Déploiement de la cryptographie homomorphe dans le cadre de l'iot

Amina Bel Korchi

N° d'ordre NNT : 2019LYSEM027

THESE de DOCTORAT DE L'UNIVERSITE DE LYON

opérée au sein de

MINES Saint-Etienne

Ecole Doctorale N° 488

Sciences, Ingénierie, Santé

Spécialité de doctorat : Microélectronique

Soutenue publiquement le 30/10/2019, par :

Amina BEL KORCHI

Déploiement de la cryptographie homomorphe dans le cadre de l'IoT

Directrice de thèse : **Nadia EL MRABET**
Encadrant de thèse : **Serge TISSOT**

Devant le jury composé de :

Mme Nora Cuppens,	Directrice de Recherche	Rapporteur
M Pascal Urien,	Professeur	Rapporteur
M Sylvain Guilley,	Professeur	Examinateur
Mme Nadia El Mrabet,	Maître de conférence	Directrice
M Serge Tissot,	Ingénieur	Encadrant

Spécialités doctorales	Responsables :	Spécialités doctorales	Responsables
SCIENCES ET GENIE DES MATERIAUX	K. Wolski Directeur de recherche	MATHEMATIQUES APPLIQUEES	O. Roustant, Maître-assistant
MECANIQUE ET INGENIERIE	S. Drapier, professeur	INFORMATIQUE	O. Boissier, Professeur
GENIE DES PROCEDES	F. Gruy, Maître de recherche	SCIENCES DES IMAGES ET DES FORMES	JC. Pinoli, Professeur
SCIENCES DE LA TERRE	B. Guy, Directeur de recherche	GENIE INDUSTRIEL	N. Absi, Maitre de recherche
SCIENCES ET GENIE DE L'ENVIRONNEMENT	D. Graillot, Directeur de recherche	MICROELECTRONIQUE	Ph. Lalevée, Professeur

EMSE : Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat (titulaires d'un doctorat d'État ou d'une HDR)

ABSI	Nabil	MR	Génie industriel	CMP
AUGUSTO	Vincent	CR	Image, Vision, Signal	CIS
AVRIL	Stéphane	PR2	Mécanique et ingénierie	CIS
BADEL	Pierre	MA(MDC)	Mécanique et ingénierie	CIS
BALBO	Flavien	PR2	Informatique	FAYOL
BASSEREAU	Jean-François	PR	Sciences et génie des matériaux	SMS
BATTON-HUBERT	Mireille	PR2	Sciences et génie de l'environnement	FAYOL
BEIGBEDER	Michel	MA(MDC)	Informatique	FAYOL
BLAYAC	Sylvain	MA(MDC)	Microélectronique	CMP
BOISSIER	Olivier	PR1	Informatique	FAYOL
BONNEFOY	Olivier	PR	Génie des Procédés	SPIN
BORBELY	Andras	MR(DR2)	Sciences et génie des matériaux	SMS
BOUCHER	Xavier	PR2	Génie Industriel	FAYOL
BRODHAG	Christian	DR	Sciences et génie de l'environnement	FAYOL
BRUCHON	Julien	MA(MDC)	Mécanique et ingénierie	SMS
CAMEIRAO	Ana	MA(MDC)	Génie des Procédés	SPIN
CHRISTIEN	Frédéric	PR	Science et génie des matériaux	SMS
DAUZERE-PERES	Stéphane	PR1	Génie Industriel	CMP
DEBAYLE	Johan	MR	Sciences des Images et des Formes	SPIN
DEGEORGE	Jean-Michel	MA(MDC)	Génie industriel	Fayol
DELAFOSSE	David	PR0	Sciences et génie des matériaux	SMS
DELORME	Xavier	MA(MDC)	Génie industriel	FAYOL
DESRAYAUD	Christophe	PR1	Mécanique et ingénierie	SMS
DJENIZIAN	Thierry	PR	Science et génie des matériaux	CMP
BERGER-DOUCE	Sandrine	PR1	Sciences de gestion	FAYOL
DRAPIER	Sylvain	PR1	Mécanique et ingénierie	SMS
DUTERTRE	Jean-Max	MA(MDC)		CMP
EL MRABET	Nadia	MA(MDC)		CMP
FAUCHEU	Jenny	MA(MDC)	Sciences et génie des matériaux	SMS
FAVERGEON	Loïc	CR	Génie des Procédés	SPIN
FEILLET	Dominique	PR1	Génie Industriel	CMP
FOREST	Valérie	MA(MDC)	Génie des Procédés	CIS
FRACZKIEWICZ	Anna	DR	Sciences et génie des matériaux	SMS
GARCIA	Daniel	MR(DR2)	Sciences de la Terre	SPIN
GAVET	Yann	MA(MDC)	Sciences des Images et des Formes	SPIN
GERINGER	Jean	MA(MDC)	Sciences et génie des matériaux	CIS
GOEURIOT	Dominique	DR	Sciences et génie des matériaux	SMS
GONDRAN	Natacha	MA(MDC)	Sciences et génie de l'environnement	FAYOL
GONZALEZ FELIU	Jesus	MA(MDC)	Sciences économiques	FAYOL
GRAILLOT	Didier	DR	Sciences et génie de l'environnement	SPIN
GROSSEAU	Philippe	DR	Génie des Procédés	SPIN
GRUY	Frédéric	PR1	Génie des Procédés	SPIN
HAN	Woo-Suck	MR	Mécanique et ingénierie	SMS
HERRI	Jean Michel	PR1	Génie des Procédés	SPIN
KERMOUCHE	Guillaume	PR2	Mécanique et Ingénierie	SMS
KLOCKER	Helmut	DR	Sciences et génie des matériaux	SMS
LAFOREST	Valérie	MR(DR2)	Sciences et génie de l'environnement	FAYOL
LERICHE	Rodolphe	CR	Mécanique et ingénierie	FAYOL
MALLIARAS	Georges	PR1	Microélectronique	CMP
MOLIMARD	Jérôme	PR2	Mécanique et ingénierie	CIS
MOUTTE	Jacques	CR	Génie des Procédés	SPIN
NAVARRO	Laurent	CR		CIS
NEUBERT	Gilles			FAYOL
NIKOLOVSKI	Jean-Pierre	Ingénieur de recherche	Mécanique et ingénierie	CMP
NORTIER	Patrice	PR1	Génie des Procédés	SPIN
O CONNOR	Rodney Philip	MA(MDC)	Microélectronique	CMP
PICARD	Gauthier	MA(MDC)	Informatique	FAYOL
PINOLI	Jean Charles	PR0	Sciences des Images et des Formes	SPIN
POURCHEZ	Jérémy	MR	Génie des Procédés	CIS
ROUSSY	Agnès	MA(MDC)	Microélectronique	CMP
ROUSTANT	Olivier	MA(MDC)	Mathématiques appliquées	FAYOL
SANAUR	Sébastien	MA(MDC)	Microélectronique	CMP
SERRIS	Eric	IRD		FAYOL
STOLARZ	Jacques	CR	Sciences et génie des matériaux	SMS
TRIA	Assia	Ingénieur de recherche	Microélectronique	CMP
VALDIVIESO	François	PR2	Sciences et génie des matériaux	SMS
VIRICELLE	Jean Paul	DR	Génie des Procédés	SPIN
WOLSKI	Krzystof	DR	Sciences et génie des matériaux	SMS
XIE	Xiaolan	PR0	Génie industriel	CIS
YUGMA	Gallian	CR	Génie industriel	CMP

Mise à jour : 30/08/2018

École Nationale Supérieure des Mines de Saint-Étienne

NNT : 2019LYSEM027

Amina BEL KORCHI

DEPLOYEMENT OF HOMOMORPHIC ENCRYPTION IN THE CONTEXT OF IOT

Speciality : Microelectronic

Keywords : Homomorphic encryption, BGV, FV, Comparaison, IoT, cloud, security, anonymity

Abstract :
The emerging technology of mobile devices allows users to access to a wide range of applications through the internet connection. As these applications require considerable computing power, they represent a challenge for devices with limited computing power, memory storage and energy. However, such a challenge could be avoided by cloud computing as it offers virtually unlimited dynamic resources for the processing and the storage of data. Nevertheless, mobile users are still reluctant to adopt this technology because traditional encryption schemes require the decryption of the data so that they can be processed. Homomorphic cryptography is a potential solution to allow arbitrary calculation of encrypted data without the need to decrypt. In practice, the cloud can calculate the sum and/or the product of encrypted data. The result is sent to the sender who can decrypt with his secret key. Although homomorphic cryptography is considered as a solution to allow secure calculations, its efficiency remains an obstacle to its implementation. The disadvantage of homomorphic encryption schemes is the size of ciphertexts. The objective of this thesis is to use an homomorphic encryption scheme in a an industrial use case related to IoT, in order to achieve an efficient and optimal implementation of an homomorphic protocol.

École Nationale Supérieure des Mines de Saint-Étienne

NNT : 2019LYSEM027

Amina BEL KORCHI

DEPLOIEMENT DE LA CRYPTOGRAPHIE HOMOMORPHE DANS LE CADRE DE L'IOT

Specialité : Microélectronique

Mots clefs : Cryptographie homomorphe, BGV, FV, Comparaison, IoT, cloud, sécurité, anonymisation

Résumé :
La technologie émergente des appareils mobiles permet aux utilisateurs d'accéder à un large éventail d'applications grâce à la connexion internet. Comme ces applications exigent une puissance de calcul considérable, elles représentent un défi pour les appareil dont la puissance de calcul, la mémoire, le stockage et l'énergie sont limités. Cependant, un tel défi pourrait être surmonté par le cloud computing car celui-ci offre des ressources dynamiques pratiquement illimitées pour le traitement et le stockage des données.
Néanmoins, les utilisateurs mobiles hésitent encore à adopter cette technologie car les schémas de chiffrement classiques exigent le déchiffrement des données pour pouvoir être traités. La cryptographie homomorphe est une solution potentielle pour permettre un calcul arbitraire des données chiffrées sans avoir à les déchiffrer. En pratique, le cloud peut calculer la somme et/ou le produit des textes chiffrés. Le résultat est envoyé à l'émetteur qui peut déchiffrer avec sa clé secrète. Bien que la cryptographie homomorphe soit considérée comme une solution pour permettre d'effectuer des calculs sécurisés, son efficacité reste un obstacle à sa mise en œuvre. L'inconvénient des schémas de chiffrement homomorphe est la taille des textes chiffrés. L'objectif de cette thèse est d'appliquer un schéma de chiffrement homomorphe dans un cas d'usage industriel lié à l'IoT, afin de réaliser une implémentation efficace et optimale d'un protocole homomorphe.

Table des matières

Résumé de la thèse

La technologie émergente des appareils mobiles permet aux utilisateurs d'accéder à un large éventail d'applications grâce à la connexion internet. Comme ces applications exigent une puissance de calcul considérable, elles représentent un défi pour les appareil dont la puissance de calcul, la mémoire, le stockage et l'énergie sont limités. Cependant, un tel défi pourrait être surmonté par le cloud computing car celui-ci offre des ressources dynamiques pratiquement illimitées pour le traitement et le stockage des données.
Néanmoins, les utilisateurs mobiles hésitent encore à adopter cette technologie car les schémas de chiffrement classiques exigent le déchiffrement des données pour pouvoir être traités. La cryptographie homomorphe est une solution potentielle pour permettre un calcul arbitraire des données chiffrées sans avoir à les déchiffrer. En pratique, le cloud peut calculer la somme et/ou le produit des textes chiffrés. Le résultat est envoyé à l'émetteur qui peut déchiffrer avec sa clé secrète. Bien que la cryptographie homomorphe soit considérée comme une solution pour permettre d'effectuer des calculs sécurisés, son efficacité reste un obstacle à sa mise en œuvre. L'inconvénient des schémas de chiffrement homomorphe est la taille des textes chiffrés. L'objectif de cette thèse est d'appliquer un schéma de chiffrement homomorphe dans un cas d'usage industriel lié à l'IoT, afin de réaliser une implémentation efficace et optimale d'un protocole homomorphe.

Remerciements

Dans le cadre de cette thèse, je tiens à remercier toutes les personnes qui m'ont aidée, encouragée, conseillée et soutenue.

Tout d'abord, je tiens à remercier les rapporteurs Nora Cuppens et Pascal Urien d'avoir relu ce manuscrit ainsi que pour leurs commentaires qui m'ont aidée à l'améliorer. Je remercie également Sylvain Guilley pour sa disponibilité et pour avoir accepté d'être président du jury.

Un très grand merci va en premier à ma directrice de thèse Nadia El Mrabet, pour son encadrement académique, sa présence permanente, sa bonne écoute, son orientation réfléchie et ses judicieux conseils, qui ont contribué à alimenter ma réflexion et au bon cheminement de ce rapport.

Je me dois aussi de remercier mon encadrant industriel de thèse Serge Tissot, pour l'intérêt qu'il a porté à ma recherche, pour le partage de son expertise au quotidien et pour sa confiance grâce à laquelle j'ai pu m'accomplir totalement dans mes missions.

La plus grande reconnaissance à mes très chers parents et mes frères qui ont œuvré pour ma réussite, de par leur amour et tendresse, leur soutien, tous les sacrifices consentis et leur précieux conseils tout au long de mon cursus. Aucun mot ne pourrait exprimer à sa juste valeur la gratitude et l'amour que je leur porte.

Ma sincère gratitude et reconnaissance à mes meilleures amies Salma, Meryem et Meriem pour le support moral et intellectuel qu'elles m'ont apporté durant toutes ces années. Qu'elles trouvent dans ce travail un modeste témoignage de mon affection la plus sincère.

je tiens également à remercier François et Sonia, mes collègues de travail, pour ces trois belles années qu'on a passé ensemble et pour les beaux souvenirs qu'on a pu créer.

Notations

Dans ce manuscrit, nous adopterons les notations suivantes :

o : Loi de composition interne.

$\circ$: Loi de composition externe.

$\boxplus$: Somme binaire.

λ : Niveau de sécurité.

$< u|v >$: Produit scalaire de u et v.

IuI : Norme infi de u.

luJ : Partie entière supérieure.

ful : Partie entière inférieure.

lul : Partie entière la plus proche.

Liste des figures

Liste des tableaux

Introduction générale

Contexte

Cette thèse s'inscrit dans le cadre du projet Saphir fondé en 2016 autour de deux partenaires industriels : KONTRON modular computers à Toulon et l'entreprise WIBU à Karlsruhe en Allemagne. Cette thèse s'est déroulée essentiellement au sein de l'entreprise KONTRON où j'ai fait partie de l'équipe de recherche et développement. Mon encadrant industriel était Serge TISSOT. L'encadrement académique de cette thèse était effectué par l'École des mines de Saint-Étienne sous la direction de Nadia EL MRABET. Le projet SAPHIR vise à offrir une sécurité de bout en bout, des capteurs au Cloud, dans le contexte de l'internet des objets (IoT) nécessitant une sécurité numérique élevée. Le projet répond aux spécificités de la sécurité numérique de l'IoT : grande surface d'attaque avec 50 billions objets d'ici 2020. Le but du projet est de chiffrer les données au niveau des gateways, puis ne les jamais exposées en texte clair, sauf à l'intérieur d'une plate-forme matérielle sécurisée (élément sécurisé) si c'est nécessaire, tout en restant exploitables par le cloud. Les recherches et les innovations mises en oeuvre portent sur le déploiement de la cryptographie homomorphe pour chiffrer les données et permettre leur traitement dans le cloud. Ainsi, les failles de sécurité induites par le facteur humain sont minimisées par une solution sécurisée présentant le chemin total des données.

Présentation de l'entreprise

Le groupe Kontron

L'entreprise a été créée en 1959 par le Dr. Branco Weiss, c'est principalement suite au rachat par BMW en 1999 que Kontron a pris une réelle ampleur sur le marché des calculateurs embarqués. Le groupe a atteint en 2012 presque deux mille cinq cents collaborateurs et un chiffre d'affaires de 590M€ à travers cinquante-cinq régions du monde, comprenant soixante-trois points de vente et cent quarante-cinq partenaires commerciaux. En 2013, la direction de l'entreprise est passée entre les mains de Rolf Schwirz qui a entrepris une importante restructuration du groupe, depuis son nouveau siège social basé à Augsburg en Allemagne. L'entreprise leader dans les technologies embarquées se concentre désormais sur treize sites à travers le monde, mais compte un nombre toujours très important de partenaires commerciaux. De plus, les alliances industrielles

avec Intel, Microsoft, et bien d'autres entreprises viennent renforcer sa position en tant qu'acteur international majeur. On compte aujourd'hui mille quatre cents employés dans le monde et un chiffre d'affaires de 500Me, en croissance permanente.

L'arrivée de l'Internet des Objets a dès le début intéressé Kontron, entraînant un nouveau positionnement stratégique pour l'entreprise. Les objectifs sont clairement évoqués : devenir le premier conseiller en matière d'application Internet of Things (IoT, Internet des Objets) et rester leader dans les technologies de calculateurs embarqués.

Présentation de Kontron Modular Computers SAS, le site de Toulon

Créé en 1986 sous le nom de «CETIA», l'entreprise était nommée «Thales Computers» à partir de l'année 2000 jusqu'en 2008. L'entreprise a été rachetée par Kontron en 2008 pour devenir KOMSA. En 2017 KONTRON a été fusionnée avec S&T pour former « Kontron S&T AG ». Appartenant à la « Business Unit » ATD, le site de Toulon comporte un effectif d'environ 110 personnes et réalise un chiffre d'affaires de 25 Me sur l'année 2014. Appartenant à la Business Unit ATD, Kontron Modular Computers SAS se concentre sur les secteurs de l'aéronautique, du transport, et de la défense. En pratique, on va retrouver près de 70% des projets dédiés au secteur militaire, 25% répartis entre l'aéronautique et le ferroviaire, et le reste sera dédié au secteur industriel, pour des missions particulières nécessitant la compétence de développement en milieu sévère. On retrouve parmi les clients de Kontron Modular Computers SAS des entreprises telles que Alstom, Boeing, Bombardier, DCNS, Raytheon, Thales ou encore Saab. Un haut niveau d'exigence est attendu par ces entreprises.

La cryptologie

La « cryptologie » signifie étymologiquement la science du secret. L'histoire de la cryptologie commence avec l'Égypte ancienne il y a plus qu'environ 4500 ans. Il s'agit principalement d'une forme de communication utilisée par les militaires et les gouvernements pour la confidentialité des messages. Cette méthode a été transformée en science par un papier écrit par Shannon. C'était sans doute le début de la cryptologie moderne, dans laquelle la vérification de l'intégrité des messages et l'authentification de l'identité des parties qui communiquent est devenue aussi essentielle que la garantie de la confidentialité des messages.

La transformation de la cryptologie d'une méthode en une science a été motivée par la popularité des communications sans fil dans les années 1920. Toutefois, c'était la naissance de l'informatique qui a sans doute facilité la recherche en cryptologie.

Le chiffrement est le processus de conversion d'un texte en clair, appelé plaintext, en un texte chiffré, incompréhensible pour un individu lambda, en

utilisant une clé de chiffrement. Le déchiffrement est le processus inverse en utilisant une clé de déchiffrement. Le texte est transmis par un expéditeur (Alice dans la littérature) au destinataire prévu (Bob) du texte en clair, via un canal de communication non sécurisé (tout tiers peut intercepter les données qui circulent par le biais d'un tel canal). Un cryptosystème peut signifier n'importe quel système qui fait appel à la cryptologie ou simplement aux algorithmes, ainsi qu'à des ensembles d'entrées possibles pour effectuer le chiffrement et le déchiffrement des messages.
La cryptologie porte sur deux parties - la cryptographie qui consiste à la conception de cryptosystèmes et la cryptanalyse qui signifie l'attaque des cryptosystèmes pour retrouver le texte en clair à partir d'un texte chiffré et sans avoir la clé. La cryptographie moderne est divisée en deux branches : la cryptographie à clé symétrique, et la cryptographie à clé publique. Dans le premier cas, Alice et Bob partagent une chaîne de code, connue sous le nom de la clé, qui n'est pas divulguée à une autre partie, et est utilisée pour chiffrer et déchiffrer. Dans le deuxième cas, une deuxième clé est utilisée en plus de la clé secrète, c'est une clé publique utilisée pour le chiffrement, et ainsi la clé secrète est utilisée pour le déchiffrement. On définit deux types de chiffrement : le chiffrement déterministe et le chiffrement probabiliste.

Chiffrement déterministe
Un algorithme de chiffrement est dit déterministe si pour un chiffré $m \in P$ et une clé $k \in K$ il retourne le même texte chiffré $E_k(m)$ pour différentes exécution de l'algorithme du chiffrement.

Chiffrement probabiliste
Un algorithme de chiffrement est dit probabiliste si un nombre aléatoire est utilisé pour le chiffrement, de tel sorte que l'exécution de l'algorithme du chiffrement pour un message $m \in P$ avec une clé $k \in K$ retourne différents textes chiffrés $E_k(m)$.

Aujourd'hui, la cryptographie est devenue le coeur de nombreux processus et applications, comme le commerce électronique, les services bancaires par Internet, les achats sur Internet, les guichets automatiques, la protection des dossiers médicaux, la transmission des ordres militaires, et même les moyens de transport. Ainsi, la cryptographie se trouve dans les téléphones, les ordinateurs, les cartes bancaires pour protéger les données et garantir la protection de la vie privée.

Cette thèse

La technologie émergente des appareils mobiles permet aux utilisateurs mobiles d'accéder à un large éventail d'applications grâce à la connexion Internet.

Comme ces applications exigent une puissance de calcul considérable, elles représentent un défi pour les appareils dont la puissance de calcul, la mémoire, le stockage et l'énergie sont limités. Cependant, un tel défi pourrait être surmonté par le cloud computing car celui-là offre des ressources dynamiques pratiquement illimitées pour le traitement et le stockage de données. Néanmoins, les utilisateurs mobiles hésitent encore à adopter cette technologie, car les schémas de chiffrement classiques exigent le déchiffrement des données pour pouvoir être traitées.
La cryptographie homomorphe une solution potentielle pour permettre un calcul arbitraire des données chiffrées sans avoir à les déchiffrer.

En pratique une fois les données chiffrées dans la machine de l'émetteur, les textes chiffrés correspondants sont envoyés au cloud pour le stockage, et ils ne seront plus déchiffrés. Le cloud peut calculer la somme et/ou le produit de ces textes chiffrés. Le résultat est envoyé à l'émetteur, qui peut déchiffrer avec sa clé secrète.
Bien que la cryptographie homomorphe soit considérée comme une solution pour permettre d'effectuer des calculs sécurisés, son efficacité reste un obstacle à sa mise en oeuvre. L'inconvénient des schémas de chiffrement homomorphes est la taille des textes chiffrés, par exemple le chiffré correspondant à un message de 16 bits est de taille 32 Koctets pour un niveau de sécurité 128 bits, ce qui va monopoliser la bande passante dans l'IoT. C'est le prix à payer de nos jours pour pouvoir bénéficier de la sécurité optimale garantie par un cryptosystème homomorphe
L'objectif de cette thèse est d'appliquer un schéma de chiffrement homomorphe dans un cadre réel, et en particulier dans le cadre l'IoT. Et cela, en étudiant l'état de l'art des cryptosystèmes homomorphes, pour pouvoir choisir celui qui est le plus adapté pour ce projet dans le cadre de l'IoT ; puis de réaliser une implémentation optimale, efficace et adapté à un contexte embarqué de celui-ci.

Les contributions scientifiques

Plusieurs travaux de recherches ont été menés pour construire des schémas de chiffrement homomorphe applicables dans un cadre réel, en dehors des laboratoires. Au cours de cette thèse, nous avons étudié les paramètres d'un schéma homomorphe choisi afin de diminuer la taille des chiffrés, tout en garantissant une sécurité optimale. Cette étude nous a permis de trouver un cas d'usage réel et pratique où la sécurité, la confidentialité et l'anonymat, garantis par le chiffrement homomorphe sont de très grande importance. Ensuite nous avons implémenté le schéma homomorphe avec les paramètres choisis pour l'embarquer dans les différents module de l'IoT. Nous avons proposé ainsi une solution pour pouvoir envoyer des alertes à partir du Cloud, en se basant sur la comparaison des données chiff avec un cryptosystème homomorphe. Ces travaux de thèse ont abouti au dépôt d'un brevet ainsi que la publication de deux articles dans deux conférences internationales.

Plan de la thèse

Dans le chapitre 1 de ce document, nous présentons des notions fondamentales en mathématiques pour ce travail, et du domaine des réseaux euclidiens (base, déterminant, minima successifs, défaut et constante d'Hermite). Nous décrivons ensuite les diff ts problèmes mathématiques utilisés dans la construction des schémas de chiffrements homomorphes. Puis nous détaillons les différents types de chiffrement homomorphe et nous donnons quelques exemples de chaque type.
Le chapitre 2 de ce document décrit le brevet qu'on a déposé, à l'offi européen des brevets EPO, au début de cette thèse pour l'application de la cryptographie homomorphe avec un secure element dans l'IoT.
Le chapitre 3 est consacré à la comparaison des deux cryptosystèmes FV et BGV basés sur le problème RLWE.
Le chapitre 4 apporte une démonstration du travail réalisé, en commençant par une première partie qui décrit l'algorithme implémenté, ensuite une deuxième partie qui décrit notre implémentation du schéma et puis une troisième et dernière partie qui est dédiée à la méthode de délégation de calcul dans le CPU que nous avons proposé.
Enfin, le dernier chapitre présente la conclusion de ses travaux de thèse et les perspectives de recherche à poursuivre.

Chapitre 1

État de l'art

Sommaire

Dans ce chapitre on présente l'état de l'art des cryptosystèmes homomorphes. On commence par donner quelques définitions mathématiques, ensuite on décrit l'historique des schémas homomorphes à partir des premiers schémas et leurs améliorations jusqu'aux derniers schémas proposés. L'étude de l'état de l'art nous a permis d'analyser les cryptosystèmes homomorphes, les avantages et les inconvénients de chaque schéma, ce qui nous a permis de faire une comparaison de deux cryptosystèmes dans le chapitre 3 parmis les meilleurs existants FV [21] et BGV [10].

1.1 Rappels mathématiques

Dans cette section, nous présentons les notions et propriétés essentielles des structures algébriques, des polynômes et des réseaux euclidiens. Toutefois, nous nous limitons à celles utiles pour l'étude de la cryptographie homomorphe.

1.1.1 Structures algébriques

Un corps est un objet algébrique constitué d'un ensemble d'éléments et de deux opérations sur cet ensemblequi satisfont à certaines relations. Cette structure est proche de notre intuition de nombres et des opérations que l'on peut leur appliquer. Avant d'énoncer les relations des deux opérations de la structure de corps, rappelons la structure de groupe et d'anneau, étant donné qu'un corps est un cas particulier de la structure d'anneau, et cette dernière est plus riche que la structure du groupe.

Groupes

Un groupe est un ensemble G muni d'une opération $*$, associant à deux éléments a et b de G un troisième élément noté $a*b$, satisfaisant les assertions suivantes :

- $*$ est une loi de composition interne, i.e, pour tous éléments a et b de G, le résultat $a*b$ est aussi dans G,
- L'opération est associative, i.e., pour tous éléments a, b et c de G, $a*(b*c) = (a*b)*c$,
- Il existe un élément e dans G, appelé élément neutre, tel que, pour tout élément a de G, $a*e = e*a = a$,
- Pour tout élément a de G, il existe un élément inverse, que nous noterons a^{-1}, tel que $a*a^{-1} = e = a^{-1}*a$.

Un groupe est dit abélien, ou commutatif, si tous éléments a et b vérifient : $a*b = b*a$.

Les Anneaux

Un anneau A est ensemble muni d'une opération d'addition et d'une opération de multiplication $A \times A \to A$, telles que :

- $(A, +)$ est un groupe abélien, avec l'élément neutre noté $0 \in A$ et appelé zéro de l'anneau,
- L'opération $\times$ est associative, i.e., pour tous éléments a, b et c de A, $a \times (b \times c) = (a \times b) \times c$,
- La multiplication $\times$ est distributive par rapport à $+$, i.e, pour tout élément a, b et c $a \times (b + c) = a \times b + a \times c$ et $(a + b) \times c = (a \times c) + (b \times c)$.

Un anneau $(A, +, \times)$ est dit commutatif si sa multiplication est commutative.
Un anneau $(A, +, \times)$ s'appelle unitaire si l'opération $\times$ admet un élément neutre noté $1 \in A$ et appelé unité de l'anneau.

Corps

Un ensemble K est appelé corps si $(K, +, \times)$ est un anneau commutatif unitaire, tel que pour tout élément dans $K \setminus \{0\}$ $(K, +, \times)$ est inversible pour la loi $\times$.

1.1.2 Polynômes

Soit $n \geq 0$ un entier. On appelle polynôme à coefficients $a_i \in K$ et en l'indéterminée X tout objet P de la forme :

$$P(X) = \sum_{i=0}^{n} (a_i \cdot X^i) = a_0 + a_1 \cdot X + \ldots + a_n \cdot X^n.$$

Si tous les coefficients du polynôme P sont nuls, on dit que P est le polynôme nul et on le note 0. On appelle monôme un polynôme dont tous les coefficients sont nuls sauf, au plus, l'un d'entre eux.

On appelle degré de P, et on note deg(P), le plus grand entier $n \geq 0$ tel que $a_n \neq 0$. Cette définition s'applique à tout polynôme de K[X] sauf au polynôme nul 0. En effet, tous les coefficients de 0 étant nuls, l'ensemble des indices des coefficients non nuls de 0 est vide, donc cet ensemble n'admet pas de plus grand élément.

On notera K[X] l'ensemble des polynômes à une indéterminée à coefficients dans l'anneau commutatif K, autrement dit, K[X] est l'anneau de polynômes à coefficients dans K.

Soient $P = \sum_{i=0}^{n} a_i \cdot X^i$ et $Q = \sum_{i=0}^{n} b_i \cdot X^i$ deux polynômes de K[X], On appelle somme de P et Q, et on note :

$$P + Q = \sum_{i=0}^{n} (a_i + b_i) \cdot X^i .$$

Le produit de P et Q est défini par :

$$P \cdot Q = \sum_{i=0}^{(2n)} \left(\sum_{k=0}^{i} a_k \cdot b_{i-k} \right) \cdot X^i \quad \text{où } a_k = 0 \text{ et } b_k = 0 \text{ pour tout } k \geq n+1.$$

On dit qu'un polynôme P de K[X] est premier ou irréductible sur le corps K s'il n'est pas constant et si ses seuls diviseurs dans K[X] sont les polynômes associés à P et les éléments non nuls de K.

Il faut bien noter que cette définition dépend essentiellement du corps K : il se peut très bien qu'un polynôme donné soit irréductible sur un corps et réductible sur un autre. Dire qu'un polynôme de K[X] est irréductible revient à dire qu'il est impossible de l'écrire comme produit de deux polynômes de K[X] de degré positif. Soit $P \in K[X]$ un polynôme irréductible, alors K[X]/(P) est un corps. Les éléments de ce corps sont les polynômes premiers avec P, de degré inférieur à celui de P.

Exemple 1.1. Soient $P = X^3 + 2X^2 + X + 1$ et $Q = 2X^3 + X^2 + 1$ deux polynômes à coefficients dans $\mathbf{Q}[X]$. La somme de ces polynômes est définie par $P+Q = 3X^3 + 3X^2 + X + 2$ et leur produit est $P \times Q = 2X^6 + 5X^5 + 4X^4 + 4X^3 + 3X^2 + X + 1$.

Exemple 1.2. Le polynôme $X^2 - 2$ est irréductible dans $\mathbf{Q}$ mais pas dans $\mathbf{R}$ car $X^2 - 2 = (X - \sqrt{2})(X + \sqrt{2})$.

Le polynôme $X^2 + 1$ est irréductible dans $\mathbf{R}$ mais pas dans $\mathbf{C}$.

1.1.3 Distribution gausssienne

Les distributions gaussiennes ont un rôle très important dans la construction de certains schémas basés sur les réseaux euclidiens. Par exemple, on a le cryptosystème GVP [26] où il est nécessaire de choisir un vecteur dans un réseau L suivant une distribution gaussienne. Cependant, l'échantillonnage gaussien sur un réseau est difficile et peut conduire à des implémentations complexes. Par contre, il est plus simple d'échantillonner des polynômes suivant une distribution gaussienne discrète sur l'anneau $R = Z[x]/x^d + 1$. Pour cela, il suffit de choisir des polynômes dans R avec des coefficients dans une distribution gaussienne.
Dans ce qui suit, nous rappelons quelques définitions de distributions gaussiennes.

Définition 1.1. Une distribution gaussienne $D_{Z,\sigma}$ sur les entiers centrés en 0 d'écart type σ est une fonction qui associe pour chaque élément $x \in \mathbf{Z}$ une fonction de probabilité $f(x)$ tel que :

$$f(x) = \exp\left(\frac{-\pi \cdot |x|^2}{\sigma^2}\right).$$

Définition 1.2. On dit qu'une distribution gaussienne discrète χ est bornée par $B > 0$ si pour tout $a \leftarrow \chi$ on a $||a||_\infty < B$.

1.1.4 Notions sur les réseaux euclidiens

Dans cette partie, on présente les réseaux euclidiens et en particulier les notions qu'on va utiliser dans ce manuscrit.

Définition 1.3. Un réseau euclidien L de dimension n est un sous-groupe additif discret de $\mathbf{Z}^n$ ($n \geq 0$). Les réseaux d'entiers sont des sous groupes de $\mathbf{Z}^n$. Un réseau euclidien L de taille d est représenté par une base $B = (b_1, b_2, \ldots, b_d)$ de d vecteurs b_i linéairement indépendants et dont les combinaisons génèrent les éléments du réseau. On note que toutes les bases d'un réseau L(B) ont le même nombre d'éléments d appelé rang ou dimension du réseau.
Le réseau euclidien L peut être noté par $L = \{x \in \mathbf{R}^n$ tels que $x = \sum_{i=1}^{d}(x_i b_i)$ et $x_i \in \mathbf{Z}\}$.
En pratique, un réseau L de rang d est représenté par l'une de ses bases $B = (b_1, b_2, \ldots, b_d)$ écrite sous forme d'une matrice appartenant à $\mathbf{R}^{d \times n}$, les lignes de la matrice sont les coordonnées des vecteurs de la base.

