Master 2 : Mathématiques et applications,
Parcours : Arithmétique Codage et Cryptologie
Université Paris 8 Vincennes Saint-Denis

Rapport de stage de fin d'études

Protocoles de calcul multipartite pour une analyse collaborative et confidentielle de données privées

Massaer SECK

Encadrants Laboratoire :
Mohamed-Lamine MESSAI
Gerald GAVIN

Tuteur université :
Julien LAVAUZELLE

Année académique : 2023 - 2024

Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué à la réussite de mon stage et à l'élaboration de ce rapport. Leur soutien et leurs conseils ont été inestimables tout au long de cette expérience.

Je souhaite particulièrement remercier Monsieur Julien Lavauzelle, mon tuteur de stage, pour son encadrement exceptionnel. Sa disponibilité, son expertise et ses conseils avisés ont grandement facilité mon apprentissage et m'ont permis de mener à bien ce projet.

Mes remerciements vont aussi à Monsieur Gavin Gerald et Monsieur Mohamed-Lamine Messai, encadrants au laboratoire ERIC. Leur accompagnement et leurs orientations techniques ont été d'une grande aide, et leur expertise a largement contribué à la qualité de mon travail.

Je tiens à exprimer ma reconnaissance envers Monsieur Stefano Morra, Monsieur Farid Mokrane et Monsieur Demba Sow pour leur précieux enseignement tout au long de ma formation. Leur engagement a été déterminant pour mon développement académique et professionnel.

Je voudrais également remercier Madame Sihem Mesnager, Monsieur Martino Berollo, ainsi que l'ensemble des enseignants du parcours ACC et le personnel administratif, que je remercie à travers Madame Erika Viator.

Un grand merci aussi à mes camarades Abdoul Aziz Ndiaye, Abdoul Ahad Fall et Cire Dioum pour les nombreuses discussions enrichissantes que nous avons eues durant le stage. Leur soutien constant a été une source de motivation et d'inspiration.

Enfin, je tiens à remercier ma famille pour son soutien indéfectible. Une mention spéciale pour ma sœur et confidente Ndeye et mon frère Saliou Gueye, dont la présence, les conseils et le soutien constant m'ont été d'une grande aide tout au long de cette aventure.

Table des matières

INTRODUCTION

Dans un monde numérique où les informations sensibles sont de plus en plus vulnérables, la protection des données et la préservation de la confidentialité sont devenues des préoccupations centrales pour les chercheurs, les entreprises, et les institutions gouvernementales. Les calculs multipartites sécurisés (Secure Multiparty Computation, SMC) et les cryptosystèmes homomorphes jouent un rôle crucial dans la résolution de ces défis en permettant à plusieurs parties de collaborer sur des données sans compromettre la confidentialité de leurs informations privées

Les calculs multipartites sécurisés se sont affirmés comme une solution novatrice pour effectuer des opérations sur des données partagées tout en garantissant que chaque partie ait accès uniquement à ses propres données et aux résultats du calcul, sans divulguer les informations des autres parties. Le **SMC** a été introduit dans les années 1980 par Andrew Yao [Yao82] à travers son célèbre problème du millionnaire. Ce problème illustre parfaitement le concept des calculs multipartites : deux millionnaires souhaitent déterminer lequel d'entre eux est le plus riche tout en préservant la confidentialité de leur fortune respective. Yao a proposé une méthode permettant aux deux millionnaires de satisfaire leur curiosité sans révéler aucune information sur leurs biens personnels. Les calculs multipartites sécurisés reposent sur des principes cryptographiques qui protègent les données tout en permettant leur traitement collaboratif. Depuis son introduction, cette approche a évolué pour intégrer des protocoles de plus en plus sophistiqués, répondant ainsi aux exigences croissantes en matière de sécurité et de performance.

En 1978, Rivest et al. [RAD+78] ont introduit un concept novateur de schémas de chiffrement, désigné sous le terme d'**homomorphismes de confidentialité**, désormais connus sous le nom de schémas de chiffrement homomorphique. Ces schémas constituent une pierre angulaire des calculs multipartites sécurisés. Initialement, ces schémas permettaient d'effectuer des calculs sur des données chiffrées, mais les opérations autorisées étaient limitées. Plus précisément, les schémas initiaux autorisaient soit une seule opération (addition ou multiplication), soit deux opérations (addition et multiplication), mais le nombre de fois où ces opérations pouvaient être répétées était limité [RSA78] [ElG85]. Ce n'est qu'en 2009 que Gentry [Gen09] a proposé le premier schéma de chiffrement homomorphique complet, capable de réaliser des additions et des multiplications

de manière illimitée, permettant ainsi l'évaluation potentielle de n'importe quelle fonction calculable sur des données chiffrées.

Cette avancée ouvre la voie à des applications concrètes où les données peuvent être traitées de manière sécurisée tout en restant protégées contre les accès non autorisés. Ce paradigme est particulièrement pertinent dans des domaines tels que l'analyse de données sensibles, les applications de santé et les systèmes de vote électronique. Dans ce cadre, mon stage au sein du laboratoire ERIC, plus précisément au sein de l'équipe SID (Systèmes d'Information Décisionnels), a été entièrement dédié au projet BI4people (Business Intelligence for the People) [BI4]. Ce projet a pour objectif de démocratiser l'accès à l'analyse interactive OLAP en proposant un service logiciel intégré destiné à simplifier l'entreposage des données. Ce service inclut l'intégration de données issues de sources variées et hétérogènes, telles que des tableaux provenant de feuilles de calcul, des documents textuels ou semi-structurés, ainsi que des données du Web, jusqu'à la visualisation et l'analyse de données simples sous forme de dataviz. Afin d'atteindre cet objectif, le service BI doit être conçu avec une approche centrée sur la confidentialité dès le début, tout en étant autonome, extrêmement simple d'utilisation, convivial et exempt de jargon technique.

L'objectif de mon stage a été de contribuer à cette ambition en explorant et en mettant en œuvre des solutions basées sur les cryptosystèmes homomorphes pour sécuriser les scénarios de Business Intelligence (BI) collaborative. Plus précisément, mes activités ont couvert les aspects suivants :

- ▷ **Découverte des cryptosystèmes homomorphes** : Analyse des cryptosystèmes et prise en main des bibliothèques cibles à partir des travaux antérieurs.
- ▷ **Étude des protocoles de calcul multipartite sécurisé** : Exploration de divers protocoles de calcul multipartite sécurisé pour évaluer leur pertinence et leur applicabilité dans le contexte du projet BI4people.
- ▷ **Développement de scénarios de BI collaborative** : Conception et mise en œuvre de protocoles de calcul multipartite visant à sécuriser les analyses collaboratives de données.
- ▷ **Évaluation des performances et de la sécurité** : Analyse des performances des protocoles en termes de temps de calcul, ainsi que l'évaluation des aspects de sécurité.
- ▷ **Intégration dans le projet BI4people** : Participation aux réunions, partage des résultats et avancées avec les autres membres de l'équipe.

Organisation du rapport

Ce rapport a pour objectif de présenter une vue d'ensemble détaillée du travail accompli et de souligner l'importance des technologies de sécurité dans le cadre des analyses de données collaboratives. Il est organisé de la manière suivante :

- ◇ Chapitre 1 : Rappels et problèmes mathématiques — Étude des problèmes mathématiques utilisés dans le cadre du stage.
- ◇ Chapitre 2 : Calcul multipartite sécurisé, chiffrement homomorphe — Introduction générale aux calculs multipartites sécurisés et étude des différentes familles de systèmes homomorphes.
- ◇ Chapitre 3 : Protocoles de calcul multipartite, contributions et résultats.

Notations

- $\mathbb{Z}_N$: groupe additif des entiers modulo N, $N \in \mathbb{N}^{\times}$.
- $\mathbb{Z}_N^*$: groupe multiplicatif des entiers inversibles modulo N, $N \in \mathbb{N}^{\times}$.
- $\phi(.)$: fonction indicatrice d'Euler
- $\lambda(.)$: fonction de Carmichael
- Pour un ensemble A, $x \xleftarrow{\$} A$ désigne le choix aléatoire d'un élément x dans A
- Pour $k \in \mathbb{N}$, $[.]_k$ désigne la réduction modulo k
- $\lceil . \rceil$: la fonction partie entière supérieure
- $\lfloor . \rceil$: la fonction arrondie entière.
- $\lfloor . \rfloor$: la fonction partie entière inférieure
- $\| \cdot \|$: la norme infinie d'un vecteur ou polynôme définie par $\|v\| = \max(v_i)$ avec v_i les coefficients de v
- $\mathcal{P}$: espace des messages clairs
- $\mathcal{C}$: espace des messages chiffrés
- Pour P_1 et P_2 deux problèmes, nous notons par $P_1 \leq P_2$ (respectivement $P_1 \equiv P_2$) la réduction polynomiale de P_1 à P_2 (respectivement l'équivalence entre P_1 et P_2).

Chapitre 1

Rappels et Problèmes Mathématiques

Dans ce premier chapitre, nous allons tout d'abord rappeler quelques définitions et résultats mathématiques qui nous serviront dans les chapitres suivants, notamment pour les schémas de chiffrement et de déchiffrement partagés de Paillier. Ensuite, nous définirons les problèmes mathématiques qui sont à la base de la sécurité des cryptosystèmes que nous utilisons, en soulignant certains liens existants entre ces problèmes.

1.1 Fonction λ de Carmichael

Nous rappelons dans cette question quelques propriétés de la fonction de Carmichael, qui seront utiles dans le schéma de Paillier 2.5.

Définition 1.1.1. *La fonction λ de Carmichael est définie par :*

1. *Si n est une puissance d'un nombre premier impair, ou si $n = 2, 4$, alors $\lambda(n) = \phi(n)$*
2. *Si $n = 2^s$ avec $s \geq 3$, alors $\lambda(n) = \frac{\phi(n)}{2}$*
3. *Si $n = p_1^{r_1}.p_2^{r_2}...p_s^{r_s}$ la décomposition de n en produit de nombre premier alors on a :*
$$\lambda(n) = ppcm\left(\lambda(p_1^{r_1}), ..., \lambda(p_r^{r_s})\right)$$

Pour $n \in \mathbb{N}$, l'entier $\lambda(n)$ est l'exposant du groupe $\mathbb{Z}_n^\times$ (le plus petit commun multiple des ordres des éléments de $\mathbb{Z}_n^\times$).

Proposition 1.1.1. *Soit $n = pq$, avec p et q premiers. On a pour tout $w \in \mathbb{Z}_{n^2}^\times$:*

$$w^{\lambda(n)} = 1 \mod n, \text{ et } w^{n\lambda(n)} = 1 \mod n^2$$

Démonstration. (voir A)

□

1.2 Quelques résultats préliminaires

Dans cette section, nous énonçons quelques résultats mathématiques préliminaires qui seront utiles pour la suite, notamment dans le chapitre 3.

Proposition 1.2.1. *Soient $p = 2p' + 1$ et $q = 2q' + 1$ deux nombres premiers sûrs, c'est-à-dire tels que p' et q' soient également premiers. Soit $N = p \times q$. Le sous-groupe des carrés dans $\mathbb{Z}_N^\times$ est cyclique d'ordre $\frac{\phi(N)}{4}$.*

Avant de démontrer cette proposition, rappelons deux résultats élémentaires de la théorie des groupes qui serviront à la preuve de cette proposition 1.2.1 et du théorème 1.2.1 ci-dessous.

Lemme 1.2.1. *Pour q un nombre premier, l'ensemble $\mathcal{S}_q$ des carrés modulo q dans $\mathbb{Z}_q^\times$ est un groupe cyclique d'ordre $\frac{q-1}{2}$.*

Lemme 1.2.2. *Soit $\mathbb{G}$ et $\mathbb{G}'$ deux groupes cycliques d'ordre n et m respectivement. Le produit $\mathbb{G} \times \mathbb{G}'$ est un groupe cyclique d'ordre nm si et seulement si n et m sont premiers entre eux.*

Démonstration. (de la proposition 1.2.1)
(voir annexe B) □

Terminons les préliminaires avec le théorème suivant.

Théorème 1.2.1. *Soit N un produit de deux nombres premiers sûrs, comme dans la proposition 1.2.1. Le sous-groupes des carrés de $\mathbb{Z}_{N^2}^\times$ est cyclique d'ordre $N\frac{\phi(N)}{4}$.*

Démonstration.

D'abord, il est facile de voir que l'application f suivante est un morphisme de groupe injectif :

$$\begin{array}{rcl} f : \mathbb{Z}_N \times \mathbb{Z}_N^* & \longrightarrow & \mathbb{Z}_{N^2}^\times \\ (a, b) & \longmapsto & (1+N)^a b^N \end{array}$$

De plus, les groupes $\mathbb{Z}_N \times \mathbb{Z}_N^*$ et $\mathbb{Z}_{N^2}^\times$ sont tous d'ordre $N\phi(N)$, ce qui implique que f est un isomorphisme.

Dans le groupe cyclique additif $\mathbb{Z}_N$ (où un carré est un multiple de 2), nous pouvons écrire :

$$x = 2\frac{x}{2} \text{ si l'entier } x \text{ est pair ou } x = 2\frac{N+x}{2} \textit{ sinon}$$

Ainsi, tout élément de $\mathbb{Z}_N$ est un carré.

D'autre part, d'après la proposition 1.2.1, le groupe des carrés de $\mathbb{Z}_N^\times$ est cyclique d'ordre $\frac{\phi(N)}{4}$.

Enfin, comme $pgcd(N, \frac{\phi(N)}{4}) = 1$, le lemme 1.2.2 permet de déduire que le groupe des carrés de $\mathbb{Z}_N \times \mathbb{Z}_N^\times$ est cyclique d'ordre $N \times \frac{\phi(N)}{4}$. Par isomorphisme, on en conclut que le groupe des carrés de $\mathbb{Z}_{N^2}^\times$ est également cyclique d'ordre $N \times \frac{\phi(N)}{4}$. □

Remarque 1.2.1. *L'isomorphisme f ci-dessus peut être généralisé. En effet, pour tout $g \in \mathbb{Z}_{N^2}^*$ d'ordre un multiple non nul de N, l'application f_g suivante est une bijection :*

$$\begin{array}{rccc} f_g : \mathbb{Z}_N \times \mathbb{Z}_N^* & \longrightarrow & \mathbb{Z}_{N^2}^\times \\ (a, b) & \longmapsto & g^a b^N \end{array}$$

L'isomorphisme de la preuve 1.2 correspond au cas particulier où l'élément $g = 1 + N \in \mathbb{Z}_{N^2}^*$, qui est d'ordre N modulo N^2 (Plus de détails dans lemme 1 [Pai99]).

1.3 Problèmes mathématiques de certains systèmes homomorphes

Dans cette section, nous examinerons les problèmes mathématiques fondamentaux qui constituent la base de la sécurité des systèmes étudiés dans le chapitre suivant. Nous définirons ces problèmes et mettrons en évidence certaines relations, de réduction ou d'équivalence existant entre eux.

