Avec le développement du chiffrement homomorphe, certains auteurs ont proposé d'autres solutions. Avant de voir cela, revenons sur la définition du terme homomorphe. On entend par là qu'il existe une opération dans le domaine chiffré notée $\square$ telle qu'elle induise sur les données en clair une autre opération notée $\circ$:

$$E[a] \square E[b] = E[a \circ b] \tag{2.2}$$

où $E[\cdot]$ représente la fonction de chiffrement. Ce genre de fonction de chiffrement n'est pas la norme. En effet, les fonctions classiques de chiffrements symétriques telles que le (DES) ou encore (AES) introduit à la Section 1.3 ne tolèrent pas de telles opérations. Le premier cryptosystème présentant de telles propriétés est le cryptosystème RSA [RSA78] :

$$E[a] \times E[b] = a^e \times b^e \mod n = (a \times b)^e \mod n = E[a \times b] \tag{2.3}$$

où e est l'exposant et n le module, tous deux publics. Pour être plus précis, le cryptosystème est multiplicativement homomorphe. Autrement dit, il existe une opération à effectuer (ici la multiplication) entre deux chiffrés pour obtenir le résultat chiffré du produit des clairs. D'autres cryptosystèmes partagent cette propriété tel que celui d'ElGamal [Elg85]. Bien que ce type de cryptosystème soit intéressant, il ne nous permettra

pas de calculer une comparaison de façon sécurisée. Ce qui n'est pas le cas du chiffrement additivement homomorphe. Comme leur nom l'indique, ces derniers permettent, suite à une opération effectuée sur deux chiffrés, d'obtenir le résultat chiffré de la somme de leur clair :

$$E[a]\square E[b] = E[a+b] \tag{2.4}$$

On peut citer le cryptosystème de Benaloh [Ben94] qui est basé sur le cryptosystème de Goldwasser–Micali [GM84] lui-même homomorphe pour la fonction XOR (fonction "ou exclusif"). Il y a également celui de Paillier [Pai99] ou encore son extension faite par Damgård–Jurik [DJ03]. Concernant les cryptosystèmes *fully* homomorphes (FHE) qui permettent à la fois d'effectuer des additions et des multiplications, il faut attendre 2005 avec [BGN05] pour obtenir la première solution qui cependant, ne permettait de faire qu'un nombre limité de multiplications. Et 2009, pour une solution complètement homomorphe grâce à Gentry [Gen+09].

Certains auteurs [BK04] proposent d'utiliser le chiffrement additivement homomorphe ou les résidus quadratiques [Fis01] précédemment introduits par Goldwasser et Micali afin de comparer deux nombres. Ce type de solution n'est pas adaptée à notre problème du fait qu'elle demande un nombre de chiffrements important. De plus, elle demande à ce que l'IHM connaisse soit directement les données à comparer en clair soit la clé privée qui permet de les déchiffrer. Rappelons que nous ne voulons à aucun moment que l'IHM ait accès à ces données.

Enfin, l'article de Kerschbaum, Biswas and Hoogh (KBH) [KBH09] propose d'utiliser le chiffrement additivement homomorphe et le concept de masquage de données pour comparer deux nombres. Le principe du masquage est le suivant : on suppose que $\mathbb{Z}_n$ est l'espace des clairs pour le cryptosystème utilisé. Pour masquer une donnée d de taille l bits, il suffit de tirer aléatoirement deux nombres r et r' de longueur k bits tels que $r' < r$ qui vérifient :

$$log_2(n) > k + l + 2. \tag{2.5}$$

Le résultat $rd - r'$ sera le résultat masqué de d. Il est alors impossible pour une personne ne connaissant ni r ni r' de retrouver d. Cependant, d et son masqué $rd - r'$ ont le même signe de par les contraintes qui portent sur r et r'. Cette propriété permet en utilisant astucieusement le cryptosystème additivement homomorphe de comparer deux valeurs de façon sécurisée.

Notre solution s'appuiera comme celle de KBH sur l'idée de masquage de données

cependant nous montrerons aussi comment réduire les coûts de calcul et de communication en ajoutant deux techniques que sont :

— la concaténation de données,

— la conversion de chiffré.

Nous utiliserons le cryptosystème additivement homomorphe de Damgård–Jurik [DJ03] (parfois noté D-J) car il est la généralisation de celui de Paillier [Pai99]. Il permet ainsi une plus grande flexibilité entre autres pour augmenter la taille du clair. Nous allons présenter plus en détail ce chiffrement dans la section suivante, mais il est intéressant de savoir que notre protocole pourrait fonctionner avec n'importe quel autre chiffrement homomorphiquement additif.

2.1.1.2 Cryptosystème de Damgård–Jurik

Le cryptosystème de Damgård–Jurik [DJ03] est une généralisation de celui de Paillier [Pai99]. C'est un cryptosystème additivement homomorphe avec un bon compromis en termes de coûts de calcul et d'expansion de chiffré. Pour définir le cryptosystème, nous avons besoin de deux grands nombres premiers p et q. Pour ce faire, on pourra s'appuyer sur des tests de primalité probabilistes tels que celui de Miller-Rabin [Rab80]. Ces deux grands nombres premiers vont alors servir à créer la clé publique K_p ainsi que la clé secrète K_s :

$$K_p = pq \quad \text{et} \quad K_s = ppcm((p-1),(q-1)) \tag{2.6}$$

où $ppcm(\cdot,\cdot)$ représente la fonction de plus petit commun multiple entre deux entrées. Pour définir le cryptosystème, il faut également disposer de n appelé exposant de Damgård–Jurik qui doit être un nombre entier strictement positif. Dans le cas où ce paramètre vaut 1, le cryptosystème est celui de Paillier. Cet exposant permet de définir l'espace des clairs qui est $\mathbb{Z}_{K_p^n} = \{0, 1, \ldots, K_p^n - 1\}$. Cela signifie que pour chiffrer et déchiffrer correctement, une valeur initiale doit appartenir à l'ensemble $\mathbb{Z}_{K_p^n}$. On définit par la suite $\mathbb{Z}^*_{K_p^n}$ qui représente tous les entiers de $\mathbb{Z}_{K_p^n}$ qui possèdent un inverse. Du fait que ce chiffrement possède deux clés : publique K_p et privé K_s distinctes cela fait de lui un chiffrement à clé publique (ou autrement appelé asymétrique). Afin de chiffrer une donnée, il reste à choisir $g \in \mathbb{Z}^*_{K_p^n}$ tel que :

$$g = (1 + K_p)^j x \mod K_p^{n+1} \tag{2.7}$$

où j est un nombre premier avec K_p et x appartient à l'ensemble $\{t^{K_p^n} | t \in \mathbb{Z}^*_{K_p^n}\}$. Dans [DJ03] les auteurs proposent également une implémentation rapide de ce cryptosystème

qui consiste à choisir :

$$g = 1 + K_p. \tag{2.8}$$

Pour autant, cette dernière ne réduit pas la sécurité du cryptosystème. Par la suite, nous utiliserons cette valeur pour g.

Maintenant que tous les paramètres sont initialisés, montrons comment effectuer le chiffrement. Pour cela, soit $m \in \mathbb{Z}_{K_p^n}$ un message que l'on souhaite chiffrer en $c \in \mathbb{Z}^*_{K_p^{n+1}}$. Pour ce faire, il faut disposer de la clé publique K_p ainsi que du paramètre n et calculer :

$$c = E[m, r] = g^m r^{K_p^n} \mod K_p^{n+1} \tag{2.9}$$

avec r un nombre aléatoire appartenant à $\mathbb{Z}^*_{K_p^n}$. On note l'introduction de la notation $E[\cdot, \cdot]$ qui représente la fonction de chiffrement. Le premier terme de cette fonction est destiné au message à chiffrer. Le deuxième est quant à lui destiné au nombre aléatoire. Il devient clair qu'un même message m peut être chiffré avec plusieurs aléas r différents, ce qui mène à différents chiffrés c. Cette propriété fait du chiffrement de Damgård–Jurik un cryptosystème probabiliste ce qui aide à assurer sa sécurité sémantique. Elle permet pour un même message de lui attribuer plusieurs chiffrés différents. Cette propriété augmente la sécurité d'un cryptosystème. Il est important de noter que pour certains cryptosystèmes qui ne sont pas sémantiquement sûr (ou déterministe), il est tout de même possible de se prévaloir de ce type d'attaque en changeant le mode de fonctionnement *e.g. Cipher Block Chaining* (CBC) [Ehr+78].

Montrons maintenant comment à partir d'un chiffré $c = g^m r^{K_p^n} \mod K_p^{n+1}$ on peut retrouver le message initial m. Pour ce faire, il faut avoir accès à la clé privée K_s. Tout d'abord, remarquons que :

$$c^{K_s} = (g^m r^{K_p^n})^{K_s} = (1 + K_p)^{mK_s} (r^{K_p^n})^{K_s} = (1 + K_p)^{mK_s \mod K_p^n} \mod K_p^{n+1} \tag{2.10}$$

Ensuite, Damgård–Jurik proposent une procédure itérative qui à partir de $(1 + K_p)^m \mod K_p^{n+1}$ permet de retrouver m [Jur03]. Cette procédure utilise la fonction $L(\cdot)$ définie telle que :

$$L(b) = \frac{b - 1}{K_p} \tag{2.11}$$

En appliquant cette fonction de façon itérative, il est possible de retrouver $m \mod K_p^n$. Pour ce faire, il faut tout d'abord extraire $m_1 = m \mod K_p$ puis $m_2 = m \mod K_p^2$ et ce grâce à m_1. Puis, continuer jusqu'à obtenir la valeur $m \mod K_p^n$ souhaitée. De façon

générale, nous allons noter $m_j = m \mod K_p^j$. Tout d'abord, il est intéressant de noter que $m_1 = L((1+K_p)^m \mod K_p^2) = m \mod K_p$. Pour voir cela, il suffit de développer :

$$L((1+K_p)^m) \mod K_p^{n+1} = (m + \binom{m}{2} + \cdots + \binom{m}{n} K_p^{n-1}) \mod K_p^n \tag{2.12}$$

où $\binom{x}{y}$ représente le coefficient binomial *i.e.* $\binom{x}{y} = \frac{x!}{(x-y)!y!}$. Maintenant, on utilise la récurrence *i.e.* $m_j = m_{j-1} + k \times K_p^{j-1}$ avec $k \in [\![0, K_p - 1]\!]$ ainsi on a :

$$L((1+K_p)^m) \mod K_p^{j+1} = (m_j + \binom{m_j}{2} + \cdots + \binom{m_j}{j} K_p^{j-1}) \mod K_p^j \tag{2.13}$$

mais comme pour $t \in [\![1, j-1]\!]$, on a $\binom{m_j}{t+1} K_p^t = \binom{m_{j-1}}{t+1} K_p^t$ car $k \times K_p^{j-1}$ s'annule modulo K_p^j à chaque fois qu'il est multiplié par K_p, cela donne :

$$L((1+K_p)^m) \mod K_p^{j+1} = (m_j + \binom{m_{j-1}}{2} + \cdots + \binom{m_{j-1}}{j} K_p^{j-1}) \mod K_p^j \tag{2.14}$$

et nous permet d'isoler le terme m_j :

$$m_j = L((1+K_p)^m) \mod K_p^{j+1} - (\binom{m_{j-1}}{2} + \cdots + \binom{m_{j-1}}{j} K_p^{j-1}) \mod K_p^j \tag{2.15}$$

On a alors une formule qui nous permet de retrouver itérativement tous les $\{m_j\}_j$. Cette procédure est notée par la fonction $F(\cdot)$ et est donnée par l'Algorithme 1.

Ainsi, le déchiffrement d'un chiffré c est donné par :

$$m = F(c^{K_s}) K_s^{-1} \mod K_p^n \tag{2.16}$$

Comme nous l'avons précisé, le chiffrement de Damgård–Jurik est additivement homomorphe autrement dit :

$$E[m_1, r_1] \square E[m_2, r_2] = E[m_1 + m_2, r_1 \circ r_2] \tag{2.17}$$

avec m_1 et m_2 deux messages appartenant à $\mathbb{Z}_{K_p^n}$. r_1 et r_2 sont quant à eux les deux aléas de $\mathbb{Z}_{K_p}^*$. Nous allons montrer cette propriété et montrer que l'opération qui se cache

Algorithm 1 Fonction $F(\cdot)$ de Damgård–Jurik.

procedure $F(a)$
  $m \leftarrow 0$
  for $j \leftarrow 1, n$ **do** ▷ $m = m_{j-1}$
    $t_1 \leftarrow L(a \mod K_p^{j+1})$
    $t_2 \leftarrow m$
    for $k \leftarrow 2, j$ **do** ▷ $t_2 = m(m-1)\dots(m-k+2)$
      $m \leftarrow m - 1$
      $t_2 \leftarrow t_2 * m \mod K_p^j$
      $t_1 \leftarrow t_1 - \frac{t_2 * K_p^{k-1}}{k!} \bmod K_p^j$ ▷ $t_1 = t_1 - C_k^i K_p^{k-1}$
    end for
    $m \leftarrow t_1$
  end for
  return $m \mod K_p^n$
end procedure

derrière $\square$ et $\circ$ est la multiplication :

$$E[m_1, r_1]E[m_2, r_2] = g^{m_1} r_1^{K_p^n} g^{m_2} r_2^{K_p^n} = g^{m_1+m_2}(r_1 r_2)^{K_p^n} = E[m_1 + m_2, r_1 r_2] \tag{2.18}$$

On peut remarquer qu'en itérant les multiplications nous avons une autre propriété qui est :

$$E[m_1, r_1]^{m_2} = E[m_1 m_2, r_1^{m_2}] \tag{2.19}$$

Le cryptosystème de Damgård–Jurik est sémantiquement sûr sous l'hypothèse *Decisional Composite Residuosity Assumption* (DCRA) [Jur03]. Pour simplifier, cela signifie que pour un attaquant $\mathcal{A}$ il est difficile de savoir, étant donné K_p et un nombre $x \in \mathbb{Z}_{K_p^2}^*$, si :

1. x est un nombre aléatoire de $\mathbb{Z}_{K_p^2}^*$.
2. ou une $n^{\text{ième}}$ puissance aléatoire de $\mathbb{Z}_{K_p^2}^*$ c'est-à-dire qu'il existerait un y tel que $x = y^n \mod K_p^2$.

Dans [Jur03], les auteurs ont aussi établi un parallèle entre la sécurité de leur protocole et ceux d'El-Gamal ou encore RSA. Il est alors intéressant de noter que le NIST [Bar20] fait l'équivalence entre taille du module et nombre de bits de sécurité comme présenté dans la Table 2.1.

Ainsi, un module de taille $\lambda = 2048$ dans Damgård–Jurik offre une sécurité de 112 bits qui commence à devenir la valeur minimale en termes de sécurité.

Maintenant que nous avons défini le cryptosystème additivement homomorphe que

Taille module λ	1024	2048	3072	7680	15360
Nombre de bits de sécurité	≤ 80	112	128	192	256

TABLE 2.1 – Équivalence taille du module et nombre de bits de sécurité.

nous allons utiliser, nous pouvons voir deux contraintes. Premièrement, les calculs de chiffrement et déchiffrement sont relativement coûteux. Ils demandent entre autre de calculer un certain nombre d'exponentiations modulaires, la fonction $F(\cdot)$ est itérative, ... De plus, on constate que pour un clair appartenant à $\mathbb{Z}_{K_p^n}$ son chiffré appartient à $\mathbb{Z}_{K_p^{n+1}}$ ce qui provoque une expansion de chiffré et par conséquent augmente les coûts de communication et de stockage. Cependant, ce cryptosystème permet, de par ses propriétés, de traiter les données de façon sécurisée ce qui nous sera indispensable par la suite. Il faut alors, pour l'IMD, dont les ressources sont contraintes, utiliser un chiffrement moins coûteux sur le plan calculatoire et de communication, mais qui peut être converti en donnée chiffrée par le chiffrement de Damgård–Jurik. C'est pour quoi, dans la partie qui suit, nous définissons un tel chiffrement.

2.1.2 Algorithmes rapides et peu coûteux assurant la sécurité

Comme nous l'avons dit, bien que le cryptosystème de Damgård–Jurik ait des coûts de communication et de calculs mesurés ceux-ci restent bien trop importants pour l'IMD. C'est pour cela que nous nous sommes penchés sur l'étude d'une fonction de chiffrement symétrique moins coûteuse que les solutions asymétriques c.f. Section 1.3. Ces chiffrements ne possèdent qu'une seule clé : la clé privée pour chiffrer et déchiffrer. Nous allons devoir :

- Trouver parmi les solutions de chiffrement symétrique une solution peu coûteuse qui est adaptable à l'IMD.
- Une fonction qui permet sous certaines conditions de convertir un chiffré sous cette fonction en un chiffré de Damgård–Jurik afin de pouvoir faire le traitement sécurisé de ces dernières.

Parmi ces chiffrements symétriques présentés Section 1.3.1, les chiffrements par flots sont généralement plus rapide que ceux par bloc, c'est pour cela que nous allons nous orienter vers un tel cryptosystème. La solution que nous présentons est bien adaptée au contexte des IMD et de l'IoT. Elle est peu coûteuse et va permettre de convertir les données en données chiffrées par Damgård–Jurik. L'algorithme qui permet de passer du chiffrement par flot vers le chiffrement additivement homomorphe est appelé CrC pour *Cryptosys-*

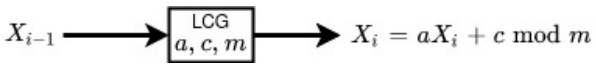


FIGURE 2.2 – Fonctionnement du LCG.

tem Conversion. Il sera présenté dans une seconde partie, commençons par décrire le chiffrement par flot utilisé.

2.1.2.1 Chiffrement par flot CLCG

Le cryptosystème utilisé est le CLCG pour *Combined Linear Congruential Generator*. Il a été proposé par Wichmann et Hill dans [WH82]. Un tel chiffrement est très rapide. Il consiste simplement à faire la somme modulaire entre la donnée à chiffrer d et Z_i qui est un nombre généré par l'algorithme. En effet, le CLCG est essentiellement un générateur de nombres pseudo-aléatoires initialisé par une graine (ou clé). De plus, il a l'avantage en comparaison d'autres chiffrements à flots, tels que Grain ou Snow de pouvoir sous certaines conditions être converti en chiffrement de Damgård–Jurik.

Le CLCG est un générateur de nombres pseudo-aléatoires composé de deux LCG (*Linear Congruential Generator*) qui sont eux-mêmes des générateurs pseudo-aléatoires. La sortie d'un LCG est calculé comme suit :

$$X_i = aX_{i-1} + c \mod m \tag{2.20}$$

où a est le multiplicateur, c l'incrément et m le module. Comme tout générateur de nombres pseudo-aléatoires, il est initialisé par une graine qui est X_0. On peut voir son fonctionnement Figure 2.2. Le CLCG est composé de deux LCG, nous notons X_i la sortie du premier relativement aux paramètres a_X, c_X, X_0 et Y_i la sortie du second relativement aux paramètres a_Y, c_Y, Y_0. Il est important de noter que le module m est le même pour les deux LCG. Ceci nous permet de calculer Z_i, la sortie du CLCG dont la graine est X_0, Y_0 comme :

$$Z_i = X_i + Y_i \mod m \tag{2.21}$$

$$= a_X X_{i-1} + c_X + a_Y Y_{i-1} + c_Y \tag{2.22}$$

La Figure 2.3 reprend cette construction. Par conséquent, le chiffrement de la donnée d s'effectue comme suit :

$$c = d + Z_i \mod m \tag{2.23}$$

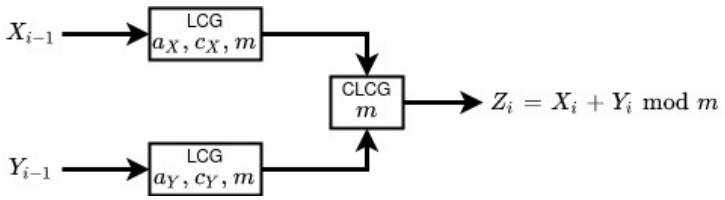


FIGURE 2.3 – Fonctionnement du CLCG.

où c est le chiffré de d. Pour déchiffrer, il suffit de faire l'opération inverse. Il est alors clair que pour que le déchiffrement soit correctement effectué, il faut que d appartienne à $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$. Tant que les paramètres $\{X_0, Y_0, c_X, x_Y\}$ du CLCG reste secret, le fait de connaître $\{a_X, a_Y\}$ et/ou le module m ne met pas en danger la sécurité du chiffrement [LEc99]. Par simplification, nous noterons $K_c = \{(X_0, Y_0), (c_X, c_Y)\}$ la clé secrète du CLCG. Maintenant que nous avons notre chiffrement léger, regardons comment faire pour le combiner au chiffrement additivement homomorphe de Damgård–Jurik.

2.1.3 Conversion de chiffrement par flot et additivement homomorphe

L'idée de convertir un chiffrement vers un autre sans avoir besoin de déchiffrer les données n'est pas nouvelle. L'intérêt étant de pouvoir utiliser au choix un cryptosystème rapide peu coûteux et ayant des coûts de communication modique ou bien un cryptosystème qui permet de traiter de façon sécurisée les données. L'idéal aurait été de pouvoir avoir un cryptosystème alliant toutes ces qualités à la fois. Cependant, à l'heure actuelle, un tel chiffrement n'existe pas. Comme exemple de conversion de chiffré, on peut donner [GHS12] qui ont proposé une solution permettant de passer du chiffrement par bloc AES vers le chiffrement *fully* homomorphe BGV [BGV14]. Le problème avec cette solution est qu'elle met approximativement 40 minutes pour faire la conversion d'un unique bloc AES. D'autres auteurs [MS13] ont de leur côté essayé de convertir un chiffré Salsa20 vers un chiffrement *fully* homomorphe. Cependant, au vu du cryptosystème utilisé, les auteurs ne s'attendaient pas à ce que le calcul se fasse en un temps raisonnable. Plus récemment dans [Can+18], les auteurs ont proposé de convertir un message chiffré par TRIVIUM [DP08] (un finaliste du concours eSTREAM [Rob08] et présenté comme un standard international [ISO12]) vers un chiffrement *fully* homomorphe. Cependant, la complexité de leur solution fait qu'elle reste limitée en pratique. Nous allons présenter dans la suite notre solution permettant de passer d'un CLCG vers un chiffrement de Damgård–Jurik.

La solution que nous allons présenter fait alors peser sur le chiffrement par flot un certain nombre de contraintes que nous verrons dans la section suivante.

2.1.3.1 Chiffrement par flot CLCG dans le domaine homomorphe

La contrainte pour pouvoir utiliser conjointement le CLCG avec Damgård–Jurik est que le module du CLCG m doit être égal à la clé publique K_p de Damgård–Jurik. En effet, ce dernier point nous permet d'introduire le concept de SCLCG pour *Secure Combined Linear Congruential Generator Cryptosystem*. Le SCLCG est l'implémentation du CLCG dans le domaine chiffré de Damgård–Jurik. Pour vérifier cela, regardons tout d'abord comment calculer dans le domaine de Damgård–Jurik la sortie d'un LCG partant de (2.20), cette dernière est telle que :

$$E[X_i, r_i^a r_c] = E[X_{i-1}, r_i]^a E[c_X, r_c] \tag{2.24}$$

avec X_0 la graine du LCG, a son multiplicateur, c son incrément et $m = K_p$ le module. De par les propriétés homomorphes de Damgård–Jurik présentées (2.18) et (2.19), on obtient bien la sortie du SLCG (*Secure Linear Congruential Generator Cryptosystem*). Maintenant que nous avons vu comment calculer un SLCG, il suffit de répéter l'opération deux fois puis d'utiliser la propriété d'homomorphie du cryptosystème pour obtenir la sortie du SCLCG :

$$E[Z_i, r_{Z,i}] = E[X_i, r_{X,i}]E[Y_i, r_{Y,i}] \tag{2.25}$$

Avec le calcul des deux SLCG qui se font comme suit :

$$\begin{aligned} E[X_i, r_{X,i}] &= E[X_{i-1}, r_{X,i-1}]^{a_X} E[c_X, r_{c_X}] \\ &= E[a_X X_{i-1} + c_X, r_{X,i-1}^{a_X} r_{c_X}] \qquad (2.26) \\ E[Y_i, r_{Y,i}] &= E[Y_{i-1}, r_{Y,i-1}]^{a_Y} E[c_Y, r_{c_Y}] \\ &= E[a_Y Y_{i-1} + c_Y, r_{Y,i-1}^{a_Y} r_{c_Y}] \qquad (2.27) \end{aligned}$$

La clé secrète du SCLCG sera alors :

$$E[K_c] = \{E[X_0, r_{X,0}], E[Y_0, r_{Y,0}], E[c_X, r_{c_X}], E[c_Y, r_{c_Y}]\} \tag{2.28}$$

On peut noter que du fait que $m = K_p^n$, la période du CLCG est très longue. En effet, comme dit précédemment, pour des questions de sécurité, K_p^n doit être codé sur au moins

2048 bits et par conséquent la période du CLCG est majorée par $K_p^n - 1 \leq 2048$ [LEc+02]. Comme la période est longue, cela assure le fait qu'il n'est pas possible de prédire les sorties du CLCG et cela préserve sa sécurité, il en est de même pour le SCLCG. Plus précisément, en ce qui concerne le SCLCG, on peut constater de par (2.26) et (2.27) que les multiplicateurs a_X et a_Y sont disponibles en clair. Cependant, cela ne réduit pas la sécurité du SCLCG du fait que ces paramètres a_X et a_Y n'ont pas vocation à être chiffrés pour le cas du CLCG [LEc99]. Par conséquent, on voit que grâce au SCLCG il est possible de générer les séquences de sortie d'un CLCG dans le domaine de Damgård–Jurik. Cela nous permettra comme nous allons le voir dans la prochaine section, d'arriver à chiffrer, mais surtout déchiffrer des données de façon sécurisée dans le domaine de Damgård–Jurik.

Cette solution a une contrainte : le modulo du CLCG doit être égal à celui de Damgård–Jurik. Cependant, comparée aux autres approches qui permettent également de convertir des chiffrés, notre solution a des coûts de communication et surtout de calcul bien moindre. En plus, elle a l'avantage de permettre d'avoir des données chiffrées dans un système de chiffrement additivement homomorphe. Or, ces chiffrements homomorphes sont largement utilisés afin de pouvoir traiter efficacement et de manière sécurisée des données [TEE12 ; Pop+11].

2.1.3.2 Système de conversion de chiffré

Dans ce paragraphe, nous allons définir le CrC [PBC21] pour *Cryptosystem Conversion*. Cette technique nous permet de passer d'une donnée chiffrée par le CLCG à une donnée chiffrée par Damgård–Jurik sans pour autant, à aucun moment, devoir la déchiffrer. Cela permet de chiffrer à moindre coût les données au niveau de l'IMD puis de les transmettre de façon sécurisée sans augmenter les coûts de communication. Pour cela, comme nous l'avons vu dans la section précédente, nous allons utiliser les propriétés du SCLCG. Pour ce faire, il faut que le module du CLCG et la clé publique du cryptosystème additif de Damgård–Jurik soient les mêmes *i.e.* $K_p = m$. Nous allons supposer que l'IMD possède tous les paramètres nécessaires pour faire le chiffrement CLCG, c'est-à-dire :

— les données à chiffrer $\{d_i\}_i$,
— les paramètres publics du CLCG $m = K_p$, ainsi que a_X et a_Y,
— les paramètres secrets du CLCG c_X, c_Y, et la graine du CLCG X_0, Y_0.

De son côté, on suppose que l'IHM possède :

— les multiplicateurs du CLCG a_X et a_Y,
— les paramètres du SCLCG $E[c_X, r_{c_X}]$, $E[c_Y, r_{c_Y}]$ et la graine $E[X_0, r_{X,0}]$, $E[Y_0, r_{Y,0}]$

le tout sécurisé par l'algorithme de Damgård–Jurik,

— les paramètres publics du cryptosystème de Damgård–Jurik à savoir K_p, n et g.

Dans un premier temps, l'IMD procède au chiffrement des données par le CLCG :

$$c_i = d_i + Z_i \mod m \tag{2.29}$$

avec Z_i la i^{eme} sortie du CLCG. L'IMD envoie alors le résultat chiffré c_i à l'IHM. Cette dernière, ne connaissant pas en clair la clé secrète du CLCG, ne peut pas déchiffrer le message. Notons, que bien qu'elle possède cette clé de façon chiffrée, comme elle ne connaît pas la clé secrète de Damgård–Jurik, elle ne peut pas la retrouver. L'IHM va alors pouvoir faire un double chiffrement de d_i. En effet, c_i est déjà chiffré par le CLCG de d_i. Elle va alors le chiffrer une deuxième fois avec Damgård–Jurik :

$$E[c_i, r_{c_i}] = g^{c_i} r_i^{K_p^n} \mod K_p^{n+1} = E[d_i + Z_i, r_i] \tag{2.30}$$

Pour des raisons de coût de calcul, il est également possible de prendre $r_i = 1$ ce qui évite de calculer les exponentiations modulaires. L'IHM possède alors $E[d_i + Z_i, r_i]$ ainsi que les paramètres lui permettant de calculer un SCLCG :

$$E[Z_i, r_{Z_i}] = E[X_{i-1}, r_{X,i-1}]^{a_X} E[c_X, r_{c_X}] E[Y_{i-1}, r_{Y,i-1}]^{a_Y} E[c_Y, r_{c_Y}] \tag{2.31}$$

Il est important que l'IMD et l'IHM soient synchronisés. Dans le cas d'une désynchronisation, l'IMD enverrait c_i à l'IHM et cette dernière calculerait $E[Z_j, r_{z,j}]$, ce qui, ne permet pas d'exploiter les données. Dans le cas où les deux entités sont bien synchrones, l'IHM calcule :

$$E[d_i, r_{d_i}] = E[c_i, r_{c_i}] E[Z_i, r_{Z_i}]^{-1} = E[d_i + Z_i, r_{c_i}] E[-Z_i, r_{Z_i}] \tag{2.32}$$

On constate que ce calcul n'est possible qu'à la condition que le chiffrement $E[\cdot]$ possède les propriétés d'homomorphies présentées par (2.18) et (2.19). L'IHM possède alors les données médicales d_i directement disponibles dans le domaine chiffré de Damgård–Jurik sans avoir à aucun moment pu les avoir en clair. Du fait qu'elle ne connaisse pas la clé secrète du cryptosystème additivement homomorphe, elle ne peut pas retrouver les données. Cependant, elle est capable de traiter efficacement ces dernières.

Toute la procédure du CrC est résumée dans la Figure 2.4.

L'utilisation du CrC, du chiffrement CLCG et de Damgård–Jurik sur les données impose un certain nombre de contraintes dans leur utilisation. La plus importante est

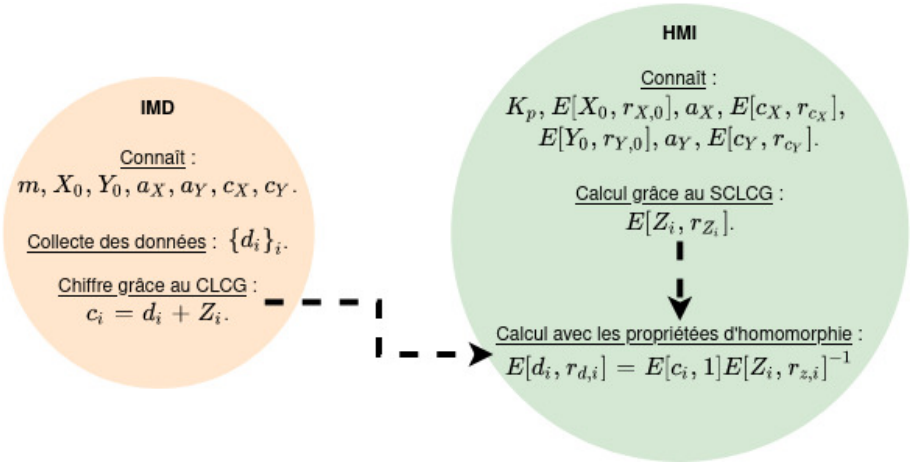


FIGURE 2.4 – Fonctionnement du CrC entre l'IMD et l'IHM.

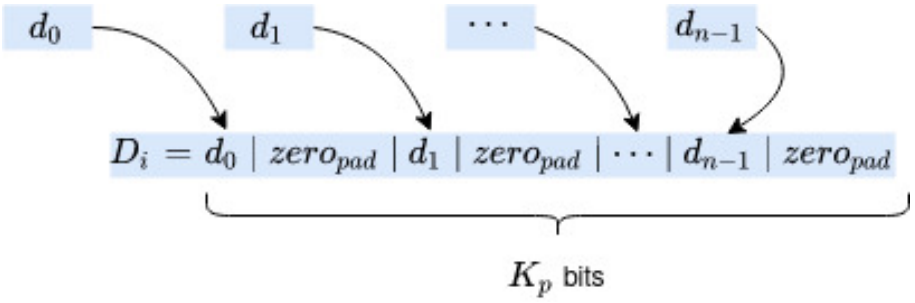


FIGURE 2.5 – Fonctionnement de la concaténation de données.

liée au fait que le modulo du CLCG est égale à la clé public de Damgård–Jurik, *i.e.* $m = K_p$. Or, comme nous l'avons vu, afin d'assurer la sécurité des données, il faut que K_p soit supérieur à 2048 bits. Comme les données médicales (ou d'autre nature) sont rarement codées sur 2048 bits, on constate que l'espace des clairs du cryptosystème de Damgård–Jurik est sous exploité avec une perte importante en ce qui concerne les coûts de communication. En effet, envoyer 2048 bits pour seulement quelques bits (*e.g.* 8 ou 11) d'information n'est pas intéressant. Nous allons voir dans la section qui suit comment pallier ce problème.

2.1.4 Attentes de la concaténation de données

Comme indiqué ci-dessus, l'espace des clairs du CLCG ou de manière équivalente du cryptosystème de Damgård–Jurik (au moins 2048 bits), est sous exploité. Une solution pour contourner ce problème est de concaténer les données tout en laissant la possibilité

de les traiter, comme présenté à la Section 1.4.3. On trouve ce type d'approche en ce qui concerne les chiffrements *fully* homomorphes (FHE). Les solutions proposées sont dites *Simple Instruction Multiple Data* (SIMD) [SV14]. Les chiffrements *fully* homomorphes permettent de chiffrer plusieurs données en une seule fois et faire des opérations sur chaque donnée du chiffré indépendamment. En ce qui concerne le chiffrement additivement homomorphe, de telles solutions sont plus complexes à mettre en œuvre et posent certaines contraintes. La concaténation, présentée Figure 2.5, consiste à réunir plusieurs données en une donnée unique de plus grande taille. Pour ce faire, les données $\{d_i\}_i$ sont concaténées l'une après l'autre en intercalant entre chaque donnée un nombre de 0 prédéfinis. Cette opération se nomme *zero padding*. Elle est essentielle pour la suite du traitement.

On remarque, par exemple, que les auteurs de [Pho+18] qui dans un contexte *deep learning* ont proposé de traiter N données d'une longueur de δ bits $\{d_j\}_{j=0,\dots,N-1}$ en les concaténant dans un unique vecteur D_i de longueur $N = \lambda/\delta$ (où λ est la taille en bits du module) avant de les chiffrer avec un algorithme de chiffrement additivement homomorphe. Le résultat est alors donné par :

$$D_i = d_0 + d_1 2^{\delta} + \cdots + d_{N-1} 2^{\delta(N-1)} \tag{2.33}$$

Afin de faire des opérations entre vecteurs concaténés, un *zero padding* devra être employé. De façon synthétique, pour expliquer l'obligation d'avoir un *zero padding* (*i.e.* $\delta \geq \lceil log_2(d_i) \rceil$ avec $\lceil \cdot \rceil$ la fonction partie entière supérieure), imaginons que ce dernier n'existe pas. On aurait deux valeurs :

$$D_0 = \sum_{i=0}^{N-1} 2^{\sum_{k=0}^{i-1} \lceil log_2(d_k) \rceil} d_i \tag{2.34}$$

$$D_1 = \sum_{j=0}^{M-1} 2^{\sum_{k=0}^{j-1} \lceil log_2(\bar{d_k}) \rceil} \bar{d_j} \tag{2.35}$$

L'objectif serait alors d'en faire la somme :

$$D_0 + D_1 = \sum_{k} 2^{k\delta} (d_k + \bar{d_k}) \tag{2.36}$$

Dans un premier temps, si les valeurs ne sont pas synchronisées, *i.e.* $2^{\lceil log_2(d_i) \rceil} \neq 2^{\lceil log_2(\bar{d_j}) \rceil}$, il ne serait alors pas possible en faisant la somme de D_0 et D_1 d'obtenir le résultat de (2.36). Or, comme nous l'avons dit, on souhaite faire un filtrage suivi d'un seuillage (2.1)

donc ce type de somme est à la base de notre problématique. Supposons maintenant que les deux vecteurs concaténés D_0 et D_1 soient tous les deux synchronisés *i.e.* :

$$D_0 = \sum_{i=0}^{N-1} 2^{\delta_i} d_i \quad \text{et} \quad D_1 = \sum_{i=0}^{N-1} 2^{\delta_i} \bar{d}_i \tag{2.37}$$

On constate tout d'abord que cette fois-ci les deux données concaténées possèdent le même nombre de données à savoir N. Cependant, cela n'est pas suffisant. En effet, pour simplifier supposons que chaque donnée $\{d_i\}_i$ et $\{\bar{d}_i\}_i$ soit codée sur δ bits. On pourrait alors réécrire :

$$D_0 = \sum_{i=0}^{N-1} 2^{i\delta} d_i \quad \text{et} \quad D_1 = \sum_{i=0}^{N-1} 2^{i\delta} \bar{d}_i \tag{2.38}$$

Supposons que d_0 et $\bar{d}_0$ valent $2^\delta - 1$. Elles sont alors bien toutes deux codées sur δ bits. Cependant, leur somme est codée sur $\delta + 1$ bits. On comprend alors qu'en effectuant la somme $D_0 + D_1$, la somme des deux valeurs d_0 et $\bar{d}_0$ va être codée sur $\delta + 1$ bits et ainsi va venir polluer la somme $d_1 + \bar{d}_1$. Il ne sera alors plus possible, en connaissant δ, de déconcaténer les valeurs pour retrouver $\{d_i + \bar{d}_i\}$. C'est pourquoi un "espace de sécurité" doit être aménagé entre chaque valeur d_i afin de s'assurer, au cours du traitement, que jamais les données concaténées débordent les unes sur les autres. Cette opération est nommée *zero padding* et la taille du *padding* dépend des données d'entrées ainsi que des opérations qui leur seront appliquées.

La concaténation permet de répondre à l'augmentation de la taille de chiffré induite par la taille de la clé publique K_p. Concaténer plusieurs données en un unique vecteur avant de le chiffrer, permet de réduire les coûts de communications. Cependant, comme nous allons le voir dans la prochaine section, la concaténation induit une réorganisation des données qui contraint les opérations envisageables.

2.1.4.1 Concaténation et conséquences sur les opérations homomorphes

Si, avec la concaténation, les coûts de communications sont maîtrisés, elle apporte également une nouvelle organisation des données. En effet, concaténer des valeurs revient à créer un vecteur de données. Considérant plusieurs vecteurs, on peut alors voir les

données comme faisant partie d'une matrice :

$$\begin{bmatrix} D_0 \\ \vdots \\ D_i \\ \vdots \\ D_{M-1} \end{bmatrix} = \begin{bmatrix} d_{0,0} & \cdots & d_{0,N-1} \\ \vdots & & \vdots \\ d_{M-1,0} & \cdots & d_{M-1,N-1} \end{bmatrix} \tag{2.39}$$

Cette visualisation nous permet de constater que, quand nous souhaitons faire la somme de deux données D_i, D_j, nous souhaitons en réalité faire la somme de chaque colonne de la matrice pour les lignes i et j. Nous constatons également qu'une telle concaténation nous empêche de sommer autre chose que la colonne l pour la donnée D_i avec la même colonne pour la donnée D_j. Dit autrement, il n'est pas possible de faire des sommes entre deux données n'appartenant pas à la même colonne. Cependant, cela n'est pas un problème dans notre contexte étant donné que nous supposons que nous avons N données provenant chacune d'un capteur différent.

Au-delà de la somme, nous pouvons également faire des sommes pondérées sur les colonnes, c'est-à-dire une opération de filtrage. En effet, il est intéressant de voir que, si l'on multiplie une donnée concaténée par une valeur V, cela revient à multiplier chaque donnée individuellement par cette même valeur V :

$$V \times D_0 = \sum_{i=0}^{N-1} 2^{i\delta} V d_i \tag{2.40}$$

Ce fonctionnement a l'avantage de permettre la multiplication de plusieurs données concaténées en une seule fois par une même valeur V. Cependant, elle a l'inconvénient de ne pas autoriser la multiplication de chaque donnée concaténée dans un même vecteur par une valeur différente. Cela a des conséquences sur les opérations de filtrage que nous souhaitons effectuer par la suite. En effet, une opération de filtrage classique est de la forme :

$$a_j = \sum_{i=0}^{M-1} w_{i,j} d_{i,j} \tag{2.41}$$

avec a_j la sortie filtrée et $\{w_{i,j}\}_{i=0,\ldots,M-1;j=0,\ldots,N-1}$ les poids du filtre, $d_{i,j}$ une donnée du $j^{\text{ième}}$ capteur contenu dans le vecteur D_i et M le nombre de vecteurs $\{D_i\}_{i=0,\ldots,M-1}$. Cette

opération, (2.41) peut être réexprimée comme un produit de matrices :

$$A = D^t W \tag{2.42}$$

Comme notre opération de concaténation empêche la multiplication des données d'un même vecteur par des poids différents, il n'est pas possible d'appliquer des filtres différents sur chaque colonne, ainsi, nous avons $\forall j, w_{i,j} = w_i$. On peut alors réécrire (2.42) comme suit :

$$A = \begin{bmatrix} a_0 \\ \vdots \\ a_j \\ \vdots \\ a_{N-1} \end{bmatrix} = \begin{bmatrix} d_{0,0} & \cdots & d_{M-1,0} \\ \vdots & & \vdots \\ d_{0,N-1} & \cdots & d_{M-1,N-1} \end{bmatrix} \begin{bmatrix} w_0 \\ \vdots \\ w_i \\ \vdots \\ w_{M-1} \end{bmatrix} \tag{2.43}$$

avec D une matrice de taille $M \times N$, W le vecteur des poids des filtres de taille $M \times 1$ et A le vecteur de λ-bit qui contient la sortie du filtrage de chaque capteur.

Maintenant que nous avons présenté les opérations, revenons sur l'intérêt et le calcul du *zero padding* qui permet de garantir l'exactitude de l'opération de filtrage et notamment le fait que le vecteur concaténé A contienne les résultats du filtrage des données de chaque colonne. En effet, le but est d'assurer que peu importe les données d'entrée, le fait de connaître les opérations qui vont être faites suffit à trouver un *zero padding* de telle sorte que jamais les données ne "débordent" les unes sur les autres.

Commençons par étudier le cas où les données ainsi que les poids du filtre sont positifs. Le nombre de bits b_j, *zero padding* inclus, qu'il est nécessaire d'allouer aux $\{d_{i,j}\}_{i=0,\dots M-1}$ dépend du nombre d'opérations effectuées, en l'occurrence M multiplications et $M-1$ additions pour l'opération de filtrage (2.41). Or, on sait qu'une multiplication impliquant deux entiers de respectivement l_a bits et l_b bits produit un résultat codé sur au plus $l_a + l_b$ bits. Nous devons alors considérer les tailles en bits de $\{d_i\}_{i=0,\dots,M-1}$ et $\{w_i\}_{i=0,\dots,M-1}$. Pour cela, on note $\{b_j^d, b^w\}$ tel que :

$$d_{i,j} < 2^{b_j^d} \quad \text{et} \quad w_i < 2^{b^w} \quad \forall i \in [\![0, M-1]\!] \tag{2.44}$$

Ainsi, $2^{b_j^d}$ (resp. 2^{b^w}) majore $d_{i,j}$ (resp. w_i) pour tous les indices i.

Par conséquent, (2.41) est majorée par :

$$a_j = \sum_{i=0}^{M-1} w_{i,j} d_{i,j} < 2^{log_2(M)+b^w+b_j^d} \tag{2.45}$$

Cette équation montre que le résultat du filtrage a_j est codée sur au plus b_j bits avec :

$$b_j = log_2(M) + b^w + b_j^d \tag{2.46}$$

En conséquence, les données $\{d_{i,j}\}_{i=0,\dots,M-1}$ doivent être concaténées sur b_j afin de créer D_i. Concrètement, en utilisant N capteurs, D_i est égal à :

$$D_i = \sum_{j=0}^{N-1} 2^{\delta_j} d_{i,j} \quad \text{avec} \quad \delta_j = \sum_{k=0}^{j-1} b_k \tag{2.47}$$

De là, (2.43) peut être calculée dans le domaine chiffré de Damgård–Jurik. Plus précisément, considérons la version chiffrée des données concaténées $\{E[D_i]\}_{i=0,\dots,M-1}$. Notons que pour le reste de cette section et pour des raisons de simplicité, nous ne notons pas les valeurs des nombres aléatoires présents dans Damgård–Jurik. Le calcul de (2.43) par l'IHM dans le domaine chiffré se fait comme suit :

1. l'IHM calcule $\forall i \in [\![0, M-1]\!]$, $E[D_i]^{w_i} = E[w_i \sum_{j=0}^{N-1} 2^{\delta_j} d_{i,j}] = E[\sum_{j=0}^{N-1} 2^{\delta_j} w_i d_{i,j}]$.

2. afin d'obtenir les résultats du filtrage des données l'IHM calcule :

$$\begin{aligned} E[A] &= E[\sum_{j=0}^{N-1} 2^{\delta_j} a_j] = \prod_{i=0}^{M-1} E[D_i]^{w_i} \\ &= E[\sum_{j=0}^{N-1} (2^{\delta_j} \sum_{i=0}^{M-1} w_i d_{i,j})] \end{aligned} \tag{2.48}$$

Généralisons maintenant ce résultat au cas où les données ainsi que les poids du filtre sont positifs ou négatifs. Les calculs sont sensiblement les mêmes. Il faut juste prendre en compte pour le calcul de b_j le bit de signe de $\{d_{i,j}\}_{i=0,\dots,M-1}$ et $\{w_i\}_{i=0,\dots,M-1}$. Ainsi, considérons que $|d_{i,j}| < 2^{b_j^d}$ et $|w_i| < 2^{b^w}$, la sortie du filtrage qui peut être négative *i.e.* $|a_j| < 2^{b_j}$, b_j est donc donnée par :

$$b_j = log_2(M) + b^w + b_j^d + 1 \tag{2.49}$$

Le nouveau calcul de δ_j donne :

$$A = \sum_{j=0}^{N-1} 2^{\delta_j} a_j \mod K_p^n \quad \text{avec} \quad \delta_j = \sum_{k=0}^{j-1} b_k \tag{2.50}$$

avec K_p^n la clé publique du cryptosystème de Damgård–Jurik. En conséquence, si A est négatif alors sa valeur va être modifiée par le calcul du module K_p^n. Ainsi, si A appartient à $[\![\frac{K_p^n+1}{2}, \ldots, K_p^n - 1]\!]$, on en déduira que A est négatif. Au contraire, si A appartient à $[\![0, \ldots, \frac{K_p^n-1}{2}]\!]$, alors il sera positif.

Il est intéressant de noter que nous souhaiterons en plus d'une opération de filtrage, calculer une opération de seuillage. Dans notre contexte applicatif, la valeur du seuil s est codée sur le même nombre de bits que celui du résultat du filtrage a_j. C'est pourquoi notre calcul des δ_j ne prend pas en compte ce paramètre. S'il le fallait, sachant que la somme de deux valeurs codées sur l_a et l_b bits est codée sur au plus $max(l_a, l_b) + 1$ bits et en définissant b_s tels que $|s| < 2^{b_s}$, il faudrait redéfinir δ_j comme étant égal à $max(\delta_j, b_s + 1)$.

Afin de simplifier les écritures dans la suite, nous noterons $\delta = max_j(b_j)$ afin de réduire autant que faire se peut le nombre d'indices.

Regardons maintenant comment, une fois tous les calculs effectués, il est possible de dé-concaténer les vecteurs afin de retrouver les résultats.

2.1.4.2 Extraction des données

Une fois que l'IHM aura, avec l'aide de l'IMD et du Serveur, réussi à calculer de façon sécurisée l'opération de filtrage suivie du seuillage, il lui faudra dé-concaténer les données.

Dans le cas où les données ainsi que les filtres sont positifs, le calcul est direct. Pour cela, il suffit d'extraire les a_j de :

$$A = \sum_{j=0}^{N-1} 2^{j\delta} a_j \mod m \tag{2.51}$$

en calculant :

$$a_j = \left\lfloor \frac{A}{2^{j\delta}} \right\rfloor \mod 2^{(j+1)\delta} \tag{2.52}$$

avec $\lfloor \cdot \rfloor$ la fonction arrondie.

Si maintenant les données ainsi que les poids des filtres peuvent être négatifs, il faut prendre en compte le modulo m induit par le cryptosystème de Damgård–Jurik. En effet, il est possible que les données $\{a_j\}_{j=0,\ldots,N-1}$ soient positives ou négatives avec A dans

l'intervalle $[\![-\frac{K_p^n-1}{2}, \frac{K_p^n-1}{2}]\!]$. Le modulo modifie alors sa valeur.

Pour voir cela, commençons avec un exemple où $A = a_0$ *i.e.* une unique valeur est concaténée dans A. On va supposer, pour simplifier les notations, que $\forall j\ a_j \in [\![2^{\delta-1} - 1, 2^{\delta-1} - 1]\!]$. Ainsi, si $a_0 \in [\![0, 2^{\delta-1} - 1]\!]$, alors $A \in [\![0, 2^{\delta-1} - 1]\!]$. Dans ce cas-ci, la dé-concaténation est directe. Par contre, si $a_0 \in [\![-2^{\delta-1} - 1, -1]\!]$ alors, du fait du modulo, on aura $A \in [\![K_p^n - 2^{\delta-1} - 1, K_p^n - 1]\!]$. On pourra retrouver a_0 en calculant :

$$a_0 = A - K_p^n \tag{2.53}$$

Maintenant, supposons que $A = a_0 + 2^\delta a_1$ nous allons tout d'abord extraire a_1. Nous nous retrouverons dans la configuration où A ne sera alors composé que d'une seule valeur a_0 ; un cas déjà traité. Pour commencer, on constate que comme $|a_0| < 2^{\delta-1}$ alors peu importe sa valeur, le signe de A est uniquement déterminé par le signe de a_1. Cependant, rappelons que nous n'avons pas accès directement à A mais à $A \mod K_p^n$. Ainsi, commençons par déterminer le signe de A. Pour cela, si $A \mod K_p^n < \frac{K_p^n-1}{2}$ alors A est positif, sinon A est négatif. Il faut calculer :

$$A = A - K_p^n \tag{2.54}$$

Pour retrouver a_1, il suffit de diviser A par 2^δ et d'arrondir à l'entier le plus proche :

$$a_1 = \left\lfloor \frac{A}{2^\delta} \right\rceil \tag{2.55}$$

où $\lfloor \cdot \rceil$ représente la fonction arrondie. Comme $|a_0| < 2^{\delta-1}$, alors $\left\lfloor \frac{a_0}{2^\delta} \right\rceil = 0$ et donc $\left\lfloor \frac{A}{2^\delta} \right\rceil = \left\lfloor \frac{a_0+2^\delta a_1}{2^\delta} \right\rceil = a_1$. Une fois que a_1 est connu, il suffit de le retrancher à A en calculant :

$$A = A - 2^\delta a_1 = a_0 \mod m \tag{2.56}$$

Il ne reste dans A plus qu'une unique valeur et nous avons montré comment l'extraire.

On peut généraliser cette procédure pour N données concaténées. C'est ce que fait l'Algorithme 2.1.4.2 qui montre comment en commençant par extraire a_{N-1} on peut alors extraire toutes les valeurs qui suivent.

Il peut être intéressant de noter que cet algorithme de dé-concaténation n'est pas unique. Nous avons présenté l'algorithme dans le cas où chaque valeur $|a_1| < 2^{\delta-1} - 1$. Cette condition peut être généralisée en suivant le prochain théorème.