Exemple 1.3. Soient $b_0 = (2,1)$ et $b_1 = (3,2)$, deux vecteurs linéairement indépendants dans $\mathbf{R}^2$.
$B = (b_0, b_1) = \begin{matrix} 2 & 1 \\ 3 & 2 \end{matrix}$ est une base du réseau L(B) dans $\mathbf{R}^2$, la figure 1.1 présente le réseau L(B).

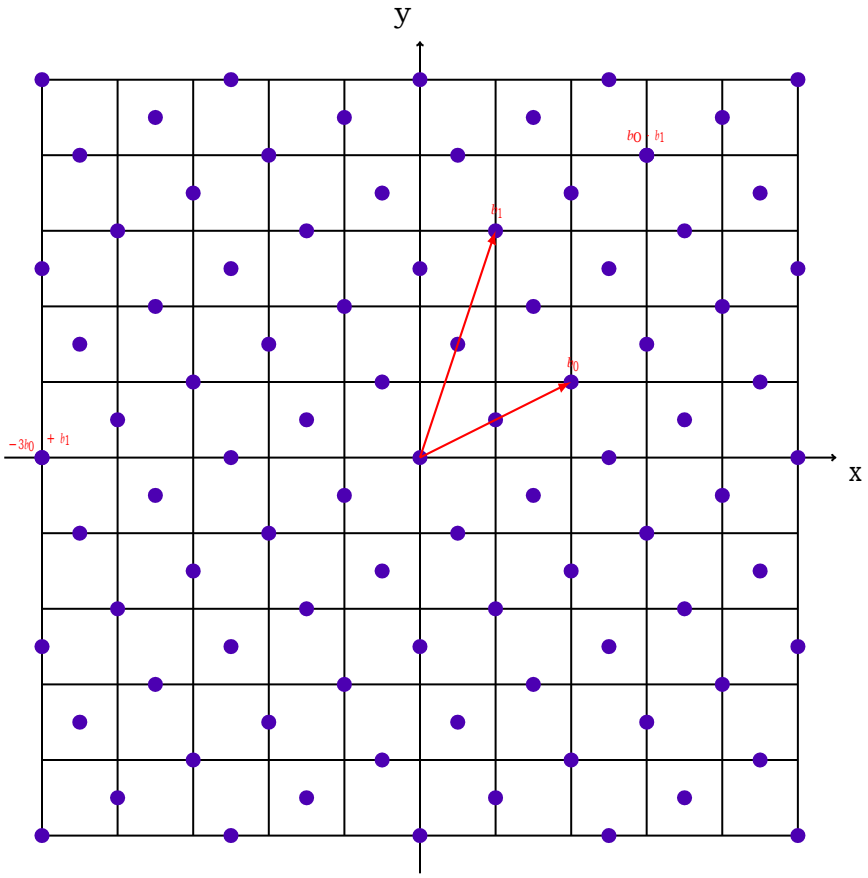

FIGURE 1.1 – Le réseau L de $\mathbf{R}^2$ défini par les vecteurs (2,1) et (1,3).

Orthogonalisation de Gram-Schmidt

La méthode de Gram-Schmidt permet de construire une base orthogonale. Elle permet d'associer aux vecteurs $(b_1, b_2, \ldots, b_d)$ d'une base B, linéairement indépendants une famille de vecteurs orthogonaux $(b_1^*, b_2^*, \ldots, b_d^*)$ d'une base B^*, tel que $B = P \cdot B^*$ où P est une matrice de passage i.e une matrice inversible et unimodulaire (matrice avec des coefficients entiers et un déterminant qui vaut ± 1). Les vecteurs b_i^* vérifient les propriétés suivantes :

$$b_1^* = b_1$$

$$b_i^* = b_i - \sum_{j=1}^{i} (m_{i,j} \cdot b_j^*) \quad pour \quad i \geq 2$$

où

$$m_{i,j} = \frac{< b_i | b_j^* >}{\|b_j^*\|^2}.$$

L'orthogonalité de Gram-Schmidt permet de quantifier l'orthogonalité des vecteurs b_i de la base B : si les coefficients $m_{i,j}$ sont petits, et si les longueurs $\|b_i^*\|$ ne décroissent pas trop vite, alors les vecteurs b_i sont plutôt orthogonaux.

Matrice de Gram :

La matrice de Gram d'un système $B = (b_1, b_2, \ldots, b_d)$ de d vecteurs linéairement indépendants, est la matrice carrée G(B) de $\mathbf{R}^{d\times d}$ définie par

$$G_{ij} = b_i \cdot b_j \quad \forall i, j \in \{1 \ldots d\}$$

La matrice de de Gram est inversible et peut s'exprimer comme $G(B) = B.^tB$.

Parallélépipède fondamental

Soit L(B) un réseau euclidien. Le parallélépipède fondamental de L(B) est défini par :

$$P(L) = \{Bx\ ,\ x \in [0,1[^n\} = \{Bx\ ,\ x \in [-\frac{1}{2}, \frac{1}{2}[^n\}$$

Exemple 1.4. Le parallélépipède fondamental, par rapport à la base $B = \begin{matrix} 2 & 1 \\ 3 & 2 \end{matrix}$ est schématisé par le parallélogramme en gris de la figure 1.2.

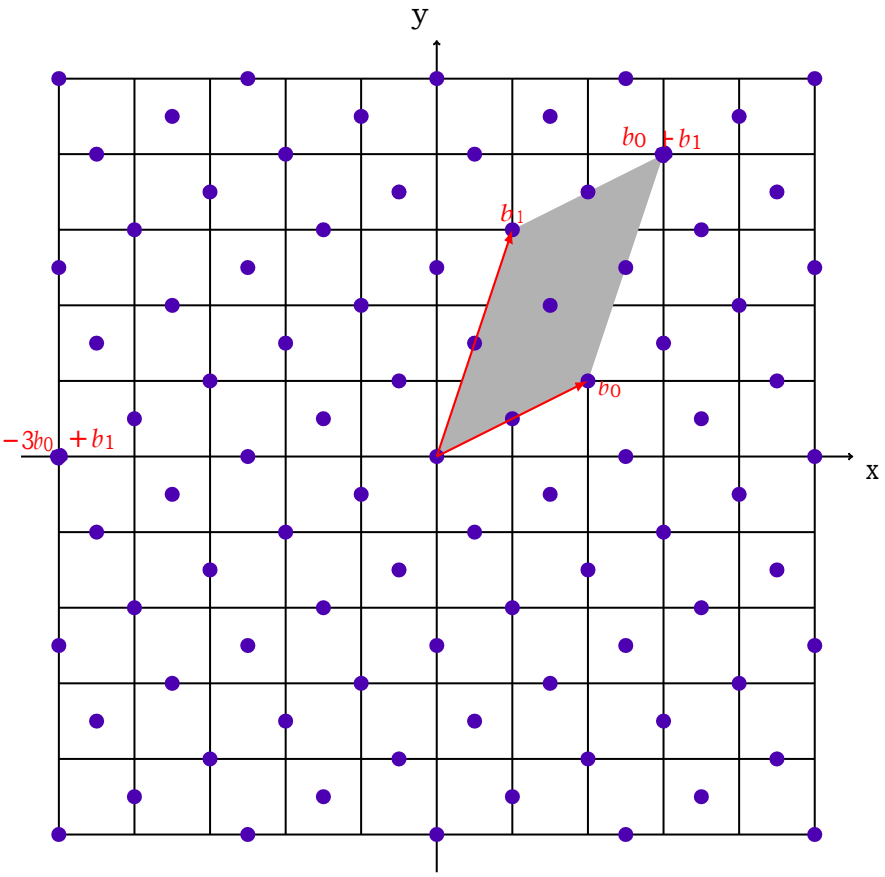

FIGURE 1.2 – Le parallélépipède fondamental du réseau L

Déterminant d'un réseau :

Le déterminant d'un réseau L(B) est défini comme étant le volume du parallélépipède fondamental P(B), il est calculé en utilisant le déterminant de la base B :

$$det(L) = Vol(P) = \sqrt{det(B.^tB)}.$$

Minima successifs

On appelle minima successifs d'un réseau L de $\mathbf{R}^d$ les réels $\lambda_1(L), \lambda_2(L) \ldots \lambda_d(L)$, tels que chaque $\lambda_i(L)$ est le plus petit réel pour lequel il existe i vecteurs libres qui sont de norme inférieure ou égale à ce réel. Soit $B(0,r)$ la boule fermée de centre 0 et de rayon $r \geq 0$, le premier minimum du réseau L de $\mathbf{R}^d$ est défini par :

$$\lambda_1(L) = min(r, B(0,r) \cap L/=0).$$

Invariant d'Hermite :

L'invariant d'Hermite d'un réseau L de $\mathbf{R}^d$ est défini par

$$\gamma(L) = \frac{\lambda_1(L)}{\sqrt[d]{det\,(L)}}.$$

Constante de Hermite :

La constante d'Hermite $\gamma_d\,(L)$ d'un réseau L de $\mathbf{R}^d$ est définie par :

$$\gamma_d\,(L) = sup\{\gamma(L)\}.$$

1.2 Quelques problèmes difficiles sur les réseaux euclidiens

Dans la section suivante on présente les problèmes NP-complet basés sur les réseaux euclidiens. Ce sont des problèmes post-quantiques, résistants aux attaques connues mêmes sur un ordinateur quantique. C'est pour cette raison que les réseaux euclidiens sont appliqués en cryptographie afin de construire des cryptosystèmes fiables pour la cryptographie d'aujourd'hui et de demain [23] [13] [39].

1.2.1 Le problème CVP

Le problème CVP (Closest Vector Problem) consiste à trouver le vecteur du réseau le plus proche d'un vecteur donné. Ce problème existe en deux variantes, présentées ci-dessous. Le problème CVP calculatoire se réduit polynomialement au CVP décisionnel [36].

Approx-CVP : Soit $B \in (Z)^{d \times n}$ une base d'un réseau L(B) et un vecteur $t \in (Q)^n$. La version calculatoire de ce problème est appelé Approx-CVP (Approximate Closest Vector Problem), elle consiste à déterminer le vecteur le plus proche de t, i.e trouver un vecteur $x \in L(B)$ tel que pour tout $y \in L(B)$, $It - xI \leq It - yI$.

Decisional-CVP : Soit $B \in (Z)^{d \times n}$ une base du réseau L(B), un vecteur $t \in (Q)^n$ et un entier $K \in \mathbf{Q}$, la version décisionnelle du problème CVP consiste à répondre à la question suivante : Existe-t-il un vecteur $x \in L(B)$ vérifiant $It - xI \leq K$.

1.2.2 Le problème SVP

Le problème SVP (Shortest Vector Problem), est une variante du problème précédent qui consiste à calculer le plus court vecteur non nul du réseau. On décrit ci-dessous quelques versions de ce problème.

SVP : Soit $B \in (Z)^{d \times n}$ une base du réseau L(B), la version calculatoire de ce problème consiste à déterminer un plus court vecteur v non nul du réseau L, tel que $IvI = \lambda_1(L)$. Notons que ce vecteur n'est pas unique, et il existe au moins deux plus court vecteurs pour un réseau donné. Soit L un réseau avec un plus court vecteur v alors $-v$ est aussi un plus court vecteur. Le problème SVP calculatoire se réduit polynomialement au SVP décisionnel.

Decisional-SVP : Soit $B \in (Z)^{d \times n}$ une base du réseau L(B) et un entier $K \in \mathbf{Q}$, La deuxième version (décisionnelle) de ce problème consiste à déterminer s'il existe un vecteur $x \in L$ non nul, tel que $It - xI \leq K$.

GapSVP : Une autre version décisionnelle du problème SVP est appelée GapSVP. Étant donné une base B du réseau L, un réel c et un facteur γ, ce problème consiste à déterminer si $\lambda_1(L(B)) \leq c$ ou $\lambda_1(L(B)) \geq \gamma \cdot c$.
La complexité du problème GapSVP dépend de la dimension du réseau, et de la taille des entrées de la matrice B. Bien entendu, plus le facteur γ augmente, plus le problème GapSVP devient facile. Il a été prouvé au début des années 1980 par van Emde Boas [50] que le problème GapSVP est NP-difficile pour $\gamma = 1$. Quelques années plus tard, en 1998 Ajtai [3] a démontré que le problème SVP est NP-difficile pour des réductions randomisées, i.e tout autre problème de la classe NP est réductible aléatoirement au problème SVP [28].

SIVP : Soit L un réseau de dimension n, le problème SIVP consiste à retourner n vecteurs indépendants $v_1, v2, \ldots, v_n$, tels que $max Iv_i I \leq \max_B Ib_i I$, où $\max_B Ib_i I$ parcourt toutes les bases $B = \{b_1, b_2, \ldots, b_n\}$ du réseau L.

1.2.3 Le problème LWE : Learning With Errors

Les cryptologues ont essayé de construire d'autres problèmes plus avancés basés sur les réseaux euclidiens. En 2005, Regev [39] a introduit le problème Learning with errors (LWE), qui consiste a trouver un secret masqué au milieu d'équations linéaires bruitées. Formellement, étant donné n une dimension entière, une distribution d'erreur χ d'écart type σ. Le problème LWE consiste à trouver s à partir de m échantillons $(a_i, < a_i, s > + e_i)$ où a_i est un vecteur tiré uniformément de Z_n^q et e_i une erreur de la distribution gaussienne χ sur Z_n^q. La version décisionnelle de ce problème consiste à distinguer ces distributions des distributions uniformes dans $\mathbf{Z}_q^n \times \mathbf{Z}_q$.
Regev a démontré [38] que la résolution du problème LWE dans le cas moyen par

un algorithme quantique implique la résolution des problèmes SIVP et gapSVP dans le pire des cas.

1.2.4 Le problème RLWE : Ring Learning With Errors

L'inconvénient des schémas basés sur le problème LWE est la taille mémoire nécessaire pour le stockage des échantillons a_i et leurs transmissions. Une solution consiste à ajouter une structure au réseau euclidien : les idéaux de réseaux. La description des idéaux de réseaux ne nécessite qu'un vecteur au lieu d'une matrice, d'où la réduction du coût du schéma de $O(n^2)$ à $O(n)$. On définit le problème RLWE [33] sur les anneaux comme suit :
Soit $R = Z[X]/f(x)$ un anneau de polynôme, $f(x) = x^d + 1$ un polynôme cyclotomique de degré $d = 2^k$, χ une distribution d'erreur sur R, et $R_q = R/qR$ un anneau quotient où $q \geq 2$ est un entier. Les éléments de $R_q = R/qR$ sont les polynômes de degré inférieur à d et premier avec $f(x)$ à coefficients modulo q. Étant donné un secret $s \in R_q$, on construit m échantillons $(a, b) = (a, (a \cdot s + e))$ où a est tiré uniformément de R_q et e de χ. Le problème RLWE consiste à trouver s à partir des échantillons. La version décisionnelle du problème consiste à distinguer ces échantillons des échantillons uniformes de R^2_q

1.3 Le chiffrement homomorphe

1.3.1 L'homomorphisme en algèbre

En algèbre un homomorphisme est une application qui relie deux structures algébriques de même nature, par exemple deux groupes ou deux espaces vectoriels. Soit f un homomorphisme de groupe de (G, o) vers $(H, \circ)$, g et g^t deux éléments de G, alors $f(g\, o\, g^t) = f(g) \circ f(g^t)$.

1.3.2 Définition du chiffrement homomorphe

Un schéma de chiffrement homomorphe est un cryptosystème qui permet de faire des calculs sur les données chiffrées sans avoir à les déchiffrer.

Soit (P, C, K, E, D) un schéma de chiffrement, où (P, o) et $(C, \circ)$ sont les groupes des messages et des chiffrés, K est l'espace des clés, $E : P \to C$ est la fonction de chiffrement, et $D : C \to P$ la fonction du déchiffrement.
Soit $a, b \in P$ deux messages et $k \in K$ une clé publique, si $E_k(a) \circ E_k(b) = E_k(a\, o\, b)$ alors le schéma est dit homomorphe. Autrement dit, un cryptosystème E est dit homomorphe si une opération algébrique effectuée sur les chiffrés équivaut à une opération algébrique sur les clairs, et génère un résultat chiffré qui correspond au chiffrement du résultat des opérations sur les clairs.
On distingue deux types de chiffrement homomorphe : le chiffrement partiellement homomorphe et le chiffrement complètement homomorphe.

1.3.3 Le chiffrement partiellement homomorphe

Un schéma de chiffrement partiellement homomorphe supporte une seule opération : l'addition ou la multiplication. On décrit ci-dessous quelques cryptosystèmes homomorphes pour l'addition, et des cryptosystèmes homomorphes pour la multiplication.

Les cryptosystèmes homomorphes additifs

Un schéma homomorphe additif permet d'appliquer une opération sur les textes chiffrés et avoir comme résultat le chiffrement de l'addition sur les messages clairs. Formellement, $E_k(a) \circ E_k(b) = E_k(a+b)$.

Paillier

Le chiffrement de PAILLIER [37], inventé en 1999 par Pascal Pailler est un système de chiffrement asymétrique, probabiliste. Sa sécurité est basée sur le problème de résidualité composée. Il est défini sur un groupe cyclique G.

Génération des clés :

La génération de clés secrète et publique du cryptosystème Paillier est décrite dans la l'Algorithme 1 ci-dessous.

Algorithme 1 Génération de clés

fonction KEYGENERATION

Choisir arbitrairement deux grands nombres premiers, indépendants, de même taille (le nombre de bits d'un nombre en base binaire) tel que : $pgcd(p \cdot q, (p-1)(q-1)) = 1$.

Calculer $n = p \cdot q$.

Calculer $\varphi(n) = (p-1) \cdot (q-1)$.

retourner $s_k = \varphi(n)$ et $p_k = n$.

fin fonction

La clé publique $p_k = p \cdot q$ est un module RSA. Il est recommandé de choisir un module de taille 2048 bits et par conséquent de choisir p et q de taille 1024 bits chacun.

Chiffrement :

Le chiffrement d'un texte clair m est calculé en utilisant la clé publique. Il est basé sur une exponentiation modulaire et un produit modulaire. L'Algorithme 2 décrit le calcul du chiffré $E(m) = g^m \cdot r^n \mod n^2$ d'un message $m \in \mathbf{Z}_n$.

Algorithme 2 Chiffrement

Entrées : $m \in \mathbf{Z}_n, p_k$.
Sortie : $\mathrm{E}(m)$.

fonction CHIFFRER(m, p_k)
Choisir un entier aléatoire $r \in \mathbf{Z}_n^*$.
Calculer $g = (n+1)$.
Calculer $\mathrm{A} = g^m \mod n^2$.
Calculer $\mathrm{B} = r^n \mod n^2$.
Calculer $\mathrm{E}(m) = \mathrm{A} \cdot \mathrm{B} \mod n^2$.
retourner $\mathrm{E}(m)$.
fin fonction

Déchiffrement :
Pour déchiffrer un chiffré $c = \mathrm{E}(m)$, on commence par retrouver le nombre aléatoire r à partir du chiffré c à l'aide de la clé secrète s_k. Ensuite, on calcule le message m en utilisant une exponentiation et un produit modulaire comme décrit dans l'Algorithme 3.

Algorithme 3 Déchiffrement

Entrées : $c = \mathrm{E}(m), s_k$.
Sortie : m.

fonction DÉCHIFFREMENT(c, s_k)
Calculer $\mathrm{A} = n^{-1} \mod s_k$.
Calculer $r = c^{\mathrm{A}} \mod n$.
Calculer $\mathrm{B} = r^{-n} \mod n^2$.
Calculer $\mathrm{C} = c \cdot \mathrm{B} \mod n^2$.
Calculer $m = (\mathrm{C} - 1)/n$.
retourner m.
fin fonction

La propriété homomorphe :
Soit c_1 et c_2 les chiffrés de m_1 et m_2 respectivement tels que $c_1 = g^{m_1} \cdot r_1^n \mod n^2$ et et $c_2 = g^{m_2} \cdot r_2^n \mod n^2$. L'Algorithme 4 décrit le calcul de la multiplication des deux chiff c_1 et c_2.

Algorithme 4 Multiplication de textes chiffrés

Entrées : c_1 et c_2.
Sortie : $\mathrm{E}(m_1 + m_2)$.

fonction MULTIPLICATION DE TEXTES CHIFFRÉS($\mathrm{E}(m)$, s_k)
Calculer $c_1 \times c_2 = (g^{m_1} \cdot r_1^n \mod n^2) \times (g^{m_2} \cdot r_2^n \mod n^2)$.
Remarquons que $c_1 \times c_2 = (g^{m_1+m_2} \cdot (r_1 \cdot r_2)^n) \mod n^2 = \mathrm{E}(m_1 + m_2)$.
retourner $\mathrm{E}(m_1 + m_2)$.
fin fonction

Goldwasser-Micali

Goldwasser et Micali [41] ont proposé en 1982 le premier système de chiffrement probabiliste, à clé publique, basé sur le problème de la résidualité quadratique, et permettant de chiffrer des nombres binaires.

Génération des clés :

La génération de clés secrète et publique de ce cryptosystème est basée sur le calcul du symbole de Legendre, et est décrite dans la l'Algorithme 5 ci-dessous.

Définition 1.4. Le symbole de Legendre [53] :
Si p est un nombre premier et a un entier, alors le symbole de Legendre $(\frac{a}{p})$ vaut :

- 0 si a est divisible par p.
- 1 si a est un résidu quadratique modulo p, mais n'est pas divisible par p, i.e il existe un entier k tel que $a \equiv k^2 mod p$.
- -1 si a n'est pas un résidu quadratique modulo p.

Le symbole de Legendre ne dépend donc que de la classe de a modulo p.
Le cas particulier p = 2 est inclus dans cette définition mais sans intérêt : $(\frac{a}{2})$ vaut 0 si a est pair et 1 sinon.
Si p est un nombre premier différent de 2 alors, pour tout entier a :

$$a^{\frac{p-1}{2}} \equiv \frac{a}{p} \mod p.$$

Algorithme 5 Génération de clés

fonction KEYGENERATION
Choisir deux nombres premiers, distincts, arbitraires p et q.
Calculer $n = p \cdot q$.
répéter
Choisir arbitrairement un entier $y \in \mathbf{Z}_n^*$.
Calculer ses symboles de Legendre $(\frac{y}{p})$ et $(\frac{y}{q})$.
jusqu'à $(\frac{y}{p}) = -1$ et $(\frac{y}{q}) = -1$.
retourner $s_k = p$ et $p_k = (n, y)$.
fin fonction

Chiffrement :
L'Algorithme 6 décrit la procédure du chiffrement d'un bit en utilisant le cryptosystème de Goldwaser-Micali.

Algorithme 6 Chiffrement

Entrées : $m \in \{0, 1\}$.
Sortie : $\mathrm{E}(m)$.

fonction CHIFFRER(m, p_k)
  Choisir uniformément $z \in \mathbf{Z}_n^*$.
  Calculer $\mathrm{A} = y^m \bmod n$.
  Calculer $\mathrm{B} = z^2 \bmod n$.
  Calculer $\mathrm{E}(m) = \mathrm{A} \cdot \mathrm{B} \bmod n$.
retourner $\mathrm{E}(m)$.
fin fonction

Déchiffrement :
Pour déchiffrer un chiffré c en utilisant une clé secrète s_k, on utilise l'Algorithme 7.

Algorithme 7 Déchiffrement

Entrées : $c = \mathrm{E}(m)$ et s_k.
Sortie : m.

fonction DÉCHIFFREMENT(c, s_k)
  Calculer $\left(\frac{c}{p}\right) = c^{\frac{p-1}{2}} \bmod p$.
  si $\left(\frac{c}{p}\right) = 1$ **alors**
    c est un résidu quadratique, donc $b = 0$.
  sinon
    c est un non-résidu quadratique, donc $b = 1$.
  fin si
retourner m.
fin fonction

La propriété homomorphe :
Soit c_1 et c_2 deux chiffrés de b_1 et b_2 respectivement où $c_1 = y^{b_1} \cdot z_1^2 \bmod n$ et $c_2 = y^{b_2} \cdot z_2^2 \bmod n$. L'Algorithme 8 décrit le calcul du produit de deux chiffrés et prouve que le schéma de Goldwaser-Micali et homomorphe pour l'addition. Formellement $c_1 \times c_2 = \mathrm{E}(b_1 + b_2)$, où $\mathrm{E}(b_1 + b_2)$ est le chiffré de $b_1 + b_2$ avec un random $z_1 \cdot z_2$.

Algorithme 8 Multiplication de textes chiffrés

Entrées : c_1 et c_2.
Sortie : $E(b_1 + b_2)$.

fonction MULTIPLICATION DE TEXTES CHIFFRÉS(c_1, c_2)
Calculer $c_1 \times c_2 = (y^{b_1} \cdot z_1^2 \mod n) \times (y^{b_2} \cdot z_2^2 \mod n)$.
Remarquer que $c_1 \times c_2 = y^{b_1+b_2} \cdot (z_1 \cdot z_2)^2 \mod n$.
Donc $c_1 \times c_2 = E(b_1 + b_2)$
retourner $E(b_1 + b_2)$.
fin fonction

Bonaloh

Le crytosystème de Benaloh [7] est une généralisation du schéma de Goldwaser-Micali, qui permet de manipuler des messages d'une longueur quelconque.

Génération des clés :

La clé publique du cryptosystème de Benaloh est constituée d'un nombre arbitraire vérifiant une certaine propriété et d'un module RSA, calculé à partir du produit de deux nombres premiers constituant la clé secrète. La génération des clés est décrite dans la l'Algorithme 9 ci-dessous.

Algorithme 9 Génération de clés

fonction KEYGENERATION
Choisir deux grands nombres premiers p et q (de taille équivalente à celle des clés RSA).
Calculer $n = p \cdot q$.
Calculer $\varphi = (p-1) \cdot (q-1)$.
Choisir r la taille de bloc tel que r vérifie les propriétés suivantes :
- r divise $p-1$,
- $pgcd(r, \frac{p-1}{r}) = 1$,
- $pgcd(q-1, r) = 1$.

Calculer $A = \frac{\varphi}{r}$.
répéter
Choisir arbitrairement un entier $y \in \mathbf{Z}_n^*$.
$B = y^A \mod n$.
jusqu'à $B /= 1 \mod n$.
retourner $s_k = (\varphi, B)$ et $p_k = (y, n)$.
fin fonction

Chiffrement :

L'Algorithme 10 décrit la procédure du chiffrement d'un message $m \in \mathbf{Z}_r$ en utilisant le cryptosystème de Benaloh. C'est un algorithme de chiffrement probabiliste, d'où la génération d'un aléa r pour chaque appel de la fonction de chiffrement.

Algorithme 10 Chiffrement

Entrées : $m \in \mathbf{Z}_r$.
Sortie : $\mathrm{E}(m) \in \mathbf{Z}_n^*$.

fonction CHIFFRER(m, p_k)
  Choisir un random $u \in \mathbf{Z}_n^*$.
  Calculer $\mathrm{A} = y^m \mod n$.
  Calculer $\mathrm{B} = u^r \mod n$.
  Calculer $\mathrm{E}(m) = \mathrm{A} \cdot \mathrm{B} \mod n$.
retourner $\mathrm{E}(m)$.
fin fonction

Déchiffrement :
Soit $c = \mathrm{E}(m)$ un chiffré de m, remarquons que :

$$z \equiv c^{\frac{\varphi}{r}} \equiv (y^m u^r)^{\frac{\varphi}{r}} \equiv (y^m)^{\frac{\varphi}{r}} (u^r)^{\frac{\varphi}{r}} \equiv (y^m)^{\frac{\varphi}{r}} u^{\varphi} \equiv \mathrm{B}^m \mod n.$$

Ainsi pour déchiffrer le chiff c il suffit de trouver le logarithme discret m de B, tel que $\mathrm{B}^m = z \mod n$ i.e $m = \log_{\mathrm{B}}(z)$. Les étapes du déchiffrement sont décrites dans l'Algorithme 11.

Algorithme 11 Déchiffrement

Entrées : $c = \mathrm{E}(m)$ et s_k.
Sortie : m.

fonction DÉCHIFFREMENT(c, s_k)
  Trouver m le logarithme discret de a, tel que $x^m = a \mod n$
  si (r est petit) **alors**
    Pour tout $i \in 0 \ldots (r-1)$, trouver i tel que $x^i \mod n = a$.
  sinon
    Utiliser l'algorithme Baby-step giant-step [51] pour retrouver m avec une complexité $\mathrm{O}(\sqrt{r})$.
  fin si
retourner m.
fin fonction

La propriété homomorphe :
Soit c_1 et c_2 deux chiffrés de m_1 et m_2 respectivement où $c_1 = y^{m_1} u_1^2 \mod n$ et $c_2 = y^{m_2} u_2^2 \mod n$. Le produit de ces deux chiffrés vérifie la propriété suivante : $c_1 \times c_2 = \mathrm{E}(m_1 + m_2)$. L'Algorithme 12 présente le calcul du produit deux chiffrés c_1 et c_2.

Algorithme 12 Multiplication de textes chiffrés

Entrées : c_1 et c_2.
Sortie : $E(m_1 + m_2)$.

fonction MULTIPLICATION DE TEXTES CHIFFRÉS(c_1, c_2)
Calculer $c_1 \times c_2 = (y^{m_1} u_1^2 \mod n) \times (y^{m_2} u_2^2 \mod n)$.
Remarquer que $c_1 \times c_2 = y^{m_1+m_2} \cdot (u_1 \cdot u_2)^2 = E(m_1 + m_2)$.
Donc $c_1 \times c_2 = E(m_1 + m_2)$.
retourner $E(m_1 + m_2)$.
fin fonction

Cryptosystèmes homomorphes multiplicatifs

Un cryptosystème homomorphe multiplicatif permet d'appliquer une opération $\circ$ sur les textes chiffrés et d'avoir comme résultat le chiffrement de la multiplication $\times$ sur les messages clairs. Formellement si $E_k(a)$ et $E_k(b)$ sont deux chiffrés de a et b, E est dit système de chiffrement homomorphe multiplicatif si $E_k(a) \circ E_k(b) = E_k(a \times b)$.

RSA

Le système de chiffrement asymétrique déterministe RSA [40] a été inventé par trois mathématiciens : Ron Rivest, Adi Shamir et Len Adleman, en 1977. Il est basé sur le problème de factorisation d'entiers.

Génération des clés :

La génération de la clé secrète et publique du cryptosystème RSA est décrite dans l'Algorithme 13.

Algorithme 13 Génération de clés

fonction KEYGENERATION
Choisir arbitrairement deux grands nombres premiers, de taille à peu près égale,
indépendants, p et q tel que : $pgcd(p, q) = 1$.
Calculer le module $n = p \cdot q$.
Choisir aléatoirement un entier $e \in \mathbf{Z}_n^*$.
Calculer le nombre d'Euler $\varphi(n) = (p-1) \cdot (q-1)$.
Calculer d l'inverse de e modulo $\varphi(n)$ tel que $e \cdot d = 1 \mod \varphi(n)$.
retourner La clé secrète $s_k = (p, q, d)$ et la clé publique $p_k = (n, e)$.
fin fonction

Chiffrement :

Soit m un entier naturel strictement inférieur à n. Le chiffrement de m est obtenu en calculant une exponentiation modulaire, tel que décrit dans l'Algorithme 14, formellement $E(m) = m^e \mod n$.

Algorithme 14 Chiffrement

Entrées : $m \in \mathbf{Z}_n$ et p_k.
Sortie : $\mathrm{E}(m) \in \mathbf{Z}_n^*$.

fonction CHIFFRER(m, p_k)
Calculer $\mathrm{E}(m) = m^e \bmod n$.
retourner $\mathrm{E}(m)$.
fin fonction

Déchiffrement :
Le calcul du déchiffrement d'un chiffré c est basé sur le calcul d'une exponentiation modulaire. Remarquons que :

$$c^d \equiv m^{e \cdot d} \equiv m^{1+k \cdot \varphi(n)} \equiv m \cdot (m^k)^{\varphi(n)} \equiv m \mod n \text{ où } m^{\varphi(n)} = 1 \mod n.$$

L'Algorithme 15 décrit le déchiffrement d'un chiffré.

Algorithme 15 Déchiffrement

Entrées : $c = \mathrm{E}(m)$ et s_k.
Sortie : m.

fonction DÉCHIFFREMENT(c, s_k)
Calculer $m = c^d \bmod n$.
retourner m.
fin fonction

La propriété homomorphe :
Soient $c_1 = m_1^e \bmod n$ et $c_2 = m_2^e \bmod n$ deux chiffrés de m_1 et m_2 respectivement, le calcul du produit de ces deux chiffrés est défini dans l'Algorithme 16.

Algorithme 16 Multiplication de textes chiffrés

Entrées : c_1 et c_2.
Sortie : $\mathrm{E}(m_1 \times m_2)$.

fonction MULTIPLICATION DE TEXTES CHIFFRÉS(c_1, c_2)
Calculer $c_1 \times c_2 = m_1^e \times m_2^e \bmod n$.
Remarquer que $c_1 \times c_2 = (m_1 \times m_2)^e \bmod n$.
Donc $c_1 \times c_2 = \mathrm{E}(m_1 \times m_2)$
retourner $\mathrm{E}(m_1 \times m_2)$.
fin fonction

ElGamal
Il s'agit d'un cryptosystème [20] à clé publique, probabiliste basé sur le problème du Logarithme discret. Il est défini sur un groupe cyclique G.

Génération des clés :
L'Algorithme 17 décrit la génération de la clé secrète s_k et la clé publique p_k du cryptosystème El Gamal.

Algorithme 17 Génération de clés

fonction KEYGENERATION
Choisir G un groupe cyclique,
Déterminer g un générateur,
Déterminer q son ordre,
choisir un nombre au hasard $a \in \{1, \ldots, q-1\}$,
Calculer $y = g^a$.
retourner $s_k = a$ et $p_k = (G, g, q, y)$.
fin fonction

Chiffrement :
Le chiffrement d'un message m est un couple (c_1, c_2), l'Algorithme 18 décrit la procédure de chiffrement d'un message avec le schéma d'ELGamal.