1.3.1 Problèmes de la factorisation, du logarithme discret et décisionnel Diffie-Hellman

Définition 1.3.1. *Soient p et q deux nombres premiers et $N = p \times q$. Le problème de la factorisation, noté $FACT[\cdot]$, est défini comme suit :*

Problème 1.3.1. ***Factorisation***

$$FACT[N] \left\| \begin{array}{lcl} \textit{Entrées} & : & N = p \times q \\ \textit{Question} & : & \textit{calculer } p \textit{ et } q \end{array} \right.$$

Définition 1.3.2. *Soient p et q deux nombres premiers, $N = p \times q$, et e un entier tel que $pgcd(e, \phi(N)) = 1$. Le problème de cassage de RSA, noté $RSA[N, e]$ ou simplement $RSA[\cdot]$, est défini comme suit :*

Problème 1.3.2. ***Cassage de RSA***

$$RSA[N, e] \left\| \begin{array}{lcl} \textit{Entrées} & : & N, e, C = M^e \\ \textit{Question} & : & \textit{calculer } M \end{array} \right.$$

Définition 1.3.3. *Soit G un groupe cyclique multiplicatif et $g \in G$ un générateur de G. Le problème du logarithme discret, noté $DLOG[\cdot]$, est défini comme suit :*

Problème 1.3.3. $DLOG[\cdot]$

$$DLOG[G] \left\| \begin{array}{lcl} \textit{Entrées} & : & g, x = g^a \\ \textit{Question} & : & \textit{calculer } a \end{array} \right.$$

Définition 1.3.4. *Soit G un groupe cyclique multiplicatif et $g \in G$ un générateur de G. Le problème CDH (computational Diffie-Hellman), est défini comme suit :*

Problème 1.3.4. ***CDH***

$$CDH[G] \left\| \begin{array}{lcl} \textit{Entrées} & : & g, u = g^a, v = g^b \\ \textit{Question} & : & \textit{calculer } g^{ab} \end{array} \right.$$

Définition 1.3.5. *Soit G un groupe cyclique multiplicatif et $g \in G$ un générateur de G. Le problème DDH (decisional Diffie-Hellman), est le problème décisionnel de CDH défini comme suit :*

Problème 1.3.5. ***DDH***

$$DDH[G] \left\| \begin{array}{lcl} \textit{Entrées} & : & g, u = g^a, v = g^b, w = g^c \\ \textit{Question} & : & \textit{décider si } c = ab \end{array} \right.$$

1.3.2 Résiduosité composite

Dans cette sous-section, nous allons définir le problème de la résiduosité composite qui est à la base du système de Paillier que nous verrons dans la suite.
Commençons par quelques définitions.
Soit $N = p \times q$, avec p et q premiers tels que $pgcd(N, \phi(N)) = 1$.
On note :

$\mathcal{B} = \left\{ x \in \mathbb{Z}_{N^2}^{\times} : N \mid \text{ord}(x) \right\}$ *: les éléments de* $\mathbb{Z}_{N^2}^{\times}$ *d'ordre un multiplie non-nul de* N

Définition 1.3.6 (N-ième résidu)**.**
Un élément ω *de* $\mathbb{Z}_{N^2}^{\times}$ *est un* N*-ième résidu modulo* N^2 *s'il existe* $x \in \mathbb{Z}_N^{\times}$ *tel que* $x^N = \omega \mod N^2$.

Définition 1.3.7. *Soit* $g \in \mathcal{B}$*. On considère la bijection* f_g *définie dans la remarque 1.2.1. Pour* $\omega \in \mathbb{Z}_{N^2}^{\times}$*, nous appelons classe de résiduosité* $N-$*ième de* w *par rapport à* g *, noté* $[\![\omega]\!]_g$*, l'unique entier* $a \in \mathbb{Z}$ *tel qu'il existe* $y \in \mathbb{Z}_N^{\times}$ *vérifiant :* $f_g(a, b) = \omega$

Le problème du résiduosité composite noté $CLASS[\cdot]$ est défini comme suit :

Problème 1.3.6. ***résiduosité composite***

$$CLASS[N, g] \left\| \begin{array}{lcl} \textit{Entrées} & : & \omega \in \mathbb{Z}_{N^2}^{\times} \\ \textit{Question} & : & \textit{calculer } a = [\![\omega]\!]_g \end{array} \right.$$

Remarque 1.3.1. *Soit* $\omega \in \mathbb{Z}_{N^2}^{\times}$*, on a :*

$[\![\omega]\!]_g = 0 \iff \exists\, x \in \mathbb{Z}_N^{\times} : \omega = x^N \iff \omega$ *est un* N*-ième résidu modulo* N^2

Proposition 1.3.1. $\forall\, g \in \mathcal{B}, \omega_1, \omega_2 \in \mathbb{Z}_{N^2}^{\times}$,

$$[\![\omega_1 \omega_2]\!]_g = [\![\omega_1]\!]_g + [\![\omega_2]\!]_g \mod N$$

Démonstration.

Posons $a = [\![\omega_1]\!]_g$ et $b = [\![\omega_2]\!]_g$.
Par définition, il existe $y_1, y_2 \in \mathbb{Z}_N^\times$ tels que :

$$g^a y_1^N = \omega_1 \qquad et \qquad g^b y_2^N = \omega_2$$

d'où

$$g^{a+b}(y_1 y_2)^N = \omega_1\omega_2 \Longrightarrow a + b = [\![\omega_1\omega_2]\!]_g$$

□

Remarque 1.3.2. *La proposition 1.3.1 ci-dessus permet de conclure que pour tout $g \in \mathcal{B}$, $[\![\cdot]\!]_g$ est un homomorphisme de $\left(\mathbb{Z}_{N^2}^\times, \times\right)$ vers $(\mathbb{Z}_N, +)$.*

Définition 1.3.8. *On considère les mêmes notations que celles de la définition 1.3.7. On définit le problème décisionnel de résiduosité composite noté $D - CLASS[.]$ par :*

Problème 1.3.7. ***Décisionnelle résiduosité composite***

$$D - CLASS[N] \left\| \begin{array}{lcl} Entrées & : & \omega,\ g \in \mathcal{B}, a \in \mathbb{Z}_N \\ Question & : & décider\ si\ a = [\![\omega]\!]_g\ . \end{array} \right.$$

1.3.3 Hiérarchie entre $RSA[N,N]$, $CLASS[N]$ et $FACT[N]$

Dans cette section, nous allons examiner les réductions entre les problèmes $RSA[N,N]$, $CLASS[N]$ et $FACT[N]$.
On définit $\mathcal{S}_N = \left\{u < N^2 | u \equiv 1 \mod N\right\}$ et L la fonction définie par :

$$\begin{array}{rcl} L : \mathcal{S}_N & \longrightarrow & \mathbb{Z} \\ u & \longmapsto & \frac{u-1}{N} \end{array}$$

Théorème 1.3.1. *Pour tout $g_1, g_2 \in \mathcal{B}$, on a :*

$$CLASS[N, g_1] \equiv CLASS[N, g_2]$$

Démonstration.

Il suffit de montrer que si on sait calculer $[\![\omega]\!]_{g_1}$ pour un certain $g_1 \in \mathcal{B}$ alors on peut calculer $[\![\omega]\!]_{g_2}$ pour un quelconque $g_2 \in \mathcal{B}$.

Soit $g_1, g_2 \in \mathcal{B}$.
Pour tout $\omega \in \mathbb{Z}_{N^2}^{\times}$, nous avons la relation suivante :

$$[\![\omega]\!]_{g_1} = [\![\omega]\!]_{g_2}[\![g_2]\!]_{g_1} \mod N$$

En effet, soit $a = [\![\omega]\!]_{g_2}$ et $b = [\![g_2]\!]_{g_1}$.
Par définition, il existe y et z dans $\mathbb{Z}_N^{\times}$ tels que :

$$g_2^a y^N = \omega \qquad \text{et} \qquad g_1^b z^N = g_2$$

d'où :

$$\omega = (g_1)^{ab}(z^a y)^N \Longrightarrow [\![\omega]\!]_{g_1} = ab = [\![\omega]\!]_{g_2}[\![g_2]\!]_{g_1} \mod N$$

Remarquons de plus que :

$$1 = [\![g_1]\!]_{g_1}$$

Cela implique que :

$$1 = [\![g_1]\!]_{g_1} = [\![g_1]\!]_{g_2}[\![g_2]\!]_{g_1} \mod N$$

Par conséquent, $[\![g_1]\!]_{g_2} = [\![g_2]\!]_{g_1}^{-1} \mod N$.
Ainsi, si nous supposons avoir un oracle pour résoudre $Class[N, g_1]$ pour tout $\omega \in \mathbb{Z}_{N^2}^{\times}$, il suffit d'interroger l'oracle pour les entrées g_2 et ω pour obtenir $[\![\omega]\!]_{g_1}$ et $[\![g_2]\!]_{g_1}$ et ainsi trouver facilement $[\![\omega]\!]_{g_2} = [\![\omega]\!]_{g_1}[\![g_2]\!]_{g_1}^{-1}$. □

Remarque 1.3.3. *Pour $g_1, g_2 \in \mathcal{B}$, la relation*

$$1 = [\![g_1]\!]_{g_2}[\![g_2]\!]_{g_1} \mod N$$

implique $[\![g_1]\!]_{g_2}$ est inversible modulo N.

Le théorème 1.3.1 ci-dessus nous conduit à noter, dans la suite, les instances du problème $CLASS[.]$ simplement par $CLASS[N]$ au lieu de $CLASS[N, g]$, sans préciser la base g.
On dit que $CLASS[.]$ est aléatoirement auto-réductible.

Lemme 1.3.1. *Soit $\omega \in \mathbb{Z}_{N^2}^{\times}$. On a :*

$$L(\omega^{\lambda(N)} \mod N^2) = \lambda(N)[\![\omega]\!]_{1+N} \mod N.$$

Démonstration.
Comme $1 + N \in \mathcal{B}$ donc il existe, d'après l'isomorphisme de la remarque 1.2.1, $(a, b) \in \mathbb{Z}_N \times \mathbb{Z}_N^\times$ tels que $\omega = (1+N)^a b^N$.
Il s'ensuit :

$$\begin{aligned} \omega^{\lambda(N)} &= (1+N)^{\lambda(N)a} b^{\lambda(N)N} \mod N^2 \\ &\equiv (1+N)^{\lambda(N)a} \mod N^2 \text{ car } b^{\lambda(N)N} = 1 \text{ (propriété de la fonction } \lambda(\cdot)) \\ &\equiv 1 + aN\lambda(N) \mod N^2 \textit{ (formule du binôme de Newton)} \end{aligned}$$

et donc

$$L(\omega^{\lambda(N)} \mod N^2) = a\lambda(N) \mod N$$

D'autre part, l'égalité $\omega = (1+N)^a b^N$ implique que $a = [\![\omega]\!]_{1+N}$, d'où

$$L(\omega^{\lambda(N)} \mod N^2) = \lambda(N)[\![\omega]\!]_{1+N} \mod N.$$

□

Théorème 1.3.2. *Pour $e \in \mathbb{N}$ tel que $pgcd(e, \phi(N)) = 1$, on a :*

$$FACT[N] \leq RSA[N, e]$$

Démonstration.
Si on connaît la factorisation de N, on peut calculer $\phi(N)$ et l'inverse de e modulo $\phi(N)$. □

Théorème 1.3.3. *On a :*

$$CLASS[N] \leq FACT[N]$$

Démonstration.
On suppose que l'on peut factoriser le module N.
Soit $\omega \in \mathbb{Z}_{N^2}^\times$ et $g \in \mathcal{B}$.

$$L(g^{\lambda(N)} \mod N^2) = \lambda(N)[\![g]\!]_{1+N} \mod N.$$

Comme souligné dans la remarque 1.3.3, $[\![g]\!]_{1+N}$ est inversible modulo N.
En outre, $pgcd(N, \phi(N)) = 1$ implique que $\lambda(N)$ est inversible modulo N, d'où $L(g^{\lambda(N)} \mod N^2)$ est inversible modulo N.
D'autre part, d'après le lemme 1.3.1 , on a :

$$\frac{L(\omega^{\lambda(N)} \mod N^2)}{L(g^{\lambda(N)} \mod N^2)} = \frac{\lambda(N)[\![\omega]\!]_{1+N}}{\lambda(N)[\![g]\!]_{1+N}} = \frac{[\![\omega]\!]_{1+N}}{[\![g]\!]_{1+N}}.$$

D'après la démonstration de la proposition 1.3.1, on a :

$$[\![\omega]\!]_{1+N} = [\![\omega]\!]_g [\![g]\!]_{1+N}.$$

Finalement,

$$\frac{L(\omega^{\lambda(N)} \mod N^2)}{L(g^{\lambda(N)} \mod N^2)} = [\![\omega]\!]_g.$$

Ainsi, la connaissance des facteurs de N permet de calculer $\lambda(N)$ et donc $[\![\omega]\!]_g$. □

Théorème 1.3.4. *On a :*

$$CLASS[N] \leq RSA[N, N]$$

Démonstration.
On suppose qu'on peut, à l'aide d'un oracle, calculer une instance quelconque du problème $RSA[N, N]$.
Soit $\omega \in \mathbb{Z}_{N^2}^{\times}$. Il existe $(a, b) \in \mathbb{Z}_N \times \mathbb{Z}_N^{\times}$ tels que $\omega = (1+N)^a b^N$. Il s'ensuit que :

$$\omega = (1+N)^a b^N \Longrightarrow \omega \mod N = b^N.$$

Ainsi, on peut calculer b en soumettant $\omega \mod N$ à l'oracle.
Puis,

$$\omega = (1+N)^a b^N \Longrightarrow (1+N)^a = 1 + Na \equiv \omega \times b^{-N} \mod N^2.$$

$$[\![\omega]\!]_{1+N} = a = L(\omega \times b^{-N} \mod N^2) \mod N.$$

Finalement, comme le problème $CLASS[N]$ est aléatoirement auto-réductible, on peut donc calculer $[\![\omega]\!]_g$ pour tout $\omega \in \mathbb{Z}_{N^2}^{\times}$ et pour tout $g \in \mathcal{B}$. □

Nous conclurons cette section en introduisant deux problèmes qui serviront de base aux systèmes que nous développerons par la suite.

1.3.4 LWE (learning with errors) et RLWE (ring learning with errors)

LWE (learning with errors)

Soit q un nombre premier, $n, m \in \mathbb{N}$. On considère le corps $\mathbb{F}_q$.

Définition 1.3.9. *Une distribution LWE est une distribution (a_i, b_i) pour $i \in \{1, \ldots, m\}$ tels que :*

- $a_i = (a_{i,1}, \ldots, a_{i,n}) \xleftarrow{\$} \mathbb{F}_q^n$ *pour* $i \in \{1, \cdots, m\}$,
- $s = (s_1, \ldots, s_n) \xleftarrow{\$} \mathbb{F}_q^n$,
- $e = (e_1, \ldots, e_m) \xleftarrow{\$} \mathcal{X}^m$ *où* $\mathcal{X}$ *une distribution (de bruit) à valeur dans* $\mathbb{F}_q$
- $b_i = \langle a_i, s \rangle + e_i \mod q$ *pour tout* $i \in \{1, \ldots, m\}$

Si on considère la matrice $A = (a_{i,j})$, alors on a la relation linéaire :

$$b = As + e \mod q$$

On note $(\lambda, q, \mathcal{X})$ les paramètres de la distribution LWE, où λ est un paramètre de sécurité, q représente la taille du module, et $\mathcal{X}$ est la distribution de bruit.