Theorem 1 (Concaténation Dé-concaténation)**.** *Soit* $\{P_j\}_{0 \le j \le N-1} \in \mathbb{N}^*$ *de telle sorte*

Algorithm 2 Algorithme de dé-concaténation.

if $A > \frac{K_p^n - 1}{2}$ **then**
  $A \leftarrow A - K_p^n$
end if
$save \leftarrow 0$
for $j = N-1\,;\, j \geq 0\,;\, j--$ **do**
  $a_j \leftarrow \left\lfloor \frac{A - save}{2^{\delta_j}} \right\rceil$
  $save \leftarrow save + a_j 2^{j\delta}$
end for

que : $P_j < P_{j+1} \quad \forall j \in [\![0, N-1]\!]$ *avec* $P_N = K_p^n$. *Prenons* $A = \sum_{j=0}^{N-1} a_j P_j \mod K_p^n$. *Alors, on peut dé-concaténer* A, i.e. *extraire toutes les valeurs* a_j *à la condition que :*

$$|\sum_{j=0}^{k} a_j P_j| \leq \frac{P_{k+1} - 1}{2} \quad \forall k \in [\![0, N-1]\!] \tag{2.57}$$

Démonstration. Grâce à (2.57), on peut noter :

$$|\sum_{j=0}^{N-1} a_j P_j| \leq \frac{K_p^n - 1}{2} \tag{2.58}$$

Pour enlever le modulo K_p^n de A, nous allons noter $A' = A - K_p^n$ si $A > \frac{K_p^n - 1}{2}$ ou, $A' = A$ sinon. Plus clairement, on a :

$$A' = \sum_{j=0}^{N-1} a_j P_j \tag{2.59}$$

Regardons plus précisément $b_{N-i} = \left\lfloor \frac{A' - \sum_{j=0}^{i-1} b_{N-j} P_{N-j}}{P_{N-i}} \right\rceil$. L'objectif est de montrer que $b_{N-i} = a_{N-i}$, $\forall i \in [\![1, N]\!]$, ce qui donnerait une formule directe pour extraire a_i. Prouvons cela par récurrence.

Commençons par $i = 1$, on a alors :

$$b_{N-1} = \left\lfloor \frac{A'}{P_{N-1}} \right\rceil \tag{2.60}$$

En appliquant (2.57), cela donne :

$$|\sum_{j=0}^{N-2} a_j P_j| \leq \frac{P_{N-1} - 1}{2} \tag{2.61}$$

$$\Leftrightarrow \frac{1}{2P_{N-1}} - \frac{1}{2} \leq \frac{\sum_{j=0}^{N-2} a_j P_j}{P_{N-1}} \leq \frac{1}{2} - \frac{1}{2P_{N-1}} \tag{2.62}$$

$$\Leftrightarrow -\frac{1}{2} < \frac{\sum_{j=0}^{N-2} a_j P_j}{P_{N-1}} < \frac{1}{2} \tag{2.63}$$

Du fait que $A' = \sum_{j=0}^{N-1} a_j P_j = \sum_{j=0}^{N-2} a_j P_j + a_{N-1} P_{N-1}$, on peut alors réécrire (2.60) comme :

$$b_{N-1} = \left\lfloor \frac{\sum_{j=0}^{N-2} a_j P_j}{P_{N-1}} + a_{N-1} \right\rceil = a_{N-1} \tag{2.64}$$

On a ainsi montré l'initialisation de la récurrence *i.e.* pour $i = 1$. Regardons maintenant l'hérédité. Nous devons montrer que l'hypothèse est vraie au rang i en supposant qu'elle est vraie jusqu'au rang $(i-1)$. On a :

$$b_{N-i} = \left\lfloor \frac{A' - \sum_{j=0}^{i-1} b_{N-j} P_{N-j}}{P_{N-i}} \right\rceil = \left\lfloor \frac{\sum_{j=0}^{N-i} a_j P_j}{P_{N-i}} \right\rceil$$

Puis, en utilisant (2.57) et un raisonnement similaire au cas de l'initialisation, cela donne :

$$-\frac{1}{2} < \frac{\sum_{j=0}^{N-i-1} a_j P_j}{P_{N-i}} < \frac{1}{2} \tag{2.65}$$

Cela permet donc de réécrire :

$$b_{N-i} = \left\lfloor \frac{\sum_{j=0}^{N-i-1} a_j P_j}{P_{N-i}} + a_{N-i} \right\rceil = a_{N-i} \tag{2.66}$$

Finalement, la preuve par récurrence est faite. Il est vrai, sous les hypothèses du théorème, d'extraire $\{a_j\}_{0 \leq j \leq N-1}$ depuis $A = \sum_{j=0}^{N-1} a_j P_j \mod K_p^n$. □

Avant d'appliquer à notre scénario tous les outils permettant d'assurer la confidentialité des données que nous avons présentées, regardons comment assurer l'intégrité ou l'authenticité de ces dernières.

2.1.5 Assurer l'intégrité des données avec le tatouage

Jusqu'à présent, nous avons montré comment assurer la confidentialité des données au travers d'algorithmes de chiffrement. En ce qui concerne l'authenticité et l'intégrité des données (c.f. Section 1.1.4), différentes stratégies peuvent être mises en œuvre telles que :

— Les MAC (*message authentification codes* [KBC97]) pour lesquels l'idée générale consiste à calculer le *hash* (*e.g.* MD5 [RD92], SHA-3 [Dwo15], ...) du message à transmettre et de l'envoyer avec ce dernier.
— Les signatures numériques telles que DSA [KG13] ou l'ECDSA [JMV01] qui permettent de garantir l'authenticité et l'intégrité d'un message.

Le problème de ces approches est qu'elles ajoutent des coûts de stockage, de calcul et de communication, puisqu'elles génèrent des méta-données. Or, comme nous l'avons déjà précisé, les IMDs sont fortement contraints en ce qui concerne leur batterie, stockage ou processeur. C'est pourquoi nous nous sommes tournés vers une autre solution : le tatouage numérique de données. L'idée du tatouage est d'insérer de façon imperceptible une marque dans un message (*e.g.* un fichier [Bit+17], une image [PHC05a], une musique [Arn00], ...) afin de protéger le contenu, assurer l'intégrité ou faire de l'authentification faible [PHC05b ; Che+10]. Même s'il en reprend les principes, il diffère de la stéganographie qui elle cherche à dissimuler un message dans un contenu [PHC05a] à des fins d'espionnage, par exemple. Dans le cas du tatouage, l'utilisateur des données est informé de la présence de la marque. Le tatouage est une solution de sécurité dite *a posteriori*. Cela signifie que le tatouage numérique ne cherche pas à assurer la sécurité des données lors des échanges et traitement. Son objectif consiste plutôt à permettre d'identifier la source de la fuite de données en tatouant l'identifiant du dernier utilisateur accédant aux données. Aujourd'hui, le principal usage du tatouage demeure la protection de la propriété intellectuelle [RDB96] de données multimédias (films, musique, ...) en tatouant une preuve de propriété dans les données [CMB00]. Par exemple, afin d'assurer l'intégrité, si l'expéditeur et le destinataire connaissent tous les deux la marque et si les données tatouées sont interceptées et modifiées durant le transfert, le destinataire pourra le savoir en comparant la marque retrouvée avec la sienne. Si les deux correspondent, alors l'intégrité est préservée. Dans le cas contraire, le destinataire sait qu'un problème est arrivé durant le transfert.

Le tatouage possède, au même titre que la cryptographie, des propriétés :

— **L'invisibilité** qui caractérise le fait que la marque doit être imperceptible une fois ajoutée au contenu. C'est une propriété fondamentale dans le domaine médical. Le médecin ne doit pas faire une mauvaise interprétation des données.
— **La capacité** d'insertion correspond au nombre de bits de la marque insérée par nombre de bits de données protégées.
— **La robustesse** correspond au fait de savoir si la marque insérée dans le contenu résiste à des modifications des données tatouées. Ces modifications peuvent être

intentionnelles et bienveillantes (*e.g.* compression) ou malveillantes (*e.g.* attaques) voire non intentionnelles (*e.g.* ajout de bruit). Un tatouage robuste permettra alors de pouvoir extraire la marque indépendamment des modifications qu'il subit. On comprend alors que cette propriété n'est pas conciliable avec le contrôle d'intégrité. Pour ce faire, le tatouage doit au contraire être fragile. Cela revient à dire que toute modification du contenu entraînera une modification de la marque.

— **La sécurité** demande quant à elle une clé de tatouage afin de pouvoir extraire la marque du contenu. Ainsi, seul un utilisateur autorisé pourra avoir accès à la marque.

— **La complexité** prend en compte le temps nécessaire à l'insertion et à l'extraction de la marque. Une complexité élevée demandera beaucoup de puissance de calcul et du temps.

— **La réversibilité** permet à partir du contenu tatouée de le restaurer.

Comme nous venons de le voir, le tatouage numérique permet grâce à une marque de vérifier l'intégrité ou l'authenticité des données. Il est intéressant de noter que des solutions combinant le chiffrement et le tatouage existent. Elles sont appelées techniques de crypto-tatouage [Bou+12]. Ces dernières permettent d'assurer la confidentialité des données au travers du chiffrement alors que le tatouage permet quant à lui d'assurer : l'intégrité, la traçabilité ou encore la protection de *copyright* et ceux dans le domaine clair et/ou dans le domaine chiffré. On peut distinguer les schémas pour lesquels la marque est accessible dans le domaine clair [BC16] ou dans le domaine chiffré [Kum+19]. Les premiers demandent généralement que la donnée soit déchiffrée avant de pouvoir vérifier la marque. Concernant les autres, les solutions les plus courantes actuellement sont de type tatouage-chiffrement conjoint (ou *joint-watermarking*). Elles permettent l'accès à la marque dans le domaine chiffré aussi bien que dans le domaine en clair [Sch+12] [Bou+16]. Ces méthodes prennent généralement appui sur des techniques de chiffrement partiel [Lia+06 ; Boh+13 ; XC14] ou homomorphe [Pai99 ; DJ03].

Dans notre travail, nous nous sommes intéressés à un algorithme de crypto-tatouage qui permet ainsi à l'IHM de vérifier l'intégrité des données issues de l'IMD sans pour autant devoir les déchiffrer.

Comme l'IHM n'est pas considérée comme une entité de confiance, nous ne souhaitons pas qu'elle accède aux données en clair. Ainsi, elle ne possède pas la clé de déchiffrement du CLCG qui est utilisé pour sécuriser les données de l'IMD.

Considérons une donnée d_i acquise par l'IMD. Soit b_k un bit de la marque $\{b_k\}_{k=1,\dots,M}$.

L'objectif est de chiffrer et insérer la marque en une opération sur la donnée d_i. Pour ce faire, la donnée d_i est transformée en la donnée tatouée d_i^w. Le principe de notre crypto-tatouage est le suivant : d_i^w est la version de la donnée d_i tatouée si, en faisant la sélection secrète d'un bit du hash de d_i^w, alors ce bit est égal à un bit b_k de la marque. Cette sélection secrète est effectuée grâce à la clé secrète de tatouage K_w. La procédure d'insertion de la marque se résume comme suit :

$$f^e_{K_w}(H(E(d_i^w))) = b_k \tag{2.67}$$

avec $H(\cdot)$ la fonction de hachage ; $E(\cdot)$ le chiffrement CLCG ; K_w la clé secrète de tatouage ; et $f^e_{K_w}(\cdot)$ la fonction d'extraction de la marque paramétrée par K_w. Du côté extraction du message, le lecteur de tatouage utilisera donc $f^e_{K_w}(\cdot)$ pour extraire la valeur d'un bit à une position donnée de d_i^w.

Dans notre scénario, au vu des capacités de calculs limitées de l'IMD, nous avons opté pour la fonction de hachage Quark-U [Aum+13] et pour la modulation *least significant bit* (LSBs) en ce qui concerne le tatouage. Cette dernière consiste à sélectionner les bits de poids le plus faible de d_i avant de changer leurs valeurs. Ce type de tatouage est fragile, non réversible et possède une complexité faible. L'Algorithme 3 résume cette procédure d'insertion. Comme on peut le voir, la procédure est itérative et change les bits de d_i^w jusqu'à ce que (2.67) soit vérifiée. En ce qui concerne la vérification de la marque, il suffit que l'entité souhaitant la faire possède la clé de tatouage K_w et la marque $\{b_k\}_{k=1,\dots,M}$. Grâce à cela, elle peut alors utiliser la fonction $f^e_{K_w}$ et la fonction de hachage $H(\cdot)$, afin d'extraire la marque et vérifier si les deux marques sont bien identiques.

On définit $Dist$ comme le nombre maximal de bits que l'on s'autorise à modifier lors de la phase d'insertion. Notre solution de crypto-tatouage est probabiliste, cela est dû au fait que l'insertion de la marque peut ne pas fonctionner. En effet, on peut imaginer qu'en essayant de modifier les $Dist$ bits de poids faibles de d_i aucune solution n'arrive à satisfaire (2.67). Cependant, un tel évènement est extrêmement rare. Pour s'en assurer, nous allons supposer que la fonction de hachage $H(\cdot)$ choisie soit parfaite. On entend par là, que les sorties de la fonction de hachage sont uniformément distribuées. Alors, dans ce cas, la probabilité de ne pas avoir réussi à insérer la marque est de $1/2^{2^{Dist}}$ qui devient rapidement très faible en fonction de la valeur $Dist$.

La Figure 2.6 résume l'utilisation de ce schéma de crypto-tatouage dans le cadre du projet *Followknee*. Comme montré, il faut au préalable que les trois entités : l'IMD,

Algorithm 3 *Crypto-tatouage.*

procedure $E(d_i^w)$
  $d_i^w = d_i$
  $j = 1$
  while $f^e_{K_w}(H(E(d_i^w))) \neq b_k$ **et** $log_2(j) \leq Dist$ **do**
    $d_i^w = d_i \oplus j$
    $j = j + 1$
  end while
  return $E(d_i^w)$
end procedure

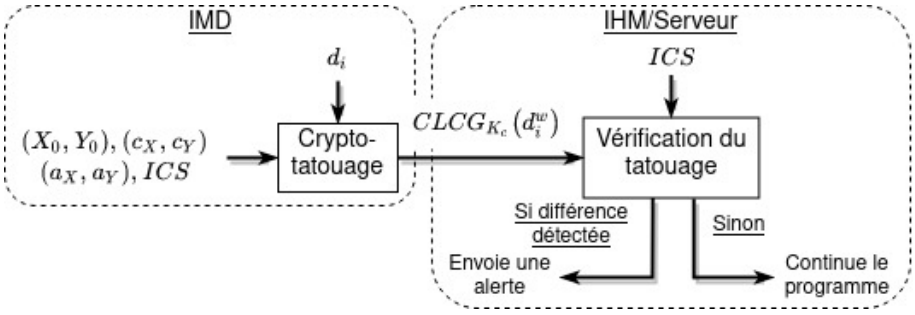


FIGURE 2.6 – Technique de crypto-tatouage pour le contrôle d'intégrité.

l'IHM et le Serveur échangent la marque $\{b_k\}_{k=1,\ldots,M}$. Cette dernière servira au contrôle d'intégrité et sera dénotée ICS pour *integrity control sequence*. L'ICS est intégrée à l'IMD au moment de son initialisation et est insérée dans chaque donnée que l'IMD transfère grâce à la technique de crypto-tatouage. Du fait que l'IMD puisse vouloir envoyer plus de données que la longueur de la marque ICS, cette dernière est alors insérée de façon répétitive. De leur côté, grâce à l'ICS, K_w et la fonction d'extraction $f^e_{K_w}(\cdot)$, l'IHM et le Serveur peuvent vérifier l'intégrité des données chiffrées par le CLCG envoyées par l'IMD.

Il est important de noter que notre technique de crypto-tatouage n'est pas sans perte d'information. En effet, la modulation LSB modifie les bits de poids faible de la donnée d_i en modifiant sa valeur. En prenant $Dist = 1$ ou 2, la distorsion induite par le tatouage est très faible et considérée acceptable comme discutée dans [NC08 ; TF19 ; Kof+08 ; BC16 ; Had+20]. Pour éviter cette perte de données, une possibilité est d'utiliser un crypto-tatouage réversible [Kha+14a]. Cette technique permet de vérifier la marque, mais aussi de l'enlever afin de retrouver la donnée originale. Le problème d'un tel type d'algorithme est son coût de calculs (*e.g.*, *sample prediction*, le calcul d'histogramme, la prise en compte des dépassements de valeur [Coa+13]) mais aussi leur besoin en termes de stockage qui

sont plus importants. De ce fait, il n'est pas envisageable de les utiliser dans notre contexte.

Nous venons de définir l'ensemble de notre scénario avec ses enjeux et ses contraintes. Nous avons également présenté les différents outils qui permettent d'assurer la sécurité du scénario tout en permettant de traiter les données. Nous allons montrer dans la suite, comment agencer toutes ces briques pour effectuer toute la chaîne depuis l'acquisition de la donnée jusqu'à son stockage final en passant par ses traitements le tout de façon sécurisée.

2.2 Sécurisation des communications et des traitements d'un implant connecté

2.2.1 Cas d'usage *Followknee* et rappel des hypothèses de sécurité

Avant de rentrer dans les détails de notre protocole de sécurisation, revenons sur le projet *Followknee* qui nous sert de cas d'usage et qui prévoit le développement d'une prothèse de genou connectée. Le scénario est rappelé en Figure 2.1.

Les opérations de filtrage et seuillage doivent être réalisées de manière sécurisée. Pour les raisons que nous avons déjà exprimées plus haut, l'IMD est considéré comme une entité honnête alors que l'IHM et le Serveur sont considérés comme honnêtes mais curieux. Ainsi :

— L'IHM ne doit alors pas être capable de connaître les données de l'IMD.
— Les poids des filtres et seuils utilisés au niveau de l'IHM sont considérés comme une propriété intellectuelle de l'entreprise les ayant trouvés ; entreprise qui ne souhaite pas les divulguer ; et le Serveur ne doit pas y avoir accès.

Dans un second temps, et pour aller plus loin, nous considérerons aussi le cas où l'IHM est corrompue. On expliquera comment le Serveur sera capable de détecter un tel évènement. La Table 2.2 résume quelle entité peut connaître quelle donnée.

2.2.2 Le protocole de sécurisation et traitement complet

Pour sécuriser l'ensemble du scénario et au vu des contraintes qui pèsent dessus, nous allons utiliser les différentes briques dont le fonctionnement a déjà été décrit plus haut dans ce chapitre. Dans un premier temps, de manière à sécuriser les communications et permettre le traitement de données sur l'IHM, voici comme elles sont déployées :

TABLE 2.2 – L'accès aux données pour le scénario ; ✓ indique que l'entité peut avoir librement accès à la donnée.

	IMD	IHM	Serveur
Données de l'IMD d_i	✓		✓
Poids des filtres w_i		✓	
Seuils s_i		✓	
Sortie du filtrage et seuillage : A			✓
Signe de A		✓	✓

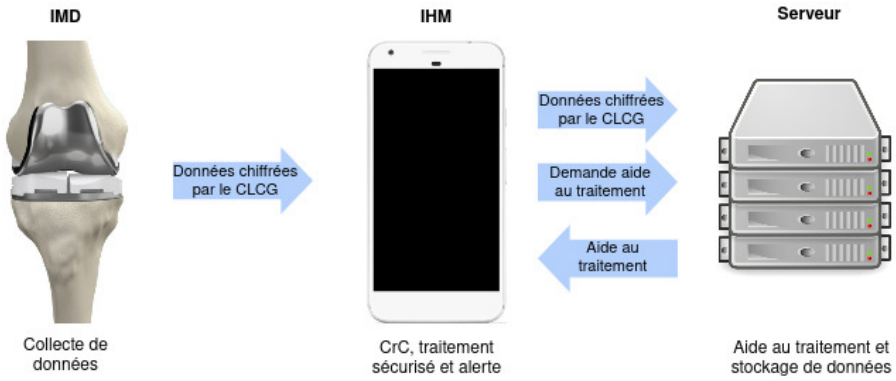


FIGURE 2.7 – Présentation du système complet.

— Un chiffrement léger de type chiffrement à flot est utilisé pour sécuriser les données de l'IMD jusqu'au Serveur.
— Le chiffrement additivement homomorphe de Damgård–Jurik est utilisé par l'IHM pour traiter les données de façon sécurisée.
— Un CrC est exploité au niveau de l'IHM pour convertir une donnée sécurisée par le chiffrement léger en une donnée sécurisée avec le chiffrement de Damgård–Jurik.
— Une technique de crypto-tatouage est appliquée au niveau de l'IMD pour assurer l'intégrité des données de l'IMD chiffrées.

Ces briques sont précisées Figure 2.7. Dans ce qui suit, nous allons commencer par expliquer comment ces briques interagissent au sein d'un protocole que nous proposons et qui sécurise l'ensemble du cas d'usage de *Followknee*. Pour des questions de simplicité, nous commençons par décrire ce protocole dans le cas où les données ne sont pas concaténées avant de montrer comment le faire dans le cas général. Nous décrirons aussi par la suite un protocole qui permet au Serveur de détecter une fraude au niveau de l'IHM.

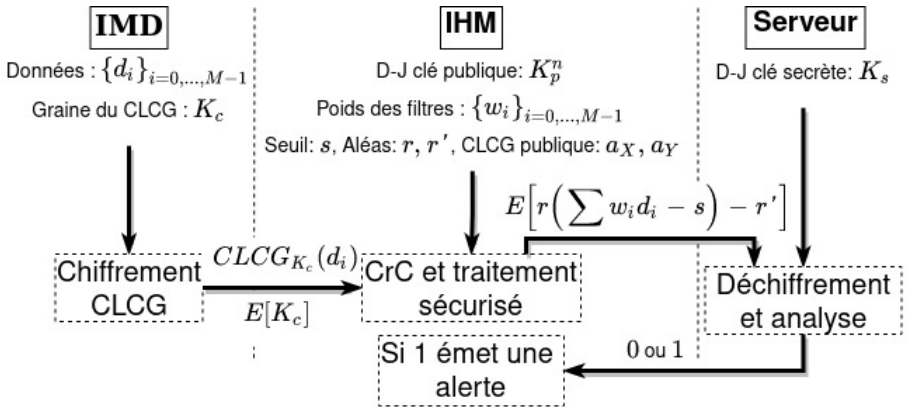


FIGURE 2.8 – Proposition de sécurisation du scénario dans le cas de données non concaténées.

2.2.2.1 Protocole sécurisé sans concaténation

Le protocole que nous proposons fonctionne selon les étapes décrites Figure 2.8. Pour commencer, revenons sur le filtrage de données. Ce dernier soulève quelques points problématiques quand on passe dans le domaine chiffré à savoir comment traiter : des nombres décimaux, rationnels, ... Dans notre contexte, et pour des questions de simplification, nous allons supposer que les données $\{d_i\}_{i=0,\dots,M-1}$, le seuil s et les poids des filtres $\{w_i\}_{i=0,\dots,M-1}$ sont des nombres entiers. Des approches existent pour traiter des entiers rationnels chiffrés homomorphiquement. Certains auteurs proposent par exemple de les quantifier [Bel+19] ou encore d'utiliser des *lattices* afin d'encoder les données [FSW03].

Revenons maintenant sur notre protocole où, dans un premier temps, l'IMD acquiert des données qu'il chiffre à l'aide du CLCG, comme présenté à la Section 2.1.2.1. On rappelle que dans notre contexte le module du CLCG m est égal à la clé publique du cryptosystème de Damgård–Jurik. Dans un second temps, l'IMD envoie à l'IHM la donnée chiffrée $c_i = d_i + Z_i \mod m$. Si cette communication est la première communication entre l'IMD et l'IHM, alors l'IMD envoie également les paramètres qui vont permettre à l'IHM de faire le CrC à savoir :

$$E[K_c] = \{E[X_0, r_{X,0}], E[Y_0, r_{Y,0}], E[c_X, r_{c_X}], E[c_Y, r_{c_Y}]\} \tag{2.68}$$

On considère que l'IHM connaît la clé publique du cryptosystème à savoir K_p^n ainsi que les paramètres publics du CLCG : a_X et a_Y. Sur cette base, l'IHM peut faire fonctionner

le SCLCG (c.f. 2.1.3.1) :

$$E[Z_i] = E[X_{i-1}]^{a_X} E[c_X] E[Y_{i-1}]^{a_Y} E[c_Y] \tag{2.69}$$

Ce qui va lui permettre de calculer le CrC et obtenir :

$$E[d_i] = E[d_i + Z_i] E[Z_i]^{-1} \tag{2.70}$$

avant de pouvoir filtrer plusieurs données de l'IHM et appliqué le seuillage à la sortie du filtre pour lever ou non une alerte.

Pour faire cela, l'IHM va au préalable chiffrer les données envoyées par l'IMD dans le domaine de Damgård–Jurik. L'IHM va reproduire ce schéma pour un certain nombre M de données afin d'obtenir $\{E[d_i]\}_{i=0,\dots,M-1}$. Une fois cet ensemble de données obtenu, l'IHM effectue l'opération de filtrage sécurisée grâce au poids des filtres $\{w_i\}_{i=0,\dots,M-1}$:

$$E[\sum_{i=0}^{M-1} d_i w_i] = \prod_{i=0}^{M-1} E[d_i]^{w_i} \tag{2.71}$$

Le filtrage effectué, le seuillage reste à faire. Pour cela, l'IHM vient retrancher au seuillage la valeur s. L'IHM connaissant le seuil s peut donc librement calculer :

$$E[\sum_{i=0}^{M-1} d_i w_i - s] = E[\sum_{i=0}^{M-1} d_i w_i] E[s]^{-1} \tag{2.72}$$

A ce stade, l'IHM possède la différence entre le seuil s et la sortie du filtre sous forme chiffrée. Aucune décision n'a encore été prise sur le fait de lever ou non une alerte. Connaître le signe de la différence est suffisant pour le faire. Il suffit alors d'effectuer un échange avec le Serveur. Cependant, la mise en œuvre de cet échange doit prendre en compte le fait que le Serveur n'a pas le droit de connaître les poids des filtres. Ainsi, la solution à ce problème consiste à masquer les données comme présenté à la Section 2.1.1.1. Ainsi, pour appliquer ce masquage, l'IHM crée deux nombres aléatoires r et r' tels que $r' << r$ puis calcule :

$$E[r(\sum_{i=0}^{M-1} d_i w_i - s) - r'] = E[\sum_{i=0}^{M-1} d_i w_i - s]^r E[r']^{-1} \tag{2.73}$$

Le résultat de cette opération est envoyé au Serveur. Le Serveur utilise la clé secrète K_s

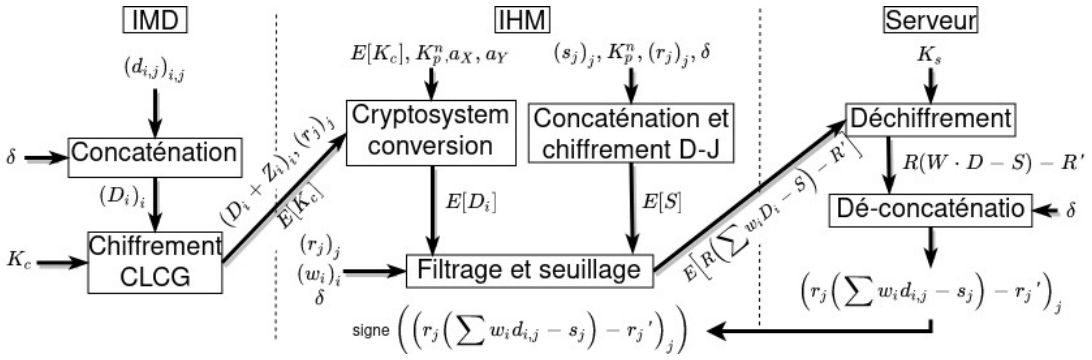


FIGURE 2.9 – Proposition de sécurisation du scénario dans le cas de données concaténées.

pour déchiffrer ce que l'IHM lui a transmis et obtient :

$$A = r(\sum_{i=0}^{M-1} d_i w_i - s) - r' \tag{2.74}$$

En fonction du signe de A, le Serveur va envoyer un 1 (A négatif) ou un 0 (A positif) à l'IHM. L'IHM connaît alors le signe du seuillage suivi du filtrage et peut lever ou non une alerte.

On peut remarquer que dans ce protocole, nous n'avons pas inclus le crypto-tatouage pour des raisons de simplicité. Cependant, ce dernier peut être ajouté en suivant son fonctionnement présenté à la Section 2.1.5.

Une analyse rapide de la sécurité montre que les communications entre les entités sont sécurisées soit par le CLCG, soit par le cryptosystème de Damgård–Jurik. De plus, l'IHM n'a jamais accès en clair aux données médicales grâce au CrC. Elle ne connaît que le signe du traitement. Le Serveur quant à lui, du fait du masquage ne peut pas retrouver les poids ni le filtre utilisé par l'IHM.

Maintenant que la sécurisation du protocole est faite pour le cas où les données ne sont pas concaténées, passons au cas où elles le sont. Pour rappel, la concaténation permet de réduire considérablement les coûts de communication et de calcul ce qui est essentiel dans notre contexte où les ressources sont limitées.

2.2.3 Protocole sécurisé avec concaténation

La Figure 2.9 illustre les adaptations nécessaires au bon fonctionnement de notre protocole avec la concaténation. Nous décrivons dans ce qui suit ces modifications.

Une première adaptation est liée au fait que la concaténation empêche de multiplier

chaque donnée d'un même vecteur par des valeurs différentes. Cela contraint notamment l'opération de masquage que nous proposons maintenant de répartir entre l'IMD et l'IHM. Pour permettre le masquage, l'IMD concatène non plus les données $\{d_{i,j}\}_{j=0,\dots,N-1}$ mais les données pré-masquées $\{d_{i,j}r_j\}_{j=0,\dots,N-1}$ où les $\{r_j\}_{j=0,\dots,N-1}$ sont les aléas utilisés pour faire la multiplication lors du masquage comme présenté dans la Section 2.1.1.1. L'IMD utilise les mêmes aléas $\{r_j\}_{j=0,\dots,N-1}$ pour les M vecteurs concaténés. Il crée donc :

$$D_i = \sum_{j=0}^{N-1} 2^{j\delta} d_{i,j} r_j \tag{2.75}$$

qu'il envoie à l'IHM. Tout comme pour le cas où les données ne sont pas concaténées, l'IMD envoie également la clé du SCLCG $E[K_c]$ au cours du premier échange avec l'IHM. Cependant, l'IMD doit, dans ce scénario, envoyer également les aléas $\{r_j\}_{j=0,\dots,N-1}$. L'IHM peut alors, à l'instar de ce qui a été présenté précédemment, faire le CrC suivi du filtrage des données concaténées :

$$E[\sum_{i=0}^{M-1} w_i D_i] \tag{2.76}$$

On voit ici aussi que chaque filtre w_i est appliqué au vecteur concaténé D_i. De manière plus explicite, l'IHM possède $E[\sum_{j=0}^{N-1} r_j 2^{j\delta} \sum_{i=0}^{M-1} w_i d_{i,j}]$ qui vaut :

$$E[r_0(w_0 d_{0,0} + \cdots + w_{M-1} d_{M-1,0}) + \cdots + r_{N-1} 2^{(N-1)\delta}(w_0 d_{0,N-1} + \cdots + w_{M-1} d_{M-1,N-1})] \tag{2.77}$$

Pour effectuer le seuillage, l'IHM concatène les seuils $\{s_j\}_{j=0,\dots,N-1}$ en les pré-masquants *i.e.* en les multipliant par les aléas $\{r_j\}_{j=0,\dots,N-1}$ c'est-à-dire :

$$E[S] = E[\sum_{j=0}^{N-1} r_j 2^{j\delta} s_j] \tag{2.78}$$

L'IHM calcule alors le filtrage comme suit :

$$E[\sum_{j=0}^{N-1} 2^{j\delta} r_j (\sum_{i=0}^{M-1} w_i d_{i,j} - s_j)] = E[\sum_{j=0}^{N-1} r_j 2^{j\delta} \sum_{i=0}^{M-1} w_i d_{i,j}] E[S]^{-1} \tag{2.79}$$

et finalise l'opération de masquage en créant les valeurs aléatoires $\{r'_j\}_{j=0,\dots,N-1}$, de telle

sorte que $r'_j < r_j$, et les concatène pour ensuite calculer :

$$\begin{aligned} E[A] &= E[\sum_{j=0}^{N-1} 2^{j\delta}(r_j(\sum_{i=0}^{M-1} w_i d_{i,j} - s_j) - r'_j)] \\ &= E[\sum_{j=0}^{N-1} 2^{j\delta} r_j(\sum_{i=0}^{M-1} w_i d_{i,j} - s_j)]E[-\sum_{j=0}^{N-1} 2^{j\delta} r'_j] \end{aligned} \tag{2.80}$$

Comme précédemment, l'IHM envoie $E[A]$ au Serveur pour connaître en retour les signes des opérations de filtrages seuillées. Pour ce faire, le Serveur déchiffre puis dé-concatène $E[A]$ en suivant la procédure présentée dans l'Algorithme 2.1.4.2. Il va alors obtenir l'ensemble des $\{a_j\}_{j=0,\ldots,N-1}$ tels que :

$$a_j = r_j(\sum_i w_i d_{i,j} - s_j) - r'_j \mod K_p^n \tag{2.81}$$

Il peut alors pour chaque donnée dé-concaténée envoyer à l'IHM un 0 si le résultat est positif et 1 s'il est négatif. L'IHM sera alors capable de savoir si les données après filtrage dépassent ou non un certain seuil.

Il est important de noter que pour cet algorithme, la valeur de δ, le paramètre de contrôle de l'opération de *zero padding* de la concaténation, doit être augmentée. En effet, lors du calcul de δ les aléas $\{r_j\}_{j=0,\ldots,N-1}$ et $\{r'_j\}_{j=0,\ldots,N-1}$ n'ont pas été pris en compte. En considérant $\{b_r^j\}_{j=0,\ldots,N-1}$ et $\{b_r'^j\}_{j=0,\ldots,N-1}$ tels que $|r_j| < 2^{b_r^j}$ et $|r'_j| < 2^{b_r'^j}$, on peut majorer les $\{a_j\}_{j=0,\ldots,N-1}$ calculés dans (2.81) par :

$$a_j < 2^{b_r^j}(2^{log_2(M)+b^w+b_j^d} + 2^{b_s^j}) \tag{2.82}$$

Il est alors possible de dé-concaténer les valeurs $\{a_j\}_{j=0,\ldots,N-1}$ du vecteur A si :

$$b_j = \max(log_2(M) + b^w + b_j^d, b_s^j) + b_r^j + 1 \tag{2.83}$$

Ainsi, les données semi-masquées $\{r_j d_{i,j}\}_{j=0,\ldots,N-1}$ sont concaténées en un vecteur D_i de la manière suivante :

$$D_i = \sum_{j=0}^{N-1} 2^{j\delta} r_j d_{i,j} \quad \text{avec} \quad \delta_j = \sum_{k=0}^{j-1} b_k \quad \text{et} \quad \delta = max_j(b_j) \tag{2.84}$$

Il en est de même pour les seuils semi-masqués $\{r_j s_j\}_{j=0,\ldots,N-1}$ et les valeurs aléatoires

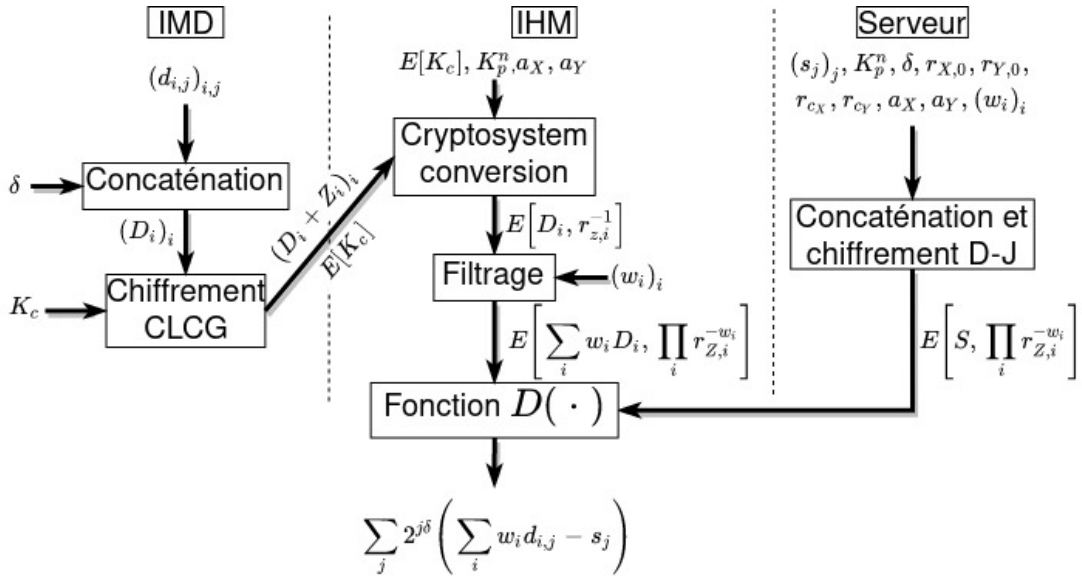


FIGURE 2.10 – Proposition de sécurisation du scénario dans le cas de données concaténées.

$\{r'_j\}_{j=0,\dots,N-1}$ concaténées comme suit :

$$S = \sum_{j=0}^{N-1} 2^{j\delta} r_j s_j \quad \text{et} \quad R' = \sum_{j=0}^{N-1} 2^{j\delta} r'_j \tag{2.85}$$

Notons à nouveau que la technique de crypto-tatouage n'a pas été ajoutée pour plus de simplicité. Elle peut l'être en changeant le chiffrement CLCG par celui présenté à la Section 2.1.5. L'IHM et le Serveur pourront alors vérifier l'intégrité grâce à la marque ICS qui aura été tatouée dans le hash des chiffrés des données de l'IMD. Notons également, qu'il est possible de tatouer le LSB de chaque donnée concaténée afin de diminuer encore d'avantage la probabilité d'échec de l'insertion de la marque c.f. Section 2.1.5. Par exemple, en ne tatouant que le dernier bit *i.e.* $Dist = 1$, pour les N données concaténées $\{d_{i,j}\}_{j=0,\dots,N-1}$ dans D_i, la probabilité d'échec lors de l'insertion est majorée par $1/2^{2^N}$. Or, comme nous le verrons dans la partie expérimentale, la valeur de N est relativement grande, de l'ordre de $N = 30$, avec pour conséquence une probabilité d'échec d'insertion extrêmement faible.

2.2.4 Une variante au scénario *Followknee*

Nous nous sommes également intéressés à une variante du cas d'usage précédent dans laquelle l'IHM et le Serveur connaissent tous les deux les poids des filtres. Les poids n'ont pas de valeur économique. Le scénario proposé est présenté Figure 2.10.

La variante de notre protocole repose sur une propriété importante du protocole de chiffrement de Damgård–Jurik qui, comme montrée dans [Bel+17], permet à partir de deux chiffrés possédant le même aléa r d'en calculer la différence :

$$D(E[a,r],E[b,r]) = a - b \mod K_p^n \tag{2.86}$$

Le protocole permettant de calculer la fonction de différence $D(\cdot)$ est présenté dans [Bel+17]. Il est important de noter que n'importe quelle entité peut calculer cette différence sans connaître la clé privée K_s. Cela est intéressant dans notre nouveau cas d'usage, car l'IHM peut en profiter pour calculer le signe de la différence entre le seuil chiffré par le Serveur et le résultat de l'opération de filtrage qu'elle a calculé.

Pour déployer cette nouvelle approche, le Serveur connaît en plus les aléas qui ont permis de chiffrer la clé privée du CLCG K_c à savoir : $r_{X,0}, r_{Y,0}, r_{c_X}$ et r_{c_Y}. Il possède également les données publiques du CLCG a_X, a_Y, les poids des filtres $\{w_i\}_{i=0,\dots,M-1}$ et les seuils $\{s_j\}_{j=0,\dots,N-1}$.

Sur cette base, le Serveur peut effectuer la concaténation des seuils pour former la donnée suivante :

$$S = \sum_{j=0}^{N-1} 2^{j\delta} s_j \tag{2.87}$$

En reprenant l'expression du SCLCG (2.25), on constate que l'aléa qui accompagne la donnée Z_i est de la forme :

$$r_{Z,i} = r_{X,i-1}^{a_X} r_{c_X} r_{Y,i-1}^{a_Y} r_{c_Y} \tag{2.88}$$

Le Serveur est alors en capacité de calculer les aléas $\{r_{Z,i}\}_{i=0,\dots,M-1}$.

En supposant que l'IHM effectue toujours le même traitement sans changer les poids des filtres $\{w_i\}_{i=0,\dots,M-1}$, le Serveur peut chiffrer S avec le même aléa que le chiffré résultant du filtrage que va faire l'IHM :

$$E[S, \prod_{i=0}^{M-1} r_{Z,i}^{-w_i}] \tag{2.89}$$

qu'il transmet à l'IHM.

De son côté, l'IHM récupère les données chiffrées CLCG de la part de l'IMD, effectue

le CrC afin d'obtenir les vecteurs chiffrés par Damgård–Jurik :

$$E[D_i, r_{Z,i}^{-1}] \tag{2.90}$$

À noter que ce résultat profite du fait que dans la procédure du CrC c.f. Section 2.1.3.2, l'IHM utilise comme aléa le nombre 1 au moment de chiffrer les données envoyées depuis l'IMD. Une fois les données de l'IMD converties en chiffrés de Damgård–Jurik, l'IHM effectue l'opération de filtrage :

$$E[\sum_{i=0}^{M-1} D_i w_i, \prod_{i=0}^{M-1} r_{Z,i}^{-w_i}] = \sum_{i=0}^{M-1} E[D_i, r_{Z,i}^{-1}]^{w_i} \tag{2.91}$$

On peut facilement constater que les deux chiffrés, celui envoyé par le Serveur (2.89) et celui de l'IHM (2.91), ont les mêmes aléas. L'IHM peut utiliser la fonction $D(\cdot)$ afin de calculer :

$$D(E[\sum_{i=0}^{M-1} D_i w_i, \prod_{i=0}^{M-1} r_{Z,i}^{-w_i}], E[S, \prod_{i=0}^{M-1} r_{Z,i}^{-w_i}]) = \sum_{i=0}^{M-1} D_i w_i - S \mod K_p^n \tag{2.92}$$

Autrement dit, l'IHM possède $\sum_{j=0}^{N-1} 2^{j\delta}(\sum_{i=0}^{M-1} w_i d_{i,j} - s_j)$ qui correspond bien à un filtrage suivi d'un seuillage sur données concaténées. Pour dé-concaténer les données, l'IHM utilise l'Algorithme 2 et détermine si les valeurs filtrées dépassent un seuil donné par le Serveur. L'IHM peut alors lever une alerte le cas échéant.

Notons que ce scénario ne nécessite pas l'utilisation du masquage, car le Serveur connaît les poids utilisés par l'IHM. Bien que ce cas d'usage soit d'intérêt, il est bien moins complexe que le précédent, sur la base duquel nous avions établi un protocole plus complet et plus pratique. C'est ce dernier que nous nous proposons de considérer pour la suite.

2.3 Analyse de complexité et résultats expérimentaux

Dans cette section, nous étudions la complexité théorique de notre système aussi bien en termes de temps de calcul qu'en coût de communication et présentons nos différents résultats expérimentaux. Nous comparerons également nos résultats à la solution KBH [KBH09] présentée dans la Section 2.1.1.1 qui est une solution performante et qui pourrait

être appliquée à notre problématique. Nous verrons cependant que notre approche est mieux adaptée.

2.3.1 Complexité du système de traitement sécurisé

La complexité de communication et de calcul pour l'IMD, l'IHM et le Serveur est fonction du nombre de capteurs N et du nombre M de données à filtrer. Notons que, du fait de leur coût calculatoire, la complexité de calcul sera évaluée en termes d'exponentiation et de multiplication modulaires.

En ce qui concerne l'IMD, il concatène $M \times N$ données semi-masquées dans M vecteurs avant de les chiffrer à l'aide du CLCG. Cela correspond à un coût de calcul de $M(N+1)$ multiplications modulaires. Le coût de communication associé au transfert de ces données concaténées chiffrées plus des aléas $\{r_j\}_{j=0,\ldots,N-1}$ est $Mlog_2(K_p^n) + Nlog_2(r_j)$ avec n l'exposant de Damgård–Jurik (voir Section 2.1.1.2).

L'IHM quant à elle commence par appliquer le CrC sur les M données concaténées reçues avant de les filtrer en suivant (2.71). Puis, elle concatène les seuils, comme donné par (2.85), avant de calculer le seuillage comme défini par (2.72) et de compléter le masquage des données (2.80). L'ensemble de ces opérations a une complexité calculatoire de $3M+2$ exponentiations modulaires et de $4M+3N-1$ multiplications modulaires. Le coût de communication entre l'IHM et le Serveur nécessaire pour effectuer le traitement sécurisé des données est lui de $log_2(K_p^{n+1})$.

De son côté, le Serveur doit déchiffrer et dé-concaténer les données que lui envoie l'IHM. Ces opérations ont un coût de 1 exponentiation modulaire ainsi que de $3N + n(n-1) + 1$ multiplications modulaires. Du fait que le Serveur n'envoie que des 0 ou des 1 à l'IHM, ses coûts de communication sont de N.

Les Tables 2.3 et Table 2.4 résument l'ensemble de ces coûts de communication et de calculs théoriques et les comparent à la solution KBH [KBH09]. Comme on peut le voir, du fait de notre stratégie de concaténation notre solution admet de meilleures performances que celles de KBH, en particulier au niveau de l'IMD qui est une entité critique dans notre système.

Après avoir établi et comparé les coûts théoriques de notre solution avec celle de [KBH09], nous avons également effectué certaines expérimentations pour les valider.

TABLE 2.3 – Comparaison de la complexité calculatoire entre notre solution et la solution KBH. On note **E** les exponentiations modulaires et **M** les multiplications modulaires.

	Notre solution	[KBH09]
IMD	M(N+1)**M**	4**E** + 2**M**
IHM	(3M+2)**E** + (4M+3N-1)**M**	5**E** + (4+n(n-1))**M**
Serveur	1**E** + (3N+n(n-1)+1)**M**	4**E** + (4+n(n-1))**M**
Nombre de données traitées	MN	2

TABLE 2.4 – Comparaison des coûts de communication entre notre solution et la solution KBH.

	Notre solution	[KBH09]
IMD	$Mlog_2(K_p^n) + Nlog_2(r_j)$	$2log_2(K_p^{n+1})$
IHM	$log_2(K_p^{n+1})$	$3log_2(K_p^{n+1})$
Serveur	N	$3log_2(K_p^{n+1})$
Nombre de données traitées	MN	2

2.3.2 Résultats expérimentaux

Afin de valider le fait que notre solution puisse être utilisée en pratique, nous avons implémenté le CLCG sur un microcontrôleur de très faible puissance basé sur l'architecture Arm Cortex-M4 32-bit RISC core qui possède :

— une fréquence de 80 MHz,
— une mémoire de 1 Mo,
— ainsi que 320 ko de SRAM.

Dans l'expérimentation qui suit, nous avons utilisé le cryptosystème de Damgård–Jurik [DJ03] paramétré avec un exposant $n = 1$ et un paramètre de sécurité $\lambda = 2048$. Sur cette base, le microcontrôleur est capable de calculer la sortie d'un CLCG toutes les 9 ms. Cela montre que notre solution peut être utilisée en situation pratique réelle, car elle est compatible avec un IMD aux puissances de calcul et/ou de mémoire fortement réduites. Pour des raisons pratiques et d'accès au matériel, nous n'avons pas pu intégrer la solution KBH sur le microcontrôleur. C'est pourquoi pour comparer cette méthode à la notre en pratique, nous les avons implémentées toutes deux sur une machine virtuelle dont la puissance est équivalente à celle d'un iPhone 5, à savoir : un cœur cadencé à 1.3 GHz avec 1 Go de mémoire. Les temps de calculs obtenus sont indiqués dans la Table 2.5. Ils correspondent au traitement de 340 données réparties en 10 vecteurs de données concaténées (*i.e.* $M = 10$ et $N = 34$). Ces résultats montrent que notre solution est bien plus performante que celle de KBH et qu'elle permet de traiter presque 500 données toutes

TABLE 2.5 – Comparaison des temps de calculs entre notre solution et la solution KBH. Les calculs sont faits sur 340 données et sur un processeur de puissance équivalente à celle d'un iPhone 5.

	Notre solution	[KBH09]
IMD	1.7 ms	10671 ms
IHM	648.8 ms	10700 ms
Serveur	32.4 ms	10570 ms
Total	682.9 ms	31941 ms

les secondes alors que le protocole de KBH ne peut en traiter que 94. Maintenant que notre solution a été validée expérimentalement, revenons sur sa sécurité.

2.4 Analyse de sécurité

Avant de rentrer en détail sur l'analyse de sécurité, rappelons que notre schéma est composé de trois entités :

— L'IMD qui acquiert des données médicales, les semi-masques, et les concatène avant de les chiffrer avec le CLCG et de les envoyer à l'IHM.

— L'IHM qui commence par appliquer le CrC sur les données reçues avant de les filtrer, seuiller et masquer de manière sécurisée. Au cours de ce processus, elle envoie au Serveur le résultat du traitement pour que ce dernier le déchiffre et lui transmet également les données reçues par l'IMD pour le stockage.

— Le Serveur stocke les données chiffrées par le CLCG et déchiffre celle chiffrées par Damgård–Jurik avant d'envoyer les signes des résultats à l'IHM pour que celle-ci prenne la décision d'émettre ou non une alerte.

Rappelons également que, comme indiqué en préambule de ce chapitre, l'IMD est considéré comme une entité de confiance alors que l'IHM et le Serveur sont eux considérés comme des adversaires honnêtes mais curieux. Plus clairement, l'IHM ne doit pas pouvoir accéder aux données médicales et le Serveur ne doit pas pouvoir accéder ni au poids des filtres ni aux seuils qui sont utilisés par l'IHM.

Comme définie dans [DGK07], la sécurité d'un protocole en présence de « *static semi-honest adversaries* » est assurée si la « *correctness* » ainsi que la « *privacy* » de ce dernier sont démontrées. Lindell défini ces deux termes dans des conditions normales d'utilisation de la manière suivante [Lin17] :

— La *correctness* (exactitude) signifie que les sorties de chaque partie/entité/processus

durant l'exécution du protocole sont correctes.

— La *privacy* signifie que rien, mis à part les sorties des protocoles, ne peuvent être appris.

Dans la suite, nous nous intéressons à ces deux propriétés pour démontrer la sécurité de notre protocole.

2.4.1 *Correctness*

En ce qui nous concerne, vérifier la *correctness* de notre protocole revient à vérifier la *correctness* des trois entités, c'est-à-dire l'IMD, l'IHM et le Serveur. Voici ce que l'on peut dire pour ces derniers :

— L'IMD chiffre à l'aide du CLCG ses données concaténées et semi-masquées $\{D_i\}_i$, avec i qui appartient à $[\![0, M-1]\!]$, en utilisant la clé secrète K_c. Comme le CLCG est une fonction déterministe, dès lors qu'elle est initialisée avec la même clé K_c, le chiffrement des $D_i + Z_i$ est forcément correct. Par conséquent, les sorties de l'IMD sont correctes.

— L'IHM effectue 5 opérations différentes : le CrC, le filtrage, le chiffrement des seuils concaténés et semi-masqués, le seuillage et enfin la complétion du masquage. Or, ces 5 opérations sont également déterministes. De ce fait, la preuve de la *correctness* de l'IHM est directe.

— En ce qui concerne le Serveur, notons Res le vecteur composé de 0 et de 1 qu'il renvoie à l'IHM. Rappelons que 0 (resp. 1) signifie que le résultat du traitement des données a_j est positif (resp. négatif). Pour établir la *correctness* du serveur, il faut montrer que si a_j est positif (resp. négatif) alors le $j^{\text{ième}}$ bits du vecteur Res est un 0 (resp. 1). Pour ce faire, on peut tout d'abord remarquer que notre concaténation se fait en accord avec le Théorème 1, dès lors si $A > \frac{K_p^n - 1}{2}$ alors on soustrait K_p^n à A pour obtenir :

$$\begin{aligned} A = [r_0(\sum_{i=0}^{M-1} w_i d_{i,0} - s_0 - s_0) - r'_0] + \cdots + \\ 2^{\delta N-1}(r_{N-1}(\sum_{i=0}^{M-1} w_i d_{i,N-1} - s_{N-1}) - r'_{N-1}) \end{aligned} \tag{2.93}$$

En utilisant l'Algorithme 2 de dé-concaténation, le Serveur est alors capable d'accéder aux valeurs $\{r_j(\sum_{i=0}^{M-1} w_i d_{i,j} - s_j) - r'_j\}_{j=0,\ldots,N-1}$. Il peut conclure que $r_j(\sum_{i=0}^{M-1} w_i d_{i,j} -$

$s_j) - r'_j$ est positif (resp. négatif) si et seulement si $\sum_{i=0}^{M-1} w_i d_{i_j} - s'_j$ est positif (resp. négatif). Cela est dû aux choix des aléas r et r' et de la relation qui les relie comme présenté à la Section 2.1.1.1. Par conséquent, le $j^{\text{ième}}$ bits de Res est égal à 0 (resp. 1) si et seulement si le $j^{\text{ième}}$ filtrage des données est au-dessus (resp. dessous) du $j^{\text{ième}}$ seuil ce qui prouve la *correctness* du Serveur.

En prouvant la *correctness* de nos trois entités, la *correctness* de notre scénario est démontrée. Passons maintenant à la preuve de la *privacy*.

2.4.2 *Privacy*

En utilisant la définition précédente, la *privacy* est assurée dans notre protocole si :

- L'IHM est capable d'obtenir les signes des résultats seuillés du filtrage des données des capteurs sans avoir accès aux données médicales en clair.
- Le Serveur doit connaître la différence entre les données et les seuils masqués.