Algorithme 18 Chiffrement

Entrées : $m \in G$ et p_k.
Sortie : $E(m) \in \mathbf{G} \times G$.

fonction CHIFFRER(m, p_k)
Choisir un entier aléatoire $r \in \{1, ..., q-1\}$,
Calculer $c_1 = g^r$.
Calculer $c_2 = m \cdot y^r$.
retourner $E(m) = (c_1, c_2)$.
fin fonction

Déchiffrement :
Pour déchiffrer un chiffré $E(m) = (c_1, c_2)$, on calcule $\frac{c_2}{c_1^a}$ comme décrit dans l'Algorithme 19.

Algorithme 19 Déchiffrement

Entrées : $c = (c_1, c_2)$ et s_k.
Sortie : m.

fonction DÉCHIFFREMENT(c, s_k)
Calculer $m = \frac{c_2}{c_1^a}$.
retourner m.
fin fonction

La propriété homomorphe :
Soient $(c_1, c_2) = (g^{r_1}, m_1 \cdot y^{r_1})$ et $(c_1^t, c_2^t) = (g^{r_2}, m_2 \cdot y^{r_2})$ deux chiffrés de m_1 et m_2

respectivement, le calcul du produit de deux chiffrés est défini dans l'Algorithme 20.

Algorithme 20 Multiplication de textes chiffrés

Entrées : c_1 et c_2.
Sortie : $E(m_1 + m_2)$.

fonction MULTIPLICATION DE TEXTES CHIFFRÉS(c_1, c_2)
Calculer $(c_1, c_2) \times (c_1^t, c_2^t) = (c_1 \cdot c_1^t, c_2 \cdot c_2^t) = (g^{r_1+r_2}, (m_1 \cdot m_2) y^{r_1+r_2}) = E(m_1 + m_2)$.
Remarquer que $(c_1 \cdot c_1^t, c_2 \cdot c_2^t) = (g^{r_1+r_2}, (m_1 \cdot m_2) y^{r_1+r_2}) = E(m_1 + m_2)$.
Donc $(c_1, c_2) \times (c_1^t, c_2^t) = E(m_1 + m_2)$.
retourner $E(m_1 + m_2)$.
fin fonction

1.3.4 Systèmes de chiffrement complètement homomorphe

Les Schémas Somewhat Homomorphe : SWHE

Plusieurs tentatives on été faites dans le but d'obtenir des schémas complètement homomorphes, permettant d'évaluer à la fois un nombre constant d'additions et de multiplications. En 2005, Boneh, Goh et Nissim ont développé le premier cryptosystème [8] permettant d'évaluer l'addition et la multiplication sur les données chiffrées, en se basant sur le problème décisionnel d'appartenance à un sous-groupe. Cependant ce schéma ne peut évaluer qu'un circuit composé d'une seule multiplication. En 2008, Aguilar-Melchor et al. [14] ont construit un schéma basé sur les réseaux euclidiens, permettant d'évaluer un nombre illimité d'additions et quelques multiplications de chiffrés. Ces schémas ont été classés dans une catégorie de schéma appelé schéma SWHE (somewhat homomorphe), et permettant d'évaluer des circuits avec un nombre de multiplication fixe.
La majorité des schémas SWHE ajoute un bruit aux chiffrés durant le chiffrement afin d'obtenir des schémas complètement homomorphes. Ce bruit augmente avec la multiplication, d'où la profondeur limitée des circuits qu'ils peuvent évaluer. Pour rafraîchir le bruit et permettre d'augmenter la taille des circuits à évaluer, plusieurs techniques ont été conçues dans le but de réduire ce bruit après le calcul du produit de chiffrés. Ces techniques sont présentées ci-dessus :

La relinéarisation

En 2011, Brakerski et Vaikuntanathan [11] ont introduit la technique de la relinéarisation ou le changement de clé, qui permet de gérer le bruit par le changement de dimension. Elle transforme un texte chiffré c_1 chiffré avec une clé s_1 en un nouveau texte chiffré c_2 chiffré avec une autre clé s_2, tel que c_1 et c_2 chiffrent le même message. Cette méthode diminue naturellement la taille des chiffrés.

La réduction de module

Cette méthode a été développée aussi en 2011, par Brakerski et Vaikuntanathan [12] pour la gestion du bruit. Elle consiste à réduire le bruit d'un chiffré en transformant un chiffré modulo q en un nouveau chiffré modulo q^t, tel que $q^t < q$. Le bruit est réduit d'un facteur $\frac{q}{q^t}$. Cette technique réduit la taille des chiffrés et la complexité de la fonction de déchiffrement.

Les Schémas Fully homomorphe : FHE

Il s'agit des schémas SWHE complétés par une technique de rafraîchissement (Bootstrapping). Un schéma FHE (Fully homomorphic encryption) permet d'évaluer un circuit de profondeur infi d'additions et de multiplications. Formellement SWHE + B$ootstrapping$ = FHE. Le premier schéma fully homomorphe a été décrit par Gentry [22] en 2009, il introduit la technique du bootstrapping. La sécurité du schéma repose sur les problèmes difficiles des idéaux de réseaux qui sont de même difficulté que ceux des réseaux en général. Grâce à ce schéma l'évaluation des circuits avec un nombre arbitraire d'additions et de multiplications est devenue possible sur le plan théorique, car il n'était pas efficace sur le plan pratique. L'implémentation de ce schéma a été réalisé en 2011 par Gentry et Halevi [25] en faisant appel à plusieurs optimisations et notamment celles utilisées dans l'implémentation de Smart et Vercauteren [44]. Ces derniers ont implémenté le schéma SWHE en omettant la procédure du bootstrapping. Le premier schéma permettant d'améliorer le schéma de Gentry est basé sur les entiers, il a été introduit en 2010 par van Dijk et al. [49]. Sa sécurité est basée sur le problème AGCD (Approximate Greatest Common Divisor). Ce schéma a été amélioré en 2011 par Jean-Sébastien Coron, Avradip Mandal, David Naccache, et Mehdi Tibouchi [16] et puis en 2012 par Jean-Sébastien Coron, David Naccache, et Mehdi Tibouchi [17]. Ensuite, plusieurs schémas homomorphe de la deuxième génération, basés sur le problème LWE ou RLWE ont été conçus. Le schéma de Brakerski et Vaikuntanathan [11] a introduit de nouvelles techniques comme le modulus switching pour retarder le besoin du bootstrapping. Puis en 2012, une nouvelle technique de gestion de bruit, appelée le modulus switching a été introduite par Brakerski et al. [10] implémentée dans la librairie HELIB [29]. Ensuite, Brakerski [9] a conçu un nouveau schéma sans modulus switching basé sur le problème LWE, qui a été transformé en 2012 en un schéma basé sur RLWE, par Fan et Vercauteren [21]. Le cryptosytème de Fan et Vercauteren (FV) est devenu l'un des schémas somewhat homomorphe les plus pratiques. Deux implémentations de ce cryptosystème existent : FV-NFLIB [43] et SEAL [42] qui a été développée par une équipe de recherche de Microsoft. En 2016, le cryptosystème FV a été amélioré en utilisant une présentation RNS [5]. Plus tard, une troisième génération de cryptosystème homomorphe a été conçue où l'étape de la relinéarisation qui consiste à diminuer la taille des chiffrées issus d'une multiplication de deux textes chiffrés, n'est plus nécessaire. Parmi les cryptosystèmes les plus connus de cette génération on trouve le schéma de GSW [27] basé sur le problème LWE, et sa variante SHIELD [32]. Récemment, une nouvelle génération de cryptosystèmes homo-

morphes a été développée, permettant de réduire le temps du bootstrapping. En 2015, Ducas et Micciancio [18] ont décrit un schéma basé sur le schéma GSW, permettant d'exécuter le bootstrapping en moins d'une seconde. Ce schéma a été amélioré par Chillotti et al. [15], qui ont réduit le temps du bootstrapping à moins de 0.1 s.

Bootstrapping

La technique du "bootstrapping" a été inventée par Craig Gentry [22] dans sa thèse en 2009. Elle consiste à rafraîchir le texte chiffré périodiquement afin de réduire le bruit et en conséquence d'évaluer un circuit de profondeur illimité. Pour rafraîchir un chiffré, le chiffré et la clé secrète sont re-chiffrés, ensuite il faut évaluer la fonction de déchiffrement sur le chiffré du chiffré en utilisant le chiffré de la clé secrète. Cette méthode est très coûteuse, c'est pour cette raison que plusieurs recherches ont été faites pour essayer de l'éviter ou d'améliorer les performances pour la rendre pratique. On présente ci-dessous la description de la technique de relinéarisation et de réduction de module, permettant d'éviter l'utilisation du bootstrapping pour des circuits de profondeur constante et non négligeable. Plusieurs améliorations du bootstrapping ont été faites par la suite, afin de bénéficier de cette technique et évaluer des circuits de profondeur illimité. Ces améliorations ont permis de réduire le temps d'exécution du bootstrapping à moins d'une seconde [19] et ensuite à 0.1 second [15].

1.3.5 Les schémas basés sur les réseaux euclidiens

Gentry

En 2009, Craig Gentry [22] a proposé le premier cryptosystème Fully homomorphe permettant d'évaluer des circuits de longueur illimitée. L'idée de Gentry consiste à construire un schéma somewhat homomorphe à clé secrète qui chiffre bit par bit en ajoutant du bruit qui augmente linéairement avec l'addition et exponentiellement avec la multiplication, et ensuite nettoyer périodiquement ce bruit pour pouvoir déchiffrer correctement. La sécurité de ce schéma repose sur les réseaux idéaux.
Le bruit de ce schéma est choisi dans un idéal I d'un anneau R $e = k \cdot I \in I \subset R$. Le message est alors chiffré en ajoutant ce bruit au message $E(m) = m + k \cdot I$. Ainsi, la procédure de déchiff t consiste à retirer ce bruit. Le choix des réseaux euclidiens pour la construction d'un système complètement homomorphe (bootstrappable) est justifié, d'une part, par la complexité du déchiff t qui est moins faible dans un réseau par rapport aux schémas classiques basés sur l'exponentiation (On ne sait pas comment la bien paralléliser), et d'autre part, le fait qu'un idéal de réseaux correspond à un idéal d'anneau de polynôme et hérite de l'addition et de la multiplication. La sécurité du schéma est basé sur le problème "Ideal Coset Problem" décrit dans la définition 1.5. On présente ci-dessous les algorithmes pour générer les clés 21, chiffrer 22, déchiffrer 23, et pour la vérification des propriétés homomorphes 24 et 25.

Génération des clés :
La clé secrète du schéma de Gentry est choisie comme étant une "bonne base" B_I, i.e une base avec des vecteurs orthogonaux et petits. En revanche, la clé publique doit être une "mauvaise base" avec des vecteurs pas assez orthogonaux ou pas assez petits. La génération du couple clé secrète, clé publique est décrite dans l'Algorithme 21.

Algorithme 21 Génération de clés

fonction KEYGENERATION
Choisir un anneau R,
Choisir un idéal I de R,
Choisir une base B_I de I,
Choisir J un idéal tel que $I + J = R$,
Générer une bonne base B_J^{sk} pour J,
Générer une mauvaise base B_J^{pk} pour J,
retourner La clé secrète $s_k = B_J^{sk}$ et la clé publique $p_k = (R, B_I, B_J^{pk})$.
fin fonction

Définition 1.5. Étant donné un anneau R, une base B_I, un algorithme $Ideal\,Gen$ et un algorithme $\underset{sk}{Samp_1}$ d'échantillonnage efficace de R. L'attaquant choisit $b \leftarrow \{0,1\}$ et $(B_J, \underset{p_k}{B_J}) \leftarrow Ideal\,Gen(R, B_I)$. Si $b = 0$ choisir $r \leftarrow Samp_1(R)$ et $t \leftarrow r \bmod B_J^{pk}$. Si $b = 1$ choisir t uniformément dans $R \bmod B_J^{pk}$. Le problème ICP (Ideal coset problem) [24] consiste à trouver le nombre b étant donné (t, B_J^{pk}).

Chiffrement :
Soit $m \in R \bmod B_I$ un message à chiffrer, tel que B_I est une base de I, pour chiffrer ce message on applique l'Algorithme 22.

Algorithme 22 Chiffrement

Entrées : $m \in R \bmod B_I$ et $p_k = (B_J, B_J^{pk})$.
Sortie : $E(m) \in B_J^{pk}$.

fonction CHIFFRER(m, p_k)
Choisir aléatoirement un bruit $e = k \cdot I$ dans l'idéal I,
Calculer $E(m) = m + e$.
retourner $E(m)$.
fin fonction

Déchiffrement :
Pour déchiffrer un chiffré c il suffit de retirer l'erreur à l'aide de la clé secrète en calculant $(c \bmod B_J^{sk}) \bmod B_I$. Notons que l'évaluation d'un vecteur v modulo une base B retourne un résultat unique :

$$v \mod B = v - B \cdot lB^{-1} \cdot vl.$$

L'Algorithme 7 décrit le déchiffrement d'un chiffré c.

Algorithme 23 Déchiffrement

Entrées : Un chiffré c et la clé secrète $s_k = \mathrm{B}_{\mathrm{J}}^{sk}$.
Sortie : m.

fonction DÉCHIFFREMENT(c, s_k)
Calculer $m = (c \mod \mathrm{B}_{\mathrm{J}}^{sk}) \mod \mathrm{B}_{\mathrm{I}}$.
retourner m.
fin fonction

La propriété homomorphe :
Les propriétés homomorphes du système sont quasi-immédiates, car les idéaux sont connus pour être stables par rapport à l'addition et à la multiplication. Soit c_1 et c_2 deux chiffrés de m_1 et m_2, tels que $c_1 = m_1 + k_1 \cdot \mathrm{I}$ et $c_1 = m_2 + k_2 \cdot \mathrm{I}$. Le calcul de la somme et du produit de m_1 et m_2 à partir de c_1 et c_2 est décrit dans l'Algorithme 24 et 25.

Algorithme 24 L'homomorphisme pour l'addition

Entrées : c_1 et c_2.
Sortie : $\mathrm{E}(m_1 + m_2)$.

fonction MULTIPLICATION DE TEXTES CHIFFRÉS(c_1, c_2)
Calculer $c_1 + c_2 = m_1 + m_2 + (k_1 + k_2) \cdot \mathrm{I}$.
Calculer $c_1 + c_2 = \mathrm{E}(m_1 + m_2)$.
retourner $\mathrm{E}(m_1 + m_2)$.
fin fonction

Algorithme 25 L'homomorphisme pour la multiplication

Entrées : c_1 et c_2.
Sortie : $\mathrm{E}(m_1 \times m_2)$.

fonction MULTIPLICATION DE TEXTES CHIFFRÉS(c_1, c_2)
Calculer $c_1 \cdot c_2 = m_1 \cdot m_2 + (m_1 \cdot k_2 + m_2 \cdot k_1 + k_1 \cdot k_2) \cdot \mathrm{I}$.
Calculer $c_1 \cdot c_2 = \mathrm{E}(m_1 \cdot m_2)$.
retourner $\mathrm{E}(m_1 \times m_2)$.
fin fonction

Remarquons que le bruit augmente après chaque opération et il est plus affecté par une multiplication que par une addition. Après un certain nombre d'opérations sur les textes chiffrés, ce bruit va finir par dépasser un certain seuil et on ne peut plus déchiffrer correctement. C'est pour cette raison que Gentry a proposé dans ce schéma la procédure de Bootstrapping permettant de rafraîchir le chiffré périodiquement. Gentry a décrit aussi une transformation de

"squashing" pour réduire la complexité de la fonction de déchiffrement et qui lui avait coûté une hypothèse de sécurité supplémentaire, comme la somme de sous-ensembles peu denses SSSP (sparse subset sum problem).

1.3.6 Les schémas basés sur les entiers

Boneh-Goh-Nissim (BGN)

Ce cryptosystème construit en 2005 par Boneh et al. [8] est le premier schéma SWHE qui peut supporter un nombre arbitraire d'additions et une seule multiplication. Il est basé sur le problème décisionnel d'appartenance à un sous-groupe.
L'inconvénient de ce cryptosystème est la taille des messages qui doit être petite pour pouvoir déchiffrer correctement.

Génération des clés :
La génération de la clé secrète et la clé publique est décrite dans l'Algorithme 26.

Algorithme 26 Génération de clés

fonction KEYGENERATION
Choisir deux nombres premiers p et q.
Calculer $n = p \cdot q$.
Choisir G et G_1 deux groupes multiplicatifs d'ordre n.
Choisir un couplage bilinéaire $e : G \times G \to G_1$
Choisir un entier positif $T < q$.
Déterminer g et u deux générateurs de G.
Calculer $h = u \cdot q$ un générateur de G d'ordre p.
retourner La clé secrète $s_k = p$ et la clé publique $p_k = \{n, g, h, G, G_1, e\}$.
fin fonction

Chiffrement :
Soit $m \in \mathbf{Z}_T$ le message à chiffrer, pour calculer le chiffrement de ce message on utilise l'Algorithme 27.

Algorithme 27 Chiffrement

Entrées : $m \in \mathbf{Z}_T$ et $p_k = \{n, g, h, G, G_1, e\}$.
Sortie : $E(m) \in G$.

fonction CHIFFRER(m, p_k)
Choisir aléatoirement un random $r \in \mathbf{Z}_n$,
Calculer $A = g^m \mod n$,
Calculer $B = h^r \mod n$,
Calculer $E(m) = A \cdot B \mod n$.
retourner $E(m)$.
fin fonction

Déchiffrement :
Pour déchiffrer un chiffré c on calcule $c = (E_r(m))^p = (g^p)^m \bmod n$ et on utilise la méthode lambda de Pollard pour trouver le logarithme discret de c dans la base g^p. L'Algorithme 28 décrit les étapes de déchiffrement. Cela nécessite $O(\sqrt{r})$ en temps et $O(log\ (T))$ en mémoire.

Algorithme 28 Déchiffrement

Entrées : Un chiffré c et la clé secrète $s_k = p$.
Sortie : m.

fonction DÉCHIFFREMENT(c, s_k)
Calculer $c^p \equiv (g^p)^m \bmod n \equiv (X)^m \bmod n$.
Utiliser la méthode de lambda de Pollard [52] pour trouver le logarithme discret de c
dans la base $X = g^p$.
retourner m.
fin fonction

La propriété homomorphe :
Soit c_1 et c_2 deux chiffrés de m_1 et m_2, tels que $E_{r_1}(m_1) = g^{m_1} h^{r_1} \bmod n$ et $E_{r_2}(m_2) = g^{m_2} h^{r_2} \bmod n$. Le calcul de la somme et du produit de m_1 et m_2 à partir de c_1 et c_2 est décrit dans l'Algorithme 29 et 30.

Algorithme 29 L'homomorphisme pour l'addition

Entrées : c_1 et c_2.
Sortie : $E(m_1 + m_2)$.

fonction MULTIPLICATION DE TEXTES CHIFFRÉS(c_1, c_2)
Choisir un nombre aléatoire $r^t \in \mathbf{Z}_n$,
Calculer $A = h^t$,
Calculer $c_1 \times c_2 \times A = g^{m_1+m_2} h^{r_1+r_2+r^t} = E(m_1 + m_2)$.
Donc $c_1 \times c_2 \times A = E(m_1 + m_2)$.
retourner $E(m_1 + m_2)$.
fin fonction

Algorithme 30 L'homomorphisme pour la multiplication

Entrées : c_1 et c_2.
Sortie : $E(m_1 \times m_2)$.

fonction MULTIPLICATION DE TEXTES CHIFFRÉS(c_1, c_2)
Déterminer g_1 d'ordre n, tel que $g_1 = e(g, g)$,
Déterminer h_1 d'ordre q_1, tel que $h_1 = e(g, h)$,
Choisir un nombre aléatoire $r \in \mathbf{Z}_n$,
Calculer $e(c_1, c_2) \cdot h_1^r = e(g^{m_1} h^{r_1}, g^{m_2} h^{r_2}) h_1^r$.

D'où $e(c_1, c_2) \cdot h_1^r = g_1^{m_1 m_2} h_1^{m_1 r_2 + r_2 m_1 + \alpha q_2 r_1 r_2 + r}$.
Donc $e(c_1, c_2) \cdot h_1^r = g_1^{m_1 m_2} h_1^{r't}$.
retourner $E(m_1 \times m_2)$.
fin fonction

DGHV

C'est le premier cryptosystème complètement homomorphe basé sur les entiers. Ce schéma a été introduit en 2010 par van Dijk, Craig Gentry, Shai Halevi and Vinod Vaikuntanathan [49]. C'est un schéma plus simple qu'efficace et sa sécurité repose sur le problème du diviseur commun approché (AGCD). Pour construire un schéma SWHE, les auteurs ont proposé un schéma plus simple que efficace, l'idée de ce schéma est que la somme et le produit de deux nombres proches d'un multiple de p soit également proche d'un multiple de p. On commence par décrire les algorithmes de la version symétrique de ce schéma et ensuite la version asymétrique. Ce schéma utilise la procédure du bootstrapping de façon analogue à celle proposée par Gentry pour passer à une construction FHE.

Génération des clés :

C'est un schéma symétrique, et donc une seule clé sera utilisée pour le chiffrement et le déchiffrement. La clé du schéma est un entier p impair.

Chiffrement :

Le chiffrement de $m \in \{0, 1\}$ est $E(m) = m + 2 \cdot r + p \cdot q$ où r et q sont deux nombres arbitraires. L'Algorithme 31 décrit le calcul du chiffré d'un bit.

Algorithme 31 Chiffrement

Entrées : $m \in \{0,1\}$ et p.
Sortie : $\mathrm{E}(m) \in \mathbf{Z}_n^*$.

fonction CHIFFRER(m, p_k)
Générer un nombre aléatoire $r << p$
Générer un grand nombre aléatoire q
Calculer $r_1 = 2 \cdot \mathrm{R}$
Calculer $r_2 = p \cdot q$
Calculer $\mathrm{E}(m) = m + r_1 + r_2$.
retourner $\mathrm{E}(m)$.
fin fonction

Déchiffrement :
Pour déchiffrer un chiffré $c = \mathrm{E}(m)$, il suffit de l'évaluer modulo 2 et ensuite modulo la clé p. On calcule $m = (c \mod 2) \mod p$.

Algorithme 32 Déchiffrement

Entrées : Un chiffré c et la clé $s_k = p$.
Sortie : m.

fonction DÉCHIFFREMENT(c, s_k)
Calculer $\mathrm{A} = c \mod 2$,
Calculer $m = \mathrm{A} \mod p$.
retourner m.
fin fonction

Propriétés homomorphes :
Soit c_1 et c_2 deux chiffrés de m_1 et m_2, les propriétés 1.1 et 1.2 démontrent que le schéma de DGHV est homomorphe pour l'addition et la multiplication et vérifie : $c_1 \times c_2 = \mathrm{E}(m_1 \times m_2)$ et $c_1 + c_2 = \mathrm{E}(m_1 + m_2)$. Le bruit de la somme est la somme des bruits et celui du produit est le produit des bruits.

Propriété 1.1. Soient $m_1, m_2 \in \{0,1\}$, $c_1 = m_1 + 2 \cdot r_1 + p \cdot q_1$ et $c_2 = m_2 + 2 \cdot r_2 + p \cdot q_2$. La somme de c_1 et c_2 est $c_1 + c_2 = \mathrm{E}(m_1 + m_2)$.

$$\begin{aligned} c_1 + c_2 &= (m_1 + 2 \cdot r_1 + p \cdot q_1) + (m_2 + 2 \cdot r_2 + p \cdot q_2) \\ &= (m_1 + m_2) + 2(r_1 + r_2) + p(q_1 + q_2) \\ &= \mathrm{E}(m_1 + m_2). \end{aligned}$$

Propriété 1.2. Soient $m_1, m_2 \in \{0,1\}$, $c_1 = m_1 + 2 \cdot r_1 + p \cdot q_1$ et $c_2 = m_2 + 2 \cdot r_2 + p \cdot q_2$. Le produit de c_1 et c_2 est $c_1 \cdot c_2 = \mathrm{E}(m_1 \cdot m_2)$.

$$\begin{aligned} c_1 \cdot c_2 &= (m_1 + 2 \cdot r_1 + p \cdot q_1) \cdot (m_2 + 2 \cdot r_2 + p \cdot q_2) \\ &= (m_1 \cdot m_2) + 2(m_1 \cdot r_2 + r_1 \cdot m_2 + 2 \cdot r_1 \cdot r_2) + p(2 \cdot r_1 \cdot q_2 + 2 \cdot r_2 \cdot q_1 + m_1 \cdot q_2 + m_2 \cdot q_1 + p \cdot q_1 \cdot q_2) \\ &= \mathrm{E}(m_1 \cdot m_2). \end{aligned}$$

La version asymétrique de ce schéma est présentée ci-dessous :

Génération des clés :
La clé secrète est un entier impair de τ bits et la clé publique est constituée de τ entier. L'Algorithme 33 décrit les étapes de la génération de clés.

Algorithme 33 Génération de clés

fonction KEYGENERATION
Générer un entier p impair de τ bits,
pour $i \in \{0 \ldots \tau\}$ **faire**
Choisir un entier $q_i \leftarrow [0, \frac{2_\gamma}{p}[$
Choisir un entier $r_i \leftarrow]-2^\rho, 2^\rho[$
Calculer $x_i = p \cdot q_i + r_i$
fin pour
Classer les x_i dans l'ordre décroissant
retourner La clé secrète $s_k = p$ et la clé publique $p_k = (x_0, \ldots, x_\tau)$.
fin fonction

Chiffrement :
Pour chiffrer $m \in \{0, 1\}$ on calcule $\mathrm{E}(m, p_k) = (m + 2 \cdot Subset\,Sum(x_i) + 2 \cdot r) \bmod x_0$ où r est un nombre aléatoire $r \leftarrow]-2^\rho, 2^\rho[$ et $Subset\,Sum(x_i) = \Sigma_{i \in \{0 \ldots \tau\}} x_i$ est la somme d'éléments d'un sous-ensemble aléatoire de x_i.

Algorithme 34 Chiffrement

Entrées : $m \in \{0, 1\}$ et $p_k = (x_0, \ldots, x_l)$.
Sortie : $\mathrm{E}(m) \in \mathbf{Z}_n^*$.

fonction CHIFFRER(m, p_k)
Choisir un sous ensemble aléatoire de $(x_0, \ldots, x_\tau)$,
Calculer $Subset\,Sum$ la somme du sous ensemble choisit,
Choisir aléatoirement un entier $r \leftarrow]-2^\rho, 2^\rho[$,
Calculer $\mathrm{E}(m) = (m + 2 \cdot Subset\,Sum(x_i) + 2 \cdot r) \bmod x_0$,
retourner $\mathrm{E}(m)$.
fin fonction

Déchiffrement :
Pour déchiffrer un chiffré $c = \mathrm{E}(m)$, il suffit de l'évaluer modulo la clé secrète et ensuite modulo 2, on calcule ainsi $\mathrm{D}(c, s_k) = (c \bmod p) \bmod 2$.

Propriétés homomorphes :
Soient c_1 et c_2 deux chiffrés respectifs de m_1 et m_2. On peut facilement vérifier avec les propriétés 1.3 et 1.4 que : $c_1 + c_2 = \mathrm{E}(m_1 + m_2)$ et $c_1 \cdot c_2 = \mathrm{E}(m_1 \cdot m_2)$. Le bruit de la somme est la somme des bruits et celui du produit est le produit

des bruits.

Propriété 1.3. Soient $m_1, m_2 \in \{0, 1\}$, $c_1 = (m_1 + 2 \cdot Subset Sum_1(x_i) + 2 \cdot r_1) \mod x_0$ et $c_2 = (m_2 + 2 \cdot Subset Sum_2(x_i) + 2 \cdot r_2) \mod x_0$. La somme de c_1 et c_2 est:

$$\begin{aligned} c_1 + c_2 &= ((m_1 + 2 \cdot Subset Sum_1(x_i) + 2 \cdot r_1) + (m_2 + 2 \cdot Subset Sum_2(x_i) + 2 \cdot r_2)) \mod x_0 \\ &= ((m_1 + m_2) + 2(Subset Sum_1(x_i) + Subset Sum_2(x_i)) + p(r_1 + r_2)) \mod x_0 \\ &= E(m_1 + m_2). \end{aligned}$$

Donc le cryptosystème DGHV est homomorphe pour l'addition.

Propriété 1.4. Soient $m_1, m_2 \in \{0, 1\}$, $c_1 = (m_1 + 2 \cdot Subset Sum_1(x_i) + 2 \cdot r_1) \mod x_0$ et $c_2 = (m_2 + 2 \cdot Subset Sum_2(x_i) + 2 \cdot r_2) \mod x_0$. Le produit de c_1 et c_2 est:

$$\begin{aligned} c_1 \cdot c_2 &= ((m_1 + 2 \cdot Subset Sum_1(x_i) + 2 \cdot r_1) \cdot (m_2 + 2 \cdot Subset Sum_2(x_i) + 2 \cdot r_2)) \mod x_0 \\ &= (m_1 \cdot m_2) + 2 \cdot Subset Sum^t + 2r^t \\ &= E(m_1 \cdot m_2). \end{aligned}$$

Où $Subset Sum^t = m_1 \cdot Subset Sum_2 + Subset Sum_1 \cdot m_2 + Subset Sum_2 \cdot Subset Sum_1 + 2 \cdot Subset Sum_1 \cdot r_2 + Subset Sum_2 \cdot r_1$,
Et $r^t = m_1 \cdot r_2 + r_1 \cdot m_2 + 2 \cdot r_1 \cdot r_2$.
Donc le cryptosystème DGHV est homomorphe pour la multiplication.

1.3.7 Les schémas basés sur les problèmes LWE/RLWE

Le cryptosystème de Brakerski et Vaikuntanathan

En 2011, Brakerski et Vaikuntanathan ont décrit deux schémas fully homomorphe. Le premier est basé sur le problème RLWE [13], et le second sur le problème LWE [11]. Dans la version LWE de ce schéma, les auteurs ont introduit le processus de relinéarisation (ou changement de clé) permettant de construire un schéma somewhat homomorphe basé sur le problème LWE plutôt que sur des hypothèses de complexité relatives aux idéaux. Ils ont proposé aussi une technique de réduction de module permettant de diminuer la taille des chiffrés afin de gérer le bruit et réduire la complexité de la fonction de déchiffrement. Cette technique permet d'éviter l'appel de "squashing" [23] proposée par Gentry, qui permet de réduire la profondeur multiplicative du circuit de déchiffrement et qui lui a imposé une hypothèse de sécurité supplémentaire. Nous présentons ci-dessous la version asymétrique et basée sur le problème RLWE de ce cryptosystème [13].

Soient λ le paramètre de sécurité, $q > 1$ le module des coefficients, et $t > 1$ le module du texte en clair, tel que $t < q$. On note par $R_q = R/qR$ et $R_t R/tR$ l'espace des chiffrés et des plaintexts. $R_q[x] = \mathbf{Z}_q[x]/f(x)$ où $\mathbf{Z}_q[x]$ est l'anneau de polynôme à coefficients modulo q, $f(x) = x^d + 1$ et $d = 2^n$. Les éléments de $R_q[x]$ sont des polynômes de degré inférieur à d et à coefficients modulo q.
Les éléments de l'anneau $R_q[x]$ sont notés en minuscule ($a \in R_q[x]$), on note

par $[a]_q$ les éléments de R obtenus en calculant tous les coefficients modulo q. Pour tout $x \in R_q$ on note par $\lfloor x \rceil$ un nombre arrondi à l'entier le plus proche, $\lceil x \rceil$ et $\lfloor x \rfloor$ des nombres arrondis à l'entier supérieur et inférieur. Soient χ et χ^t deux distributions gaussiennes sur R_q définies respectivement par les déviations standards σ et σ^t, tel que $\sigma < \sigma^t$. La notation $x \leftarrow \chi$ est utilisée pour tirer aléatoirement x d'une distribution χ, et $x \xleftarrow{\$} \chi$ pour tirer uniformément x de χ.

Génération des clés :
La génération de la clé secrète, la clé publique et la clé d'évaluation est décrite dans l'Algorithme 35.

Algorithme 35 Génération de clés

fonction KEYGENERATION
Choisir uniformément la clé secrète $s_k \xleftarrow{\$} \chi$
Choisir uniformément $a_0 \xleftarrow{\$} R_q$
Choisir uniformément $e_0 \xleftarrow{\$} \chi$
Calculer $r_0 = a_0 \cdot s_k$
Calculer $r_1 = t \cdot e_0$
Calculer $b_0 = r_0 + r_1$,
retourner La clé secrète s_k et la clé publique $p_k = (a_0, b_0)$.
fin fonction

Chiffrement :
Soit $m \in R_t$ un message à chiffrer, $p_k = (a_0, b_0)$ la clé publique, le chiffrement de m est composé d'un couple de polynômes (c_0, c_1). L'Algorithme 36 décrit le chiffrement d'un message.

Algorithme 36 Chiffrement

Entrées : $m \in R_t$ et $p_k = (a_0, b_0)$
Sortie : $E(m) \in R_q^2$

fonction CHIFFRER(m, p_k)
Choisir uniformément $v \xleftarrow{\$} \chi$
Choisir uniformément $e \xleftarrow{\$} \chi^t$
Choisir uniformément $e^{tt} \xleftarrow{\$} \chi^t$
Calculer $r_0 = a_0 \cdot_t v$
Calculer $r_1 = t \cdot e$
Calculer $r_2 = b_0 \cdot_{tt} v$
Calculer $r_3 = t \cdot e$
Calculer $c_1 = -(r_0 + r_1)$
Calculer $c_0 = r_3 + r_4 + m$
retourner $E(m) = (c_0, c_1)$
fin fonction

Déchiffrement :
L'Algorithme 37 décrit le déchiffrement d'un chiff $c = (c_0, c_1)$ en utilisant une clé secrète s_k .