Problème 1.3.8. ***LWE calculatoire***

$$\textbf{\textit{LWE} $-$ \textit{calculatoire}} \left\| \begin{array}{lcl} \textit{Entrées} & : & \textit{une distribution } (a_i, b_i) = (a_i, \langle a_i, s \rangle + e_i \mod q) \\ \textit{Question} & : & \textit{calculer } s \end{array} \right.$$

Problème 1.3.9. ***LWE décisionnelle***

$$\textbf{\textit{LWE} $-$ \textit{décisionnelle}} \left\| \begin{array}{lcl} \textit{Entrées} & : & \textit{une distribution LWE } \mathcal{D}_1 = (a_i, b_i) \textit{ et} \\ & & \textit{une distribution uniforme } \mathcal{D}_2(a_i', b_i') \\ \textit{Question} & : & \textit{distinguer } \mathcal{D}_1 \textit{ de } \mathcal{D}_2 \end{array} \right.$$

RLWE (ring learning with errors)

Le problème *RLWE* est une variante de *LWE* qui consiste à remplacer l'anneau $\mathbb{F}_q$ par l'anneau quotient $R_q = \mathbb{F}_q[x]/\langle f(x) \rangle$ pour un polynôme f. La distribution $\mathcal{X}$ sera à valeur dans $R = \mathbb{Z}[x]/(f(x))$. Ainsi, les a_i et s_i sont des polynômes de R_q et le produit entre a_i et s est le produit polynômial sur R_q.

Remarque 1.3.4. *Dans ce qui suit, nous abuserons souvent de la notation en représentant un polynôme* $a(x) = a_0 + a_1 x + \cdots + a_{l-1} x^{l-1}$ *de degré* $l - 1$ *dans un anneau* $\mathbb{K}[x]$ *par le vecteur* $(a_0, a_1, \ldots, a_{l-1}) \in \mathbb{K}^l$. *On note également par* $(\lambda, q, \mathcal{X}, f)$ *les paramètres de RLWE.*

Chapitre 2

Calcul Multipartite Sécurisé, Chiffrement Homomorphe

2.1 Calcul Multipartite Sécurisé

Abordons cette section en examinant le problème suivant : plusieurs entités, mutuellement méfiantes, souhaitent calculer conjointement des fonctions sur leurs données respectives sans que les données des différentes parties impliquées ne soient révélées aux autres, tout en garantissant que le résultat soit exact. Une solution simple à ce problème consisterait à faire appel à une autorité de confiance, chargée de recueillir les données de chaque entité, de calculer le résultat, puis de le transmettre aux différentes entités par le biais d'un canal sécurisé. Toutefois, cette hypothèse d'une autorité de confiance est problématique. Une alternative consiste à utiliser un protocole de calcul multipartite sécurisé.

Définition 2.1.1. *Le calcul multipartite sécurisé permet à plusieurs parties de calculer une fonction sur leurs données privées sans révéler aucune information.*

Comme mentionné dans l'introduction, ce concept a été introduit en 1982 par Yao [Yao82]. Aujourd'hui, le cadre des calculs multipartites est très vaste, couvrant un large éventail d'applications.

2.1.1 Hypothèses

Dans nos hypothèses, nous supposons que les parties impliquées dans nos protocoles ne se connaissent pas et qu'un serveur, potentiellement curieux mais respectueux du protocole, est chargé de la gestion des calculs

entre les parties et joue le rôle d'intermédiaire entre ces dernières. Nous postulons également l'existence d'une autorité de confiance responsable de la génération des clés, qui ne participe pas aux protocoles et ne peut être sollicitée par les parties pour aucune requête.

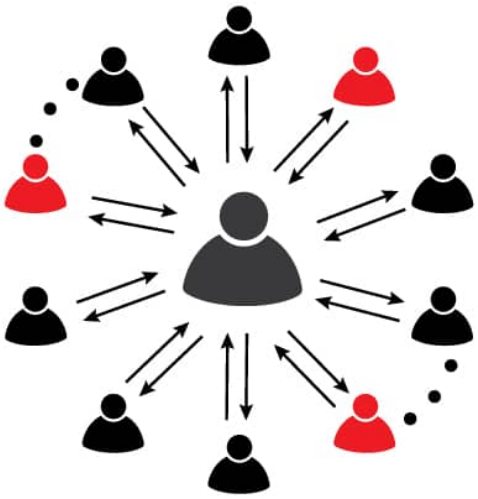

FIGURE 2.1 – Hypothèses sur les protocoles de calcul multipartite

2.1.2 Partage de secret

Dans cette section, nous aborderons le concept de partage de secret, une technique essentielle pour assurer la sécurité dans les calculs multipartites sécurisés. Il convient de noter que le partage de secret dépasse le cadre de la cryptographie et des calculs multipartites. Toutefois, il joue un rôle crucial dans le calcul multipartite sécurisé, notamment lorsqu'il s'agit de traiter des fonctions conjointes telles que le déchiffrement, qui nécessite le partage de la clé secrète.

Pour conserver une donnée secrète, il existe plusieurs stratégies. Confier un seul exemplaire du secret à une personne de confiance comporte un risque de perte totale en cas de défaillance de cette personne (disparition,

malhonnêteté, piratage,...). Distribuer des copies exactes à plusieurs personnes augmente le risque de compromission.

Pour surmonter ces risques, on peut découper le secret en plusieurs morceaux, distribués à n personnes (ou n machines, selon le contexte), avec les critères suivants :

- À partir d'un certain nombre k de personnes collaborant, le secret peut être reconstruit.
- Si moins de k personnes se réunissent, elles ne peuvent obtenir aucune information sur le secret.

Cet entier k est appelé seuil.

Il existe plusieurs manières de partager le secret, souvent basées sur des codes correcteurs d'erreurs. Dans ce rapport, nous nous limiterons à deux schémas de partage de secret : le schéma de Shamir [Sha79] et le schéma de McEliece et Sarwate.

2.1.3 Schéma de Shamir

Le schéma de partage de secret de Shamir est basé sur le résultat mathématique suivant :

Proposition 2.1.1. *Soit $\mathbb{K}$ un corps, $t \in \mathbb{N}$, et $\alpha_1, \alpha_2, \ldots, \alpha_t \in \mathbb{K}$ distincts deux à deux. Il existe un unique polynôme $P \in \mathbb{K}[X]$ de degré au plus $t-1$ tel que la courbe définie par P dans un plan affine passe par les points $(\alpha_i, P(\alpha_i))$ pour tout $1 \leq i \leq t$.*

Dans le schéma de Shamir, le secret est un élément $a_0 \in \mathbb{K}$. L'entité qui détient le secret génère un polynôme

$$P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1} \in \mathbb{K}[x],$$

où les coefficients a_i sont choisis au hasard dans $\mathbb{K}$. On a donc $a_0 = P(0)$.

Ainsi, si cette entité veut partager le secret a_0 entre $n (n \geq t)$ personnes $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$, elle choisit n coefficients $\alpha_1, \ldots, \alpha_n$ dans $\mathbb{K}^\times$, deux à deux distincts et donne à chaque partie $\mathcal{P}_i$ le couple $(\alpha_i, P(\alpha_i))$.

Toute coalition $I \in \mathcal{P}(\{1, \ldots, n\})$, composée d'au moins t parties, peut retrouver le secret a_0 par interpolation de Lagrange (en évaluant le polynôme en 0).
Notons également que toute coalition de moins de t parties n'apprend rien du secret a_0.

Remarque 2.1.1. *Rappelons que les polynômes de Lagrange sont donnés par :*

$$L_i(x) = \prod_{j=1, j\neq i}^{t} \frac{x - \alpha_j}{\alpha_i - \alpha_j} \text{ et } P(x) = \sum_{i=1}^{t} L_i(x) P(\alpha_i)$$

Si on définit

$$\beta_i = \prod_{j=1, j\neq i}^{t} \frac{\alpha_j}{\alpha_j - \alpha_i} = L_i(0) \text{ pour tout } i \in \{1, \ldots, t\}$$

alors, on a :

$$a_0 = P(0) = \sum_{i=1}^{t} P(\alpha_i)\beta_i$$

Remarque 2.1.2. *Le schéma de partage de secret de Shamir constitue un cas particulier du schéma de McEliece et Sarwate lorsqu'on utilise des codes de Reed-Solomon généralisés (voir annexe C).*

2.2 Chiffrement homomorphe

Dans cette section, nous présenterons les systèmes de chiffrement homomorphe analysés dans une étude antérieure réalisée lors d'un stage précédant le mien [Doa+23]. Nous commencerons par définir le concept de chiffrement homomorphe ainsi que les différentes familles de ces systèmes, afin de fournir une vue d'ensemble globale. Nous détaillerons ensuite chacun des systèmes étudiés.

Définition 2.2.1. *Soit $\mathbb{F}$ une fonction de chiffrement sur un ensemble de messages clairs $\mathcal{A}$ vers un ensemble de messages chiffrés $\mathcal{B}$. La fonction $\mathbb{F}$ est homomorphe, s'il existe une opération interne $*$ sur $\mathcal{A}$ et une opération interne $\circ$ sur $\mathcal{B}$ tels que :*

$$\mathbb{F}(m_1) \circ \mathbb{F}(m_2) = \mathbb{F}(m_1 * m_2), \quad \forall m_1, m_2 \in \mathcal{A}$$

Rivest, Adleman et Dertouzos sont les premiers à avoir introduit le concept de chiffrement homomorphe dans [RAD+78] en 1978.
Plus formellement, un schéma de chiffrement homomorphe est composé des quatre algorithmes suivants :
On note les paramètres par *para*.

1. Génération.HE(*para*)$\longrightarrow$ *clés* $= (pk, sk)$
   avec *pk* la clé publique et *sk* la clé secrète.

2. Chiffrement : $\mathbb{F}.HE(para, pk, message) \longrightarrow$ chiffré
3. *Déchiffrement*$.HE(para, sk, chiffré) \longrightarrow$ message
4. *Évaluation*$.HE(para, pk, (enc_1, \ldots, enc_n)) \longrightarrow \mathbb{F}.HE(para, pk, \mathbb{F}(m_1, m_2, \ldots, m_n))$, avec $enc_i = \mathbb{F}.HE(para, pk, m_i)$.

On distingue trois familles de chiffrement homomorphes :

2.2.1 Chiffrement partiellement homomorphe

Définition 2.2.2. *Un système de chiffrement partiellement homomorphe est un système dans lequel une seule opération, parmi l'addition et la multiplication, sur l'espace des textes en clair respecte la définition 2.2.1 mentionnée ci-dessus.*

Remarque 2.2.1. *Si la loi interne sur les textes en clair est l'addition (respectivement la multiplication), le système est qualifié d'additivement homomorphe (respectivement multiplicativement homomorphe). Avec ces systèmes, il est possible de répéter l'opération homomorphique autant de fois que souhaité.*

2.2.2 Chiffrement presque homomorphe

Définition 2.2.3. *Un système de chiffrement presque homomorphe est un système dans lequel on a :*

1. *une opération sur l'espace des textes en clair qui respecte la définition 2.2.1.*
2. *une autre opération $\diamond$ sur l'espace des textes en clair et une opération $\star$ sur l'espace des chiffrés telles que :*

   $$\mathbb{F}_{\approx}(m_1) \star \mathbb{F}_{\approx}(m_2) = \mathbb{F}_{\approx}(m_1 \diamond m_2), \quad \forall m_1, m_2 \in \mathcal{A} \text{ (espace des textes en clair)}$$

   avec $\mathbb{F}$ la fonction de chiffrement et $\mathbb{F}_{\approx}(m)$ correspondant à un élément de l'espace ambiant des chiffrés, qui se déchiffre en m avec une probabilité d'erreur arbitrairement croissante lors d'applications successives de la loi $\star$.

Remarque 2.2.2. *Avec les systèmes presque homomorphes, on peut effectuer des opérations d'addition et de multiplication sur les textes chiffrés, mais en un nombre limité de fois à cause de l'erreur (bruit) qui s'ajoute.*

2.2.3 Chiffrement entièrement homomorphe

Définition 2.2.4. *Un système de chiffrement homomorphe est un système permettant d'effectuer de manière illimitée des opérations d'addition et de multiplication sur des textes chiffrés.*

Remarque 2.2.3. *Contrairement aux systèmes partiellement et presque homomorphes, où il est possible d'évaluer uniquement certaines fonctions calculables, les systèmes entièrement homomorphes permettent d'évaluer l'ensemble des fonctions calculables. En effet, les opérations d'addition et de multiplication, correspondant respectivement aux portes logiques XOR et AND, permettent de réaliser toutes les fonctions possibles sous la forme de circuits booléens.*

Ces systèmes sont obtenus en combinant les schémas de chiffrement presque homomorphes avec des techniques de réduction du bruit, telles que le bootstrapping, que nous détaillerons dans le prochain chapitre.

2.3 RSA

Le système RSA, développé par Ron Rivest, Adi Shamir et Leonard Adleman [RSA78], est un algorithme de cryptographie asymétrique qui a révolutionné la sécurisation des communications numériques. Sa sécurité repose sur la difficulté le problème de la factorisation.

2.3.1 Génération des clés RSA

Algorithm 1 : Génération de clé RSA

Entrée : $t \geq 1$ paramètre de sécurité
Sortie : Une paire $(p_k, s_k) \in \mathcal{K}$.

1: choisir deux nombres premiers p et q d'au moins t bits.
2: calculer $N = p \times q$.
3: choisir $e \in \mathbb{N}$ tel que $pgcd(e, \phi(N)) = 1$.
4: calculer $d := e^{-1} \mod \phi(N)$.
5: $p_k = (N, e)$ et $s_k = d$.
6: retourner (p_k, s_k).

2.3.2 Chiffrement RSA

Algorithm 2 : Chiffrement RSA

Entrée : $M \in \mathcal{M} = \mathbb{Z}_N$ et $p_k = (N, e)$.
Sortie : $c \in \mathcal{C} = \mathbb{Z}_N$
1: calculer $c = M^e \mod N$
2: retourner c

2.3.3 Déchiffrement RSA

Algorithm 3 : Déchiffrement RSA

Entrée : $c \in \mathcal{C}$, $s_k = d$
Sortie : $m' \in \mathcal{M}$
1: calculer $m' = c^d \mod N$
2: retourner m'

Démonstration. (voir annexe D)

□

Remarque 2.3.1. *(Propriété homomorphique)*

Soit (N, e) une clé publique RSA, et M_1, M_2 deux messages en clair. Notons Enc_RSA la fonction de chiffrement. Nous avons :

$$Enc\_RSA(M_1) \times Enc\_RSA(M_2) = M_1^e \times M_2^e = (M_1 M_2)^e = Enc\_RSA(M_1 M_2)$$

La fonction de chiffrement RSA est multiplicativement homomorphe. Elle appartient à la famille des systèmes partiellement homomorphes.

2.3.4 Sécurité du chiffrement RSA

La sécurité du système repose sur la difficulté du problème de la factorisation mentionné dans la définition 1.3.1. Cependant, plusieurs attaques contre le système existent [Bon+99],[BD99], [Wie90] [Sti05]. Ainsi, en plus de choisir un module RSA de taille suffisante (3072 bits recommandés), il est crucial de prendre en compte plusieurs autres paramètres. Nous en citons quelques-uns dans la liste ci-dessous.

- ▷ Les facteurs p et q de N ne doivent pas être trop proches. Il est recommandé de choisir p et q de sorte que $|p - q| \gg N^{\frac{1}{4}}$.
- ▷ Évitez les petites valeurs de e telles que $e = 3$ [Bon+99].
- ▷ Éviter les petites valeurs de d. Par exemple :
  - ▷ $d < \frac{1}{3}N^{\frac{1}{4}}$ attaque de Wienner [Wie90]
  - ▷ $d < N^{0.292}$ attaque de Boneh et Durfee [BD99]

2.4 ElGamal

Le cryptosystème d'ElGamal [ElG85] est un système de chiffrement asymétrique basé sur le problème du logarithme discret, inventé en 1984 par Taher ElGamal.