Il est possible de prouver cela en utilisant le formalisme introduit dans [DGK07]. Premièrement, supposons que l'IHM essaie de connaître les données médicales $\{d_{i,j}\}_{i=0,\ldots,M-1;j=0,\ldots,N-1}$.
Avant d'aborder cette preuve, rappelons ce que l'IHM connaît :

- les données concaténées semi-masquées chiffrées $\{D_i + Z_i\}_{i=0,\ldots,M-1}$,
- les aléas $\{r_j\}_{j=0,\ldots,N-1}$,
- la clé du SCLCG $E[K_c]$,
- les signes des résultats seuillés de filtrage envoyés par le Serveur : $Res = \{\text{sign}(r_j(\sum_{i=0}^{M-1} w_i d_{i,j} - s_j) - r'_j)\}_{j=0,\ldots,N-1}$,
- la clé publique du cryptosystème de Damgård–Jurik K_p^n,
- les poids des filtres $\{w_i\}_{i=0,\ldots,M-1}$,
- le terme du *zero padding* δ.

Pour prouver que l'IHM n'apprend pas plus d'information que ce qu'elle peut tirer de Res, il faut montrer qu'il est possible de simuler des données d'entrées de l'IHM sans que cette dernière ne soit capable de distinguer ces données simulées de vraies données, ainsi :

1. Comme le CLCG est un générateur de nombres pseudo-aléatoires, les M données chiffrées qui sont envoyées depuis l'IMD jusqu'à l'IHM peuvent être simulées en prenant M nombres aléatoires appartenant à $\mathbb{Z}_{K_p^n}$. L'IHM n'est alors pas capable de faire la distinction entre ces M valeurs aléatoires et M valeurs que lui enverrait l'IMD dans un contexte légitime.
2. La clé du SCLCG $E[K_c]$ qui est le chiffré de la clé du CLCG par l'algorithme de

Damgård–Jurik peut également être simulée en chiffrant des nombres aléatoires appartenant à $\mathbb{Z}_{K_p^n}$. Comme le cryptosystème de Damgård–Jurik est sémantiquement sûr, l'IHM ne pourra pas distinguer la vraie clé de celle simulée.

3. Les valeurs aléatoires $\{r_j\}_{j=0,\dots,N-1}$ peuvent être simulées en prenant N valeurs aléatoires appartenant à $[\![0, 2^{b_j^r} - 1]\!]$, elles aussi indistinguables de données réelles du point de vue de l'IHM.
4. Le vecteur *Res* renvoyé par le Serveur dépend des données envoyées par l'IMD. Comme une même valeur de *Res* peut être obtenue à partir de différentes entrées, l'IHM ne peut pas tirer d'information de *Res*.

Ces constructions permettent de prouver la *privacy* de l'IHM.

Une analyse similaire peut-être faite pour le Serveur. En effet, le Serveur reçoit : $E[A] = E[\sum_{j=0}^{N-1} 2^{j\delta}(r_j(\sum_{i=0}^{M-1} w_i d_{i,j} - s_j) - r'_j)]$ qu'il déchiffre puis dé-concatène pour accéder aux différents $\{a_j = r_j(\sum_{i=0}^{M-1} w_i d_{i,j}) - r'_j \mod K_p^n\}_{j=0,\dots,N-1}$. En se rappelant la façon dont sont traitées les données négatives dans le domaine de Damgård–Jurik (voir Section 2.2.3), le Serveur peut simuler les valeurs d'entrée en générant de façon uniforme les résultats de filtrage $\{\bar{a_j}\}_{j=0,\dots,N-1}$ tels que :

$$\bar{a_j} \in [\![0, 2^\delta - 1]\!] \text{ si } a_j \geq 0 \text{ et } \bar{a_j} \in [\![-2^\delta + 1, -1]\!] \text{ sinon} \tag{2.94}$$

Comme les données sont masquées, alors les a_j sont uniformément distribués dans $[\![-2^{b_j} + 1, 2^{b_j} - 1]\!]$. Ce qui a pour conséquence de rendre impossible la différenciation des quantités a_j et $\bar{a_j}$. De plus, comme ces deux valeurs ont le même signe, le Serveur ne peut pas les distinguer. Du fait que les entrées du Serveur sont des données chiffrées par Damgård–Jurik qui est sémantiquement sûr, alors le Serveur n'est pas capable de faire la distinction entre les vraies valeurs chiffrées et les valeurs simulées chiffrées. Par conséquent, la *privacy* du Serveur est prouvée.

2.4.3 La sécurité du CLCG

En termes de sécurité, le protocole de Damgård–Jurik a déjà été étudié et prouvé sûr dans [DJ03]. L'analyse de sécurité du CLCG n'a pas reçue la même attention. Néanmoins, ce chiffrement peut être considéré comme sûr sous certaines contraintes. En effet, bien qu'un CLCG, soit la combinaison de deux LCGs, il a été prouvé dans [LT91 ; LEc96] que sous certaines hypothèses, un CLCG peut être approximé par un LCG. La sécurité des LCGs a été étudiées en termes de tests d'aléatoire dans [LS07] et d'attaque dans [Kra92].

En regardant [LS07], il apparaît que lorsque les paramètres du LCG ou du CLCG sont grands (*e.g.* un module $m = 2^{61} - 1$, un multiplicateur $a = 2^{30} - 2^{20}$ et un incrément $c = 0$), le nombre de tests qui ne sont pas passés est relativement faible. Dans notre cas et comme présenté à la Section 2.1.3.2, les paramètres du CLCG dépendent de la taille de la clé publique du cryptosystème de Damgård–Jurik K_p. Du fait que pour des raisons de sécurité la taille en bits de notre clé publique est de 2048, la taille du module du CLCG est également de 2048 bits donc de valeur bien plus grande que 2^{61}. De son côté, [Kra92], montre plusieurs attaques sur le LCG mais, pour être opérationnelles, elles ont besoin de plusieurs sorties $\{X_i\}_i$ pour retrouver les paramètres du LCG et, par conséquent, toute la séquence du LCG. Pour revenir à notre scénario, en supposant que le LCG soit utilisé plutôt que le CLCG, un attaquant n'aurait pas accès aux sorties du LCG $\{X_i\}_i$.

Dans notre contexte, du fait que les séquences sont générées par l'IMD et qu'elles sont utilisées pour chiffrer les données concaténées D_i, on peut se demander s'il est possible de les retrouver à partir des données chiffrées $c_i = d_i + X_i \mod m$. Deux stratégies peuvent être mises en œuvre pour arriver à cette fin :

1. Une recherche exhaustive peut être entreprise. Elle consiste à prendre des données d_i et à appliquer l'attaque présentée dans [Kra92] en testant à chaque fois $X_i = c_i - d_i \mod m$.
2. Une attaque par dictionnaire comme dans [LS05] dans laquelle l'attaquant cherche la donnée d_i la plus probable dans le but de retrouver X_i.

Ces deux attaques semblent complexes dans notre cas ; l'attaque par dictionnaire ayant cependant la complexité la plus faible des deux. Cela est dû aux raisons suivantes :

1. La taille du modulo est égale à la taille de la clé K_p et comme on concatène les données, l'attaquant ne doit pas simplement deviner une valeur d_i mais toutes les valeurs $\{d_{i,j}\}_{j=0,\ldots,N-1}$ concaténées dans D_i. Ceci rend l'attaque par dictionnaire encore plus complexe. Par exemple, en supposant que nos données $d_{i,j}$ sont codées sur 8 bits et que la clé de Damgård–Jurik est codée sur 2048 bits, alors, on peut concaténer 34 valeurs dans un même vecteur. Ainsi, quand bien même un attaquant posséderait un dictionnaire n'incluant que 2^4 valeur pour chaque d_i, il devra alors essayer de deviner 2^{4*34} valeurs avant d'être sûr de retrouver X_i. Une telle complexité est bien trop grande pour être envisagée à l'heure actuelle.
2. Le fait que dans notre protocole les données médicales d_i sont semi-masquées avant d'être chiffrées accentue encore la complexité de l'attaque. En effet, il faudrait faire

une attaque par force brute, car une attaque par dictionnaire n'aurait plus de sens. Cela revient à essayer 2^{2048} valeurs afin d'être sûr de retrouver X_i

Pour résumer un attaquant n'est donc pas capable d'attaquer le LCG pour retrouver les X_i ce qui prouve sa sécurité et par déduction celle du CLCG.

Après avoir parlé de la sécurité de notre protocole, abordons une amélioration de ce dernier prenant en compte une IHM considérée comme une adversaire malicieuse.

2.4.4 Cas d'une IHM malicieuse

Dans ces travaux, nous avons également étudié le cas où l'IHM ne serait pas seulement honnête mais curieuse, mais plutôt malicieuse. Ce scénario est tout à fait crédible, car l'IHM est supposée être un appareil mobile comme un smartphone ou une tablette ; des types d'appareil facilement volés ou hackés.

Supposons que l'IHM va activement essayer de retrouver les données de santé $\{d_{i,j}\}_{i,j}$ en ne suivant pas les instructions du protocole. Dans notre contexte, la meilleure attaque du point de vue de l'IHM serait de faire une attaque par dichotomie. Pour illustrer cette attaque et afin de simplifier l'explication de notre contre-mesure, prenons la version de notre protocole pour laquelle les données ne sont pas concaténées. Pour parvenir à ses fins, le plus simple pour l'IHM est de chercher à obtenir la clé secrète du CLCG K_c. L'IHM n'a cette clé que sous forme chiffrée au travers de l'algorithme de Damgård–Jurik (*i.e.* $E[K_c] = \{E[X_0], E[Y_0], E[c_X], E[c_Y]\}$). Ainsi, pour mener cette attaque, l'IHM à la place d'envoyer au serveur la valeur $A = R(\sum_i w_i D_i - S) - R' \mod K_p^n$ (voir (2.80)) en attendant en retour les signes des différences, envoie la quantité $E[X_0 - X']$,où $X' = 2^{\lambda/2}$ et λ est le paramètre de sécurité de l'algorithme de Damgård–Jurik. Réceptionnant le signe de la différence $X_0 - X'$, l'IHM est donc bien en capacité de mener une attaque par dichotomie en réitérant cette opération en modifiant X'. Ce type d'attaque est logarithmique, ce qui signifie que l'IHM n'aura besoin que de λ échanges entre elle et le Serveur pour retrouver X_0. La même attaque peut être menée sur les autres paramètres de la clé $E[K_c]$. L'IHM sera à même de déchiffrer le CLCG et ainsi accéder en clair aux données médicales. Une attaque similaire existe pour la version du protocole ou les données sont concaténées. Les principes sont les mêmes. De ce fait, nous ne la détaillons pas ici.

Pour se défendre contre un tel évènement, nous proposons d'ajouter à notre protocole un schéma *zero-knowledge* dont le but est de tester si l'IHM calcule correctement les données ou si elle enfreint le protocole. L'idée générale derrière notre schéma *zero-*

knowledge est de demander à l'IHM de re-traiter des données qu'elle a déjà traitées. Dans ce sens, le Serveur va lui envoyer des données D_i sélectionnées secrètement et chiffrées par Damgård–Jurik. Il est ensuite demandé à l'IHM de refaire le traitement normal, mis à part la phase CrC. La détection d'un mauvais comportement de l'IHM lors de l'exécution de ce schéma se fait en comparant les signes des données traitées et retraitées. Ainsi, si lors du premier traitement le signe est négatif (resp. positif), alors que lors du deuxième traitement le signe est positif (resp. négatif) le Serveur saura que l'IHM est corrompue. Il est important de noter que du fait que le cryptosystème additivement homomorphe utilisé est sémantiquement sûr, l'IHM est incapable de savoir quelles données D_i elle doit traiter à nouveau. Elle ne peut pas adapter ses poids et seuils pour essayer de tromper le Serveur.

Maintenant que notre scénario a été présenté, testé et prouvé sûr, passons à la conclusion de ce chapitre.

2.5 Conclusion

Dans ce chapitre, nous avons montré comment il était de possible de concaténer des données en un vecteur chiffré en une seule fois avec un chiffrement additivement homomorphe. De cette manière, nous pouvons pleinement bénéficier de l'espace des clairs du cryptosystème de Damgård–Jurik. Un autre avantage par rapport aux approches qui utilisent le FHE est que les opérations sont plus rapides au détriment des capacités de traitement. En effet, un PHE additif ne permet pas toutes les opérations contrairement à un FHE.

Nous avons développé et expérimenté cette approche dans le contexte d'une prothèse de genou (IMD) connectée. Nous avons également proposé un protocole permettant de sécuriser le traitement des données émises par cette prothèse sur un smartphone qui peut être honnête mais curieux, voire malhonnête. Cela nous a permis de montrer comment notre opération de concaténation permet d'améliorer grandement le débit de la communication de l'implant.

Ce n'est pas la seule originalité de ce protocole. En effet, il couple la concaténation, un chiffrement léger de type CLCG et une opération de conversion de chiffré vers le cryptosystème de Damgård–Jurik (CrC) qui permet à un appareil mobile de traiter ces données chiffrées homomorphiquement sous forme matricielle assurant également un gain en termes de puissance de calcul. Le cas d'usage étudié est, lui aussi, original. Au-delà

du fait qu'il peut aider à prévenir le plus tôt possible le patient d'un signe d'anomalie perçu par la prothèse, et du besoin d'aller voir son médecin rapidement, il considère que ce service rendu (paramètres de filtrage et de seuillage) est la propriété d'une entreprise qui souhaite les garder secret.

Comme nous avons pu le voir, notre protocole englobe un nombre important de contraintes que nous avons cependant pu gérer pour arriver à une solution opérationnelle. Il a été implémenté sur des appareils possédant des caractéristiques (*e.g.* mémoire, puissances) similaires aux appareils utilisés dans de telles applications. Cela nous a permis de constater que nos algorithmes de chiffrement, concaténation et masquage sont performants et qu'ils peuvent être utilisés en pratique avec la possibilité de traiter presque 500 données par seconde. Cette approche est actuellement en cours d'intégration dans le cadre du projet *Followknee*.

Au-delà, la preuve de sécurité de notre protocole a également été établie. Et, nous avons montré comment, dans le cas où l'IHM serait malveillante, le serveur pourrait à l'aide d'un protocole étendu arriver à le détecter.

Pour conclure, la solution que nous avons présentée ici peut être généralisée à l'ensemble du domaine de l'Internet des objets (IoT). En effet, les implants connectés, qui font partie des IoT, constituent une catégorie particulière avec des contraintes d'énergie, de puissance de calcul et de mémoire très fortes. Avec des objets connectés plus puissants, notre approche offrira un gain de performance accru.

Après avoir travaillé sur la sécurisation de traitements linéaires sur données concaténées, dans le prochain chapitre, nous abordons les problèmes du traitement d'images compressées, dans le cadre de l'apprentissage automatique ou *machine learning*.

Chapitre 3

Traitements de données compressées - Apprentissage automatique sur des images compressées

Comme nous l'avons dit dans le Chapitre 1, il est pertinent de pouvoir traiter des données sous forme chiffrée, notamment en santé où les volumes générés sont très importants, dans la mesure où le temps de traitement des données est réduit en évitant l'étape de décompression. L'objectif des travaux présentés dans ce chapitre est d'étudier la possibilité d'utiliser des algorithmes de *machine learning* avec en entrée des images compressées. Ces choix sont motivés par deux facteurs :

1. Ces algorithmes sont de plus en plus employés pour analyser, prédire ou classifier automatiquement des données ou des événements dans tous les secteurs et domaines d'activités où la réutilisation de données est possible : les voitures autonomes, la finance ou encore la biométrie (*e.g.* reconnaissance faciale) [KR16 ; GLC00 ; Nav+17]. Comme présenté dans la Section 1.2, on peut également les utiliser dans le domaine médical [Lit+17] tant pour le dépistage, le diagnostic, le pronostic, la médication, le suivi du patient que pour l'amélioration du système de soins en général. Ils sont majoritairement à la base d'outils d'aide à la décision.

2. Parmi les données de santé, l'imagerie médicale joue un rôle essentiel dans la prise en charge des patients. Comme présenté à la Section 1.1.3, il existe de nombreuses modalités d'imagerie (scanners, IRM, échographies, ...) permettant d'observer de manière précise et non invasive nombre de paramètres comme les tissus, les organes, le squelette, l'activité cérébrale, etc. À l'hôpital, ces modalités génèrent des images de plus en plus nombreuses pour un examen avec une résolution de plus

en plus haute [Eri+98], c'est-à-dire des téraoctets (To) de données annuellement [MBM03]. C'est pourquoi la compression d'images est largement employée afin de réduire l'espace de stockage et la bande passante utilisés. À noter, qu'aujourd'hui, de nombreux de travaux de recherche visent le développement d'algorithmes de *machine learning* en imagerie médicale notamment pour y détecter des signes d'anomalies et ainsi aider le praticien.

Comme nous allons le voir, par construction, les algorithmes de *machine learning* nécessitent une quantité importante de données ; plusieurs milliers d'images afin d'être initialisés [SSP03]. Signalons que, pour fonctionner, le nombre d'images qu'ils traitent est également élevé. De plus, ces algorithmes de *machine learning* doivent accéder aux pixels des images pour fonctionner. Autrement dit, pour qu'une image soit mise en entrée d'un algorithme de *machine learning*, elle doit être décompressée. Ces algorithmes de compression et de décompression ne sont pas instantanés. Ils ont un certain coût de calcul. Nous nous sommes alors interrogés sur la possibilité de mettre en entrée d'un réseau de *machine learning* des images compressées ou partiellement dé-compressées, sur comment le faire et son impact en termes d'efficacité de l'algorithme de *machine learning*. Le standard de compression d'images que nous avons choisi de prendre en compte est JPEG, introduit à la Section 1.4.2. S'il est l'algorithme de compression prédominant pour le grand public, il est accepté par le standard DICOM, le standard de référence en imagerie médicale, et largement utilisé par les professionnels de santé.

Dans ce chapitre, nous avons utilisé des algorithmes de *machine learning* présentés Section 1.2.3 :

— Le réseau de neurones artificiels ou *artificial Neural Network* (NN).
— Le réseau neuronal convolutif ou *Convolutional Neural Network* (CNN).

Ainsi, sur la base de ces algorithmes et d'images compressées en JPEG, nous avons étudié quelles données extraites des différentes étapes de la chaîne de compression/décompression JPEG peuvent être exploitées pour faire l'apprentissage d'un modèle *machine learning* et avec quels gains ou pertes en termes de précision.

Notre travail n'est pas le premier sur cette problématique. Des auteurs tels que Dejean-Servières *et al.* [Dej+17] ont cherché à évaluer l'impact d'une compression avec pertes (*e.g.* JPEG et JPEG 2000), de distorsion d'images (*e.g.* brouillage ou bruit) sur la précision des réseaux de *machine learning*. D'autres travaux [Gue+18 ; FG16 ; UD17] ont essayé d'entraîner des réseaux avec les coefficients de la transformée en cosinus discret de blocs 8×8 pixels (une étape de la compression JPEG). Dans les deux cas, l'idée sous-jacente de

ces travaux est de faire l'apprentissage de réseau de *machine learning* avec des données issues des premières étapes de l'algorithme JPEG. Ils ne sont pas allés plus loin, en exploitant des issues d'autres étapes de la compression JPEG. C'est ce que nous avons fait [Pis+20].

Pour nos expérimentations, nous avons utilisé deux banques de données pour le *machine learning* : MNIST [LCB98] et CIFAR-10 [Kri09]. Nous n'avons pas exploité d'images médicales, non pas en raison de l'absence de jeux de données, mais surtout en raison de puissance de calcul limitée, de la taille des images ou volumes d'images à traiter pour un nombre important de tests à réaliser. Les réseaux de *machine learning* que nous avons explorés sont très classiques voire des standards pour la communauté et servent à remplir des tâches de classification. Ils ont été proposés par d'autres auteurs, appris sur ces mêmes banques de données et obtiennent de bonnes performances. Nous avons cependant eu à les adapter aux cas de données partiellement décompressées. Comme nous le verrons, nous avons également étudié l'influence du taux de compression sur les précisions des modèles obtenus ; rappelons que JPEG est un standard de compression avec perte, contrôlé par le facteur de qualité JPEG. Nous verrons qu'il existe un compromis entre le temps passé à décompresser partiellement des données et la perte de précision induite.

Dans ce chapitre, nous revenons dans un premier temps sur le fonctionnement des NN et CNN. Par la suite, nous décrivons l'algorithme de compression JPEG et comment nous avons créé nos différentes bases de données d'apprentissage. Avant de conclure, nous présentons les architectures utilisées que nous avons déployées et discutons des résultats obtenus ainsi que des futurs approches envisagées afin de les améliorer.

3.1 L'apprentissage automatique par réseaux de neurones

3.1.1 Principes élémentaires des réseaux de neurones

Dans la Section 1.2.3, nous avons introduit le concept de réseaux de neurones artificiels. Comme présenté en Figure 3.1, ces réseaux sont composés d'un ensemble de neurones regroupés en couche prenant en entrée des données avec en sortie le résultat de la tâche à effectuer. Dans le cas d'une image, chaque pixel est une entrée du réseau. Les couches d'entrée et de sortie sont reliées par des couches cachées de neurones. Aujourd'hui, il existe de nombreuses architectures de réseaux ayant différentes propriétés, avec une littérature

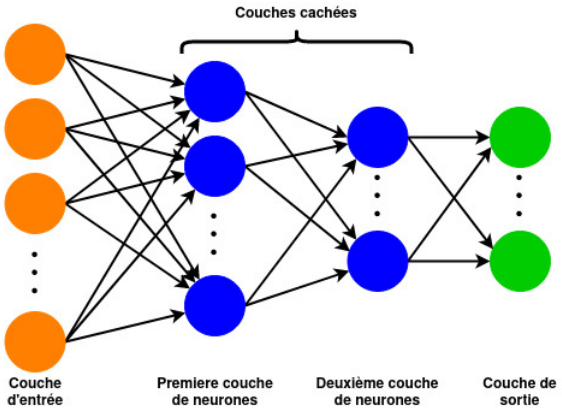


FIGURE 3.1 – Architecture d'un réseau de neurones artificiels.

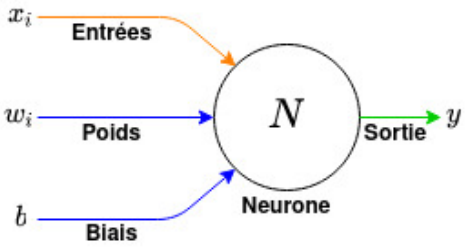


FIGURE 3.2 – Fonctionnement d'un neurone.

très riche à ce sujet. Il n'est pas possible pour nous d'être exhaustif ici et le lecteur pourra se rapporter, par exemple, aux ouvrages [GBC16 ; Gér19]. À noter que ces différentes propriétés ont des conséquences sur les performances d'un réseau. Ce dernier pourra, par exemple, intégrer des couches dites *fully connected* (cas où les neurones d'une couche sont connectés à tous ceux d'une autre couche), avoir un nombre plus ou moins important de couches cachées, utiliser des couches de convolution, ...

Le principe de fonctionnement d'un neurone est le suivant :

— Il possède des poids $\{w_i\}_i$ en même nombre que les entrées et un biais b.

— Il peut ou non être suivi d'une fonction d'activation telle que ReLU, TanH, Sigmoïde, ... notée $f(\cdot)$.

— Une fois toutes les entrées $\{x_i\}_i$ reçues, il calcule :

$$y = f(\sum_i w_i x_i + b) \tag{3.1}$$

— La sortie d'un neurone y peut être l'entrée de plusieurs neurones de la couche suivante ou la sortie du réseau.

Ce fonctionnement est illustré Figure 3.2.

Aujourd'hui, toute la problématique des réseaux de neurones artificiels repose sur leur conception : nombre de couches, de neurones par couches, quelle fonction d'activation à utiliser, ... Une deuxième difficulté est de trouver les poids $\{w_i\}_i$ et biais b qui maximisent la performance de classification, on parlera alors d'*accuracy* ou précision. L'objectif de cette mesure est d'évaluer, pour un ensemble donné dont on connaît la classe, le pourcentage de classification correcte faite par le réseau. Des outils de mesures complémentaires et plus poussés tels que les matrices de confusion peuvent également être mis en œuvre.

On appelle la phase de recherche de poids et biais la phase d'apprentissage. On note que cette dernière nécessite un grand nombre de données *e.g.* 70 000 images composent la banque de données MNIST et 60 000 celle de CIFAR-10.

Il est important de parler des ensembles de données de test et d'apprentissage. Il existe en effet, un risque de sur-apprentissage (*overfitting*). Dans ce cas, le modèle appris est très discriminant sur les données d'apprentissage sans être capable de classer correctement des entrées qui lui sont inconnues. La solution pour éviter cela est de créer deux ensembles d'apprentissage distincts : un utilisé pour l'apprentissage et l'autre pour tester si l'apprentissage a bien fonctionné. On parlera alors de *training dataset* et *test dataset*.

Cependant, la phase d'apprentissage peut être couplée à une phase de recherche d'architecture ou plus simplement de recherche d'hyperparamètres. Si l'on utilise seulement les deux ensembles de données précédents, il peut également y avoir un phénomène de sur-apprentissage en créant un réseau qui répond spécialement bien à ces deux ensembles de données. La solution consiste alors à créer un troisième ensemble de données dit *validation dataset* qui est utilisé pendant la recherche des hyperparamètres. Dans ce contexte, le *test dataset* ne sera utilisé que lorsque la structure ainsi que l'apprentissage seront faits.

Remarquons, qu'il existe plusieurs manières de séparer un ensemble de données dont on connaît les classes associées. Les ratios associés aux différents ensembles peuvent varier en fonction des entrées étudiées (*e.g.* leur nombre de données dont la classe est connue, le nombre d'hyperparamètres, ...).

Rappelons que dans ce travail, notre objectif n'est pas de proposer une architecture, ainsi nos banques de données ne seront séparées qu'en deux ensembles : le *training dataset* et *test dataset*.

Après avoir vu les réseaux de neurones artificiels, nous allons parler des réseaux convolutifs. Une fois ces réseaux définis, nous verrons plus exactement le fonctionnement de leur phase d'apprentissage.

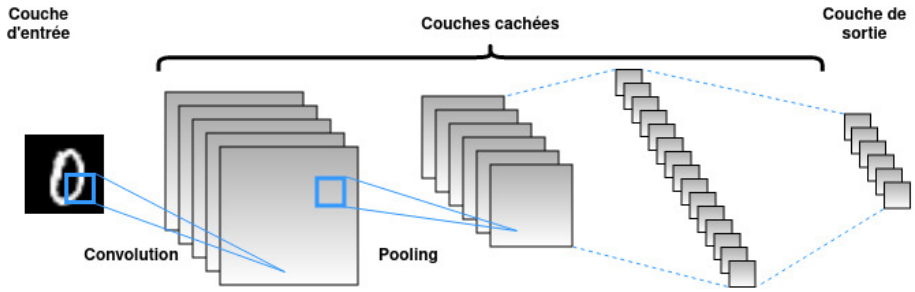


FIGURE 3.3 – Architecture d'un réseau neuronal convolutif.

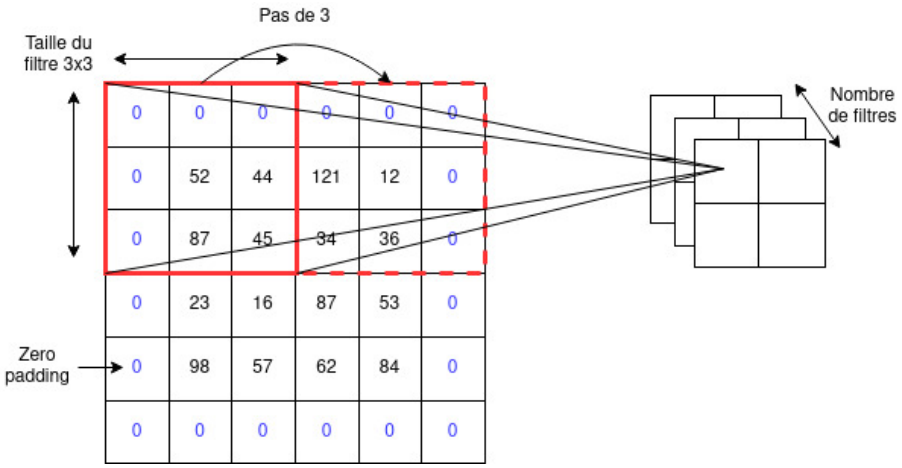


FIGURE 3.4 – Présentation des différents paramètres d'un CNN.

3.1.2 Les avancées proposées par les réseaux convolutifs

Une percée dans le domaine du *machine learning* a été l'introduction des réseaux neuronaux convolutifs (CNN). Ces derniers permettent en général d'obtenir de meilleures précisions que les NN. En particulier, le concours ImageNet de 2012 a été remporté par un CNN comme présenté Section 1.2.3. Un exemple d'architecture CNN est présenté Figure 3.3 dans le cas où l'entrée du CNN est une image en niveau de gris, ici une donnée à 2 dimensions.

Comme on peut le voir, les pixels de l'image sont mis en entrée du réseau. Le réseau effectue une convolution régie par plusieurs paramètres comme illustrée Figure 3.4 dans le cas d'une image de taille 4×4 :

— La taille de la convolution (*kernel size*) indique les dimensions du filtre *i.e.* le nombre de lignes et de colonnes. Dans l'exemple de la Figure 3.4, il s'agit d'une convolution de taille 3×3.

— Le nombre de filtres influence la profondeur de la sortie autrement dit, la taille de

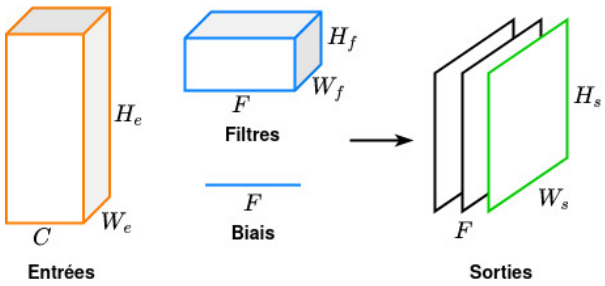


FIGURE 3.5 – Couche de convolution 2D.

la troisième dimension.

— Le pas utilisé (*stride*) est le pas de translation/décalage de la convolution. Pour la Figure 3.4, le pas est de 3.

— Le *zero padding* sert quand la taille du filtre associé au pas ne correspond pas à la taille de l'image d'entrée. Dans l'exemple de la Figure 3.4, comme la taille des convolutions et du pas n'est pas compatible avec la taille de l'image (*i.e.*, il n'existe pas n entier tel que $3 + 3n = 4$). La solution consiste à utiliser un *zero padding* sur l'entrée pour augmenter sa taille à 6×6. Notons que dans ce chapitre et le précédent, les concepts de *zero padding* sont différents, cependant le principe reste le même.

La taille de la convolution ainsi que le pas influencent la taille des sorties (*i.e.* elle influence la taille de ses deux premières dimensions). Le nombre de filtres modifie quant à lui la taille de la troisième dimension.

Dans le cas où le nombre de canaux (la taille de la troisième dimension) de l'entrée est égale à 1, la sortie d'une convolution se calcule comme suit :

$$s_{I,J,K} = (\sum_{i=1}^{H_f} \sum_{j=1}^{W_f} e_{SI+i-1,SJ+j-1} \times f_{i,j,K}) + b_K \tag{3.2}$$

où $s_{a,b,c}$ est la sortie de la ligne a colonne b couche c ; $e_{a,b}$ est l'entrée située ligne a colonne b ; $f_{a,b,c}$ est le coefficient du filtre à la position (ligne a, colonne b, couche c) ; b_c est le biais associé à la couche de sortie c ; S est le pas. Bien sûr, le calcul se complexifie lorsque la couche d'entrée possède plusieurs canaux comme présenté Figure 3.5, où F est le nombre de filtres, C le nombre de canaux de l'entrée, W_e sa largeur et H_e sa hauteur. Il en est de même pour les filtres et la sortie où W_f et W_s représentent respectivement leur largeur et

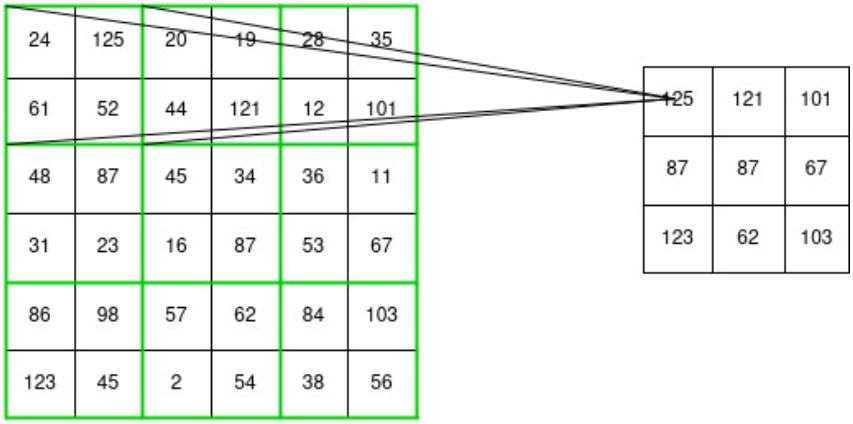


FIGURE 3.6 – Fonctionnement d'une couche *Max Pooling*.

H_f et H_s leur hauteur. Le calcul d'une sortie est alors donné par :

$$s_{I,J,K} = (\sum_{i=1}^{H_f} \sum_{j=1}^{W_f} \sum_{k=1}^{C} e_{SI+i-1,SJ+j-1,k} \times f_{i,j,K}) + b_K \tag{3.3}$$

Dans un CNN, les couches de convolution sont généralement suivies d'autres couches de convolution ou bien de couches de sous-échantillonnage (*pooling*). Leur but est de compresser l'information en réduisant la taille des images intermédiaires. Le *Max Pooling* [Nag+11] est présenté Figure 3.6 à titre illustratif. Ce dernier divise l'image en carré de taille fixe (ici 2×2) puis calcule le maximum de chaque carré. D'autres couches existent comme l'*average pooling*. Ces dernières permettent de réduire la complexité calculatoire en réduisant les dimensions des objets à traiter. Elles peuvent également permettre d'améliorer les performances des réseaux [SMB10].

Pour finir, toutes ces couches sont généralement reliées à des couches de neurones *fully connected* en sortie de réseau afin d'obtenir la prédiction du label.

Nous venons de voir comment fonctionne la classification d'une entrée pour des paramètres (poids et biais) donnés. Cette opération est appelée classification (*feedforward*). Concentrons-nous maintenant sur la recherche des poids et biais de ces réseaux.

3.1.3 Apprentissage pour les algorithmes de *machine learning*

Cette section présente les techniques employées afin d'obtenir les poids et biais qui permettent d'obtenir une précision de classification convenable. Dans le domaine du *machine learning*, il est important de savoir qu'il existe deux types principaux d'apprentissages :

l'apprentissage **supervisé** et l'apprentissage **non supervisé** (c.f. Section 1.2.3).

Dans notre travail, nous nous sommes intéressés à des méthodes d'apprentissage supervisé. Pour prendre en compte des situations où le jeu de données d'apprentissage est relativement restreint, il est possible de faire appel à des méthodes de *data augmentation*. Parmi ces techniques, on peut citer celles qui appliquent des transformations géométriques [Won+16] aux données accessibles telles que : les rotations, translations, ...

D'autres prétraitements sont généralement appliqués sur les données d'apprentissage. On peut, par exemple, les normaliser ou standardiser :

$$\frac{valeur - moyenne}{ecart\ type} \tag{3.4}$$

Le but est de ramener les données dans l'intervalle $[-1, 1]$ avec une moyenne centrée en 0. D'autres prétraitements tels que la PCA [WEG87] sont également envisageables. Ces prétraitements permettent d'éviter les effets indésirables des valeurs aberrantes sur la précision [JS11] ou d'extraire les informations importantes de données volumineuses ce qui peut contribuer à la précision du réseau, mais surtout accélérer les temps de calculs [MH+08].

L'apprentissage des poids et biais des réseaux de *machine learning*, s'appuie le plus souvent sur la rétropropagation du gradient ou *backpropagation*. Dans la première phase, les poids et biais sont initialisés avec, par exemple, des valeurs aléatoires tirées d'une distribution voulue [GB10] (*e.g.* gaussienne) ou avec des poids d'un réseau déjà entraîné.

Cela fait, la phase d'apprentissage en elle-même peut commencer. Pour ce faire, les données annotées sont mises en entrée du réseau, afin que ce dernier les classifie. Ensuite, une fonction de coût (*loss* ou *cost function*) est appliquée afin d'évaluer la différence entre la classe réelle de la donnée et celle octroyée par le réseau. L'objectif de la phase d'apprentissage est de minimiser cette fonction *i.e.* la classification des exemples par le réseau doit se rapprocher de leurs classes. Comme fonction de coût, on peut citer : le MSE (*Mean Squared Error* [JS92], la norme $\mathcal{L}_1$ ou encore l'entropie croisée (*cross-entropy loss*) [KW17].

L'étape de *backpropagation* consiste à rétropropager le gradient de l'erreur dans tous les neurones du réseau de *machine learning*. Le calcul de la *backpropagation* utilise une fonction d'optimisation (*optimizer*) qui permet d'accélérer cette étape. Il existe différentes fonctions d'optimisation comme *Stochastic Gradient Descent* (SGD), *Adaptive Gradient* (AdaGrad), *Root Mean Square Propagation* (RMSProp), *Adaptive Moment Estimation*

(Adam) [Bot10 ; DHS11 ; TH12 ; KB17]. De telles fonctions sont elles-mêmes paramétrables : elles peuvent avoir un pas (*learning rate*) plus ou moins grand, voir fluctuant au cours de l'apprentissage, un *momentum*, etc. Ainsi dans le cas de SGD, la mise à jour d'un poids w s'effectue comme suit :

$$w = w - \eta \nabla E(w) \tag{3.5}$$

où : η est le *learning rate* et $E(\cdot)$ la fonction à minimiser. La phase de classification des données est suivie d'une phase de *backpropagation* qui permet de corriger les poids et biais du réseau conformément à la fonction de coût. Puis, on recommence avec une nouvelle donnée de l'ensemble d'apprentissage jusqu'à les avoir toutes parcourues. On dit que le réseau a appris sur une époque (*epoch*) quand toutes les données de l'ensemble d'apprentissage ont été utilisées une fois. L'arrêt de l'apprentissage peut se faire en suivant plusieurs critères : atteinte d'une précision donnée, diminution du *loss* sous un seuil donné ou encore après avoir atteint un nombre d'étapes (*epoch*) d'apprentissage prédéfini.

Dans ce que l'on vient de présenter, une *backpropagation* est effectuée après chaque *feedforward*, or il est possible de faire le *feedforward* sur plusieurs données et de faire seulement ensuite la *backpropagation* de toutes ces données en une fois. Cette technique est appelée *mini-batch* [Li+14]. Elle permet de réduire les coûts de calcul, la *backpropagation* n'étant pas calculée à chaque fois. Elle permet également d'améliorer la convergence du réseau. La mise à jour des poids s'effectue de la façon suivante :

$$w = w - \frac{\eta}{n} \nabla \sum_{i=1}^{n} E_i(w) \tag{3.6}$$

avec n le nombre de données présentes dans le *mini-batch* et $E_i(\cdot)$ la fonction à minimiser pour la $i^{\text{ième}}$ donnée.

Pour finir, on pourra signaler que d'autres techniques peuvent être employées durant la phase d'apprentissage afin de réduire les temps de calcul, d'augmenter les performances du réseau, ... tels que la technique de décrochage ou abandon (autrement appelé *dropout*). Elle permet de réduire l'*overfitting* en ignorant des neurones sélectionnés aléatoirement durant la phase d'apprentissage.

Maintenant que nous avons analysé le fonctionnement des réseaux de *machine learning* et présenté ceux que nous allons utiliser (le NN et le CNN), passons à l'étude de la fonction de compression JPEG.

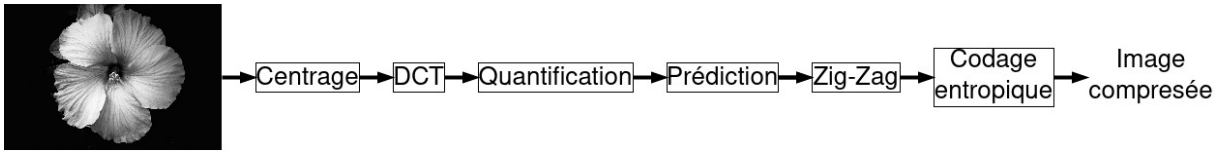


FIGURE 3.7 – Les étapes de l'algorithme de compression JPEG.

3.2 JPEG : le standard de compression d'images

Dans cette section, nous résumons les principes essentiels de cet algorithme. Par ailleurs, afin de simplifier l'explication, nous les présentons dans le cas des images en niveau de gris codées sur 8 bits. Concernant les images en couleur, des étapes de transformation de couleur RGB vers YC_rC_b ainsi que le sous-échantillonnage des composantes C_r et C_b sont à prendre en compte.

Pour revenir à l'algorithme JPEG, on peut considérer que ce dernier comporte sept grandes étapes résumées Figure 3.7 qui sont les suivantes :

— **Centrage de la dynamique** Les niveaux de gris (ng) des pixels de l'image sont centrés sur 0, en soustrayant à chaque pixel 128 ng.
— **Découpage en bloc** L'image est découpée en bloc de 64 pixels selon une grille de taille 8×8. Par la suite, chaque bloc sera traité indépendamment.
— **Transformée DCT** La transformée en cosinus discret (ou DCT pour *Discrete Cosine Transform*) est appliquée sur chaque bloc. 64 coefficients DCT sont obtenus, chacun associé à une fréquence spatiale (verticale et horizontale). L'intérêt d'une telle transformation est de décorréler le signal image et de compacter l'énergie du signal sur un nombre réduit de coefficients, par ailleurs localisés dans les basses fréquences. Le plus souvent, c'est la DCT-II [ANR74] qui est exploitée. La transformée DCT B_f est donnée par :

$$B_f(i,j) = \frac{2}{N} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} B(x,y) \times \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right] \quad (3.7)$$

où, $B(x,y)$ correspond au pixel du bloc B situé à la position (x,y) ; $B_f(i,j)$ est le coefficient DCT positionné à la fréquence spatiale (i,j) ; et $C(u)$ est une fonction

définie par le standard comme suit :

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{si } u = 0 \\ 1 & \text{si } u > 0 \end{cases} \tag{3.8}$$

— **Quantification** C'est cette opération qui est au cœur du contrôle du compromis taux de compression et préservation de la qualité de l'image. La quantification introduit une perte d'information et ce faisant, permet d'augmenter le taux de compression. Cette étape tire avantage des capacités visuelles humaines. En effet, l'œil a du mal à distinguer les détails des images, des informations associées aux coefficients DCT hautes fréquences. Ainsi, chaque coefficient de la DCT est quantifié par un scalaire, un pas de quantification, dont la valeur est fonction de la position fréquentielle du coefficient. Plus clairement, la version quantifiée du coefficient $B_f(i,j)$ est obtenue suivant :

$$B_f^q(i,j) = \text{round}\left(\frac{B_f(i,j)}{\Delta(i,j)}\right) \text{ pour } i,j \in [\![0,7]\!] \tag{3.9}$$

Le scalaire $\Delta(i,j)$ est donné par une matrice de quantification Q. Cette matrice dépend elle-même du facteur de qualité JPEG δ utilisé, le paramètre de JPEG accessible à l'utilisateur. Ce facteur est un entier dont la valeur appartient à $[0, 100]$, 100 étant la valeur pour laquelle la qualité de l'image est préservée à 100% ; à quelques erreurs d'arrondis cependant, du fait du passage de l'image codée en entier à l'espace DCT où les coefficients sont des nombres réels. Ainsi, plus le facteur de δ est grand, plus la qualité de l'image est préservée et moins la compression est efficace. Par conséquent, pour une qualité de 100, tous les éléments $\Delta(i,j)$ de la matrice de quantification Q valent 1. La matrice Q est la plupart du temps définie au moment de l'implémentation de l'algorithme. On présente un exemple de matrice de quantification associé à une qualité de 80 dans la Figure 3.8. On constate que les scalaires de quantification $\Delta(i,j)$ sont élevés pour les fréquences élevées et faibles dans le cas contraire. À ce titre, JPEG peut être vu comme un filtre passe-bas. Il est fonction du facteur de qualité δ car, plus le facteur de qualité est bas plus les détails de l'image sont perdus.

— **Prédiction du coefficient DC** Ce dernier correspond au coefficient DCT quantifié $B_f^q(0,0)$, qui est proportionnel à la moyenne des niveaux de gris des pixels

$$\Delta_{80} = \begin{bmatrix} 6 & 4 & 4 & 6 & 10 & 16 & 20 & 24 \\ 5 & 5 & 6 & 8 & 10 & 23 & 24 & 22 \\ 6 & 5 & 6 & 10 & 16 & 23 & 28 & 22 \\ 6 & 7 & 9 & 12 & 20 & 35 & 32 & 25 \\ 7 & 9 & 15 & 22 & 27 & 44 & 41 & 31 \\ 10 & 14 & 22 & 26 & 32 & 42 & 45 & 37 \\ 20 & 26 & 31 & 35 & 41 & 48 & 48 & 40 \\ 29 & 37 & 38 & 39 & 45 & 40 & 41 & 40 \end{bmatrix}$$

FIGURE 3.8 – Exemple d'une matrice de quantification de l'algorithme JPEG pour un facteur de qualité 80.

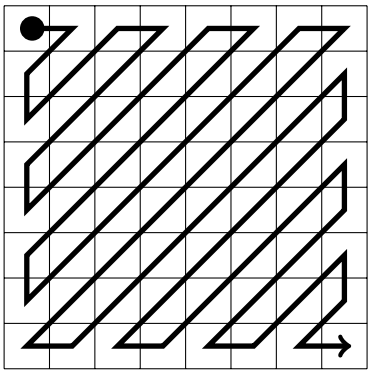


FIGURE 3.9 – La réorganisation en Zig-Zag des coefficients DCT dans l'algorithme JPEG.

du bloc. Sa valeur est prédite en utilisant la valeur du DC du bloc précédent. L'idée derrière la prédiction est de tirer profit de la corrélation qui existe entre le niveau de gris moyen de deux blocs adjacents. Cela permet de réduire la taille de l'information à compresser.

— **Transformée en Zig-Zag** La transformation Zig-Zag est illustrée dans la Figure 3.9. Son objectif est de réarranger les blocs à 2 dimensions en un vecteur d'une seule dimension. Cette réorganisation permet de passer d'une représentation fréquentielle à deux dimensions à une version monodimensionnelle, tout en s'assurant que deux coefficients DCT AC consécutifs sont de fréquence spatiale proche. Cela permet de maximiser les séquences de 0 se trouvant à la fin du vecteur. En effet, ces valeurs étant associées à des hautes fréquences, d'énergie faible et, quantifiées par un pas de quantification relativement grand, après arrondi, elles sont souvent mises à 0. Ces longues séquences permettront par la suite d'optimiser le taux de compression du codeur entropique qui suit.

— **Codage entropique** C'est le codage entropique qui génère l'essentiel du flux binaire JPEG avant encapsulation avec les différents tags/marqueurs indiquant, par exemple, le début/la fin de l'image, la fin d'un bloc, etc. Dans notre contexte, nous utilisons l'algorithme de Huffman mais d'autres algorithmes tels que le codage arithmétique peuvent être employés. Le codage combine le *Run Length Encoding* (RLE) avec deux codages de Huffman. Pour chaque bloc, la valeur de prédiction de l'erreur DC est remplacée par un couple ($Size$, $Value$) où : $Value$ est la représentation binaire signée de la prédiction de l'erreur DC. $Size$ est le mot de code de Huffman qui spécifie la taille en bits de $Value$. Ensuite, les coefficients AC sont codés séquentiellement en commençant par appliquer le RLE suivi du codage de Huffman. Ainsi, les séquences de 0 sont remplacées par le couple ($RunLength$, $Size$) où : $RunLength$ est le nombre de 0 consécutifs entre deux valeurs de AC non nul ; $Size$ est le nombre de bits nécessaire pour coder la valeur du AC non nul qui finit la séquence de valeurs AC égales à 0. Par la suite, le couple ($RunLength$, $Size$) est remplacé par le mot de code de Huffman correspondant auquel on accole la valeur en binaire signée du AC qui suit.

Les étapes ci-dessus présentent l'algorithme de compression JPEG. Pour décompresser, il suffit de partir du flux binaire et de faire les étapes précédentes en sens inverse. En supplément, la compression JPEG utilise un en-tête fournissant certaines caractéristiques sur l'algorithme de compression. Cependant, du fait du peu d'intérêt de cet entête dans le cadre de nos travaux, nous avons fait le choix de ne pas le préciser dans ce document.

3.2.1 Les bases de données partiellement décompressées

Pour nos expérimentations, nous avons constitué différentes bases de données pour la phase d'apprentissage. Partant d'un jeu d'images test compressées JPEG, nous avons décompressé partiellement chacune de ces images et pour chaque étape de l'algorithme JPEG, nous avons constitué une base de données.

Une contrainte forte, que nous avons rencontrée, est le fait que les images compressées par JPEG forment un flux binaire qui est de longueur variable. En effet, deux images de même taille (même nombre de pixels) ont généralement des flux binaires de tailles différentes. Il n'est pas facile de mettre directement ce flux en entrée de réseaux de *machine learning*. On peut alors se demander s'il est possible de le découper pour le mettre en entrée des réseaux. Rappelons que notre objectif est de pouvoir comparer les performances d'un réseau qui travaille soit sur les pixels, soit sur des données d'images partiellement

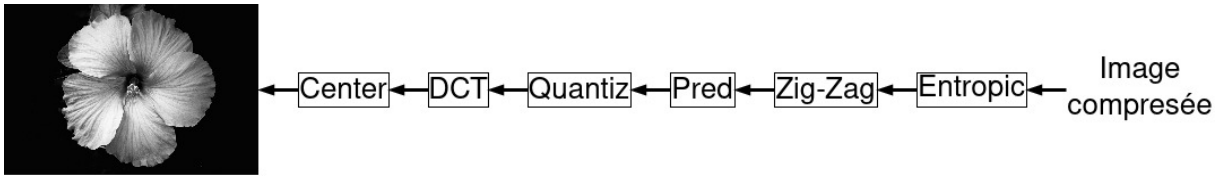


FIGURE 3.10 – Les bases de données créées durant la décompression de l'algorithme de JPEG.

compressées. Pour résoudre ce problème, il faut trouver un moyen fiable pour découper ces flux de différentes longueurs et les harmoniser. Ce sujet va au-delà du périmètre de ces travaux de thèse.

En ce qui nous concerne, les bases de données constituées se positionnent toutes après l'étape de décompression entropique du flux binaire. Plus clairement, comme le montre la Figure 3.10, partant du flux binaire, nos bases de données sont les suivantes :

- ***Entropic*** : représente la base de données des images partiellement décompressées obtenues après avoir effectué le décodage entropique.
- ***Zig-Zag*** : comprend les données partiellement décompressées obtenues après la transformée Zig-Zag inverse.
- ***Pred*** : permet de revenir aux données avant la prédiction de l'erreur du DC.
- ***Quantiz*** : restaure les données jusqu'avant l'étape de quantification.
- ***DCT*** : contient les données partiellement décompressées obtenues après la DCT inverse.
- ***Center*** : représente les images totalement décompressées.

Ainsi, une entité souhaitant accéder à la base de données partiellement décompressée *Pred* le fera bien plus rapidement que si elle souhaite accéder aux données totalement décompressées.

Du fait que les CNN que nous utilisons prennent en entrée des images 2D, nous avons ajouté une étape supplémentaire pour la conception de la base de données *Entropic*. Cette base de données est formée de vecteurs d'une dimension composés de 64 valeurs. L'idée est de la transformer en matrice 8×8 en réorganisant les coefficients comme proposé Figure 3.11. Ceci nous permet alors de comparer équitablement l'impact de la décompression partielle sur la précision d'architecture de *machine learning* identique.

Maintenant que la préparation des bases de données a été précisée, regardons quelles sont les banques de données que nous utilisons pour nos tests.

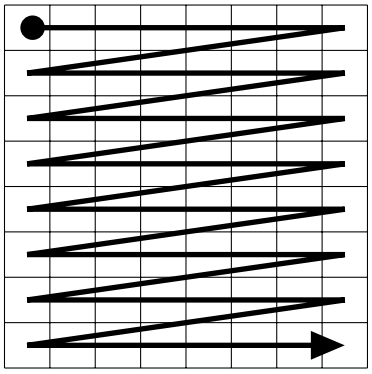


FIGURE 3.11 – Réorganisation en bloc de la base de données *Entropic*.