Algorithme 37 Déchiffrement

Entrées : c et s_k
Sortie : m

fonction DÉCHIFFREMENT(c, s_k)
Calculer $r_0 = c_1 \cdot s_k$
Calculer $m = (r_0 + c_0) \mod t$
retourner m
fin fonction

Propriétés homomorphes :
Soient c_1 et c_2 deux chiffrés respectifs de m_1 et m_2. On peut facilement vérifier avec les propriétés 1.5 que $c_1 + c_2 = \mathrm{E}(m_1 + m_2)$.

Propriété 1.5. Soient $m_1, m_2 \in R_t$, $c_1 = (b_0v_1 + te_1 + m_1, -a_0v_1 - te_1)$ et $c_2 = (b_0v_2 + te_2 + m_2, -a_0v_2 - te_2)$. La somme de c_1 et c_2 est $c_1 + c_2 = \mathrm{E}(m_1 + m_2)$.

$$\begin{aligned} c_1 + c_2 &= (b_0(v_1 + v_2) + t(e_1 + e_2) + (m_1 + m_2), -a_0(v_1 + v_2) - t(e_1 + e_2)) \\ &= \mathrm{E}(m_1 + m_2). \end{aligned}$$

La multiplication de deux chiffrés $c[1]$ et $c[2]$ consiste à calculer le produit tensoriel de $c[1] = (c[1][0], c[1][1])$ et $c[2] = (c[2][0], c[2][1])$. Le produit tensoriel augmente la taille du chiffré, donc on multiplie par une matrice différente après chaque produit.
Le produit de deux chiffrés est défini comme suit :

$$(c[1][0], c[1][1]) \cdot (c[2][0], c[2][1]) = (ct_0, ct_1, ct_2),$$

où

$$\begin{aligned} ct_0 &= [c[1][0] \cdot c[2][0]]_q, \\ ct_1 &= [(c[1][0] \cdot c[2][1] + c[1][1] \cdot c[2][0])]_q, \\ ct_2 &= [c[1][1] \cdot c[2][1]]_q. \end{aligned}$$

1.4 Synthèse sur l'état de l'art

Nous avons effectué un état de l'art des crypotsystèmes homomorphes les plus populaires. Nous avons commencé par les cryptosystèmes partiellement homomorphe, tel que RSA et ensuite les cryptosystèmes somewhat homomorphe, puis nous avons terminé par les cryptosystèmes fully homomorphe. De cet état de l'art il est sorti que les schémas basés sur le problème RLWE sont les plus intéressants pour une utilisation pratique. Parmis ces schémas on trouve le cryptosystème FV [21] et BGV [10]. Dans le chapitre 3, nous présenterons de manière détaillée ces deux schémas. Puis, nous effectuerons une comparaison de ces derniers.

Chapitre 2

Utilisation d'un secure element pour l'IoT

Sommaire

Au début de cette thèse, on a déposé un brevet [48] sur l'utilisation d'un secure element pour aider la cryptographie homomorphe à être déployée dans la pratique. Ce brevet a été déposé par Kontron, aux États-unis, au Canada et en Europe. Ce chapitre est une traduction que j'ai effectué d'un extrait de ce brevet.

2.1 Contexte

Au cours des dernières années, les systèmes IoT (Internet of Things) ont évolué en tant que systèmes d'objets physiques interdépendants dotés de capacités de calcul, de détection et de mise en réseau permettant aux objets de collecter et d'échanger des données sans nécessiter d'interaction d'humain à humain ou d'humain à ordinateur. Un système IoT permet aux objets physiques

d'être détectés et contrôlés de manière autonome, ce qui permet d'obtenir une intégration plus directe du monde physique dans les systèmes informatiques. Le terme "Things" au sens de l'IoT peut désigner une grande variété d'objets, tels que, par exemple, les personnes ayant des implants de moniteur cardiaque, les animaux munis de micropuces implantées, les automobiles munies de capteurs intégrés ou tout autre objet naturel ou fabriqué par l'homme qui peut se voir attribuer un numéro d'identification unique, généralement une adresse IP, et qui peut être fourni avec la possibilité de transférer des données sur un réseau.

Un système d'IoT comprend généralement des capteurs et des actionneurs qui fournissent et reçoivent des données à partir d'un cloud par des passerelles ou des agrégateurs de données.
Les moteurs d'analyse peuvent être utilisés pour analyser les données recueillies afin de prendre des décisions affectant et contrôlant les objets dans l'environnement IoT. Ces moteurs peuvent être des clouds analytics lorsque les processus d'analyse des données sont fournis par le biais d'un cloud public ou privé. Il peuvent être des noeuds de réseau aussi, les capteurs eux-mêmes ou d'autres dispositifs à l'extérieur du cloud (l'analytique en périphérie de réseau appelée edge analytics) sans qu'il soit nécessaire d'envoyer les données au cloud pour les analyser.

Prenons l'exemple des machines de fabrication industrielle, connectées à un système IoT, et qui envoient des données en quantités massives au Cloud. En analysant ces données par le biais du cloud analytics, les informations de contrôle peuvent être dérivées et appliquées aux machines pour préserver ou améliorer leur état de fonctionnement. De plus, une défaillance probable d'une partie spécifique d'une machine particulière peut être identifiée et la machine peut s'arrêter automatiquement. Ainsi une alerte, peut être envoyée à un responsable d'usine afin que la pièce puisse être remplacée ou que la panne soit réparée.
Généralement, le traitement de données est effectué sur des textes en clair, ce qui permet d'avoir des failles de sécurité permettant aux pirates informatiques ayant accès aux bases de données de prendre connaissance des données rapportées (confidentialité) et/ou pour compromettre silencieusement une base de données précieuse en injectant des données corrompues (intégrité) dans l'IoT.

2.2 Paramètres du système

Il est nécessaire de disposer d'une technique qui permet d'effectuer des analyses dans le cloud, dans un environnement IoT, sans exposer le texte en clair des données recueillies, peu importe si les données qui se déplacent sur le réseau, sont stockées temporairement ou en permanence dans un ordinateur principal ou en passant par les registres/caches/unités logiques arithmétiques du processeur.

Selon un premier aspect, il est nécessaire d'avoir un élément de calcul pour l'exécution sécurisée des opération sur les données chiffrés. Cet élément peut être un « secure element », possédant au moins un processeur et une mémoire.

2.2.1 Le secure element

Un secure element est un dispositif inviolable qui fournit un environnement de stockage et d'exécution sécurisé. Il assure généralement l'intégrité et la confidentialité de son contenu. Il peut être fourni comme un microcontrôleur indépendant (ex., puces)- souvent offert avec une certification de sécurité comme la FIPS-140-2 ou Common Criteria- et peut prendre différentes formes, par exemple sous forme d'une carte à puce, où la puce est intégrée dans une carte physique, sous la forme d'une UICC (Universal Integrated Circuit Card) ou d'une carte SIM et sou forme d'une carte à puce SD, où la puce est intégrée sur une carte SD. Les secures elements peuvent également être fournis en tant que secure element embarqué, où la puce est liée directement à une carte mère. Un secure element, héberge une ou plusieurs applications, communément appelées on-card-applications, qui peuvent interagir avec les applications off-card-applications, fourni sur l'hôte du secure element à travers une API (application programming interface). Contrairement au processeur, le secure element peut être assez simple et dédié aux tâches spécialisées afin qu'il soit possible d'obtenir une certification en démontrant qu'il ne peut pas être attaqué en interne ou par les méthodes connues de canaux cachés.

Ainsi, les données sont déchiffrés dans le secure element, dans lequel les opérations sur des données clairs seront effectuées, après avoir déchiffrés les textes chiff Le secure element va ensuite chiffrer les résultats pour les retourner au Cloud. De cette façon, on s'assure que les données déchiffrées ne sont jamais exposées en dehors de l'environnement sécurisé, fournie par le secure element. les données ne sont pas visibles par la machine de stockage et de traitement et, par conséquent, elle sont sécurisées même en cas d'atteinte à la sécurité de l'ordinateur.

Le secure element de l'unité de calcul peut fournir ainsi une API, qui peut être utilisée par l'unité de calcul pour interagir avec le secure element, tel que c'est décrit ici, par exemple, pour transmettre les données chiffrées (données individuelles ou plusieurs données) au secure element, et lui demander de déchiffrer ces données et effectuer en interne l'opération demandée sur les données déchiffrées, le secure element renvoi ainsi le résultat après l'avoir chiffré.

2.2.2 Le capteur

Le capteur qui collecte les données chiffrées dans l'environnement IoT peut être un capteur intelligent, par exemple, un capteur qui comprend des ressources intégrées permettant d'exécuter des fonctions prédéfinies, par exemple, les fonctions de traitement du signal ou les fonctions de chiffrement, avant de

transmettre les données détectées à un récepteur. Les données collectées par le capteur intelligent peuvent être chiffrées par le capteur lui-même et transmises à l'unité de calcul en conséquence. Si le capteur n'est pas un capteur intelligent, il ne peut détecter que des données environnementales, dans lesquelles les données détectées sont traitées et chiffrées par une unité de traitement qui est connectée au capteur. Les données chiffrées peuvent ensuite être transmises de l'unité de traitement à l'unité de calcul. Dans une variante spécifique, l'unité de traitement peut être l'unité de calcul elle-même, de sorte que le l'unité de calcul soit un capteur intelligent.

2.2.3 Les opérations

Les opérations demandées par l'unité de calcul peuvent faire partie d'un calcul analytique effectué dans l'environnement IoT. Dans les calculs analytiques, les opérations mathématiques peuvent typiquement être effectuées sur les données collectées par le capteur. Ainsi, l'opération demandée peut être une opération mathématique, telle qu'une simple opération arithmétique (ex., addition, multiplication, etc.). Avant de retourner le résultat de l'opération à l'unité calcul, le secure element peut chiffrer ce résultat pour que l'unité de calcul obtienne un résultat sous forme chiffrée.

Dans un autre type de demande au secure element, l'opération demandée peut être une opération de comparaison entre deux ou plusieurs données chiffrées. L'opération de comparaison peut comprendre des opérations telles que "égal à", "inférieur à", "supérieur à". Les parties comparées des données chiffrées peuvent comprendre, par exemple, une première partie qui a été obtenue à partir d'un premier capteur et une seconde partie qui a été obtenu à partir d'un deuxième capteur, ou bien la première et la deuxième partie peuvent avoir été reçue ultérieurement du même capteur. Le résultat de la comparaison peut être retourné à partir du secure element sans qu'il soit nécessaire d'appliquer un chiff t supplémentaire, de sorte que l'unité de calcul obtienne le résultat de l'opération de comparaison sous forme non chiffrée, en particulier sous forme de valeur booléenne non chiffrée. Bien que le retour de données non chiffrées puisse être considéré comme une fuite possible d'informations dans le cas d'une attaque, il peut parfois être utile d'avoir au moins un processeur dans l'unité de calcul pour effectuer des branchements conditionnels basés sur un résultat booléen non chiffré d'une comparaison. Dans ce cas, il peut être utile que l'API du secure element propose de travailler avec des opérateurs mathématiques chiffrés pour que le renvoi d'un texte clair booléen ne donnent pas beaucoup d'informations sur les opérandes des opérateurs.

2.2.4 L'unité de calcul

Il est donc évident que l'unité de calcul décrite ici doit être en mesure d'effectuer des opérations sur les données chiffrées sans jamais avoir accès au texte en clair des données chiffrées. L'unité de calcul peut être, par exemple, un cap-

teur intelligent, un commutateur, une passerelle ou un autre nœud du réseau. Alternativement, l'unité de calcul peut être physique ou virtuelle fourni dans le cloud. L'utilisation d'une ou de plusieurs unités de calcul de ce type permet éventuellement de fournir un chiffrement de bout en bout dans l'IoT et permet de chiffrer les données dans ou en dehors du capteur, ainsi que d'analyser des données dans le edge analytics (l'analytique en périphérie de réseau) ou dans le cloud analytics sans exposer le texte en clair à aucun moment. Une fois les données sont chiffrées à proximité de l'objet, toutes erreurs liées au facteur humain (mauvaise configuration des paramètres de sécurité, corruption d'individus, etc.) peuvent être évitées.

2.3 Les protocoles

2.3.1 La comparaison des données chiffrées

Dans le cas de l'opération de comparaison, l'unité de calcul peut être capable d'envoyer des alertes et/ou de contrôler les flux de programme vu que le résultat d'une opération de comparaison peut être visible par cet unité de calcul (seul le résultat peut être visible, et pas les données comparées elles-mêmes). En outre, au moins une mémoire de l'unité de calcul peut donc contenir des instructions exécutables par un processeur de telle sorte que l'unité de calcul puisse être utilisée pour déclencher une alerte en fonction du résultat de l'opération. Dans une requête d'API fournie par le secure element, l'opération peut être une opération de comparaison chiffrée. Prenons l'exemple suivant, 6537298 peut être le chiffrement du " égal à" et 22223333 peut être le chiffrement de l'opérateur "moins de", et ainsi de suite. En raison d'un facteur aléatoire dans le schéma de chiffrement utilisé, le même opérateur de comparaison est chiffré sous différent textes chiffrés avec le même algorithme. Dans ce cas, on ne peut ni savoir quel type de comparaison est effectué ni sur quelles données la comparaison est effectuée.

2.3.2 Les différents schémas de chiffrement

En général, les données chiffrées obtenues par l'unité de calcul peuvent être chiffrées avec un algorithme de chiffrement homomorphe ou non homomorphe. Les schémas de chiffrement non homomorphes peuvent être des cryptosystèmes symétriques bien connus avec une seule clé de chiffrement/déchiffrement, telles que l'AES [46], ou des algorithmes asymétriques avec une clé secrète et une clé publique, tels que le RSA [40], par exemple. Les schémas de chiffrement homomorphes sont connus depuis la thèse de doctorat de Craig Gentry "A Fully Homomorphic Encryption Scheme" de septembre 2009. Le chiffrement homomorphe est une forme de chiffrement qui permet d'effectuer des calculs directement sur les textes chiffrés. Un calcul homomorphe d'une opération (ex., addition, multiplication, etc.) effectuée sur des données chiffrées génère un résultat chiffré qui, une fois déchiffré, correspond au résultat de la même opération effectuée sur le texte en clair des données chiffrées. Lorsque les additions et les

multiplications peuvent être effectuées sur les textes chiffrés, le schéma de chiffrement est qualifié de Fully Homomorphic Encryption (FHE) (chiffrement complètement homomorphe). Toutes les opérations mathématiques peuvent alors être dérivées des opérateurs d'addition et de multiplication.

2.3.3 Le chiffrement homomorphe

L'application du chiffrement homomorphe nécessite généralement d'importantes ressources de traitement (en particulier, de la vitesse de traitement, de grandes ressources mémoire, et, par conséquent, la vitesse du réseau) qui peuvent seulement exister dans les environnements de cloud, même pour des opérations simples. Ainsi, si les données sont chiffrées avec un schéma de chiffrement homomorphe, les calculs homomorphes nécessitant d'importantes ressources peuvent être évités ou accélérés par l'application des mesures décrites ci-dessus : Il s'agit d'une technique d'exécution d'opérations sur les données chiffrées par l'intermédiaire du secure element fixé à l'unité de calcul (qui peut être intégré à l'unité de calcul, ou distant). Cela peut être particulièrement appliqué dans le cas du rafraîchissement des textes chiffrés. Le rafraîchissement (Bootstrapping) fait partie du schéma de Gentry dans lequel il est possible de calculer un nombre arbitraire d'additions et de multiplications en rafraîchissant le texte chiffré périodiquement chaque fois que le bruit devient trop grand pour permettre un déchiffrement correcte. Dans ces cas, chaque calcul de rafraîchissement homomorphe peut être remplacé par un simple déchiffrement suivi d'un nouveau chiffrement des données dans le secure element. De plus, dans le cas d'une opération de comparaison, il est évident que même si le résultat d'une comparaison peut être calculé en cryptographie homomorphe - le résultat de la comparaison (p. ex. résultat booléen) est toujours chiffré, donc ce résultat ne peut pas être utilisé pour déclencher des alertes et/ou faire des branchements conditionnels. De telles mesures peuvent être réalisées par la technique décrite ci-dessus en effectuant les opérations dans le secure element.

2.3.4 La gestion des clés de chiffrement

Afin de déchiff et rechiffrer les données dans les schémas homomorphes et non homomorphes, le secure element peut stocker une clé de déchiffrement et une clé de chiffrement. Ces clés peuvent être fournies au secure element par l'unité de calcul lors de la mise en service ou pendant le processus de démarrage, par exemple, à l'aide d'un algorithme standard de distribution de clés, tel que les schémas de chiffrement asymétrique ou l'algorithme Diffie-Hellman.

2.3.5 Quelques contre-mesures nécessaires

Dans d'autres implémentations, en particulier lorsque les données sont chiffrées par des cryptosystèmes déterministes, certaines mesures peuvent être prises afin d'empêcher de deviner les valeurs en clair des données chiffrées. Pour cela la valeur d'un texte clair peut être complétée par une ou plusieurs valeurs de

rembourrage aléatoire avant de générer le texte chiffré, c'est-à-dire avant que la donnée ne soit réellement chiffrée. De cette façon, il est possible d'éviter que le chiffrement de numéros (tels que 0, 1, 2, ect.) puisse être deviné par un attaquant en se basant sur des calculs comme "$x - x$" ou "y/y" dans le secure element. Dans le même but, le secure element peut être configuré pour rejeter, ou pour ignorer silencieusement l'opération et marquer le résultat comme invalide, si l'opération demandée est prédéfinie comme une opération qui permet de deviner le texte clair d'une donnée chiffrée (ex., "$x - x$" ou "y/y").

2.3.6 L'intégrité des données

En outre, pour protéger l'intégrité d'une donnée ou d'un ensemble d'éléments de données, une somme de contrôle ou une valeur de hachage peuvent être ajoutés avant le chiffrement initial ou le rechiffrement du résultat, de sorte qu'il n'est pas possible d'introduire dans une base de données une donnée sans posséder la clé de chiffrement/déchiffrement à proximité du capteur ou dedans.

Par ailleurs, même si une nouvelle donnée ne peut être générée par un pirate informatique, l'intégrité peut être menacée dans le cas où des données chiffrées existantes sont dupliquées ou corrompues. Pour éviter cette possibilité, un champ de traçabilité peut être ajouté à chaque élément de données (avant le chiffrement/rechiffrement), en stockant initialement un numéro de séquence de données et/ou un numéro d'appareil unique. Chaque fois qu'une opération est effectuée sur un ou plusieurs éléments de données, conduisant à une donnée résultante, une opération (identique ou alternative comme un simple ajout qui pourrait inclure un opérateur codé) peut être effectuée sur le champ de traçabilité, de sorte que les données résultantes soient balisées avec le résultat de l'opération effectuée sur le champ de traçabilité, l'étiquette est stockée sur le même champ de traçabilité. De cette façon, une traçabilité correcte des opérations effectuées sur les données attendues peut être obtenue en même temps que chaque fin de calcul. Cela permet de vérifier que le résultat final a été obtenu par le traitement de l'ensemble de données correctes demandées. La vérification finale du champ de traçabilité peut être effectuée à l'intérieur du secure element avant d'entrer les données résultantes dans la base de données chiffrées.

Dans d'autres implémentations, le secure element peut être configuré pour appliquer un minuteur qui déclenche sa réinitialisation si aucune clé cryptographique correcte ne lui a été fournie par l'unité de calcul avant l'expiration du minuteur. Le minuteur peut être utilisé comme une horloge de surveillance qui (par exemple, après une période de temps prédéfinie après l'initialisation ou le processus de démarrage de l'unité de calcul et/ou sur des périodes de temps régulières par la suite) déclenche une réinitialisation du secure element (et optionnellement de l'unité de calcul également) pour éviter l'analyse des données chiffrées, la possibilité d'exporter les données, la possibilité de modifier un programme, et d'empêcher le reciblage de la plate-forme de l'unité de calcul

à un autre but.

Il ressort de ce qui précède que l'utilisation d'un secure element pour effectuer en toute sécurité une opération sur des données chiffrées est avantageuse à plusieurs égards. Il sera toutefois entendu qu'en l'absence d'un secure element dans l'unité de calcul, la réalisation d'une opération sur des données chiffrées est toujours possible grâce à l'utilisation de la cryptographie homomorphe, mais avec une charge de travail potentiellement beaucoup plus lourde.

2.3.7 Unité de calcul supplémentaire

Selon un deuxième aspect, une autre unité de calcul est fournie pour l'exécution sécurisée d'une opération sur données chiffrées dans un environnement Internet des objets (IoT). L'unité de calcul comprend au moins un processeur et une mémoire. La mémoire contient des instructions exécutables par le processeur de sorte que l'unité de calcul puisse obtenir des données chiffrées collectées par un capteur prévu à cet effet dans l'environnement IoT, ces données sont chiffrées avec un schéma homomorphe, et effectuer un calcul homomorphe sur les données chiffrées pour générer un résultat chiffré qui, lorsqu'il est déchiffré, correspond à un résultat de l'opération effectuée sur le texte en clair des données chiffrées.

Les caractéristiques décrites ci-dessus en ce qui concerne l'unité de calcul du premier aspect sont applicables à l'unité de calcul dans un second aspect et peuvent être constituées par l'unité de calcul du second aspect. Ceci s'applique en particulier au capteur qui recueille les données chiffrées et aux caractéristiques d'utilisation des schémas de chiffrement homomorphes. Les répétitions inutiles sont ainsi omises. L'opération réalisée par l'intermédiaire du calcul homomorphe peut être un calcul mathématique telle qu'une simple opération arithmétique (par ex, addition, multiplication, etc.).

2.4 Conclusion

Dans ce brevet nous avons défini un exemple complet du stockage et d'analyse de données dans l'IoT, où on utilise la cryptographie homomorphe ou un schéma de cryptographie classique pour garantir la sécurité des données à partir des capteurs jusqu'au cloud.

La contribution majeure de ce brevet est d'utiliser un secure element dans ces deux types de chiffrement. Pour la cryptographie homomorphe, le secure element permet d'effectuer des comparaisons sur des données chiffrées pour aider le système dans le cloud à prendre des décisions et ainsi envoyer des alertes à la machine de l'administrateur.

On peut utiliser aussi le secure element pour éviter d'avoir à faire appel à la fonction de rafraîchissement qui est trop coûteuse. La fonction de rafraîchissement appelée aussi Bootstrapping permet de transformer un chiffré bruité

issu d'un circuit d'addition et/ou multiplication à un chiffré contenant un bruit équivalent à celui d'un texte fraîchement chiffré. En conséquence, le secure element peut permettre d'éviter ce rafraîchissement en déchiffrant le chiffré et en le chiffrant ensuite.

Si on utilise un schéma de chiffrement classique, comme le RSA ou l'AES, le secure element permet dans ce cas d'effectuer des calculs sur des données chiffrées, sans avoir à les dévoiler au cloud. En pratique, le secure element va déchiffrer les chiffrés, effectuer les calculs demandés et puis chiffrer le résultat avant de l'envoyer au cloud.

Chapitre 3

Comparaison des cryptosystèmes FV et BGV

Sommaire

L'étude de l'état de l'art nous a conduit à déduire que les cryptosystèmes homomorphes basés sur le problème RLWE sont les plus adaptés à une utilisation concrète. Dans ce chapitre nous décrivons les cryptoystèmes BGV et FV, ainsi que les différentes librairies existantes qui implémentent ces deux schémas. Nous comparons aussi ces deux cryptosystèmes en nous basant sur les performances des librairies par rapport aux différents paramètres générés en utilisant l'estimateur de Martin Albrecht. Cette étude de paramètres et l'analyse des deux schémas va nous permettre de choisir par la suite le cryptosystème le plus adapté à notre cas d'usage dans l'IoT.

3.1 Notations

La structure algébrique utilisée par le problème RLWE [34] est l'anneau de polynôme $R = Z[x]/f(x)$, où $Z[x]$ est l'anneau de polynôme à coefficients dans Z et $f(x)$ un polynôme cyclotomique de degré d. En pratique, on choisit $f(x) = x^d + 1$ et $d = 2^n$. Les éléments de R sont des polynômes de degré inférieur à d et à coefficients dans Z. Soit q le module des coefficients, $R_q[x] = Z_q[x]/f(x)$ où $Z_q[x]$ est l'anneau de polynôme à coefficients modulo q. Les éléments de $R_q[x]$ sont des polynômes de degré inférieur à d et à coefficients modulo q.
Les éléments de l'anneau $R_q[x]$ sont notés en minuscule ($a \in R_q[x]$), on note par $[a]_q$ les éléments de R obtenus en calculant tous les coefficients modulo q. Pour tout $x \in R_q$ on note par $\lfloor x \rceil$ un nombre arrondi à l'entier le plus proche, $\lceil x \rceil$ et $\lfloor x \rfloor$ des nombres arrondis à l'entier supérieur et inférieur. Soit D une distribution, la notation $x \leftarrow D$ est utilisée pour tirer aléatoirement x d'une distribution D, et $x \xleftarrow{\$} D$ pour tirer uniformément x de D.

3.2 Le cryptosystème de Fan et Vercauteren (FV)

Le cryptosystème Somewhat homomorphe de Fan et Vercauteren (FV) [21] a été développé en 2012, sa sécurité est basée sur le problème RLWE.
Soit λ le paramètre de sécurité, $q > 1$ le module des coefficients utilisé pour définir l'anneau de polynôme R_q, et $t > 1$ le module du texte en clair, où $t < q$. On note par R_q et R_t l'espace des chiffrés et des plaintexts.
Soit $params$ l'ensemble de paramètres du schéma, $params = (R, d, q, t, \chi_{err}, \chi_{key})$.

La génération des clés :
Le cryptosystème FV est un schéma à clé publique, la génération de clé retourne une clé publique p_k et une clé secrète s_k. La clé secrète s_k est un polynôme tiré aléatoirement dans la distribution χ_{key}, $s_k \leftarrow \chi_{key}$, en pratique $\chi_{key} = R_2$ et s_k est un polynôme à coefficients binaires et à degré inférieur à d. Pour générer la clé publique, on commence par choisir a aléatoirement de R_q $a \xleftarrow{\$} R_q$, et une erreur aléatoire e de χ_{err} $e \leftarrow \chi_{err}$. La clé publique p_k est un couple de deux polynômes,

$$p_k = ([-(a \cdot s_k + e)]_q, a).$$

Soit rlk une clé de relinéarisation utilisée pour réduire la taille du chiffré après la multiplication des chiffrés. Cette clé est composée de l sous clés rlk_i :

$$rlk[i] = ([-(a_i \cdot s_k + e_i) + T^i \cdot s^2]_q, a_i),$$

où $l = \lfloor log_T(q) \rfloor$, T est un entier aléatoire indépendant du module du texte en clair t et pour tout $i = 0 \ldots l$ on choisit a_i de R_q $a_i \xleftarrow{\$} R_q$, on choisit e_i dans χ_{err} $e_i \xleftarrow{\$} \chi_{err}$,. La génération des clés est décrite dans l'Algorithme 38.

Algorithme 38 Génération des clés pour FV

Input $params = (R, d, q, t, \chi_{err}, \chi_{key})$.
Output s_k, p_k, rlk.

fonction GÉNÉRATION DE CLÉS($params$)
$s_k \xleftarrow{\$} \chi_{key}$.
$a \xleftarrow{\$} R_q$.
$e \leftarrow \chi_{err}$.
Calculer $r_1 = a \cdot s_k \mod q$.
Calculer $r_2 = (r_1 + e) \mod q$.
Calculer $r_3 = -r_2 \mod q$.
$p_k = (r_3, a)$.
Choisir T un entier indépendant de t.
Calculer $l = \lfloor \log_T(q) \rfloor$.
pour $i = 1$ à l **faire**
$a_i \xleftarrow{\$} R_q$.
$e_i \leftarrow \chi_{err}$.
Calculer $r_1 = ai \cdot s_k \mod q$.
Calculer $r_2 = (r_1 + e_i) \mod q$.
Calculer $r_3 = -r_2 \mod q$.
Calculer $r_4 = T^i \mod q$.
Calculer $r_5 = s_k^2 \mod q$.
Calculer $r_6 = r_4 \cdot r_5 \mod q$.
Calculer $r_7 = (r_3 + r_6) \mod q$.
Calculer $rlk[i] = (r_7, a_i)$.
fin pour
retourner s_k, p_k et rlk.
fin fonction

Le chiffrement :

Pour chiffrer un message $m \in R_t$, on commence par calculer $\delta = \lfloor \frac{q}{t} \rfloor$, on choisit $u \leftarrow \chi_{key}$, et $e_1, e_2 \leftarrow \chi_{err}$.
Le chiffrement de m est un chiffré de deux polynômes défini comme suit :

$$E(m) = ([p_0 \cdot u + e_1 + \delta \cdot m]_q, [p_1 \cdot u + e_2]_q). \tag{3.1}$$

On remarque que le chiffré est un couple de données, le premier élément peut être considéré comme le chiffré, et le deuxième comme un bruit. La fonction de chiffrement est décrite dans l'Algorithme 39.

Algorithme 39 Chiffrement d'un message avec FV

Input $m \in R_t$ and p_k.
Output $E(m) = (c[0], c[1])$.

fonction CHIFFREMENT(m, p_k)
Calculer $\delta = \lfloor \frac{q}{t} \rfloor$
Tirer u de χ_{key} $u \leftarrow \chi_{key}$
Tirer e_1 de χ_{err} $e_1 \leftarrow \chi_{err}$
Tirer e_2 de χ_{err} $e_2 \leftarrow \chi_{err}$
Calculer $c[0] = [p_0 \cdot u]_q$
Calculer $r_0 = [c[0] + e_1]_q$
Calculer $c[0] = [r_0 + \delta \cdot m]_q$
Calculer $r_0 = [p_1 \cdot u]_q$
Calculer $c[1] = [r_0 + e_2]_q$
$E(m) = (c[0], c[1])$
retourner $E(m)$
fin fonction

Le déchiffrement :
Pour déchiffrer le texte chiffré $C = (c[0], c[1])$ on évalue l'équation suivante :

$$D(C) = [\lfloor \frac{t \cdot [c[0] + c[1] \cdot s_k]_q}{q} \rceil]_t.$$

L'Algorithme 40 décrit la fonction de déchiffrement.

Algorithme 40 déchiffrement d'un chiffré avec FV

Input $C = (c[0], c[1])$ et s_k.
Output $D(C)$.

fonction DÉCHIFFREMENT(C, s_k)
$r_0 = c[1] \cdot s_k$
$D(C) = [c[0] + r_0]_q$
$r_0 = t \cdot D(C)$
$D(C) = [\lfloor \frac{r_0}{q} \rceil]_t$
retourner $D(C)$
fin fonction

L'addition :
L'addition des deux chiffrés $c[1] = (c[1][0], c[1][1])$ et $c[2] = (c[2][0], c[2][1])$ est réalisée en calculant la somme de deux polynômes avec une réduction modulaire. Le résultat de la somme des chiffrés est

$$c[1] + c[2] = ([c[1][0] + c[2][0]]_q, [c[1][1] + c[2][1]]_q). \quad (3.2)$$

L'équation suivante démontre que la somme de deux chiffrés est égale au chiffrement de la somme de plaintexts correspondants.

$$[c[1][0] + c[2][0]]_q = [p_0 \cdot (u_1 + u_2) + (e_1 + e_1^t) + \delta \cdot (m_1 + m_2)]_q,$$

$$[c[1][1] + c[2][1]]_q = [p_1 \cdot (u_1 + u_2) + (e_2 + e_2^t)]_q.$$

Algorithme 41 Addition des chiffrés pour FV

Input $c[1] = (c[1][0], c[1][1])$ et $c[2] = (c[2][0], c[2][1])$.
Output $S = c[1] + c[2]$.

fonction ADDITION($c[1]$, $c[2]$)
  $S[0] = c[1][0] + c[2][0]$
  $S[1] = c[1][1] + c[2][1]$
retourner $S = (S[0], S[1])$.
fin fonction

La multiplication($c[1]$, $c[2]$) :
La multiplication de deux chiffrés $c[1]$ et $c[2]$ consiste à calculer le produit tensoriel de $c[1] = (c[1][0], c[1][1])$ et $c[2] = (c[2][0], c[2][1])$ et puis de le multiplier par t/q. Le produit tensoriel augmente la taille du chiffré.
Le produit de deux chiffrés est défini comme suit :

$$(c[1][0], c[1][1]) \cdot (c[2][0], c[2][1]) = (ct_0, ct_1, ct_2),$$

où

$$ct_0 = [l\frac{t \cdot (c[1][0] \cdot c[2][0])}{q}l]_q,$$
$$ct_1 = [l\frac{t \cdot (c[1][0] \cdot c[2][1] + c[1][1] \cdot c[2][0])}{q}l]_q,$$
$$ct_2 = [l\frac{t \cdot (c[1][1] \cdot c[2][1])}{q}l]_q.$$

Le résultat de la multiplication est un triplet (ct_0, ct_1, ct_2), l'Algorithme 42 décrit la procédure de multiplication. Pour transformer un chiffré de trois éléments (ct_0, ct_1, ct_2) en un chiffré de deux éléments (ct_0^t, ct_1^t), on utilise la technique de relinéarisation.