2.4.1 Génération des clés ElGamal

Algorithm 4 : Génération des clés ElGamal

Entrée : un groupe cyclique G d'ordre n.
Sortie : une paire $(p_k, s_k) \in \mathcal{K}$

1: choisir un générateur g de G.
2: $a \xleftarrow{\$} \mathbb{Z}_n$
3: calculer $\alpha = g^a$
4: $p_k = (g, \alpha)$ et $s_k = a$
5: retourner (p_k, s_k).

2.4.2 Chiffrement ElGamal

Algorithm 5 : Chiffrement ElGamal

Entrée : $M \in G$ et $p_k = (g, \alpha)$.
Sortie : $c = (c_1, c_2) \in \mathcal{C} = G^2$

1: $k \xleftarrow{\$} \mathbb{Z}_n$
2: calculer $c_1 = g^k$ et $c_2 = M\alpha^k$
3: retourner $c = (c_1, c_2)$

2.4.3 Déchiffrement ElGamal

Algorithm 6 : Déchiffrement ElGamal

Entrée : $c \in \mathcal{C}, s_k = a$
Sortie : $m' \in \mathcal{M}$
1: calculer $\beta = c_1^a$
2: retourner $m' = c_2 / \beta$

Démonstration. Trivial. □

Remarque 2.4.1. *(Propriété homomorphique)*

Soit (g, α) une clé publique ElGamal, et M_1, M_2 des messages en clair. Notons Enc_ElGamal la fonction de chiffrement.
Soit $C = (g^k, M_1\alpha^k) = Enc\_ElGamal(M_1)$, et $C' = (g^{k'}, M_2\alpha^{k'}) = Enc\_ElGamal(M_2)$ des chiffrements de M_1 et M_2 respectivement.
En considérant le produit composant par composant comme opération sur les chiffrés, nous avons :

$$c \times c' = (g^k, M_1\alpha^k) \times (g^{k'}, M_2\alpha^{k'}) = (g^{k+k'}, M_1M_2\alpha^{k+k'}) = Enc\_ElGamal(M_1M_2)$$

Tout comme la fonction de chiffrement RSA, la fonction Enc_ElGamal est homomorphe multiplicativement et appartient à la famille des systèmes partiellement homomorphes.

2.4.4 Sécurité du chiffrement ElGamal

Théorème 2.4.1. *Sous l'hypothèse DDH, ElGamal est sûr au sens IND-CPA [1][TY98].*

2.5 Paillier

Dans cette section, nous allons présenter l'un des trois cryptosystèmes de Paillier [Pai99], qui constitue un cas particulier du système de Damgård et al. [DJN10] pour $s = 1$.
Le système de Paillier est additivement homomorphe, et il existe un protocole interactif [CDN01] permettant d'effectuer le produit entre chiffrés que nous détaillerons dans le chapitre 3.

1. CPA (indistinguishable chosen-plaintext attack) pour indiscernable sous une attaque en clair choisie)

2.5.1 Génération des clés

Algorithm 7 : Génération des clés Paillier

Entrée : $t \geq 1$ paramètre de sécurité
Sortie : une paire de clés (p_k, s_k)

1: choisir deux nombres premiers p et q d'au moins t bits et calculer $N = p \times q$.
2: **si** $pgcd(N, \phi(N)) \neq 1$ **alors**
3: revenir à l'étape 1
4: **fin du si**
5: calculer $\lambda(N)$
6: choisir un élément g d'ordre N modulo N^2.
7: $p_k = (N, g)$ et $s_k = \lambda(N)$
8: retourner (p_k, s_k).

2.5.2 Chiffrement

Algorithm 8 : Chiffrement Paillier

Entrée : $M \in \mathbb{Z}_N$ et $p_k = (N, g)$.
Sortie : $c \in \mathcal{C} = \mathbb{Z}_{N^2}^*$

1: $x \xleftarrow{\$} \mathbb{Z}_N^\times$
2: calculer $c = g^M x^N \mod N^2$
3: retourner c

2.5.3 Déchiffrement

Algorithm 9 : Déchiffrement Paillier

Entrée : $c \in \mathcal{C}, s_k = \lambda(n)$
Sortie : $m' \in \mathcal{M}$

1: calculer $u = L(c^{\lambda(N)} \mod N^2)$
2: calculer $v = L(g^{\lambda(N)} \mod N^2)$
3: retourner $m' = \frac{u}{v} \mod N$

Démonstration. (voir annexe E)

□

Remarque 2.5.1. *(Propriétés homomorphique)*

Soit (N, g) une clé publique ElGamal, et M, M_1, M_2 des messages en clair. Notons Enc_Pai la fonction de chiffrement de Paillier.
Nous avons les propriétés homomorphiques suivantes :

$$\begin{cases} Enc\_Pai(M_1) \times Enc\_Pai(M_2) & = & Enc\_Pai(M_1 + M_2) \\ Enc\_Pai(M_1) / Enc\_Pai(M_2) & = & Enc\_Pai(M_1 - M_2) \\ [Enc\_Pai(M)]^k & = & Enc\_Pai(k \times M)\ ,\ \forall k \in \mathbb{N} \end{cases}$$

La fonction de chiffrement de Paillier, bien qu'elle appartienne à la famille des systèmes partiellement homomorphes comme RSA et ElGamal, présente des propriétés homomorphiques intéressantes, notamment sur les types d'opérations qu'on peut faire sur les chiffrés

2.5.4 Sécurité du chiffrement de Paillier

Pour M un message en clair et (N, g) une clef publique, nous avons un texte chiffré c de M défini par :

$$c = g^M x^N = f_g(M, x)$$

C'est-à-dire,

$$M = CLASS[N, g](c)$$

Théorème 2.5.1. *Sous l'hypothèse du problème $CLASS[N, g]$, la fonction de chiffrement Paillier est à sens unique.*

Remarque 2.5.2. *Damgård et Jurik [DJ00] ont proposé des modifications pour améliorer le schéma de Paillier. En effet,*

- ▷ *l'élément $g = 1 + N$ est d'ordre N modulo N^2. Avec un tel g, on a $g^M = (1 + N)^M \equiv 1 + NM \mod N^2$ et donc, au lieu de faire une exponentiation, une seule multiplication suffit.*
- ▷ *choisir p et q sous la forme $p = 2p' + 1$ et $q = 2q' + 1$ avec p' et q' premiers entre eux permet de satisfaire la condition $(N, \phi(N)) = 1$.*
- ▷ *de plus, prendre une clé secrète d telle que :*

$$d \equiv 0 \mod \lambda(N) \text{ et } d \equiv 1 \mod N$$

simplifie le déchiffrement, car nous avons simplement :

$$L(c^d \mod N^2) = L\left[\left((1 + N)^M x^N\right)^d \mod N^2\right] = M$$

Nous présentons dans le chapitre 3 nos résultats d'implémentation des systèmes de Paillier ainsi que de Damgård et Jurik, en comparant leurs algorithmes de chiffrement et de déchiffrement.

Remarque 2.5.3. *(Choix du secret d)*
Dans nos implémentations, nous avons choisi le secret d de la façon suivante : Comme $p = 2p' + 1$ *et* $q = 2q' + 1$ *avec* p' *et* q' *premiers, on a* $\phi(N) = 4p'q'$ *et* $\lambda(N) = 2p'q'$ *et* $pgcd(N, \lambda(N)) = 1$.
Ainsi, il existe $(a, b) \in \mathbb{Z}^- \times \mathbb{N}$ *tels que :*

$$aN + b\lambda(N) = 1$$

Si on pose $d = b\lambda(N)$*, alors on a :*

$$\begin{cases} d > 0 \\ d \equiv 1 \mod N \\ d \equiv 0 \mod \lambda(N) \end{cases}$$

Un tel d convient pour le secret s_k.

Notre choix du secret d est libre et résulte de la question que nous nous sommes posée sur l'existence d'un tel d. Remarquant que N *et* $\lambda(N)$ *sont premiers entre eux, ainsi que les conditions imposées sur d concernant* N *et* $\lambda(N)$*, nous avons naturellement pensé à une relation de Bézout entre les deux. De plus, imposer que b soit dans* $\mathbb{N}$ *permet de garantir que d est positif.*

Proposition 2.5.1. *D'après la proposition 1.3.1, la sécurité du schéma de Damgård et Jurik avec* $g = 1 + N$ *et celle du schéma de Paillier sont équivalentes.*

Démonstration. La démonstration découle de la proposition 1.3.1. □

2.6 BFV

En 2012, J. Fan et F. Vercauteren [FV12] ont adapté le schéma proposé par Brakerski [Bra12] en le transposant du problème *LWE* au problème *RLWE*.

2.6.1 Paramètres du système BFV

Nous utilisons les mêmes notations que dans la section 1.3.4. Pour un paramètre de sécurité λ, nous considérons le quadruplet $(\lambda, q, \mathcal{X}, f)$

comme les paramètres du problème $RLWE$, où $f(x)$ est un polynôme cyclotomique. Nous utilisons également les notations $R_q = \mathbb{Z}_q[x]/(f(x))$ et $R = \mathbb{Z}[x]/(f(x))$. L'espace des clairs est $R_t = R/tR$ pour $t \in \mathbb{N}$, où les coefficients sont pris entre $\left(-\frac{t}{2}, \frac{t}{2}\right]$, et nous définissons $\Delta = \lfloor \frac{q}{t} \rfloor$.
Pour un polynôme $p \in R$, on définit la norme infinie $\|p\|$ de p en considérant p comme un vecteur défini par ses coefficients.

2.6.2 Génération des clés BFV

Algorithm 10 : génération des clés BFV

Entrée : paramètre $(\lambda, q, \mathcal{X}, f)$
Sortie : un couple (p_k, s_k)

1: $s \xleftarrow{\$} \mathcal{X}$
2: $e \xleftarrow{\$} \mathcal{X}$
3: $a \xleftarrow{\$} R_q$
4: $p_k = (p_0, p_1) = ([-(a.s + e)]_q, a)$
5: $s_k = s$
6: retourner (p_k, s_k)

2.6.3 Chiffrement

Algorithm 11 : Chiffrement BFV

Entrée : un message $m \in R_t$, une clé publique $pk = (p_0, p_1)$
Sortie : un chiffré $c = (c_0, c_1) \in Rq^2$

1: $u, e_1, e_2 \xleftarrow{\$} \mathcal{X}$
2: calculer $c_0 = [p_0.u + e_1 + \Delta.m]_q$ et $c_1 = [p_1.u + e_2]_q$
3: retourner $c = (c_0, c_1)$

2.6.4 déchiffrement

Algorithm 12 : déchiffrement BFV

Entrée : un chiffré $c \in Rq^2$, une clé privée $s_k = s$
Sortie : un message $m' \in R_t$
1: $\omega = [c_0 + c_1.s]_q$
2: retourner $m' = \left[\lfloor \frac{t}{q}.\omega \rceil\right]_t$

Preuve du déchiffrement.
On a :

$$\begin{aligned} \omega &= [c_0 + c_1.s]_q \\ &= [p_0.u + e_1 + \Delta.m + (p_1.u + e_2)s]_q \\ &= [-(a.s + e).u + e_1 + \Delta.m + (a.u + e_2)s]_q \\ &= [-a.s.u - e.u + e_1 + \Delta.m + a.u.s + e_2.s]_q \\ &= [\Delta.m - e.u + e_1 + e_2.s]_q \end{aligned}$$

Posons $v = -e.u + e_1 + e_2.s$ (le bruit). Alors, on a $\omega = [\Delta.m + v]_q$.
Il existe $r \in R$ tel que : $\omega = \Delta.m + v + q.r$.
Posons $\epsilon := \frac{q}{t} - \Delta$, alors $\epsilon < 1$.
Il suit :

$$\begin{aligned} \frac{t}{q}\omega &= \frac{t}{q}\left[(\frac{q}{t} - \epsilon)m + v + q.r\right] \\ &= m + \frac{t}{q}(v - \epsilon.m) + t.r \end{aligned}$$

Ainsi, on a $\left[\lfloor \frac{t}{q}.\omega \rceil\right]_t = m$ à condition que $\|\frac{t}{q}(v - \epsilon.m)\| < \frac{1}{2}$. □

Remarque 2.6.1. *On observe que le déchiffrement d'un message m avec l'algorithme de déchiffrement donne un message de la forme $m' = m + erreur$.*
*Ainsi, l'exactitude du déchiffrement dépend du terme **erreur**. La proposition suivante fournit une condition sur la distribution $\mathcal{X}$ qui garantit l'exactitude du déchiffrement.*

Avant de donner la proposition, nous introduirons la définition suivante, en utilisant les mêmes notations que celles employées dans la sous-section 2.6.1 concernant les paramètres.

Définition 2.6.1. *On définit le facteur d'expansion de R noté δ_R par :*

$$\delta_R = \max\left\{\frac{\|a.b\|}{\|a\|\|b\|} : a, b \in R\right\}$$

Proposition 2.6.1. *Soit $B > 0$ tel que $\|\mathcal{X}\| = \{\|p\| : p \in \mathcal{X}\} < B$.*
Si

$$2.\delta_R.B^2 + B < \Delta/2$$

alors le déchiffrement est correcte.

Démonstration. On a

$$\omega = [\Delta.m + v]_q \text{ avec } v = -e.u + e_1 + e_2.s$$

Pour tout $a, b \in R$ on a :

$$\|a.b\| \leq \delta_R.\|a\|\|b\| \leq \delta_R B^2$$

donc $\|v\| = \| - e.u + e_1 + e_2.s\| \leq 2\delta_R B^2 + B$.
Pour les détails du reste de la preuve, nous renvoyons à [FV12]. □

Proposition 2.6.2. *(Propriétés homomorphiques)*
Soit m_1, m_2 deux messages clairs, et $c_1 = (c_0^1, c_1^1)$, $c_2 = (c_0^2, c_1^2)$ les textes chiffrés de m_1 et m_2 respectivement.

$$\begin{cases} (c_0^1, c_1^1) &= ([p_0.u + e_1 + \Delta.m_1]_q, [p_1.u + e_2]_q) \\ (c_0^2, c_1^2) &= ([p_0.u' + e_1' + \Delta.m_2]_q, [p_1.u' + e_2']_q) \end{cases}$$

On a :

$$(c_0^1 + c_0^2, c_1^1 + c_1^2) = (\left[p_0.(u+u') + (e_1+e_1') + \Delta.(m_1+m_2)\right]_q, \left[p_1.(u+u') + (e_2+e_2')\right]_q)$$

Si l'on définit

$$c_0 = c_0^1 + c_0^2 \text{ et } c_1 = c_1^1 + c_1^2$$

alors le couple $c = (c_0, c_1)$ est un texte chiffré valide du message $m_1 + m_2$.

Remarque 2.6.2. *Dans [FV12], une manière simple de manipuler les opérations sur les chiffrés est d'associer à chaque chiffré $c = (c_0, c_1) \in R_q^2$ l'élément $c(s) = c_0 + c_1 \cdot s \in R_q$.*
Ainsi avec cette correspondance, pour deux textes chiffrés $c = (c_0 + c_1)$ et $c' = (c_0' + c_1')$ correspondant respectivement aux messages m et m' on obtient :

$$c(s) + c'(s) = c_0 + c_0' + (c_1 + c_1') \cdot s \text{ dans } R_q$$

qui est un texte chiffré valide du message $m + m'$.
Pour un chiffré $c = (c_0, c_1)$, bien que $c(s)$ soit un élément de R_q, nous allons, dans le cadre des opérations homomorphiques, interpréter $c(s)$ comme un polynôme de degré 1 en s. Autrement dit, nous considérerons les polynômes c_0 et c_1 comme les coefficients du polynôme $c(s)$.