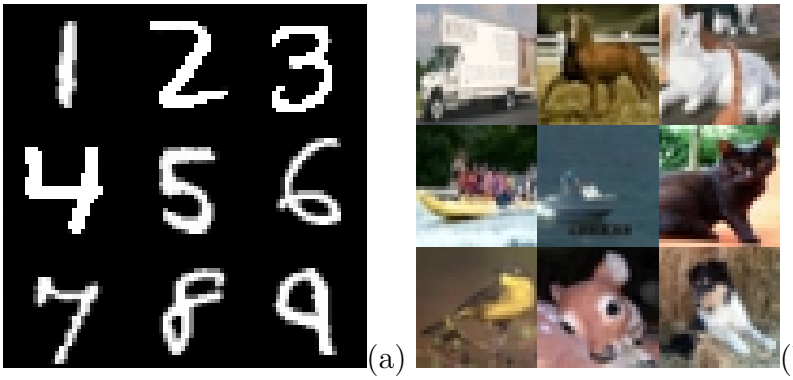
3.3 Résultats expérimentaux

Pour commencer cette section, nous présentons les banques d'images utilisées. Puis, nous étudierons les temps nécessaires à la décompression partielle de chaque base de données : *Entropic*, *Zig-Zag*, *Pred*, ... Chaque base de données se décline pour plusieurs facteurs de qualité δ : 100, 90, 80, 70 et 60. Nous verrons par la suite l'impact de ces bases de données sur la précision des modèles de NN et CNN. Pour des raisons de comparaison équitable des résultats, les réseaux ont été implémentés avec les mêmes hyperparamètres (*e.g.* même nombre d'époques) selon leur type, NN ou CNN. Une fois les bases de données obtenues, elles ont été standardisées avant d'être mises en entrées des réseaux de *machine learning*. L'apprentissage a été implanté grâce aux programmes Keras [Cho+15], TensorFlow [Aba+16] et cuDNN [Che+14] sur une carte graphique Nvidia GTX2080 TI. Les résultats présentés sont donnés en moyenne après 10 tests. Afin d'obtenir les temps de calcul requis pour générer les différentes bases de données partiellement décompressées nous avons implémenté l'algorithme JPEG. Commençons par revenir sur les banques de données utilisées.

3.3.1 Le choix des banques de données

Pour nos travaux, nous avons utilisé les banques de données MNIST et CIFAR-10 dont des exemples sont présentés Figure 3.12 :

— **MNIST** (*Modified National Institute of Standards and Technology database*) Figure 3.12(a) : est une banque de données composée d'images de chiffres écrits à la main. Elle contient au total 70 000 images en niveau de gris de 28×28 pixels. Ces images sont découpées en 60 000 images utilisées pour l'entraînement et 10 000



(a) (b)

FIGURE 3.12 – Exemples d'images appartenant à la banque de données : (a) MNIST ; b) CIFAR-10.

pour le test.

— **CIFAR-10** (*Canadian Institute For Advanced Research*) 3.12(b) : est composée de 60 000 images couleurs de taille 32×32 pixels. Ces images sont réparties en 10 différentes classes : avion, bateau, camion, cerf, chat, cheval, chien, grenouille, oiseau et voiture. Chacune de ces classes comprends 6 000 images. 50 000 images sont utilisées pour l'entraînement et 10 000 pour le test. Dans notre phase d'expérimentation, ces images couleurs sont converties en image en niveau de gris afin de simplifier les étapes de la décompression JPEG.

Après que les banques de données aient été présentées, regardons les architectures NN et CNN qui apprendront à classer ces images.

3.3.2 Les architectures NN

Pour comparer notre travail à d'autres, nous avons réutilisé le NN proposé par Fu et Guimaraes dans [FG16]. Leur architecture n'est composée que d'une seule couche cachée. Nous avons également utilisé un autre réseau de NN qui lui possède deux couches cachées. L'idée étant de voir si le deuxième réseau donne ou non de meilleures performances. Ces deux modèles ont été entraînés sur MNIST durant 200 époques avec une taille de *mini-batch* de 128.

Avant d'analyser les précisions obtenues, regardons la Figure 3.13 de gauche. Cette dernière représente le temps en secondes qu'il faut pour partiellement décompresser MNIST afin de créer les bases de données présentes en abscisse. On observe alors que pour les qualités 70 ou 60, le temps nécessaire pour décompresser entièrement les images de la banque de données MNIST est presque deux fois plus long que le temps nécessaire pour

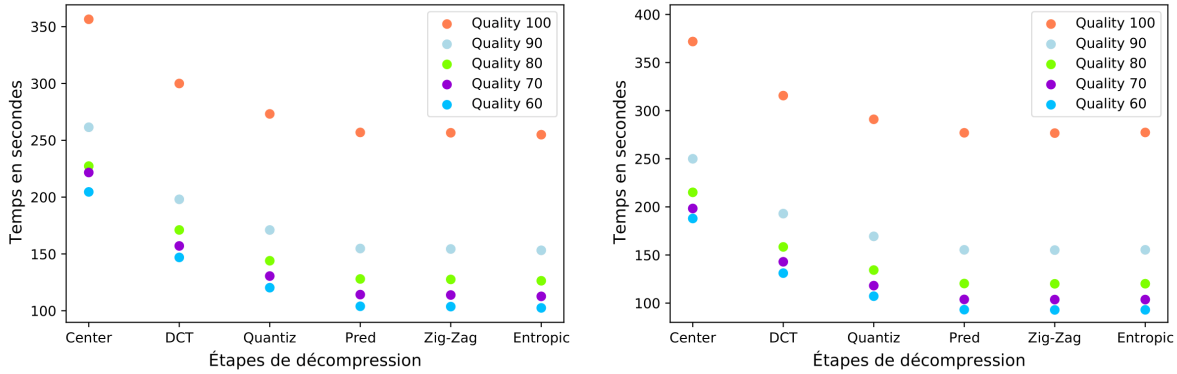


FIGURE 3.13 – Temps de calculs utilisés pour faire la décompression partiel JPEG des différentes bases de données pour plusieurs facteurs de qualité différents pour la banque de données MNIST à gauche et CIFAR-10 à droite.

seulement les décompresser partiellement jusqu'à obtenir la base de donnée *Entropic* (c.f. 3.2.1). Par conséquent, si le volume d'image utilisé est important les gains de temps ne sont pas négligeables. Regardons maintenant la précision obtenue sur ces réseaux.

3.3.2.1 Modèle avec une seule couche cachée

Le premier modèle que nous entraînons est celui de Fu et Guimaraes [FG16] composé d'une unique couche de 1024 neurones activés par la fonction Sigmoïde donnée par :

$$F(x) = \frac{1}{1 + e^{-x}} \tag{3.10}$$

avec x l'entrée et $F(x)$ la sortie. Cette couche est *fully connected* aux entrées et sorties. Durant l'apprentissage, un *dropout* est utilisé. La sortie est une couche *softmax* dont la formule est :

$$F(s)_i = \frac{e^{s_i}}{\sum_k e^{s_k}} \tag{3.11}$$

où s est le vecteur de sortie, s_i les éléments de ce vecteur et $F(s)_i$ la $i^{\text{ième}}$ sortie. Une entrée est classée par le réseau comme appartenant à la classe i si la valeur $F(s)_i$ est la plus élevée.

La Figure 3.14 présente le temps de calcul nécessaire pour faire l'apprentissage des deux réseaux de neurones sur les différentes bases de données (c.f. 3.2.1) avec différentes qualités. On constate que le temps de calcul n'est pas influencé par la base de données d'entrée. Cela peut être expliqué par le fait que les données sont standardisées et que les

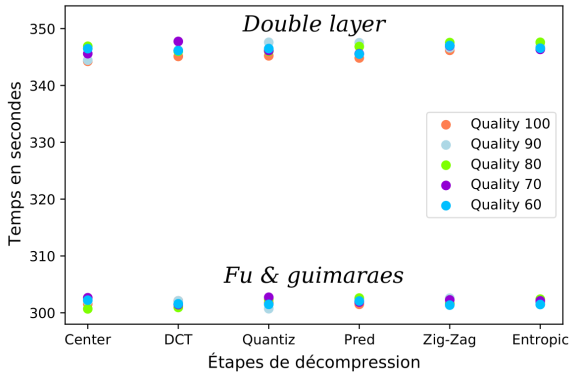


FIGURE 3.14 – Temps de calcul nécessaire pour faire l'apprentissage des réseaux de NN selon les étapes de décompression JPEG en fonction des étapes de décompression et du facteur de qualité JPEG.

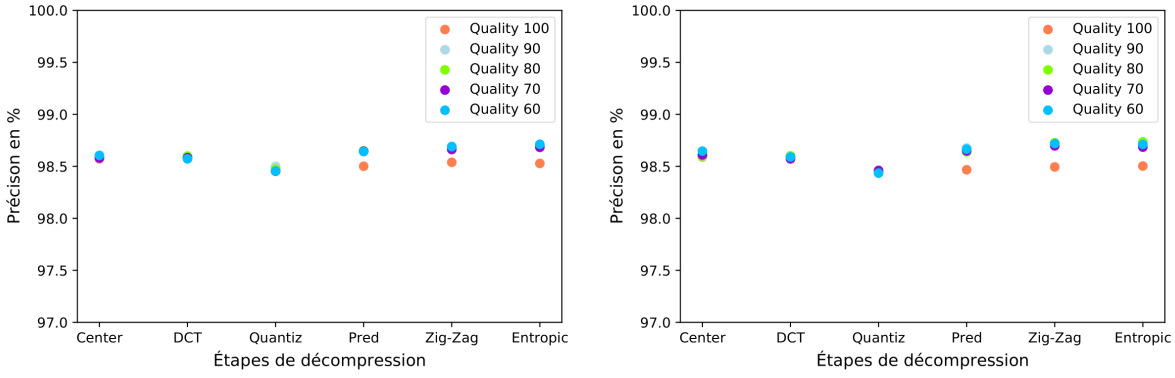


FIGURE 3.15 – Précision du modèle NN à : (gauche) une seule couche cachée, (droite) deux couches cachées en fonction des étapes de décompression et du facteur de qualité JPEG.

hyperparamètres restent inchangés. Ainsi, au niveau du temps de calcul il n'y a pas de différences entre mettre en entrée des réseaux NN des données partiellement décompressées ou directement les pixels.

On présente Figure 3.15 de gauche la précision obtenue en fonction des différentes bases de données partiellement décompressées et du facteur de qualité. On constate que ces paramètres influencent peu la précision du réseau. La performance la plus faible est obtenue pour la base de données *Quantiz* avec une précision de 98.5%. Ces résultats sont cohérents avec ceux obtenus dans [FG16] où les auteurs n'appliquaient que la DCT obtenant une précision de 98%. La faible différence peut être expliquée par le fait que les auteurs ne centraient pas les valeurs de niveau de gris des pixels. De plus, avec uniquement des données dont on a enlevé le codage entropique (*i.e.* qui appartiennent à *Entropic* ou *Zig-Zag*), la précision reste de l'ordre de 98.5%. On peut conclure que, sur MNIST avec un réseau NN composé d'une unique couche cachée, la précision reste bonne peu importe le niveau de décompression partielle des données d'entrée et le facteur de qualité JPEG.

Penchons-nous maintenant sur le réseau composé de deux couches cachées pour voir si les performances obtenues peuvent être améliorées.

3.3.2.2 Modèle à deux couches cachées

Ce réseau est composé d'une première couche de 750 neurones *fully connected* à une deuxième couche de 200 neurones. Les deux couches sont activées par la fonction sigmoïde et durant la phase d'apprentissage un *dropout* est utilisé. La sortie est une couche *softmax*.

La Figure 3.14 montre également les temps de calcul nécessaires pour faire l'apprentissage en fonction des bases de données (niveau de décompression) et du facteur de qualité utilisé. On peut tirer une conclusion similaire à celle pour le réseau à une seule couche cachée à savoir que ces deux paramètres n'influencent pas ou peu le temps de calcul. Cependant, quand bien même le nombre de neurones est légèrement moindre que celui de l'architecture précédente, les temps de calculs sont plus longs. Cela vient du fait que l'architecture du réseau est composée de deux couches cachées.

La Figure 3.15 de droite présente les différentes précisions obtenues. Une fois encore, la précision varie peu que ce soit en fonction des bases de données (niveau de décompression) ou du facteur de qualité JPEG utilisé. Par contre, ce réseau n'améliore pas la précision par rapport au réseau précédent.

Avant de passer à la partie de discussion des résultats, regardons ce qu'il en est pour les CNN.

Layer	Conv1	Conv2		Conv3	MaxPool1	Conv4	MaxPool2		Dense	
Dimension	64	64	Dropout1	64		128		Dropout2	512	Dropout3
Kernel	4×4	3×3	$p = 0.25$	3×3	3×3	3×3	3×3	$p = 0.25$		$p = 0.5$
Stride	2×2	1×1		1×1	2×2	1×1	2×2			

TABLE 3.1 – Architecture Ulicny et Dahyot [UD17].

3.3.3 Les architectures CNN

Tout comme pour les réseaux de NN, nous avons utilisé des réseaux déjà présents dans la littérature :

— le modèle proposé par Ulicny et Dahyot (U&D) dans [UD17],

— le modèle de Keras [Cho+15].

Comme recommandé dans [UD17], les deux modèles ont été entraînés durant 300 époques sur la banque de données CIFAR-10 en utilisant un *mini-batch* de taille 256. L'algorithme d'optimisation SGD a été employé avec un *learning rate* de 0.1 et un *momentum* de 0.85. Le modèle de Ulicny et Dahyot a été entraîné avec ou sans *batch normalization* (BN) après chaque couche.

La Figure 3.13 de droite présente les temps de décompression partielle pour chaque base de données et facteur de qualité JPEG de la banque de donnée CIFAR-10. On peut avoir la même conclusion que pour la banque de données MNIST à savoir : il est possible, en fonction du facteur de qualité et de la base de données utilisée, de gagner jusqu'à 50% du temps de décompression par rapport à une décompression complète. Regardons plus précisément les résultats obtenus sur ces architectures.

3.3.3.1 Modèle d'Ulicny et Dahyot

L'architecture du modèle d'Ulicny et Dahyot [UD17] est présentée Table 3.1. Elle se termine par une couche *softmax*, les autres neurones sont activés par ReLU et les convolutions sont régularisées par la norme $\mathcal{L}_2$.

La Figure 3.16 présente l'influence qu'a le facteur de qualité JPEG et les bases de données utilisées sur la précision du modèle (avec ou sans BN). Pour commencer, on remarque que peu importe la base de données partiellement décompressée utilisée, la précision diminue d'environ 5% entre un facteur de qualité de 100 et un facteur de qualité de 60.

Concernant les bases de données partiellement décompressées, la majorité des pertes de précision advient après l'étape de DCT et de Zig-Zag. Ainsi, peu importe le facteur

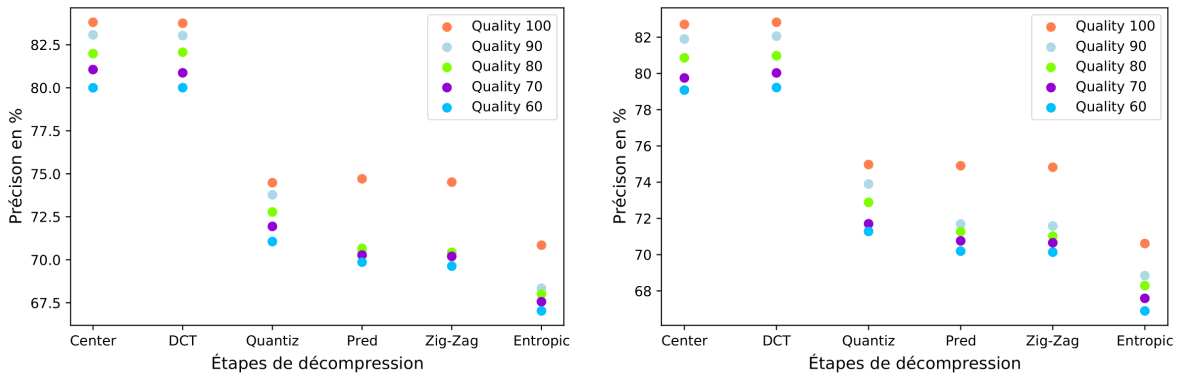


FIGURE 3.16 – Précision du réseau de Ulicny et Dahyot [UD17] à : gauche avec BN, droite sans BN en fonction des étapes de décompression et du facteur de qualité JPEG.

de qualité utilisé, travailler avec des données acquises après DCT admet une perte de l'ordre de 10%. Une explication peut être que la DCT transforme des données spatiales en données fréquentielles. Or, l'étape de convolution, elle, reste inchangée. Néanmoins, une convolution dans le domaine spatial est différente d'une convolution dans le domaine fréquentiel. Une autre explication vient du fait que la DCT dé-corrèle les coefficients et qu'elle concentre l'énergie sur une petite partie d'entre eux. En conséquence, l'architecture CNN devrait être revue afin de mieux s'appliquer à ces problématiques.

La même Figure 3.16 montre que l'étape de quantification dégrade, elle aussi, les performances de l'ordre de 3.5% (pour les facteurs de qualité autre que 100). Un autre effet de la quantification est que pour des facteurs de qualité différents : 90, 80, 70 et 60 la variation dans la précision du réseau est faible (de l'ordre de 2%). Une explication vient peut-être du fait qu'après la quantification, la précision des coefficients DCT est perdue. Le CNN a alors du mal à distinguer entre ces coefficients partiellement décompressés avec différents facteurs de qualité. La petite variation de précision peut être liée à la différence relativement faible entre les coefficients quantifiés. Plus le facteur de qualité est faible, plus les données quantifiées se ressemblent, jusqu'à être égales entre elles une fois 0 atteint.

On peut également conclure à partir de la Figure 3.16 que la prédiction des DC n'a qu'une faible influence sur la précision, ce qui n'est pas le cas du Zig-Zag. Pour cette raison, mettre en entrée des données appartenant à la base de données *Entropic* et réorganiser d'une façon différente des données appartenant à la base de données *Zig-Zag* dégrade les performances d'environ 3%. Cela montre que l'organisation générale des données a une forte influence sur la précision du réseau. Il serait alors éventuellement préférable de

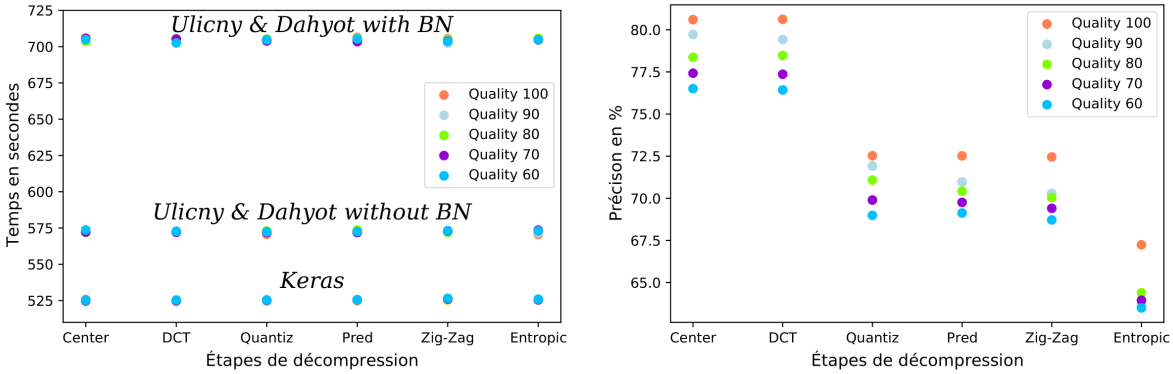


FIGURE 3.17 – À gauche le temps requis pour entraîner le modèle, à droite la précision pour le réseau Keras en fonction des étapes de décompression et du facteur de qualité JPEG.

Layer	Conv1	Conv2	MaxPool1		Conv3	Conv4	MaxPool2		Dense	
Dimension	32	32		Dropout1	64	64		Dropout2	512	Dropout3
Kernel	3×3	3×3	2×2	$p = 0.25$	3×3	3×3	2×2	$p = 0.5$		$p = 0.5$

TABLE 3.2 – Architecture Keras.

travailler avec un kernel de taille 8×8 plus adapté au bloc du protocole JPEG. Il en va de même pour la taille des convolutions. Il pourrait également être intéressant de voir ce que pourrait donner des convolutions à une seule dimension.

Enfin, la Figure 3.16 montre que l'intérêt de la BN est limité au niveau de la précision du réseau alors que son coût calculatoire, présenté Figure 3.17 de gauche, est élevé. L'augmentation est de l'ordre de 135 secondes ou de façon équivalente 22% du temps pour un faible un gain de précision. Nos résultats ne confirment pas ceux de Ulicny et Dahyot. Dans leur article, les auteurs n'avaient pas de telles pertes de précision. Cela peut être dû à différents facteurs :

— Le centrage des valeurs de niveau de gris des pixels.
— Nos images couleurs sont transformées en niveau de gris.
— Tous les paramètres de l'architecture utilisée n'ont pas été rendus publics.

Passons au réseau proposé par Keras [Cho+15] avant de conclure.

3.3.3.2 Modèle de Keras

Les paramètres du réseau Keras [Cho+15] sont présentés Table 3.2. Tout comme le modèle précédent [UD17], il se termine par une couche *softmax*. La Figure 3.17 présente la précision du modèle en fonction des différentes bases de données partiellement décom-

pressées et des différents facteurs de qualité JPEG. Ce modèle est généralement moins performant que celui d'Ulicny et Dahyot. La perte de précision peut aller jusqu'à 5% pour la base de données *Entropic* avec une qualité JPEG de 90. La raison à cela est que les deux réseaux présentent des architectures différentes. Or, comme dit précédemment, les architectures peuvent être plus ou moins performantes en fonction des données à classer. Une autre raison vient du fait qu'ici toutes les convolutions sont de taille 3×3, ce qui ne semble pas adapté aux tailles de blocs 8×8 de JPEG. C'est pourquoi le modèle calcule à certain moment des convolutions entre éléments de blocs différents. De plus, on peut tirer de ce modèle les mêmes conclusions que celles faites pour le modèle de Ulicny et Dahyot. En particulier, les étapes DCT et transformation Zig-Zag influencent fortement et de façon néfaste la précision du réseau.

Discutons maintenant des résultats obtenus.

3.3.4 Discussion des résultats obtenus

Pour rappel, nous avons expérimenté sur MNIST deux réseaux NN [FG16]. La conclusion a montré que ces deux réseaux souffrent peu au niveau de la précision du facteur de qualité JPEG et de l'ensemble de données partiellement décompressées utilisées. Ainsi, entraîner ces réseaux avec des données décompressées appartenant à la base de données *Zig-Zag* avec un facteur de qualité 60 permet de gagner jusqu'à 49% du temps de décompression pour une précision quasiment inchangée par rapport au même entraînement sur données décompressées. Nous avons également montré qu'utiliser des réseaux à deux couches cachées ne permet pas d'améliorer la performance. Il pourrait alors être intéressant de voir si des réseaux plus adaptés à la problématique permettraient d'obtenir une précision encore meilleure.

En ce qui concerne les réseaux CNN, les étapes de décompression ont une forte influence sur la précision finale. On pense en particulier aux étapes de DCT et de réorganisation Zig-Zag. Ainsi, il pourrait être intéressant de proposer une architecture plus adaptée aux données JPEG. En effet, comme présenté au début de ce chapitre les données JPEG sont découpées en bloc de taille 8×8 or, les convolutions que nous utilisons sont souvent de taille 3×3 avec un pas de 1 ou 2. Il pourrait également être intéressant de voir ce que peut donner une convolution dans l'espace fréquentiel pour mieux comprendre la perte de précision. Des convolutions à une seule dimension pourraient être employées afin de ne pas devoir réorganiser les données une fois le codage entropique enlevé. Toutefois, nos résultats montrent qu'il existe un compromis entre la perte de précision et le gain en

temps de calcul fait en ne décompressant pas entièrement les données. Il est, par exemple, possible d'utiliser des données appartenant à la base de données *Pred* ce qui permet de sauver 50% du temps de décompression pour une perte de précision de l'ordre de 7.5% sur l'architecture Keras avec un facteur de qualité JPEG 60.

3.4 Conclusion

Dans ce chapitre, nous cherchions à réaliser un apprentissage avec des réseaux de *machine learning* sur des images compressées ou partiellement compressées. Un apprentissage sur de telles données permettrait de s'affranchir au moins partiellement de la décompression et des temps qui lui sont associés. L'objectif consistait à étudier l'impact des étapes de compression ainsi que du facteur de qualité sur la précision des réseaux de *machine learning*. Cela nous a mené à étudier le fonctionnement de l'algorithme de compression JPEG ainsi que celui des NN et CNN.

Pour ce faire, nous avons créé des bases de données partiellement décompressées correspondantes à chaque étape de l'algorithme de compression JPEG prenant en compte les banques de données MNIST et CIFAR-10 et plusieurs facteurs de qualité JPEG. Dans un premier temps, nous avons testé deux réseaux NN [FG16] qui ont appris sur toutes ces bases de données de la banque MNIST. Cela nous a permis de montrer que pour ces architectures ni l'étape de décompression, ni le facteur de qualité JPEG n'a une grande influence sur la performance du réseau. Cela nous permet de conclure qu'il est possible de gagner presque 50% du temps de décompression des images sans impact sur la précision.

La même expérimentation avec des réseaux CNN a par contre montré des pertes de précision notamment après la DCT et la réorganisation Zig-Zag. Nous avons exposé plusieurs raisons qui peuvent expliquer ce phénomène. On en déduit qu'il existe un besoin de créer des réseaux CNN plus adaptés à ce type de données tant en termes d'organisation des données que de fonctionnement dans un espace transformé. Finalement, un compromis entre le taux de décompression et la perte de précision a pu être établi. Ces travaux ont donné lieu à une publication [Pis+20].

Il serait alors intéressant que les résultats présentés ici soient confirmés par d'autres investigations sur d'autres banques de données et que des réseaux de *machine learning* plus adaptés à ce type de données soient proposés.

En conclusion de ce chapitre, nous avons montré qu'il est possible d'exploiter des données partiellement compressées pour construire des réseaux de *machine learning*. Partant

de ce principe, dans le prochain chapitre, nous chercherons à savoir s'il est possible de décompresser de façon sécurisée, au moins partiellement, des données compressées chiffrées. Plus précisément nous étudierons le codage de Huffman qui est à la base de JPEG. En arrière-plan, l'idée est d'élaborer une chaîne de traitement dans laquelle les images sont tout d'abord compressées puis chiffrées. Une fois chiffrées, elles sont externalisées vers un *cloud*. Après transmission, elles seront alors partiellement décompressées de manière sécurisée afin d'être traitées. Dans ce chapitre, nous avons montré qu'il est possible de faire l'apprentissage et la classification de telles données sous forme partiellement compressée.

Chapitre 4

Décompression partielle sécurisée de données compressées JPEG

Le fil conducteur de ces travaux de thèse est d'essayer de créer une chaîne de traitement permettant de compresser des données avant de les chiffrer et de les externaliser. Nous avons montré dans le chapitre précédent qu'il est possible d'appliquer des méthodes d'apprentissage automatique sur des données partiellement décompressées [Pis+20]. Une fois ces données compressées chiffrées disponibles sur le *cloud*, un objectif est d'offrir la possibilité de les décompresser de façon sécurisée dans le but de les traiter [Niy+20] une fois encore de façon sécurisée. Ces étapes sont schématisées Figure 4.1 dans le cas des images. Dans ce chapitre, nous proposons une solution permettant de partiellement décompresser des données JPEG.

Au cœur de ce chapitre se trouve la sécurisation du décodeur entropique de Huffman, introduit à la Section 1.4.1. La contrainte forte est que le Serveur, réalisant cette action de décompression sécurisée, ne doit en aucun cas pouvoir retrouver les symboles codés, *i.e.* il ne doit ni connaître le nombre de mots de Huffman décodés ni leur longueur.

D'autres auteurs ont cherché à répondre à ce problème. Dans [Ma+18], Ma *et al.* ont proposé de décompresser de façon sécurisée des données JPEG. Pour résumer de façon succincte leur méthode, sous l'hypothèse que chaque bit du flux binaire JPEG est chiffré individuellement, celle-ci compare les premiers bits chiffrés à décompresser avec l'ensemble des mots de code de la table de Huffman correspondante. Lorsqu'une similitude est trouvée, le mot de code est retiré ou remplacé du flux binaire chiffré. L'algorithme continu jusqu'à avoir complètement lu le flux binaire. Le problème principal de cette solution est que le Serveur, sur lequel la décompression sécurisée est opérée, accède en clair à certaines informations. En particulier, il connaît la taille du mot de code décompressé. Or, dans le codage de Huffman, la probabilité d'occurrence est liée à la longueur du mot de code, cela constitue une fuite d'information. Il suffit que le serveur ait une idée *a priori* de la distribution des données pour inférer partiellement ou non les clairs des données.

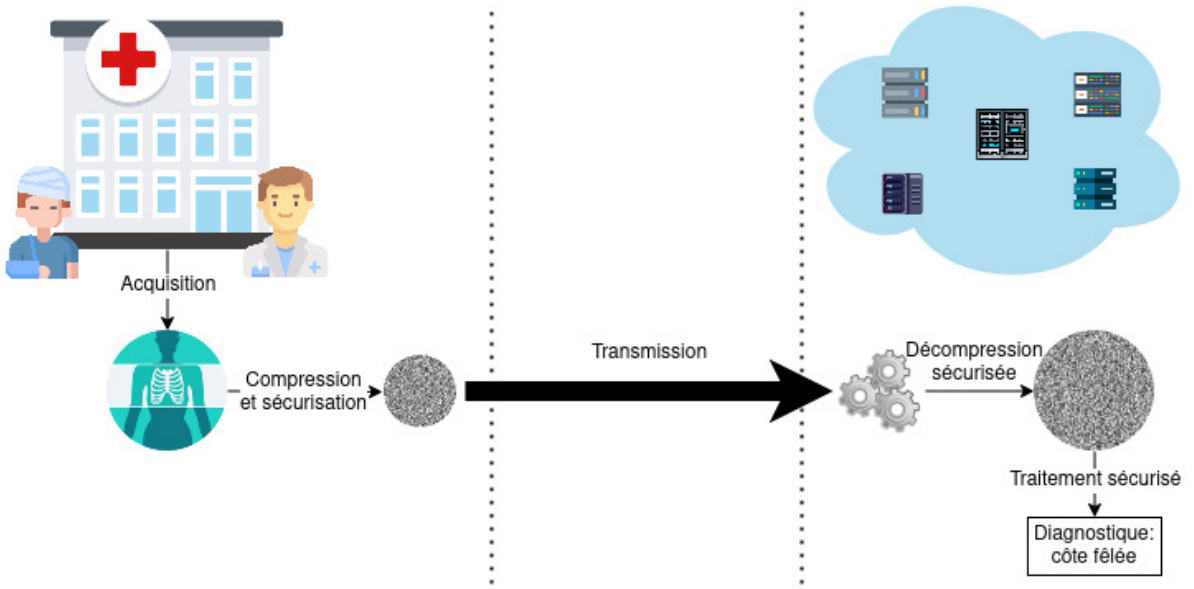


FIGURE 4.1 – Chaîne de traitement sécurisée d'images médicales compressées.

Pour s'affranchir de ce type de problématique, nous avons décidé de travailler au niveau binaire. Comme nous le verrons, le codage de Huffman s'appuyant sur un arbre de décision binaire, cela implique d'évaluer de façon sécurisée un tel arbre. Plus clairement, l'objectif consiste à lire le flux binaire bit par bit et évaluer les nœuds de l'arbre jusqu'à ses feuilles sans cependant permettre à un attaquant de savoir sur quel nœud est le pointeur de décodage, ni s'il est arrivé sur une feuille qui lui donnerait à la fois une idée du mot code décodé ou de sa longueur. Nous reviendrons plus précisément sur ces aspects dans ce qui suit.

Nous verrons également que notre proposition va au-delà du décodage de Huffman. La sécurisation d'un arbre de décision sous ces contraintes peut être réutilisée afin d'évaluer d'autres modèles de *machine learning* et inversement. On peut citer, par exemple, certains arbres de décision (DT) qui permettent de réaliser des tâches complexes en imagerie médicale [Lit+17], détection de tentative d'hameçonnage, catégorisation de logiciel malveillant [KR16], ...

Les auteurs de [TBK20 ; Bos+15] ont proposé des solutions afin de sécuriser un arbre de décision dans l'objectif de déléguer à un serveur la classification des données de clients tout en préservant la confidentialité des données et du modèle. Le problème inhérent de ces méthodes est que le Serveur connaît le nombre et la taille des données d'entrée : il

s'agit d'une vulnérabilité.

Au cours de ce chapitre, nous proposons donc une solution permettant de décompresser de façon sécurisée des mots de code de Huffman. Cette solution est basée sur les cryptosystèmes *fully* homomorphe (FHE). Elle ne souffre pas des problématiques liées aux solutions [Ma+18 ; TBK20 ; Bos+15]. Elle utilise un concept de mémoire et d'évaluation d'une condition : "si alors, sinon" dans le domaine chiffré. Nous reviendrons sur ses performances qui sont meilleures que celles de [Ma+18] dans la partie expérimentale. Pour commencer, introduisons de façon plus formelle l'algorithme de Huffman.

4.1 Décompression de l'algorithme JPEG

Comme présenté à la Section 3.2, l'algorithme du standard JPEG est composé de plusieurs étapes : le découpage en blocs de taille 8×8, le calcul de la DCT, la quantification et le codage entropique. Cette dernière étape s'appuie essentiellement sur le codage de Huffman, déjà introduit à la Section 1.4.1.

4.1.1 Algorithme de compression de Huffman

Le codage de Huffman associe aux symboles les plus fréquents d'une source d'information des mots de code courts et aux symboles rares des mots de code plus longs. Les sorties du codage de Huffman sont donc de longueurs variables. De façon plus formelle, définissons $\mathcal{A}$ un alphabet de N symboles $\{s_i\}_{1 \leq i \leq N}$ comme, par exemple, les niveaux de gris des pixels d'images émises par une source, avec leur probabilité d'occurrence associée $\bar{P} = \{P_{s_i}\}_{1 \leq i \leq N}$ de telle sorte que :

$$\sum_{i=1}^{N} P_{s_i} = 1. \tag{4.1}$$

Le codage de Huffman fonctionne comme suit :

1. La liste de probabilités d'occurrence $\bar{P}$ est triée de façon croissante.
2. Les deux symboles (ou groupes de symboles) de plus petites probabilités d'occurrence notés s_i et s_j sont regroupés. Ces deux symboles (ou groupes de symboles) forment le nouveau groupe (s_i, s_j) dont la probabilité associée est $P_{s_i} + P_{s_j}$.
3. Le groupe s_i est codé par un 0 et le groupe s_j par un 1.

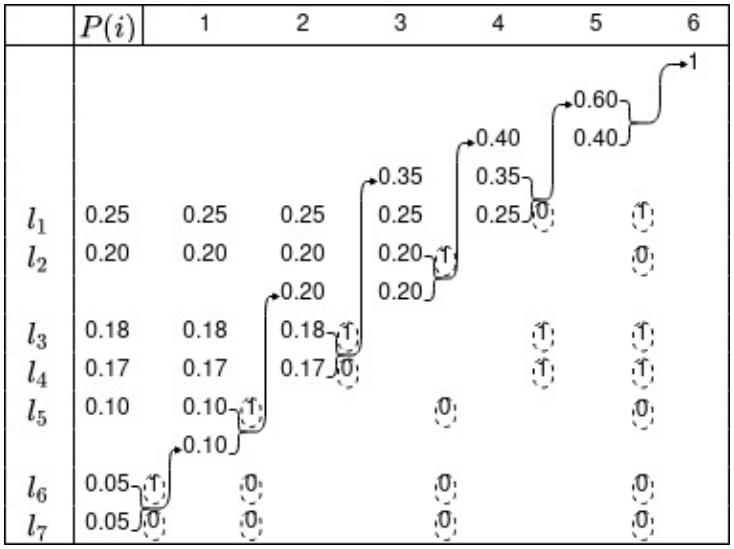


FIGURE 4.2 – Exemple de la construction des mots de code de Huffman.

4. On réitère le processus jusqu'à ce que la liste $\bar{P}$ ne contienne plus qu'un unique groupe de symboles de probabilité 1.

Ce processus permet d'associer de façon unique un mot de code pour chaque symbole.

Un exemple de cette procédure est présenté pour 7 symboles différents ou messages $\{l_i\}_{1\leq i\leq 7}$ dans la Figure 4.2. Dans cet exemple, nous avons choisi d'attribuer 0 au symbole de plus faible probabilité d'occurrence et 1 à celui ayant la plus forte probabilité d'occurrence. Il est important de noter que comme présenté dans [Huf52], ce choix est arbitraire et pourrait être différent. Ainsi, dans cet exemple, les deux symboles les plus rares l_6 et l_7 se voient attribuer les mots de code les plus longs 0001 et 0000 respectivement. Concernant les deux symboles les plus fréquents l_1 et l_2, les mots de code associés sont les plus courts 10 et 01 respectivement.

Ces mots de code peuvent être réarrangés dans un arbre de Huffman noté $\mathcal{H}$ présenté Figure 4.3. Plus précisément, on définit un arbre comme un ensemble composé d'un premier nœud appelé racine duquel part une ou plusieurs branches vers des nœuds fils. Dans le cas où un nœud ne possède lui-même pas de nœud fils, il est alors appelé feuille de l'arbre. En ce qui concerne les arbres de Huffman, ces derniers sont des arbres binaires dans le cas où les mots de code le sont aussi. Dès lors, leurs nœuds ne peuvent avoir qu'au plus deux branches qui mènent soit vers :

— deux nœuds fils,

— un nœud fils et une feuille,

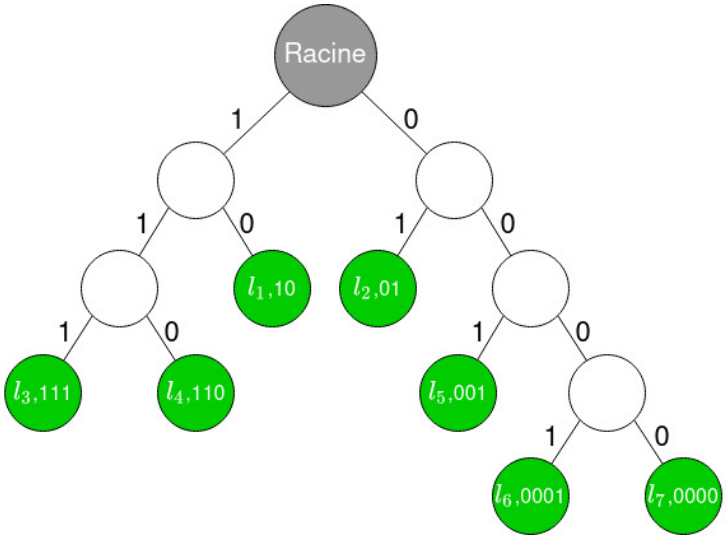


FIGURE 4.3 – Construction de l'arbre de Huffman $\mathcal{H}$ ayant 5 nœuds.

— deux feuilles.

On attribuera à la branche qui part à gauche le bit 1 et celle qui part à droite le bit 0.

Pour décompresser un mot de code, il suffit d'utiliser l'arbre $\mathcal{H}$. Pour ce faire, on se place à la racine puis on lit bit par bit le mot de code à décoder et pour chaque bit on suit la branche correspondante. Une fois tous les bits lus, une feuille est atteinte. Cette feuille correspond au mot de code associé à un unique symbole de la source. Cette procédure peut se traduire de la façon suivante : si le bit lu est égal à 1 prendre la branche de gauche, sinon prendre celle de droite. C'est ce type de condition : "si alors, sinon" que nous allons devoir évaluer de façon sécurisée. Nous montrerons que ceci est possible grâce au chiffrement *fully* homomorphe (FHE). Mais avant cela, il nous faut revenir sur un autre problème consistant à identifier de façon unique chaque nœud et feuille de l'arbre. Cela nous sera utile quand nous chercherons, durant la phase de décompression sécurisée, l'emplacement dans l'arbre où nous nous trouvons.

4.1.2 L'algorithme de parcours en largeur

Pour pouvoir sécuriser l'arbre de décision de Huffman et l'évaluer sans divulguer ses paramètres, il est impératif de pouvoir s'y positionner et repérer. Ainsi, il faut que chaque nœud et feuille soit repéré de manière unique. La Figure 4.3 représente un arbre de Huffman $\mathcal{H}$ ayant 5 nœuds (sans compter la racine). Une solution pour attribuer de façon unique un index à chacun de ces éléments est d'utiliser l'algorithme de parcours en largeur,

Algorithm 4 Algorithme de parcours en largeur d'un arbre binaire.

Soit $\mathcal{Q}$ une queue
On donne à la racine l'index $\leftarrow 0$
$\mathcal{Q}$.enqueue(racine)
$Cpt \leftarrow 0$
while $\mathcal{Q}$ n'est pas vide **do**
    $V \leftarrow \mathcal{Q}$.dequeue()
    for dir dans $[gauche, droite]$ **do**
        if $V.dir$ n'est pas une feuille **then**
            Cpt += 1
            $V.dir$ prends l'index $\leftarrow Cpt$
            $\mathcal{Q}$.enqueue($V.dir$)
        end if
    end for
end while

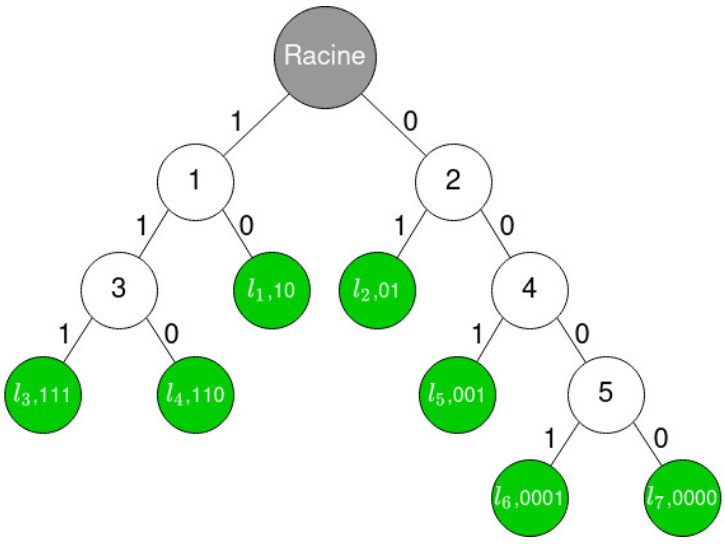


FIGURE 4.4 – Utilisation de l'algorithme de recherche en largeur.

comme on peut le voir sur la Figure 4.4.

De façon synthétique, l'algorithme de parcours en largeur utilise une queue ou file dans laquelle il commence par mettre la racine et lui attribue l'index 0 puis :

1. Il extrait le premier élément de la queue.

2. Il parcourt les nœuds fils de cet élément. Si un nœud fils n'est pas une feuille, l'index suivant lui est attribué et il est ajouté dans la queue.

Cet algorithme reprend à l'étape 1 jusqu'à ce que la queue soit vide. De cette manière, tous les nœuds auront été visités et un unique index leur aura été attribué. Ainsi, être sur le nœud 5 revient à dire que l'on a déjà lu 000 conformément à la Figure 4.4.

Cet algorithme défini, revenons sur la possibilité de calculer de manière sécurisée la décompression de Huffman. Comme précisé lors de la section précédente, le calcul de la décompression de Huffman impose le fait d'évaluer une condition : "si alors, sinon". Ce type de condition n'est en général pas évaluable dans le domaine chiffré. Cependant, le cas particulier suivant :

1: $Bool \in \{0, 1\}$
2: **if** $Bool = 0$ **then**
3: $\quad Res = a$
4: **else** ▷ **si** $Bool = 1$
5: $\quad Res = b$
6: **end if**

est évaluable dans le domaine chiffré *fully* homomorphe. En effet, ce type de condition peut se réécrire sous la forme :

$$Res = (1 + (-Bool)) \times a + Bool \times b \tag{4.2}$$

Cette équation ne fait intervenir que des additions et des multiplications qu'un PHE uniquement additif ou multiplicatif ne pourrait pas évaluer. Cependant, un FHE est capable d'évaluer ces opérations de façon sécurisée. Dans la prochaine section, nous revenons plus longuement sur les cryptosystèmes FHE.

4.2 Additionner et multiplier des données chiffrées : les cryptosystèmes *fully* homomorphes

Les cryptosystèmes FHE, introduits dans la Section 1.3.2, permettent de calculer de façon sécurisée des additions et multiplications sur données chiffrées. Ce type de chiffrement permet plus de possibilités de traitements [LKS16] que les cryptosystèmes additifs tel que celui de Damgård–Jurik présenté à la Section 2.1.1.2. On peut également les distinguer des chiffrements *Somewhat Homomorphic Encryption* (SWHE) qui ne permettent qu'un nombre limité d'opérations. On peut par exemple citer l'algorithme BGN introduit dans [BGN05] par Boneh, Goh et Nissim qui permet de faire un nombre illimité d'additions, mais une unique multiplication. C'est en 2009 que Gentry [Gen+09] propose une solution permettant de faire un nombre illimité de multiplications. L'inconvénient de cette solution est son coût de calcul très élevé.

De façon synthétique, l'idée est d'utiliser un SWHE pour lequel chaque chiffré possède un bruit (*noise*). Plus clairement, un bruit s'ajoute aux données. Après chaque opération, le bruit augmente, *i.e.* le résultat d'une addition/multiplication dans le domaine chiffré est plus bruité que les données originales. Le bruit a pour conséquence de rendre inintelligible le déchiffrement s'il dépasse un certain seuil. Ainsi, le nombre d'opérations est limité. On peut noter qu'il n'y a pas forcément de symétrie entre l'augmentation du bruit induite par une addition ou une multiplication dans le domaine chiffré. La solution de Gentry [Gen+09], appelée *bootstrapping*, consiste à évaluer dans le domaine chiffré le déchiffrement de la donnée sécurisée. Cette opération de déchiffrement est sécurisée, car elle est évaluée dans le domaine FHE. Son résultat reste chiffré, mais permet de "rafraîchir" le bruit *i.e.* le diminuer pour revenir à un bruit permettant de faire de nouvelles opérations homomorphes. Par conséquent, l'utilisation des chiffrements *fully* homomorphes demande une capacité de calcul bien supérieure à celle des chiffrements simplement additivement ou multiplicativement homomorphes.

Des cryptosystèmes plus performants, tels que celui de Brakerski-Gentry-Vaikuntanathan (BGV) [BGV14] ou Brakerski/Fan-Vercauteren (BFV) [FV12], sont depuis apparus. Dans la suite de ce chapitre, nous utiliserons le FHE de BFV pour effectuer nos tests. Regardons dans la prochaine section comment fonctionne ce cryptosystème.

4.2.1 Le cryptosystème FHE de BFV

La sécurité de ce cryptosystème repose sur le problème *ring learning with errors* (RLWE) [LPR10]. Soit un anneau polynomial :

$$R \stackrel{\text{def}}{=} \mathbb{Z}[x]/f(x) \tag{4.3}$$

où $\mathbb{Z}$ est l'anneau des entiers et $f(x) \in \mathbb{Z}[x]$ est un polynôme cyclotomique $\Phi_m(x)$ dont le degré $\varphi(m)$ dépend du paramètre de sécurité λ. Soit un entier $q = q(\lambda) \geq 2$. Cela permet de définir l'anneau $R_q = R/qR$. Soit s un élément tiré de façon uniforme dans R_q (que l'on note $s \xleftarrow{\$} R_q$), soit $\mathcal{X}$ une distribution sur R, $a \xleftarrow{\$} R_q$ et $e \xleftarrow{\$} \mathcal{X}$. L'objectif du problème de décision du RLWE est de distinguer $(a, [a \cdot s + e]_q)$ (où $[\cdot]_q$ désigne la réduction modulo q d'un entier) de deux éléments tirés uniformément dans R_q.

Le chiffrement de BFV est construit comme suit :

— L'espace des clairs est R_t pour un certain entier $t > 1$.
— La clé secrète $sk = s$ avec $s \xleftarrow{\$} \mathcal{X}$.
— La clé publique $pk = ([-(as + e)]_q, a)$ avec $a \xleftarrow{\$} R_q$ et $e \xleftarrow{\$} \mathcal{X}$.

Pour chiffrer un message $m \in R_t$ en utilisant la clé publique $pk = (p_0, p_1)$, il faut tirer $u, e_1, e_2 \leftarrow \mathcal{X}$ puis calculer :

$$ct = ([\Delta m + p_0 u + e_1]_q, [p_1 u + e_2]_q) \in (R_q)^2 \tag{4.4}$$

où Δ est la partie entière inférieure de la division de q par t. Cette définition permet de voir que les chiffrés sont des tableaux de polynômes de R_q. Dans le cas d'un *fresh ciphertext*, autrement dit un chiffré n'ayant subi aucune opération, le tableau contient deux polynômes.

Le déchiffrement du chiffré $ct = (c_0, c_1)$ en utilisant la clé secrète $s = sk$ est donné par :

$$\left[\left\lfloor \frac{t}{q}[c_0 + c_1 s]_q \right\rceil\right]_t \tag{4.5}$$

où $\lfloor \cdot \rceil$ désigne la fonction partie entière inférieure. Plus précisément, regardons les diffé-

rentes étapes du déchiffrement :

$$\left[\left\lfloor \frac{t}{q}[c_0 + c_1 s]_q \right\rceil\right]_t = \left[\left\lfloor \frac{t}{q}[\Delta m - eu + e_1 + e_2 s]_q \right\rceil\right]_t \tag{4.6}$$

$$= \left[\left\lfloor \frac{t}{q}(\Delta m + v + qr) \right\rceil\right]_t \tag{4.7}$$

$$= \left[\left\lfloor m + \frac{t}{q}(v - \epsilon m) + tr \right\rceil\right]_t \tag{4.8}$$

$$= [m + tr]_t = m \tag{4.9}$$

où $\epsilon = \frac{q}{t} - \Delta$.

Ce chiffrement est *fully* homomorphe *i.e.* on note $\oplus$ l'opération d'addition et $\otimes$ celle de multiplication. Considérons deux messages m_1 et m_2 et leurs chiffrés $ct_1 = (c_0, c_1) = ([\Delta m_1 + p_0 u + e_1]_q, [p_1 u + e_2]_q)$ et $ct_2 = (c'_0, c'_1) = ([\Delta m_2 + p_0 u' + e'_1]_q, [p_1 u' + e'_2]_q)$. Le calcul de l'addition sécurisée est le suivant :

$$ct_1 \oplus ct_2 = (c_0 + c'_0, c_1 + c'_1) \tag{4.10}$$

$$= ([\Delta(m_1 + m_2) + p_0(u + u') + e_1 + e'_1]_q, \\ [p_1(u + u') + e_2 + e'_2]_q) \tag{4.11}$$

Le chiffré obtenu dans (4.11) est un chiffré valide de $m_1 + m_2$. La multiplication sécurisée s'obtient en calculant :

$$(\bar{c_0}, \bar{c_1}, \bar{c_2}) =$$

$$\left(\left[\left\lfloor \frac{t}{q} c_0 c'_0 \right\rceil\right]_q, \left[\left\lfloor \frac{t}{q}(c_0 c'_1 + c_1 c'_0) \right\rceil\right]_q, \left[\left\lfloor \frac{t}{q} c_1 c'_1 \right\rceil\right]_q\right) \tag{4.12}$$

Le problème associé à la multiplication sécurisée est l'augmentation du nombre de polynômes dans le chiffré. Il existe cependant une méthode de relinéarisation qui permet de réduire ce nombre de polynômes comme présenté dans [FV12]. Cette opération permet d'obtenir un chiffré composé de seulement deux polynômes à partir de (4.12).

Il est également possible de calculer la négation $-m$ d'une valeur chiffrée m. Pour cela, il suffit de changer le signe de chacun des polynômes du chiffré ct comme dans [Lai17].

Enfin, dans (4.7) le terme v est appelé le bruit. On constate que chaque opération chiffrée, que ce soit l'addition, la multiplication ou bien la relinéarisation, font augmenter

le bruit du chiffré. Ainsi, les chiffrés ne peuvent être correctement déchiffrés que si leur bruit ne dépasse pas un certain seuil.

Passons maintenant à la sécurisation de la décompression d'un flux de données compressées par Huffman.

4.3 Décompression sécurisée de mots de code de Huffman

L'objectif de cette section est de montrer comment décompresser de manière sécurisée un flux de données compressées par Huffman sur un Serveur qui n'est pas de confiance. Supposons qu'une fois décompressé, la taille des symboles obtenus est codée sur d bits.

Dans la suite de cette section, nous commençons par montrer comment effectuer cette opération sur des données en clair en utilisant des opérations compatibles avec un FHE. Puis, sur cette base, nous présentons la version sécurisée de cette méthode dans le domaine chiffré. Pour finir, nous montrerons une méthode permettant de moduler la longueur du flux binaire sans incidence sur la décompression et avec pour effet d'augmenter la sécurité de la solution. À noter que pour des questions de simplicité, les symboles que nous allons considérer sont codés sur $d = 8$ bits, ce qui correspond à prendre en compte la compression par Huffman des pixels d'une image en niveau de gris.