Algorithme 42 Multiplication des chiffrés pour FV

Input $c[1] = (c[1][0], c[1][1])$ et $c[2] = (c[2][0], c[2][1])$.
Output $\mathrm{M} = c[1] \cdot c[2]$.

fonction MULTIPLICATION($c[1]$, $c[2]$)
$ct_0 = t \cdot (c[1][0] \cdot c[2][0])$
$r_0 = \frac{ct_0}{q}$
$ct_0 = [lr_0l]_q$

$ct_1 = c[1][0] \cdot c[2][1]$
$r_0 = ct_1 + c[1][1] \cdot c[2][0]$
$ct_1 = t \cdot r_0$
$r_0 = \frac{ct_1}{q}$
$ct_1 = [lr_0l]_q$

$ct_2 = t \cdot (c[1][1] \cdot c[2][1])$
$r_0 = \frac{ct_2}{q}$
$ct_2 = [lr_0l]_q$
retourner $\mathrm{M} = (ct_0, ct_1, ct_2)$
fin fonction

La relinéarisation$(params, rlk, (ct_0, ct_1, ct_2))$**:**
L'étape de relinéarisation réduit le nombre d'éléments d'un chiffré issu d'une multiplication, en transformant un chiffré à trois éléments en un chiffré de deux éléments. La relinéarisation peut être utilisée après chaque multiplication. On commence par écrire ct_2 en base T, $ct_2 = \sum_{i=0}^{l} c[2]^{(i)} \mathrm{T}^i$ où $c[2]^{(i)} \in R_t$, et on utilise la clé de relinéarisation pour calculer le nouveau chiffré (ct_0^t, ct_1^t).

$$ct_0^t = [ct_0 + \sum_{i=0}^{l} rlk[i][0] \cdot c[2]^{(i)}]_q,$$
$$ct_1^t = [ct_1 + \sum_{i=0}^{l} rlk[i][1] \cdot c[2]^{(i)}]_q.$$

Le bruit du chiffré augmente après l'étape de relinéarisation. Ainsi pour éviter de faire appel à cette méthode dans le cas d'un circuit de profondeur un, on peut garder le chiffré avec trois éléments (ct_0, ct_1, ct_2) et le déchiff directement :

$$\mathrm{D}(ct_0, ct_1, ct_2) = [l\frac{t \cdot [ct_0 + ct_1 \cdot s_k + ct_2 \cdot s_k^2]_q}{q}l]_t.$$

Algorithme 43 Relinéarisation du chiffré pour FV

Input (ct_0, ct_1, ct_2).
Output (ct'_0, ct'_1).

fonction RELINÉARISATION$(rlk, (ct_0, ct_1, ct_2))$
Décomposer ct_2 dans la base T : $ct_2 = \sum_{i=0}^{l} c[2]^{(i)} T^i$ où $c[2]^{(i)} \in R_T$.
Calculer $ct'^{t}_0 = [ct_0 + \sum_{i=0}^{l} rlk[i][0] \cdot c[2]^{(i)}]_q$
Calculer $ct'^{t}_1 = [ct_1 + \sum_{i=0}^{l} rlk[i][1] \cdot c[2]^{(i)}]_q$
retourner (ct'^{t}_0, ct'^{t}_1)
fin fonction

3.2.1 La librairie FV-NFLlib

FV-NFLlib [43] est une librairie qui implémente le cryptosystème FV, développée en 2016 en C++, et basée sur la librairie NFLlib [35] spécialisée dans la cryptographie sur les réseaux idéaux. Dans cette section, nous allons présenter dans le Tableau 3.7 et 3.8, les performances de cette bibliothèque et les ressources mémoires nécessaires pour le stockage des clés et des chiffrés.

Opération	$d = 2048$, $\|q\| = 62$, $L = 1$	$d = 4096$, $\|q\| = 124$, $L = 2$	$d = 8192$, $\|q\| = 257$, $L = 6$
Génération de la Clé secrète (μs)	226	618	2740
Génération de la Clé publique (μs)	398	1592	5145
Génération de la clé de relinéarisation (μs)	656	233481	37645
Chiffrement (μs)	309	1092	4497
Addition (μs)	15	51	420
Multiplication (μs)	4700	10676	42881
Relinéarisation (μs)	415	1631	12306
Déchiffrement (μs)	739	2326	5242

TABLEAU 3.1 – Performances de la librairie FV-NFLlib (Intel(R) Core(TM) i5 at 2.4 GHz).

Structure	$d = 2048$, $\|q\| = 62$, $L = 1$	$d = 4096$, $\|q\| = 124$, $L = 2$	$d = 8192$, $\|q\| = 257$, $L = 6$
Clé secrète (K*octets*)	16	64	254
Clé publique (K*octets*)	32	127	508
Clé de relinéarisation (K*octets*)	32	127	508
Chiffré (K*octets*)	32	127	508

TABLEAU 3.2 – Taille des clés et des chiffrés dans la librairie FV-NFLlib

3.2.2 La librairie SEAL

SEAL [42] est une librairie Open Source, développée en 2015 en C++ et C# par une équipe Microsoft. Elle implémente le cryptosystème FV sans dépendances externes d'autres bibliothèques. Dans SEAL, l'utilisateur choisit le niveau de sécurité : 128 ou 192, le module du texte en clair et le degré du polynôme cyclotomique : 1024, 2048, 4096 et 8192. Le programme retourne la taille des coefficients des polynômes et le budget de bruit après chaque opération effectuée sur les chiffrés.
Les tableaux 3.3 et 3.4 présentent les performances et les tailles qu'on a obtenu en utilisant cette librairie.

Opération	$d = 2048$, $\|q\| = 56$, $L = 1$	$d = 4096$, $\|q\| = 110$, $L = 2$	$d = 8192$, $\|q\| = 219$, $L = 6$	$d = 16384$, $\|q\| = 441$, $L = 11$
Génération de la Clé secrète (µs)	5870	738	12989	34297
Génération de la Clé publique (µs)	21013	43998	143150	631134
Génération de la clé de relinéarisation (µs)	96560	361343	2425150	19011300
Chiffrement (µs)	26716	47256	97628	222531
Addition (µs)	11	62	152	691
Multiplication (µs)	4914	18142	67118	295329
Relinéarisation (µs)	349	3895	30548	256176
Déchiffrement de (c_0, c_1) (µs)	471	1463	3987	24833
Déchiffrement de (c_0, c_1, c_2) (µs)	558	—	—	—

TABLEAU 3.3 – Performances de la librairie SEAL(Intel(R) Core(TM) i5 at 2.4 GHz).

Opération	$d = 2048$, $\|q\| = 56$, L = 1	$d = 4096$, $\|q\| = 110$, L = 2	$d = 8192$, $\|q\| = 219$, L = 6	$d = 16384$, $\|q\| = 441$, L = 11
Clé secrète (K*octets*)	16	65	262	1048
Clé publique (K*octets*)	32	131	524	2097
Clé de relinéarisation (K*octets*)	32	131	524	2097
Chiffré (c_0, c_1) (K*octets*)	32	131	524	2097
Chiffré (c_0, c_1, c_2) (K*octets*)	49	—	—	—

TABLEAU 3.4 – Taille des clés et des chiffrés dans la librairie SEAL

3.3 La version RNS du cryptosystème de Fan et Vercauteren

La version RNS du Cryptosystème FV [6] est une implémentation du schéma FV qui utilise une représentation RNS, qu'on va définir ci-dessous, des coefficients des polynômes. Chaque élément de l'anneau R_q est représenté sous forme d'un vecteur de plusieurs éléments dans R_{q_i} où les q_i sont des petits modules. Cette technique permet d'optimiser le stockage des coefficients et d'accélérer les calculs.

3.3.1 RNS : Residue Number System

Le système modulaire de représentation [47], est un système de numération non positionnel, basé sur le théorème des restes chinois (CRT). Le RNS permet de représenter les nombres par leurs restes modulo un ensemble d'entiers premiers entre eux.

Théorème 3.1 (des restes chinois). Soient $n_1, \ldots, n_k$ des entiers naturels non nuls, premiers entre eux, tels que $\forall i \neq j \; pgcd(n_i, n_j) = 1$ et $n = \prod_{i=1}^{k} n_i$. L'application ϕ défini ci-dessous est un isomorphisme d'anneaux :

$$\begin{aligned} \phi \; : \; \mathbf{Z}/n\mathbf{Z} &\to \mathbf{Z}/n_1\mathbf{Z} \times \cdots \times \mathbf{Z}/n_k\mathbf{Z} \\ x &\mapsto (x \bmod n_1, \ldots, x \bmod n_k). \end{aligned}$$

Ainsi, pour tout élément $x \in \mathbf{Z}/n\mathbf{Z}$, il existe un unique ensemble d'éléments $(x_1, \ldots, x_k) \in \mathbf{Z}/n_1\mathbf{Z} \times \ldots \mathbf{Z}/n_k\mathbf{Z}$ tels que :

$$\begin{cases} x \equiv x_1 \pmod{n_1} \\ \vdots \\ x \equiv x_k \pmod{n_k}. \end{cases}$$

Soit $\hat{n}_i = \frac{n}{n_i}$ pour tout $0 \leqslant i < n$ tel que pgcd$(\hat{n}_i, n_i) = 1$, donc, d'après le théorème de Bézout, il existe deux entiers u_i et v_i tels que : $n_i u_i + \hat{n}_i v_i = 1$. L'algorithme d'Euclide étendu permet de calculer ces entiers. L'unique entier $x \in \mathbf{Z}/n\mathbf{Z}$ pour résoudre ce système est donc :

$$x = \left(\sum_{i=0}^{n-1} x_i t_i\right) \bmod n,$$

tel que $t_i = \hat{n}_i v_i$, $t_i \equiv 1 \pmod{n_i}$ et $t_i \equiv 0 \pmod{n_j}$, pour $i \neq j$.

Le tuple $(n_1, \ldots, n_k)$ du théorème 3.1 est une base RNS pour l'entier n. Avec cette base, la représentation RNS d'un entier $x \in \mathbf{Z}/n\mathbf{Z}$ est le tuple $(x \bmod n_1, \ldots, x \bmod n_k)$. Représenter les éléments de $\mathbf{Z}/n\mathbf{Z}$ de cette manière permet d'optimiser le stockage des coefficients et d'accélérer les calculs.

3.3.2 Les opérations arithmétiques dans RNS

Cette décomposition permet de transformer les opérations arithmétiques de base dans $\mathbf{Z}/n\mathbf{Z}$ en des opérations équivalentes et indépendantes, réalisés d'une manière parallèle, dans les anneaux $\mathbf{Z}/n_i\mathbf{Z}$, sans avoir de mécanisme de propagation de retenue à gérer.

Propriété 3.1. Soient $(a_0, \ldots, a_{n-1})$ et $(b_0, \ldots, b_{n-1})$ les représentations RNS respectives de $a, b \in \mathbf{Z}/n\mathbf{Z}$.
Le calcul en RNS de la somme est représenté par :

$$((a_0 + b_0) \bmod n_0, \ldots, (a_{n-1} + b_{n-1}) \bmod n_{k-1}),$$

et la représentation RNS du produit $(ab) \bmod n$ est :

$$((a_0 b_0) \bmod n_0, \ldots, (a_{n-1} b_{n-1}) \bmod n_{k-1}).$$

3.3.3 La librairie SEAL-RNS

Cette version de la librairie SEAL conserve les mêmes paramètres que la version initiale de SEAL. En utilisant le même matériel que les librairies ci-dessus, nous obtenons les résultats présentés dans le Tableau 3.5 et 3.6.

Opération	$d = 2048$, $\|q\| = 54$, L = 1	$d = 4096$, $\|q\| = 110$, L = 2	$d = 8192$, $\|q\| = 219$, L = 6	$d = 16384$, $\|q\| = 441$, L = 11
Génération de la Clé secrète (μs)	9138	6579.61	13434	29413
Génération de la Clé publique (μs)	19875	55711.8	114649	346418
Génération de la clé de relinéarisation (μs)	76629	281265	1314920	6539110
Chiffrement (μs)	23164	46923	92497	184910
Addition (μs)	11	97	108	331
Multiplication (μs)	2675	10522	29419	95559
Relinéarisation (μs)	440	1176	11490	42240
Déchiffrement de (c_0, c_1) (μs)	257	786	2844	6534
Déchiffrement de (c_0, c_1, c_2) (μs)	275	—	—	—

TABLEAU 3.5 – Performances de la librairie SEAL-RNS (Intel(R) Core(TM) i5 at 2.4 GHz).

Structure	$d = 2048$, $\|q\| = 54$, L = 1	$d = 4096$, $\|q\| = 110$, L = 2	$d = 8192$, $\|q\| = 219$, L = 6	$d = 16384$, $\|q\| = 441$, L = 11
Clé secrète (K*octets*)	16	65	262	1048
Clé publique (K*octets*)	32	131	524	2097
Clé de relinéarisation (K*octets*)	32	131	524	2097
Chiffré (c_0, c_1) (K*octets*)	32	131	524	2097
Chiffré (c_0, c_1, c_2) (K*octets*)	49	—	—	—

TABLEAU 3.6 – Taille des clés et des chiffrés dans la librairie SEAL-RNS

3.4 Comparaison des librairies SEAL et FV-NFLlib

Dans SEAL et SEAL-RNS, le déchiffrement du résultat d'une multiplication de chiffrés peut être effectué par deux méthodes différentes. La méthode standard, où l'on multiplie deux chiffrés, on appelle la procédure de relinéarisation et puis on déchiffre, et la seconde méthode qui consiste à déchiffrer directement le résultat d'une multiplication. Soit $(c[0], c[1], c[2])$ le résultat du produit de deux chiffrés et s_k une clé secrète, pour déchiffrer ce chiffré on calcule $c[0] + c[1] \cdot s + c[2] \cdot s^2$. La première méthode est utilisée lorsque la profondeur du circuit est un et la seconde méthode est utilisée lorsque nous avons besoin d'évaluer un circuit de plus d'une multiplication, par contre FV-NFLlib seulement juste la deuxième méthode.
Notons que la bibliothèque FV-NFLlib (Tableau 3.7) est plus rapide que les bibliothèques SEAL (Tableau 3.3), mais FV-NFLlib ne supporte pas les circuits

avec des profondeurs élevées, le niveau maximum supporté par cette bibliothèque est 6. Au niveau des tailles des clés et des chiffrés, elles sont de même ordre de grandeur dans les deux librairies FV-NFLlib (Tableau 3.8) et SEAL (Tableau 3.4).
Nous recommandons donc d'utiliser FV-NFLlib pour les petits circuits, et d'utiliser SEAL pour évaluer des circuits plus grands.

3.5 Le cryptosystème de Brakerski, Gentry et Vaikuntanathan (BGV)

C'est un cryptosystème développé en 2011 par Brakerski, Gentry et Vaikuntanathan [10], il hérite la plupart de ses propriétés du schéma de Brakerski et Vaikuntanathan [12]. Ce schéma utilise la technique du bootstrapping de Gentry comme optimisation et donc propose une version somewhat homomorphe qui peut être transformée en Fully homomorphe en utilisant le boostraping. Il est présent sous deux versions dont une est basée sur les entiers (LWE [39]) et l'autre sur les vecteurs (RLWE [34]). De plus, ce schéma propose d'utiliser deux optimisations qui sont le batching et le modulus switching. Commençons par définir la technique du batching, appelée aussi le mode SIMD (Single Instruction on Multiple Data).

Le Batching :
C'est une technique présentée pour la première fois dans l'article de Smart et Vercauteren [45]. Elle permet de chiffrer un vecteur de message et de l'empaqueter dans le même chiffré. Soit p un nombre premier avec q, tel que $p = 1 \mod 2d$ et $d = 2^n$.
Soit $R_p = \mathbf{Z}_p[x]/(x^d + 1)$ l'espace des messages, le polynôme $x^d + 1$ peut être écrit sous forme de produit de facteurs linéaires :

$$x^d + 1 = \prod_{i=1}^{d} (x - \alpha_i),$$

où $\alpha_i = \alpha^{2i+1}$ et α est une racine primitive 2d-ième de l'unité. Le nombre premier p peut être factoriser en un produit d'idéaux p_i dans R, où p_i est l'idéal engendré par $(p, x - \alpha_i)$. En appliquant le théorème des restes chinois, on obtient

$$R_p = \mathbf{Z}_p[x]/(x^d + 1) \equiv R_{p_1} \times R_{p_2} \ldots \times R_{p_d}.$$

Ainsi, la technique du batching consiste à empaqueter d messages $m_i \in R_{p_i}$ dans le même vecteur et de chiffrer ce vecteur de messages au lieu de chiffrer un seul message $m \in R_p$. Cette technique permet de bénéficier du traitement multi-coeur et du calcul SIMD. Ainsi, n'importe quelle fonction peut être évaluée homomorphiquement sur plusieurs entrées en conservant la même efficacité d'une seule entrée.

Le modulus switching :

La seconde optimisation présentée par le schéma BGV est la réduction de dimension, appelée aussi le changement de module (modulus switching). Cette technique permet une meilleure gestion du bruit contenu dans les chiffrés sans avoir besoin d'utiliser une clé d'évaluation. Les auteurs ont constaté que la profondeur du circuit à évaluer dépend de la taille de l'erreur, qui croît exponentiellement dans le schéma de Brakerski et Vaikuntanathan. Soit $q = x^k$ le modulo du schéma, si on calcule le produit de deux chiffrés avec un bruit de taille x, le bruit du produit est approximativement x^2, si on calcule un autre produit de chiffré le bruit du résultat est x^4, et ainsi de suite.
Cette nouvelle technique proposée dans ce schéma consiste à effectuer un changement de module après chaque multiplication de q à p, tel que $p < q$, afin de réduire l'erreur. Concrètement, une chaîne de module $q_i \equiv \frac{q}{x^i}$, doit être choisie pour maintenir le niveau de bruit à un niveau constant même après une multiplication.

La définition du schéma BGV :

Les éléments de la version basée sur le problème RLWE de ce schéma sont choisis dans l'anneau polynomial $R = Z[x]/f(x)$, où $f(x) = x^d + 1$ est un polynôme cyclotomique et $d = 2^n$. Concrètement R contient des polynômes de degré inférieur à n à coefficients dans $Z[x]$. On définit une distribution gaussienne χ sur R et une chaîne de modules $q_1 < q_2 < \ldots < q_L$ et leurs sous-anneaux R_{q_i} où L est la longueur du circuit (le nombre maximum de multiplication que nous pouvons effectuer sur les chiffrés), en cas de dépassement des multiplications L, nous devons appliquer la technique du boostrapping pour rafraîchir le chiffré. R_{q_i} désigne les polynômes de degré inférieur à n avec des coefficients modulo q_i. R_p est l'espace des textes en clairs où p est le module de texte en clair.

La génération de clés :

Soit s_k la clé secrète du schéma, la clé publique p_k est construite en masquant la clé secrète par un bruit choisit dans une distribution gaussienne sur R. On pose $s_k = (1, s)$ et $p_k = (a \cdot s_k + 2 \cdot e, a)$ la clé publique où $a \in R_{q_L}$.

Algorithme 44 Génération des clés pour BGV

Input $params = (R, d, q, t, \chi_{err}, \chi_{key})$.
Output s_k, p_k.

fonction GÉNÉRATION DE CLÉS($params$)
Choisir les éléments de la clé secrète s dans une distribution gaussienne $s \leftarrow \chi$,
Fixer $s_k = (1, s_1, \ldots, s_n)$,
Choisir uniformément un polynôme $a \leftarrow R_{qL}$,
Choisir aléatoirement une erreur $e \leftarrow \chi$,
Calculer $p_k = (a \cdot s_k + 2 \cdot e, a)$,
retourner s_k et $p_k = (p_{k0}, p_{k1})$.
fin fonction

Le chiffrement :
Soit $m \in \{0, 1\}$ un message donné, le chiffré de ce message est un couple de polynôme

$$E(m) = (m + p_k \cdot r, a \cdot r) \in R^2_{q_i},$$

où q_i est le modulo correspondant au niveau courant.

Algorithme 45 Chiffrement d'un message pour BGV

Input $m \in \{0, 1\}$ and $p_k = (p_{k0}, p_{k1})$.
Output $E(m) = (c[0], c[1])$.

fonction CHIFFREMENT(m, p_k)
Choisir aléatoirement une erreur $e \leftarrow \chi$
Calculer $A = p_{k0} \cdot r$
Calculer $c_0 = m + A$
Calculer $c_1 = p_{k1} \cdot r$
$E(m) = (c[0], c[1])$
retourner $E(m)$
fin fonction

Le déchiffrement :
Soit $c = (c_0, c_1)$ un chiffré, pour le déchiffrer on calcule l'équation suivante :

$$D(c) = ((c_0 - c_1 \cdot s_k) \mod q) \mod 2.$$

L'Algorithme 46 décrit la fonction de déchiffrement.

Algorithme 46 Déchiffrement d'un chiffré pour BGV

Input $c = (c[0], c[1])$ et s_k.
Output $m = \mathrm{D}(c)$.

fonction DÉCHIFFREMENT(c, s_k)
$r_0 = (c_1 \cdot s_k) \mod q$
$r_1 = (c_0 - r_0) \mod q$
$\mathrm{D}(c) = r_1 \mod 2$
retourner $m = \mathrm{D}(c)$
fin fonction

L'addition(c_1, c_2) :
La somme de deux chiffrés $c_1 = (c_1[0], c_1[1])$ et $c_2 = (c_2[0], c_2[1])$ est réalisée en calculant la somme des polynômes suivie d'une réduction modulaire

$$c_1 + c_2 = ([c_1[0] + c_2[0]]_q, [c_1[1] + c_2[1]]_q).$$

L'Algorithme 47 décrit la somme de chiffrés. L'équation suivante démontre que la somme de deux chiffrés est égale au chiffrement de la somme de plaintexts correspondants.

$$[c[1][0] + c[2][0]]_q = [(m_1 + m_2) + p_k(r_1 + r_2)]_q,$$

$$[c[1][1] + c[2][1]]_q = [a(r_1 + r_2)]_q.$$

Algorithme 47 Addition de chiffrés pour BGV

Input $c_1 = (c_1[0], c_1[1])$ et $c_2 = (c_2[0], c_2[1])$.
Output $\mathrm{S} = c_1 + c_2$.

fonction ADDITION$(c[1], c[2])$
$\mathrm{S0} = c_1[0] + c_2[0]$
$\mathrm{S1} = c_1[1] + c_2[1]$
retourner $\mathrm{S} = (\mathrm{S0}, \mathrm{S1})$.
fin fonction

Multiplication(c_1, c_2) :
La multiplication de deux chiffrés $c[1]$ et $c[2]$ consiste à calculer le produit tensoriel de $c[1] = (c[1][0], c[1][1])$ et $c[2] = (c[2][0], c[2][1])$. Le produit tensoriel augmente la taille du chiffré, donc on multiplie par une matrice différente après chaque produit.
Le produit de deux chiffrés est défini comme suit :

$$(c[1][0], c[1][1]) \cdot (c[2][0], c[2][1]) = (ct_0, ct_1, ct_2),$$

où

$$\begin{aligned} ct_0 &= [c[1][0]\cdot c[2][0]]_q, \\ ct_1 &= [(c[1][0]\cdot c[2][1]+c[1][1]\cdot c[2][0])]_q, \\ ct_2 &= [c[1][1]\cdot c[2][1]]_q. \end{aligned}$$

L'Algorithme 48 décrit la procédure de multiplication.

Algorithme 48 Multiplication de chiffrés pour BGV

Input $c[1]=(c[1][0],c[1][1])$ et $c[2]=(c[2][0],c[2][1])$.
Output $M=c[1]\cdot c[2]$.

fonction MULTIPLICATION($c[1]$, $c[2]$)
$ct_0=[c[1][0]\cdot c[2][0]]_q$

$r_0=c[1][0]\cdot c[2][1]$
$r_1=c[1][1]\cdot c[2][0]$
$ct_1=[r_0+r_1]_q$

$ct_2=[c[1][1]\cdot c[2][1]]_q$
retourner $M=(ct_0,ct_1,ct_2)$
fin fonction

Modulus switching :
Le modulus switching consiste à réduire le bruit d'un chiffré en transformant un chiffré modulo q à un nouveau chiffré modulo q^t tel que $q>q^t$.
Après chaque multiplication des chiffrés, la taille des chiffrés augmente. Pour réduire ce bruit on fait appel aux fonction SwitchKeyGen et SwitchKey qui sont hérités du schéma de Brakerski et Vaikuntanathan [11]. On commence par la fonction SwitchKeyGen 49, et ensuite la fonction SwitchKey 50. On définit le modulus switching pour réduire le bruit d'un ciphertext c calculé modulo q, et le transformer en un nouveau ciphertext c^t modulo q^t dans la propriété décrite ci-dessous.

Propriété 3.2. Pour transformer un chiffré $c=(c_1,c_2)$ calculé modulo q en un nouveau ciphertext $c^t=(c_1^t,c_2^t)$ modulo q^t, il suffit de retrouver les deux parties c_1^t et c_2^t du nouveau ciphertexts les plus proches de $(\frac{q}{q^t})\cdot c$, et vérifiant :

$$c_1^t = c_1 \mod 2 \text{ et } c_2^t = c_2 \mod 2.$$

Relinéarisation : réduction de dimension :
Après avoir calculé le produit de deux chiffrés la taille du chiffré obtenu augmente, c'est pour cette raison que les auteurs ont proposé la technique de relinéarisation pour pouvoir réduire la taille d'un chiffré issu d'un produit.
La procédure de relinéarisation proposée par Brakerski et Vaikanthanatun [11] consiste à transformer un chiffré $c=(c_1,c_2,c_3)$ issu d'une multiplication est

déchiffré avec la clé secrète $s \otimes s$ à un nouveau chiffré $c^t = (c_1^t, c_2^t)$ déchiffré avec la clé secrète s. On décrit dans les algorithmes ci dessous 49 et 50 les deux procédures nécessaires pour effectuer la réduction de dimension d'un chiffré. On commence par décrire les deux fonctions BitDecomposition 3.3 et Powersof2 3.4 qu'on va utiliser par la suite dans les algorithmes de relinéarisation.

Propriété 3.3 (BitDecomposition)**.** Cette fonction consiste à décomposer $x \in R_q^n$ en base binaire. Formellement $x = \sum_{j=0}^{l\log(q)J} 2^j \cdot u_j$, où les u_j sont des vecteurs dans $R_2^{n \cdot fl\log(q)l}$. Cette étape retourne $(u_0, u_1, \ldots, u_{l\log(q)J}) \in R_2$.

Propriété 3.4 (Powersof2)**.** C'est une fonction retourner le vecteur

$$(x, 2 \cdot x, \ldots, 2^{l\log(q)J} \cdot x) \in R_n^{l\log(q)J}.$$

La première étape de relinéarisation est appelée SwitchkeyGen 49, elle prend en entrée deux clés secrètes et retourne une matrice $\tau_{s_1 \leftarrow s_2}$ qui va permettre le changement de clé de s_1 à s_2.

Algorithme 49 SwitchkeyGen

Entrées : $s_1 \in R_q^{n_1}$ et $s_2 \in R_q^{n_2}$.
Sortie : $\tau_{s_1 \leftarrow s_2}$.

fonction SWITCHKEYGEN(s_1, s_2)
  Calculer $N = n_1 \cdot l\log(q)J$,
  Générer une clé publique A par rapport à la clé secrète s_2 et $N = n_1 \cdot l\log(q)J$,
  Calculer $A^t = Powerof2(s_1)$,
  Ajouter A^t à la première colonne de A, calculer $B = A + A^t$,
retourner $\tau = B$.
fin fonction

La deuxième étape de relinéarisation est appelée switchkey, elle prend en entrée la transformation τ, un chiffré c_1 d'un message m, déchiffré avec une clé s_1. Elle consiste à retourner un chiffré c_2 du même message m, déchiffré avec une clé s_2.

Algorithme 50 Switchkey

Entrées : $\tau_{s_1 \leftarrow s_2}, c_1, n_1, n_2, q$.
Sortie : c_2.

fonction SWITCHKEY$(\tau_{s_1 \leftarrow s_2}, c_1, n_1, n_2, q)$
  Calculer la matrice $M == Bit Decomposition(c_1)^T$,
  Calculer $c_2 = M \cdot \tau$,
retourner c_2.
fin fonction

3.5.1 Le bootstrapping

Pour rafraîchir un chiffré du schéma BGV, on commence par le chiffrer de nouveau et ensuite on doit évaluer le circuit de déchiffrement sur le nouveau chiffré.
Soit $c = (c[0], c[1])$ un chiffré d'un message m. Pour rafraîchir ce chiffré, on commence par chiffrer de nouveau ses éléments $c[0]$ et $c[1]$ et chiffrer de nouveau la clé secrète s_k, puis évaluer le circuit du déchiffrement sur le nouveau texte chiffré : $(E(c[0]), E(c[1]))$ en utilisant la nouvelle clé secrète $E(s_k)$. Le résultat de ce calcul retourne un nouveau chiffré qui chiffre le même message m.

$$\begin{aligned} D_{E(s_k)}(E(c[0]), E(c[1])) &= E(c[0]) - E(c[1]) \times E(s_k) \\ &= E(c[0] - c[1] \times s_k) \\ &= E(m). \end{aligned}$$

3.5.2 La librairie HELIB

Il s'agit d'une librairie développée en 2013 par Shai Halevi et Victor Shoup [30], elle implémente le schéma BGV en C++ et utilise la bibliothèque mathématique NTL.
En utilisant le même matériel que les librairies ci-dessus, nous obtenons les résultats suivants :
Pour $\lambda = 128, \sigma = 3.2, d = 2048, L = 1$ et $p = 65537$

Opération	Temps d'exécution en μs
Génération de clés	218647
Chiffrement	11435969
Déchiffrement	62265
Addition	33
Multiplication	6112

TABLEAU 3.7 – Performances de la librairie HELIB

Le tableau ci-dessous présente la taille mémoire nécessaire pour le stockage les clés et les chiffrés de la librairie HELIB.

Structure	Taille en Koctets
Clé secrète	17614
Clé publique	17481
Chiffré	148

TABLEAU 3.8 – Taille des clés et des chiffrés dans la librairie SEAL-RNS

On remarque bien que la taille des chiffrés dans HELIB est très grande par rapport à celle de la librairie FV-NFLlib et SEAL. Ceci est dû au stockage des chiffrés par rapport à toutes la chaîne des modulos q_i de tous les niveaux.

3.5.3 Comparaison des cryptosystèmes FV et BGV

Malgré la contribution majeure du schéma BGV sur le plan théorique et en particulier la technique du modulus switching, ce schéma n'est pas très efficace quand il s'agit d'évaluer des circuits de grande taille, vu que la profondeur du circuit et en particulier la chaîne des modulos impacte la taille des textes chiffrés et le temps de génération de clé. En revanche, le schéma BGV est performant au niveau des temps d'exécution des opérations, grâce à la technique du batching, et peut aussi évaluer des circuits de profondeur illimité. C'est pour cette raison que ses auteurs ont proposé le bootstrapping aussi pour pouvoir fixer une petite profondeur et appeler le bootstrapping si c'est nécessaire. Le cryptosystème FV est plus efficace au niveau des tailles des clés et des chiffrés.
Par conséquent, le cryptosystème BGV peut être utilisé lorsqu'on manipule des circuits de profondeur illimité et quand on peut chiffrer un vecteur messages en même temps. Par contre le cryptosystème FV est plus adapté au circuit de petite profondeur et pour les cas d'usage où on ne peut pas attendre l'arrivée de plusieurs messages pour lancer le chiffrement.

3.6 La génération des paramètres pour FV

Dans cette section, nous allons expliquer comment générer les paramètres (λ, q, d, L, t) du cryptosystème FV en utilisant un estimateur dédié aux paramètres des schéma basé sur le problèmes LWE et RLWE.

3.6.1 L'estimateur des paramètres de LWE/RLWE

Il s'agit d'un module de Sage [4] développé en 2015 par Martin Albrecht pour estimer la sécurité concrète des paramètres données. C'est un moyen simple de choisir des paramètres résistants aux attaques connues. L'utilisateur doit donner les paramètres d(degré), α(taux d'erreur), q(module) comme entrées à l'estimateur qui retournera le nombre d'opérations binaires pour attaquer ses paramètres, l'espace mémoire nécessaire pour les attaques et le nombre d'appels de l'oracle LWE.
Soit d le degré de polynôme, q le modulo des coefficients, $\alpha \ll 1$ le taux d'erreur et $\sigma = \frac{\alpha \cdot q}{\sqrt{\nu}_{2\cdot\pi}}$ l'écart type. Pour des raisons d'optimisations on choisit $d = 2^k$, $\sigma = 3.19$ et $q = 2^h$. L'Algorithme 51 génère des paramètres en utilisant l'estimateur[4].

Algorithme 51 La génération de paramètres

Input $d = 2^k$, σ et $q = 2^h$.
Output le niveau de sécurité correspondant.

fonction ESTIMATION DE SÉCURITÉ($d = 2^k, \sigma = 3.19, q = 2^h$)
Utiliser l'estimateur pour retourner le coût de la meilleure attaque sur les paramètres choisis
tantque La meilleur attaque coûte moins que λ **faire**
décrémenter h
fin tantque
retourner q, d et σ
fin fonction

3.6.2 Exemples de paramètres

On présente ci-dessous quelques exemples de paramètres d et q que nous avons obtenus en utilisant l'estimateur de Martin Albrecht [4], ainsi que la profondeur du circuit L obtenue en utilisant la bibliothèque SEAL [42].
Pour un paramètre de sécurité $\lambda = 128$, un module de texte en clair de 1,16 et 30 bits (la taille maximale du module de texte en clair acceptée par cette bibliothèque est 31 bits), et un degré de polynôme 1024, 2048, 4096 et 8192 nous obtenons les résultats ci-dessous :

d : Degré du polynôme	$\|q\|$: La taille du modulo des coefficients	L : La profondeur du circuit
1024	29	1
2048	56	2
4096	110	6
8192	219	11

TABLEAU 3.9 – Taille des paramètres pour un niveau de sécurité 128 et des textes en clair de 1 bit

d : Degré du polynôme	$\|q\|$: La taille du modulo des coefficients	L : La profondeur du circuit
1024	29	0
2048	56	1
4096	110	2
8192	219	6
16383	441	11

TABLEAU 3.10 – Taille des paramètres pour un niveau de sécurité 128 et des textes en clair de 16 bit

d : Degré du polynôme	$\|q\|$: La taille du modulo des coefficients	L : La profondeur du circuit
1024	29	0
2048	56	0
4096	110	1
8192	219	3
16383	441	4

TABLEAU 3.11 – Taille des paramètres pour un niveau de sécurité 128 et des textes en clair de 30 bit

d : Degré du polynôme	$\|q\|$: La taille du modulo des coefficients	L : La profondeur du circuit
1024	20	0
2048	39	1
4096	77	4
8192	153	7

TABLEAU 3.12 – Taille des paramètres pour un niveau de sécurité 192 et des textes en clair de 1 bit

d : Degré du polynôme	$\|q\|$: La taille du modulo des coefficients	L : La profondeur du circuit
2048	39	0
4096	77	1
8192	153	4

TABLEAU 3.13 – Taille des paramètres pour un niveau de sécurité 192 et des textes en clair de 16 bit

d : Degré du polynôme	$\|q\|$: La taille du modulo des coefficients	L : La profondeur du circuit
2048	39	0
4096	77	0
8192	153	1

TABLEAU 3.14 – Taille des paramètres pour un niveau de sécurité 192 et des textes en clair de 30 bit

En analysant les tableaux 3.9, 3.10, 3.11, 3.12, 3.13, et 3.14 on remarque bien que pour un niveau de sécurité, un degré d et un modulo q donnés, la profondeur du circuit diminue si on augmente la taille du texte en clair. On remarque aussi que pour un degré d, un modulo q et une taille de texte en clair donnés, la profondeur du circuit diminue si on augmente le niveau de sécurité. De plus, la profondeur du circuit augmente pour un niveau de sécurité donné et une taille de texte en clair donnée si on augmente les paramètres q et d.