Examinons maintenant la multiplication.
Soient $c(s) = c_0 + c_1(s)$ et $c'(s) = c'_0 + c'_1(s)$ correspondant à deux chiffrés c et c' de deux messages respectifs m et m'.
On a :

$$c(s) \times c'(s) = (c_0 + c_1(s)) \times (c'_0 + c'_1(s)) = c_0c'_0 + (c_0c'_1 + c_1c'_0)s + c_1c'_1s^2$$

On obtient un produit de degré 2 en s qu'on ne peut pas directement associer à un couple dans R_q. Comme vu précédemment dans la preuve du déchiffrement, pour un chiffré $c'' = (c''_0, c''_1)$, il est nécessaire de calculer $c''_0 + c''_1 \cdot s$ dans l'algorithme de déchiffrement. Pour remédier à ce problème, une procédure appelée relinéarisation est proposée dans [FV12]. Cette procédure permet de réduire un chiffré de degré 2 à un chiffré de degré 1, et nécessite l'introduction d'une clé de relinéarisation *rlk*. Plus précisément, l'objectif de cette procédure est de transformer $c_0 + c_1 \cdot s + c_2 \cdot s^2$, en un polynôme de degré 1 en trouvant c'_0 et c'_1 tels que :

$$[c_0 + c_1 \cdot s + c_2 \cdot s^2]_q = [c'_0 + c'_1 \cdot s + r]_q$$

avec $\|r\|$ est petit.

2.6.5 Relinéarisation

On considère les paramètres définis dans la section 2.6.1.
Nous présentons ici l'une des deux versions de la relinéarisation décrites dans [FV12], où la clé de relinéarisation *rlk*) est définie, en choisissant une base $T \in \mathbb{N}$, comme suit

$$rlk := \left[\left(\left[-(a_i \cdot s + e_i) + T^i \cdot s^2 \right]_q, a_i \right) : i \in [0..l] \right], a_i \xleftarrow{\$} R_q, e_i \xleftarrow{\$} \mathcal{X}, l = \lfloor \log_T(q) \rfloor$$

Pour transformer le chiffré $c = c_0 + c_1 \cdot s + c_2 \cdot s^2$, on convertit c_2 dans la base T :

$$c_2 = \sum_{i=0}^{l} T^i \cdot c_2^{(i)} \mod q \text{ où les } c_2^{(i)} \in R_T$$

Avant de poursuivre la procédure, examinons un exemple de conversion :

Exemples 2.6.1. *Prenons $q = 5$, $T = 3$ et un polynôme f de degré 8. Les polynômes sont donc de degré au plus égal à 7 et $l = \lfloor \log_3(5) \rfloor = 1$. Considérons le polynôme $c(x) = 3x^4 + 2x^3 + 4x^2 + 3x + 1 \in R_5$. Les coefficients du polynôme c en base 3 sont :*

$$1 = 1, \quad 2 = 2, \quad 3 = 10 \quad et \quad 4 = 11$$

ainsi, nous pouvons écrire :

$$3x^4 = 1.3^1x^4,\ 2x^3 = 2.3^0x^3,\ 4x^2 = (1.3^1 + 1.3^0)x^2,\ 3x = 1.3^1x \text{ et } 1 = 1.3^0$$

Il suit :

$$c(x) = (1+x^2+2x^3).3^0 + (x+x^2+x^4).3^1 \text{ i.e } c^{(0)} = 1+x^2-x^3 \text{ et } c^{(1)} = x+x^2+x^4$$

Revenons maintenant à la procédure.
Après conversion, nous définissons ensuite c'_0 et c'_1 comme suit :

$$c'_0 = \left[c_0 + \sum_{i=0}^{l} rlk[i][0] \cdot c_2^{(i)}\right]_q \text{ et } c'_1 = \left[c_1 + \sum_{i=0}^{l} rlk[i][1] \cdot c_2^{(i)}\right]_q .$$

Alors, on obtient :

$$\begin{cases} c'_0 = \left[c_0 - \underbrace{\sum_{i=0}^{i=l} a_i \cdot c_2^{(i)} s}_{*} - \sum_{i=0}^{i=l} e_i c_2^{(i)} + \underbrace{\sum_{i=0}^{l} T^i c_2^{(i)} \cdot s^2}_{c_2 \cdot s^2}\right]_q \\ c'_1 \cdot s = \left[c_1 \cdot s + \underbrace{\sum_{i=0}^{i=l} a_i \cdot c_2^{(i)} \cdot s}_{*}\right] \end{cases}$$

d'où

$$c'_0 + c'_1 \cdot s = c_0 + c_1 \cdot s + c_2 \cdot s^2 - \sum_{i=0}^{l} e_i c_2^{(i)} \mod q$$

Il suffit donc de prendre

$$r = \sum_{i=0}^{l} e_i c_2^{(i)}$$

Proposition 2.6.3. *On a :*

$$\|r\| \leq (l+1)B \cdot T \cdot \delta_R/2$$

Démonstration. On a :

$$\| \sum_{i=0}^{l} e_i c_2^{(i)} \| \leq \sum_{i=0}^{l} \| e_i c_2^{(i)} \|$$

Pour tout $i \in \{0, \ldots, l\}$, les coefficients des polynômes e_i et $c_2^{(i)}$ sont respectivement dans R_q et R_T donc on a :

$$\|e_i c_2^{(i)}\| \leq \delta_R \cdot \|e_i\| \cdot \|c_2^{(i)}\| \leq \delta_R \cdot B\frac{T}{2}$$

finalement

$$\|r\| \leq (l+1)\delta_R \cdot B\frac{T}{2}$$

□

Remarque 2.6.3.

- ▷ *La taille de la clé de relinéarisation est donnée par l.*
- ▷ *Le nombre de multiplications effectuées lors de la relinéarisation est $2l$, chaque multiplication impliquant un facteur de R_q et un facteur de R_T.*
- ▷ *Notons enfin que les couples présents dans la clé de relinéarisation ne sont pas des échantillons de la distribution RLWE. Par conséquent, nous faisons l'hypothèse que le schéma reste sécurisé, c'est-à-dire que l'accès de l'adversaire à cette clé ne compromet pas la sécurité du schéma.*

Après L multiplications, le bruit est approximativement de l'ordre de $2B\delta_R^{2L+1}t^L$. Comme le déchiffrement est possible uniquement lorsque ce bruit est inférieur à $\Delta/2$, cela implique que le schéma ne peut réaliser qu'un nombre limité de multiplications, ce qui en fait un schéma presque homomorphe. Pour atteindre un schéma complètement homomorphe, l'utilisation de la technique de bootstrapping est nécessaire.

2.6.6 Le bootstrapping

L'approche de bootstrapping de Gentry [Gen09] vise à réduire le bruit avant d'atteindre le niveau maximal au-delà duquel le déchiffrement échoue.

L'idée de Gentry peut être résumée comme suit : supposons que le seuil de bruit pour le déchiffrement est N. Si nous avons un algorithme **Rechiffre** qui prend un chiffré $F(m)$ (où F est la fonction de chiffrement) avec un bruit e et produit un nouveau chiffré $F(\tilde{m})$ qui chiffre également m, mais avec un paramètre de bruit $\tilde{e}$ beaucoup plus petit que e (permettant le déchiffrement), alors cet algorithme **Rechiffre** est suffisant pour construire un schéma complètement homomorphe à partir d'un schéma presque homomorphe. En effet, il suffit, avant d'additionner ou de multiplier $F(m)$ et $F(m')$, d'appliquer **Rechiffre** à $F(m)$ et $F(m')$ pour garantir que leurs paramètres de bruit soient suffisamment petits pour que le paramètre de

bruit des chiffrés $F(m + m')$ et $F(m \times m')$ soit inférieur à N.

Ainsi, grâce à la technique de bootstrapping et aux algorithmes 2.6.2, 2.6.3 et 2.6.4, le schéma BFV devient complètement homomorphe [FV12].

2.7 BGV

À l'instar du schéma BFV, le schéma Brakerski-Gentry-Vaikuntanathan (BGV) [BGV12] appartient également à la famille des systèmes entièrement homomorphes, dont la sécurité repose sur la difficulté du problème RLWE. Cependant, des différences notables existent entre les schémas BGV et BFV.

L'une des principales distinctions entre BGV et BFV réside dans l'utilisation de plusieurs modules $q_1, q_2, \ldots, q_L$, où $q_i < q_{i+1}$. L'entier L représente la profondeur du circuit, c'est-à-dire le nombre maximal de multiplications pouvant être effectuées sur les textes chiffrés.

De plus, dans le schéma BGV, comme nous le verrons, le terme d'erreur est multiplié par un module p, ce qui permet de gérer le bruit lors du déchiffrement par l'application de réductions modulaires successives.

Le schéma BGV introduit également une technique, nécessitant des clés dites de permutation, pour compresser un chiffré après une multiplication, similaire à la linéarisation dans le schéma BFV mentionnée précédemment.

Enfin, une procédure appelée changement de module (modulus switching) permet de convertir un chiffré d'un module à un autre.

2.7.1 Paramètres du système BGV

Nous considérons les mêmes notations de la section 1.3.4 pour RLWE. Pour λ un paramètre de sécurité, p un nombre premier, et un polynôme cyclotomique f, nous considérons le quadruplet RLWE$(\lambda, p, \mathcal{X}, f)$ (avec donc $\mathcal{X}$ une distribution gaussienne dans $R = \mathbb{Z}[x]/(f(x))$ de moyenne μ et d'écart type δ). L'espace des textes en clair est R_p.
On considère $p_1, p_2, \ldots, p_L$ des nombres premiers et on définit pour tout $i \in \{1, \ldots L\}$, le module $q_i = \prod_{k=1}^{i} p_k$.

Pour chaque $n \in \mathbb{N}$, on définit l'anneau $R_n = \mathbb{Z}_n[x]/(f(x))$. Dans le schéma BGV original, les clés publiques ainsi que les clés de permutation sont représentées sous forme de matrices, mais peuvent également être exprimées comme des vecteurs sur un anneau polynomial. Nous présentons le schéma BGV dans sa forme vectorielle.

Algorithm 13 : génération des clés BGV

Entrée : $param = (q_1, q_1, \cdots, q_L, \mathcal{X}, p,)$
Sortie : un couple (p_k, s_k) et une liste de L quadruplet de clés de permutation.

1: $s = s_k \xleftarrow{\$} R_L$
2: $a \xleftarrow{\$} R_L$
3: $e \xleftarrow{\$} \mathcal{X}$
4: calculer $b = [-(a.s + p.e)]_L$
5: définir $p_k = (a, b)$
6: calculer les clefs de permutation (a_i, b_i, t_i, i), $i \in \{1, \ldots, L\}$ avec :
7: $a_i \xleftarrow{\$} R_{q_i}$, $e_i \xleftarrow{\$} \mathcal{X}$, $t_i \xleftarrow{\$} \mathbb{N}$ et $b_i = [-(a_i s + pe_i - t_i s^2)]_{t_i.q_i}$
8: $b_i = [-(a_i s + pe_i - t_i s^2)]_{t_i.q_i}$
9: retourner (p_k, s_k) et la liste des (a_i, b_i, t_i, i)

2.7.2 Chiffrement

Algorithm 14 : Chiffrement BGV

Entrée : un message $m \in R_p$, la clé publique $pk = (a, b)$
Sortie : un chiffré $c = (c_0, c_1) \in R_L^2$

1: choisir $v \xleftarrow{\$} \{-1, 0, 1\}^l$
2: choisir $e_1, e_2 \xleftarrow{\$} \mathcal{X}$
3: calculer $c_0 = [b.v + p.e_0 + m]_L$ et $c_1 = [a.v + p.e_1]_L$
4: définir et retourner $c = (c_0, c_1, L)$

2.7.3 déchiffrement

Algorithm 15 : déchiffrement BGV

Entrée : un chiffré $c = (c_0, c_1, i)$, la clé privée $s_k = s$
Sortie : un clair $m \in R_p$
1: calculer $m = [[c_0 + c_1.s]_{q_i}]_p$

Démonstration. (Validité du déchiffrement)

La preuve découle de l'observation suivante :
Soient p, q deux nombres premiers tels que $p < q$, $m \in \mathbb{Z}_q$ et $e \in \mathbb{N}$.
On a alors :

$$0 < m + e.p < q \Longrightarrow [[m + e.p]_q]_p = m$$

dans le déchiffrement, on a :

$$\begin{aligned}
[[c_0 + c_1.s]_{q_i}]_p &= [[[b.v + p.e_0 + m + (a.v + p.e_1)s]_L]_{q_i}]_p \\
&= [[[-(a.s + p.e).v + p.e_0 + m + (a.v + p.e_1)s]_L]_{q_i}]_p \\
&= [[[-a.s.v - p.e.v + p.e_0 + m + a.v.s + p.e_1.s]_L]_{q_i}]_p \\
&= [[[m + p(-e.v + e_0 + e_1.s)]_L]_{q_i}]_p \\
&= m
\end{aligned}$$

□

Regardons maintenant les propriétés homomorphiques de la fonction de chiffrement.

2.7.4 Addition, Multiplication et le Modulus switching

Soient $c = (c_0, c_1)$ et $c' = (c'_0, c'_1)$ deux chiffrés correspondant aux messages m et m', respectivement, et à un même module q_i, où $i \in \{1, \ldots, L\}$.
On a :

$$c + c' = \left(c_0 + c'_0, c_1 + c'_1\right)$$

qui correspond à un chiffré de $m + m'$.

Pour la multiplication, en adoptant la notation polynomiale comme dans BFV, on a :

$$c(x) \times c'(x) = (c_0 + c_1(x)) \times (c'_0 + c'_1(x)) = c_0c'_0 + (c_0c'_1 + c_1c'_0)x + c_1c'_1x^2$$

La conversion du chiffré résultant de $c(x) \times c'(x)$ en un chiffré de degré 1 nécessite les étapes suivantes :

▷ on pose : $d_0 = c_0c'_0,\ d_1 = c_0c'_1 + c_1c'_0$ et $d_2 = c_1c'_1$
▷ on définit le couple (c^*_0, c^*_1) à l'aide de la clé de relinéarisation (a_i, b_i, t_i, i) par : $c^*_0 = t_id_0 + b_id_2 \mod t_iq_i$ *et* $c^*_1 = t_id_1 + a_id_2 \mod t_iq_i$

Le couple (c^*_0, c^*_1) est un élément de $R^2_{t_iq_i}$. Ainsi, nous allons utiliser l'opération de changement de module (modulus switching) décrite ci-dessous pour le transformer en un élément de $R^2_{q_i}$, puis en un élément de $R^2_{q_{i-1}}$.

Algorithm 16 : modulus switching BGV

Entrée : un chiffré $c = (c_0, c_1) \in R^2_{q_i}$ avec le module q_i, un module q_j
Sortie : un chiffré $c' = (c'_0, c'_1) \in R^2_{q_j}$ avec le module q_j,

1: calculer $r := \frac{q_j}{q_i} = \mod q_j$
2: définir $c'_0 = rc_0 \mod q_j$ et $c'_1 = rc_1 \mod q_j$
3: retourner $c' = (c'_0, c'_1, j)$

Notons que le changement de module (modulus switching) transforme un texte chiffré d'un module en un autre, tout en permettant également de réduire le bruit de multiplication.

Continuons à présent avec la multiplication.
En utilisant la procédure de changement de module que nous venons de décrire, pour le couple (c^*_0, c^*_1), on définit :

$$\tilde{c} = (rc^*_0, rc^*_1)\ ,\ avec\ r = \frac{q_i}{q_it_i} \mod q_i = t_i^{-1} \mod q_i$$

Proposition 2.7.1. *Le couple $\tilde{c}$ est un chiffré de $m \times m'$ sous le module q_i.*

Démonstration.