4.3.1 La décompression de Huffman

Notre méthode consiste à lire le flux binaire bit à bit. Prenons un arbre de Huffman $\mathcal{H}$ et un de ses mots de code c de longueur n :

$$c = c_0, \ldots, c_{n-1}. \tag{4.13}$$

Notre méthode consiste à placer le pointeur sur la racine de l'arbre $\mathcal{H}$. Par la suite, le premier bit du flux est lu. Si ce dernier est égal à 1, la branche de gauche est choisie. Sinon, c'est celle de droite. On continue de lire le flux binaire bit après bit jusqu'à terminer sur une feuille. Une fois arrivé sur une feuille, le symbole est décodé.

Cette méthode en clair pose certaines difficultés dans le domaine chiffré. Comme elle parcourt l'arbre, le chemin emprunté est mémorisé durant la décompression. Or, dans le domaine chiffré se rappeler du chemin n'est pas possible.

La solution pour pallier ce problème est d'utiliser l'algorithme de parcours en largeur présenté à la Section 4.1.2. Ce dernier permet d'attribuer de façon unique un index à chaque nœud i et feuille j de l'arbre.

Supposons que l'on dispose d'une mémoire B_i pour chaque nœud de l'arbre i tel que :

$$B_i = 1 \text{ si le pointeur est sur le nœud } i \quad \text{et} \quad B_i = 0 \text{ sinon.} \tag{4.14}$$

La procédure de décompression est la suivante :

- Le Serveur se place sur le nœud i pour lequel $B_i = 1$ ou sur la racine de l'arbre $\mathcal{H}$ si tous les $\{B_i\}_i = 0$.
- Le Serveur lit le prochain bit du flux.
- Si le bit est égal à 1 (resp. 0), il suit la branche de gauche (resp. droite) pour atteindre le nœud d'index i'.
- La mémoire du nœud i est réinitialisée à 0 : $B_i = 0$.
- Si le nœud atteint n'est pas une feuille, alors la mémoire du nœud i' passe à 1 : $B_{i'} = 1$. Cela permet au Serveur lors de la prochaine itération de ce processus de se placer sur le bon nœud.
- Si le nœud atteint est une feuille, le symbole est décodé.

Il est intéressant de noter qu'au plus un unique B_i vaut 1 et tous les autres sont nuls. Plus précisément, tous les B_i sont nuls si l'on se trouve sur la racine (au début de la décompression) et si l'on atteint une feuille (fin de la décompression). Or, comme nous le verrons dans la suite, finir de lire un mot de code impose de réinitialiser la mémoire sur la racine de $\mathcal{H}$. Le but de cette opération est de permettre la décompression des mots de code suivant. Ainsi, on peut dire que ces deux événements : être sur la racine et finir de lire un mot de code sont simultanés.

Voici un exemple de cette méthode. L'objectif est de décoder le mot de code 111 en utilisant l'arbre présenté sur la Figure 4.4. Initialisons les variables $\{B_i\}_{i=1,\dots,5} = 0$, puis l'algorithme est le suivant :

- Le Serveur sait qu'il doit se placer sur la racine car $\forall i \in [\![1, 5]\!]$, $B_i = 0$.
- Il lit le premier bit $c_0 = 1$ et atteint le nœud d'index 1.
- Ainsi, seule la mémoire B_1 passe à 1 : $B_1 \leftarrow 1$.
- Le Serveur sait qu'il doit se placer sur le nœud 1 car $B_1 = 1$.
- Il lit le second bit $c_1 = 1$ et atteint le nœud d'index 3.
- Les mémoires B_1 et B_3 sont mises à jour : $B_1 \leftarrow 0$ et $B_3 \leftarrow 1$.
- Le Serveur sait qu'il doit se placer sur le nœud 1 car $B_3 = 1$.

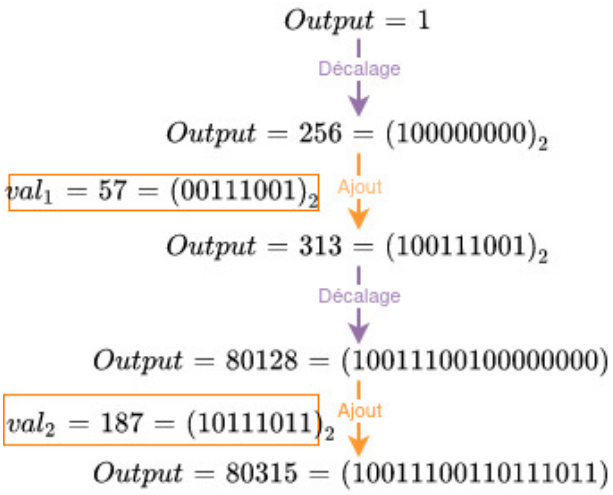


FIGURE 4.5 – Exemple fonctionnement *Output*.

— Il lit le troisième bit $c_1 = 1$ et atteint la feuille l_3.

— La mémoire de B_3 est réinitialisée : $B_3 \leftarrow 0$.

La valeur de la feuille l_3 a bien été atteinte. De plus à la fin de l'algorithme, tous les $\{B_i\}_{i=1,\dots,5}$ sont à 0 ainsi l'algorithme est réinitialisé.

Pour résumer, afin de décompresser des mots de code de Huffman, nous avons besoin de son arbre $\mathcal{H}$ ainsi que d'une mémoire B_i initialisée à 0 pour chacun de ses nœuds. Dans la suite, nous regarderons plus concrètement comment mettre en équation cette décompression.

4.3.2 Variables et opérations associées à la décompression

Dans un premier temps, supposons qu'une fois qu'un mot de code est décodé, sa valeur chiffrée est créée. Cette solution n'est pas sécurisée, car le Serveur connaît la taille du mot de code.

Reprenons le cas des images avec des pixels sur 8 bits $d = 8$ pour illustrer notre solution à ce problème. Notre idée est d'utiliser une variable *Output* initialisée à 1 au début de la décompression puis de décaler de $d = 8$ bits la valeur stockée dans *Output* et de venir inscrire la nouvelle valeur décodée à la fin de *Output*. La Figure 4.5 reprend une telle construction.

Dans notre solution, à chaque fois qu'un symbole est décodé, au lieu de renvoyer sa valeur, cette dernière est concaténée dans *Output*. Ainsi, quand *Output* sera chiffré, il sera impossible à l'attaquant de savoir quand et si une valeur lui a été concaténée.

Comme nous l'avons déjà précisé à la Section 4.1.2, il n'est pas possible d'évaluer de façon générale la condition : "si alors, sinon" dans le domaine chiffré FHE ; condition que nous avons besoin d'évaluer pour parcourir l'arbre. Nous devons nous restreindre à un cas dégénéré (4.2) :

$$Res = (1 + (-Bool)) \times a + Bool \times b$$

Cette équation devra être calculée pour chaque nœud et feuille de l'arbre de Huffman $\mathcal{H}$. Nous devrons alors pour chaque bit lu, combiner le résultat de ces opérations afin de savoir si oui ou non une feuille a été atteinte. Les seules valeurs non nulles qui seront calculées correspondront à l'équivalent de leur décompression dans le domaine clair.

De façon plus précise, dans cette section, nous allons voir comment :

— Créer une variable Val prenant la valeur $leaf_j$ contenue dans la $j^{\text{ième}}$ feuille de l'arbre si cette feuille est atteinte et 0 sinon.

— Créer une variable $Shift$ qui vaut 2^d à chaque fois qu'une feuille est atteinte et 1 sinon.

— Concaténer les valeurs des mots de code décodés dans la variable $Output$.

— Faire la mise à jour des variables booléennes tels que les mémoires B_i.

L'objectif final est d'utiliser la variable Val pour obtenir la décompression des mots de code de Huffman. La variable $Shift$ sert à décaler suffisamment les valeurs précédemment stockées dans $Output$ afin d'en stocker de nouvelles sans écrire sur les anciennes. $Output$ stocke les valeurs décodées en les concaténant les unes à la suite des autres. Enfin, la mise à jour des variables permet au programme de se situer correctement dans l'arbre de Huffman pour chaque nouveau bit lu. La procédure est présentée dans l'Algorithme 5 et les calculs sont précisés dans les sections suivantes.

Commençons par regarder comment calculer la valeur Val pour chaque bit c_k lu.

4.3.2.1 Extraction de la valeur contenue dans une feuille

Supposons, comme expliqué à la section 4.3.1, qu'à chaque nœud est associé une mémoire B_i. Cela nous permet de définir α comme :

$$\alpha = \sum_{i=1}^{nb_{nodes}} B_i \tag{4.15}$$

où nb_{nodes} correspond au nombre de nœuds de l'arbre. Ainsi, α est une valeur booléenne toujours égale à 1 sauf dans le cas où le pointeur se situe sur la racine. Auquel cas, α vaut

Algorithm 5 Procédure de décompression d'un flux $c = \{c_k\}_k$ composé de mot de code de Huffman.

Initialisation :
$Output = 1, \{b_i\}_i = 0, \alpha = 0$ et $\beta = 1$
for $c_k \in c$ **do**
    Réinitialisation :
    $Val = 0$ et $Shift = 1$
    Calculs :
    $Val+ = \cdots$ et $Shift+ = \cdots$ ▷ conformément aux eq. 4.18,4.19,4.20,4.21
    $Output = Output \times Shift + Val$
    Mises à jour :
    $\{b_i\}_i = \cdots$ ▷ conformément à la Table 4.1.
    $\alpha = \sum_{i=1}^{nb_{nodes}} B_i$ et $\beta = -\alpha + 1$
end for
return $Output$

0. De façon similaire, définissons β le complément de α :

$$\beta = -\alpha + 1. \tag{4.16}$$

β est aussi une valeur booléenne de telle sorte que :

$$\beta = 1 \text{ ssi } \alpha = 0 \quad \text{et} \quad \beta = 0 \text{ ssi } \alpha = 1 \tag{4.17}$$

Contrairement à α, β est toujours égale à 0 excepté dans le cas où le pointeur se situe sur la racine et où elle vaut 1.

Ces deux variables nous permettent de calculer Val. À chaque lecture d'un bit c_k, Val est initialisée à 0. Ensuite, pour chaque feuille j, on ajoute à Val la valeur stockée dans cette feuille suivant certaines conditions :

1. Supposons que le nœud parent de la feuille j soit la racine :
   — Si le bit à lire pour aller du nœud père à la feuille j est égal à 0 :

$$Val \mathrel{+}= \beta \times (-c_k + 1) \times leaf_j \tag{4.18}$$

   — Dans le cas contraire :

$$Val \mathrel{+}= \beta \times c_k \times leaf_j \tag{4.19}$$

2. Si le nœud père de la feuille j n'est pas la racine :

— si le bit à lire pour atteindre la feuille j est égal à 0 :

$$Val \mathrel{+}= B_i \times (-c_k + 1) \times leaf_j \tag{4.20}$$

— Dans le cas contraire :

$$Val \mathrel{+}= B_i \times c_k \times leaf_j \tag{4.21}$$

Par conséquent, (4.18,4.19) (resp. (4.20,4.21)) valent $leaf_j$ si et seulement si $\beta = 1$ (resp. $B_i = 1$) et $c_k = 0$ pour (4.18,4.20) (resp. $c_k = 1$ pour (4.19,4.21)). Toutefois, $\beta = 1$ (resp. $B_i = 1$) si et seulement si l'on se trouve sur la racine (resp. nœud i) et $c_k = 0$ (resp. $c_k = 1$) est ce qui est attendu dans le but d'atteindre la feuille j dans (4.18,4.20) (resp. (4.19,4.21)).

Comme ajouter plusieurs fois, 0 ne change pas la valeur de Val, les calculs présentés donnent à Val la valeur 0 si aucune feuille n'est atteinte et $leaf_j$ (parfois notée l_j) si la feuille j est atteinte.

Après avoir montré comment calculer Val, regardons comment calculer $Shift$ en utilisant une idée similaire.

4.3.2.2 Calcul du décalage

La méthode pour calculer $Shift$ est similaire à celle de Val. On initialise $Shift$ par 1 au lieu de 0 pour Val. Il suffit ensuite de remplacer Val par $Shift$ et $leaf_j$ par $2^d - 1$ dans (4.18,4.19,4.20,4.21). On peut alors tirer pour $Shift$ des conclusions similaires à celles de Val : $Shift$ vaut 1 si aucune feuille n'est atteinte et 2^d sinon.

Grâce à Val et $Shift$ revenons maintenant de façon plus formelle sur le calcul de $Output$.

4.3.2.3 Sauvegarde des données décodées

La variable $Output$, est initialisée à 1 au début du programme. Cependant, contrairement aux deux valeurs précédentes, elle n'est jamais réinitialisée. Pour calculer $Output$, reprenons notre exemple d'un flux binaire composé de mots de code de Huffman. Supposons que nous commençons par lire un mot $c = c_0, \ldots, c_{n-1}$. Pour c_0 jusqu'à c_{n-2} nous avons :

$$Val = 0 \quad \text{et} \quad Shift = 1 \tag{4.22}$$

	parent : la racine	parent : le nœud $i' \neq$ la racine
branche $c_k = 0$	$\beta \times (-c_k + 1)$	$B_{i'} \times (-c_k + 1)$
branche $c_k = 1$	$\beta \times c_k$	$B_{i'} \times c_k$

TABLE 4.1 – Mise à jour des B_i.

car nous n'avons pas encore atteint une feuille. Alors, pour chaque bit $\{c_k\}_{k=0,\dots,n-2}$ le calcul suivant :

$$Output = Output \times Shift + Val \tag{4.23}$$

ne change pas la valeur de $Output$.

Cependant, lorsque nous finissons de lire le mot de code *i.e.* nous lisons le bit c_{n-1}, nous atteignons la feuille j :

$$Val = leaf_j \quad \text{et} \quad Shift = 2^d \tag{4.24}$$

Le calcul (4.23) va décaler la valeur stockée dans $Output$ de d bits puis venir inscrire la valeur Val dans cet emplacement. Cela signifie que la valeur de $Output$ en binaire commence par un 1 suivi des d bits de $leaf_j$ qui représente la valeur décompressée du mot de code de Huffman. Ceci permet de stocker dans $Output$ les valeurs décompressées.

Regardons maintenant comment mettre à jour les variables booléennes B_i pour chaque nœud i.

4.3.2.4 Mise à jour des mémoires des valeurs booléennes

Au début du programme, toutes les mémoires B_i sont initialisées à 0 *i.e.*, le pointeur est sur la racine. Par la suite, les $\{B_i\}_i$ sont mises à jour après chaque bit c_k lu. Nous devons distinguer 4 cas différents comme vu Section 4.3.2.1. Chaque B_i est calculée conformément aux résultats présentés dans le Tableau 4.1. Ainsi, toutes les B_i valent 0 sauf au plus une qui vaut 1.

Une fois que les $\{B_i\}_i$ sont mises à jour, on peut passer à α et β. Leur mise à jour s'effectue en suivant leur définition (4.15,4.16). Tant qu'aucune feuille n'est atteinte $\alpha = 1$ et $\beta = 0$. Dans le cas contraire, $\alpha = 0$ et $\beta = 1$.

Les mises à jour des $\{B_i\}_i$, de α et de β se font après le calcul de $Val, Shift$ et $Output$. Ceci permet de réinitialiser tous les B_i à 0, α à 0 et β à 1 dans le cas où une feuille est atteinte *i.e.* le pointeur dans l'arbre de $\mathcal{H}$ se positionne sur la racine. Quant à elles, les valeurs Val et $Shift$ sont réinitialisées à chaque lecture d'un nouveau bit. $Output$ n'est

pas réinitialisé, il stocke toujours les valeurs qui ont déjà été concaténées en lui. Ceci nous permet alors d'itérer cette procédure sur tout un flux binaire.

Dans la section suivante, nous exprimerons dans le domaine chiffré l'ensemble des opérations présentées dans cette partie.

4.3.3 Passage aux données chiffrées

En passant dans le domaine chiffré, on suppose que le Serveur a accès à un flux binaire chiffré bit à bit : à chaque tour, le Serveur lit un bit $E[c_k]$ où $E[\cdot]$ est un chiffrement *fully* homomorphe comme celui présenté à la Section 4.2.1. Le Serveur crée un ensemble de données chiffrées : $E[\alpha] = E[0], E[\beta] = E[1]$ et l'ensemble de mémoire binaire $\{E[B_i]\}_i$ toutes initialisées à $E[0]$.

Pour les autres variables Val, $Shift$ et $Output$, nous nous sommes efforcés de les calculer avec des opérations évaluables dans le domaine chiffré FHE. Ainsi, il est possible de reformuler (4.18,4.19,4.20,4.21) dans le domaine FHE. Pour commencer, après chaque lecture d'un nouveau bit chiffré $E[c_k]$, $E[Val]$ est initialisé à $E[0]$ puis pour chaque feuille en fonction de leur nœud parent :

$$E[Val] \oplus= E[\beta] \otimes (-E[c_k] \oplus E[1]) \otimes E[leaf_j] \tag{4.25}$$

$$E[Val] \oplus= E[\beta] \otimes E[c_k] \otimes E[leaf_j] \tag{4.26}$$

$$E[Val] \oplus= E[B_i] \otimes (-E[c_k] \oplus E[1]) \otimes E[leaf_j] \tag{4.27}$$

$$E[Val] \oplus= E[B_i] \otimes E[c_k] \otimes E[leaf_j] \tag{4.28}$$

Le calcul de $E[Val]$ est sécurisé et a les mêmes propriétés que Val. Plus clairement, $E[Val] = E[leaf_j]$ si et seulement si la feuille j est atteinte, sinon $E[Val] = E[0]$.

Il en va de même pour $E[Shift]$, les opérations proposées à la section 4.3.2.2 sont évaluables dans le domaine chiffré FHE. Il suffit d'initialiser $E[Shift]$ à $E[1]$ pour chaque bit chiffré lu. Ensuite, on remplace dans (4.25,4.26,4.27,4.28) $E[Val]$ par $E[Shift]$ et $E[leaf_j]$ par $E[2^d - 1]$. De nouveau, les conclusions valables pour $Shift$ le sont pour $E[Shift]$.

Pour calculer $E[Output]$, on utilise les valeurs de $E[Val]$ et $E[Shift]$:

$$E[Output] = E[Output] \otimes E[Shift] \oplus E[Val]. \tag{4.29}$$

Tout comme $Output$, $E[Output]$ est initialisé une unique fois à $E[1]$ au début du pro-

gramme. $E[Shift]$ sert à libérer la place nécessaire dans $E[Output]$ pour venir écrire $E[Val]$.

La mise à jour des mémoires chiffrées est identique à celle proposée Tableau 4.1 à ceci prés que les valeurs en clair sont remplacées par leurs valeurs chiffrées :

$$c_k \leftarrow E[c_k] \quad \beta \leftarrow E[\beta] \quad B_{i'} \leftarrow E[B_{i'}] \quad \text{et} \quad 1 \leftarrow E[1] \tag{4.30}$$

Les $\{E[B_i]\}_i$ sont ainsi mises à jour après chaque lecture de bit chiffré $E[c_k]$.

Pour finir, les deux valeurs booléennes chiffrées $E[\alpha]$ et $E[\beta]$ sont respectivement initialisées à $E[0]$ et $E[1]$. Par la suite, les formules :

$$E[\alpha] = \oplus_{i=1}^{nb_{nodes}} E[B_i] \tag{4.31}$$

$$E[\beta] = -E[\alpha] \oplus E[1] \tag{4.32}$$

permettent leur mise à jour dans le domaine chiffré. L'ensemble de la procédure est résumé dans l'Algorithme 6.

Dans la prochaine partie, nous reviendrons rapidement sur le scénario global avec l'utilisation et la mise à jour des variables chiffrées afin de décompresser un flux de données compressées Huffman.

Algorithm 6 Procédure de décompression sécurisée d'un flux chiffré bits à bits ($E[c] = \{E[c_k]\}_k$) composé de mot de code de Huffman.

Initialisation :
$E[Output] = E[1], \{E[b_i]\}_i = E[0], E[\alpha] = E[0]$ et $E[\beta] = E[1]$
for $E[c_k] \in E[c]$ **do**
    Réinitialisation :
    $E[Val] = E[0]$ et $E[Shift] = E[1]$
    Calculs :
    $E[Val] \oplus = \cdots$ et $E[Shift] \oplus = \cdots$ ▷ conformément aux eq. 4.25,4.26,4.27,4.28
    $E[Output] = E[Output] \otimes E[Shift] \oplus E[Val]$
    Mises à jour :
    $\{E[b_i]\}_i = \cdots$ ▷ conformément à l'eq. 4.30
    $E[\alpha] = \oplus_{i=1}^{nb_{nodes}} E[B_i]$ et $E[\beta] = -E[\alpha] \oplus E[1]$
end for
return $E[Output]$

4.3.4 Scénario global

Jusqu'ici nous avons montré comment décompresser un flux binaire composé de mot de code de Huffman. Ce type de flux peut poser un problème de sécurité. En connaissant la longueur du flux, il est possible de retrouver de l'information sur la longueur des mots de code utilisés. Prenons un exemple comprenant un flux binaire composé de 2 bits et l'arbre de Huffman présenté sur la Figure 4.4. Le Serveur saura alors que le mot de code ne peut être que l_1 ou l_2. Pour résoudre ce problème, nous proposons une solution qui consiste à ajouter à la fin du flux binaire des informations "inutiles" qui ne seront pas décodées, mais qui permettent d'allonger le flux. Comme la longueur du flux n'est plus indexée sur les symboles qui le composent, il n'est pas possible d'inférer sa composition en connaissant sa longueur. Pour effectuer cette modification, il faut qu'un mot X de l'arbre de Huffman $\mathcal{H}$ ne soit pas attribué. Le flux est alors complété par X en entier ou simplement partiellement afin d'en augmenter sa taille. Le mot de code n'étant pas attribué, cela n'a pas de conséquence sur la décompression. Cette opération est notée $X_{padding}$.

Cette technique permet à partir d'un flux binaire composé de N mots de code $\{c^l\}_{l=1,\dots,N}$ de le découper en plusieurs flux binaires de taille inférieure. Cela permet d'avoir plus de données à traiter, mais de longueur plus faible :

$$\begin{aligned} E[C^1] &= E[c^1||\cdots||c^m||X_{padding}] \\ E[C^2] &= E[c^{m+1}||\cdots||c^n||X_{padding}] \\ &\vdots \end{aligned} \tag{4.33}$$

$$\tag{4.34}$$

Cette technique a évidemment un impact négatif sur la compression car elle ajoute des bits à la fin des flux binaires.

Pour décompresser un flux binaire $E[C^l]$, il suffit d'initialiser $E[Output]$ à $E[1]$ et tous les $\{E[B_i]\}_i$ à $E[0]$ puis pour chaque bit $E[C^l_k]$ lu :

— Calculer $E[Val]$ et $E[Shift]$ conformément aux équations de la Section 4.3.3.

— Calculer $E[Output]$ comme présenté (4.29).

— Faire la mise à jour des $\{E[B_i]\}_i$ suivie de celles de $E[\alpha]$ et $E[\beta]$.

et ce jusqu'à avoir complètement lu le flux binaire. Ainsi, l'écriture binaire de $E[Output]$ commencera par un 1 suivi de $d = 8$ bits représentant la valeur décodée du premier mot de

code du flux. Si tous les bits de $E[Output]$ n'ont pas été lus, les 8 prochains représentent la valeur décodée du second mot de code du flux et ainsi de suite. Le mot de code X n'ayant pas été attribué, il ne vient pas modifier les valeurs stockées dans $E[Output]$.

Le fait de découper un flux en d'autres plus petits permet de rendre les calculs praticables avec des FHE actuels. En effet, plus le nombre d'opérations chiffrées augmente plus ces dernières deviennent lentes. D'autre part, l'espace attribué au texte en clair dans les FHE est fixe comme présenté à la Section 4.2.1. Or, une fois décompressées, les données sont généralement plus grandes. Ainsi, si cette taille dépasse la taille du clair, le FHE utilisé ne fonctionne pas. Le découpage permet de pallier ce problème en s'assurant lors de la compression et de la création des flux binaires que la taille des données décompressées n'est pas supérieure à celle du texte clair dans le FHE utilisé.

Maintenant que nous avons vu le fonctionnement de notre protocole, passons à la partie expérimentale.

4.4 Résultats expérimentaux

Ces expérimentations sont basées sur le cryptosystème BFV présenté dans la section 4.2.1. Nous utilisons une implémentation de ce cryptosystème faite par une équipe de Microsoft dans la librairie SEAL [Mic20]. Pour commencer, regardons les données que nous décompressons.

4.4.1 Les données utilisées

Notre expérimentation est basée sur des images appartenant à MNIST. Une présentation de cette base de données a été faite à la Section 3.3.1. Les pixels sont codés sur $d = 8$ bits. Dans un premier temps, nous avons calculé la table d'occurrence des pixels pour toutes les images de MNIST, ce qui nous a permis de créer l'arbre de Huffman associé. Durant la création de cet arbre, nous avons ajouté un symbole X non attribué avec une unique occurrence ce qui permet de lui attribuer un mot de code long, ceci pénalise peu le codage de Huffman. Dans notre cas, le mot de code associé est $(00010001100)_2$.

Les capacités de calcul étant limitées, nous avons effectué plusieurs tests avec des longueurs de flux allant de 11 à 22 bits. Nous nous sommes assurés que chaque flux comportait entre 2 et 5 pixels compressés.

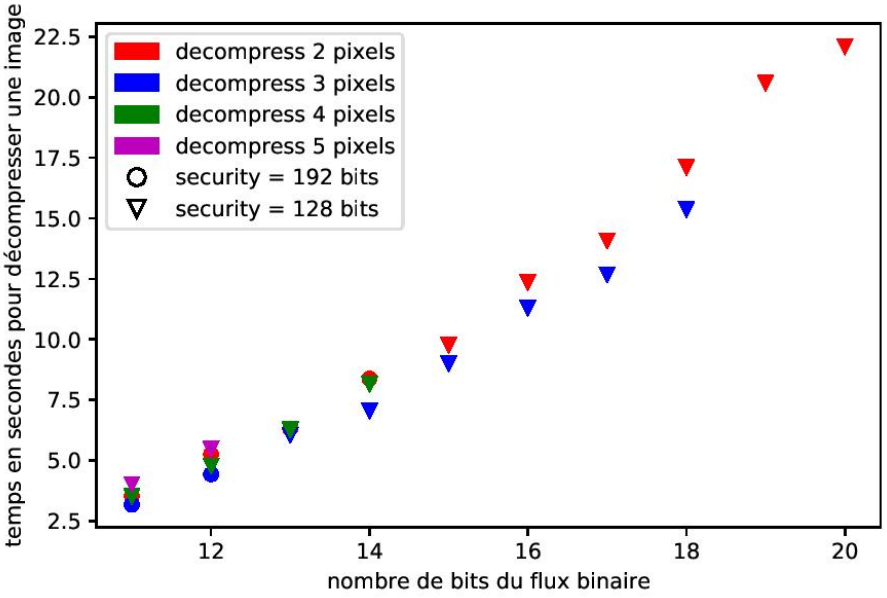


FIGURE 4.6 – Temps nécessaire pour décompresser une image MNIST (28×28 pixels) en fonction de la taille du flux binaire et du nombre de pixels compressés.

4.4.2 Les capacités de calcul utilisées

L'ordinateur utilisé pour effectuer ces calculs est équipé d'un Ryzen 7 2700X et de 32 Go de RAM.

Les résultats des expérimentations sont présentés Figure 4.6. Ainsi, pour décompresser une image décomposée en flux binaire de 18 bits qui compressent 3 pixels il faut environ 15 secondes. Notre solution permet de décompresser des images de 28×28 pixels compressées par Huffman en quelques secondes. En comparaison, les résultats obtenus dans [Ma+18] pour effectuer la décompression d'un bloc 8×8 compressé JPEG prend entre 1h30 et 2h.

Après avoir vu les performances expérimentales de notre solution, revenons sur sa sécurité.

4.5 Sécurité

Les objectifs de sécurité sont les suivants, le Serveur ne doit pas :

— connaître les mots de code de Huffman, ni les valeurs décodées,

— savoir où dans l'arbre de Huffman $\mathcal{H}$ le pointeur se trouve,

— en connaissant la longueur du flux, pouvoir trouver d'information sur les mots de

code qui le compose.

Si le Serveur connaît le nombre de bits à décoder cela constitue un problème de sécurité. Un exemple d'une telle attaque a été donné à la Section 4.3.4. Ainsi, la taille du flux binaire doit pouvoir être modifiée afin de ne donner aucune information au Serveur. Pour cela, nous utilisons le $X_{padding}$.

Par conséquent, une fois que le Serveur décode le dernier mot du flux binaire ($X_{padding}$ exclu), il commence à décoder le $X_{padding}$. Cependant, comme aucune valeur n'est associée à X, la valeur de $E[Val]$ reste toujours égale à $E[0]$ et celle de $E[Shift]$ à $E[1]$. Ainsi, le calcul de $E[Output]$ (4.29) ne change jamais la valeur stockée dans ce dernier. Ainsi, la décompression de $X_{padding}$ peut être vue comme un calcul fictif qui ne sert qu'à augmenter la taille du flux binaire pour des raisons de sécurité.

Plus généralement, toutes les données sont chiffrées par un FHE qui assure leur confidentialité. Discutons maintenant des résultats obtenus.

4.6 Discussion

L'implémentation de cette méthode ne minimise pas la longueur de la plus grande séquence de multiplication. En effet, en regardant l'Algorithme 6, on constate que pour chaque lecture d'un bit c_k, $E[Output]$ est multiplié par $E[Shift]$. En développant les calculs, cela revient à dire qu'après k bits lus, la valeur de $E[Output]$ est au moins composée de k multiplications. Or, plus le nombre de multiplications successives est élevé, plus le bruit et les temps de calcul augmentent. Réduire la profondeur de multiplication augmenterait les performances.

Tout au long de ce travail, nous avons proposé une solution permettant de sécuriser la décompression de Huffman. Cependant, nous n'avons pas parlé du problème de la transmission des données. En effet, nous supposons que le Serveur connaît le flux binaire chiffré bit à bit. Le problème étant que la taille d'une donnée chiffrée par un FHE est plus grande que la taille de la donnée en clair (expansion de chiffré). Ainsi, envoyer directement les bits du flux binaire chiffré a un coût de communication élevé. Il existe plusieurs solutions à ce problème. Une première consiste à utiliser une entité de confiance qui génère un nombre aléatoire r qu'elle envoie en clair au client et en chiffré au Serveur. Le client l'ajoute aux données transmises au Serveur, qui peut alors retrouver les données originales dans le domaine chiffré en faisant une soustraction homomorphe. Une autre solution consiste à chiffrer les données du côté client avec un autre cryptosystème (sans expansion de chiffré)

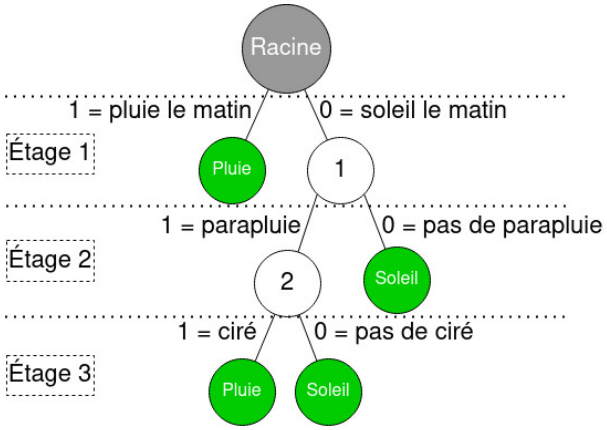


FIGURE 4.7 – Exemple d'un arbre de décision.

dont le déchiffrement est évaluable dans le domaine chiffré. Certaines solutions utilisant l'AES ou le CLCG par exemple [GHS12 ; PBC19] permettent de retrouver les données originales dans le domaine FHE sans jamais y accéder en clair.

Au-delà, les outils développés dans cette première partie de chapitre restent génériques et peuvent être appliqués à d'autres problèmes. C'est pourquoi dans la prochaine section, nous en proposons une extension.

4.7 Extension de la méthode aux arbres de décision binaires et au RLE

Dans cette section, nous montrons comment sécuriser un arbre de décision binaire (DT) ainsi que l'algorithme RLE (*run-length encoding*) présent dans JPEG.

4.7.1 Les arbres de décision binaires

En ce qui concerne les arbres de décision binaires, la différence principale avec Huffman est que le flux binaire est composé de mot de même taille. Pour mieux comprendre, prenons l'exemple d'un arbre de décision cherchant à savoir s'il va pleuvoir l'après-midi :

— S'il pleut durant la matinée alors il pleuvra l'après-midi.
— Sinon, si les passants ont un parapluie ainsi qu'un ciré alors il pleuvra l'après-midi.
— Dans tous les autres cas, il fera beau.

Cet exemple est résumé dans la Figure 4.7. Ainsi le Serveur souhaitant évaluer la sortie de cet arbre recevra toujours en entrée 3 bits. Le premier bit renseigne la météo du matin,

le second et le troisième la présence ou l'absence de parapluie ou de ciré respectivement. La réception de $(1, 0, 0)$ doit mener à conclure qu'il va pleuvoir durant l'après-midi. Il est important de voir que ce cas diffère de Huffman. En effet, pour Huffman si une feuille est atteinte cela signifie que le mot de code est fini d'être lu. Dans le cas des DT, ce n'est pas forcément le cas. Par exemple, dans la Figure 4.7 la feuille "Pluie" fille de la racine est atteinte après avoir seulement lu 1 bit. Or, le symbole d'entrée est toujours de taille 3 bits.

Une première solution pour pallier ce problème consiste à compléter l'arbre jusqu'à avoir des feuilles qui sont toutes au même niveau. Plus précisément, dans l'exemple présenté, cela revient à remplacer la feuille "Soleil" qui est la fille du nœud 1 par un nouveau nœud lui-même père de deux feuilles "Soleil". Il faudrait également compléter de façon similaire la branche qui part de la racine avec un bit égal à 1. Bien que fonctionnelle, cette solution semble coûteuse dans le cas d'arbres de grande profondeur. C'est pour cela que nous proposons une autre méthode.

L'idée est d'utiliser une méthode similaire à celle de Huffman. Ici, nous calculons normalement $E[Val], E[Shift]$ ainsi que $E[Output]$, mais nous n'aurons cependant pas besoin de α et β qui servent uniquement à savoir si l'on se trouve ou non sur la racine. Or, dans le cas des DT, la réinitialisation est automatique, car les mots dans le flux binaire sont de taille constante n (dans notre exemple la taille est de 3). Ainsi, il nous suffit de réinitialiser nos variables à la racine après n lectures de bits en utilisant un compteur k. Ce compteur permet de seulement mettre à jour les variables de l'étage k de l'arbre. Ainsi, dans notre exemple, si nous lisons le premier bit c_1, les seules solutions possibles sont d'arriver sur la feuille "Pluie" ou le nœud 1 ce qui donne :

$$E[Val] = E[c_1] \otimes E[Pluie],\ E[Shift] = E[1] \oplus E[c_1] \otimes E[2^d - 1] \text{ et } E[B_1] = E[1] - E[c_1] \tag{4.35}$$

Seules les variables de l'étage 1 sont mises à jour. Le calcul de $E[Output]$ reste inchangé pour tous les étages. Nous lisons le prochain bit *i.e.* c_2 et mettons seulement à jour les variables du second étage :

$$\begin{aligned} E[Val] &= (E[1] - E[c_2]) \otimes E[Soleil] \otimes E[B_1], \\ E[Shift] &= E[1] \oplus (E[1] - E[c_2]) \otimes E[2^d - 1] \otimes E[B_1] \text{ et} \\ E[B_2] &= E[c_2] \otimes E[B_1] \end{aligned} \tag{4.36}$$

Il ne reste plus qu'à faire les calculs de l'étage 3 :

$$E[Val] = (E[c_3] \otimes E[Pluie] \oplus (E[1] - E[c_3]) \otimes E[Soleil]) \otimes E[B_2],\ E[Shift] = E[1] \oplus E[2^d - 1] \otimes E[B_2]$$

Finalement, comme $n = 3$ bits ont été lus, il suffit de réitérer le programme. Cette méthode est plus efficace que celle présentée pour Huffman car le Serveur a accès à plus d'informations. En effet, il sait à quel étage de l'arbre, il se trouve ce qui permet d'optimiser les calculs.

Cette solution peut être utilisée pour évaluer d'autres DT binaires. Comparée aux solutions présentées en introduction de ce chapitre, la notre permet de mettre en entrée d'un DT un flux contenant un nombre de données à classer inconnu pour le Serveur ce qui augmente la sécurité. Regardons maintenant comment effectuer la décompression sécurisée du RLE.

4.7.2 Le Run-Length Encoding

Le RLE (*Run-length encoding*) est un algorithme de compression qui consiste à remplacer plusieurs occurrences consécutives d'un même symbole par son nombre d'occurrences suivi du symbole. Par exemple : "aaabbbb" serait compressé en "3a4b".

Ici, nous nous intéressons plus précisément au codage RLE utilisé pour les valeurs AC dans l'algorithme JPEG. À la différence du RLE précédent, ce codage RLE produit deux valeurs (run/len). La valeur run code le nombre de 0 placés avant une valeur non nulle qui suit ce mot de code et qui est codée sur len bits.

Plus concrètement, supposons que le code de Huffman 00110 code le symbole RLE (3,4). Prenons un flux binaire s valant 001101101. Alors, pour décoder ce flux on remarque que $s = 001101101$ où la partie en rouge code le RLE (3,4). Cela signifie que le décodage de s donne $d = (0, 0, 0, -5)$ où les trois 0 viennent du run du RLE. Le -5 vient du fait que le len du RLE précise que les 4 bits suivant ce mot de code forment une valeur en binaire signée *i.e.* $(1101)_2 = -5$.

L'idée est de créer une variable $E[Track]$ qui "traque" si on lit le mot de code de Huffman correspondant au RLE ou si on lit les bits qui suivent. Cette valeur permet la décompression du mot de code associé au RLE. D'autre part, nous allons montrer comment décompresser la valeur binaire signée qui le suit.

En ce qui concerne la décompression du RLE, nous utilisons la méthode présentée dans ce chapitre avec deux changements :

— $E[Val]$ n'est plus utilisé.
— Le calcul de $E[Shift]$ se fait comme celui de $E[Val]$ à ceci près que les $leaf_j$ sont remplacées par $2^{run_j \times d} - 1$ où run_j correspond au run du RLE associé à la feuille j.

Nous supposons également avoir $E[Track]$ qui vaut $E[0]$ lors de la décompression du RLE et $E[1]$ lors de la décompression des bits qui suivent le mot de code.

Pour s'assurer que lors de la lecture des bits qui suivent le mot de code, les variables qui permettent de décoder le RLE ne modifient pas la sortie $E[Output]$, il faut toutes les mettre à zéro. La solution consiste à toutes les multiplier (exceptée $E[Output]$) par $E[1]-E[Track]$. Ainsi, lorsque l'on lit les bits du mot de code RLE, cette multiplication ne change rien. Par contre, lorsque nous lisons les bits qui suivent ce mot de code, toutes les variables deviennent nulles et ne changent pas la valeur de $E[Output]$. Passons maintenant à la décompression des bits qui suivent le mot de code.

On définit le jeu de variables $\{E[Len_i]\}_{1 \leq i \leq m} = 0$ qui sert à stocker la valeur len du RLE où m est la valeur du plus grand des len. En utilisant une méthode similaire à celle de $E[Val]$, on attribue à $E[Len_j]$ la valeur $E[1]$ quand la feuille j est atteinte, et $E[0]$ sinon. Tout comme les $\{B_i\}_i$, au plus un unique $\{E[Len_i]\}_i$ vaut $E[1]$ et les autres sont nuls. Pour mettre à jour les $\{E[Len_i]\}_i$, il suffit de calculer :

$$\text{Si } i \in [\![1, m-1]\!] \text{ alors } E[Len_i] = E[Len_{i+1}] \text{ sinon } E[len_m] = E[0] \tag{4.37}$$

Ces variables $\{E[Len_i]\}_{1 \leq i \leq m}$ permettent de créer $E[Track]$:

$$E[Track] = \sum_{i=1}^{m} E[Len_i] \tag{4.38}$$

Ainsi créée, $E[Track]$ vérifie bien les propriétés précédemment énoncées.

Comme la valeur suivant le mot de code est en binaire signé, il est important de se rappeler son signe tout au long de sa lecture. Pour cela, on crée $E[Trigger]$ qui passe de 0 à 1 uniquement quand on finit de lire le mot de code associé au RLE. Le calcule de $E[Trigger]$ se fait de façon analogue à $E[Val]$ en remplaçant $leaf_j$ par $E[1]$.

On peut alors créer $E[\nu]$ qui vaut 1 durant toute la lecture des bits qui suivent le mot de code si le bit de signe est à 1 et 0 sinon :

$$E[\nu] = E[c_k]E[Trigger] \oplus E[\nu]E[Track] \tag{4.39}$$

On introduit $E[Shift']$ qui libère de la place dans $E[Output]$ afin d'écrire les bits de la valeur qui suit le mot de code. Afin que toutes les valeurs décodées fassent la même taille, un facteur d'expansion est ajouté :

$$E[Shift'] = E[Track](E[2] \oplus E[Trigger](\sum_i E[Len_i](E[2^{m-i}] - E[2]))) \tag{4.40}$$

On crée également $E[Val']$ qui prend en compte ce facteur d'expansion ainsi que le bit de signe :

$$\begin{aligned} E[Val'] =& E[Track](E[\nu]E[c_k] \oplus (-E[\nu] \oplus E[1])(-E[c_k] \oplus E[1]) \oplus \\ & E[Trigger](-E[c_k] \oplus E[1]) \sum_i E[Len_i]E[2^{n-i}]) \end{aligned} \tag{4.41}$$

Finalement, toutes ces valeurs nous permettent de calculer :

$$E[Output] = E[Output](E[Shift] \oplus E[Shift']) \oplus E[Val']. \tag{4.42}$$

On note que : soit $E[Shift]$ est non nul auquel cas cela signifie que nous sommes en cours de lecture du mot de code de Huffman, soit $E[Shift']$ et $E[Val']$ sont non-nuls, ce qui signifie que les bits suivant le mot de code sont en cours de lecture.

Ce programme permet de décompresser de façon sécurisée le RLE présenté. Passons maintenant à la conclusion de ce chapitre.

4.8 Conclusion

Dans ce chapitre, nous sommes revenus sur le codage de Huffman qui permet de compresser des données. Nous avons rappelé que ce codage fait partie d'algorithmes de compression plus élaborés tels que JPEG ou MP3. L'objectif a alors été de faire la décompression sécurisée d'un flux binaire composé de mot de code de Huffman. Pour ce faire, nous avons commencé par expliquer que notre solution est itérative sur les bits du flux binaire. Elle permet de décoder les mots de code de Huffman. Pour cela, nous avons introduit différentes variables nécessaires à la décompression ainsi que les opérations de mise à jour qui leur sont associées. Ces variables permettent d'évaluer de façon sécurisée l'arbre de Huffman $\mathcal{H}$. Le Serveur effectuant cette tâche n'a alors aucune information sur les données qu'il décompresse, leur nombre ou la position du pointeur dans l'arbre.

Toutes les opérations présentées ont été établies de sorte qu'elles soient évaluables dans le domaine chiffré FHE. Cela nous a permis de retranscrire la méthode de décompression dans le domaine chiffré afin de la sécuriser. Cette procédure a été décrite à l'aide d'un algorithme.

Finalement, nous sommes revenus sur des problèmes de puissance de calcul et de sécurité vis-à-vis de la connaissance de la longueur du flux binaire. Une solution consistant à ajouter à la fin du flux binaire un mot de code non assigné a été proposée. Cette solution diminue les performances de la compression, mais améliore la sécurité.

Les expérimentations confirment que la méthode permet de décompresser des images MNIST de 28×28 pixels codées par Huffman en quelques secondes en fonction de la taille du flux binaire choisi. Alors que, [Ma+18] met entre 1h30 et 2h pour décoder un bloc 8×8 JPEG. De plus, la solution que nous avons proposée présente plusieurs outils utilisables dans d'autres contextes. Nous avons ainsi proposé deux extensions : une permettant de sécuriser des DT binaires et une autre permettant de décompresser le RLE des coefficients AC de JPEG. Il reste à traiter les différents tags JPEG ainsi que de l'en-tête, puis de continuer la décompression en suivant les étapes (réorganisation Zig-Zag, déquantification, ...). Finalement, il est important de noter que cette solution reste générale et bénéficierait positivement d'un algorithme FHE plus performant. Un article reprenant ces travaux est en cours de rédaction.

Cette solution s'inscrit dans la continuité du chapitre précédent où nous avons montré qu'il est possible de faire l'apprentissage ainsi que la classification sur données partiellement décompressées. Dans cette section, nous avons montré comment décoder un codage de Huffman qui est la dernière étape de l'algorithme de compression JPEG. Il serait alors intéressant d'appliquer cette méthode en prenant en compte le codage RLE des coefficients DC et AC afin de présenter une solution complète permettant d'externaliser des données compressées et sécurisées sur un Serveur avant d'entreprendre leur décompression dans le but d'entrainer un réseau de *machine learning* le tout de façon sécurisée.

Dans le chapitre suivant, nous reviendrons sur différents outils de sécurisation et de traitements présentés tout au long de ce manuscrit afin de les allier. Ceci nous permettra, entre autre, de présenter comment, en utilisant un cryptosystème additivement homomorphe couplé avec du calcul multipartite, il est possible de sécuriser la classification ainsi que l'apprentissage pour un réseau de neurones artificiels.

Chapitre 5

Chaîne de traitement sécurisée de données compressées

Dans les précédents chapitres, nous avons présenté plusieurs solutions permettant d'utiliser de façon conjointe le chiffrement, la compression ainsi que les traitements de données, plus clairement :

- Au Chapitre 2, nous avons sécurisé une opération de filtrage suivi d'un seuillage sur des données concaténées.
- Au Chapitre 3, nous avons montré qu'il était possible de faire l'apprentissage d'un réseau de *machine learning* sur des données partiellement décompressées.
- Au Chapitre 4, nous avons effectué la décompression d'un flux binaire composé de mots de code de Huffman de façon sécurisée.

Dans ce chapitre, nous allons reprendre certaines de ces techniques afin de présenter une solution complète de traitements sécurisés de données compressées et chiffrées : la sécurisation de réseau de *machine learning* sur données compressées.

Ce chapitre s'articule de la manière suivante. En première partie, nous allons présenter une méthode permettant d'évaluer de manière sécurisée l'apprentissage et la classification de données concaténées sécurisées par un cryptosystème additivement homomorphe grâce à des calculs multipartites (*i.e.* faisant intervenir plusieurs parties). Dans la seconde partie, nous reviendrons sur les résultats présentés aux Chapitres 3 et 4 afin de proposer une solution complète fondée sur les FHE.

5.1 Classification et apprentissage sécurisé par un PHE de données concaténées

Notre objectif ici est d'envisager une entité (une personne/un patient, une entreprise/un hôpital, ...) qui possède des données et qui souhaiterait concevoir un réseau de

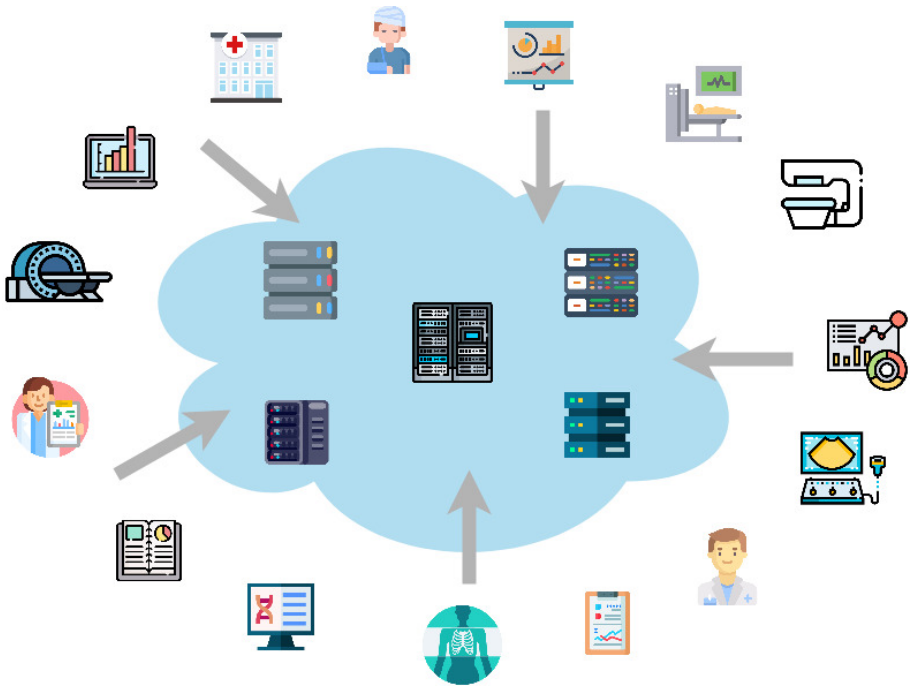


FIGURE 5.1 – Scénario composé de l'entité détentrice de données (EDD) symbolisée par des patients, médecins, hôpitaux... qui envoient leurs données numériques vers le *cloud* lui-même constitué de plusieurs compagnies, serveurs.

machine learning permettant de les classifier. Cette entité sera plus couramment appelée Entité Détentrice de Donnée notée EDD. En vis-à-vis, nous avons un *cloud* comprenant plusieurs serveurs appartenant à une seule ou plusieurs compagnies. Ce *cloud* possède une puissance de calcul et de stockage importante (plus importante qu'un particulier ou qu'une institution) et pourrait ainsi proposer des prix, des temps de traitement, une externalisation et des garanties de traitement sécurisé attrayantes pour l'EDD. La Figure 5.1 présente un schéma concis de ce scénario que nous allons suivre tout au long de cette section.

Dans un premier temps, nous faisons l'hypothèse que les données sont sécurisées à l'aide d'un algorithme de chiffrement puis transmises jusqu'au *cloud*. Ensuite, afin de pouvoir faire un apprentissage sécurisé de ces données, il est indispensable que le type de chiffrement choisi permette certaines opérations sur les chiffrés sans pour autant en révéler leur contenu. Comme nous l'avons vu précédemment, la solution la moins coûteuse actuellement, que ce soit en termes de temps ou en coût de calcul, est d'adopter un chiffrement additivement homomorphe. L'exemple d'un tel chiffrement est présenté Section 2.1.1.2. Ainsi, si nous revenons à notre scénario, nous avons deux possibilités :

1. Les données sont directement sécurisées avec un algorithme de chiffrement additif avant d'être transmises. Cette solution présente l'avantage d'être simple cependant elle implique un coût de communication important.
2. Il est possible de mettre en place une solution similaire à celle présentée dans la Section 2.1.3.2. Cette dernière consiste à chiffrer les données à l'aide d'un chiffrement léger sans expansion de chiffré avant de les transférer. Une fois disponible sur le *cloud*, les données peuvent être converties en données chiffrées homomorphiquement grâce à l'algorithme CrC. Cette technique peut être couplée à la concaténation de données présentée à la Section 2.1.4.

Une fois les données disponibles sur le *cloud* et sécurisées par un algorithme de chiffrement qui permet leur traitement, le *cloud* peut commencer la phase d'entraînement du réseau. Cette phase a pour but de faire l'apprentissage des poids et biais d'un réseau. Cependant, le *cloud* ne doit en aucun cas connaître les données d'entrées/sorties et les paramètres du réseau dans le contexte où il est considéré comme une entité honnête mais curieuse.

La structure du réseau : nombre de couches, fonction d'activation, fonction de coût, ... doit avoir été discutée au préalable entre les deux entités : celle possédant les données (l'EDD) et celle possédant la puissance de calcul (le *cloud*). On peut également imaginer un système d'itération à partir duquel le *cloud* fait l'apprentissage des données et envoie les poids appris à l'EDD. Cette dernière pourrait ainsi juger rapidement si la précision du modèle est suffisante ou non pour son application et le cas échéant demander au *cloud* de modifier la structure du réseau (ajout de couches, diminution du nombre de neurones, ...). Le *cloud* aurait pour tâche de refaire l'apprentissage jusqu'à l'obtention d'une précision du réseau suffisante aux yeux de l'EDD.

Dans un deuxième temps, et ce, indépendamment ou non de la phase d'apprentissage, l'EDD pourrait tout simplement chercher à faire classer ses données par le *cloud*. Dans le cas où le *cloud* a au préalable procédé à l'apprentissage d'un tel réseau, ce dernier posséderait déjà les poids et biais chiffrés. Dans les autres cas, il incombera à l'EDD de transmettre au *cloud* de façon sécurisée les poids dont elle aura elle-même fait l'apprentissage ou délégué ce dernier à un autre ensemble de serveurs. Cette transmission peut une fois de plus faire appel à la solution présentée à la Section 2.1.3.2. Ainsi, le *cloud* pourrait retourner à l'EDD le résultat sécurisé de la classification de ses données.

En plus de ces différents objectifs et contraintes, nous avons fait les opérations d'apprentissage et de classification sur des données concaténées comme vu Section 2.1.4. Le

but est ainsi de fortement diminuer les coûts de communication ainsi que les temps de calcul tout en garantissant la sécurité des données.

Maintenant précisons plus en détail le rôle et la composition des entités en présence.