3.7 Conclusion

Dans ce chapitre, on a fait la comparaison des deux cryptosystèmes homomorphes FV et BGV, basés sur le problème RLWE. Cette comparaison nous a permis de choisir le cryptosystème qu'on utilisera par la suite. Il s'agit du schéma FV qui se rapproche le plus d'une utilisation concrète, dans la mesure où il permet de chiffrer un seul message contrairement au BGV qui chiffre un lot de messages. De plus, la taille des textes chiffrés avec FV est plus petite que celle des chiffrés de BGV. Nous avons étudié ensuite les paramètres des schémas basés sur le problème RLWE et en particulier le schéma FV afin de choisir les paramètres permettant une implémentation efficace du schéma de FV, qu'on va présenter dans le chapitre suivant, en tenant compte des contraintes d'un environnement IoT. Ces contraintes nous ont amenés à choisir un degré $d = 2048$, un modulo de taille $|q| = 56$, et une profondeur de circuit $L = 1$ pour un niveau de sécurité 128 bits.

Chapitre 4

Le démonstrateur

Sommaire

Ce chapitre est une preuve de concept que la cryptographie homomorphe peut être déployée dans la pratique et peut être utilisée pour assurer la sécurité et le traitement de données des clients avec du matériel existant. Nous présentons un cas concret d'utilisation de la cryptographie homomorphe qui n'a pas été cité auparavant dans la littérature, en utilisant le cryptosystème de Fan et Vercauteren (FV) qui est le plus adapté au cicruit de petite taille comme cité dans le chapitre précédent, et nous proposons une implémentation pratique de ce schéma et son déploiement dans le cadre de l'IoT. Ce cas d'usage est décrit dans la section suivante.

4.1 Cas d'usage

Le chiffrement homomorphe est une solution pour résoudre les principaux problèmes de l'IoT : la securité, le stockage et les calculs délégués sur le cloud. Dans l'IoT, des dispositifs matériels détectent des phénomènes physiques tels que la lumière, la chaleur, le mouvement, l'humidité ou la pression et envoient généralement des signaux qui sont convertis en affichage lisible par l'homme sur le dispositif lui-même ou transmit électroniquement sur un réseau pour traitement ultérieur. C'est dispositifs sont appelés des capteurs.

Considérons un cas d'usage dans l'IoT où différents capteurs envoient des données au cloud à travers plusieurs passerelles. Le but de l'utilisation du cloud est de stocker les données et faire des traitements dessus. À chaque fois que la passerelle reçoit des messages des capteurs, elle les chiffre de façon homomorphe et les envoie au cloud. Pour notre cas, le cloud est en mesure de stocker les données et effectue quelques calculs basés sur l'addition et la multiplication des données collectées à diff ts moments et dans divers contextes géographiques.
Le protocole utilisé pour l'envoi de messages des capteurs aux passerelles est LORA [2] (Long Range Wireless Protocol). Le cloud comprend un serveur MQTT (publish/subscribe) [1], ainsi pour stocker une donnée dans le cloud, la passerelle envoie une commande "publish" pour publier la donnée dans le cloud et pour récupérer une donnée du cloud, la machine de l'administrateur envoie une commande "subscribe" pour se souscrire à un sujet. Ce scénario est illustré par la Figure 4.1.

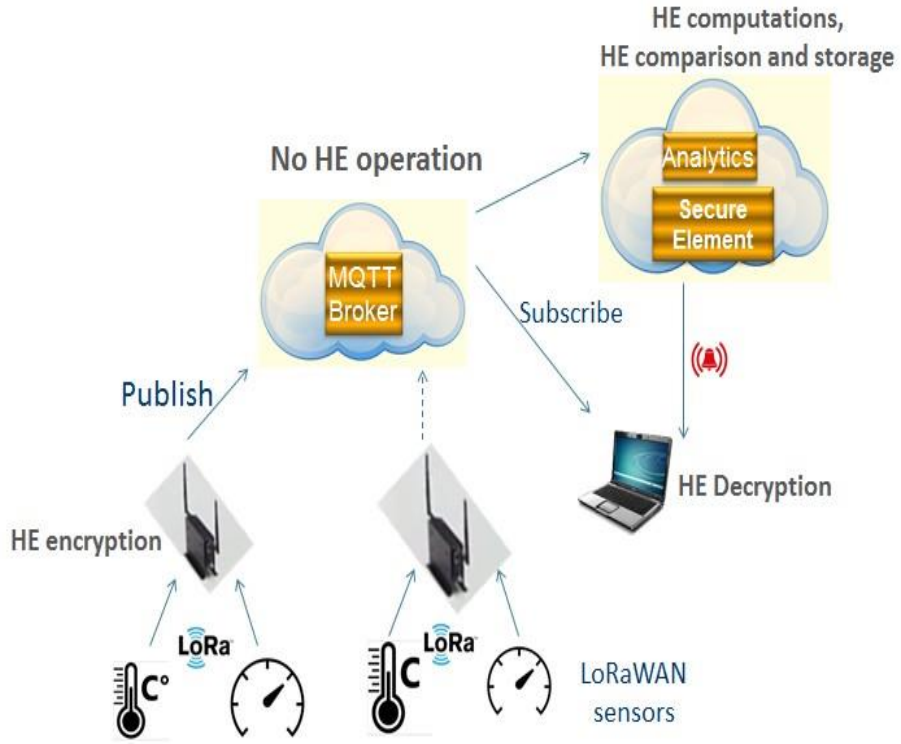

FIGURE 4.1 – Cas d'usage IoT

4.1.1 Cas d'usage des images

Le chiffrement homomorphe peut être utilisé pour assurer la sécurité et le traitement d'image dans le cloud tels que la transformation de couleur, la modification de la saturation et le changement de luminosité. Dans notre modéle, chaque image est représentée par une matrice de pixels, et chiffrer une image revient à chiffrer l'ensemble des pixels. De plus, les opérations du traitement d'image sont basées sur des produits et des sommes de matrices. Prenons l'exemple d'une image de 1024 × 1024 pixels, l'utilisateur envoi l'image à la passerelle qui chiffre chaque pixel par le cryptosystème homomorphe FV.

Chaque chiffré de pixel est de taille 32 koctets, donc la taille globale de l'image chiffrée est 32 Goctets. Si nous utilisons un Ethernet de $0.1 Moctes/s$ nous aurons besoin de 100 heures pour envoyer l'image chiffrée de la passerelle vers le Cloud. Ce cas d'usage n'est pas très efficace et c'est pour cette raison qu'on va se contenter de décrire des cas d'usage où la taille des données à chiffrer est plus petite et plus adaptées aux contraintes de la cryptographie homomorphe.

4.1.2 Cas d'usage plus réalistes

Imaginons un cas d'usage dans un port maritime dans lequel il est très utile d'accélérer les expéditions de produits, ce besoin peut être résolu en prévoyant suffisamment de camions pour transporter la marchandise. Les compagnies maritimes doivent vérifier en temps réel la marchandise transportée sans révéler des informations sur les clients. D'autant plus que les armateurs sont engagés pour protéger le secret professionnel. Les entreprises vérifient la valeur de marchandises, leur coût global, leur contenu et leur poids afin d'anticiper leur arrivée au port maritime, en prévoyant des camions ou des trains pour le transport des marchandises et le passage des douanes.
Dans ce cas d'usage, chaque compagnie possède plusieurs navires et plusieurs conteneurs. De plus, chaque navire possède une passerelle et chaque conteneur possède un capteur. Les conteneurs envoient des données à la passerelle du navire qui les chiffre avant de les envoyer au cloud. En pratique, lorsque le cloud reçoit les données il peut effectuer des calculs sur les données chiffrées sans avoir à les déchiffrer comme c'est décrit dans la Figure 4.2, il peut calculer la somme des conteneurs détenant les produits de certaines entreprises dans le port maritime en augmentant le nombre de conteneurs. Nous pouvons aussi calculer la valeur des marchandises dans ces conteneurs et la somme des poids de conteneurs avant de les charger sur les navires en faisant plusieurs additions. De plus, nous pouvons effectuer une conversion des devises des prix des produits en calculant le produit du prix de chaque produit et la devise du pays. Ce calcul nécessite un niveau de multiplication.

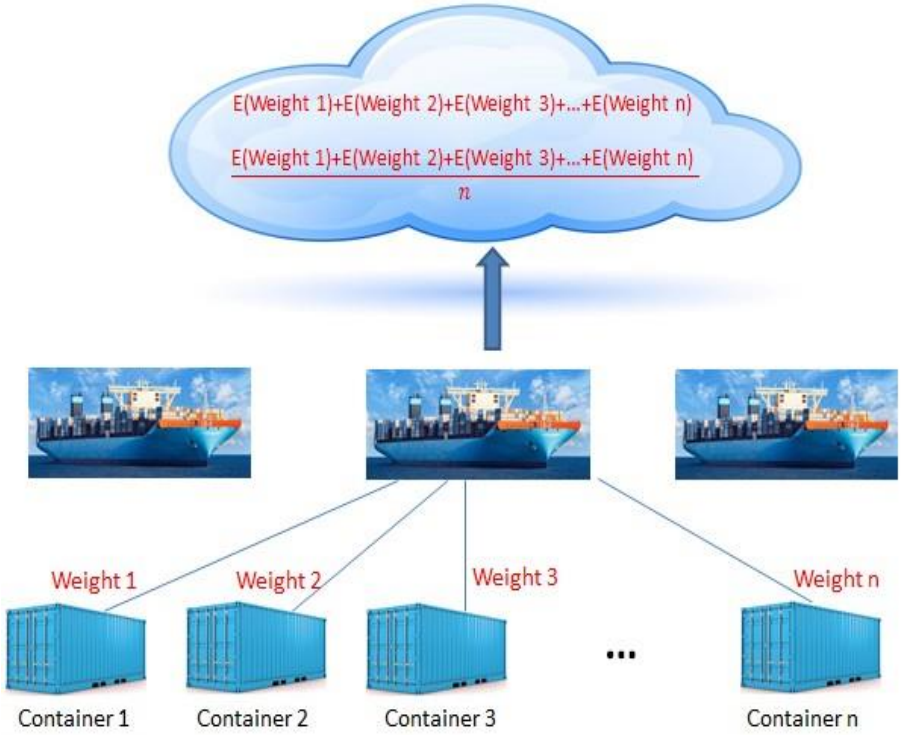

FIGURE 4.2 – Application de la cryptographie homomorphe dans un port maritime

On peut aussi décrire un scénario dans lequel différents supermarchés de différentes entreprises ont besoin de stocker et de calculer le nombre des ventes de produits dans le cadre de la gestion des stocks. L'objectif de ces entreprises est de stocker dans le cloud le chiffrement de ces données sans révéler leur identité en raison de la concurrence. Dans le cloud, nous pouvons calculer la somme de vente de chaque produit afin d'approvisionner le stock des supermarchés si nécessaire.

Prenons l'exemple d'un autre cas d'usage dans le transport et en particulier dans les trains. Ce cas d'usage va nous permettre de vérifier le niveau d'eau dans les toilettes. Supposons que plusieurs trains de plusieurs entreprises possèdent une passerelle par train et plusieurs capteurs pour chaque wagon, chaque passerelle reçoit des messages des capteurs, les chiffre et ensuite les envoie vers le Cloud, celui-ci regroupe les données et effectue des calculs dessus. Le cloud détient un secure element qui permet de prendre des décisions et en particulier faire des comparaisons sur les données chiffrées sur les données regroupées, en déchiffrant les données et en comparant les textes en clairs. La méthode de comparaison dans un secure element est décrite en détail dans la Section 4.3. Les appareils envoient systématiquement le niveau d'eau au cloud via la passerelle et à l'intérieur du cloud, ensuite le secure element compare le niveau d'eau au niveau minimum acceptable et par conséquent envoie des alertes à l'administrateur en cas de manque d'eau dans les toilettes. Le chiffrement homomorphe garantit la confidentialité, le respect de la vie privée et l'anonymisation de l'origine des conteneurs, tout en permettant le traitement et le stockage des données dans un cloud publique.

Nous avons choisi d'utiliser le système de cryptographie homomorphe FV dans les cas d'usage décrit ci-dessus, parce qu'il le plus adapté aux circuits de petite profondeur. De nombreuses bibliothèques [43], [31] implémentent le cryptosystème FV, mais elles ne sont pas portables dans la passerelle. La taille de la bibliothèque SEAL est 5 Moctets et la taille de la bibliothèque FV-NFLIB plus la bibliothèque NFLIB nécessaire pour la faire tourner est 5 Moctets aussi. Nous avons choisi de mettre en œuvre notre propre API pour l'intégrer dans la passerelle, le cloud et dans la machine administrateur.

4.2 Implémentation

Nous avons implémenté le cryptosystème FV en langage C. Il a été intégré dans les modules IoT suivants : la passerelle, le cloud et la machine de l'administrateur. Il s'agit d'un code portable, autonome et indépendant des autres bibliothèques. Cette implémentation peut être considérée comme une preuve de concept où nous n'utilisons que la bibliothèque mathématique standard "math.h" sans parallélisme SIMD et sans traitement multi-coeur. Nous utilisons un ensemble de paramètres permettant d'évaluer des circuits d'un seule niveau de multiplication, avec un niveau de sécurité de 128 bits, un degré 2048 pour le polynôme cyclotomique et 56 bits pour les coefficients polynômes, on a choisi l'anneau R_2 comme distribution de clé secrète χ_{key}, $\sigma = 3.1$, et χ_{err} une distribution gaussienne bornée par $B = 31$ pour choisir les coefficients du bruit. Nous avons choisi ces paramètres parce qu'ils sont les plus petits permettant de construire un schéma capable d'évaluer des circuits d'une multiplication avec un niveau de sécurité correct. Ce schéma permet d'évaluer un niveau de multiplication et chiffre des plaintexts de 16 bits. Le code est intégré et exécuté sur une machine d'administrateur qui détient un processeur Intel Core i5 à 2,4 GHz, une passerelle avec un processeur Intel Atom à 1,91 GHz, et une machine virtuelle utilisant un processeur Intel Xeon à 2,40 GHz au niveau du cloud.

4.2.1 Description de l'implémentation

Nous avons implémenté notre propre code à partir de zéro afin d'obtenir une implémentation homogène, et portable dans la passerelle. Dans cette implémentation, les coefficients des polynômes sont calculés modulo q et stockés dans une structure de type long long capable de contenir 64 bits. Si q est une puissance de 2, par exemple $q = 2^{56}$, le calcul de modulo q correspond à une opération AND. Soit N et k deux entiers où $q = 2^k$, l'opération de module vérifie l'équation suivante : $N \bmod 2^k = N \,\&\, 2^k - 1$.
Dans notre code, nous avons développé notre propre arithmétique 128 bits afin de permettre le stockage du résultat d'une multiplication de deux long long. Soit A et B deux entiers de 64 bits, de type long long, on note $A = A_1|A_2$ et $B = B_1|B_2$ où A_1, A_2, B_1, B_1, B_2 sont des entiers de 32 bits chacun. Pour calculer

le produit de A et B nous utilisons l'Algorithme 52.

Algorithme 52 *L'arithmétique 128 bits*

Entrées $A = A_1|A_2$ and $B = B_1|B_2$.
Sortie Le produit $A \times B$.

fonction MULTIPLIER DEUX LONG LONG(A, B)
Calculer $A_2 \cdot B_2 = r_1|r_2$.
Calculer $A_1 \cdot B_2 = r_3|r_4$.
Calculer $A_2 \cdot B_1 = r_5|r_6$.
Calculer $A_1 \cdot B_1 = r_7|r_8$.
Calculer $S_1 = r_1 * 2^{32} + r_4 + r_6$.
7: Calculer $M_1 = (r_1 * 2^{32} + r_4 + r_6) \bmod 2^{32}$.
Calculer $S_2 = r_4 * 2^{32} + r_5 * 2^{32} + r_8 + M_1$.
Calculer $M_2 = (r_4 * 2^{32} + r_5 * 2^{32} + r_8 + M_1) \bmod 2^{32}$.
Calculer $S_3 = r_7 * 2^{32} + M_2$.
retourner $A \cdot B = S_3|S_2|S_1|r_2$.
fin fonction

Dans cette implémentation on a repris l'algorithme de Fan et Vercauteren en omettant l'étape de la relinéarisation, parce que nous avons besoin d'assurer une seule multiplication. Autrement dit, le résultat de la multiplication des chiffrés est un texte chiffré de trois éléments, et nous pouvons facilement faire des additions entre des textes chiffrés de trois éléments et des additions entre un texte chiffré de trois éléments et un texte chiffré de deux éléments. La relinéarisation augmente le niveau du bruit dans le chiffré à cause de la multiplication par la clé de relinéarisation, et donc nous ne pouvons même pas effectuer une seule multiplication avec les paramètres choisis ($\lambda = 128$, $d = 2048$, $|q| = 56$). Par conséquent, il faut augmenter les paramètres pour pouvoir déchiffrer correctement. En revanche, en augmentant les paramètres les coefficients des polynômes ne peuvent plus être stockés dans une structure long long. Donc on a choisi de garder les chiffrés tels qu'ils sont, avec trois éléments, après la multiplication et implémenter le cryptosystème FV avec les paramètres : $\lambda = 128$, $d = 2048$, et $|q| = 56$. On présente ci-dessous les fonctions de l'algorithme de FV que nous avons implémenté.

4.2.2 Les algorithmes implémentés

Soient λ le paramètre de sécurité, $q > 1$ le module des coefficients, et $t > 1$ le module du texte en clair, tel que $t < q$. On note par R_q et R_t l'espace des chiffrés et des plaintexts. $R_q[x] = Z_q[x]/f(x)$ où $Z_q[x]$ est l'anneau de polynôme à coefficients modulo q, $f(x) = x^d + 1$ et $d = 2^n$. Les éléments de $R_q[x]$ sont des polynômes de degré inférieur à d et à coefficients modulo q.
Les éléments de l'anneau $R_q[x]$ sont notés en minuscule ($a \in R_q[x]$), on note par

$[a]_q$ les éléments de R obtenus en calculant tous les coefficients modulo q.
Pour tout $x \in R_q$ on note par lxl un nombre arrondi à l'entier le plus proche, fxl et lxJ des nombres arrondis à l'entier supérieur et inférieur. La notation $x \leftarrow D$ est utilisée pour tirer aléatoirement x d'une distribution D, et $x \xleftarrow{\$} D$ pour tirer uniformément x de D.
Soit χ une distribution gaussienne sur R_s avec une déviation standard σ. Les deux distributions χ_{err} et χ_{key} sont utilisées pour tirer les erreurs et la clé secrète du schéma. Ces distributions χ_{err} et χ_{key} sont bornées avec une borne B où $B = 10 \cdot \sigma$. On a choisi l'anneau R_2 comme distribution de clé secrète χ_{key}, $\sigma = 3.1$, et χ_{err} une distribution gaussienne bornée par $B = 31$. Soit $params$ l'ensemble de paramètres du schéma, $params = (R, d, q, t, \chi_{err}, \chi_{key})$.
L'Algorithme 53 décrit les étapes de génération de la clé privée s_k et la clé publique $p_k = (p_k[0], p_k[1])$.

Algorithme 53 Génération des clés

Entrées $params = (R, d, q, t, \chi_{err}, \chi_{key})$.
Sortie $s_k et p_k$.

fonction GÉNÉRATION DE CLÉS($params$)
$s_k \leftarrow \chi_{key}$
$a \xleftarrow{\$} R_q$
$e \leftarrow \chi_{err}$
$p_k = ([-(a \cdot s_k + e)]_q, a)$
retourner s_k et p_k.

fin fonction

L'Algorithme 54 décrit la procédure de chiffrement d'un message m en utilisant la clé publique $p_k = (p_k[0], p_k[1])$.

Algorithme 54 Chiffrement d'un message

Entrées $m \in R_t$ et p_k.
Sortie $E(m) = (c[0], c[1])$.

fonction CHIFFREMENT(m, p_k)
$\delta = l\frac{q}{t}J$
$u \leftarrow \chi_{key}$
$e_1 \leftarrow \chi_{err}$
$e_2 \leftarrow \chi_{err}$
$c[0] = [p_0 \cdot u]_q$
$r_0 = [c[0] + e_1]_q$
$c[0] = [r_0 + \delta \cdot m]_q$
$r_0 = [p_1 \cdot u]_q$
$c[1] = [r_0 + e_2]_q$
$E(m) = (c[0], c[1])$
retourner $E(m)$
fin fonction

L'Algorithme 55 décrit la procédure de déchiffrement d'un chiffré $(c[0], c[1])$ en utilisant la clé secrète s_k.

Algorithme 55 déchiffrement d'un chiffré $(c[0], c[1])$

Entrées $C = (c[0], c[1])$ et s_k.
Sortie $D(C)$.

fonction DÉCHIFFREMENT(C, s_k)
$r_0 = c[1] \cdot s_k$
$D(C) = [c[0] + r_0]_q$
$r_0 = t \cdot D(C)$
$D(C) = [l\frac{r_0}{q}l]_t$
retourner $D(C)$
fin fonction

L'Algorithme 56 décrit la procédure de déchiffrement d'un chiffré $(c[0], c[1], c[2])$, qui a subi une multiplication, en utilisant la clé secrète s_k.

Algorithme 56 déchiffrement d'un chiffré $(c[0], c[1], c[2])$

Entrées $C = (c[0], c[1], c[2])$ et s_k.
Sortie D(C).

fonction DÉCHIFFREMENT(C, s_k)
$r_0 = c[1] \cdot s_k$
$s_2 = s_k \cdot s_k$
$r_1 = c[2] \cdot s_2$
$D(C) = [c[0] + r_0 + r_1]_q$
$r_0 = t \cdot D(C)$
$D(C) = [l\frac{r_0}{q}l]_t$
retourner D(C)
fin fonction

L'Algorithme 57 décrit le calcul de la somme de deux chiffrés $c(c[1][0], c[1][1])$ et $(c[2][0], c[2][1])$.

Algorithme 57 Addition de chiffrés $c(c[1][0], c[1][1])$ et $(c[2][0], c[2][1])$

Entrées $c[1] = (c[1][0], c[1][1])$ et $c[2] = (c[2][0], c[2][1])$.
Sortie $S = c[1] + c[2]$.

fonction ADDITION$(c[1], c[2])$
$S[0] = c[1][0] + c[2][0]$
$S[1] = c[1][1] + c[2][1]$
retourner $S = (S[0], S[1])$.
fin fonction

L'Algorithme 58 décrit le calcul de la somme d'un chiffré résultant d'une multiplication $c(c[1][0], c[1][1], c[1][2])$ et un chiffré qui n'a jamais subi de multiplication $(c[2][0], c[2][1])$.

Algorithme 58 Addition de chiffrés $c(c[1][0], c[1][1], c[1][2])$ et $(c[2][0], c[2][1])$

Entrées $c[1] = (c[1][0], c[1][1], c[1][2])$ et $c[2] = (c[2][0], c[2][1])$.
Sortie $S = c[1] + c[2]$.

fonction ADDITION$(c[1], c[2])$
$S[0] = c[1][0] + c[2][0]$
$S[1] = c[1][1] + c[2][1]$
$S[2] = c[1][2]$
retourner $S = (S[0], S[1], S[2])$.
fin fonction

L'Algorithme 59 décrit le calcul de la somme de deux chiffrés issus d'une multiplication $c(c[1][0], c[1][1], c[1][2])$ et $(c[2][0], c[2][1], c[2][2])$.

Algorithme 59 Addition de chiffrés $c(c[1][0], c[1][1], c[1][2])$ et $(c[2][0], c[2][1], c[2][2])$

Entrées $c[1] = (c[1][0], c[1][1])$ et $c[2] = (c[2][0], c[2][1])$.
Sortie $S = c[1] + c[2]$.

fonction ADDITION($c[1]$, $c[2]$)
$S[0] = c[1][0] + c[2][0]$
$S[1] = c[1][1] + c[2][1]$
$S[2] = c[1][2] + c[2][2]$
retourner $S = (S[0], S[1], S[2])$.
fin fonction

Cet Algorithme 58 décrit le calcul du produit de deux chiffrés $c(c[1][0], c[1][1])$ et $(c[2][0], c[2][1])$. Rappelons que notre implémentation permet d'évaluer uniquement des circuits d'un seul niveau de multiplication, d'où le besoin d'une fonction permettant d'effectuer le produit sur des chiffrés qui n'ont jamais subi de multiplication.

Algorithme 60 Multiplication de chiffrés $c(c[1][0], c[1][1])$ et $c[2] = (c[2][0], c[2][1])$

Entrées $c[1] = (c[1][0], c[1][1])$ et $c[2] = (c[2][0], c[2][1])$.
Sortie $M = c[1] \cdot c[2]$.

fonction MULTIPLICATION($c[1]$, $c[2]$)
$ct_0 = t \cdot (c[1][0] \cdot c[2][0])$
$r_0 = \frac{ct_0}{q}$
$ct_0 = [lr_0l]_q$

$ct_1 = c[1][0] \cdot c[2][1]$
$r_0 = ct_1 + c[1][1] \cdot c[2][0]$
$ct_1 = t \cdot r_0$
$r_0 = \frac{ct_1}{q}$
$ct_1 = [lr_0l]_q$

$ct_2 = t \cdot (c[1][1] \cdot c[2][1])$
$r_0 = \frac{ct_2}{q}$
$ct_2 = [lr_0l]_q$
retourner $M = (ct_0, ct_1, ct_2)$
fin fonction

Notre implémentation peut évaluer tous les circuits qui peuvent être de la forme décrite dans les figures Figure 4.3 et Figure 4.4 .

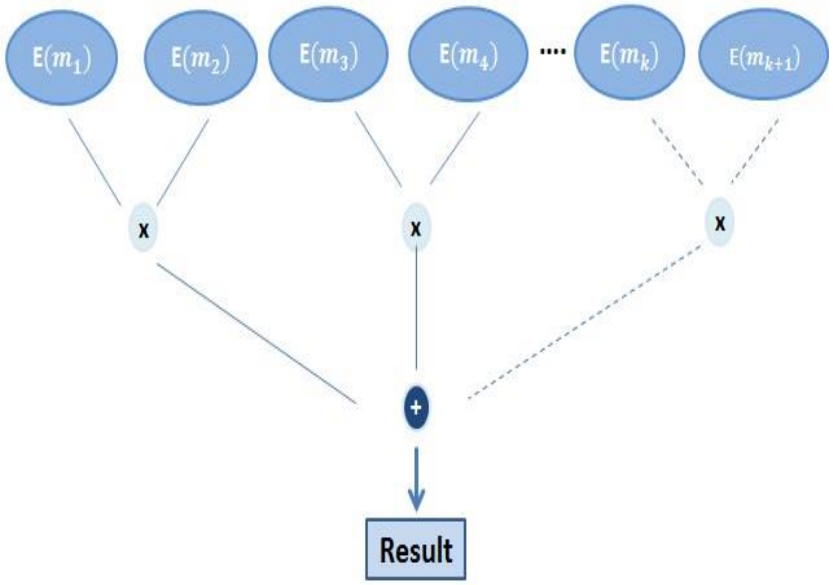

FIGURE 4.3 – Circuit permettant de calculer une moyenne pondérée

Un cas d'utilisation pratique du circuit de la Figure 4.3 est le calcul de la moyenne pondérée, ou le produit de matrices utilisé dans le traitement d'images pour transformer les couleurs, modifier la saturation et changer la luminosité. Ce circuit peut être utilisé dans le contexte du port maritime pour convertir la valeur de la marchandise de l'euro en yens par exemple.

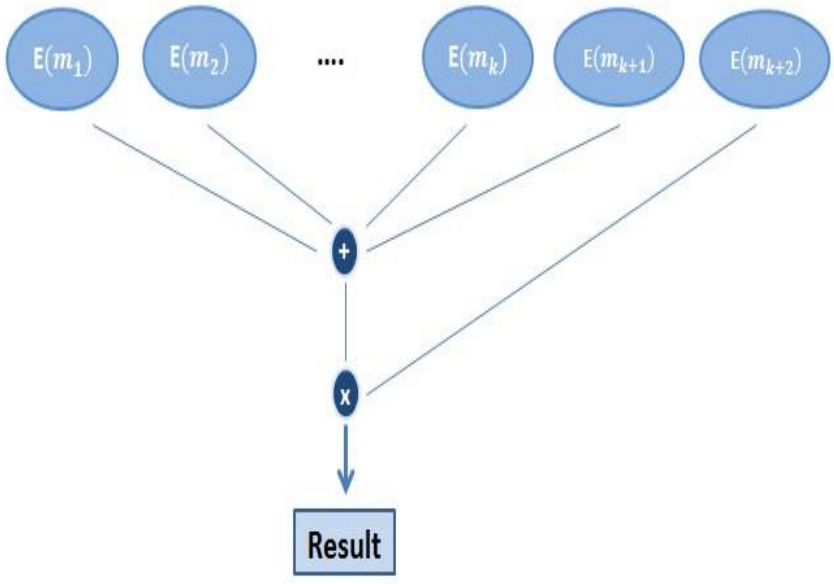

FIGURE 4.4 – Circuit permettant de calculer une moyenne

Le circuit de la figure Figure 4.4 peut être utilisé pour calculer une moyenne, ce calcul est pratique dans le cas d'usage du port maritime où les armateurs ont besoin de calculer le poids moyen des conteneurs présent dans le port d'une certaine compagnie en calculant la somme des poids et ensuite diviser par le nombre de conteneurs, cette division peut être écrite sous forme de multiplication $\frac{1}{sum}$.

4.2.3 Performances

Dans ce paragraphe, nous présentons les performances de notre implémentation sous différentes plateformes. Le même programme a été embarqué dans la passerelle, le cloud et la machine administrateur, nous avons ajouté quelques options pour spécifier quelle fonction du code nous voulons exécuter. Les fonctions de génération de clés et de chiffrement peuvent être exécutées dans la passerelle (processeur Intel Atom à 1.91GHz). Dans le cloud public (processeur Intel Xeon à 2.40GHz), nous ne devons exécuter que les fonctions d'addition et de multiplication de chiffrés. En outre la fonction de déchiffrement ne peut être exécutée que sur la machine de l'administrateur. Nous commençons par donner la taille des textes chiffrés et des clés de notre implémentation qui manipule des textes clair de 16 bits. La taille du texte chiffré est 32 koctets, la taille de la clé secrète est de 2048 bits et la taille de la clé publique est 32 koctets. Les temps d'exécution de toutes les fonctions à l'intérieur des différents modules de notre cas d'usage sont décrits dans le tableau Table 4.1. Nos résultats prouvent que nous pouvons utiliser le chiffrement homomorphe dans un cadre réel. Le timing pourrait probablement être amélioré, mais notre premier objectif était de déployer une implémentation homogène du cryptosystème FV entre différentes plateformes et un cloud.

Opération	Temps d'exécution dans la machine de l'administrateur	Temps d'exécution dans le Cloud	Temps d'exécution dans la passerelle
Génération de la clé secrète(*ms*)	0.122	—	0.151
Génération de la clé publique(*ms*)	70.994	—	175.946
Chiffrement (*ms*)	94.598	—	233.007
Déchiffrement d'un texte chiffré normal(*ms*)	18.934	—	—
Déchiffrement d'un texte chiffré avec 3 éléments (*ms*)	62.177	—	—
Addition (*ms*)	0.085	0.022	—
Multiplication (*ms*)	177.553	203.626	—

TABLEAU 4.1 – Temps d'exécution des fonctions

4.3 Comparaison des textes chiffrés homomorphes

Le chiffrement homomorphe est très prometteur pour l'industrie. Cependant, il est impossible de prendre une décision dans le cloud, basée sur la comparaison de deux chiffrés homomorphes. Le résultat d'une comparaison de deux chiffrés est un booléen chiffré, par conséquent, on ne peut pas prendre des décisions ou envoyer des alertes depuis le cloud en se basant sur ce résultat. L'idée présentée dans cette section consiste à utiliser un secure element dans le cloud pour déchiffrer en toute sécurité les chiffrés pour pouvoir comparer les plaintexts correspondants afin de pouvoir prendre des décisions, comme décrit précédemment dans l'exemple du niveau d'eau dans les toilettes des trains.

4.3.1 Description de la méthode

Nous avons proposé un système où nous utilisons un secure element à l'intérieur du Cloud afin de déchiffrer les chiffrés et comparer les plaintexts correspondants pour pouvoir prendre des décisions et puis l'envoi d'alertes si nécessaire.
La fonction de déchiffrement $D(C) = [l^{\frac{t\cdot[c[0]+c[1]\cdot s_k]_q}{q}} l]_t$ sera embarquée dans le secure element. Mais la contrainte qui se pose est que le calcul de cette fonction est très coûteux au niveau de la mémoire nécessaire pour le stockage des variables intermédiaires et en particulier le calcul de $(c[1]\cdot s_k) \bmod q$.
Nous avons proposé alors d'éviter de calculer le produit $(c[1]\cdot s_k) \bmod q$, qui nécessite des ressources mémoires et un temps d'exécution importants. Ainsi, le calcul de ce produit sera délégué au CPU, sans révéler la clé secrète. On commence par générer à l'intérieur du secure element un polynôme aléatoire $Random \in R_2$ avec des coefficients binaires. Nous calculons le masque de la clé secrète $M(s_k) = (s_k \oplus Random) \bmod q$, qui est la somme coefficient par coefficient des polynômes s_k et $Random$. Ensuite on envoit le masque $M(s_k)$ et le polynôme $c[1]$ au CPU. De son coté, le CPU calcule $(c[1]\cdot M(s_k)) \bmod q$ et envoie le résultat au secure element. Ce dernier extrait la partie aléatoire du résultat reçu du CPU, en calculant :

$$[(c[1]\cdot M(s_k)) \mod q - (c[1]\cdot Random) \mod q)] \mod q = (c[1]\cdot s_k) \mod q.$$

Notons que le calcul du produit $(c[1]\cdot Random) \bmod q$ nécessite moins de mémoire que le calcul du produit $(c[1]\cdot s_k) \bmod q$. En effet, si on choisit $|q| = 56$ et $s_k \leftarrow R_q$ alors c_1 et s_k sont des polynômes de degré 2048 avec des coefficients de taille 56 bits. D'autre part $Random$ est un polynôme de degré 2048 avec des coefficients de un bit. Donc le produit de c_1 et $Random$ revient à faire que des sommes des coefficients de type long long, contrairement au produit de c_1 et $Random$ qui nécessite de faire des produits des entiers de type long long suivi de sommes.
L'Algorithme 61 présente notre méthode pour déléguer le produit au CPU. Les lignes 2, 3, 4, et 5 de l'algorithme sont exécutés une seule fois au démarrage du secure element. En outre, le même polynôme aléatoire et le masque de clé secrète sont utilisés à chaque fois qu'une délégation du produit au CPU est faite.