On a modulo q_i

$$\begin{aligned} r \times c^*_0 + r \times c^*_1 s &= r\left(t_id_0 + b_id_2\right) + r\left(t_id_1 + a_id_2\right)s \\ &= d_0 + rb_id_2 + d_1s + ra_id_2s \\ &= d_0 + d_1s + rd_2\left(b_i + a_is\right) \\ &= d_0 + d_1s + rd_2\left(-pe_i + t_is^2\right)\ car\ b_i = -(a_is + pe_i - t_is^2) \\ &= d_0 + d_1s + d_2s^2 - rd_2pe_i \end{aligned}$$

En outre on peut écrire :

$$d_0 + d_1s + d_2s^2 = m \times m' + p \times erreur$$

d'où $r \times c_0^* + r \times c_1^* s = m \times m' + p \times erreur - rd_2 pe_i$.

si $p \times erreur - rd_2 pe_i < q_i$ alors

$$[[r \times c_0^* + r \times c_1^* s]_{q_i}]_p = [[m \times m' + p\,(erreur - rd_2 e_i)]_{q_i}]_p = m \times m'$$

□

2.8 CKKS

Le schéma CKKS (Cheon Kim Kim Song) [Che+17], également connu sous le nom de Chiffrement Homomorphe pour l'Arithmétique des Nombres Approximatifs (HEAAN), fait partie des schémas basés sur le problème RLWE. Développé pour permettre le calcul homomorphe sur des nombres réels, il permet des opérations d'addition et de multiplication sur des messages chiffrés. Le schéma CKKS introduit également une nouvelle procédure appelée **RESCALING** pour réduire le bruit en transformant un texte chiffré en un autre texte chiffré avec un module différent, comme dans la procédure de (modulus switching) permutation de module du schéma BGV. Une autre particularité de CKKS est l'introduction d'un système de codage pour les nombres complexes, que nous détaillerons dans la section suivante.

Soit M une puissance de deux et $N = \phi(M)$. On définit les structures mathématiques suivantes :

- ▷ $\mathbb{Z}_M^*$,
- ▷ $\Phi_M(X)$ le $M-$ ième polynôme cyclotomique ; $\Phi_M(X) = X^N + 1$
- ▷ $\mathcal{S} = \mathbb{R}[X]/\Phi_M(X)$,
- ▷ $\mathcal{R} = \mathbb{Z}[X]/\Phi_M(X)$,
- ▷ $\zeta = \exp(-2\pi i/M)$ une racine $M-$ième de l'unité,
- ▷ $\sigma : \mathcal{S} \longrightarrow \mathbb{C}^N$ tel que $\sigma(p) = \left(p(\zeta^j)\right)_{j \in \mathbb{Z}_M^*}$ [2].
- ▷ $\mathbb{H} = \left\{(z_j)_{j \in \mathbb{Z}_M^*} : z_{-j} = \overline{z_j}, \forall j \in \mathbb{Z}_M^*\right\} \subset \mathbb{C}^N$,
- ▷ soit T un sous-groupe [3] de $\in \mathbb{Z}_M^*$ tel que : $\mathbb{Z}_M^*/T = \{\pm 1\}$,
- ▷ $\pi : \mathbb{H} \longrightarrow \mathbb{C}^{\frac{N}{2}}$ la projection canonique définit par :

$$z = (z_j)_{j \in \mathbb{Z}_M^*} \longmapsto \pi(z) = (z_j)_{j \in T}$$

2. Les composantes des vecteurs dans $\mathbb{C}^N$ seront indexées par les éléments de $\mathbb{Z}_M^*$

3. Un tel sous-groupe T existe puisque $\mathbb{Z}_M^*$ est abélien et d'ordre pair (Réciproque du théorème de Lagrange pour les groupes abéliens).

Notons que :

$$\pi^{-1}(z)[j] = z_j \ \textit{si} \ \ j \in T \ \ \textit{et} \ \ \pi^{-1}(z)[j] = \overline{z_{-j}} \ \textit{sinon.}^{4}$$

L'application π^{-1} est bien définie. En effet, appelons ν la surjection canonique de $\mathbb{Z}_M^*$ vers le groupe quotient $\mathbb{Z}_M^*/T = \{\pm 1\}$.

Pour $j \in \mathbb{Z}_M^*$, nous avons :

$$j \notin T \Longrightarrow \nu(j) = -1 \Longrightarrow \nu(-j) = 1.$$

Autrement dit, si $j \notin T$, alors $-j \in T$.

Considérons l'exemple suivant :

Exemples 2.8.1. *Soit $M = 16 = 2^4$, alors, $N = \phi(16) = 8$.*
L'ensemble $\mathbb{Z}_M^*$ est donné par $\{1,3,5,7,9,11,13,15\}$, et un sous groupe T approprié est $\{1,3,9,11\}$.
Soit $z = (z_1, z_3, z_5, z_7, z_9, z_{11}, z_{13}, z_{15}) \in \mathbb{C}^N$.
Alors, nous avons :

$$\pi(z) = (z_1, z_3, z_9, z_{11})\,.$$

Réciproquement, pour $z = (z_1, z_3, z_9, z_{11})$, nous avons :

$$\pi^{-1}(z) = (z_1, z_3, \overline{z_{-5}}, \overline{z_{-7}}, z_9, z_{11}, \overline{z_{-13}}, \overline{z_{-15}})\,,$$

en outre :

$$\begin{cases} -5 & \equiv & 11, \\ -7 & \equiv & 9, \\ -13 & \equiv & 3, \\ -15 & \equiv & 1. \end{cases}$$

Ainsi, nous obtenons :

$$\pi^{-1}(z) = (z_1, z_3, \overline{z_{11}}, \overline{z_9}, z_9, z_{11}, \overline{z_3}, \overline{z_1})\,.$$

Remarque 2.8.1. *(inversion de σ)*

Pour $z \in \mathbb{C}^N$, calculer l'image réciproque de $z = \left(z_j\right)_{j \in \mathbb{Z}_M^*}$ par σ revient à trouver un polynôme $m(X) \in \mathcal{S}$ tel que $(m(\zeta^j)_{j \in \mathbb{Z}_M^*}) = z$.
Posons

$$m(X) = \sum_{k=0}^{N-1} \alpha_k X^k$$

4. $\pi^{-1}(z)[j]$ désigne la composante d'indice j de $\pi^{-1}(z)$

alors, un tel polynôme doit vérifier pour tout $j \in \mathbb{Z}_M^*$:

$$m(\zeta^j) = \sum_{k=0}^{N-1} \alpha_k \zeta^{kj}$$

on doit donc trouver les α_k qui sont solutions du système :

$$\sum_{k=0}^{N-1} \alpha_k \zeta^{kj} = z_j \text{ pour tout } j \in \mathbb{Z}_M^*$$

Notons $\mathbb{Z}_M^* = (j_1, j_2, \cdots, j_N)$ On obtient le système matricielle suivant :

$$\begin{pmatrix} 1 & \zeta^{j_1} & \zeta^{2j_1} & \dots & \zeta^{(N-1)j_1} \\ 1 & \zeta^{j_2} & \zeta^{2j_2} & \dots & \zeta^{(N-1)j_2} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \zeta^{j_N} & \zeta^{2j_N} & \dots & \zeta^{(N-1)j_N} \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{N-1} \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{pmatrix}$$

On obtient donc un système linéaire de N équations à N inconnus avec une matrice de Vandermonde. La résolution du système permet donc de trouver le polynôme $m(X) = \sigma^{-1}(z) \in \mathcal{S}$.

2.8.1 Codage sur les complexes

Définissons à présent les fonctions d'encodage et de décodage. La fonction d'encodage prend un facteur d'échelle Δ et sera noté par Enc_Δ définie comme suit :

$$\begin{array}{rccl} Enc_\Delta : & \mathbb{C}^{\frac{N}{2}} & \longrightarrow & \mathcal{R} \\ & \mathrm{z} & \longmapsto & \sigma^{-1}(\lfloor \Delta.\pi^{-1}(z) \rceil_{\sigma(\mathcal{R})}) \end{array}$$

avec $\lfloor \cdot \rceil_{\sigma(\mathcal{R})}$ une fonction d'arrondi qui permet de convertir un vecteur $v \in \mathcal{C}^N$ en un vecteur noté $\lfloor v \rceil_{\sigma(\mathcal{R})} \in \sigma(\mathcal{R})$, avec une certaine erreur d'arrondi. Cette conversion est nécessaire car, pour un vecteur $z \in \mathbb{C}^{\frac{N}{2}}$, l'élément $\pi^{-1}(z)$ n'appartient pas nécessairement à $\sigma(\mathcal{R})$. Ce processus est appelé **discrétisation** [Che+17], et diverses méthodes pour le réaliser ont été proposées par Lyubashevsky et al. [LPR13], que nous ne traiterons pas dans ce document.

La fonction de décodage est noté par Dec_Δ et est définie par :

$$\begin{array}{rccl} Dec_\Delta : & \mathcal{R} & \longrightarrow & \mathbb{C}^{\frac{N}{2}} \\ & m & \longmapsto & \pi \circ \sigma(\Delta^{-1} m) \end{array}$$

2.8.2 Les paramètres

Algorithm 17 : génération des paramètres CKKS

Entrée : λ paramètre de sécurité, L le nombre de niveau
Sortie : les paramètres

1: on fixe une base p et un module q_0
2: on définit $q_i = p^i q_0$ pour tout $0 < i \leq L$
3: on choisit une puissance de 2, $M = M(\lambda, q_L)$
3: définir un réel $\sigma = \sigma(\lambda, q_L)$
3: définir un entier $h = h(\lambda, q_L)$
4: retourner $param = (p, L, M, q_0, \sigma, h)$

On note aussi par :

- $\triangleright$ $\mathcal{H}(h)$, avec $h \in \mathbb{N}$, est une distribution dans $\mathcal{R}$ avec des coefficients dans $\{0, \pm 1\}$ et de poids h, c'est-à-dire que le nombre de coefficients non nuls est h.
- $\triangleright$ $\mathcal{X}$ la distribution gaussienne dans $\mathcal{R}$ de variance σ^2.
- $\triangleright$ $\mathcal{R}_{q_L} = \mathbb{Z}_{q_L}[X]/(X^N + 1)$
- $\triangleright$ $\mathcal{ZO}(\rho)$, pour tout $0 < \rho < 1$, est une distribution dans $\mathcal{R}$ avec des coefficients dans $\{0, \pm 1\}$, où les probabilités sont de $\frac{\rho}{2}$ pour -1 et 1, et de $1 - \rho$ pour 0.

2.8.3 Génération des clés

Algorithm 18 : génération des clés CKKS

Entrée : les paramètres $param = (p, L, M, q_0, \sigma, h)$
Sortie : un couple (p_k, s_k)

1: $s \xleftarrow{\$} \mathcal{H}(h)$
2: $a \xleftarrow{\$} \mathcal{R}_{q_L}$
3: $e \xleftarrow{\$} \mathcal{X}$
4: calculer $b = [-a.s + e]_{q_L}$
5: définir $p_k = (a, b)$
6: définir $s_k = s$
7: retourner (p_k, s_k)

2.8.4 Chiffrement

Algorithm 19 : Chiffrement CKKS

Entrée : un message $m \in \mathcal{M} = \mathcal{R}_{q_L}$

Sortie : Un chiffré $c \in \mathcal{C} = \left(\mathcal{R}_{q_L}\right)^2$

1: choisir $v \xleftarrow{\$} \mathcal{Z}O(0.5)$
2: choisir $e_0, e_1 \xleftarrow{\$} \mathcal{X}$
3: calculer $c_0 = [b.v + e_0 + m]_L$ $c_1 = [a.v + e_1]_L$
4: définir et $c = (c_0, c_1)$
5: retourner c

2.8.5 déchiffrement

Algorithm 20 : déchiffrement CKKS

Entrée : un niveau l un chiffré $c = (c_0, c_1) \in \mathcal{R}_{q_l}^2$, une clé privée s

Sortie : un message $m \in \mathcal{R}_{q_l}$

1: retourner $m = \left[c_0 + c_1.s\right]_{q_l}$

Proposition 2.8.1. *(Propriétés homomorphiques)*
Comme les schémas précédents, ***CKKS*** *permet d'effectuer des additions et des multiplications entre textes chiffrés, avec une relinéarisation nécessaire pour les opérations de multiplication.*

2.8.6 Le processus de RESCALING

Le RESCALING de CKKS correspond au changement de module dans BGV. Il est décrit comme suit :

Soit $l \in \{0, \ldots, L\}$ et $c \in \mathcal{R}_{q_l}^2$ un texte chiffré avec le module q_l. Le processus RESCALING consiste à transformer le texte chiffré c en un texte chiffré $c' = \left\lfloor \frac{q_{l'}}{q_l} c \right\rceil \pmod{q_{l'}} \in \mathcal{R}_{q_{l'}}^2$ avec $l' < l$.

Chapitre 3

Protocole SMC, Contributions, Résultats

L'idéal pour un protocole de calcul multipartite serait d'avoir un schéma de chiffrement totalement homomorphe. Compte tenu des six systèmes dont nous disposons, nous avons des schémas presque homomorphes tels que BFV, BGV et CKKS, qui peuvent être rendus complètement homomorphes à l'aide de la technique de bootstrapping de Gentry. Cependant, cette technique est très coûteuse et difficile à mettre en œuvre, ce qui pourrait conduire à des protocoles très inefficaces.

D'autre part, les systèmes partiellement homomorphes comme RSA, ElGamal et Paillier n'autorisent pas toutes les opérations de manière canonique. Parmi ces systèmes, Paillier se distingue en offrant le plus de possibilités. Comme nous l'avons vu dans le chapitre 2, il permet d'effectuer des opérations d'addition, de soustraction et de multiplication par une constante sur des données chiffrées. De plus, Paillier inclut un protocole interactif pour le produit entre deux valeurs chiffrées. Ces caractéristiques font de Paillier un choix privilégié pour les protocoles de calcul multipartites que nous proposons pour le projet BI4people.

Nous commençons par présenter le protocole de déchiffrement partagé de Paillier. Ensuite, nous proposons des protocoles génériques utilisant Paillier pour calculer la moyenne (et donc la somme) ainsi que la variance de données chiffrées. Nous présentons également un protocole pour le produit de deux valeurs chiffrées avec Paillier, ainsi que deux protocoles pour la gestion des clés des parties, pouvant être utilisés dans divers protocoles de calcul multipartite.

3.1 Déchiffrement partagé à seuil

Dans le cadre de ce schéma de partage de déchiffrement, nous considérons un ensemble de n parties $\mathcal{P}_1, \mathcal{P}_2 \ldots, \mathcal{P}_n$ et une paire de clés cryptographiques : une clé publique et une clé privée. La clé privée est répartie entre les différentes parties. L'objectif est de permettre le déchiffrement d'un texte chiffré c de telle manière qu'une coalition d'au moins t parties, où $t \leq n$, soit nécessaire pour effectuer le déchiffrement.