5.1.1 Présentation du scénario d'application et des entités impliquées

De façon plus précise, l'EDD est l'entité qui crée, recueille, synthétise, ... les données. Par exemple, un centre hospitalier acquérant des images médicales (*e.g.* IRM, scanners), un médecin prescrivant un traitement à un patient ou même une entreprise numérique recueillant des données utilisateurs. Pour des raisons de coût, d'efficacité et de réutilisation des données comme présenté Section 1.1.5, les données sont envoyées sur le *cloud*. Cet envoi peut être sécurisé par un algorithme de chiffrement additivement homomorphe ou tout autre type de chiffrement permettant par la suite de les convertir de façon sécurisée vers un chiffrement permettant leur traitement.

L'entité *cloud* quant à elle regroupe plusieurs serveurs. Comme nous le verrons en détail lors de l'exécution du protocole sécurisé, cette multitude de serveurs est indispensable afin de garantir que dans le cas où les serveurs sont des entités honnêtes mais curieuses, la sécurité globale des données et paramètres du *machine learning* est conservée. Plus précisément, afin de sécuriser des opérations élémentaires, mais essentielles pour un *machine learning* telles que la multiplication, la comparaison et le maximum entre deux valeurs en utilisant un chiffrement additivement homomorphe et des données concaténées, nous aurons besoin de deux, trois voire quatre serveurs distincts. De plus, ce nombre de serveurs peut être flexible en fonction des paramètres que l'on accepte de leur octroyer, mais également du nombre de communications nécessaires à l'achèvement du protocole ou encore de son temps d'exécution. Il devient clair que les différents serveurs du *cloud* peuvent ou non être détenus par une ou plusieurs compagnies. Il est même envisageable de changer de fournisseur de serveur au cours d'un apprentissage, d'une classification (au risque d'augmenter les coûts de calculs et de communication) voire même en fonction des opérations à sécuriser *e.g.* une compagnie A pour le calcul sécurisé du maximum et une compagnie B pour celui de la multiplication.

Nous avons discuté de l'acquisition et de la transmission sécurisée des données jusqu'au *cloud* où elles se trouvent disponibles sous forme chiffrée par un algorithme additivement homomorphe. Rappelons que la concaténation des données permet d'optimiser les temps

ainsi que les coûts de calculs.

5.1.2 Sur données concaténées sécurisées

Pour rappel, un chiffrement additivement homomorphe tel que Paillier [Pai99] ou Damgård–Jurik [DJ03] offre une sécurité suffisante à condition que leurs paramètres de clé secrète p et q soient générés sur un grand nombre de bits par exemple 2048 (c.f. Section 2.1.1.2). Cela implique que la taille d'un chiffré est grande. Ainsi, comme vu en détail dans la Section 2.1.4, il est possible d'organiser des données en un vecteur avant de les chiffrer afin de réduire les coûts de communications et de calcul. Cette opération permet de traiter en une fois plusieurs données sécurisées avec quelques contraintes supplémentaires (*e.g.* traitement de données sous forme matricielle, impossibilité de multiplier deux valeurs concaténées par un scalaire différent).

Dans ce travail, nous reprenons l'idée présentée à la Section 2.1.4 afin de rendre notre solution plus efficiente. Nous noterons tout de même que les contraintes qui pèsent sur nos calculs sécurisés sont délétères pour certains d'entre eux. Par exemple, il n'est pas possible de faire la somme de toutes les valeurs concaténées dans un même vecteur, car il n'est pas possible de les extraire. Or, ce type de calcul est indispensable afin d'obtenir la classification de données ou bien pour l'apprentissage d'un réseau de *machine learning*. C'est à cet égard que le nombre de serveurs impliqués au niveau du *cloud* doit être supérieur à celui de la solution *Followknee*.

Dans ce qui suit nous allons regarder comment il est possible d'effectuer l'apprentissage ou la classification de données concaténées au travers d'un réseau de *machine learning*, sans cependant les sécuriser avec un cryptosystème PHE.

Un premier point important à considérer est que les poids ou biais d'un réseau de *machine learning* sont généralement de petites données réelles (typiquement entre -1 et 1) c.f. Section 3.1.3 alors que les données traitées par un chiffrement additivement homomorphe sont des entiers c.f. Section 2.1.1.2. Une solution simple présentée dans [Bel+19] consiste à quantifier (*i.e.* multiplier) les poids et biais par un facteur noté δ et de garder la partie entière de cette quantification. Cette opération induit une perte de précision proportionnelle à δ. Cependant, comme nous le verrons dans la partie expérimentale, quand bien même cette approximation affecte la classification et l'apprentissage d'un réseau, elle n'est pas rédhibitoire.

Une fois les paramètres (poids et biais) du réseau quantifiés, ces derniers se retrouvent codés sur δ bits. Dans le cas où δ est inférieur à la taille du clair d'un chiffrement additi-

vement homomorphe, on peut utiliser la technique de concaténation.

Toutefois, il faut faire attention au débordement entre les valeurs d'un même vecteur, problème présenté dans la Section 2.1.4.1. En effet, entre autres opérations utiles à un réseau de *machine learning*, il y a la multiplication de deux valeurs. Or, en supposant qu'elles soient codées sur δ bits le résultat de leur multiplication sera codé sur au maximum 2δ bits. Il est primordial que l'espace de stockage de chaque valeur dans le vecteur concaténé soit au minimum plus grand que 2δ. Il faut également que chaque multiplication soit suivie d'une division par 2^{δ}. Cela permet d'assurer que le résultat de la multiplication suivi d'une division est codé sur au maximum δ bits. En revanche, cette opération de division induit également une perte de précision. En effet, les données chiffrées par un chiffrement additivement homomorphe sont des entiers par conséquent le résultat de la division est arrondi.

Dans le but de calculer correctement le *zero padding* à effectuer pour la classification, il faut prendre en compte les additions que fait le réseau. En particulier, on rappelle que la sortie d'un neurone présentée Section 3.1.1 est calculée comme suit :

$$y = \sum_i x_i \times w_i + b \tag{5.1}$$

où les $\{x_i\}_i$ sont les entrées, les $\{w_i\}_i$ les poids et b le biais.

Il en est de même en ce qui concerne la convolution 2D d'un CNN, voir Section 3.1.2, dont la sortie est donnée par le calcul suivant :

$$s_{I,J,K} = (\sum_{i=1}^{H_f} \sum_{j=1}^{W_f} \sum_{k=1}^{C} e_{SI+i-1,SJ+j-1,k} \times f_{i,j,K}) + b_K \tag{5.2}$$

De plus, dans les modèles que nous allons sécuriser, nous utiliserons la fonction d'activation ReLU (*Rectified Linear Unit*) qui est très utilisée ([UD17], [Cho+15], [Gér19]) et qui est telle que :

$$ReLU(x) = \begin{cases} x & \text{si } x > 0 \\ 0 & \text{sinon.} \end{cases} \tag{5.3}$$

or, cette fonction n'a pas d'incidence sur le nombre de bits maximal d'une sortie.

Enfin, si l'architecture du réseau de *machine learning* comporte plusieurs couches, l'augmentation de la taille de la sortie d'une couche se reporte sur l'entrée de la couche suivante. Une fois toutes ces considérations prises en compte, une valeur de *padding* peut être trouvée afin de garantir qu'aucun débordement ne surviendra pour la classification.

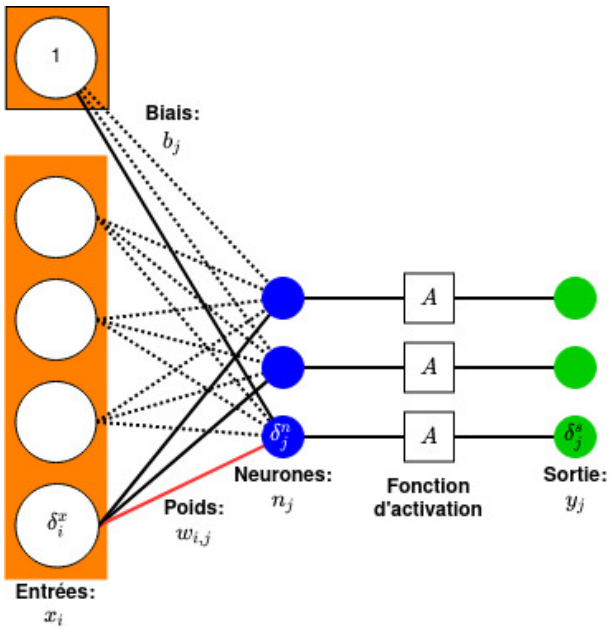


FIGURE 5.2 – Rétro propagation du gradient sur une couche de neurone.

La *backpropagation* ou rétro propagation du gradient nécessite d'effectuer des multiplications, suivies de divisions par le pas de quantification δ. Comme la fonction d'activation est ReLU, le calcul de sa dérivée durant la mise à jour des gradients n'augmente pas la taille du clair :

$$ReLU'(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{sinon.} \end{cases} \tag{5.4}$$

En prenant l'exemple d'une couche de neurone comme présenté Figure 5.2, le calcul du gradient pour le poids $w_{i,j}$ est donné par :

$$\begin{aligned} \frac{\delta E}{\delta w_{i,j}} &= \frac{\delta E}{\delta y_j} \frac{\delta y_j}{\delta n_j} \frac{\delta n_j}{\delta w_{i,j}} \\ &= \delta_j^s \times \delta_j^n \times x_i \end{aligned} \tag{5.5}$$

où E est l'erreur et δ_j^s est la rétro propagation de cette erreur jusqu'à la $j^{\text{ième}}$ sortie de l'activation de cette couche. δ_j^n vaut quant à lui :

$$\delta_j^n = Comp(\sum_i x_i w_{i,j} + b_j, 0) \tag{5.6}$$

où $Comp(\cdot,\cdot)$ est l'opération de comparaison. On a ainsi deux cas de figure :

$$\frac{\delta E}{\delta w_{i,j}} = \begin{cases} \delta_j^s \times x_i & \text{si } \sum_i x_i w_{i,j} + b_j > 0 \\ 0 & \text{sinon.} \end{cases} \tag{5.7}$$

Par conséquent, cette opération de mise à jour des poids ne demande qu'une division par le facteur de quantification δ.

Cependant, comme l'erreur est rétro propagée de couche en couche, le gradient de l'erreur calculé au niveau des entrées x_i de l'exemple de la Figure 5.2 est une somme :

$$\frac{\delta E}{\delta x_i} = \frac{\delta E}{\delta y_j}\frac{\delta y_j}{\delta n_j}\frac{\delta n_j}{\delta x_i} \tag{5.8}$$

$$= \sum_j \delta_j^s \times \delta_j^n \times w_{i,j} \tag{5.9}$$

Ainsi, en plus d'une division par δ, il faut aussi prendre en compte l'expansion de chiffré induite par cette somme.

Une fois que toutes ces contraintes ont été prises en compte, la concaténation des données peut être effectuée. Regardons maintenant comment les réseaux de *machine learning* fonctionnent avec des données concaténées et quelles sont les contraintes induites quand on passe en données chiffrées.

5.1.2.1 *Machine learning* sur données concaténées en clair

Il est désormais possible de faire la classification de toutes les données concaténées dans un même vecteur en une seule et unique fois. Cette technique permet ainsi d'optimiser l'espace du clair et ainsi de rendre les calculs ainsi que les communications plus efficientes lors de calcul sur données chiffrées. Nous pourrons vérifier ceci dans la partie expérimentale où nous comparerons nos résultats avec ceux de [Bel+19] qui n'utilisent pas la concaténation de données.

La *backpropagation* bénéficie également de cette amélioration. Quantifier par δ après chaque multiplication permet aux *padding* de la classification et de la *backpropagation* d'être les mêmes (*i.e.* d'avoir la même taille). Cela optimise la phase d'apprentissage qui comprend une série de classifications suivie de *backpropagation*. Étant donné que les deux vecteurs employés lors de la classification et de la *backpropagation* ont un *padding* similaire, ils contiennent le même nombre de données. Ainsi, il n'est pas utile de faire un nombre de classifications différent du nombre de *backpropagation* lors de l'apprentissage.

Or, comme rappelé Section 3.1.3, il est possible d'utiliser la technique de *mini-batch* lors de l'apprentissage. Ce procédé consiste à faire l'apprentissage puis la *backpropagation* d'un nombre prédéterminé de données avant de faire la mise à jour des poids et biais en faisant la somme des gradients d'erreurs. Puisque les calculs de la *backpropagation* sont faits sur données concaténées, il suffit de calculer la somme des gradients d'erreur associés aux différents poids et biais des différentes couches afin d'effectuer un *mini-batch*.

Nous avons montré que les calculs sur données concaténées étaient particulièrement indiquées dans le cadre d'apprentissage utilisant le *mini-batch* et la classification d'un réseau de *machine learning*. Regardons maintenant comment concilier ces deux techniques et leurs contraintes.

5.1.2.2 Sur données concaténées chiffrées

Comme présenté Section 2.1.4.1, l'association du chiffrement additivement homomorphe et de la concaténation de données impose des contraintes sur les opérations que l'on peut effectuer sur un vecteur concaténé chiffré. Il n'est ainsi plus possible une fois passé dans le domaine chiffré de comparer, multiplier, trouver le maximum de deux chiffrés. Il n'est pas non plus possible de faire la somme des données concaténées d'un même vecteur. Or, ces opérations sont effectuées lors du calcul de la classification ou de la *backpropagation*.

Une solution consiste à faire intervenir des serveurs supplémentaires. Ainsi, il devient possible de distribuer ces opérations entre serveurs tout en assurant leur sécurité dans le cas où les serveurs sont honnêtes mais curieux.

Dans toute la suite de cette section, nous supposons qu'un premier serveur noté P_1 possède $E[A]$ et $E[B]$, les chiffrés des deux vecteurs de données concaténées A et B avec :

$$A = \sum_i \delta^i a_i \tag{5.10}$$

$$B = \sum_i \delta^i b_i \tag{5.11}$$

où $E[\cdot]$ représente le chiffrement additivement homomorphe utilisé, δ le pas de quantification, les $\{a_i\}_i$ et $\{b_i\}_i$ les données que P_1 cherche à comparer, multiplier, ... entre elles.

L'opération de comparaison sécurisée avec 2 serveurs :

Dans un premier temps, montrons comment calculer de façon sécurisée la comparai-

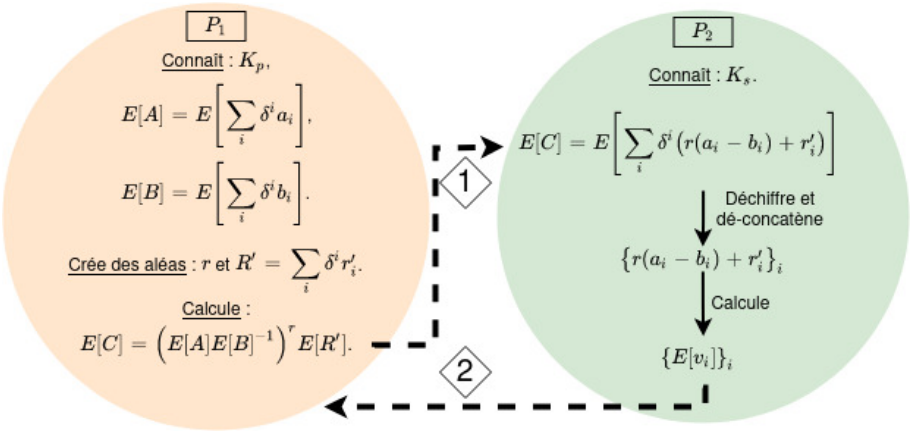


FIGURE 5.3 – Schéma de comparaison sécurisée entre $E[A]$ et $E[B]$ impliquant 2 serveurs : $Comp_e^{P_1,P_2}(E[A], E[B])$.

son entre deux vecteurs de données concaténées chiffrés par un chiffrement additivement homomorphe.

Ainsi, P_1 cherche à comparer les $\{a_i\}_i$ aux $\{b_i\}_i$, *i.e.* P_1 cherche l'ensemble $\{v_i\}_i$ tel que :

$$E[v_i] = E[Comp(a_i, b_i)] \tag{5.12}$$

L'opération de comparaison sécurisée entre deux serveurs P_1 et P_2 notée $Comp_e^{P_1,P_2}(E[A], E[B])$ est présentée Figure 5.3.

Dans un premier temps, le serveur P_1 crée des nombres aléatoires r et $\{r'_i\}_i$ avec $r'_i < r$ pour tout i. Comme vu dans la Section 2.1.1.1, ces aléas servent à masquer les données de la manière suivante :

$$c = rx + r' \tag{5.13}$$

où x est la donnée à masquer, r et r' sont deux valeurs aléatoires telles que $r' < r$.

Ainsi, une personne connaissant c peut en déduire le signe de x mais pas sa valeur :

$$c = \begin{cases} \text{positif} & \text{si } x > 0 \\ \text{négatif} & \text{sinon.} \end{cases} \tag{5.14}$$

Comme P_1 connaît les chiffrés additivement homomorphes $E[A]$ et $E[B]$, il peut calculer :

$$E[C] = (E[A]E[B]^{-1})^r E[R'] \tag{5.15}$$

grâce aux propriétés homomorphes du chiffrement. La donnée C vaut alors $C = \sum_i c_i$ avec

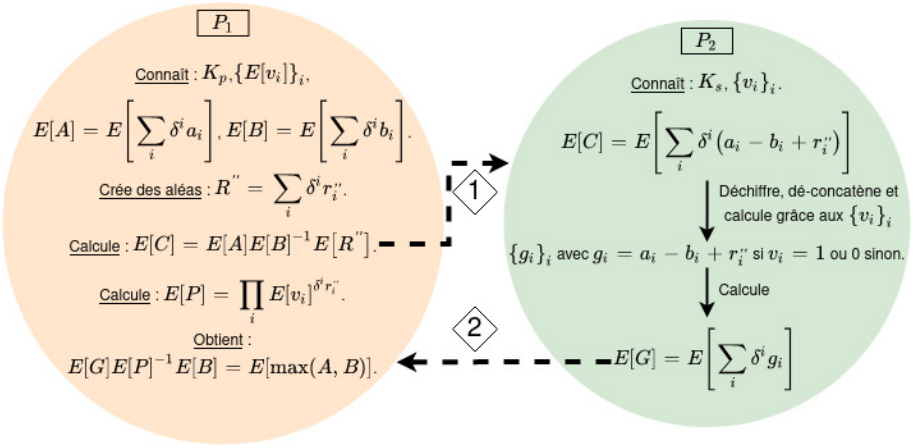


FIGURE 5.4 – Schéma du calcul du maximum sécurisé entre $E[A]$ et $E[B]$ impliquant 2 serveurs : $Max_e^{P_1,P_2}(E[A], E[B])$.

$c_i = \delta^i(r(a_i - b_i) + r'_i)$. P_1 peut transmettre $E[C]$ à P_2.

P_2 étant en possession de la clé secrète du schéma de chiffrement K_s, il peut procéder au déchiffrement de $E[C]$. Ensuite, il peut dé-concaténer C en l'ensemble des valeurs $\{c_i\}_i$. Il est alors possible pour P_2 de connaître les signes des $\{a_i - b_i\}_i$ notés $\{v_i\}_i$. Par convention, nous affectons à v_i la valeur 1 si le signe est positif et 0 sinon. Pour autant, du fait du masquage des données, P_2 ne peut pas inférer d'informations autres que le signe de leur différence sur les données $\{a_i\}_i$ ou $\{b_i\}_i$. P_2 procède au chiffrement des $\{v_i\}_i$ et transmet à P_1 l'ensemble $\{E[v_i]\}_i$.

Le protocole est terminé du fait que P_1 est maintenant en possession des signes chiffrés des différences $\{a_i - b_i\}_i$. Autrement dit, si $v_i = 1$ alors $a_i > b_i$ et $a_i < b_i$ si $v_i = 0$.

Cette opération de comparaison sécurisée est à la base de nombreuses autres opérations sécurisées plus complexes que nous allons présenter par la suite telle que l'opération qui renvoie le maximum entre deux valeurs.

L'opération maximum sécurisée :

Cette opération peut être réalisée en utilisant deux ou trois serveurs. Nous verrons en conclusion les différences entre ces deux cas.

- **L'opération maximum sécurisée avec 2 serveurs :**

P_1 souhaite connaître le résultat chiffré du maximum entre A et B, *i.e.* il cherche à connaître les résultats chiffrés des maximums entre les a_i et les b_i. Toute cette procédure est résumée Figure 5.4.

On suppose qu'au préalable P_1 et P_2 ont effectué le protocole de comparaison sécurisé

entre A et B *i.e.* P_2 connaît les $\{v_i\}_i$ et P_1 les $\{E[v_i]\}_i$.

Dans un premier temps, P_1 crée un ensemble de valeurs aléatoires $\{r''_i\}_i$ qu'il concatène dans un vecteur $R'' = \sum_i \delta^i r''_i$. Il peut alors, grâce aux propriétés d'homomorphie du chiffrement, calculer et envoyer à P_2 :

$$E[C] = (E[A]E[B]^{-1})E[R''] \tag{5.16}$$

À l'instar de ce qui a été présenté pour la comparaison sécurisée, P_2 peut déchiffrer et dé-concaténer ces données. Comme P_2 connaît les v_i, il obtient l'ensemble $\{g_i\}_i$ avec :

$$g_i = \begin{cases} a_i - b_i + r''_i & \text{si } v_i = 1 \\ 0 & \text{sinon.} \end{cases} \tag{5.17}$$

P_2 effectue la concaténation et le chiffrement des $\{g_i\}_i$ dans le vecteur chiffré $E[G] = E[\sum_i \delta^i g_i]$ qu'il transmet à P_1.

De son côté, P_1 connaît l'ensemble des $\{E[v_i]\}_i$, et calcule :

$$E[P] = \prod_i E[v_i]^{\delta^i r''_i} \tag{5.18}$$

d'où $P = \sum_i \delta^i p_i$ avec :

$$p_i = \begin{cases} r''_i & \text{si } v_i = 1 \\ 0 & \text{sinon.} \end{cases} \tag{5.19}$$

Finalement P_1 calcule le produit :

$$E[G]E[P]^{-1}E[B] = E[max(A, B)] \tag{5.20}$$

où $max(\cdot, \cdot)$ est la fonction qui renvoie le vecteur concaténé des maximums entre les valeurs contenues dans A et B car $E[G]$ et $E[P]$ contiennent respectivement les valeurs $a_i - b_i + r''_i$ et r''_i dans le cas où $a_i > b_i$ et 0 sinon (c.f. 5.17, 5.19). En utilisant la propriété d'homomorphie du chiffrement, on obtient que $E[G - P]$ vaut $a_i - b_i$ ou 0 en fonction du signe de $a_i - b_i$. Enfin, l'ajout de $E[B]$ permet d'obtenir le résultat souhaité, *i.e.* a_i dans le cas où $a_i > b_i$ et b_i sinon.

Au cours de cette opération ni P_1 ni P_2 ne peuvent obtenir d'information concernant a_i et b_i, hormis le signe de leur différence pour P_2. Or, dans notre contexte, le signe de cette différence n'est pas considéré comme une information à sécuriser. Ainsi, tant que

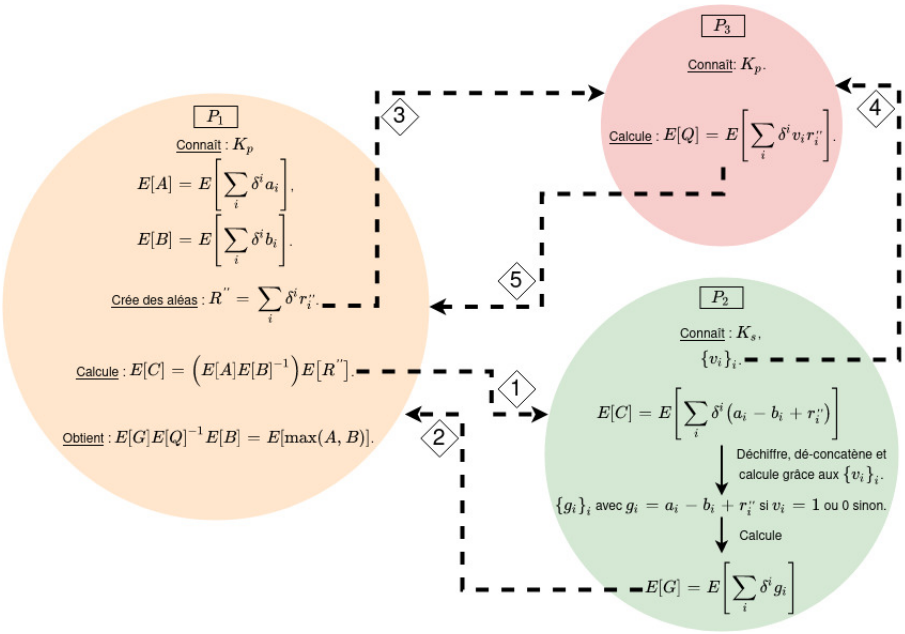


FIGURE 5.5 – Schéma du calcul du maximum comparaison sécurisée entre $E[A]$ et $E[B]$ impliquant 3 serveurs : $Max_e^{P_1,P_2,P_3}(E[A], E[B])$.

P_1 et P_2 restent des entités honnêtes mais curieuses, la confidentialité des données est garantie.

- **L'opération maximum sécurisée avec 3 serveurs :**

Cette fois-ci, regardons comment faire un maximum sécurisé entre deux valeurs chiffrées, mais en utilisant 3 serveurs : P_1, P_2 et P_3. Le but en ajoutant ce troisième serveur est que le coût des communications ne soit plus fonction du nombre de données concaténées dans un vecteur. La Figure 5.5 reprend cette procédure.

Le serveur P_2 a un rôle sensiblement similaire au cas avec 2 serveurs : le calcul des $\{g_i\}_i$. P_3 doit fournir à P_1 un chiffré permettant d'enlever le masquage des $\{g_i\}_i$. À la différence du précédent protocole, on suppose que seul P_2 connaît les signes de la différence des $\{a_i - b_i\}_i$ (*i.e.* les $\{v_i\}_i$), ceci permet un gain en termes de communication, car P_2 n'a plus besoin d'envoyer à P_1 les chiffrés de toutes ces valeurs.

On retrouve les mêmes étapes :

— P_1 crée les r_i'' qu'il concatène afin de calculer $E[C] = E[A]E[B]^{-1}E[R'']$.

— P_2 déchiffre et dé-concatène $E[C]$.

— P_2 peut grâce aux $\{v_i\}$ calculer $E[G]$ qu'il transmet à P_1.

Maintenant, dans ce protocole avec 3 serveurs, P_1 transmet $\{r_i''\}_i$ à P_3. P_2, quant à lui envoie les $\{v_i\}_i$ à P_3. Le troisième serveur P_3 peut alors calculer $E[Q]$ grâce à la clé

	[Bel+19]	2 serveurs	3 serveurs
$P_1 \to P_2$	2nλ	2λ	2λ
$P_1 \to P_3$			β
$P_2 \to P_1$	nλ	(n+1)λ	λ
$P_2 \to P_3$			γ
$P_3 \to P_1$			λ
Total	3nλ	(n+3)λ	4λ+β+γ

TABLE 5.1 – Comparaison des coûts de communications pour la fonction maximum sécurisée.

publique du cryptosystème K_p :

$$E[Q] = E[\sum_i \delta^i v_i r_i'']$$ (5.21)

Or :

$$v_i r_i'' = \begin{cases} r_i'' & \text{si } v_i = 1 \\ 0 & \text{sinon.} \end{cases}$$ (5.22)

P_3 communique alors $E[Q]$ à P_1.

Pour finir, P_1 calcule $E[G]E[Q]^{-1}E[B] = E[max(A, B)]$. En effet, $E[Q]$ est égale au $E[P]$ du protocole précédent (c.f. 5.22).

Ce protocole de maximum sécurisé avec 3 serveurs a un coût de communication bien moindre que celui avec 2 serveurs. De plus, ce coût ne dépend plus du nombre de données concaténées comme nous allons le résumer dans la section suivante.

- **Les coûts théoriques :**

On présente ici les coûts théoriques de communication des 3 méthodes du calcul de maximum :

1. Ceux présentés dans le papier [Bel+19] qui ne tient pas compte de la concaténation de données.
2. Ceux des 2 serveurs sur données concaténées.
3. Ceux des 3 serveurs sur données concaténées.

On note les coûts d'envoi suivant : λ pour celui d'un chiffré, β pour celui de l'ensemble des $\{r_i''\}_i$ et γ celui des $\{v_i\}_i$. On estime également que le nombre de données concaténées dans un vecteur est égal à n.

La Table 5.1 résume le coût de communication entre les différents serveurs identifiés par $P_i \to P_j$ pour l'ensemble des communications du serveur i vers le serveur j.

	[Bel+19]	2 serveurs	3 serveurs
P_1	n(3**C** + 3**E**)	2**C** + (n+1)**E**	2**C** + **E**
P_2	n(**C** + 3**D** + **E**)	n**C** + 2**D** + **E**	**C** + 2**D**
P_3			**C**
Total	n(4**C** + 3**D** + 4**E**)	(n+2)**C** + 2**D** + (n+2)**E**	4**C** + 2**D** + **E**

TABLE 5.2 – Comparaison des coûts de calcul pour la fonction maximum sécurisée.

Dans le cas de 2 serveurs, notre solution est plus performante que celle du papier [Bel+19] puisqu'elle bénéficie de la concaténation de données. Pour s'en convaincre, on peut estimer concaténer 40 données par vecteur (ce qui est une valeur réaliste comme nous le verrons par la suite). Par conséquent, le coût de communication total serait de 120λ pour [Bel+19] contre seulement 43λ pour notre solution à deux serveurs. Plus généralement, notre solution est presque 3 fois plus performante en termes de communication que celle présentée dans [Bel+19]. Pour plus de détail, on peut surtout observer que ce gain s'effectue au niveau de la communication $P_1 \to P_2$ alors que la communication inverse (*i.e.* $P_2 \to P_1$) est très légèrement plus coûteuse.

En ce qui concerne la solution avec 3 serveurs, elle surpasse en termes de coût de communication les deux autres du fait que le nombre n de données concaténées n'intervient pas dans son calcul. Plus précisément, n est un paramètre caché derrière β et γ mais, par leur construction, on peut majorer le coût de communication total pour 3 serveurs par 6λ. Dans le cas de 40 données concaténées, cela améliore le résultat avec 2 serveurs d'un facteur 7 et celui de [Bel+19] d'un facteur 20.

En ce qui concerne les coûts de calculs, et afin de simplifier les écritures, nous nous contentons d'évaluer les coûts principaux :

— **C** le nombre de chiffrements,

— **D** le nombre de déchiffrements,

— **E** le nombre d'exponentiations modulaires.

Par exemple, on ne comptera pas la génération et la concaténation des $\{r'_i\}_i$ nécessaires à la création de R'.

Comme montré dans la Table 5.2, la solution avec 2 serveurs est environ deux fois plus performante que celle de [Bel+19] au niveau du nombre de chiffrements et d'exponentiations modulaires et est plus performante d'un facteur n en ce qui concerne le nombre de déchiffrements.

La solution avec 3 serveurs, ne faisant pas intervenir n dans ses coûts de calcul, est quant à elle presque n fois plus performante que celle avec 2 serveurs en ce qui concerne le

nombre de chiffrements et d'exponentiations modulaires. Ainsi, dans le cas d'une concaténation de 40 données, la solution avec 3 serveurs est environ 40 fois plus performante que celle présentée par [Bel+19].

Il faut tout de même modérer cette amélioration de performance par le fait que toutes les opérations ne sont pas comptées (*e.g.* génération de nombre aléatoire, multiplications). Les opérations de maximum sécurisé avec 2 et 3 serveurs nécessitent de concaténer et déconcaténer les données ce que la solution de [Bel+19] ne demande pas. Bien que ces opérations soient relativement peu coûteuses leur impact sur les temps de calculs restent réels.

L'opération de multiplication sécurisée :

Tout comme l'opération maximum, cette opération de multiplication sécurisée entre des données chiffrées concaténées peut se faire de deux manières différentes avec 3 ou 4 serveurs. Nous verrons dans la partie résultats les avantages et inconvénients des deux méthodes.

- **L'opération de multiplication sécurisée avec 3 serveurs :**

L'idée derrière cette méthode est de déléguer à P_2 le calcul des produits des données masquées $\{a_i\}_i$ et $\{b_i\}_i$. Le serveur P_3 doit fournir à P_1 de quoi retirer le bruit induit suite au calcul des produits faits par P_2. Ces opérations sont illustrées Figure 5.6.

Pour commencer, P_1 crée deux vecteurs aléatoires concaténés R_A et R_B tel que $R_A = \sum_i \delta^i r_i^a$ et $R_B = \sum_i \delta^i r_i^b$ et les envoie à P_3. Ensuite, P_1 calcule et transmet à P_2 : $E[A]E[R_A]$ et $E[B]E[R_B]$.

À son tour, P_2 déchiffre et dé-concatène les données reçues avant d'en faire le produit :

$$E[P_2] = E[\sum_i \delta^i (a_i + r_i^a)(b_i + r_i^b)] \tag{5.23}$$

P_2 transmet $\{a_i r_i^a\}_i$ ainsi que $\{b_i r_i^b\}_i$ à P_3 et $E[P_2]$ à P_1.

P_3 peut, par conséquent, calculer :

$$\{(E[a_i + r_i^a]E[r_i^a]^{-1})^{r_i^b}\}_i = \{E[a_i r_i^b]\}_i \tag{5.24}$$

Il procède de façon analogue avec $E[b_i + r_i^b]$ afin d'obtenir $\{E[b_i r_i^a]\}_i$. Il génère également $\{E[r_i^a r_i^b]\}i$. Enfin, en multipliant et en concaténant ces trois chiffrés ($\{E[a_i r_i^b]\}_i$, $\{E[b_i r_i^a]\}_i$ et $\{E[r_i^a r_i^b]\}i$), il obtient :

$$E[P_3] = E[\sum_i \delta^i (a_i r_i^b + b_i r_i^a + r_i^a r_i^b)] \tag{5.25}$$

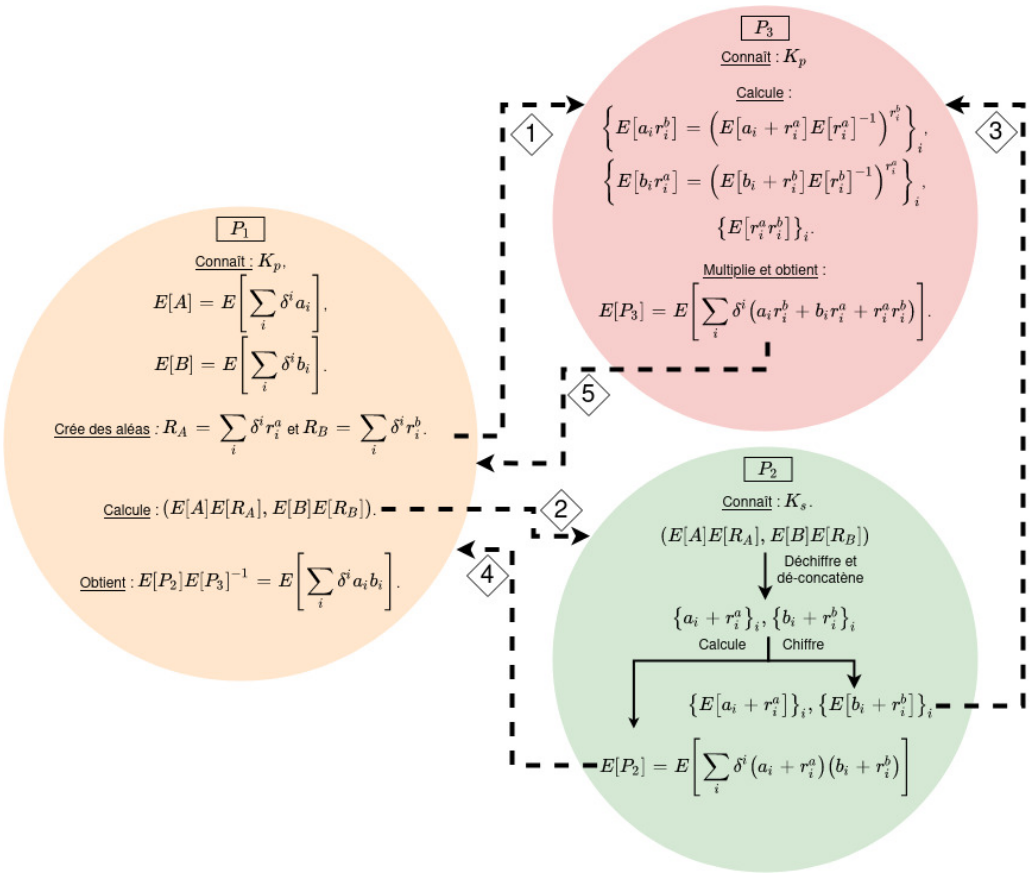


FIGURE 5.6 – Schéma de multiplication sécurisée entre $E[A]$ et $E[B]$ impliquant 3 serveurs : $Mul_e^{P_1,P_2,P_3}(E[A], E[B])$.

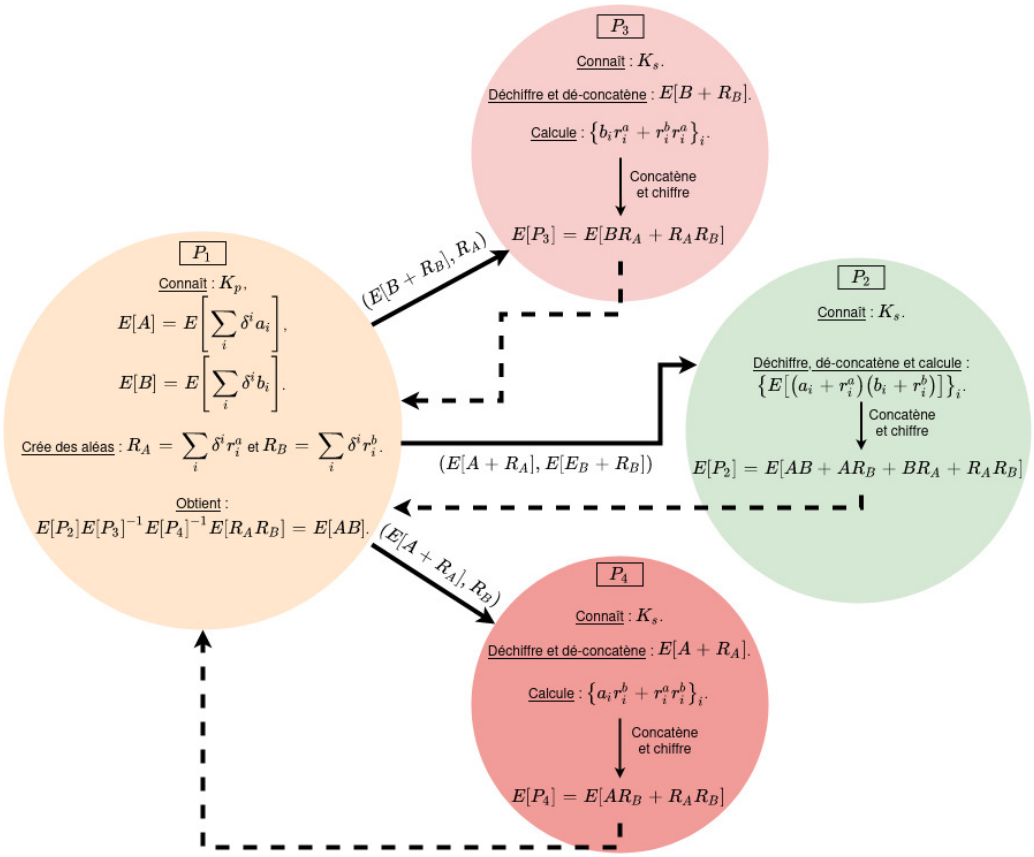


FIGURE 5.7 – Schéma de multiplication sécurisée entre $E[A]$ et $E[B]$ impliquant 4 serveurs : $Mul_e^{P_1,P_2,P_3,P_4}(E[A], E[B])$.

qu'il transmet à P_1.

Pour finir, P_1 soustrait ce qu'il a reçu de P_3 à ce qu'il a reçu de P_2 :

$$E[\sum_i \delta^i (a_i + r_i^a)(b_i + r_i^b)]E[\sum_i \delta^i (a_i r_i^b + b_i r_i^a + r_i^a r_i^b)]^{-1} = E[\sum_i \delta^i a_i b_i] \tag{5.26}$$

P_1 obtient alors $E[AB]$ le produit de $E[A]$ et $E[B]$.

L'opération de multiplication sécurisée avec 4 serveurs :

Tout comme lors du calcul du maximum sécurisé, le but ici est que le nombre de données concaténées n'influence pas les coûts de communication. Cette objectif nous oblige à utiliser un serveur supplémentaire noté P_4, comme présenté à la Figure 5.7. Pour cela, il faut partager entre P_3 et P_4 le calcul du terme permettant d'enlever le bruit généré par les produits que fait P_2. Certaines étapes restent cependant identiques au cas avec 3 serveurs :

	[Bel+19]	3 serveurs	4 serveurs
$P_1 \to P_2$	2nλ	2λ	$\lambda + \beta$
$P_1 \to P_3$		2β	$\lambda + \beta$
$P_1 \to P_4$			2λ
$P_2 \to P_1$	nλ	λ	λ
$P_2 \to P_3$		2nλ	
$P_3 \to P_1$		λ	λ
$P_4 \to P_1$			λ
Total	3nλ	$(2n + 4)\lambda + 2\beta$	$7\lambda + 2\beta$

TABLE 5.3 – Comparaison des coûts de communications pour la fonction de multiplications sécurisée.

— P_1 crée R_A et R_B et transmet $E[A + R_A]$ ainsi que $E[B + R_B]$ à P_2.
— P_2 déchiffre, dé-concatène et calcule le produit $E[P_2] = E[(A+R_A)(B+R_B)]$ qu'il envoie à P_1.

P_1 transmet cette fois-ci $E[B + R_B]$ ainsi que R_A à P_3 et $E[A + R_A]$ ainsi que R_B à P_4. Les rôles de P_3 et P_4 étant symétriques, nous ne présentons que celui de P_3. Attention néanmoins, P_4 a des paramètres d'entrées qui lui sont propres.

P_3 déchiffre et dé-concatène $E[B + R_B]$ et calcule :

$$E[P_3] = E[\sum_i \delta^i r_i^a (b_i + r_i^b)] \tag{5.27}$$

P_3 envoie à P_1 le résultat de cette opération.

P_1 connaît alors $E[P_2]$, $E[P_3]$ et $E[P_4] = E[\sum_i \delta^i r_i^b (a_i + r_i^a)]$. Étant donné que P_1 possède R_A et R_B, il peut retrouver le produit de A et B comme suit :

$$E[AB] = E[P2]E[P_3]^{-1}E[P_4]^{-1}E[R_A R_B] \tag{5.28}$$

Finalement, P_1 a bien effectué le calcul sécurisé de la multiplication des deux données chiffrées $E[A]$ et $E[B]$.

- **Les coûts théoriques :**

Nous présentons et comparons ici les coûts de communication des 3 méthodes : nos deux approches et la méthode de [Bel+19]. En utilisant les mêmes notations que pour le protocole du maximum, nous obtenons la Table 5.3.

On constate au travers de cette dernière que les coûts de communications sont réduits d'environ un tiers entre [Bel+19] et notre méthode avec données concaténées. Cependant,

	[Bel+19]	3 serveurs	4 serveurs
P_1	n(3**C** + 2**E**)	2**C**	3**C**
P_2	2n**D**	(2n+1)**C** + 2**D**	**C** + 2**D**
P_3		n**C** + 2n**E**	**C** + **D**
P_4			**C** + **D**
Total	n(3**C** + 2**D** + 2**E**)	(3n+3)**C** + 2**D** +2n**E**	6**C** + 4**D**

TABLE 5.4 – Comparaison des coûts de calcul pour la fonction maximum sécurisée.

il est important de noter que la méthode de [Bel+19] ne fait intervenir que 2 serveurs alors que notre proposition en demande 3.

En ce qui concerne notre méthode de calcul sécurisé de la multiplication avec 4 serveurs, on voit comme à l'instar de la proposition avec 3 serveurs pour le maximum sécurisé, qu'elle ne fait pas directement intervenir le nombre de données concaténées n. Quand bien même le nombre de données n est un paramètre sous-jacent du facteur β, ce dernier peut être majoré par λ. Par conséquent, le coût de communication entre notre méthode avec 4 serveurs et la méthode [Bel+19] est inférieur d'environ un facteur $\frac{n}{3}$. Avec $n = 40$, celle-ci donne des coûts de calculs divisés par environ 13.

En ce qui concerne les coûts de calcul et en gardant les mêmes notations que pour le maximum sécurisé, la Table 5.4 nous présente les efforts que chaque serveur doit fournir. Les contraintes induites par la concaténation des données font que les coûts de calcul entre la solution de [Bel+19] et celle présentée avec 3 serveurs ne se trouvent optimiser que pour le nombre de déchiffrements. En effet, ce dernier est diminué d'un facteur n, cependant le nombre des autres opérations (*i.e.* chiffrement et exponentiation modulaire) reste quant à lui similaire.

En revanche, en ce qui concerne la solution avec 4 serveurs, comme cette dernière se passe des calculs d'exponentiations modulaires, elle est moins coûteuse en termes de chiffrements et déchiffrements d'un facteur $\frac{n}{2}$ par rapport à [Bel+19].

Il faut pondérer ces résultats par le fait que bien que nous prenions en compte les principales opérations (en termes de coûts) d'autres telles que la concaténation, déconcaténation, ne sont pas considérées. Bien que bien moins coûteuses elles n'apparaissent pas dans le protocole de [Bel+19].

Enfin, il est important de noter que chaque solution a un paramétrage ainsi qu'un fonctionnement qui lui est propre. Ainsi, bien que la solution avec 4 serveurs demande à ce que deux d'entre eux possèdent la clé secrète K_s, les serveurs P_2, P_3 et P_4 ne communiquent jamais entre eux.

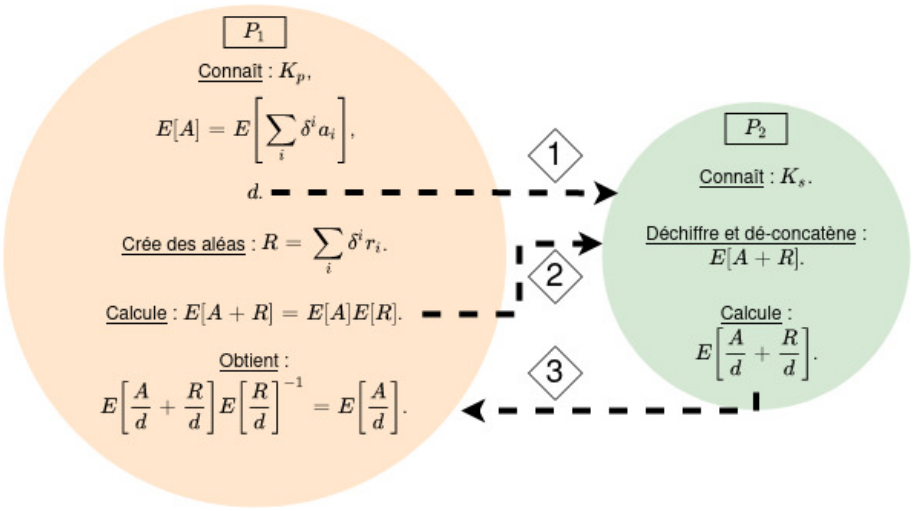


FIGURE 5.8 – Schéma de division sécurisée entre $E[A]$ et d impliquant 2 serveurs : $Div_e^{P_1,P_2}(E[A], d)$.

L'opération de division sécurisée :

Cette opération prendra place après une multiplication et pendant la phase d'apprentissage afin de diviser par le facteur de quantification δ. En effet, comme présenté dans la Section 3.1.3, lors de la phase d'apprentissage, il faut multiplier les gradients d'erreur par le *learning rate* or ce dernier est généralement un nombre décimal compris entre 0 et 1. Cependant, le cryptosystème de Damgård–Jurik (c.f. 2.1.1.2) travaille uniquement sur les entiers. Il n'est ainsi pas possible de multiplier un chiffré par une valeur décimale. La solution à ce problème est d'effectuer la division par l'inverse du *learning rate* et d'approximer à l'entier le plus proche du résultat. D'autre part, si le *mini-batch* est utilisé, ce dernier demande à ce que les gradients des erreurs soient divisés par la taille du *batch*.

Dans cette section, nous proposons une méthode, résumée par la Figure 5.8, qui permet de diviser chaque valeur d'une donnée concaténée chiffrée $E[A]$ par une valeur en clair d.

Le protocole s'effectue comme suit :

— P_1 crée les $\{r_i\}_i$, les concatène dans le vecteur R, masque $E[A]$ avec R et envoie le résultat $E[A+R]$ à P_2.

— P_2 déchiffre et dé-concatène $E[A+R]$ et effectue le calcul de $round(\frac{a_i+r_i}{d})$ (où $round(\cdot)$ est la fonction arrondie).

— Finalement, P_2 concatène puis chiffre ses résultats dans le vecteur $E[\frac{A}{d} + \frac{R}{d}]$ qu'il transmet à P_1.

— P_1 calcule $\{round(\frac{r_i}{d})\}_i$ et fait le produit de $E[\frac{A}{d} + \frac{R}{d}]$ par $E[\frac{R}{d}]^{-1}$ afin d'obtenir

	[Bel+19]	2 serveurs
$P_1 \to P_2$	$\mathrm{n}\lambda + \theta$	$\lambda + \theta$
$P_2 \to P_1$	$\mathrm{n}\lambda$	λ
Total	$2\mathrm{n}\lambda + \theta$	$2\lambda + \theta$

TABLE 5.5 – Comparaison des coûts de communications pour $Div_e^{P_1,P_2}(E[A], d)$.

	[Bel+19]	2 serveurs
P_1	$2\mathrm{n}\mathbf{C}$	$2\mathbf{C}$
P_2	$\mathrm{n}(\mathbf{C} + \mathbf{D})$	$\mathbf{C} + \mathbf{D}$
Total	$\mathrm{n}(3\mathbf{C} + \mathbf{D})$	$3\mathbf{C} + \mathbf{D}$

TABLE 5.6 – Comparaison des coûts de calculs pour $Div_e^{P_1,P_2}(E[A], d)$.

$E[\frac{A}{d}]$.

Ce protocole permet d'obtenir une approximation de la division. En effet, à cause des opérations $round(\cdot)$, il peut y avoir une perte de précision. Par exemple, prenons $a_i = 16$, $r_i = 9$ et $d = 7$. P_2 calcule $round(\frac{16+9}{7}) = 4$ et P_1 soustrait à ce résultat $round(\frac{9}{7}) = 1$ pour obtenir $\frac{a_i}{d} = 3$ or $round(\frac{16}{7}) = 2$. Ainsi, ce protocole peut au maximum concéder une erreur de 1 par rapport au protocole dans le clair.

- **Les coûts théoriques :**

La Table 5.5 donne les coûts théoriques de cette approche, utilisant les mêmes notations que précédemment. Ici, seuls sont pris en compte l'algorithme de division de [Bel+19] et celui sur données concaténées que nous venons de présenter. La Table 5.5 permet de déduire que notre solution est moins coûteuse d'un facteur n que celle de [Bel+19]. Dans le cas où $n = 40$, cela permet d'avoir des coûts de communication fortement diminués.

En ce qui concerne les coûts de calculs, ils sont fournis par la Table 5.6. Ces derniers sont environ diminués d'un facteur n. Cependant, il faut tenir compte des opérations telles que la concaténation et la dé-concaténation que nous ne prenons pas en compte.

L'opération de la somme des données concaténées sécurisée :

Cette opération, dont le fonctionnement est précisé Figure 5.9, ne nécessite que 2 serveurs. Comme précédemment, P_1 commence par créer des nombres aléatoires $\{r_i\}_i$ qu'il concatène dans le but de masquer $E[A]$. Le serveur P_2 reçoit alors $E[A + R]$ qu'il déchiffre et dé-concatène. P_2 procède à la somme des valeurs dé-concaténées puis chiffre le résultat :

$$E[P_2] = E[\sum_i (a_i + r_i)] = E[\sum_i a_i + \sum_i r_i] \tag{5.29}$$

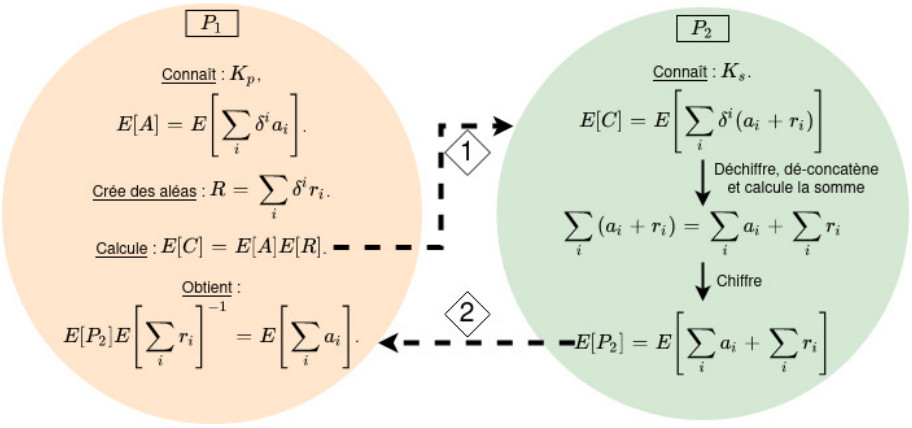


FIGURE 5.9 – Schéma de la somme sécurisée des données concaténées dans $E[A]$ impliquant 2 serveurs : $Sum_e^{P_1,P_2}(E[A])$.