Algorithme 61 L'algorithme de délégation du produit de polynômes au CPU

Entrées $(c[1], s_k)$.
Sortie $c[1] \cdot s_k$.

1: **fonction** DÉLÉGATION$(c[1], s_k)$
2: Le secure element génère un polynôme random $Random \in R_2$.
3: Le secure element calcule $A = c[1] \cdot Random$.
4: Le secure element calcule $M(s_k) = s_k \oplus Random$.
5: Le secure element envoi $M(s_k)$ au CPU.
6: Le CPU calcule $B = c[1] \cdot M(s_k)$.
7: Le CPU envoi le résultat B au secure element.
8: Le secure element calcule $B - A = c[1] \cdot s_k$.
9: **retourner** $c[1] \cdot s_k$
10: **fin fonction**

4.3.2 Analyse de sécurité

Pour permettre l'envoi sécurisé de la clé secrète, nous avons proposé ci-dessus d'envoyer le masque $M(s_k) = s_k \oplus Random$ au CPU. Chaque coefficient de $M(s_k)$ est calculé en ajoutant le coefficient correspondant de la clé secrète s_k et le polynôme aléatoire $Random$, $M(s_k)[i] = (s_k[i] \oplus Random[i]) \mod q$. La valeur de chaque $M(s_k)[i]$ est soit $s_k[i]$ soit $s_k[i] + 1$, car $Random[i] \in \{0, 1\}$, alors l'attaque exhaustive sur $M(s_k)[i]$ est une attaque sur un polynôme de degré d à coefficients binaires. Comme d est égal ou supérieur à 1024, pour un niveau de sécurité 128, l'attaque en force brute sur le masque n'est pas possible.

4.4 Synthèse sur le démonstrateur

Dans ce chapitre, nous avons présenté notre implémentation pratique et autonome du cryptosystème FV en langage C pour utiliser le chiffrement homomorphe dans la vie réelle. Cette implémentation a été embarquée dans une passerelle et un cloud, permettant d'évaluer des circuits d'un seul niveau et assurant une sécurité de 128 bits. Les armateurs peuvent utiliser cette implémentation dans le port maritime pour chiffrer les données des conteneurs dans les passerelles et effectuer des calculs dans le cloud, comme la somme pondérée des conteneurs, le poids des conteneurs avant leur chargement sur le navire, ou pour effectuer la conversion monétaire des marchandises. Grâce à notre implémentation, les armateurs peuvent manipuler des données chiff tout en respectant la concurrence entre les entreprises et sans impact sur la sécurité, la confidentialité et l'anonymat. Nous avons également décrit un cas d'utilisation dans le contexte de la gestion des stocks où la comparaison est nécessaire pour l'approvisionnement des stocks des supermarchés. En conséquence, nous avons proposé une méthode permettant de prendre des décisions basées sur la compa-

raison des chiffrés homomorphes. Cette méthode consiste à déchiff les textes chiffrés de FV et à comparer les textes en clairs dans un secure element. En utilisant notre contribution, l'exécution de la fonction du déchiffrement à l'intérieur du secure element devient possible en déléguant la multiplication polynomiale au CPU.

Conclusion

Malgré la contribution majeure de la cryptographie homomorphe sur le plan théorique, elle est pour l'instant ineffi en pratique. La taille des paramètres et notamment celles des chiffrés présente un obstacle pour sa réelle mise en pratique.

Durant cette thèse industrielle, nous avons étudié le déploiement de la cryptographie homomorphe dans le cadre de l'IoT. Le début de cette thèse a été marqué par le dépôt d'un brevet sur l'utilisation du secure element dans le cloud pour aider la cryptographie homomorphe et la rendre plus efficace dans la pratique.

Dans ce travail, nous avons fait une étude détaillée de l'état de l'art de la cryptographie homomorphe, en commençant par les schémas partiellement homomorphes ensuite les schémas somewhat homorphe puis les schémas fully homomorphe. Cette étude de l'art nous a permis de considérer que les cryptosystèmes les plus efficaces sont ceux qui sont basés sur le problème RLWE. D'où la comparaison des deux cryptosystèmes FV et BGV que nous avons effectué dans le troisième chapitre. A l'issu de cette comparaison, nous avons déduit que le schéma le plus adapté à notre cas d'usage est le schéma FV. En conséquence, nous avons étudié la génération des paramètres de ce cryptosystème, qui consiste à générer les paramètres du problème RLWE. Le but de cette étude est d'aider un ingénieur, qui n'est pas spécialisé en mathématiques à choisir les paramètres d'un schéma basé sur le problème RLWE, et en particulier ceux de FV. Ainsi, on a choisi les paramètres qui permettent de générer des chiffrés de petites tailles tout en garantissant une sécurité de 128 bits et d'évaluer des cicruits d'au moins une seule multiplication. De ce fait, on a retenu des polynômes de degré 2048 avec des coefficients de 56 bits, pour construire des chiffrés de 32 koctets qui chiffre des messages de 16 bits, et évaluer un circuit d'un seul niveau de profondeur.

Dans cette thèse, nous avons présenté un démonstrateur de la cryptographie homomorphe, dans lequel nous avons décrit plusieurs cas d'usage du chiffrement homomorphe dans la vie réelle, dans le port, dans les trains, et la gestion des stocks des supermarchés. Nous avons implémenté une version homogène, et indépendante du cryptosystème homomorphe FV de Fan et Vercauteren, en respectant un niveau de sécurité 128, et en choisissant les paramètres les plus bas pour permettre d'évaluer une seule multiplication. Cette implémentation

a fait l'objet d'un travail publié dans la conférence internationale avec acte et comité de sélection Cyberworlds 2019. Ce démonstrateur permet de chiffrer les données collectées au niveau des passerelles réseau en utilisant le schéma implémenté de FV et ensuite d'eff des calculs sur ces données au niveau d'un cloud publique puis de renvoyer le résultat pour être déchiffré dans la machine de l'administrateur.

Dans ce travail, nous avons abordé aussi la problématique de la comparaison des données chiffrées en homomorphe. En cryptographie homomorphe, le résultat de la comparaison de deux chiffrés est chiffré, donc on ne peut ni effectuer un branchement conditionnel, ni envoyer une alerte à l'administrateur du réseau en se basant sur le résultat d'une comparaison. Nous avons proposé une nouvelle technique permettant de comparer deux chiffrés en utilisant un secure element, et cela en déchiffrant les données dans le secure element, et puis en comparant les données en clairs. La seule contrainte de cette méthode était l'espace mémoire du secure element, c'est pour cela qu'on a proposé de déléguer la partie la plus coûteuse au niveau mémoire (le produit du polynôme c_1 et la clé s_k) de la fonction du déchiffrement au CPU. Ce travail a été publié à la conférence internationale avec acte et comité de sélection CRISIS 2019 (international conference on risks and security of internet and systems). En conséquence, notre démonstrateur couvre les opérations indispensables dans le cadre de l'IoT comme la somme, la moyenne et la comparaison des données collectées.

Dans notre implémentation, on a omis la phase d'optimisation du temps de calcul, qu'on a considéré comme perspectives pour ce travail. D'une part Le SIMD et le traitement multi-coeur sont parmi ses perspectives qui permettent d'accélérer les temps d'exécution. D'autre part, l'application du RNS et du AMNS permet d'avoir de meilleures performances et un stockage plus optimale.

Bibliographie

Bibliographie

[1] MQTT (MQ Telemetry Transport) . https://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport.

[2] what is lora ? https://www.semtech.com/lora/what-is-lora.

[3] Miklós Ajtai. The shortest vector problem in l2 is np-hard for randomized reductions (extended abstract). Electronic Colloquium on Computational Complexity (ECCC), 4, 1997.

[4] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. J. Mathematical Cryptology, 9(3) :169–203, 2015.

[5] Jean-Claude Bajard, Julien Eynard, Anwar Hasan, and Vincent Zucca. A full rns variant of fv like somewhat homomorphic encryption schemes. Cryptology ePrint Archive, Report 2016/510, 2016. https://eprint.iacr.org/2016/510.

[6] Jean-Claude Bajard, Julien Eynard, M. Anwar Hasan, and Vincent Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. In Roberto Avanzi and Howard M. Heys, editors, Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers, volume 10532 of Lecture Notes in Computer Science, pages 423–442. Springer, 2016.

[7] Josh Benaloh. Dense probabilistic encryption. page 120–128. Workshop on Selected Areas of Cryptography, May 1994.

[8] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In Joe Kilian, editor, Theory of Cryptography, pages 325–341, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[9] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. Cryptology ePrint Archive, Report 2012/078, 2012. https://eprint.iacr.org/2012/078.

[10] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Report 2011/277, 2011. https://eprint.iacr.org/2011/277.

[11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. Cryptology ePrint Archive, Report 2011/344, 2011. https://eprint.iacr.org/2011/344.

[12] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. Cryptology ePrint Archive, Report 2011/344, 2011. https://eprint.iacr.org/2011/344.

[13] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In Phillip Rogaway, editor, Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings, volume 6841 of Lecture Notes in Computer Science, pages 505–524. Springer, 2011.

[14] Philippe Gaborit Carlos Aguilar Melchor and Javier Herranz. Additively homomorphic encryption with d-operand multiplications. Cryptology ePrint Archive, Report 2008/378, 2008. https://eprint.iacr.org/2008/378.

[15] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption : Bootstrapping in less than 0.1 seconds. Cryptology ePrint Archive, Report 2016/870, 2016. https://eprint.iacr.org/2016/870.

[16] Jean-Sebastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. Cryptology ePrint Archive, Report 2011/441, 2011. https://eprint.iacr.org/2011/441.

[17] Jean-Sebastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2011/440, 2011. https://eprint.iacr.org/2011/440.

[18] Léo Ducas and Daniele Micciancio. Fhew : Bootstrapping homomorphic encryption in less than a second. Cryptology ePrint Archive, Report 2014/816, 2014. https://eprint.iacr.org/2014/816.

[19] Léo Ducas and Daniele Micciancio. Fhew : Bootstrapping homomorphic encryption in less than a second. Cryptology ePrint Archive, Report 2014/816, 2014. https://eprint.iacr.org/2014/816.

[20] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George Robert Blakley and David Chaum,

editors, Advances in Cryptology, pages 10–18, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[21] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive, 2012:144, 2012.

[22] Craig Gentry. A Fully Homomorphic Encryption Scheme. PhD thesis, Stanford, CA, USA, 2009.

[23] Craig Gentry. A fully homomorphic encryption scheme. phd thesis, stanford university. 2009.

[24] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Stoc, volume 9, pages 169–178, 2009.

[25] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. Cryptology ePrint Archive, Report 2010/520, 2010. https://eprint.iacr.org/2010/520.

[26] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. Cryptology ePrint Archive, Report 2007/432, 2007. https://eprint.iacr.org/2007/432.

[27] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors : Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I, volume 8042 of Lecture Notes in Computer Science, pages 75–92. Springer, 2013.

[28] Mariya Georgieva. Probabilistic analysis of reduced cryptographic Euclidean networks. Theses, Université de Caen, December 2013.

[29] Shai Halevi and Victor Shoup. Algorithms in helib. In Juan A. Garay and Rosario Gennaro, editors, Advances in Cryptology – CRYPTO 2014, pages 554–571, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[30] Shai Halevi and Victor Shoup. Algorithms in helib. Cryptology ePrint Archive, Report 2014/106, 2014. https://eprint.iacr.org/2014/106.

[31] Kim Laine Hao Chen and Rachel Player. Simple encrypted arithmetic library - seal (v2.1).

[32] Alhassan Khedr, Glenn Gulak, and Vinod Vaikuntanathan. Shield : Scalable homomorphic implementation of encrypted data-classifiers. Cryptology ePrint Archive, Report 2014/838, 2014. https://eprint.iacr.org/2014/838.

[33] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. Cryptology ePrint Archive, Report 2012/230, 2012. https://eprint.iacr.org/2012/230.

[34] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. J. ACM, 60(6) :43 :1–43 :35, 2013.

[35] Carlos Aguilar Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrède Lepoint. Nfllib : Ntt-based fast lattice library. In Kazue Sako, editor, Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings, volume 9610 of Lecture Notes in Computer Science, pages 341–356. Springer, 2016.

[36] Daniele Micciancio and Shafi Goldwasser. Complexity of Lattice Problems : a cryptographic perspective, volume 671 of The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, Massachusetts, March 2002.

[37] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, Advances in Cryptology — EUROCRYPT '99, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[38] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. J. ACM, 56(6) :34 :1–34 :40, September 2009.

[39] Oded Regev. The learning with errors problem (invited survey). In Proceedings of the 25th Annual IEEE Conference on Computational Complexity, CCC 2010, Cambridge, Massachusetts, USA, June 9-12, 2010, pages 191–204. IEEE Computer Society, 2010.

[40] Adi Shamir Ronald L Rivest and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. In Jacques Stern, editor, Advances in Cryptology — EUROCRYPT '99, page 120–126. Communications of the ACM, 21(2), 1978.

[41] Kazue Sako. GOLDWASSER-MICALI ENCRYPTION SCHEME, pages 241–242. Springer US, Boston, MA, 2005.

[42] Microsoft SEAL (release 3.0). http://sealcrypto.org, October 2018. Microsoft Research, Redmond, WA.

[43] Victor Shoup Shai Halevi. Design and implementation of a homomorphic-encryption library. https ://researcher.watson.ibm.com/researcher/files/us-shaih/he-library.pdf.

[44] N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, Public Key Cryptography – PKC 2010, pages 420–443, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[45] N.P.Smartand F.Vercauteren. Fullyhomomorphicsimdoperations.Cryptology ePrint Archive, Report 2011/133, 2011. https://eprint.iacr.org/2011/133.

[46] NIST-FIPS Standard. Announcing the advanced encryption standard (aes). Federal Information Processing Standards Publication, 197(1-51) :3–3, 2001.

[47] Fred J Taylor. Residue arithmetic a tutorial with examples. Computer, (5) :50–62, 1984.

[48] Serge Tissot, Amina Belkorchi, and Thomas Richard. Techniqueforsecurely performing an operation in an iot environment, January 18 2018. US Patent App. 15/649,615.

[49] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2009/616, 2009. https://eprint.iacr.org/2009/616.

[50] P. van Emde-Boas. Another NP-complete partition problem and the complexity of computing short vectors in a lattice. Report. Department of Mathematics. University of Amsterdam. Department, Univ., 1981.

[51] Wikipedia contributors. Baby-step giant-step — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Baby-step_giant-step&oldid=903574318, 2019. [Online ; accessed 28-July-2019].

[52] Wikipedia contributors. Pollard's kangaroo algorithm — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Pollard%27s_kangaroo_algorithm&oldid=908217516, 2019. [Online ; accessed 28-July-2019].

[53] Wikipédia. Symbole de legendre — wikipédia, l'encyclopédie libre, 2019. [En ligne ; Page disponible le 30-avril-2019].

Annexe

Problèmes sous-jacents à la cryptographie basée sur la théorie des nombres

Factorisation Tout nombre supérieur à 2 peut s'écrire comme produit de facteurs premiers. Cependant, étant donné un nombre, il est généralement difficile de trouver ces facteurs premiers.

Problème RSA Le problème RSA correspond à l'extraction d'une racine e-ième modulo un entier $N = p \cdot q$. Actuellement, la meilleure façon de procéder est de factoriser le module N. Formellement, étant donnés $N = p \cdot q$, $c = m^e \bmod N$, et $e > 1$, trouver m dans la version calculatoire, décider si un tel m existe dans la version décisionnelle.

Résidualité quadratique Le problème de résidualité quadratique peut être vu comme une instance du problème RSA où $e = 2$. Dans la version décisionnelle, le but est de déterminer si c est un carré modulo $N = p \cdot q$, dans la version calculatoire, l'attaquant doit trouver x tel que $x^2 = c \bmod N$.

Problème de résidualité composée Le problème de résidualité composée consiste à décider si $c \in \mathbf{Z}_{n^2}$ est une puissance N-ième modulo N^2, où $N = p \cdot q$, et de trouver ses racines dans la version décisionnelle.

Problème de résidualité d'ordre supérieur Ce problème est une généralisation du problème de résidualité quadratique. Étant donnés des entiers $N = p \cdot q$ et $d \mid p - 1$, déterminer si $x \in Z$ est une puissance d-ième modulo N. Formellement : déterminer si $x = \alpha^d \bmod N$, et trouver α dans la version calculatoire.

Logarithme discret Étant donnés un groupe cyclique G, un générateur g de ce groupe, et $h = g^x \in G$, le problème du logarithme discret consiste à trouver x.

Problème de Diffie-Hellman Le protocole de Diffie Hellman permet l'échange des clés dans le domaine du chiffrement symétrique. Le problème de Diffie Hellman consiste à distinguer les distribitions $(g^a, g^b, g^{a \cdot b})$ et (g^a, g^b, r) modulo le nombre premier p (pour r aléatoire), et à calculer $g^{a \cdot b}$ à partir de g^a et g^b dans la version calculatoire.

Problème du diviseur commun approché : AGCD Etant donnés des entiers proches d'un multiple d'un nombre caché , le but est de trouver ce nombre. Formellement soit $x_1, x_2, \ldots, x_n$ des entiers tel que $x_i = p \cdot q_i + 2 \cdot r_i$ pour $1 \leq in \geq$, le problème consiste à retrouver p.

Sparse subset sum problem: SSP Le problème de la somme de sous-ensembles (subset sum problem) est un problème de décision NP-complet qui consiste à déterminer étant donné un ensemble d'entiers $\mathrm{E} = a_1, a_2, \ldots, a_n$, un entier $t \in \mathbf{Z}$ et $\mathrm{N} \in \mathbf{Z}$ s'il un existe un sous-ensemble de E avec un grand nombre d'éléments dont la somme de ces éléments est égale à $k \mod \mathrm{N}$.

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

TEPZZ¥ 7Z¥ _A_T

(11) **EP 3 270 321 A1**

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
17.01.2018 Bulletin 2018/03

(21) Application number: **16179444.1**

(22) Date of filing: **14.07.2016**

(51) Int Cl.:
G06F 21/62 *(2013.01)* ***H04L 9/00*** *(2006.01)*
H04L 29/06 *(2006.01)* ***H04W 4/00*** *(2018.01)*
H04W 12/02 *(2009.01)*

(84) Designated Contracting States:
AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR
Designated Extension States:
BA ME
Designated Validation States:
MA MD

(71) Applicant: **Kontron Modular Computers SAS**
83078 Toulon Cedex 9 (FR)

(72) Inventors:
- **Tissot, Serge**
  83400 Hyeres (FR)
- **Belkorchi, Amina**
  83220 Le Pradet (FR)
- **Richard, Thomas**
  42340 Veauche (FR)

(74) Representative: **Frenkel, Matthias Alexander**
Wuesthoff & Wuesthoff
Patentanwälte PartG mbB
Schweigerstrasse 2
81541 München (DE)

(54) **TECHNIQUE FOR SECURELY PERFORMING AN OPERATION IN AN IOT ENVIRONMENT**

(57) The present disclosure relates to a computing unit for securely performing an operation on encrypted data in an Internet of Things, IoT, environment. The computing unit comprises a secure element, at least one processor and at least one memory, wherein the at least one memory contains instructions executable by the at least one processor such that the computing unit is operable to obtain (S302) encrypted data collected by a sensor provided in the IoT environment, pass (S304) the encrypted data to the secure element requesting the secure element to decrypt the encrypted data and to perform an operation on the decrypted data, and obtain (S306) an encrypted or non-encrypted result of the operation from the secure element.

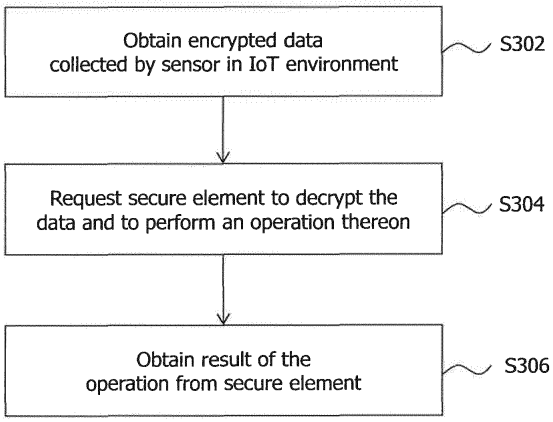

Fig. 3

EP 3 270 321 A1

Description

Technical Field

[0001] The present disclosure generally relates to Internet of Things (IoT) environments. More particularly, the present disclosure relates to a computing unit for securely performing an operation on encrypted data in an IoT environment, a method for securely performing an operation on encrypted data in an IoT environment as well as to a computer program for executing the method.

Background

[0002] Over the recent years, IoT systems have evolved as systems of interrelated physical objects equipped with computing, sensing and networking capabilities enabling the objects to collect and exchange data without requiring human-to-human or human-to-computer interaction. An IoT system allows physical objects to be sensed and controlled autonomously, enabling for a more direct integration of the physical world into computer-based systems. "Things" in the sense of IoT may refer to a wide variety of objects, such as, e.g., persons with heart monitor implants, animals with biochip transponders, automobiles with built-in sensors, or any other natural or man-made objects that can be assigned a unique identifier, typically an IP address, and that can be provided with the ability to transfer data over a network.

[0003] An IoT system typically comprises sensors and actuators that provide and receive data from a cloud through gateways or data aggregators. Analytics engines may be used to analyze the gathered data to make decisions affecting and controlling objects in the IoT environment. Analytics engines may, on the one hand, perform so called "cloud analytics" where data analytics processes are provided through a public or private cloud computing environment. Analytics engines may, on the other hand, also run in the field, e.g., on edge nodes of a network, such as on the sensors themselves, network switches or other devices outside the cloud, and perform so called "edge analytics" without a need to send the data to the cloud for analysis purposes.

[0004] In an example, industrial manufacturing machines may be connected to an IoT system and streaming data from these machines may create massive amounts of operational data. By performing analysis of the data through analytics engines, control information may be derived and applied to the machines to preserve or enhance their operational state. Also, a likely failure of a specific part of a particular machine may be identified and the machine may automatically be shut down. An alert may be sent to a plant manager so that the part can be replaced or the failure otherwise be fixed.

[0005] Typically, analytics computations are performed on plaintext of the gathered data, thereby opening security holes enabling hackers that gain access to the relevant computing systems to manipulate the control of the objects, to get know ledge of the reported data (confidentiality) and/or to silently compromise a precious database by injecting corrupted data (integrity) in the IoT environment.

Summary

[0006] Accordingly, there is a need for a technique that allows performing edge or cloud analytics in an IoT environment without exposing plaintext of the gathered data, no matter whether the data is traveling on the network, stored temporarily or permanently in a computer main memory/storage subsystem or passing through processor registers/caches/arithmetic logical units.

[0007] According to a first aspect, a computing unit for securely performing an operation on encrypted data in an Internet of Things (IoT) environment is provided. The computing unit comprises a secure element, at least one processor and at least one memory. The at least one memory contains instructions executable by the at least one processor such that the computing unit is operable to obtain encrypted data collected by a sensor provided in the IoT environment, pass the encrypted data to the secure element requesting the secure element to decrypt the encrypted data and to perform an operation on the decrypted data, and obtain a result of the operation from the secure element.

[0008] The secure element may thus be employed to securely perform desired operations on the encrypted data. In particular, the computing unit may not decrypt the encrypted data and perform the operation on the decrypted data itself, but may rather rely on the secure element to perform these tasks. In this way, it is made sure that the decrypted data, i.e., the plaintext of the encrypted data, is never exposed outside the secure environment provided by the secure element. The decrypted data may thus not be visible to the computing unit itself and, therefore, even in case of a security breach on the computing unit, it may not be possible for an intruder to gain access to the actual plaintext of the encrypted data or to inject corrupted data.

[0009] Secure elements are well known in the art. A secure element may be a tamper-proof device that provides a secure storage and execution environment. A secure element generally ensures integrity and confidentiality of its content. Secure elements may be provided as independent microcontrollers (e.g., chips) - often offered with a security certification like FIPS-140-2 or Common Criteria - and may come in different forms, such as in the form of smart cards, where the chip is embedded in a physical card, in the form of UICC (Universal Integrated Circuit Card) or SIM cards, and in the form of smart SD cards, where the chip is integrated on an SD card. Secure elements may also come as embedded secure elements, where the chip may be bonded directly to a device motherboard, for example. A secure element may host one or more applications - commonly called on-card applications - that may interact with off-card applications

provided on the host of the secure element through an application programming interface (API). On the contrary to the at least one processor of the computing unit, the secure element may be simple enough and dedicated to specialized tasks so that it is feasible to get certification by demonstrating it cannot be attacked internally or by known side channel methods.

[0010] The secure element of the computing unit may thus provide an API which may be used by the computing unit to interact with the secure element as described herein, e.g., for passing the encrypted data (single data or plurality of data) to the secure element, requesting the secure element to decrypt the data and to perform internally the requested operation on the decrypted data as well as obtaining the result of the operation from the secure element. When the computing unit passes the encrypted data to the secure element, the secure element may decrypt the encrypted data to generate plaintext of the encrypted data accordingly. The secure element may then perform the requested operation on the plaintext of the encrypted data and, once the operation is complete, the secure element may return a result of the operation to the computing unit.

[0011] The sensor which collects the encrypted data in the IoT environment may be a smart sensor, for example, i.e., a sensor which comprises built-in computing resources that allow performing predefined functions, e.g., signal processing functions or encryption functions, before passing the sensed data to a receiver. The data collected by the smart sensor may be encrypted by the sensor itself and transmitted to the computing unit accordingly. If the sensor is not a smart sensor, the sensor may sense environmental data only, wherein the sensed data is processed and encrypted by a processing unit which the sensor is connected to. The encrypted data may then be transmitted from the processing unit to the computing unit. In one specific variant, the processing unit may be the computing unit itself so that the computing unit itself is a smart sensor.

[0012] The operation requested by the computing unit may be part of an analytics computation performed in the IoT environment. In analytics computations, mathematical operations may typically be performed on the data collected by the sensor. Thus, in one implementation, the requested operation may be a mathematical operation, such as a simple arithmetic operation (e.g., addition, multiplication, etc.). Before returning the result of the operation to the computing unit, the secure element may encrypt the result of the operation so that the computing unit obtains the result of the operation in encrypted form.

[0013] In another type of request to the secure element, the requested operation may be a comparison operation between two or more portions of the encrypted data. The comparison operation may comprise operations such as "equal to", "less than", "greater than", or the like. The compared portions of the encrypted data may comprise, for example, a first portion which has been obtained from a first sensor and a second portion which has been obtained from a second sensor, or the first and the second portion may have been received subsequently from the same sensor. The result of the comparison operation may be returned from the secure element without applying further encryption so that the computing unit obtains the result of the comparison operation in unencrypted form, particularly as an unencrypted Boolean value. Although returning non-encrypted data could be considered as a possible leak of information in the case of an attack, it may sometimes be useful for the at least one processor of the computing unit to perform conditional branch based on an uncrypted Boolean result of a comparison. In this case, it may be useful that the API of the secure element offers to work with encrypted mathematical operators so that returning a plaintext Boolean does not give much information on the operands of the operators.

[0014] It is thus apparent that the computing unit described herein may be enabled to perform operations on the encrypted data without ever having access to plaintext of the encrypted data. In case of mathematical operations, operations can be performed on the sensed data although the data is not visible to the computing unit itself. The computing unit may e.g. be a smart sensor, a network switch, a gateway or another edge node of a network outside a cloud. Alternatively, the computing unit may be a physical or virtual computing unit provided in a cloud. Using one or more computing units of such type eventually allows providing end to end encryption in the IoT environment and allows encrypting data at or close to a sensor as well as performing edge or cloud analytics without exposing plaintext of the data at any time. Once the data is encrypted close to the object, any error related to human factors (wrong setup of classical security parameters, corruption of individuals, etc.) may be avoided. Protection thus provided may be called end to no end rather than end to end protection.

[0015] In case of comparison operations, the computing unit may be enabled to raise alerts and/or to control program flow since the result of a comparison operation may be visible to the computing unit (only the result may be visible, not the compared data itself). The at least one memory of the computing unit may thus further contain instructions executable by the at least one processor such that the computing unit is operable to raise an alert based on the result of the operation. In one particular API request provided by the secure element, the operation may be an encrypted comparison operation. In a pure example, 6537298 may be the encryption of the "equal to" operator and 22223333 may be the encryption of the "less than" operator, and so on. Due to a random factor in the employed encryption scheme, the same comparison operator may not always be encrypted with the same value. In this case, it may neither be revealed what type of comparison is performed nor on what data the comparison is performed.

[0016] In general, the encrypted data obtained by the computing unit may be encrypted using a homomorphic

encryption scheme or using a non-homomorphic encryption scheme. Non-homomorphic encryption schemes may include well known symmetric algorithms with single encryption/decryption keys, such as AES, or asymmetric algorithms with private and public key encryption schemes, such as RSA, for example. Plausible fully homomorphic encryption schemes, on the other hand, are known from Craig Gentry's PhD thesis "A Fully Homomorphic Encryption Scheme" of September 2009, for example. Homomorphic encryption is a form of encryption that allows computations to be carried out directly on ciphertext. A homomorphic computation of an operation (e.g., addition, multiplication, etc.) carried out on encrypted data generates an encrypted result that, when decrypted, matches the result of the same operation performed on the plaintext of the encrypted data. When both additions and multiplications could be performed on ciphertext, the encryption scheme is qualified as Fully Homomorphic Encryption (FHE). All mathematical operations can then be derived from the addition and multiplication operators.

[0017] The application of homomorphic cryptography generally requires significant processing resources (in particular, processing speed, large amounts of memory and, as a consequence, network speed) that may only exist in cloud environments even for modest plaintext operation complexity. Thus, if the encrypted data is encrypted using a homomorphic encryption scheme, typical resource consuming homomorphic computations can be avoided or accelerated by applying the above-described technique of performing operations on the encrypted data via the secure element attached to the computing unit (which can be local to the computing unit, or remote on premise). This may particularly be true for homomorphic refresh operations. Homomorphic refresh operations may form part of Gentry's cryptosystem in which it is possible to compute arbitrary numbers of additions and multiplications without increasing noise too much by "refreshing" the ciphertext periodically whenever the noise gets too large to allow correct future decryption. In these cases, each homomorphic refresh computation may be replaced by a simple decryption and fresh re-encryption of the data within the secure element. Further, in case of comparison operations, it is apparent that - although the result of a comparison can be computed using homomorphic cryptography - the comparison result (e.g., a Boolean result) is still encrypted so that the result may not be used for raising alerts and/or for implementing conditional program flows. Such measures can rather be achieved by the above-described technique of performing operations via the secure element of the computing unit.

[0018] For purposes of the decrypting and re-encrypting the data in both homomorphic and non-homomorphic encryption schemes, the secure element may store a corresponding decryption/encryption key. The decryption/encryption key may be provided to the secure element by the computing unit during a commissioning phase or during the boot process, for example, using a standard key distribution algorithm, such as asymmetric cryptography or the Diffie-Hellman algorithm.

[0019] In other implementations, particularly when the encrypted data is encrypted using a non-homomorphic encryption scheme, measures may be taken to prevent guessing plaintext values of the encrypted data. For this purpose, a plaintext value of the encrypted data may be complemented with one or more random padding values before generating the encrypted data, i.e., before actually encrypting the data. In this way, it may be prevented that encryption of numbers, such as 0, 1, 2, ... may be guessed by a hacker by submitting to the secure element calculations like "x - x" or "y / y". For the same purpose, the secure element may be configured to reject, or to silently ignore the operation and mark the result as invalid, a requested operation if the operation is predefined as an operation that allows guessing a plaintext value of the encrypted data (again, including operations such as "x - x" or "y / y").

[0020] In addition, to protect the integrity of a data element, or a set of data elements, a checksum or hash value can be added before initial encryption or result re-encryption, so that it is not possible to introduce in a database any data element without owning the initial encryption/decryption key at or close to the sensor. A plaintext value of the encrypted data may thus be supplemented with a checksum or hash value.

[0021] In a further refinement, even if a new data element cannot be generated by a hacker, integrity might still be affected in a case where some existing encrypted data would be duplicated or swapped by corrupting a program. To avoid this possibility, a traceability field may be appended to each data element (before encryption/re-encryption), initially storing a data sequence number and/or a unique device number. Each time an operation is performed on one or more data elements, leading to a resulting data, an operation (the same or alternatively just a simple addition which might include a coded operator involved) may be performed on the traceability field, so that the resulting data is tagged with the result of the operation performed on the traceability field, the tag being stored on the same traceability field. In this way, a traceability of the correct operations performed on the expected ordered data may be obtained along with each data result, making it possible at the end of the calculations to verify that the final result was obtained by processing the correct ordered set of data. The final verification of the traceability field may be made inside the secure element, before entering the resulting data into the encrypted database.