Plus formellement, un schéma de déchiffrement à seuil est défini par quatre algorithmes :

1. **un algorithme de génération de clés**
   - **Entrée** : n nombre de parties, un seuil t, paramètres de sécurité.
   - **Sortie** : p_k : clé publique, $S_k = \left(s_{k_1}, s_{k_2}, \ldots, s_{k_n}\right)$: liste des clés privées, $V_k = \left(v_{k_1}, v_{k_2}, \ldots, v_{k_n}\right)$: liste des clés de vérification.
2. **un algorithme de chiffrement**
   - **Entrée** : p_k : clé publique, un message clair m.
   - **Sortie** : un texte chiffré c.
3. **un algorithme de décryptage partiel**
   - **Entrée** : un indice $i \in \{1, \ldots, n\}$, la clé privée s_{k_i}, un texte chiffré c.
   - **Sortie** : un déchiffrement partiel δ_i et une preuve pr_i de la validité du déchiffrement δ_i.
4. **un algorithme de combinaison**
   - **Entrée** : un ensemble $S = \{k_1, \ldots, k_t\}$ de t parties, $\Delta = \left((\delta_{k_1}, pr_{k_1}), (\delta_{k_2}, pr_{k_2}), \ldots, (\delta_{k_t}, pr_{k_t})\right)$ une liste de t déchiffrements partiels, $V_k = \left(v_{k_1}, v_{k_2}, \ldots, v_{k_t}\right)$ une liste de t clés de vérification.
   - **Sortie** : un message clair m ou un message d'échec.

3.2 Schéma de déchiffrement à seuil de Paillier

3.2.1 Préliminaires

Soit n le nombre de parties et t le seuil avec $t \leq n$.
On note s la clé secrète qui sera partagée par le protocole de Shamir 2.1.3.

On définit donc un polynôme f de degré t à coefficients dans $\mathbb{Z}$ tel que $f(0) = s$. Chaque partie $\mathcal{P}_i$ reçoit la part $(i, f(i))$ pour tout $i \in \{1, \ldots, n\}$. Posons $\Delta = n!$. Pour toute partie S de $\mathcal{P}(\{1, \ldots, n\})$ de cardinal $t+1$, on retrouve le secret s par la formule ci-dessus :

$$s = \sum_{i \in S} f(i)\beta_i \tag{1}$$

avec

$$\beta_i = \prod_{k \in S \setminus \{i\}} \frac{j}{j-i} = L_i(0) \text{ où } L_i(x) = \prod_{k \in S \setminus \{i\}} \frac{x-j}{i-j}$$

On définit, pour tout $i \in \{1, \ldots, n\}$ et pour tout $j \in S$, le coefficient $\lambda_i^S(j)$ par :

$$\lambda_i^S(j) = \Delta \prod_{k \in S \setminus \{j\}} \frac{i-k}{j-k}$$

Remarque 3.2.1. *Pour tout $i \in \{1, \ldots, n\}$ et pour tout $j \in S$, $\lambda_i^S(j) \in \mathbb{Z}$. En effet, les $j-k$ sont deux à deux distincts et inférieurs à n donc il sont des facteurs de Δ, il en résulte que*

$$\prod_{k \in S \setminus \{j\}} (j-k) \text{ divise } \Delta$$

Nous avons aussi la relation :

$$\Delta f(0) = \sum_{j \in S} \lambda_0^S(j) f(j)$$

Étant donné que les $f(j)$ sont des entiers, la multiplication par Δ permet d'exprimer Δs comme une somme de termes entiers. Cette propriété sera utile ultérieurement dans l'algorithme de combinaison.

3.2.2 Les algorithmes du schéma de déchiffrement à seuil de Paillier

On considère la version de Damgård et Jurik [DJ00] de Paillier. Les quatre algorithmes associés au schéma à seuil de Paillier sont les suivants :

Algorithm 21 : génération des clés

Entrée : $\lambda \geq 1$ paramètre de sécurité, $n, t \in \mathbb{N}$ avec $(t < n)$.
Sortie : un triplet (p_k, S_k, V_k) comme décrit plus haut.

1: choisir deux nombres premiers sûrs $p = 2p' + 1$ et $q = 2q' + 1$, de λ bits.
2: définir $N = p \times q$ et $g = 1 + N$
3: calculer $m = \frac{\phi(N)}{4} = p'q'$
4: choisir β aléatoirement dans $\mathbb{Z}_N^*$
5: poser $a_0 = \beta \times m$ et choisir aléatoirement t valeurs $a_1, a_2, \ldots, a_t$ dans $\{0, 1, \ldots, N \times m - 1\}$
6: définir $f(X) = \sum_{k=0}^{t} a_k X^k$
7: calculer $s_{k_i} = f(i) \mod Nm$ pour tout $i \in \{1, \ldots, n\}$
8: choisir ν un générateur du groupe des carrés dans $\mathbb{Z}_{N^2}^*$
9: calculer $v_{k_i} = \nu^{\Delta s_{k_i}} \mod N^2$ pour tout $i \in \{1, \ldots, n\}$
10: $p_k = (g, N)$
11: $S_k = \left(s_{k_1}, s_{k_2}, \ldots, s_{k_n}\right)$: la liste des clés secrètes de déchiffrement partiel.
12: $V_k = \left(v_{k_1}, v_{k_2}, \ldots, v_{k_n}\right)$: la liste des clés secrètes de vérification
13: retourner (p_k, S_k, V_k)

Algorithm 22 : chiffrement

Entrée : un message clair $M \in \mathbb{Z}_N$, la clé publique $p_k = (g, N)$
Sortie : un chiffré c

1: $x \xleftarrow{\$} \mathbb{Z}_N^*$
2: $c = g^M x^N \mod N^2 \equiv (1 + MN)x^N \mod N^2$

Dans le déchiffrement partiel, chaque partie calcule une preuve de déchiffrement correcte, garantissant qu'elle a respecté le protocole. Cette preuve consiste en un protocole non interactif entre la partie et l'entité chargée de combiner les parts de déchiffrement.

Algorithm 23 : déchiffrement partiel

Entrée : un indice $i \in \{1, \ldots, n\}$, la clé privé s_{k_i} et un chiffré c
Sortie : un couple $(\delta_i, \text{preuve}_i)$: déchiffrement partiel δ_i et d'une preuve.

1: calculer $\delta_i = c^{2\Delta s_{k_i}} \mod N^2$
2: génère une preuve $preuve_i$ d'égalité du logarithme discret de $\delta_i^2 \mod N^2$ et $v_{k_i} \mod N^2$ en bases $c^{4\Delta}$ et ν^{Δ} respectivement.
3: retourner $(\delta_i, preuve_i)$

Algorithm 24 : combinaison

Entrée : Une coalition S de $t+1$ parties et une liste de $t+1$ déchiffrements partiels.
Sortie : un message clair M ou un message d'échec
1: une vérification des parts de déchiffrement
2: **si** les vérifications sont valide : **alors**
3: retourner $M = L\left(\prod_{j \in S} \delta_j^{2\lambda_0^S(j)} \mod N^2\right)\left(4\Delta^2 m\beta\right)^{-1} \mod N$
4: **fin du si**
5: **Sinon** retourner un message d'échec

Démonstration. (validité algorithme de combinaison)
(voir annexe F) □

Remarque 3.2.2.
Le choix d'un élément aléatoire β dans $\mathbb{Z}_N^*$ dans le cadre du schéma de déchiffrement partagé a pour objectif de masquer le secret m. En effet, si l'on considère le secret $sk = m$, alors on obtient : $M = L\left(\prod_{j \in S} \delta_j^{2\lambda_0^S(j)} \mod N^2\right) = 4\Delta^2 Mm \mod N$. Ainsi, un attaquant qui accède aux déchiffrements partiels d'un message M obtient Mm. Il lui faut donc disposer des déchiffrements partiels de deux messages M_1 et M_2 pour obtenir $M_1 m$ et $M_2 m$. Si les messages M_1 et M_2 sont premiers entre eux, il peut retrouver m par un simple calcul de $pgcd$.

En choisissant $sk = m \times \beta$ avec β aléatoire dans $\mathbb{Z}_N^*$, le produit $m \times \beta$ sera également aléatoire, de sorte qu'un attaquant qui accède aux déchiffrements partiels n'aura que $m \times \beta$, qui est aléatoire dans $\mathbb{Z}_N^*$.

3.2.3 Preuve de déchiffrement : preuve non interactive d'égalité du logarithme discret

Pour chaque partie $\mathcal{P}_i$, la preuve de déchiffrement $preuve_i$ générée par $\mathcal{P}_i$ et la clé de validité v_{k_i} sont liées par la clé secrète sk_i. En effet, on a :

$$\delta_i^2 = c^{4\Delta s_{k_i}} \; et \; v_{k_i} = \nu^{\Delta s_{k_i}} \Longrightarrow \log_{c^{4\Delta}}(\delta_i^2) = \log_{\nu^{\Delta}}(v_{k_i}) = s_{k_i}$$

Posons

$$\tilde{x} = c^{4\Delta} \; et \; \tilde{y} = \nu^{\Delta}$$

La vérification de la preuve consiste donc à établir l'égalité des logarithmes discrets de $\delta_i^2 \mod N^2$ et $v_{ki} \mod N^2$ dans les bases $\tilde{x}$ et $\tilde{y}$, respectivement. Le choix de δ_i^2 plutôt que δ_i s'explique par le fait que nous opérons dans le groupe des carrés $\mathbb{Z}_{N^2}^*$. Les déchiffrements partiels étant des carrés, le vérificateur recevant δ_i ne peut pas vérifier cela de manière efficace ; ainsi, δ_i^2 est utilisé à la place de δ_i afin de garantir que le prouveur travaille dans le groupe des carrés.
Nous utiliserons le protocole non-interactif de Shoup présenté dans [Sho00], qui est une adaptation du protocole interactif de Chaum et Pedersen [CP92]. Le protocole peut être décrit comme suit :

On considère une fonction de hachage H de taille l_1. On note l_2 la taille de N, $\mathcal{P}$ le prouveur et $\mathcal{V}$ le vérificateur.

Algorithm 25 : Protocole de preuve de validité

1: Prouveur
2: $r \xleftarrow{\$} \left[0, 2^{l_2+2l_1}\right[$
3: $x' := \tilde{x}^r$ et $y' = \tilde{y}^r$
4: $h := H(\tilde{x}, \tilde{y}, \delta_i, v_{k_i}, x', y')$
5: $z := r + s_{k_i} \times h$
6: $\mathcal{P} \xrightarrow{(h,z)} \mathcal{V}$
7: Vérificateur
8: $X := \dfrac{\tilde{x}^z}{\delta_i^{2h}}$ et $Y := \dfrac{\tilde{y}^z}{v_{k_i}^h}$
9: Vérifier si $h \stackrel{?}{=} H(\tilde{x}, \tilde{y}, \delta_i, v_{k_i}, X, Y)$
10: Et vérifier si $y \stackrel{?}{<} 2^{2l_2+l_1}$
11: Retourner $[h == H(\tilde{x}, \tilde{y}, \delta_i, v_{k_i}, X, Y) \; et \; (z < 2^{l_2+2l_1})]$

Exactitude du schéma.
Si le protocole est respecté, on aura, avec une probabilité écrasante :

$$z < 2^{l_2+2l_1} \; \textit{(voir annexe G)}$$

En outre :

$$X = \frac{\tilde{x}^z}{\delta_i^{2h}} = \frac{\tilde{x}^{r+h\times s_{k_i}}}{\tilde{x}^{h\times s_{k_i}}} = \frac{\tilde{x}^r \tilde{x}^{h\times s_{k_i}}}{\tilde{x}^{s_{k_i} h}} = \tilde{x}^r = x'$$

et

$$Y = \frac{\tilde{y}^z}{v_{k_i}^h} = \frac{\tilde{y}^{r+s_{k_i}\times h}}{v_k^{\Delta s_{k_i}\times h}} = \frac{\tilde{y}^r \tilde{y}^{s_{k_i}\times h}}{\tilde{y}^{s_{k_i}\times h}} = \tilde{y}^r = y'$$

□

Remarque 3.2.3. *Considérons un nombre n de parties tel que $n > \max(p, q)$, où p et q sont des facteurs de N. D'après le théorème 1.2.1, l'ordre du groupe des carrés est $N \times \frac{\phi(N)}{4} = p'q'pq$, dans ce cas, $N \times \phi(N)$ divise n! et $v_{k_i} = 1$ et $c^{2\Delta s_{k_i}} = \delta_i = 1$ pour tout i. Il est donc nécessaire que le nombre de parties soit inférieur aux diviseurs premiers de N.*

3.3 Schéma de déchiffrement à seuil basé sur RLWE

Dans cette section, nous présentons un schéma de déchiffrement à seuil reposant sur le problème **RLWE**. Pour un ensemble de n parties, nous commençons par introduire un schéma initial avec une structure d'accès n-sur-n. Nous détaillons ensuite une procédure permettant de modifier ce schéma afin d'obtenir une structure d'accès t-sur-n, pour tout $t < n$, en utilisant le partage de secret de Shamir.

3.3.1 Préliminaires

Dans les schémas de chiffrement entièrement homomorphe (FHE) présentés dans le chapitre 2, le déchiffrement d'un texte chiffré (c_0, c_1) nécessite le calcul de $\langle (c_0, c_1), (1, s) \rangle = c_0 + c_1 s$, où s représente le secret. Par conséquent, un partage additif de s permet de partager le déchiffrement.
Soit :

- ▷ $K = \mathbb{F}_q$ un corps
- ▷ $\mathcal{R} = K[x]/(x^l + 1)$ avec l une puissance de 2
- ▷ $\mathcal{X}$ une distribution d'erreur dans $\mathcal{R}$
- ▷ $n \in \mathbb{N}$, le nombre de parties
- ▷ $\mathcal{P}_1, \ldots, \mathcal{P}_n$ les différentes parties

On considère le quadruplet **RLWE** $(\lambda, q, \mathcal{X}, x^l + 1)$, nous définissons $param = (\lambda, q, \mathcal{X}, n)$.

Algorithm 26 : Génération des clés du déchiffrement partagé de **RLWE**

Entrée : $param$

Sortie : une clé publique p_k et une liste de clés de déchiffrement $((p_{0,1}; s_1), (p_{0,2}; s_2), \cdots, (p_{0,N}; s_N))$

1: $p_1 \xleftarrow{\$} \mathcal{R}$
2: **pour** chaque $\mathcal{P}_i,\ i \in \{1, \ldots, n\}$: **faire**
3: $\quad s_i \xleftarrow{\$} \mathcal{R}$
4: $\quad e_{0,i} \xleftarrow{\$} \mathcal{X}$
5: $\quad$ calcule $p_{0,i} = -s_i p_1 + e_{0,i}$
6: **fin du pour**
7: calculer $p_0 = \sum p_{0,i}$
8: $p_k = (p_0, p_1)$: la clé publique
9: $((p_{0,1}; s_1), (p_{0,2}; s_2), \ldots, (p_{0,n}; s_n))$: la liste des clés privées

Algorithm 27 : Chiffrement

Entrée : $param, p_k = (p_0, p_1)$ et un message m

Sortie : un couple (c_0, c_1)

1: $u \xleftarrow{\$} \mathcal{R}$
2: $e_0, e_1 \xleftarrow{\$} \mathcal{X}$
3: $c_0 = m + up_0 + e_0$ *et* $c_1 = up_1 + e_1$
4: retourner $c = (c_0, c_1)$

Algorithm 28 : déchiffrement partiel

Entrée : $param, c = (c_0, c_1), i \in \{1, \ldots, n\}$ et une clé de déchiffrement s_i

Sortie : $h_i \in \mathcal{R}$

1: $e_i \xleftarrow{\$} \mathcal{X}$
2: calculer $h_i = c_1 s_i + e_i$
3: retourner h_i

Algorithm 29 : Combinaison

Entrée : les parts de déchiffrement h_i

Sortie : un clair m

1: calculer $m' = c_0 + \sum_{i=1}^{n} h_i$
2: retourner m'

Démonstration. (validité du déchiffrement)
On a :

$$\begin{aligned} m' &= c_0 + \sum h_i \\ &= m + up_0 + e_0 + \sum(c_1 s_i + e_i) \\ &= m + u\sum(-s_i p_1 + e_{0,i}) + e_0 + \sum(up_1 + e_1)s_i + \sum e_i \\ &= m - up_1 \sum s_i + u \sum e_{0,i} + e_0 + up_1 \sum s_i + e_1 s + \sum e_i \\ &= m + \sum(ue_{0,i} + e_i) + e_0 + e_1 s \end{aligned}$$

□

L'algorithme de déchiffrement retourne une valeur approchée du message ; par conséquent, l'utilisation de ce schéma requiert l'application de techniques d'encodage des messages en clair ou de techniques de mise à l'échelle, comme celles employées dans le schéma **BFV**. Nous n'aborderons pas ici le développement de ces méthodes.
Examinons à présent comment passer d'un schéma avec une structure d'accès n-sur-n à un schéma avec une structure d'accès t-sur-n.