$E[P_2]$ est envoyé à P_1, qui connaissant les $\{r_i\}_i$, peut les retrancher afin d'enlever le bruit du masquage :

$$E[P_2]E[\sum_i r_i]^{-1} = E[\sum_i a_i] \tag{5.30}$$

P_1 obtient finalement $E[\sum_i a_i]$.

- **Les coûts théoriques :**

Du point de vue des coûts théoriques de notre approche, il n'est pas possible de les comparer à ceux de [Bel+19] car les auteurs n'utilisent pas la concaténation. Il n'est pas non plus nécessaire de faire intervenir 3 serveurs, car comme nous allons le voir les coûts sont déjà faibles.

En ce qui concerne les communications, les coûts sont les suivants :

— $P_1 \to P_2$, P_1 n'envoie qu'un seul chiffré ainsi le coût est λ.

— $P_2 \to P_1$, de même P_2 ne retransmet qu'un seul chiffré pour un coût de λ.

Pour ce qui est des coûts de calcul :

— P_1 ne calcule que 2 chiffrements et une exponentiation modulaire. Il est à noter que cette dernière opération peut être évitée. Comme P_1 connaît en clair les $\{r_i\}$, il peut alors directement chiffrer :

$$E[(-\sum_i r_i) \mod K_p] = E[\sum_i r_i]^{-1} \tag{5.31}$$

ce qui est une optimisation. On peut ainsi approximer les coûts de calcul d'un tel algorithme par $2\mathbf{C}$

— P_2 ne procède qu'à un déchiffrement suivi d'un chiffrement, les coûts sont $\mathbf{C} + \mathbf{D}$.

Quand bien même les coûts de communication et de calculs sont faibles, il est rappelé qu'une telle opération n'est pas nécessaire dans le contexte de données non concaténées.

Maintenant que nous avons présenté le schéma, l'arrangement des données ainsi que le traitement sécurisé des opérations de base, regardons comment sécuriser l'apprentissage et la classification au travers d'un réseau de *machine learning*.

5.1.3 Un modèle de *machine learning* sécurisé sur données concaténées

Dans les sections précédentes, nous avons défini les entités en présence, le type de cryptosystème utilisé, les tenants et aboutissants de la concaténation de données et enfin la sécurisation de certains calculs de base sur données concaténées. Nous allons maintenant voir comment agencer toutes ces briques pour arriver dans un premier temps à faire la classification sécurisée de données au travers d'un réseau de *machine learning*, puis dans un second temps comment réaliser l'apprentissage.

Dans les deux sous-sections qui suivent, nous supposons que le serveur P_1 possède l'ensemble des données de l'EDD concaténées et chiffrées par l'algorithme de Damgård–Jurik.

5.1.3.1 La classification sécurisée sur données concaténées

Pour commencer, supposons que le serveur P_1 possède en plus des données chiffrées, les poids et biais c.f. 3.1.1 d'une architecture de *machine learning* capable de classer les données avec une précision suffisante.

Nous avons vu en préambule comment calculer l'opération de multiplication sécurisée. Comme le cryptosystème de Damgård–Jurik utilisé est additivement homomorphe, ni les calculs des produits $\{w_i x_i\}_i$ ni leur somme $\sum_i w_i x_i + b$ ne posent de problème. Il reste à voir la fonction d'activation ReLU $f(x)$. Cette dernière comme rappeler (5.3), vaut x si $x > 0$ et 0 sinon. Elle peut également être vue comme le maximum entre x et 0. Or, nous avons montré à la Section 5.1.2.2 comment faire le calcul sécurisé du maximum. Il est donc possible de sécuriser l'ensemble des calculs fait par un neurone.

Comme un réseau de neurones n'est qu'une succession de couches de neurones, il nous est alors possible de sécuriser l'ensemble du réseau de *machine learning* jusqu'à la couche de sortie. Le *cloud* calcule ainsi la classification chiffrée des données d'entrées qu'il retourne à l'EDD. L'EDD peut les déchiffrer puis les dé-concaténer pour obtenir les résultats des

classifications.

La classification permet une fois que les poids et biais d'une architecture sont trouvés de connaître la classe à laquelle appartient la donnée d'entrée. Cependant, l'ensemble des opérations présentées ici seront réutilisées pendant l'apprentissage lors du calcul du *feedforward*.

5.1.3.2 L'apprentissage sécurisé sur données concaténées

Regardons maintenant comment sécuriser l'apprentissage. Comme présentée dans la Section 3.1.3, il est composé de deux étapes :

1. Le *feedforward*, à partir des poids actuels calcule, la classification de l'entrée.
2. La *backpropagation*, à partir des différences entre la vraie classification des données et celle proposée par le *feedforward*, calcule le gradient d'erreur pour chaque poids et biais de chaque couche.

Par conséquent, le principe du *machine learning* est d'enchaîner suffisamment de *feedforward* puis de *backpropagation* avec des données différentes afin d'obtenir une précision suffisante lors de la classification.

Dans notre étude, nous avons utilisé le *mean squared error* (MSE) ou erreur quadratique moyenne comme fonction de coût. Cette dernière est définie par :

$$\frac{1}{2n}\sum_{i=1}^{n}(t_i - o_i)^2 \tag{5.32}$$

où o_i est la sortie du $i^{\text{ième}}$ neurone lors de la phase de *feedforward* et t_i est la valeur que devrait avoir ce neurone si la classification était correcte. La *backpropagation* ne calcule pas directement la formule du MSE (5.32) mais plutôt sa dérivée :

$$\frac{\delta MSE(t_i, o_i)}{\delta o_i} = -\frac{1}{n}\sum_{i=1}^{n} t_i - o_i \tag{5.33}$$

Ce calcul ne fait intervenir que des additions, soustractions et divisions, il nous est alors possible de le calculer de façon sécurisée avec des entrées chiffrées par un protocole additivement homomorphe.

Au cours des tests, nous avons utilisé la fonction d'optimisation *Stochastic Gradient Descent* (SGD) ou algorithme du gradient stochastique. Cette fonction d'optimisation met à jour tous les poids et biais de chaque couche après un *feedforward*. Pour cela, on calcule

le gradient de l'erreur associé que l'on multiplie au *learning rate* avant de le soustraire à la valeur poids ou biais actuel (c.f. Section 3.1.3). Un exemple est donné (5.5) dans le cas d'une couche de neurones. Un tel calcul dans le contexte où le réseau de *machine learning* utilisé est un réseau de neurones, avec comme fonctions d'activation ReLU, ne fait intervenir que des multiplications et des comparaisons avec 0, deux opérations que nous avons sécurisées.

Enfin, du fait que nous utilisons des données concaténées, nous pouvons employer la technique de *mini-batch*. Lorsque nous effectuons un *feedforward* sécurisé, les calculs se font cette fois sur un vecteur concaténé, *i.e.* sur plusieurs données en même temps. Le *zero padding* assure que les données ne débordent pas les unes sur les autres, ni durant la phase *feedforward* ni durant la *backpropagation*. Cela permet de calculer directement après un *feedforward* une *backpropagation* et ainsi obtenir pour chaque poids et biais un vecteur contenant les gradients de l'erreur. Ainsi, il nous reste à faire la somme de toutes ces valeurs concaténées ce que nous avons sécurisé et dont le schéma de principe est présenté dans la Figure 5.9.

Enfin, le *mini-batch* force à diviser le résultat de la somme par le nombre de valeurs sommées. Cette opération est à coupler à l'opération de multiplication par le *learning rate*. Ce dernier étant généralement entre 0 et 1 et ne pouvant pas travailler sur des valeurs décimales avec Damgård–Jurik, nous devons alors diviser par l'inverse du *learning rate*. Or, les étapes de division par la taille du *mini-batch* et de division par l'inverse du *learning rate* sont à faire au même moment, il est ainsi possible de les coupler.

Comme notre architecture de *machine learning* est un réseau de neurones qui utilise la fonction d'activation ReLU, de coût MSE, d'optimisation SGD et le *mini-batch*, sa sécurisation au travers de l'algorithme de Damgård–Jurik est possible.

5.1.3.3 Résultats expérimentaux

Au cours de nos tests, nous avons utilisé le réseau de neurones présenté dans la Figure 5.10. Ce réseau est composé de 2 entrées qui sont connectées à une première couche de 10 neurones, elle-même connectée à une seconde couche de 10 neurones. Enfin, la dernière couche de neurones est reliée à 2 sorties. Du fait que nous souhaitons faire l'apprentissage et la classification des opérations *AND* et *XOR* entre deux valeurs, cela implique que la première couche de notre réseau doit elle-même être composée de deux entrées. Concernant la sortie, la classification se fait comme suit :

— Si la valeur que prend la première sortie s_0 est supérieure à celle de s_1, alors le

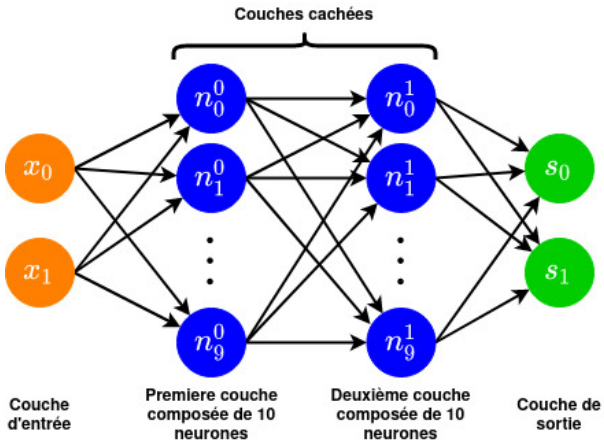


FIGURE 5.10 – L'architecture de notre réseau de neurones.

x_0	x_1		AND	Sortie		XOR	Sortie
0	0		0	$s_0 > s_1$		0	$s_0 > s_1$
0	1		0	$s_0 > s_1$		1	$s_0 < s_1$
1	0		0	$s_0 > s_1$		1	$s_0 < s_1$
1	1		1	$s_0 < s_1$		0	$s_0 > s_1$

TABLE 5.7 – Valeurs des opérations AND et XOR ainsi que des sorties de la classification en fonction des données d'entrées.

résultat de l'opération AND ou XOR est classifié comme étant 0.

— Dans le cas contraire, la classification donne 1.

Cela est résumé dans la Table 5.7.

Un point important que nous n'avons pas précisé jusqu'ici est que nous quantifions le label des données d'entraînement. En effet, sans cette quantification, le label vaudrait $(s_0, s_1) = [1, 0]$ ou $[0, 1]$ en fonction des entrées. Cependant, la sortie de la classification sécurisée est quantifiée par δ qui doit être relativement grand (*e.g.* 2^{20}). Or, dans ce cas, le calcul du MSE ne serait que peu pertinent. Cela est dû au fait que l'ajout ou le retrait de 1 à une grande valeur δ n'aura que peu d'influence. Ce qui n'est plus le cas si l'on quantifie la valeur du label par δ *i.e.* $(s_0, s_1) = [\delta, 0]$ ou $[0, \delta]$.

Pour ce qui concerne le cryptosystème de Damgård–Jurik, nous avons une clé publique de 2048 bits. Le pas de quantification est $\delta = 2^{20}$, ce qui nous permet d'avoir $n = 40$ données concaténées par vecteur.

En ce qui concerne l'apprentissage, ce dernier est fait sur un nombre de 8000 données. La vérification quant à elle se fait sur un ensemble de 2000 entrées. Les temps donnés pour l'entraînement sont ceux pour une seule époque, qui comme nous allons le voir permet

	Classification		Apprentissage	
Paramétrage	**minimal**	**maximal**	**minimal**	**maximal**
AND et *XOR*	18.5 s	15.4 s	3h55 min	1h25 min

TABLE 5.8 – Temps de calculs de la classification et de l'apprentissage sécurisés.

d'atteindre un taux de convergence acceptable.

Nous avons également choisi deux paramétrages possibles :

1. Le paramétrage **minimal** : celui qui utilise le nombre minimal de serveurs pour le calcul sécurisé de la multiplication et du maximum. Il est par conséquent le moins performant en termes de coût de communication et de temps de calcul.
2. Le paramétrage **maximal** : qui, quant à lui, utilise le nombre maximal de serveurs. C'est pourquoi il est le plus performant.

La Table 5.8 reprend les temps de calculs pour les paramètres donnés en fonction des deux paramétrages **minimal** ou **maximal** pour la classification et l'apprentissage. Puisque les opérations *AND* et *XOR* utilisent la même architecture, les temps de calculs sont identiques.

En résumé, cette table confirme que pour la classification ou l'apprentissage la solution **maximale** est plus rapide que la **minimale** et les coûts de communications sont réduits.

Plus particulièrement, le calcul de la classification est relativement rapide entre 15 et 19 secondes pour classifier 40 données ce qui donne environ 2.1 données par seconde pour la solution **minimale** et presque 2.6 pour la **maximale**. Ces temps restent raisonnables dans un cas réel.

Pour l'apprentissage, l'utilisation d'un serveur supplémentaire améliore les temps de calcul : il est presque 2.8 fois plus rapide d'utiliser la solution **maximale** que la **minimale**.

L'apprentissage prend pour une époque moins de 4h pour la solution **minimale** et moins de 1h30 pour la **maximale**. Ces temps de calculs restent acceptables et raisonnables en utilisation réelle. Il reste maintenant à parler de la convergence de ces modèles.

Nous essayons d'apprendre des opérations binaires (*AND* et *XOR*) *i.e.* nos jeux de données sont composées des entrées : $[0,0], [0,1], [1,0]$ et $[1,1]$. De plus, nous supposons que ces jeux de données de test et d'apprentissage sont uniformément distribués *i.e.* la distribution des entrées est d'environ 25% pour chacune. On ne peut alors avoir comme précision de classification qu'un multiple de 25%. On cherche à savoir, lors de nos classifications, combien convergent vers une valeur de précision de 100% *i.e.* combien de réseaux après apprentissage classifient correctement toutes les entrées.

En utilisant le même réseau, mais en faisant l'apprentissage sur données en clair et avec des valeurs décimales, nous obtenons les convergences suivantes :

— 100% pour le *AND*, ce qui signifie que pour chaque valeur de poids et biais initiaux que l'on utilise, tous ont permis au réseau de converger vers une précision de 100% et ceux en une seule époque.

— 98.2% pour le *XOR* pour une époque, puis 1.6% supplémentaire si on fait 2 époques et enfin 0.2% qui ne convergent pas après 20 époques. Cela revient à dire qu'en une époque pour l'apprentissage 98.2% des tests convergeaient vers 100%. En continuant l'apprentissage avec une seconde époque, on obtient 99.8% des tests qui convergeaient, mais même en faisant 20 époques 0.2% certains tests ne convergeaient pas.

Cela semble cohérent, car l'opération *XOR* semble être plus complexe que l'opération *AND*.

En ce qui concerne l'apprentissage sécurisé, il faut prendre en compte 2 facteurs qui provoquent une perte de précision :

1. Les données sont quantifiées par δ.
2. La fonction de division sécurisée n'est en fait qu'une approximation.

Les résultats de notre approche sécurisée restent très satisfaisants même s'ils sont légèrement inférieures de 1.4% pour le *AND* et 0.6% pour le *XOR* par rapport à l'apprentissage en clair. Il est intéressant de noter que parmi les tests qui n'ont pas convergé, certains ont eu pour cause que les neurones ReLU étaient morts *i.e.* peu importe les arguments d'entrée, la sortie était toujours égale à 0. Or, une fois qu'un neurone ReLU est mort, il ne peut plus apprendre.

Après avoir discuté de nos résultats expérimentaux, concluons cette partie.

5.1.4 Discussion des résultats

Dans cette section, nous avons présenté une solution permettant de faire l'apprentissage ainsi que la classification sur des données concaténées de façon sécurisée. Nous avons utilisé un PHE additivement homomorphe et avons réutilisé la technique de concaténation présentée à la section 2.1.4. Une originalité de ce travail est l'utilisation de calculs multipartites impliquant plusieurs serveurs qui à l'aide du PHE permet de sécuriser les opérations de : maximum, multiplication, division et comparaison.

Comme les données utilisées sont concaténées, les coûts de calculs ainsi que de communications sont optimisés. Cette concaténation est particulièrement bien adaptée à la technique du *mini-batch* utilisée pour la phase d'apprentissage du modèle de *machine learning*.

Finalement, les résultats expérimentaux permettent de voir que la solution proposée offre une bonne convergence. De plus, en fonction de la paramétrisation générale de la solution, les temps de calcul peuvent varier de 1h25 à 3h55 pour la sécurisation de l'apprentissage et de 18.5s à 15.4s pour la classification de 40 données. Ceci nous permet alors de conclure qu'en utilisation réelle, cet algorithme est performant.

Maintenant que nous avons montré comment sécuriser un algorithme de *machine learning* sur données concaténées avec un PHE, passons à la même sécurisation, mais cette fois-ci sur des données compressées et chiffrées à l'aide d'un FHE.

5.2 Classification sécurisée par un FHE de données compressées

Dans cette section, nous utilisons un FHE qui permet plus d'opérations que les PHE au détriment cependant des performances générales (c.f. Section 4.2).

Nous allons montrer plusieurs scénarios dans lesquels les données sont compressées et chiffrées par un client. Ce dernier cherche à les externaliser vers un Serveur afin de pouvoir les stocker et les traiter.

Il est important de noter que, dans cette section, nous supposerons que les données sont disponibles sous forme chiffrée sur le Serveur effectuant leur décompression sécurisée. Cette hypothèse implique de prendre en compte les coûts de communication. Si les données sont chiffrées bit à bit avec un FHE et envoyées tel quel au Serveur, les coûts de communication sont plus importants que si les données sont envoyées de façon décompressée et sécurisée directement sur le Serveur. Pour résoudre ce problème, il existe des techniques permettant d'utiliser des chiffrements n'ayant pas d'expansion de chiffré telle que celles présentées dans la Section 4.6.

Pour commencer, nous nous intéressons à des données compressées à l'aide du codage entropique de Huffman. Par la suite, nous étudierons le cas d'images compressées par JPEG.

5.2.1 Le cas de données compressées de Huffman

Supposons que le Serveur possède des données client directement compressées grâce au codage entropique de Huffman. Deux solutions sont alors possibles :

1. Faire la classification sécurisée directement sur ces données compressées.
2. Décoder les données avant d'utiliser un algorithme de *machine learning* le tout de manière sécurisée.

En ce qui concerne le premier scénario, cela pose plusieurs problèmes. Premièrement, il y a la représentation des données. Huffman étant un codage entropique à longueur variable, les données compressées ne sont pas de tailles identiques. Or, le nombre d'entrées d'un réseau de *machine learning* et constant. Nous avons fait une étude préliminaire en utilisant des images et en réorganisant les pixels d'une des façons suivantes :

— en créant un flux contenant tous les pixels,
— en créant un flux pour chaque ligne de pixels de l'image.

Or, les tests n'ont pas été concluants. Il serait intéressant de les continuer avec des réseaux de *machine learning* plus complexes ou plus élaborés.

Pour ce qui est de la deuxième méthode, la Section 4 nous permet de sécuriser le décodage des données. Il reste à montrer comment évaluer de façon sécurisée un réseau de *machine learning* avec des données chiffrées à l'aide d'un FHE.

Chabanne *et al.* ont montré dans [Cha+17] comment évaluer de façon sécurisée un CNN. Le problème de ces modèles de *machine learning* est qu'ils utilisent des fonctions d'activation non-linéaires donc non évaluables par un FHE afin de classer les données. Leur idée consiste à approximer certaines fonctions d'activation par des polynômes qui eux sont évaluables par un FHE. Ceci permet en utilisant en entrée du réseau des données chiffrées par un FHE d'en obtenir la classification sécurisée.

La combinaison de ces deux méthodes permet d'effectuer la décompression sécurisée des données avant d'entreprendre leur classification, elle aussi, sécurisée.

Regardons maintenant comment faire dans le cas où les données client sont des images compressées par JPEG.

5.2.2 Application aux images compressées JPEG

L'objectif est de sécuriser la décompression de mot de code de Huffman afin d'en faire la classification sécurisée. Ici, nous supposons que le Serveur possède des images compressées JPEG chiffrées. Nous souhaitons les décompresser partiellement afin d'entreprendre

leur classification. Pour cela, nous nous appuyons sur les résultats de la Section 3 qui montre comment faire l'apprentissage ainsi que la classification sécurisée sur des images partiellement décompressées. Par la suite, dans la Section 4, nous avons proposé une méthode permettant de décompresser des données codées par Huffman. Or, le codage de Huffman est la dernière étape de la compression JPEG. Ainsi, en appliquant cette méthode à des données compressées par JPEG il est théoriquement possible de partiellement les décompresser et ainsi de faire leur classification sécurisée.

Il est alors possible de combiner l'idée de Chabanne *et al.* [Cha+17] avec notre solution permettant de décompresser de façon sécurisée des données Huffman.

Cela permet d'obtenir une solution globale qui décompresse partiellement des données avant d'en faire la classification, le tout de façon sécurisée.

Cependant, il est important de noter qu'au vu de sa complexité, cette technique reste théorique. Les cryptosystèmes *fully* homomorphes actuels ne permettent pas de réaliser en des temps pratiques autant d'opérations. Cependant, si un jour un FHE performant apparaît, toutes les opérations décrites dans cette thèse pourront être implémentées.

Maintenant que les solutions théoriques pour la classification avec des données FHE compressées Huffman ou JPEG ont été présentées, concluons ce chapitre.

5.3 Conclusion

Dans une première partie de ce chapitre, nous avons proposé une méthode permettant d'effectuer l'apprentissage et la classification de données concaténées sur un réseau de neurones artificiels de manière sécurisée. Nous avons utilisé un cryptosystème additivement homomorphe ainsi que du calcul multipartite afin de sécuriser l'ensemble des opérations nécessaires pour l'apprentissage et la classification d'un réseau de neurones. Une nouveauté est la sécurisation des opérations de maximum, multiplication, division et comparaison sur données concaténées sécurisées par un PHE additif. Dans la partie expérimentale, nous avons montré que notre solution était performante et surpassait celle de [Bel+19]. En ce qui concerne la classification, il est possible de traiter plusieurs données en quelques secondes. Pour ce qui est de l'apprentissage, ce dernier prend entre 1h25 et 3h55 en fonction des paramétrages du système. Plus généralement, cette solution montre comment calculer certaines opérations telles que le maximum ou la comparaison entre des valeurs concaténées et chiffrées par un PHE additif. Ces solutions basées sur des calculs multipartites peuvent être réutilisées dans d'autres contextes afin d'optimiser les coûts de

calcul et de communication.

Dans la suite, nous avons proposé de combiner les différentes solutions des Sections 3 et 4 afin de montrer comment classer à l'aide d'un algorithme de *machine learning* et de façon sécurisée des données compressées Huffman. La première solution proposée, dont l'objectif est de faire l'apprentissage sur données compressées Huffman, n'a pas montré de résultats probants. Cependant, une étude plus approfondie serait intéressante. La deuxième solution consiste à décompresser de façon sécurisée les données avant de les utiliser dans le modèle de *machine learning*. Cette solution est théoriquement possible. Cependant, avec les cryptosystèmes *fully* homomorphes actuels, la complexité algorithmique reste trop élevée.

Pour finir, nous avons étudié un scénario dans lequel le client externalise des données compressées JPEG. Cela nous a permis de montrer qu'il est théoriquement possible d'en faire la décompression partielle de façon sécurisée avant de les labelliser. Cette solution s'appuie sur la classification de données partiellement décompressées montrée précédemment.

Pour conclure cette thèse, nous reprendrons l'ensemble des problématiques traitées ainsi que les différentes publications qui en découlent. Nous ouvrirons également le sujet à de nouvelles perspectives qui sont la continuité des travaux présentés dans ce manuscrit.

Conclusion

Durant les dernières décennies, l'évolution des technologies de l'information et des communications comme en électronique, permettent la collecte massive de données et leur traitement. En particulier, dans le domaine médical, le volume d'information qui concerne la santé d'un individu est de plus en plus important. Ces informations : imagerie médicale, analyse biologique, test d'effort, ... sont numérisées et maintenant le plus souvent externalisées vers un *cloud*. L'augmentation du volume, la complémentarité ainsi que de la précision des données permettent aux spécialistes : médecins, chirurgiens, cardiologues, ... de mieux appréhender les symptômes des patients et ainsi leur fournir un traitement plus adapté. L'objectif de l'externalisation est de réduire les coûts d'entretiens et de maintenance de l'infrastructure informatique en les délégants à une compagnie externe. Cette externalisation promeut également d'autres avantages : une meilleure interopérabilité entre les différents acteurs de la santé et offre la possibilité de traiter massivement les données au travers d'algorithmes *Big Data* et d'intelligence artificielle. Ce type de traitement peut permettre d'effectuer un dépistage précoce ou encore identifier des redondances dans certains syndromes ou des risques de pathologies et ainsi prévenir au plus tôt les complications et maladies. D'autre part, la sophistication des *Implantable Medical Devices* (IMD) permet de fournir au patient un traitement toujours plus adapté, tout en recueillant des données de santé en continu afin qu'elles soient analysées rapidement. Comme nous l'avons montré dans le cas d'une prothèse de genoux, ce type d'analyse précoce permet d'identifier des anomalies dans les mesures et indique ainsi au patient d'aller consulter au plus vite son médecin. Ce dernier pourra alors lui fournir le traitement adéquat et ainsi empêcher l'infection qui mènerait à une ré-intervention chirurgicale.

Après avoir discuté des données et de leur exploitation, notamment dans le domaine médical, nous nous sommes intéressés à leur sécurité. Ce type de données est à forte valeur ajoutée pour un attaquant qui pourrait les utiliser pour son compte ou les revendre à un tiers. Ces données peuvent être la cause de discrimination à l'embauche, à l'obtention d'un prêt bancaire, ... Il est alors primordial d'en assurer la sécurité notamment au travers de leur confidentialité, intégrité et traçabilité. Gouvernements et instances internationales, conscients de cette problématique, ont promulgué un certains nombres de réglementa-

tions à leur égard. Ces réglementations expriment la façon d'utiliser, de conserver et de distribuer des données personnelles. Elles s'accompagnent d'amendes dissuasives en cas de fraude. Aujourd'hui, il existe des méthodes permettant de protéger les données. Comme nous l'avons vu, certaines sont peu coûteuses en termes de calcul et de communication, mais ne permettent pas le traitement sécurisé des données. D'autres méthodes permettent de le faire, au détriment cependant, des performances.

Dans le même temps, il est apparu que les données sont généralement stockées et transmises sous forme compressée afin de diminuer les coûts. Or, les traitements s'effectuent sur des données décompressées. Nous avons analysé le standard de compression d'image JPEG afin d'étudier les perspectives de traitements sur données compressées ou partiellement compressées. Cela nous a permis de conclure dans le Chapitre 1 par la nécessité de regrouper les trois problématiques de traitement, compression et sécurité des données dans une seule et unique chaîne de traitement.

Dans cette optique, au Chapitre 2, nous avons abordé la problématique de la sécurisation d'un traitement de données issue d'un IMD. Un point critique de ce scénario est le fait que l'IMD soit un appareil avec de fortes contraintes en termes de puissance de calcul, de batterie, de mémoire, ... Or, les cryptosystèmes additivement homomorphes que nous utilisons pour sécuriser le traitement des données, sont coûteux. Nous avons proposé un algorithme permettant d'utiliser de façon conjointe des cryptosystèmes rapides et légers au niveau de l'IMD et des *Partially Homomorphic Encryptions* (PHE) au niveau de l'Interface Homme-Machine (IHM). Cette méthode permet de diminuer la pression calculatoire présente au niveau de l'IMD tout en sécurisant le traitement des données. Une autre problématique inhérente au PHE est apparue. Ces derniers, pour des raisons de sécurité, nécessitent d'avoir un grand espace des données en clair. Or, les données utilisées sont généralement de longueur faible. Nous avons proposé une méthode de concaténation des données permettant d'optimiser les coûts de communication, mais également de calcul. Comme les traitements sont chiffrés, cette méthode fait en sorte de prévenir tout débordement de calculs entre données concaténées. Nous avons également proposé une solution permettant de coupler à notre protocole une technique de tatouage numérique. Cette dernière assure en plus de la confidentialité des données, leur intégrité et leur traçabilité. Les expérimentations ont montré que les calculs entrepris par l'IMD sont rapides. D'une manière générale, l'algorithme est performant et permet de traiter presque 500 données chaque seconde.

Ce premier traitement sécurisé sur données concaténées nous a mené à étudier la

possibilité de faire des traitements plus élaborés au travers d'algorithmes de *machine learning* sur données (*e.g.* images médicales) compressées. Pour cela, nous avons commencé par étudier la structure de l'algorithme de compression JPEG afin d'en connaître toutes les étapes. Un second travail a consisté à établir pour chaque étape de JPEG une base de données d'images partiellement décompressées. Ces bases de données ont permis d'étudier l'influence des étapes de la décompression sur les performances d'algorithmes de *machine learning*. Nous avons utilisé des architectures NN et CNN pour nos tests. Nos résultats ont montré que pour des images en niveaux de gris de petite taille (28×28 pixels) issues de MNIST, les étapes de compression n'affectaient pas la performance des réseaux NN. Cependant, en ce qui concerne des images plus complexes de taille 32×32 pixels appartenant à CIFAR-10, la compression a une influence négative sur la performance des réseaux CNN. À cet égard, l'étape de compression DCT est particulièrement néfaste. Il est alors apparu qu'un compromis entre le temps de décompression et la perte de performance des réseaux était envisageable.

À la suite de ces travaux, nous avons cherché à montrer qu'il est possible de décompresser de façon sécurisée l'algorithme de Huffman qui constitue la dernière étape de la compression JPEG. L'idée sous-jacente de cette étude est de pouvoir proposer une chaîne de traitement permettant à un client de compresser des images par JPEG avant de les chiffrer puis de les externaliser vers un *cloud*. Une fois les données externalisées, l'objectif du *cloud* est de réaliser leur décompression afin de les traiter à l'aide d'un algorithme de *machine learning* le tout de façon sécurisée. Pour commencer, nous nous sommes intéressés à l'algorithme de compression de Huffman. Certaines conditions permettant d'évaluer cet algorithme nécessitent de calculer à la fois des multiplications et des additions. Or, ce type d'opération ne peut être simultanément sécurisé qu'au travers de chiffrements *fully* homomorphes (FHE) qui sont malheureusement moins performants que les PHE en termes de puissance de calcul. Cela nous a permis néanmoins de proposer une méthode de décompression d'un mot de code pour laquelle toutes les opérations entreprises sont évaluables par un FHE. Ainsi, nous avons montré comment décompresser un flux binaire contenant plusieurs mots de code de Huffman. Nos résultats expérimentaux montre que la méthode reste performante : quelques secondes suffisent pour décoder une image 28×28 pixels. Cependant, la méthode est pénalisée par le FHE utilisé : les performances actuelles des FHE restent limitées. Grâce aux outils développés pour répondre à la problématique de sécurisation de la décompression d'un flux binaire de mot de code de Huffman, nous avons montré comment sécuriser d'autres applications : évaluation d'un DT et décompression

d'un RLE.

Finalement, nous avons montré comment, en utilisant les différentes solutions proposées dans ce manuscrit, il est possible de sécuriser une chaîne de traitement effectuée par un algorithme de *machine learning* sur données partiellement compressées ou concaténées. Pour commencer, à l'aide d'un PHE et un calcul multipartite, nous avons sécurisé l'apprentissage ainsi que la classification de données concaténées. Nous avons sécurisé grâce au calcul multipartite des opérations telles que le maximum et la comparaison entre des données chiffrées par un PHE additif. Cette solution est performante que ce soit pour la classification où en moins de 20 secondes, 40 données peuvent être étiquetées ou pour l'apprentissage qui prend entre 1h25 et 3h55 en fonction du paramétrage choisi. Pour continuer cette étude, nous avons proposé un scénario dans lequel des données sont directement codées par Huffman et dont la classification sécurisée est effectuée par un modèle de *machine learning*. Cette étude préliminaire n'a pas donné de résultats probants. Cependant, elle reste intéressante et une étude plus approfondie pourrait être envisagée. Finalement, nous avons montré comment en partant de données compressées JPEG, il est théoriquement possible de les décompresser partiellement avant de les classer. Cependant, au vu des performances actuelles des FHE, la solution ne semble pas envisageable en application réelle. Toutefois, de nouveaux FHE plus performants à venir pourraient permettre d'implémenter la méthode proposée. De plus, cette méthode a l'avantage de définir un certain nombre de nouveaux concepts et outils réutilisables dans d'autres applications.

Pour finir, au fil des avancées de nos travaux, d'autres problématiques et questionnements sont apparus. Nous aurions aimé essayer d'y répondre, mais le manque de temps, nous a contraint à y renoncer. Nous proposons d'en citer quelques-uns comme réflexions et perspectives futures :

— En ce qui concerne les travaux sur la sécurisation des IMD, l'objectif principal est de proposer des traitements performants et efficaces. Ces derniers permettent au médecin de mieux analyser et suivre les symptômes et donc de mieux traiter le patient. Il aurait été intéressant d'étudier la sécurisation d'autres types d'opérations ou des traitements sur données appartenant à des capteurs différents. Nous aurions également pu, dans une moindre mesure, et sous d'autres contraintes, appliquer des calculs multipartites afin de proposer des traitements plus sophistiqués. Ces derniers fourniraient une analyse plus fine des données. Dans une idée similaire, un passage d'un PHE à un FHE peut être envisagé.

— Pour l'apprentissage sur données compressées, les architectures utilisées lors des

tests étaient des architectures issues de la littérature scientifique. Il pourrait être intéressant de proposer de nouvelles architectures spécifiques au problème traité afin d'évaluer leurs performances. Une étude proposant un apprentissage directement sur le flux binaire pourrait également être envisagé. Il ne faut cependant pas perdre de vue, au cours de cette étude, qu'utiliser un réseau de *machine learning* très complexe engendrera potentiellement un coût calculatoire supérieur au coût d'une unique décompression totale des données. Des tests avec d'autres jeux de données (*e.g.* ImageNet), de plus grande taille voir même sur d'autres types de données (*e.g.* fichier audio MP3) pourraient consolider les résultats obtenus. Comme nous l'avons vu à la fin de ce manuscrit, une étude plus approfondie de l'apprentissage sur données codées par Huffman serait intéressante. Cette dernière réorganiserait les mots de code et définirait une architecture de *machine learning* plus adaptée au problème.

— La décompression de Huffman sécurisée pourrait possiblement être optimisée en calculant la profondeur multiplicative minimale théorique de notre solution. Ceci mènerait éventuellement à des optimisations de certaines opérations et permettrait le traitement de flux binaire de plus grande taille, voir des traitements plus rapides. Il peut être intéressant de faire la même étude en utilisant un FHE dont l'espace du clair est $\mathbb{Z}_2$ ce qui pourrait améliorer les temps de calculs de la solution. Plus généralement, faire la décompression de toutes les étapes de JPEG à l'aide d'un FHE serait pertinente.

— Il serait intéressant de généraliser l'apprentissage et la classification sécurisée d'un NN à l'aide de calcul multipartite et d'un PHE sur un CNN. D'autres fonctions d'activation pourraient être sécurisées. Ceci permettrait d'augmenter le type et la diversité des modèles que l'on sécuriserait. D'un point de vue calculatoire, une implémentation des calculs sur GPU pourrait également être entreprise. En effet, la majorité des calculs sont parallélisables et bénéficieraient ainsi d'une telle implémentation. Enfin, un tatouage numérique du réseau ou des données permettrait de protéger le propriétaire des données ainsi que le *cloud* sur lequel les calculs sont effectués.

BIBLIOGRAPHIE

[Aba+16] Martín ABADI et al., *TensorFlow : Large-Scale Machine Learning on Heterogeneous Distributed Systems*, 2016, arXiv : 1603.04467 [cs.DC].

[AFN88] AFNOR, *Terminologie relative à la fiabilité-maintenabilité-disponibilité*, rapp. tech., nov. 1988.

[ANR74] Nasir AHMED, T. NATARAJAN et K. R. RAO, « Discrete Cosine Transform », in : *IEEE Transactions on Computers* C-23.*1* (jan. 1974), p. 90-93, ISSN : 1557-9956, DOI : 10.1109/T-C.1974.223784, URL : https://doi.org/10.1109/T-C.1974.223784.

[Arn00] Michael ARNOLD, « Audio watermarking : features, applications and algorithms », in : *2000 IEEE International Conference on Multimedia and Expo. ICME2000. Proceedings. Latest Advances in the Fast Changing World of Multimedia (Cat. No.00TH8532)*, t. 2, 2000 IEEE International Conference on Multimedia and Expo. ICME2000. Proceedings. Latest Advances in the Fast Changing World of Multimedia (Cat. No.00TH8532), juill. 2000, p. 1013-1016, ISBN : 0-7803-6536-4, DOI : 10.1109/ICME.2000.871531, URL : https://doi.org/10.1109/ICME.2000.871531.

[Aum+13] Jean-Philippe AUMASSON et al., « Quark : A Lightweight Hash », in : *Journal of Cryptology* 26.*2* (avr. 2013), p. 313-339, ISSN : 1432-1378, DOI : 10.1007/s00145-012-9125-6, URL : https://doi.org/10.1007/s00145-012-9125-6.

[Bar20] Elaine BARKER, « Recommendation for Key Management : Part 1 - General », in : *NIST Special Publication 800-56A Revision 3*, mai 2020, p. 1-171, DOI : 10.6028/NIST.SP.800-57pt1r5, URL : https://doi.org/10.6028/NIST.SP.800-57pt1r5.

[BC16] Dalel BOUSLIMI et Gouenou COATRIEUX, « A crypto-watermarking system for ensuring reliability control and traceability of medical images », in : *Signal Processing : Image Communication* 47 (sept. 2016), p. 160-169, ISSN : 0923-

5965, DOI : 10.1016/j.image.2016.05.021, URL : https://doi.org/10.1016/j.image.2016.05.021.

[Bel+17] Reda BELLAFQIRA et al., « Proxy Re-Encryption Based on Homomorphic Encryption », in : *Proceedings of the 33rd Annual Computer Security Applications Conference*, ACSAC 2017, Orlando, FL, USA : Association for Computing Machinery, déc. 2017, p. 154-161, ISBN : 978-1-4503-5345-8, DOI : 10.1145/3134600.3134616, URL : https://doi.org/10.1145/3134600.3134616.

[Bel+19] Reda BELLAFQIRA et al., « Secure Multilayer Perceptron Based on Homomorphic Encryption », in : *Digital Forensics and Watermarking*, sous la dir. de Chang D. YOO et al., Cham : Springer International Publishing, 2019, p. 322-336, ISBN : 978-3-030-11389-6.

[Ben94] Josh BENALOH, « Dense Probabilistic Encryption », in : *Selected Areas of Cryptography*, mai 1994, p. 1-9.

[Ber05] Daniel J BERNSTEIN, *Salsa20 specification*, 2005.

[BGN05] Dan BONEH, Eu-Jin GOH et Kobbi NISSIM, « Evaluating 2-DNF Formulas on Ciphertexts », in : *Theory of Cryptography*, sous la dir. de Joe KILIAN, Berlin, Heidelberg : Springer Berlin Heidelberg, fév. 2005, p. 325-341, ISBN : 978-3-540-30576-7, DOI : 10.1007/978-3-540-30576-7_18, URL : https://doi.org/10.1007/978-3-540-30576-7_18.

[BGV14] Zvika BRAKERSKI, Craig GENTRY et Vinod VAIKUNTANATHAN, « (Leveled) fully homomorphic encryption without bootstrapping », in : *ACM Transactions on Computation Theory (TOCT)* 6.*3* (juill. 2014), p. 1-36, ISSN : 1942-3454, DOI : 10.1145/2633600, URL : https://doi.org/10.1145/2633600.

[Bit+17] Ahmad W. BITAR et al., « Blind digital watermarking in PDF documents using Spread Transform Dither Modulation », in : *Multimedia Tools and Applications* 76.*1* (jan. 2017), p. 143-161, ISSN : 1573-7721, DOI : 10.1007/s11042-015-3034-2, URL : https://doi.org/10.1007/s11042-015-3034-2.

[BK04] Ian F. BLAKE et Vladimir KOLESNIKOV, « Strong Conditional Oblivious Transfer and Computing on Intervals », in : *Advances in Cryptology - ASIACRYPT 2004*, sous la dir. de Pil Joong LEE, Berlin, Heidelberg : Springer

Berlin Heidelberg, déc. 2004, p. 515-529, ISBN : 978-3-540-30539-2, DOI : 10.1007/978-3-540-30539-2_36, URL : https://doi.org/10.1007/978-3-540-30539-2_36.

[Bog+07] Andrey BOGDANOV et al., « PRESENT : An Ultra-Lightweight Block Cipher », in : *Cryptographic Hardware and Embedded Systems - CHES 2007*, sous la dir. de Pascal PAILLIER et Ingrid VERBAUWHEDE, Berlin, Heidelberg : Springer Berlin Heidelberg, sept. 2007, p. 450-466, ISBN : 978-3-540-74735-2, DOI : 10.1007/978-3-540-74735-2_31, URL : https://doi.org/10.1007/978-3-540-74735-2_31.

[Boh+13] Andras BOHO et al., « End-To-End Security for Video Distribution : The Combination of Encryption, Watermarking, and Video Adaptation », in : *IEEE Signal Processing Magazine* 30.*2* (mars 2013), p. 97-107, ISSN : 1053-5888, DOI : 10.1109/MSP.2012.2230220, URL : https://doi.org/10.1109/MSP.2012.2230220.

[Bor+12] Julia BORGHOFF et al., « PRINCE – A Low-Latency Block Cipher for Pervasive Computing Applications », in : *Advances in Cryptology – ASIACRYPT 2012*, sous la dir. de Xiaoyun WANG et Kazue SAKO, Berlin, Heidelberg : Springer Berlin Heidelberg, déc. 2012, p. 208-225, ISBN : 978-3-642-34961-4, DOI : 10.1007/978-3-642-34961-4_14, URL : https://doi.org/10.1007/978-3-642-34961-4_14.

[Bos+15] Raphael BOST et al., « Machine learning classification over encrypted data. », in : *NDSS*, fév. 2015, p. 1-14, ISBN : 1-891562-38-X, DOI : 10.14722/ndss.2015.23241, URL : https://doi.org/10.14722/ndss.2015.23241.

[Bot10] Léon BOTTOU, « Large-Scale Machine Learning with Stochastic Gradient Descent », in : *Proceedings of COMPSTAT'2010*, sous la dir. d'Yves LECHEVALLIER et Gilbert SAPORTA, Heidelberg : Physica-Verlag HD, sept. 2010, p. 177-186, ISBN : 978-3-7908-2604-3, DOI : 10.1007/978-3-7908-2604-3_16, URL : https://doi.org/10.1007/978-3-7908-2604-3_16.

[Bou+12] Dalel BOUSLIMI et al., « A Joint Encryption/Watermarking System for Verifying the Reliability of Medical Images », in : *IEEE Transactions on Information Technology in Biomedicine* 16.*5* (sept. 2012), p. 891-899, ISSN : 1089-7771, DOI : 10.1109/TITB.2012.2207730, URL : https://doi.org/10.1109/TITB.2012.2207730.

[Bou+16] Dalel Bouslimi et al., « Data hiding in encrypted images based on predefined watermark embedding before encryption process », in : *Signal Processing : Image Communication* 47 (sept. 2016), p. 263-270, ISSN : 0923-5965, DOI : 10.1016/j.image.2016.06.012, URL : https://doi.org/10.1016/j.image.2016.06.012.

[Bou+17] Guillaume Bouzillé et al., « Sharing health big data for research - A design by use cases : the INSHARE platform approach », in : *The 16th World Congress on Medical and Health Informatics (MedInfo2017)*, t. 245, Precision Healthcare through Informatics, series : Studies in Health Technology and Informatics, Hangzhou, China, août 2017, p. 303-307, DOI : 10.3233/978-1-61499-830-3-303, URL : https://doi.org/10.3233/978-1-61499-830-3-303.

[Bou13] Dalel Bouslimi, « Protection de données d'imagerie par tatouage et chiffrement-Application à la télémédecine », thèse de doct., Télécom Bretagne, 2013.

[Bou97] Thomas Boutell, « Png (portable network graphics) specification version 1.0 », in : *Network Working Group* (1997), p. 1-102.

[BP17] Alex Biryukov et Leo Perrin, *State of the Art in Lightweight Symmetric Cryptography*, Cryptology ePrint Archive, Report 2017/511, https://eprint.iacr.org/2017/511, 2017.

[Cac+00] Christian Cachin et al., « One-Round Secure Computation and Secure Autonomous Mobile Agents », in : *Automata, Languages and Programming*, sous la dir. d'Ugo Montanari, José D. P. Rolim et Emo Welzl, Berlin, Heidelberg : Springer Berlin Heidelberg, fév. 2000, p. 512-523, ISBN : 978-3-540-45022-1, DOI : 10.1007/3-540-45022-X_43, URL : https://doi.org/10.1007/3-540-45022-X_43.

[Can+18] Anne Canteaut et al., « Stream Ciphers : A Practical Solution for Efficient Homomorphic-Ciphertext Compression », in : *Journal of Cryptology* 31.*3* (juill. 2018), p. 885-916, ISSN : 1432-1378, DOI : 10.1007/s00145-017-9273-9, URL : https://doi.org/10.1007/s00145-017-9273-9.

[Cha+12] Sang-Yoon Chang et al., « Body Area Network Security : Robust Key Establishment Using Human Body Channel », in : *Proceedings of the 3rd USENIX Conference on Health Security and Privacy*, HealthSec'12, Bellevue, WA : USENIX Association, août 2012, p. 1-10.

[Cha+17] Hervé Chabanne et al., *Privacy-Preserving Classification on Deep Neural Network*, Cryptology ePrint Archive, Report 2017/035, https://eprint.iacr.org/2017/035, 2017.

[Che+10] Abbas Cheddad et al., « Digital image steganography : Survey and analysis of current methods », in : *Signal Processing* 90.*3* (mars 2010), p. 727-752, issn : 0165-1684, doi : 10.1016/j.sigpro.2009.08.010, url : https://doi.org/10.1016/j.sigpro.2009.08.010.

[Che+14] Sharan Chetlur et al., *cuDNN : Efficient Primitives for Deep Learning*, 2014, arXiv : 1410.0759 [cs.NE].

[Cho+15] François Chollet et al., *Keras*, https://keras.io, 2015.

[CMB00] I. J. Cox, M. L. Miller et J. A. Bloom, « Watermarking applications and their properties », in : *Proceedings International Conference on Information Technology : Coding and Computing (Cat. No.PR00540)*, Proceedings International Conference on Information Technology : Coding and Computing (Cat. No.PR00540), mars 2000, p. 6-10, isbn : 0-7695-0540-6, doi : 10.1109/ITCC.2000.844175, url : https://doi.org/10.1109/ITCC.2000.844175.

[Coa+13] Gouenou Coatrieux et al., « Reversible Watermarking Based on Invariant Image Classification and Dynamic Histogram Shifting », in : *IEEE Transactions on Information Forensics and Security* 8.*1* (jan. 2013), p. 111-120, issn : 1556-6021, doi : 10.1109/TIFS.2012.2224108, url : https://doi.org/10.1109/TIFS.2012.2224108.

[Com17] Medicare Payment Advisory Commission, *An overview of the medical device industry*, rapp. tech., Washington, DC : Medical Payment Advisory Commission, 2017, p. 1-242, url : http://www.medpac.gov/docs/default-source/reports/jun17_ch7.pdf?sfvrsn=0.

[Con16] Parlement européen et Conseil, *Règlement (UE) 2016/679 du Parlement européen et du Conseil du 27 avril 2016 relatif à la protection des personnes physiques à l'égard du traitement des données à caractère personnel et à la libre circulation de ces données, et abrogeant la directive 95/46/CE (règlement général sur la protection des données)*, avr. 2016, url : https://eur-lex.europa.eu/legal-content/FR/TXT/?uri=CELEX%3A32016R0679.

[Con90] Parlement européen et Conseil, *Directive du Conseil 90/385/CEE : concernant le rapprochement des législations des États membres relatives aux dispositifs médicaux implantables actifs*, juin 1990, URL : http://data.europa.eu/eli/dir/1990/385/2007-10-11.

[Dav78] Ruth M. Davis, « The data encryption standard in perspective », in : *IEEE Communications Society Magazine* 16.*6* (nov. 1978), p. 5-9, ISSN : 0148-9615, DOI : 10.1109/MCOM.1978.1089771, URL : https://doi.org/10.1109/MCOM.1978.1089771.

[DDA96] Liliane Dusserre, Henry Ducrot et François-André Allaërt, *L'information médicale, l'ordinateur et la loi*, Editions médicales internationales, 1996.

[Dej+17] Mathieu Dejean-Servières et al., *Study of the Impact of Standard Image Compression Techniques on Performance of Image Classification with a Convolutional Neural Network*, Research Report, INSA Rennes ; Univ Rennes ; IETR ; Institut Pascal, déc. 2017, p. 1-21, URL : https://hal.archives-ouvertes.fr/hal-01725126.

[Den+10] Tamara Denning et al., « Patients, Pacemakers, and Implantable Defibrillators : Human Values and Security for Wireless Implantable Medical Devices », in : *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, Atlanta, Georgia, USA : Association for Computing Machinery, avr. 2010, p. 917-926, ISBN : 9781605589299, DOI : 10.1145/1753326.1753462, URL : https://doi.org/10.1145/1753326.1753462.

[Den+89] John S Denker et al., « Neural network recognizer for hand-written zip code digits », in : *Advances in neural information processing systems*, 1989, p. 323-331.

[DFK08] Tamara Denning, Kevin Fu et Tadayoshi Kohno, « Absence Makes the Heart Grow Fonder : New Directions for Implantable Medical Device Security », in : *Proceedings of the 3rd Conference on Hot Topics in Security*, HOTSEC'08, San Jose, CA : USENIX Association, jan. 2008, p. 1-7.

[DGK07] Ivan Damgård, Martin Geisler et Mikkel Krøigaard, « Efficient and Secure Comparison for On-Line Auctions », in : *Information Security and Privacy*, Berlin, Heidelberg : Springer Berlin Heidelberg, 2007, p. 416-430, isbn : 978-3-540-73458-1, doi : 10.1007/978-3-540-73458-1_30, url : https://doi.org/10.1007/978-3-540-73458-1_30.

[DHS11] John Duchi, Elad Hazan et Yoram Singer, « Adaptive Subgradient Methods for Online Learning and Stochastic Optimization », in : *Journal of Machine Learning Research* 12.*61* (2011), p. 2121-2159, url : http://jmlr.org/papers/v12/duchi11a.html.

[DJ03] Ivan Damgård et Mads Jurik, « A Length-Flexible Threshold Cryptosystem with Applications », in : *Information Security and Privacy*, sous la dir. de Rei Safavi-Naini et Jennifer Seberry, Berlin, Heidelberg : Springer Berlin Heidelberg, juin 2003, p. 350-364, isbn : 978-3-540-45067-2, doi : 10.1007/3-540-45067-X_30, url : https://doi.org/10.1007/3-540-45067-X_30.

[DP08] Christophe De Cannière et Bart Preneel, « Trivium », in : *New Stream Cipher Designs : The eSTREAM Finalists*, sous la dir. de Matthew Robshaw et Olivier Billet, Berlin, Heidelberg : Springer Berlin Heidelberg, 2008, p. 244-266, isbn : 978-3-540-68351-3, doi : 10.1007/978-3-540-68351-3_18, url : https://doi.org/10.1007/978-3-540-68351-3_18.

[DR01] Joan Daemen et Vincent Rijmen, « Reijndael : The Advanced Encryption Standard », in : *Dr. Dobb's Journal : Software Tools for the Professional Programmer* 26.*3* (mars 2001), p. 137-139, issn : 1044-789X.

[Dud13] Jarek Duda, « Asymmetric numeral systems as close to capacity low state entropycoders », in : *CoRR* abs/1311.2540 (2013), arXiv : 1311.2540, url : http://arxiv.org/abs/1311.2540.

[Dwo15] Morris Dworkin, *SHA-3 Standard : Permutation-Based Hash and Extendable-Output Functions*, août 2015, doi : 10.6028/NIST.FIPS.202, url : https://doi.org/10.6028/NIST.FIPS.202.

[Ehr+78] William F Ehrsam et al., *Message verification and transmission error detection by block chaining*, US Patent 4,074,066, fév. 1978.