[0022] In still further implementations, the secure element may be configured to apply a timer that triggers a reset of the secure element, if a correct cryptographic key is not provided by the computing unit to the secure element before expiration of the timer. The timer may be employed as a watchdog timer that (e.g., after a predefined time period after the initialization or boot process

of the computing unit and/or on regular time periods thereafter) triggers a reset of the secure element (and optionally of the computing unit as well) to avoid analysis of the encrypted data, opportunity to export the data, opportunity to alter a program, and to prevent platform re-targeting of the computing unit to another purpose.

[0023] From the foregoing, it may be gathered that employing a secure element for securely performing an operation on encrypted data is advantageous in various respects. It will be understood, however, that, in the absence of a secure element in the computing unit, securely performing an operation on encrypted data is still possible through the use of homomorphic cryptography, though with a potentially much higher workload.

[0024] Thus, according to a second aspect, another computing unit for securely performing an operation on encrypted data in an Internet of Things (IoT) environment is provided. The computing unit comprises at least one processor and at least one memory. The at least one memory contains instructions executable by the at least one processor such that the computing unit is operable to obtain encrypted data collected by a sensor provided in the IoT environment, wherein the encrypted data is encrypted using a homomorphic encryption scheme, and perform a homomorphic computation of an operation on the encrypted data to generate an encrypted result that, when decrypted, matches a result of the operation performed on plaintext of the encrypted data.

[0025] Those features described above in relation to the computing unit of the first aspect which are applicable to the computing unit of the second aspect may be comprised by the computing unit of the second aspect as well. This particularly applies to the sensor which collects the encrypted data and to the characteristics of using homomorphic encryption schemes. Unnecessary repetitions are thus omitted. The operation performed through the homomorphic computation may be a mathematical operation, such as a simple arithmetic operation (e.g., addition, multiplication, etc.).

[0026] According to a third aspect, a method for securely performing an operation on encrypted data in an Internet of Things (IoT) environment is provided. The method is performed by a computing unit which comprises a secure element. The method comprises obtaining encrypted data collected by a sensor provided in the IoT environment, passing the encrypted data to the secure element requesting the secure element to decrypt the encrypted data and to perform an operation on the decrypted data, and obtaining a result of the operation from the secure element.

[0027] The method may be performed by the computing unit according to the first aspect. All apparatus features described herein with reference to the first aspect may thus also be embodied as functions, services or steps in the method of the third aspect.

[0028] According to a fourth aspect, a method for securely performing an operation on encrypted data in an Internet of Things (IoT) environment is provided. The method is performed by a computing unit and comprises obtaining encrypted data collected by a sensor provided in the IoT environment, wherein the encrypted data is encrypted using a homomorphic encryption scheme, and performing a homomorphic computation of an operation on the encrypted data to generate an encrypted result that, when decrypted, matches a result of the operation performed on plaintext of the encrypted data.

[0029] The method may be performed by the computing unit according to the second aspect. All apparatus features described herein with reference to the second aspect may thus also be embodied as functions, services or steps in the method of the fourth aspect.

[0030] According to a fifth aspect, a computer program product is provided. The computer program product comprises program code portions for performing the method of either the third aspect or the fourth aspect when the computer program product is executed on a computing device. The computing device may be the computing unit according to the first aspect or the second aspect accordingly. The computer program product may be stored on a computer readable recording medium, such as a semiconductor memory, DVD, CD-ROM, or the like.

[0031] All of the aspects described herein may be implemented by hardware circuitry and/or by software. Even if some of the aspects are described herein with respect to a computing unit, these aspects may also be implemented as a method or as a computer program for performing or executing the method. Likewise, aspects described as or with reference to a method may be realized by suitable components in a computing unit, or by means of the computer program.

Brief Description of the Drawings

[0032] In the following, the present disclosure will further be described with reference to exemplary implementations illustrated in the figures, in which:

Figure 1 schematically illustrates an IoT environment in which the technique of to the present disclosure may be practiced;

Figure 2 schematically illustrates a composition of a computing unit for securely performing an operation on encrypted data according to the present disclosure;

Figure 3 schematically illustrates a flowchart of a method which may be performed by the computing unit of Figure 2; and

Figure 4 schematically illustrates a flowchart of another method which may be performed by the computing unit of Figure 2.

Detailed Description

[0033] In the following description, for purposes of explanation and not limitation, specific details are set forth in order to provide a thorough understanding of the present disclosure. It will be apparent to one skilled in the art that the present disclosure may be practiced in other implementations that depart from these specific details.

[0034] Figure 1 schematically illustrates an Internet of Things (IoT) environment 100 in which analytics engines 102a, 102b and 102c are provided to perform analysis of data received from sensors 104a and 104b. Among the analytics engines 102a, 102b and 102c, analytics engine 102a is hosted on a network gateway and is configured to perform edge analytics on data received from the sensors 104a and 104b. In the illustrated example, the network gateway is equipped with a secure element. Analytics engine 102b, on the other hand, is hosted on a computing unit in a cloud computing environment and is configured to perform cloud analytics on the data received via the network gateway from the sensors 104a and 104b. The computing unit which hosts analytics en- gine 102b is equipped with a secure element. Similarly, analytics engine 102c is hosted on a computing unit in a cloud computing environment and is configured to perform cloud analytics on the data received via the network gateway from the sensors 104a and 104b as well. In contrast to analytics engine 102b, however, analytics engine 102c is not equipped with a secure element.

[0035] In the illustrated example, the sensors 104a and 104b are provided in the form of smart sensors which have built-in computing resources that allow performing signal processing functions and encryption functions, for example. The data collected by the sensors 104a and 104b is encrypted by the sensors 104a and 104b themselves and then transmitted - in encrypted form - to analytics engine 102a where the encrypted data is subjected to an analytics procedure. The encrypted data, or a portion thereof, may further be distributed from analytics engine 102a to analytics engines 102b and 102c where other parts of an overall analytics computation on the encrypted data may be performed. As a result of the analytics computation, control information may be derived and applied to controllable objects (not shown) in the IoT environment 100 to control their operational state. If, as a result of the analytics computation, alerts need to be raised, these alerts may be sent from the analytics engines 102a, 102b and 102c to a workstation 106 which may be a plant manager's workstation, for example.

[0036] Figure 2 schematically illustrates an exemplary composition of a computing unit 200 which can be used for securely performing an operation on encrypted data in the IoT environment 100. The computing unit 200 may correspond to a computing unit which hosts one of the analytics engines 102a, 102b and 102c. The computing unit 200 comprises a processor 202 and a memory 204, wherein the memory 204 contains instructions that are executable by the processor 202 such that the computing unit 200 is operable to perform the functions described herein.

[0037] It will be understood that, in a cloud architecture, such as in the case of analytics engines 102b and 102c, the computing unit 200 may be a physical computing unit, but may be a virtualized computing unit as well, such as a virtual machine (VM). It will further be understood that the computing unit 200 does not necessarily have to be a standalone computing unit, but may be implemented as a component - realized in software and/or hardware - on a single or on multiple computing units (being either physical or virtual) in the cloud environment.

[0038] The computing unit 200 may further comprise a secure element 206, which is an optional component of the computing unit 200. The secure element 206 may be employed to securely performing desired operations on the encrypted data. The computing unit 200 may not decrypt the encrypted data itself, but may rather rely on the secure element 206 to perform this task. In this way, it is made sure that the decrypted data, i.e., the plaintext of the encrypted data, is never exposed outside the secure environment provided by the secure element 206. In particular, the decrypted data may not be visible to the computing unit 200 itself and, thus, even in case of a security breach on the computing unit 200, it may not be possible for an intruder to gain access to the actual plaintext of the encrypted data.

[0039] The secure element 206 is a tamper-proof device that provides a secure storage and execution environment. The secure element 206 may be provided as an independent microcontroller (e.g., chip) and may come in different forms, such as in the form of a smart card, where the chip is embedded in a physical card, in the form of a UICC (Universal Integrated Circuit Card) or SIM card, and in the form of a smart SD card, where the chip is integrated on an SD card. The secure element 206 may also come in the form of an embedded secure element, where the chip may be bonded directly to a motherboard of the computing unit 200, for example. The secure element 206 may provide an API which may be used by the computing unit 200 to interact with the secure element 206 as described herein below with reference to Figure 3.

[0040] Figure 3 schematically illustrates a flow chart of a method for securely performing an operation on encrypted data in the IoT environment 100. The method may be performed by the computing unit 200 which - with reference to the example of Figure 1 - may be the computing unit 200 which hosts analytics engine 102a or analytics engine 102b. The method begins at step S302, at which the computing unit 200 obtains encrypted data collected by at least one of the sensors 104a and 104b. At step S304, the computing unit 200 passes the encrypted data to the secure element 206 requesting the secure element 206 to decrypt the encrypted data and to perform an operation on the decrypted data. The secure element 206 decrypts the encrypted data to generate plaintext of

the encrypted data and then performs the requested operation on the plaintext of the encrypted data. Once the operation is complete, the secure element 206 returns a result of the operation to the computing unit 200. Finally, at step S306, the computing unit 200 obtains the result of the operation from the secure element 206.

[0041] The operation requested by the computing unit 200 may be part of an analytics computation performed in the IoT environment 100. The requested operation may be a mathematical operation, such as a simple arithmetic operation (e.g., addition, multiplication, etc.). Before returning the result of the operation to the computing unit 200, the secure element 206 may encrypt the result of the operation so that the computing unit 200 obtains the result of the operation in encrypted form.

[0042] The requested operation may also be a comparison operation between two or more portions of the encrypted data. The comparison operation may comprise operations such as "equal to", "less than", "greater than", or the like. The compared portions of the encrypted data may comprise, for example, a first portion which has been obtained from the sensor 104a and a second portion which has been obtained from the sensor 104b, or the first and the second portion may have been received subsequently from one of the sensors 104a or 104b. The result of the comparison operation may be returned from the secure element 206 without applying further encryption so that the computing unit 200 obtains the result of the comparison operation in unencrypted form.

[0043] It is thus apparent that the computing unit 200 may be enabled to perform operations on the encrypted data without ever having access to plaintext of the encrypted data. In case of mathematical operations, operations can be performed on the sensed data although the data is not visible to the computing unit 200 itself. Using one or more computing units in accordance with computing unit 200, such as for hosting the analytics engines 102a and 102b, for example, eventually allows providing end to end encryption in the IoT environment 100 and allows encrypting data at the sensors 104a and 104b as well as performing edge or cloud analytics without exposing plaintext of the data at any time.

[0044] In case of comparison operations, the computing unit 200 may be enabled to raise alerts and/or to control program flows when the result of a comparison operation is visible to the computing unit 200 (only the result may be visible, not the compared data itself). The computing unit 200 may thus send an alert to the workstation 106. The operation may be an encrypted comparison operation so that it is neither revealed what type of comparison is performed nor on what data the comparison is performed.

[0045] In general, the encrypted data obtained by the computing unit 200 may be encrypted using a homomorphic encryption scheme or using a non-homomorphic encryption scheme. Non-homomorphic encryption schemes may include well known private or public key encryption schemes, such as RSA or symmetric algorithms with a unique encryption/decryption key, such as AES, for example. Plausible homomorphic encryption schemes, on the other hand, are known from in Craig Gentry's PhD thesis "A Fully Homomorphic Encryption Scheme" of September 2009, for example. Homomorphic encryption is a form of encryption that allows computations to be carried out directly on ciphertext. A homomorphic computation of an operation (e.g., addition, multiplication, etc.) carried out on encrypted data generates an encrypted result that, when decrypted, matches the result of the same operation performed on the plaintext of the encrypted data.

[0046] The application of homomorphic cryptography generally requires significant processing resources (in particular, large amounts of memory) that may only exist in cloud environments even for modest plaintext operation complexity. Thus, if the encrypted data is encrypted using a homomorphic encryption scheme, typical resource consuming homomorphic computations can be avoided or accelerated by applying the above-described technique of performing operations on the encrypted data via the secure element 206 of the computing unit 200. This may particularly be true for homomorphic refresh operations. Homomorphic refresh operations may form part of Gentry's cryptosystem in which it is possible to compute arbitrary numbers of additions and multiplications without increasing noise too much by "refreshing" the ciphertext periodically whenever the noise gets too large. In these cases, each homomorphic refresh computation may be replaced by a simple decryption, application of corresponding operations and re-encryption within the secure element 206. Further, in case of comparison operations, it is apparent that - although the result of a comparison can be computed using homomorphic cryptography - the comparison result (e.g., a Boolean result) is still encrypted so that the result may not be used for raising alerts and/or for implementing conditional program flows. Such measures can rather be achieved by application of the above-described technique of performing operations via the secure element 206 of the computing unit 200.

[0047] For purposes of the decrypting the encrypted data in both homomorphic and non-homomorphic encryption schemes, the secure element 206 may store a corresponding decryption key. The decryption key may be provided to the secure element 206 by the computing unit 200 in an initialization or boot process, for example.

[0048] When the encrypted data is encrypted using a non-homomorphic encryption scheme, measures may be taken to prevent guessing plaintext values of the encrypted data. For this purpose, a plaintext value of the encrypted data may be complemented with one or more random padding values before generating the encrypted data, i.e., before actually encrypting the data. In this way, it may be prevented that encryption of numbers, such as 0, 1, 2, ... may be guessed by submitting calculations like "x - x" or "y / y" to the secure element 206.

[0049] The secure element 206 may further be config-

ured to apply a timer that triggers a reset of the secure element 206, if a correct cryptographic key is not provided by the computing unit 200 to the secure element 206 before expiry of the timer. The timer may be employed as a watchdog timer that (e.g., after a predefined time period after the initialization or boot process of the computing unit 200 and/or on regular time periods thereafter) triggers a reset of the secure element 206 (and optionally of the computing unit 200 as well) to avoid analysis of the encrypted data and to prevent platform retargeting of the computing unit 200 to another purpose.

[0050] Figure 4 schematically illustrates a flow chart of another method for securely performing an operation on encrypted data in the IoT environment 100. The method may be performed by the computing unit 200 which, in this case, does not comprise a secure element and which - with reference to the example of Figure 1 - may be the computing unit 200 which hosts analytics engine 102c. The method begins at step S402, at which the computing unit 200 obtains encrypted data collected by at least one of the sensors 104a and 104b. The encrypted data is encrypted using a homomorphic encryption scheme. At step S404, the computing unit 200 performs a homomorphic computation of an operation on the encrypted data to generate an encrypted result that, when decrypted, matches a result of the operation performed on plaintext of the encrypted data. The operation performed through the homomorphic computation may be a mathematical operation, such as a simple arithmetic operation (e.g., addition, multiplication, etc.), for example.

[0051] Thus, in the absence of a secure element in the computing unit 200, securely performing an operation on encrypted data is still possible through the use of homomorphic cryptography, though with a potentially higher workload.

[0052] It is believed that the advantages of the technique presented herein will be fully understood from the foregoing description, and it will be apparent that various changes may be made in the form, constructions and arrangement of the exemplary aspects thereof without departing from the scope of the disclosure or without sacrificing all of its advantageous effects. Because the technique presented herein can be varied in many ways, it will be recognized that the disclosure should be limited only by the scope of the claims that follow.

Claims

1. A computing unit (200) for securely performing an operation on encrypted data in an Internet of Things, IoT, environment (100), the computing unit (200) comprising a secure element (206), at least one processor (202) and at least one memory (204), the at least one memory (204) containing instructions executable by the at least one processor (202) such that the computing unit (200) is operable to:

   obtain (S302) encrypted data collected by a sensor (104) provided in the IoT environment (100);
   pass (S304) the encrypted data to the secure element (206) requesting the secure element (206) to decrypt the encrypted data and to perform an operation on the decrypted data; and
   obtain (S306) a result of the operation from the secure element (206).

2. The computing unit (200) of claim 1, wherein the operation is a mathematical operation and the result of the operation is encrypted.

3. The computing unit (200) of claim 1, wherein the operation is a comparison operation between two or more portions of the encrypted data and the result of the operation is an unencrypted Boolean value.

4. The computing unit (200) of claim 3, the at least one memory (204) further containing instructions executable by the at least one processor (202) such that the computing unit (200) is operable to:

   raise an alert based on the result of the operation or perform a conditional branch in a program flow based on the result of the operation.

5. The computing unit (200) of claim 4, wherein the operation is an encrypted comparison operation.

6. The computing unit (200) of any one of claims 1 to 5, wherein the encrypted data is encrypted using a homomorphic encryption scheme.

7. The computing unit (200) of any one of claims 1 to 5, wherein the encrypted data is encrypted using a non-homomorphic encryption scheme.

8. The computing unit (200) of any one of claims 1 to 7, wherein a plaintext value of the encrypted data is complemented with one or more random padding values before generating the encrypted data, and/or wherein the secure element (206) is configured to reject the operation, or to silently ignore the operation and mark the result as invalid, if the operation is predefined as an operation that allows guessing a plaintext value of the encrypted data.

9. The computing unit (200) of any one of claims 1 to 8, wherein a plaintext value of the encrypted data is supplemented with a checksum or hash value, and/or
   wherein a traceability field is appended to each plaintext value of the encrypted data, the traceability field initialized to at least one of a data sequence number and a unique device number, and then updated during calculations according to the same operations, or with a simple addition of the traceability field plus

a code of the operation, performed on the data.

10. The computing unit (200) of any one of claims 1 to 9, wherein the secure element (206) is configured to apply a timer that triggers a reset of the secure element (206), or the whole computing unit (200), if a correct cryptographic key is not provided by the computing unit (200) to the secure element (206) before expiry of the timer.

11. A computing unit (200) for securely performing an operation on encrypted data in an Internet of Things, IoT, environment (100), the computing unit (200) comprising at least one processor (202) and at least one memory (204), the at least one memory (204) containing instructions executable by the at least one processor (202) such that the computing unit (200) is operable to:

obtain (S402) encrypted data collected by a sensor (104) provided in the IoT environment (100), the encrypted data being encrypted using a homomorphic encryption scheme; and
perform (S404) a homomorphic computation of an operation on the encrypted data to generate an encrypted result that, when decrypted, matches a result of the operation performed on plaintext of the encrypted data.

12. A method for securely performing an operation on encrypted data in an Internet of Things, IoT, environment (100), the method being performed by a computing unit (200) which comprises a secure element (206) and comprising:

obtaining (S302) encrypted data collected by a sensor (104) provided in the IoT environment (100);
passing (S304) the encrypted data to the secure element (206) requesting the secure element (206) to decrypt the encrypted data and to perform an operation on the decrypted data; and
obtaining (S306) a result of the operation from the secure element (206).

13. A method for securely performing an operation on encrypted data in an Internet of Things, IoT, environment (100), the method being performed by a computing unit (200) and comprising:

obtaining (S402) encrypted data collected by a sensor (104) provided in the IoT environment (100), the encrypted data being encrypted using a homomorphic encryption scheme; and
performing (S404) a homomorphic computation of an operation on the encrypted data to generate an encrypted result that, when decrypted, matches a result of the operation performed on plaintext of the encrypted data.

14. A computer program product comprising program code portions for carrying out the method of claim 12 or 13 when the computer program product is executed on a computing device.

15. A computer readable recording medium storing a computer program product according to claim 14.

Amended claims in accordance with Rule 137(2) EPC.

1. A computing unit (200) for securely performing an operation on encrypted data in an Internet of Things, IoT, environment (100), the computing unit (200) comprising a secure element (206), at least one processor (202) and at least one memory (204), the at least one memory (204) containing instructions executable by the at least one processor (202) such that the computing unit (200) is operable to:

obtain (S302) encrypted data collected by a sensor (104) provided in the IoT environment (100);
pass (S304) the encrypted data to the secure element (206) requesting the secure element (206) to decrypt the encrypted data and to perform an operation on the decrypted data;
obtain (S306) a result of the operation from the secure element (206), wherein the operation is a comparison operation between two or more portions of the encrypted data, wherein the result of the operation is an unencrypted Boolean value, and wherein the decrypted data is not visible to the computing unit (200); and
perform a conditional branch in a program flow based on the result of the operation.

2. The computing unit (200) of claim 1, the at least one memory (204) further containing instructions executable by the at least one processor (202) such that the computing unit (200) is operable to:

raise an alert based on the result of the operation.

3. The computing unit (200) of claim 1 or 2, wherein the operation is an encrypted comparison operation.

4. The computing unit (200) of any one of claims 1 to 3, wherein the encrypted data is encrypted using a homomorphic encryption scheme.

5. The computing unit (200) of any one of claims 1 to 3, wherein the encrypted data is encrypted using a non-homomorphic encryption scheme.

6. The computing unit (200) of any one of claims 1 to 5, wherein a plaintext value of the encrypted data is complemented with one or more random padding values before generating the encrypted data, and/or wherein the secure element (206) is configured to reject the operation, or to silently ignore the operation and mark the result as invalid, if the operation is predefined as an operation that allows guessing a plaintext value of the encrypted data.

7. The computing unit (200) of any one of claims 1 to 6, wherein a plaintext value of the encrypted data is supplemented with a checksum or hash value, and/or
wherein a traceability field is appended to each plaintext value of the encrypted data, the traceability field initialized to at least one of a data sequence number and a unique device number, and then updated during calculations according to the same operations, or with a simple addition of the traceability field plus a code of the operation, performed on the data.

8. The computing unit (200) of claim 7, wherein the secure element (206) is configured to apply a timer that triggers a reset of the secure element (206), or the whole computing unit (200), if a correct cryptographic key is not provided by the computing unit (200) to the secure element (206) before expiry of the timer.

9. A method for securely performing an operation on encrypted data in an Internet of Things, IoT, environment (100), the method being performed by a computing unit (200) which comprises a secure element (206) and comprising:

obtaining (S302) encrypted data collected by a sensor (104) provided in the IoT environment (100);
passing (S304) the encrypted data to the secure element (206) requesting the secure element (206) to decrypt the encrypted data and to perform an operation on the decrypted data;
obtaining (S306) a result of the operation from the secure element (206), wherein the operation is a comparison operation between two or more portions of the encrypted data, wherein the result of the operation is an unencrypted Boolean value, and wherein the decrypted data is not visible to the computing unit (200); and
performing a conditional branch in a program flow based on the result of the operation.

10. A computer program product comprising program code portions for carrying out the method of claim 9 when the computer program product is executed on a computing device.

11. A computer readable recording medium storing a computer program product according to claim 10.

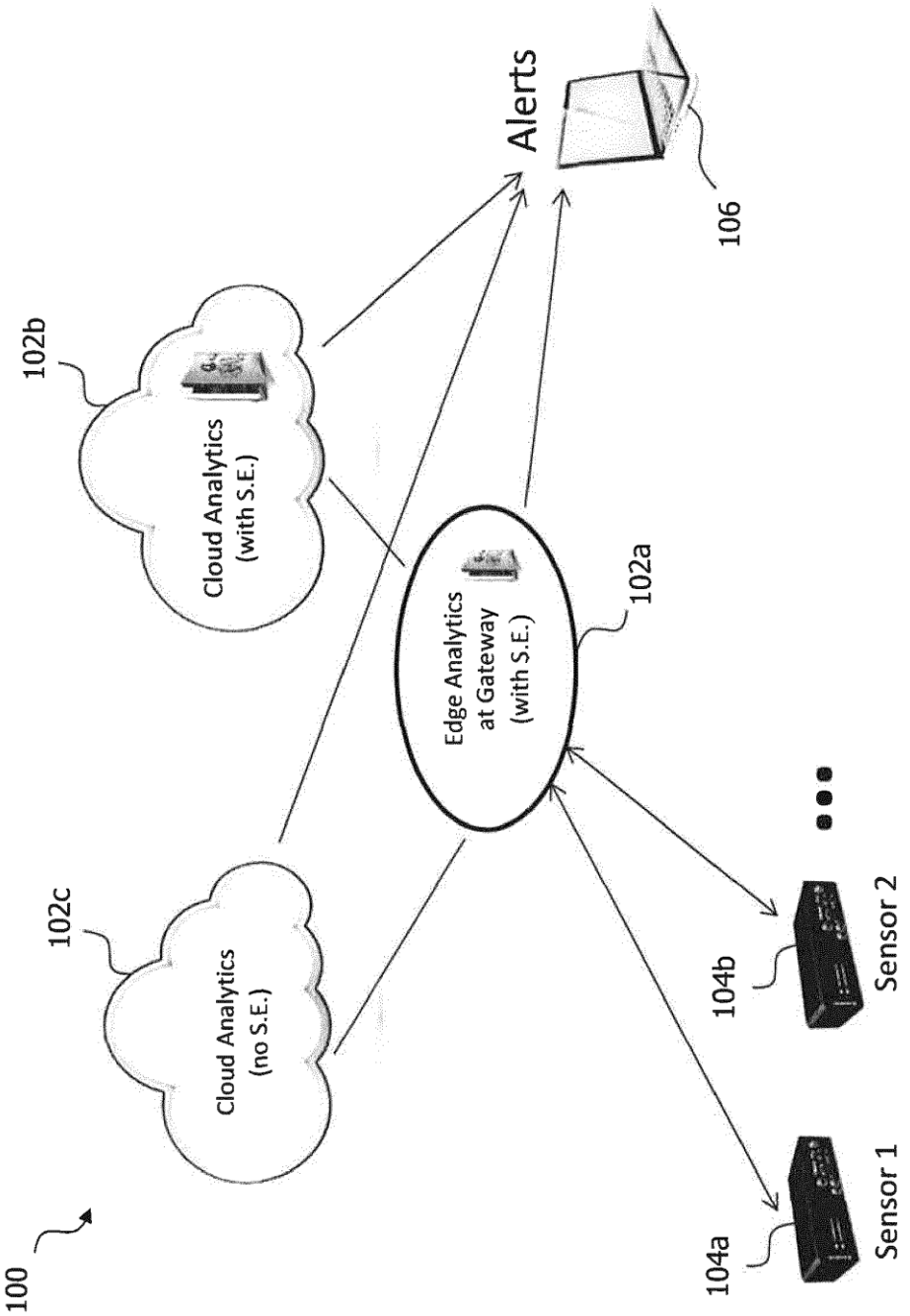
100
102c
Cloud Analytics
(no S.E.)
102b
Cloud Analytics
(with S.E.)
Edge Analytics
at Gateway
(with S.E.)
102a
Alerts
106
104a
Sensor 1
104b
Sensor 2

Fig. 1

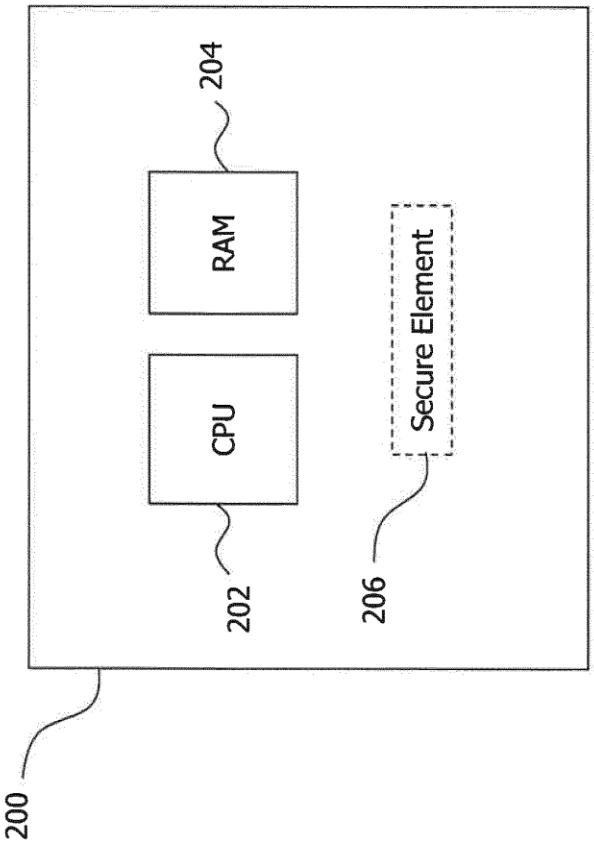

Fig. 2

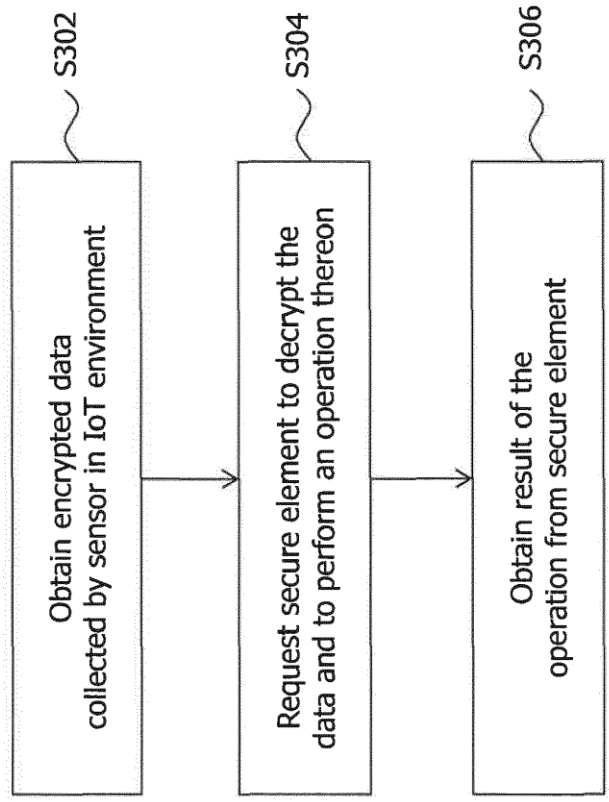
S302
Obtain encrypted data collected by sensor in IoT environment
S304
Request secure element to decrypt the data and to perform an operation thereon
S306
Obtain result of the operation from secure element

Fig. 3

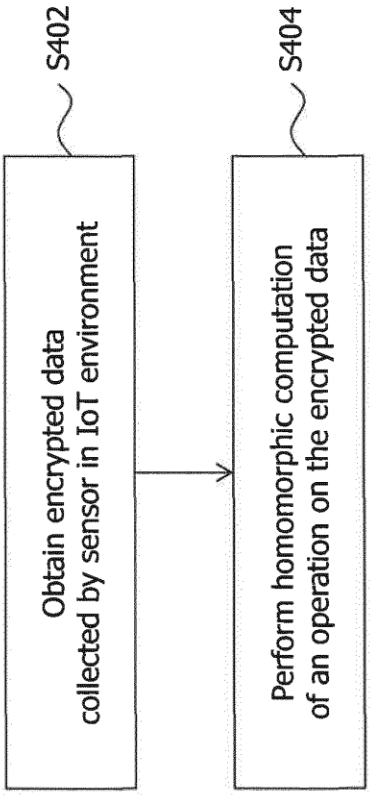
S402
Obtain encrypted data collected by sensor in IoT environment
S404
Perform homomorphic computation of an operation on the encrypted data

Fig. 4

Europäisches Patentamt
European Patent Office
Office européen des brevets

EUROPEAN SEARCH REPORT

Application Number
EP 16 17 9444

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (IPC)
X Y	US 2014/195818 A1 (NEUMANN CHRISTOPH [FR] ET AL) 10 July 2014 (2014-07-10) * paragraphs [0004], [0007], [0046] - [0049], [0055] - [0058]; claims 1,4,6,7,8,10,13; figure 2 * -----	1-8, 11-15 9,10	INV. G06F21/62 H04L9/00 H04L29/06 H04W4/00 H04W12/02
Y	US 2007/002140 A1 (BENSON GREG [US] ET AL) 4 January 2007 (2007-01-04) * paragraph [0076] - paragraph [0080]; figure Fig 6b * -----	9	
Y	US 2006/143446 A1 (FRANK ALEXANDER [US] ET AL) 29 June 2006 (2006-06-29) * paragraphs [0003], [0025], [0055]; claims 1,6,8 * -----	10	
A	SIMONE CIRANI ET AL: "Enforcing Security Mechanisms in the IP-Based Internet of Things: An Algorithmic Overview", ALGORITHMS, vol. 6, no. 2, 2 April 2013 (2013-04-02), pages 197-226, XP055319898, ISSN: 1999-4893, DOI: 10.3390/a6020197 * section 3.4. Homomorphic Encryption Schemes, section 5. Processing Data in the Encrypted Domain: Secure Data Aggregation * -----	1-15	

TECHNICAL FIELDS SEARCHED (IPC)
G06F H04L H04W

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
Munich	17 November 2016	Rothmaier, Gerrit

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 16 17 9444

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

17-11-2016

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2014195818 A1	10-07-2014	CN 103973443 A	06-08-2014
		EP 2755158 A1	16-07-2014
		EP 2755159 A1	16-07-2014
		JP 2014134799 A	24-07-2014
		KR 20140090571 A	17-07-2014
		US 2014195818 A1	10-07-2014
US 2007002140 A1	04-01-2007	US 2006291657 A1	28-12-2006
		US 2007002139 A1	04-01-2007
		US 2007002140 A1	04-01-2007
		US 2007008410 A1	11-01-2007
		US 2007011105 A1	11-01-2007
		US 2007011106 A1	11-01-2007
		US 2007011107 A1	11-01-2007
		US 2007011108 A1	11-01-2007
		US 2007022057 A1	25-01-2007
		US 2007022079 A1	25-01-2007
		US 2007030143 A1	08-02-2007
		US 2007182544 A1	09-08-2007
		US 2009210378 A1	20-08-2009
		US 2010090822 A1	15-04-2010
		US 2012150782 A1	14-06-2012
		US 2013120135 A1	16-05-2013
		US 2014058990 A1	27-02-2014
		US 2016019384 A1	21-01-2016
		WO 2006119323 A2	09-11-2006
		WO 2007027239 A1	08-03-2007
US 2006143446 A1	29-06-2006	BR PI0519080 A2	23-12-2008
		CN 101116070 A	30-01-2008
		EP 1829274 A2	05-09-2007
		JP 4945454 B2	06-06-2012
		JP 2008525892 A	17-07-2008
		KR 20070097031 A	02-10-2007
		RU 2007123617 A	27-12-2008
		US 2006143446 A1	29-06-2006
		WO 2006071630 A2	06-07-2006

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82

REFERENCES CITED IN THE DESCRIPTION

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Non-patent literature cited in the description

- A Fully Homomorphic Encryption Scheme. *Craig Gentry's PhD thesis*, September 2009 **[0016] [0045]**