[EJ03] Patrik Ekdahl et Thomas Johansson, « A New Version of the Stream Cipher SNOW », in : *Selected Areas in Cryptography*, sous la dir. de Kaisa Nyberg et Howard Heys, Berlin, Heidelberg : Springer Berlin Heidelberg, fév. 2003, p. 47-61, isbn : 978-3-540-36492-4, doi : 10.1007/3-540-36492-7_5, url : https://doi.org/10.1007/3-540-36492-7_5.

[Elg85] Taher Elgamal, « A public key cryptosystem and a signature scheme based on discrete logarithms », in : *IEEE Transactions on Information Theory* 31.*4* (1985), p. 469-472, issn : 1557-9654, doi : 10.1109/TIT.1985.1057074, url : https://doi.org/10.1109/TIT.1985.1057074.

[Eri+98] Bradley J. Erickson et al., « Wavelet compression of medical images. », in : *Radiology* 206.*3* (mars 1998), p. 599-607, issn : 0033-8419, doi : 10.1148/radiology.206.3.9494473, url : https://doi.org/10.1148/radiology.206.3.9494473.

[Etz00] Amitai Etzioni, « The new enemy of privacy : Big bucks », in : *Challenge* 43.*3* (2000), p. 91-106, doi : 10.1080/05775132.2000.11472156, url : https://doi.org/10.1080/05775132.2000.11472156.

[FG16] Dan Fu et Gabriel Guimaraes, *Using compression to speed up image classification in artificial neural networks*, rapp. tech., oct. 2016, p. 1-10, url : http://www.danfu.org/files/CompressionImageClassification.pdf.

[FH89] Evelyn Fix et J. L. Hodges, « Discriminatory Analysis. Nonparametric Discrimination : Consistency Properties », in : *International Statistical Review / Revue Internationale de Statistique* 57.*3* (1989), p. 238-247, issn : 03067734, 17515823, doi : 10.2307/1403797, url : https://doi.org/10.2307/1403797.

[Fia88] Orestes Fiandra, « The first pacemaker implant in America », in : *Pacing and Clinical Electrophysiology* 11.*8* (août 1988), p. 1234-1238, issn : 0147-8389, doi : 10.1111/j.1540-8159.1988.tb03977.x, url : https://doi.org/10.1111/j.1540-8159.1988.tb03977.x.

[Fis01] Marc Fischlin, « A Cost-Effective Pay-Per-Multiplication Comparison Method for Millionaires », in : *Topics in Cryptology - CT-RSA 2001*, sous la dir. de David Naccache, Berlin, Heidelberg : Springer Berlin Heidelberg, avr. 2001, p. 457-471, isbn : 978-3-540-45353-6, doi : 10.1007/3-540-45353-9_33, url : https://doi.org/10.1007/3-540-45353-9_33.

[Fle+19] Marine Flechet et al., « Machine learning versus physicians' prediction of acute kidney injury in critically ill adults : a prospective evaluation of the AKIpredictor », in : *Critical Care* 23.*1* (août 2019), p. 1-10, ISSN : 1364-8535, DOI : 10.1186/s13054-019-2563-x, URL : https://doi.org/10.1186/s13054-019-2563-x.

[FM82] Kunihiko Fukushima et Sei Miyake, « Neocognitron : A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition », in : *Competition and Cooperation in Neural Nets*, sous la dir. de Shun-ichi Amari et Michael A. Arbib, Berlin, Heidelberg : Springer Berlin Heidelberg, fév. 1982, p. 267-285, ISBN : 978-3-642-46466-9, DOI : 10.1007/978-3-642-46466-9_18, URL : https://doi.org/10.1007/978-3-642-46466-9_18.

[FSW03] Pierre-Alain Fouque, Jacques Stern et Geert-Jan Wackers, « CryptoComputing with Rationals », in : *Financial Cryptography*, sous la dir. de Matt Blaze, Berlin, Heidelberg : Springer Berlin Heidelberg, 2003, p. 136-146, ISBN : 978-3-540-36504-4, DOI : 10.1007/3-540-36504-4_10, URL : https://doi.org/10.1007/3-540-36504-4_10.

[FV12] Junfeng Fan et Frederik Vercauteren, *Somewhat Practical Fully Homomorphic Encryption*, Cryptology ePrint Archive, Report 2012/144, https://eprint.iacr.org/2012/144, 2012.

[GB10] Xavier Glorot et Yoshua Bengio, « Understanding the difficulty of training deep feedforward neural networks », in : *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, sous la dir. d'Yee Whye Teh et Mike Titterington, t. 9, Proceedings of Machine Learning Research, Chia Laguna Resort, Sardinia, Italy : PMLR, mai 2010, p. 249-256, URL : http://proceedings.mlr.press/v9/glorot10a.html.

[GBC16] Ian Goodfellow, Yoshua Bengio et Aaron Courville, *Deep learning*, MIT press, 2016.

[GCL91] Richard A. Garibaldi, Deborah Cushing et Trudy Lerer, « Risk factors for postoperative infection », in : *The American Journal of Medicine* 91.*3, Supplement 2* (sept. 1991), S158-S163, ISSN : 0002-9343, DOI : 10.1016/0002-9343(91)90362-2, URL : https://doi.org/10.1016/0002-9343(91)90362-2.

[GD98] M. W. Gardner et S. R. Dorling, « Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences », in : *Atmospheric Environment* 32.*14* (août 1998), p. 2627-2636, issn : 1352-2310, doi : 10.1016/S1352-2310(97)00447-0, url : https://doi.org/10.1016/S1352-2310(97)00447-0.

[Gen+09] Craig Gentry et al., *A fully homomorphic encryption scheme*, t. 20, 9, Stanford university Stanford, 2009, p. 1-209.

[Gér19] Aurélien Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : Concepts, tools, and techniques to build intelligent systems*, O'Reilly Media, 2019.

[GH91] ilson. Greatbatch et Curtis F. Holmes, « History of implantable devices », in : *IEEE Engineering in Medicine and Biology Magazine* 10.*3* (sept. 1991), p. 38-41, issn : 1937-4186, doi : 10.1109/51.84185, url : https://doi.org/10.1109/51.84185.

[GHH16] Ananda Mohon Ghosh, Debashish Halder et S. K. Alamgir Hossain, « Remote health monitoring system through IoT », in : *2016 5th International Conference on Informatics, Electronics and Vision (ICIEV)*, mai 2016, p. 921-926, isbn : 978-1-5090-1269-5, doi : 10.1109/ICIEV.2016.7760135, url : https://doi.org/10.1109/ICIEV.2016.7760135.

[GHS12] Craig Gentry, Shai Halevi et Nigel P. Smart, « Homomorphic Evaluation of the AES Circuit », in : *Advances in Cryptology – CRYPTO 2012*, sous la dir. de Reihaneh Safavi-Naini et Ran Canetti, Berlin, Heidelberg : Springer Berlin Heidelberg, août 2012, p. 850-867, isbn : 978-3-642-32009-5, doi : 10.1007/978-3-642-32009-5_49, url : https://doi.org/10.1007/978-3-642-32009-5_49.

[GLC00] Guodong Guo, Stan Z. Li et Kapluk Chan, « Face recognition by support vector machines », in : *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*, Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580), mars 2000, p. 196-201, isbn : 0-7695-0580-5, doi : 10.1109/AFGR.2000.840634, url : https://doi.org/10.1109/AFGR.2000.840634.

[GM84] Shafi Goldwasser et Silvio Micali, « Probabilistic encryption », in : *Journal of Computer and System Sciences* 28.*2* (avr. 1984), p. 270-299, ISSN : 0022-0000, DOI : 10.1016/0022-0000(84)90070-9, URL : https://doi.org/10.1016/0022-0000(84)90070-9.

[Gol09] Oded Goldreich, *Foundations of Cryptography Volume 2, Basic Applications*, 1st, USA : Cambridge University Press, sept. 2009, p. 1-452, ISBN : 978-0-521-11991-7.

[Gub+13] Jayavardhana Gubbi et al., « Internet of Things (IoT) : A vision, architectural elements, and future directions », in : *Future Generation Computer Systems* 29.*7* (sept. 2013), p. 1645-1660, ISSN : 0167-739X, DOI : 10.1016/j.future.2013.01.010, URL : https://doi.org/10.1016/j.future.2013.01.010.

[Gue+18] Lionel Gueguen et al., « Faster Neural Networks Straight from JPEG », in : *Advances in Neural Information Processing Systems*, sous la dir. de S. Bengio et al., t. 31, Curran Associates, Inc., 2018, p. 1-12, URL : https://proceedings.neurips.cc/paper/2018/file/7af6266cc52234b5aa339b16695f7fc4-Paper.pdf.

[Had+20] Sahar Haddad et al., « Joint Watermarking-Encryption-JPEG-LS for Medical Image Reliability Control in Encrypted and Compressed Domains », in : *IEEE Transactions on Information Forensics and Security* 15 (2020), p. 2556-2569, ISSN : 1556-6021, DOI : 10.1109/TIFS.2020.2972159, URL : https://doi.org/10.1109/TIFS.2020.2972159.

[Hal+08] Daniel Halperin et al., « Pacemakers and Implantable Cardiac Defibrillators : Software Radio Attacks and Zero-Power Defenses », in : *2008 IEEE Symposium on Security and Privacy (sp 2008)*, mai 2008, p. 129-142, DOI : 10.1109/SP.2008.31, URL : https://doi.org/10.1109/SP.2008.31.

[HD13] Xiali Hei et Xiaojiang Du, *Security for wireless implantable medical devices*, Springer, avr. 2013, p. 1-56, ISBN : 978-1-4614-7153-0, DOI : 10.1007/978-1-4614-7153-0, URL : https://doi.org/10.1007/978-1-4614-7153-0.

[Hea+18] U.S. Department of Health et al., *Content of Premarket Submissions for Management of Cybersecurity in Medical Devices*, oct. 2018.

[HJM07] Martin Hell, Thomas Johansson et Willi Meier, « Grain : a stream cipher for constrained environments », in : *International Journal of Wireless and Mobile Computing* 2.*1* (mai 2007), p. 86-93, doi : 10.1504/IJWMC.2007.013798, url : https://doi.org/10.1504/IJWMC.2007.013798.

[Hon+06] Deukjo Hong et al., « HIGHT : A New Block Cipher Suitable for Low-Resource Device », in : *Cryptographic Hardware and Embedded Systems - CHES 2006*, sous la dir. de Louis Goubin et Mitsuru Matsui, Berlin, Heidelberg : Springer Berlin Heidelberg, oct. 2006, p. 46-59, isbn : 978-3-540-46561-4, doi : 10.1007/11894063_4, url : https://doi.org/10.1007/11894063_4.

[HS06] Geoffrey E Hinton et Ruslan R Salakhutdinov, « Reducing the dimensionality of data with neural networks », in : *science* 313.*5786* (2006), p. 504-507.

[Hud+18] Graham Hudson et al., « JPEG-1 standard 25 years : past, present, and future reasons for a success », in : *Journal of Electronic Imaging* 27.*4* (août 2018), p. 1-19, doi : 10.1117/1.JEI.27.4.040901, url : https://doi.org/10.1117/1.JEI.27.4.040901.

[Huf52] David A. Huffman, « A Method for the Construction of Minimum-Redundancy Codes », in : *Proceedings of the IRE* 40.*9* (sept. 1952), p. 1098-1101, issn : 2162-6634, doi : 10.1109/JRPROC.1952.273898, url : https://doi.org/10.1109/JRPROC.1952.273898.

[ISO12] ISO/IEC, *Information technology — Security techniques — Lightweight cryptography — Part 3 : Stream ciphers*, Standard, Geneva, CH : International Organization for Standardization, oct. 2012, p. 1-26.

[ISO18] ISO, *Biological evaluation of medical devices — Part 1 : Evaluation and testing within a risk management process*, Standard, Geneva, CH : International Organization for Standardization, oct. 2018, p. 1-41.

[JMV01] Don Johnson, Alfred Menezes et Scott Vanstone, « The Elliptic Curve Digital Signature Algorithm (ECDSA) », in : *International Journal of Information Security* 1.*1* (août 2001), p. 36-63, issn : 1615-5262, doi : 10.1007/s102070100002, url : https://doi.org/10.1007/s102070100002.

[Jou13] Yeun-Ho JOUNG, « Development of implantable medical devices : from an engineering perspective », in : *International neurourology journal* 17.*3* (sept. 2013), p. 98-106, ISSN : 2093-4777, DOI : 10.5213/inj.2013.17.3.98, URL : https://doi.org/10.5213/inj.2013.17.3.98.

[JS11] T JAYALAKSHMI et A SANTHAKUMARAN, « Statistical normalization and back propagation for classification », in : *International Journal of Computer Theory and Engineering* 3.*1* (2011), p. 89-93.

[JS92] W. JAMES et Charles STEIN, « Estimation with Quadratic Loss », in : *Breakthroughs in Statistics : Foundations and Basic Theory*, sous la dir. de Samuel KOTZ et Norman L. JOHNSON, New York, NY : Springer New York, 1992, p. 443-460, ISBN : 978-1-4612-0919-5, DOI : 10.1007/978-1-4612-0919-5_30, URL : https://doi.org/10.1007/978-1-4612-0919-5_30.

[Jur03] Mads Johan JURIK, *Extensions to the paillier cryptosystem with applications to cryptological protocols*, Citeseer, août 2003.

[Kar99] Brandenburg KARLHEINZ, « mp3 and aac explained », in : *journal of the audio engineering society* (sept. 1999), p. 1-12.

[KB17] Diederik P. KINGMA et Jimmy BA, *Adam : A Method for Stochastic Optimization*, 2017, arXiv : 1412.6980 [cs.LG].

[KBC97] Hugo KRAWCZYK, Mihir BELLARE et Ran CANETTI, *HMAC : Keyed-Hashing for Message Authentication*, fév. 1997.

[KBH09] Florian KERSCHBAUM, Debmalya BISWAS et Sebastiaan de HOOGH, « Performance Comparison of Secure Comparison Protocols », in : *2009 20th International Workshop on Database and Expert Systems Application*, août 2009, p. 133-136, DOI : 10.1109/DEXA.2009.37, URL : https://doi.org/10.1109/DEXA.2009.37.

[KG13] Cameron F. KERRY et Patrick D. GALLAGHER, « Digital Signature Standard (DSS) », in : *FIPS PUB 186-4*, juill. 2013, p. 1-121, DOI : 10.6028/NIST.FIPS.186-4, URL : http://dx.doi.org/10.6028/NIST.FIPS.186-4.

[Kha+14a] Asifullah KHAN et al., « A recent survey of reversible watermarking techniques », in : *Information Sciences* 279 (sept. 2014), p. 251-272, ISSN : 0020-0255, DOI : 10.1016/j.ins.2014.03.118, URL : https://doi.org/10.1016/j.ins.2014.03.118.

[Kha+14b] Wahid Khan et al., « Implantable Medical Devices », in : *Focal Controlled Drug Delivery*, sous la dir. d'Abraham J. Domb et Wahid Khan, Boston, MA : Springer US, 2014, p. 33-59, ISBN : 978-1-4614-9434-8, DOI : 10.1007/978-1-4614-9434-8_2, URL : https://doi.org/10.1007/978-1-4614-9434-8_2.

[Kim+15] Younghyun Kim et al., « Vibration-based secure side channel for medical devices », in : *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, juin 2015, p. 1-6, DOI : 10.1145/2744769.2744928, URL : https://doi.org/10.1145/2744769.2744928.

[Kof+08] David Koff et al., « Pan-Canadian Evaluation of Irreversible Compression Ratios ("Lossy" Compression) for Development of National Guidelines », in : *Journal of digital imaging* 22.*6* (oct. 2008), p. 569, ISSN : 1618-727X, DOI : 10.1007/s10278-008-9139-7, URL : https://doi.org/10.1007/s10278-008-9139-7.

[Kot+02] Geoffrey Kotzar et al., « Evaluation of MEMS materials of construction for implantable medical devices », in : *Biomaterials* 23.*13* (juill. 2002), p. 2737-2750, ISSN : 0142-9612, DOI : 10.1016/S0142-9612(02)00007-8, URL : https://doi.org/10.1016/S0142-9612(02)00007-8.

[KP18] Anoop Kumar et Jonathan Pillai, « Chapter 13 - Implantable drug delivery systems : An overview », in : *Nanostructures for the Engineering of Cells, Tissues and Organs*, William Andrew Publishing, jan. 2018, p. 473-511, ISBN : 978-0-12-813665-2, DOI : 10.1016/B978-0-12-813665-2.00013-2, URL : https://doi.org/10.1016/B978-0-12-813665-2.00013-2.

[KR16] B. Shravan Kumar et Vadlamani Ravi, « A survey of the applications of text mining in financial domain », in : *Knowledge-Based Systems* 114 (déc. 2016), p. 128-147, ISSN : 0950-7051, DOI : 10.1016/j.knosys.2016.10.003, URL : https://doi.org/10.1016/j.knosys.2016.10.003.

[Kra02] Elliot Krames, « Implantable devices for pain control : spinal cord stimulation and intrathecal therapies », in : *Best Practice & Research Clinical Anaesthesiology* 16.*4* (déc. 2002), p. 619-649, ISSN : 1521-6896, DOI : 10.1053/bean.2002.0263, URL : https://doi.org/10.1053/bean.2002.0263.

[Kra92] Hugo KRAWCZYK, « How to predict congruential generators », in : *Journal of Algorithms* 13.*4* (déc. 1992), p. 527-545, ISSN : 0196-6774, DOI : 10.1016/0196-6774(92)90054-G, URL : https://doi.org/10.1016/0196-6774(92)90054-G.

[Kri09] Alex KRIZHEVSKY, « Learning multiple layers of features from tiny images », in : (2009).

[KSH12] Alex KRIZHEVSKY, Ilya SUTSKEVER et Geoffrey E HINTON, « ImageNet Classification with Deep Convolutional Neural Networks », in : *Advances in Neural Information Processing Systems*, sous la dir. de F. PEREIRA et al., t. 25, Curran Associates, Inc., 2012, URL : https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[Kum+19] C. Vinoth KUMAR et al., « Encrypted separable reversible watermarking with authentication and error correction », in : *Multimedia Tools and Applications* 78.*6* (mars 2019), p. 7005-7027, ISSN : 1573-7721, DOI : 10.1007/s11042-018-6450-2, URL : https://doi.org/10.1007/s11042-018-6450-2.

[KW17] Janocha KATARZYNA et Marian Czarnecki WOJCIECH, *On Loss Functions for Deep Neural Networks in Classification*, 2017, arXiv : 1702.05659 [cs.LG].

[Kwo+19] Donghwoon KWON et al., « A survey of deep learning-based network anomaly detection », in : *Cluster Computing* 22.*1* (jan. 2019), p. 949-961, ISSN : 1573-7543, DOI : 10.1007/s10586-017-1117-8, URL : https://doi.org/10.1007/s10586-017-1117-8.

[Laf19] Pauline LAFORGE, « Le dossier médical partagé », in : *Actualités Pharmaceutiques* 58.*584, Supplement* (mars 2019), p. 29-30, ISSN : 0515-3700, DOI : 10.1016/j.actpha.2019.01.028, URL : https://doi.org/10.1016/j.actpha.2019.01.028.

[Lai17] Kim LAINE, *Simple encrypted arithmetic library 2.3. 1*, 2017.

[LAJ11] Chunxiao LI, Raghunathan ANAND et Niraj K. JHA, « Hijacking an insulin pump : Security attacks and defenses for a diabetes therapy system », in : *2011 IEEE 13th International Conference on e-Health Networking, Applications and Services*, juin 2011, p. 150-156, ISBN : 978-1-61284-697-2, DOI :

10.1109/HEALTH.2011.6026732, URL : https://doi.org/10.1109/HEALTH.2011.6026732.

[Lar+03] Berit LARSSON et al., « Lessons from the first patient with an implanted pacemaker : 1958-2001 », in : *Pacing and Clinical Electrophysiology* 26.*1p1* (mars 2003), p. 114-124, DOI : 10.1046/j.1460-9592.2003.00162.x, URL : https://doi.org/.

[Lat+11] Benoît LATRÉ et al., « A survey on wireless body area networks », in : *Wireless Networks* 17.*1* (jan. 2011), p. 1-18, ISSN : 1572-8196, DOI : 10.1007/s11276-010-0252-4, URL : https://doi.org/10.1007/s11276-010-0252-4.

[LCB98] Yann LECUN, Corinna CORTES et Christopher J. C. BURGES, « The MNIST database of handwritten digits », in : *http ://yann. lecun. com/exdb/mnist/* (1998).

[LEc+02] Pierre L'ECUYER et al., « An Object-Oriented Random-Number Package with Many Long Streams and Substreams », in : *Operations Research* 50.*6* (déc. 2002), p. 1073-1075, DOI : 10.1287/opre.50.6.1073.358, URL : https://doi.org/10.1287/opre.50.6.1073.358.

[LeC+89] Y. LECUN et al., « Backpropagation Applied to Handwritten Zip Code Recognition », in : *Neural Computation* 1.*4* (1989), p. 541-551, ISSN : 0899-7667, DOI : 10.1162/neco.1989.1.4.541, URL : https://doi.org/10.1162/neco.1989.1.4.541.

[LEc96] Pierre L'ECUYER, « Combined Multiple Recursive Random Number Generators », in : *Operations Research* 44.*5* (oct. 1996), p. 816-822, ISSN : 0030-364X, DOI : 10.1287/opre.44.5.816, URL : https://doi.org/10.1287/opre.44.5.816.

[LEc99] Pierre L'ECUYER, « Tables of linear congruential generators of different sizes and good lattice structure », in : *Mathematics of Computation* 68.*225* (jan. 1999), p. 249-260, ISSN : 0025-5718, DOI : 10.1090/S0025-5718-99-00996-5, URL : https://doi.org/10.1090/S0025-5718-99-00996-5.

[Li+14] Mu LI et al., « Efficient Mini-Batch Training for Stochastic Optimization », in : *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, New York, New York,

USA : Association for Computing Machinery, août 2014, p. 661-670, ISBN : 978-1-4503-2956-9, DOI : 10.1145/2623330.2623612, URL : https://doi.org/10.1145/2623330.2623612.

[Lia+06] Shiguo LIAN et al., « Commutative watermarking and encryption for media data », in : *Optical Engineering* 45.*8* (août 2006), p. 1-3, DOI : 10.1117/1.2333510, URL : https://doi.org/10.1117/1.2333510.

[Lib] Commission Nationale de l'Informatique et des LIBERTÉSL, *Qu'est-ce ce qu'une donnée de santé ?*, URL : https://www.cnil.fr/fr/quest-ce-ce-quune-donnee-de-sante.

[Lin17] Yehuda LINDELL, « How to Simulate It – A Tutorial on the Simulation Proof Technique », in : *Tutorials on the Foundations of Cryptography : Dedicated to Oded Goldreich*, sous la dir. d'Yehuda LINDELL, Cham : Springer International Publishing, 2017, p. 277-346, ISBN : 978-3-319-57048-8, DOI : 10.1007/978-3-319-57048-8_6, URL : https://doi.org/10.1007/978-3-319-57048-8_6.

[Lit+17] Geert LITJENS et al., « A survey on deep learning in medical image analysis », in : *Medical Image Analysis* 42 (juill. 2017), p. 60-88, ISSN : 1361-8415, DOI : 10.1016/j.media.2017.07.005, URL : https://doi.org/10.1016/j.media.2017.07.005.

[LKS16] Wen-jie LU, Shohei KAWASAKI et Jun SAKUMA, *Using Fully Homomorphic Encryption for Statistical Analysis of Categorical, Ordinal and Numerical Data*, Cryptology ePrint Archive, Report 2016/1163, https://eprint.iacr.org/2016/1163, 2016.

[LPR10] Vadim LYUBASHEVSKY, Chris PEIKERT et Oded REGEV, « On Ideal Lattices and Learning with Errors over Rings », in : *Advances in Cryptology – EUROCRYPT 2010*, sous la dir. d'Henri GILBERT, Berlin, Heidelberg : Springer Berlin Heidelberg, avr. 2010, p. 1-23, ISBN : 978-3-642-13190-5, DOI : 10.1007/978-3-642-13190-5_1, URL : https://doi.org/10.1007/978-3-642-13190-5_1.

[LS05] Chung-Chih LI et Bo SUN, « Using Linear Congruential Generators for Cryptographic Purposes », in : *Computers and Their Applications*, mars 2005, p. 13-19, ISBN : 1-880843-54-4.

[LS07] Pierre L'Ecuyer et Richard Simard, « TestU01 : AC library for empirical testing of random number generators », in : *ACM Transactions on Mathematical Software (TOMS)* 33.*4* (août 2007), p. 1-40, issn : 0098-3500, doi : 10.1145/1268776.1268777, url : https://doi.org/10.1145/1268776.1268777.

[LT91] Pierre L'Ecuyer et Shu Tezuka, « Structural properties for two classes of combined random number generators », in : *Mathematics of Computation* 57.*196* (sept. 1991), p. 735-746, issn : 0025-5718, doi : 10.1090/S0025-5718-1991-1094954-3, url : https://doi.org/10.1090/S0025-5718-1991-1094954-3.

[Ma+18] Xiaojing Ma et al., « JPEG Decompression in the Homomorphic Encryption Domain », in : *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, Seoul, Republic of Korea : Association for Computing Machinery, oct. 2018, p. 905-913, isbn : 9781450356657, doi : 10.1145/3240508.3240672, url : https://doi.org/10.1145/3240508.3240672.

[Mar+01] Jacques Marescaux et al., « Transatlantic robot-assisted telesurgery », in : *Nature* 413.*6854* (sept. 2001), p. 379-380, issn : 1476-4687, doi : 10.1038/35096636, url : https://doi.org/10.1038/35096636.

[MBM03] Johan Montagnat, Vincent Breton et Isabelle Magnin, « Using grid technologies to face medical image analysis challenges », in : *Biogrid'03 workshop*, 2003, p. 588-593.

[MDG08] Mario Mustra, Kresimir Delac et Mislav Grgic, « Overview of the DICOM standard », in : *2008 50th International Symposium ELMAR*, t. 1, 2008, p. 39-44.

[MH+08] Junita Mohamad-Saleh, Brian S Hoyle et al., « Improved neural network performance using principal component analysis on Matlab », in : *International journal of the computer, the internet and Management* 16.*2* (2008), p. 1-8.

[Mic20] Microsoft, *Microsoft SEAL (release 3.5)*, Microsoft Research, Redmond, WA., avr. 2020, url : https://github.com/Microsoft/SEAL.

[MP11] Harry G. Mond et Alessandro Proclemer, « The 11th World Survey of Cardiac Pacing and Implantable Cardioverter-Defibrillators : Calendar Year 2009–A World Society of Arrhythmia's Project », in : *Pacing and Clinical Electrophysiology* 34.*8* (août 2011), p. 1013-1027, ISSN : 0147-8389, DOI : 10.1111/j.1540-8159.2011.03150.x, URL : https://doi.org/10.1111/j.1540-8159.2011.03150.x.

[MS13] Silvia Mella et Ruggero Susella, « On the Homomorphic Computation of Symmetric Cryptographic Primitives », in : *Cryptography and Coding*, sous la dir. de Martijn Stam, Berlin, Heidelberg : Springer Berlin Heidelberg, déc. 2013, p. 28-44, ISBN : 978-3-642-45239-0, DOI : 10.1007/978-3-642-45239-0_3, URL : https://doi.org/10.1007/978-3-642-45239-0_3.

[Nag+11] Jawad Nagi et al., « Max-pooling convolutional neural networks for vision-based hand gesture recognition », in : *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, 2011, p. 342-347, DOI : 10.1109/ICSIPA.2011.6144164, URL : https://doi.org/10.1109/ICSIPA.2011.6144164.

[Nag+18] Yuki Nagai et al., « Digital watermarking for deep neural networks », in : *International Journal of Multimedia Information Retrieval* 7.*1* (mars 2018), p. 3-16, ISSN : 2192-662X, DOI : 10.1007/s13735-018-0147-1, URL : https://doi.org/10.1007/s13735-018-0147-1.

[Nav+17] Pedro J. Navarro et al., « A Machine Learning Approach to Pedestrian Detection for Autonomous Vehicles Using High-Definition 3D Range Data », in : *Sensors* 17.*1* (déc. 2017), p. 1-20, ISSN : 1424-8220, DOI : 10.3390/s17010018, URL : https://doi.org/10.3390/s17010018.

[NC08] Amine Naït-Ali et Christine Cavaro-Ménard, *Compression of Biomedical Images and Signals*, John Wiley & Sons, Ltd, jan. 2008, p. 1-292, ISBN : 978-1-848-21028-8, DOI : 10.1002/9780470611159, URL : https://doi.org/10.1002/9780470611159.

[Niy+20] David Niyitegeka et al., « Secure Collapsing Method Based on Fully Homomorphic Encryption », in : *Studies in health technology and informatics* 270 (juin 2020), p. 412-416, ISSN : 0926-9630, DOI : 10.3233/shti200193, URL : https://doi.org/10.3233/shti200193.

[ODO20] Robin Ohannessian, Tu Anh Duong et Anna Odone, « Global Telemedicine Implementation and Integration Within Health Systems to Fight the COVID-19 Pandemic : A Call to Action », in : *JMIR Public Health Surveill* 6.*2* (avr. 2020), p. 1-4, issn : 2369-2960, doi : 10.2196/18810, url : https://doi.org/10.2196/18810.

[PA18] Laurie Pycroft et Tipu Z. Aziz, « Security of implantable medical devices with wireless connections : The dangers of cyber-attacks », in : *Expert Review of Medical Devices* 15.*6* (juin 2018), p. 403-406, issn : 1743-4440, doi : 10.1080/17434440.2018.1483235, url : https://doi.org/10.1080/17434440.2018.1483235.

[Pai99] Pascal Paillier, « Public-Key Cryptosystems Based on Composite Degree Residuosity Classes », in : *Advances in Cryptology — EUROCRYPT '99*, sous la dir. de Jacques Stern, Berlin, Heidelberg : Springer Berlin Heidelberg, avr. 1999, p. 223-238, isbn : 978-3-540-48910-8, doi : 10.1007/3-540-48910-X_16, url : https://doi.org/10.1007/3-540-48910-X_16.

[PBC19] Maxime Pistono, Reda Bellafqira et Gouenou Coatrieux, « Secure Processing of Stream Cipher Encrypted Data Issued from IOT : Application to a Connected Knee Prosthesis », in : *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, juill. 2019, p. 6494-6497, doi : 10.1109/EMBC.2019.8857055, url : https://doi.org/10.1109/EMBC.2019.8857055.

[PBC21] Maxime Pistono, Reda Bellafqira et Gouenou Coatrieux, « Cryptosystem Conversion, Packing and Matrix Processing of Homomorphically Encrypted Data : Application to IOT Devices », in : *IEEE Access* 9 (2021), p. 28302-28316, issn : 2169-3536, doi : 10.1109/ACCESS.2021.3058849, url : https://doi.org/10.1109/ACCESS.2021.3058849.

[PHC05a] V. M. Potdar, S. Han et E. Chang, « A survey of digital image watermarking techniques », in : *INDIN '05. 2005 3rd IEEE International Conference on Industrial Informatics, 2005.* INDIN '05. 2005 3rd IEEE International Conference on Industrial Informatics, 2005. Août 2005, p. 709-716, doi : 10.1109/INDIN.2005.1560462, url : https://doi.org/10.1109/INDIN.2005.1560462.

[PHC05b] Vidyasagar M. Potdar, Song Han et Elizabeth Chang, « A survey of digital image watermarking techniques », in : *INDIN '05. 2005 3rd IEEE International Conference on Industrial Informatics*, août 2005, p. 709-716, doi : 10.1109/INDIN.2005.1560462, url : https://doi.org/10.1109/INDIN.2005.1560462.

[Pho+18] Le Trieu Phong et al., « Privacy-Preserving Deep Learning via Additively Homomorphic Encryption », in : *IEEE Transactions on Information Forensics and Security* 13.*5* (mai 2018), p. 1333-1345, issn : 1556-6021, doi : 10.1109/TIFS.2017.2787987, url : https://doi.org/10.1109/TIFS.2017.2787987.

[Pis+20] Maxime Pistono et al., « Training Machine Learning on JPEG Compressed Images », in : *2020 Data Compression Conference (DCC)*, 2020, p. 388-388, doi : 10.1109/DCC47342.2020.00070, url : https://doi.org/10.1109/DCC47342.2020.00070.

[Pop+11] Raluca Ada Popa et al., « CryptDB : Protecting Confidentiality with Encrypted Query Processing », in : *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, Cascais, Portugal : Association for Computing Machinery, oct. 2011, p. 85-100, isbn : 978-1-450-30977-6, doi : 10.1145/2043556.2043566, url : https://doi.org/10.1145/2043556.2043566.

[PSD06] Francis Puisieux, Monique Seiller et Jean Philippe Devissaguet, « Les systèmes de délivrance des médicaments : un réel progress pour la thérapeutique* », in : *Annales Pharmaceutiques Françaises* 64.*4* (juill. 2006), p. 219-259, issn : 0003-4509, doi : 10.1016/S0003-4509(06)75318-4, url : https://doi.org/10.1016/S0003-4509(06)75318-4.

[Rab80] Michael O. Rabin, « Probabilistic algorithm for testing primality », in : *Journal of Number Theory* 12.*1* (fév. 1980), p. 128-138, issn : 0022-314X, doi : 10.1016/0022-314X(80)90084-0, url : https://doi.org/10.1016/0022-314X(80)90084-0.

[Rab81] Michael O. Rabin, *How To Exchange Secrets with Oblivious Transfer*, rapp. tech., Harvard University, 1981.

[Rat+17] Heena Rathore et al., « A review of security challenges, attacks and resolutions for wireless medical devices », in : *13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, juin 2017, p. 1495-1501, DOI : 10.1109/IWCMC.2017.7986505, URL : https://doi.org/10.1109/IWCMC.2017.7986505.

[RC19] Acumen Research et Consulting, *The global implantable medical devices market is expected to grow at a CAGR of around 7.3% over the forecast period 2019 to 2026 and expected to reach the market value of around US$ 153.8 Bn by 2026*, rapp. tech., 2019, p. 1-190, URL : https://www.acumenresearchandconsulting.com/implantable-medical-devices-market.

[RD92] Ronald Rivest et S Dusse, *The MD5 message-digest algorithm*, 1992.

[RDB96] J.J.K. Ó Ruanaidh, W.J. Dowling et F.M. Boland, « Watermarking digital images for copyright protection », in : *IEE Proceedings - Vision, Image and Signal Processing* 143 (4 août 1996), p. 250-256, ISSN : 1350-245X, URL : https://digital-library.theiet.org/content/journals/10.1049/ip-vis_19960711.

[Rob08] Matthew Robshaw, « The eSTREAM Project », in : *New Stream Cipher Designs : The eSTREAM Finalists*, sous la dir. de Matthew Robshaw et Olivier Billet, Berlin, Heidelberg : Springer Berlin Heidelberg, 2008, p. 1-6, ISBN : 978-3-540-68351-3, DOI : 10.1007/978-3-540-68351-3_1, URL : https://doi.org/10.1007/978-3-540-68351-3_1.

[Ros+20] Ron Ross et al., « Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations », in : *NIST Special Publication 800-171 Revision 2*, fév. 2020, p. 1-101, DOI : 10.6028/NIST.SP.800-171r2, URL : https://doi.org/10.1109/EMBC.2019.8857055.

[RSA78] Ronald. L. Rivest, Adi Shamir et Leonard Adleman, « A Method for Obtaining Digital Signatures and Public-Key Cryptosystems », in : *Commun. ACM* 21.*2* (fév. 1978), p. 120-126, ISSN : 0001-0782, DOI : 10.1145/359340.359342, URL : https://doi.org/10.1145/359340.359342.

[Rus+14] Michael Rushanan et al., « SoK : Security and Privacy in Implantable Medical Devices and Body Area Networks », in : *2014 IEEE Symposium on Security and Privacy*, mai 2014, p. 524-539, DOI : 10.1109/SP.2014.40, URL : https://doi.org/10.1109/SP.2014.40.

[Sal+12] Muhammad T. Salam et al., « An Implantable Closedloop Asynchronous Drug Delivery System for the Treatment of Refractory Epilepsy », in : *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 20.*4* (juill. 2012), p. 432-442, ISSN : 1558-0210, DOI : 10.1109/TNSRE.2012.2189020, URL : https://doi.org/10.1109/TNSRE.2012.2189020.

[Sch+12] Roland Schmitz et al., « A New Approach to Commutative Watermarking-Encryption », in : *Communications and Multimedia Security*, sous la dir. de Bart De Decker et David W. Chadwick, Berlin, Heidelberg : Springer Berlin Heidelberg, sept. 2012, p. 117-130, ISBN : 978-3-642-32805-3, DOI : 10.1007/978-3-642-32805-3_10, URL : https://doi.org/10.1007/978-3-642-32805-3_10.

[Sch+14] Marin L. Schweizer et al., « Costs Associated With Surgical Site Infections in Veterans Affairs Hospitals », in : *JAMA Surgery* 149.*6* (juin 2014), p. 575-581, ISSN : 2168-6254, DOI : 10.1001/jamasurg.2013.4663, URL : https://doi.org/10.1001/jamasurg.2013.4663.

[See+98] Timothy M. Seese et al., « Characterization of tissue morphology, angiogenesis, and temperature in the adaptive response of muscle tissue to chronic heating », in : *Laboratory investigation ; a journal of technical methods and pathology* 78.*12* (déc. 1998), p. 1553-1562, ISSN : 0023-6837, URL : http://europepmc.org/abstract/MED/9881955.

[SL91] S.R. Safavian et D. Landgrebe, « A survey of decision tree classifier methodology », in : *IEEE Transactions on Systems, Man, and Cybernetics* 21.*3* (1991), p. 660-674, ISSN : 2168-2909, DOI : 10.1109/21.97458, URL : https://doi.org/10.1109/21.97458.

[SMB10] Dominik Scherer, Andreas Müller et Sven Behnke, « Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition », in : *Artificial Neural Networks – ICANN 2010*, sous la dir. de Konstantinos Diamantaras, Wlodek Duch et Lazaros S. Iliadis, Berlin, Heidelberg : Springer Berlin Heidelberg, 2010, p. 92-101, ISBN : 978-3-642-15825-4, DOI :

10.1007/978-3-642-15825-4_10, URL : https://doi.org/10.1007/978-3-642-15825-4_10.

[SSP03] Patrice. Y. SIMARD, Dave STEINKRAUS et John C. PLATT, « Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis », in : *7th International Conference on Document Analysis and Recognition*, t. 3, Los Alamitos, CA, USA : IEEE Computer Society, août 2003, p. 1-6, DOI : 10.1109/ICDAR.2003.1227801, URL : https://doi.org/10.1109/ICDAR.2003.1227801.

[Sum20] Rupali Swain SUMANT UGALMUGLE, *Implantable Medical Devices Market Size By Product (Cardiovascular Implantable Cardiac Defibrillators [ICDs], Stents, Pacemaker, Implantable Cardiac Monitors [ICM], Ventricular Assist Devices, Cardiac Resynchronization Therapy Devices [CRT], Orthopedic Joint Reconstruction, Spinal Devices, Trauma Fixation Devices, Ophthalmology Intraocular Lenses and Glaucoma Implants, Plastic Surgery Breast Implants, Gluteal Implants, Neurology Deep Brain Stimulators, Dental Dental Implants, Dental Crowns and Abutment), By Type (Diagnostic, Treatment), By Nature of Device (Active, Static/Non-active/Passive), By End-use (Hospitals, Multi-specialty Centres, Ambulatory Surgical Centers, Clinics), Industry Analysis Report, Regional Outlook, Application Potential, Competitive Market Share & Forecast, 2019 - 2025*, rapp. tech., jan. 2020, p. 1-250, URL : https://www.gminsights.com/industry-analysis/implantable-medical-devices-market.

[SV14] Nigel P. SMART et Frederik VERCAUTEREN, « Fully Homomorphic SIMD Operations », in : *Designs, codes and cryptography* 71.*1* (avr. 2014), p. 57-81, ISSN : 1573-7586, DOI : 10.1007/s10623-012-9720-4, URL : https://doi.org/10.1007/s10623-012-9720-4.

[Tat16] Deepa TATKARE, *Implantable Medical Devices Market by Product (Orthopedic Implants, Dental Implants, Breast Implants, Cardiovascular Implants, Intraocular Lens, and Other Implants) - Global Opportunity Analysis and Industry Forecast, 2014-2022*, rapp. tech., déc. 2016, p. 1-208, URL : https://www.alliedmarketresearch.com/implantable-medical-devices-market.

[TBK20] Anselme Tueno, Yordan Boev et Florian Kerschbaum, « Non-interactive Private Decision Tree Evaluation », in : *Data and Applications Security and Privacy XXXIV*, sous la dir. d'Anoop Singhal et Jaideep Vaidya, Cham : Springer International Publishing, juin 2020, p. 174-194, isbn : 978-3-030-49669-2, doi : 10.1007/978-3-030-49669-2_10, url : https://doi.org/10.1007/978-3-030-49669-2_10.

[TEE12] Maha Tebaa, Saïd El Hajji et Abdellatif El Ghazi, « Homomorphic encryption applied to the cloud computing security », in : *2012 Proceedings of the World Congress on Engineering*, juin 2012, p. 86-89, isbn : 978-1-4673-1053-6, doi : 10.1109/JNS2.2012.6249248, url : https://doi.org/10.1109/JNS2.2012.6249248.

[TF19] Abhishek Tiwari et Tiago H. Falk, « Lossless electrocardiogram signal compression : A review of existing methods », in : *Biomedical Signal Processing and Control* 51 (mai 2019), p. 338-346, issn : 1746-8094, doi : 10.1016/j.bspc.2019.03.004, url : https://doi.org/10.1016/j.bspc.2019.03.004.

[TH12] Tijmen Tieleman et Geoffrey Hinton, « Lecture 6.5-rmsprop : Divide the gradient by a running average of its recent magnitude », in : *COURSERA : Neural networks for machine learning* 4.*2* (2012), p. 26-31.

[Thi+14] Danan Thilakanathan et al., « A platform for secure monitoring and sharing of generic health data in the Cloud », in : *Future Generation Computer Systems* 35 (juin 2014), Special Section : Integration of Cloud Computing and Body Sensor Networks ; Guest Editors : Giancarlo Fortino and Mukaddim Pathan, p. 102-113, issn : 0167-739X, doi : 10.1016/j.future.2013.09.011, url : https://doi.org/10.1016/j.future.2013.09.011.

[TM02] David Taubman et Michael Marcellin, *JPEG2000 Image Compression Fundamentals, Standards and Practice*, t. 642, Springer US, 2002, p. 1-777, isbn : 978-1-4615-0799-4, doi : 10.1007/978-1-4615-0799-4, url : https://doi.org/10.1007/978-1-4615-0799-4.

[UD17] Matej Ulicny et Rozenn Dahyot, « On using cnn with dct based image data », in : *Proceedings of the 19th Irish Machine Vision and Image Processing conference IMVIP*, t. 2, 2017, p. 1-8.

[VB17] Paul Voigt et Axel von dem Bussche, *The EU General Data Protection Regulation (GDPR) : A Practical Guide*, Springer Publishing Company, Incorporated, 2017, p. 1-383, isbn : 978-3-319-57959-7, doi : 10.1007/978-3-319-57959-7, url : https://doi.org/10.1007/978-3-319-57959-7.

[Ver+20] Joost Verbraeken et al., « A Survey on Distributed Machine Learning », in : *ACM Comput. Surv.* 53.*2* (mars 2020), issn : 0360-0300, doi : 10.1145/3377454, url : https://doi.org/10.1145/3377454.

[VI91] Kelly G. Vince et John N. Insall, « The Total Condylar Knee Prosthesis », in : *Total Knee Replacement*, sous la dir. de Richard S. Laskin, London : Springer London, 1991, p. 85-111, isbn : 978-1-4471-1825-1, doi : 10.1007/978-1-4471-1825-1_7, url : https://doi.org/10.1007/978-1-4471-1825-1_7.

[Wal92] Gregory K. Wallace, « The JPEG still picture compression standard », in : *IEEE Transactions on Consumer Electronics* 38.*1* (fév. 1992), p. xviii-xxxiv, issn : 1558-4127, doi : 10.1109/30.125072, url : https://doi.org/10.1109/30.125072.

[WAS03] Jill Weissberg-Benchell, Jeanne Antisdel-Lomaglio et Roopa Seshadri, « Insulin Pump Therapy », in : *Diabetes Care* 26.*4* (avr. 2003), p. 1079-1087, issn : 0149-5992, doi : 10.2337/diacare.26.4.1079, url : https://doi.org/10.2337/diacare.26.4.1079.

[WEG87] Svante Wold, Kim Esbensen et Paul Geladi, « Principal component analysis », in : *Chemometrics and Intelligent Laboratory Systems* 2.*1* (1987), p. 37-52, issn : 0169-7439, doi : 10.1016/0169-7439(87)80084-9, url : https://doi.org/10.1016/0169-7439(87)80084-9.

[WH82] Brian A. Wichmann et I. David Hill, « Algorithm AS 183 : An Efficient and Portable Pseudo-Random Number Generator », in : *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 31.*2* (1982), p. 188-190, issn : 00359254, 14679876, doi : 10.2307/2347988, url : https://doi.org/10.2307/2347988.

[Wol08] Patrick D. Wolf, « Thermal Considerations for the Design of an Implanted Cortical Brain–Machine Interface (BMI) », in : *Indwelling Neural Implants : Strategies for Contending with the in Vivo Environment*, Frontiers in Neuroengineering, CRC Press/Taylor &;; Francis, Boca Raton (FL), jan. 2008,

p. 63-86, ISBN : 9780849393624, DOI : 10.1201/9781420009309.ch3, URL : https://doi.org/10.1201/9781420009309.ch3.

[Won+16] Sebastien C. WONG et al., « Understanding Data Augmentation for Classification : When to Warp ? », in : *2016 International Conference on Digital Image Computing : Techniques and Applications (DICTA)*, 2016 International Conference on Digital Image Computing : Techniques and Applications (DICTA), nov. 2016, p. 1-6, ISBN : 978-1-5090-2897-9, DOI : 10.1109/DICTA.2016.7797091, URL : https://doi.org/10.1109/DICTA.2016.7797091.

[Wu+15] Zhizheng WU et al., « Deep neural networks employing Multi-Task Learning and stacked bottleneck features for speech synthesis », in : *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, p. 4460-4464, DOI : 10.1109/ICASSP.2015.7178814, URL : https://doi.org/10.1109/ICASSP.2015.7178814.

[XC14] Di XIAO et Shoukuo CHEN, « Separable data hiding in encrypted image based on compressive sensing », in : *Electronics Letters* 50.*8* (avr. 2014), p. 598-600, ISSN : 0013-5194, DOI : 10.1049/el.2013.3806, URL : https://doi.org/10.1049/el.2013.3806.

[Xu+11] Fengyuan XU et al., « IMDGuard : Securing implantable medical devices with the external wearable guardian », in : *2011 Proceedings IEEE INFOCOM*, avr. 2011, p. 1862-1870, DOI : 10.1109/INFCOM.2011.5934987, URL : https://doi.org/10.1109/INFCOM.2011.5934987.

[Yao82] Andrew C. YAO, « Protocols for secure computations », in : *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, nov. 1982, p. 160-164, DOI : 10.1109/SFCS.1982.38, URL : https://doi.org/10.1109/SFCS.1982.38.

[Yao86] Andrew Chi-Chih YAO, « How to generate and exchange secrets », in : *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, 1986, p. 162-167, DOI : 10.1109/SFCS.1986.25, URL : https://doi.org/10.1109/SFCS.1986.25.

[ZCB16] Rita ZGHEIB, Emmanuel CONCHON et Rémi BASTIDE, « Engineering IoT Healthcare Applications : Towards a Semantic Data Driven Sustainable Architecture », in : *eHealth 360°*, sous la dir. de Kostas GIOKAS, Laszlo BOKOR

et Frank Hopfgartner, Cham : Springer International Publishing, déc. 2016, p. 407-418, isbn : 978-3-319-49655-9, doi : 10.1007/978-3-319-49655-9_49, url : https://doi.org/10.1007/978-3-319-49655-9_49.

DOCTORAT BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Traitements sécurisés de données compressées – Application à l'apprentissage automatique et aux implants connectés

Mot clés : Chiffrement homomorphe, *machine learning*, compression

Résumé : Aujourd'hui les données sont de plus en plus externalisées et réutilisées que ce soit dans la santé ou de façon plus générale. L'intérêt de ces techniques est de réduire les coûts associés aux infrastructures informatiques, mais également de permettre un accès simplifié aux données tout en laissant la possibilité de les traiter grâce à des algorithmes de *Big data*. En effet, des algorithmes de *machine learning* appliqués sur ces données permettent d'aider le professionnel dans sa prise de décision. Il est néanmoins primordial de rappeler que les données personnelles donc en particulier celle appartenant au secteur de la santé sont des données sensibles. En conséquence de quoi, des réglementations nationales ou internationales imposent leur sécurisation. Cela passe par le fait d'assurer leur confidentialité au travers d'algorithmes de chiffrement ainsi que leur intégrité et traçabilité grâce au tatouage numérique. D'autre part, il est nécessaire de comprendre que les volumes de données utilisées sont très important. Ceci impose que les données et en particulier celle issues de l'imagerie sont compressées afin de réduire les coûts de stockage. C'est pourquoi dans cette thèse, nous nous intéresserons aux trois problématiques associées aux données que sont leur sécurisation, traitement et compression. L'objectif sera alors d'associer ces trois propriétés afin de fournir une solution globale permettant de traiter de façon sécurisée des données compressées.

Au cours de ces trois années de thèse, nous avons commencé par nous intéresser aux algorithmes de sécurisation de données qui permettent de les traiter. Ces algorithmes plus connus sous le nom de cryptosystèmes homomorphes existent sous plusieurs formes. Le chiffrement additivement homomorphe permet un nombre limité de traitement, mais est relativement peu coûteuse. Le *fully* homomorphe augmente les possibilités de traitements au détriment des performances. Notre première solution propose d'allier un cryptosystème additivement homomorphe avec un système de concaténation qui permet une compression dans le domaine chiffré. Elle permet d'effectuer un traitement matriciel sécurisé des données. Cette solution s'inscrit dans le projet *Followknee* dans lequel l'objectif est de traiter de façon sécurisée des données issues d'une prothèse connectée sur un smartphone. Par la suite, nous avons entrepris d'observer l'influence de la compression sur la précision des modèles de *machine learning*. L'objectif de cette partie étant de voir s'il est possible de ne décompresser que partiellement des données et ainsi gagner en termes de coûts calculatoires avant de les utiliser dans de tels modèles. Suite à quoi, nous avons essayé de décompresser partiellement et de façon sécurisée des données grâce à des cryptosystèmes *fully* homomorphes. Ceci représente la première étape d'une chaîne de traitement plus complexe permettant de compresser des données puis de les chiffrer avant de les externaliser vers un *cloud*. Une fois externalisées, ces données pourront être partiellement décompressées afin d'être utilisées de façon sécurisée au travers d'algorithmes de traitement. Pour finir, nous avons étudié la possibilité d'utiliser un cryptosystème additivement homomorphe en combinaison de calcul multipartite afin de sécuriser l'apprentissage d'un réseau de neurones artificiels.

Title: Secure compressed data processing - Application to machine learning and connected implants

Keywords: Homomorphic encryption, machine learning, compression

Abstract: Nowadays, medical data and data in general are more and more outsourced to the cloud. The purpose of outsourcing is to reduce the costs associated with operations of an IT structure. Moreover, it allows reusing data and big data analysis. More precisely, different professional can access data remotely, which simplifies information sharing. Furthermore, machine learning algorithms can process outsourced data in order to help professionals in their decision-making. However, these data are personal and sensitive, which is particularly true for medical one. According to this fact, data have to be acquired, transmitted and processed in a secure way. To do so, national and international laws regulate their uses. Hence, data confidentiality, integrity and traceability have to be ensured using encryption and watermarking. In addition, the processed data size is significant. This is why they are generally compressed in order to reduce storage and communication cost. This is especially true regarding medical images. That is the reason why through this thesis we were interested in data : security, compression and processing. The objective is to associate these three properties in a method allowing to securely process compressed data.

During these three years of thesis, we started by focusing on algorithms that allow to securely process data. These algorithms are known as homomorphic encryption. For example, additively homomorphic cryptosystems only allow to secure the addition operation, while fully homomorphic cryptosystems allow to securely evaluate bots addition and multiplication. However, fully homomorphic cryptosystems are computationally more expensive than additively homomorphic cryptosystems. Our first solution jointly uses an additively homomorphic cryptosystem, a lightweight cryptosystem and a compression into the encrypted domain. This solution allows to securely process matrix data. It was developed for the Followknee project, whose objective is to securely process data issued from a prosthesis on a smartphone. Next, we were interested in analyzing the influence of feeding machine learning model with compressed or partially compressed data. The objective was to know if it is possible or not to only partially decompress data and thus reduce calculation costs before feeding machine learning model. Thereafter, we try to securely decompress data using fully homomorphic cryptosystems. This constitutes the first step of a more complex processing chain. This chain aims to compress data before outsourcing them to a cloud. Once data are on the cloud, they can be partially decompressed in order to feed machine learning algorithm. Finally, we proposed a way to combine multi-party computation with additively homomorphic cryptosystems aiming to secure the artificial neural network training and inference.