L'idée est que chaque partie partage sa clé de telle sorte qu'un seuil d'au moins t parties soit nécessaire pour reconstituer la clé. Le partage de la clé sera réalisé à l'aide du protocole de Shamir. Notons que l'on pourrait partager la liste des coefficients des polynômes, mais cela conduirait à un protocole coûteux, ce qui nous amène à supposer que les clés secrètes sont limitées à des polynômes constants. En conséquence, nous supposons que l'espace des clés est le corps $\mathbb{K}$.

3.3.2 Protocole de Partage et de Récupération des Clés

Considérons les notations de la partie précédente avec les n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ détenant chacune sa clé s_i et un coefficient public $\alpha_i \in \mathbb{K}$. La clé secrète globale est donnée par

$$s = \sum_{i=1}^{n} s_i.$$

Chaque partie $\mathcal{P}_i$ choisit un polynôme $S_i(x)$ de degré $t-1$ dont le terme constant est s_i et envoie $S_i(\alpha_j)$ à chaque partie $\mathcal{P}_j$ pour tout $j \in \{1, \ldots, n\} \setminus \{i\}$.

Pour toute coalition $\mathcal{E}$ d'au moins t parties, appelons λ_j les coefficients de Lagrange dans la reconstitution du secret s_i.

On a donc :

$$\forall i \in \{1,\dots,n\}\,, \quad s_i = \sum_{j\in E} S_i(\alpha_j)\lambda_j \Longrightarrow \sum_{i=1}^{n} s_i = \sum_{i=1}^{n}\sum_{j\in E} S_i(\alpha_j)\lambda_j = \sum_{j\in E}\lambda_j \sum_{i=1}^{n} S_i(\alpha_j).$$

Comme les λ_j dépendent des α_i qui sont publics, on définit

$$\tilde{s}_j = \sum_{i=1}^{n} S_i(\alpha_j),$$

alors on a :

$$\sum_{i=1}^{n} s_i = \sum_{j\in E} \lambda_j \tilde{s}_j.$$

De cette manière, le secret **s** peut être calculé avec toute coalition de t parties parmi les n.

Remarque 3.3.1. *Avec ce protocole, toute coalition $\mathcal{E}$ d'au moins t parties peut déchiffrer le message. Il suffit pour cela que les parties $\mathcal{P}_j$, $j \in \mathcal{E}$, calculent les $\tilde{s}_j$ et utilisent $s'_j = \lambda_j \tilde{s}_j$ comme clé secrète dans l'algorithme de déchiffrement partiel.*

Remarque 3.3.2. *Notons que pour mettre en œuvre ce schéma, il a été nécessaire de restreindre l'espace des clés au corps K. La contrainte est que, dans le partage de secret de Shamir, le calcul des coefficients de Lagrange nécessite que les $\alpha_i - \alpha_j$ soient inversibles pour tout $i \neq j$. Ainsi, il est possible d'étendre l'espace des clés secrètes à tout nouvel anneau $\mathcal{R}$, sous réserve de choisir les α_i de sorte que $\alpha_i - \alpha_j \in \mathcal{R}^\times$ pour tout $i \neq j$.*

Pour les anneaux $\mathcal{R} = \mathbb{Z}_q[x]/f(x)$, on peut considérer la factorisation de $q = p_1 \cdot p_2 \cdots p_k$ avec les p_i premiers. Si on définit $\tilde{p} = \min\{p_i \mid i = 1,\dots,k\}$, alors on peut choisir les α_i dans $\mathbb{Z}^*_{\tilde{p}}$. Ce choix des α_i dans $\mathbb{Z}^*_{\tilde{p}}$ tend à limiter le nombre de parties à $\tilde{p} - 1$,

Toutefois, ce n'est généralement pas un problème, car les facteurs de q sont déjà régis par les contraintes du schéma de chiffrement et sont typiquement supérieurs à 2^{11} dans le cadre des chiffrement entièrement homomorphes [MBH23].

3.4 Protocole MPC sur Paillier

3.4.1 Protocole de multiplication de Paillier

Nous adaptons un protocole de Cramer et al. [CDN01] pour réaliser le produit de deux chiffrés de Paillier.

Supposons qu'une entité nommée Alice et un serveur collaborent pour calculer le produit de deux chiffrement X et Y, correspondant respectivement aux messages clairs x et y. On suppose que seul le serveur détient la clé secrète. Alice cherche à garantir que le serveur ne puisse pas accéder au message en clair résultant du chiffrement du produit $x \times y$.
Nous notons par E_p et D_s les fonctions de chiffrement et de déchiffrement, respectivement avec un module N. Alice est désignée par $\mathcal{A}$ et le serveur par $\mathcal{S}$. Le protocole se déroule de la manière suivante (on note par $\star$ le produit entre deux chiffrés) :

Algorithm 30 : Protocole interactif pour le calcul du produit de deux chiffrés.

Entrée : X et Y les chiffrés de x et y respectivement
Sortie : $Z = X \star Y$ un chiffré de xy

1: $\mathcal{A}$
   1. $r, s \xleftarrow{\$} \mathbb{Z}_N$
   2. $R = E_p(r)$
   3. $S = E_p(s)$
   4. $X \star R = E_p(r + x)$
   5. $Y \star S = E_p(y + s)$
   6. $\underrightarrow{\mathcal{A} \text{ envoie la paire } (X \star R, Y \star S) \text{ à } \mathcal{S}}$

2: $\mathcal{S}$
   1. $r + x = D_s(X \star R)$
   2. $y + s = D_s(Y \star S)$
   3. $P = E_p\left[(r + x) \times (y + s)\right]$
   4. $\underrightarrow{\text{envoie } P \text{ à } \mathcal{A}}$

3: $\mathcal{A}$
   1. $A = E_p(-rs)$
   2. $B = Y^{-r}$
   3. $C = X^{-s}$
   4. $Z = P \star A \star B \star C = E_p(xy)$

Démonstration. (de la validité du protocole) Nous avons :

$$\begin{aligned} Z &= \mathrm{P} \star A \star B \star C \\ &= \mathrm{E}_p[(ry + rs + xy + xs) - rs - ry - xs] \\ &= \mathrm{E}_p(xy) \end{aligned}$$

□

Remarque 3.4.1. *Dans ce protocole, le serveur n'a jamais accès aux clairs x et y, mais uniquement aux éléments aléatoires $x + r$ et $y + s$, et il n'apprend donc rien des clairs x et y.*

3.4.2 Protocoles de gestion de clés : première proposition

Les protocoles que nous proposons a pour objectif d'assurer une gestion efficace des clés dans le cadre d'un calcul multipartite sécurisé avec le chiffrement de Paillier. Étant donné que des parties peuvent se joindre au protocole à tout moment, il est crucial de disposer d'un système de gestion des clés approprié afin d'éviter la génération de nouvelles clés pour chaque nouvelle partie. Nous supposons que le protocole est conçu pour accueillir jusqu'à n parties, bien qu'au départ seules k parties soient actives. Une autorité de confiance (**AC**) est chargée de la génération des clés ; toutefois, après cette étape, elle ne peut plus participer aux opérations ni être sollicitée pour d'autres tâches. Un serveur honnête est responsable des calculs entre les parties et assure la liaison entre elles. Nous supposons également que les parties ne se connaissent pas entre elles, et qu'une partie détentrice d'une clé ne peut pas en demander davantage, afin de prévenir l'accumulation par une seule partie de suffisamment de clés pour déchiffrer les données cryptées. Le déchiffrement d'une clé nécessite la participation d'un seuil de t parties. Initialement, la **AC** génère une clé publique $pk = (g, N)$, k clés secrètes pour les k parties activement engagées dans le protocole, ainsi que $n - k$ clés supplémentaires, qui sont chiffrées et mises à la disposition du serveur.

Nous désignons par $\mathcal{P}_1, \ldots, \mathcal{P}_k$ les parties déjà présentes dans le protocole, par $\mathcal{S}$ le serveur, et par $sk_{k+1}, sk_{k+2}, \ldots, sk_n$ les clés attribuées aux éventuelles nouvelles parties. La fonction de chiffrement est notée Enc ; les autres notations sont celles de l'algorithme 21. Considérons $\mathcal{P}_{k+i}$, avec $i \in \{1, \ldots, n-k\}$, une nouvelle partie souhaitant rejoindre le protocole. Le protocole que nous proposons est le suivant :

Algorithm 31 : Protocole de partage sécurisé de clé avec Pailler

Entrée : une coalition $\mathcal{E}$ de t parties, une nouvelle partie $\mathcal{P}_{k+i}$, $i \in \{1, \ldots, n-k\}$

Sortie : une clé privée sk_{k+i}

1: $\mathcal{P}_{k+i} : \xrightarrow{\text{demande de clé}} \mathcal{S}$

2: $\mathcal{P}_{k+i} \xleftarrow{\text{clé publique } p_k} \mathcal{S}$

3: $\mathcal{P}_{k+i}$: choisit $r \xleftarrow{\$} \mathbb{Z}_N$

4: $\mathcal{P}_{k+i} \xrightarrow{Enc(r)} \mathcal{S}$

5: $\mathcal{S}$: calcule $Enc(r) \times Enc(sk_{k+i}) = Enc(sk_{k+i} + r)$

6: déchiffrement de $Enc(sk_{k+i} + r)$

7: $\mathcal{P}_{k+i} \xleftarrow{sk_{k+i}+r} \mathcal{S}$

8: $\mathcal{P}_{k+i}$: calcule : $sk_{k+i} + r - r = sk_{k+1}$

Remarque 3.4.2. *Jusqu'à présent, nous n'avons mentionné que la clé secrète de déchiffrement partiel sk_i et non celle de vérification $vk_i = v^{\Delta sk_i}$. En effet, une clé de vérification peut être retrouvée à partir de la clé secrète associée, puisque les paramètres Δ et v sont publics.*
*Ainsi, la **CA** génère les paires (sk_i, vk_i) seulement pour les k premières parties et partage les chiffrés des clés de déchiffrement des éventuelles nouvelles parties comme décrit ci-dessus.*

Remarque 3.4.3. *Le chiffrement des clés ne peut être réalisé directement en raison de la différence entre l'espace des messages en clair, $\mathbb{Z}_N$, et l'espace des clés secrètes $\mathbb{Z}_{Nm}$. Ainsi, lorsque la clé sk est supérieure à N, nous proposons une procédure pour le chiffrement des clés sk.*

Avant de présenter notre procédure, nous soulignons qu'une approche naïve consisterait à diviser sk en deux parties, ce qui nécessiterait deux opérations de chiffrement et serait coûteux en ressources, car chaque déchiffrement requiert la participation de t parties. En revanche, notre procédure nécessite une seule opération de chiffrement et permet à la nouvelle partie de récupérer efficacement la clé.

Chiffrement des clés

Supposons que nous disposions d'une clé sk telle que $sk > N$. L'entité de confiance souhaite fournir au serveur le chiffrement de cette clé afin qu'elle puisse être récupérée par une partie rejoignant le protocole.

Considérons un corps $\mathbb{F} = \mathbb{Z}_p$ tel que $p > N \times m$. L'autorité de confiance (**AC**) calcule un triplet $(a, q, r) \in \mathbb{F}^3$ tel que :

$$a \times sk = N \times q + r \; dans \; \mathbb{Z} \quad \text{avec} \quad r < N.$$

Ainsi, la **CA** fournit au serveur les valeurs a, q, et le chiffrement de r. Une fois r déchiffré, la partie pourra calculer sk en utilisant la formule suivante :

$$sk = a^{-1}(N \times q - r) \quad dans \quad \mathbb{F}.$$

Analysons maintenant l'algorithme destiné au calcul du triplet (a, q, r).

Algorithm 32 : Calcul du triplet a, q et r

Entrée : sk, N,
Sortie : a, q, r

1: $(a, q) \xleftarrow{\$} \mathbb{F}^2$
2: calculer $r := a \times sk - q \times N$
3: **si** $r > N$ (dans $\mathbb{N}$) : **alors**
4: calculer $q' := \lfloor r/N \rfloor$ *dans* $\mathbb{N}$ *: le quotient de r par N*
5: calculer $r' := r\%N$ *dans* $\mathbb{N}$ *: le reste de r par N*
6: prendre $q := q + q'$ et $r = r'$
7: **fin du si**
8: retourner (a, q, r)

3.4.3 Protocoles de gestion de clés : deuxième proposition

Dans la première proposition, la gestion des clés nécessitait le chiffrement des clés au début par l'autorité de confiance et le déchiffrement impliqué toujours une coalition de partie. Nous proposons une nouvelle façon de gérer les clés sans recourir aux chiffrements. Les hypothèses pour la nouvelle proposition sont les suivantes :

- On a une autorité de confiance (**CA**) chargé de la génération des clés.
- On suppose, au début, qu'on a que k parties qu'on note $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$ et que le protocole peut contenir jusqu'à $n (n > k)$ parties
- On suppose que parmi les k parties, il y a un sous ensemble $E \in \mathcal{P}(\{1, \ldots, n\})$ de cardinale t qui on un statut particulier notamment celui de gérer les clés pour les nouvelles parties. Sans perte de généraliter supposons que $E = \{1, \ldots, t\}$

Avec ces hypothèse, la **CA** va générer n clés : $sk_1, sk_2, \ldots, sk_n$. Elle donne à chacun des k parties déjà existent une clé. Les $n - k$ autre clés restante seront mis à la disposition des parties $(\mathcal{P})_{j \in E}$ de la façon suivante :

- définir un corps $\mathbb{F}$ de cardinale assez grand pouvant contenir les clés (supérieur à $N \times m$ dans notre cas pour Paillier)
- Pour chaque clé sk parmi les $n - k$ restant, définir α et β dans $\mathbb{F}$ tels que $sk = \alpha \times \beta$

— choisir $(\alpha_1, \ldots, \alpha_t), (\beta_1, \ldots, \beta_t) \in \mathbb{F}^t$ tels que $\alpha = \sum_{i=1}^{t} \alpha_i,\ \beta = \sum_{i=1}^{t} \beta_i$

— Pour tout $j \in E$, la **CA** envoie le couple (α_j, β_j) à $\mathcal{P}_j$.

Ainsi, une nouvelle partie $\mathcal{P}_{k+i}$ pour $i \in \{1, \ldots, n-k\}$ reçoit sa clé avec le Protocole suivant de Beaver.

Algorithm 33 : partage de clé deuxième proposition

Entrée : $(\alpha_1, \ldots, \alpha_t), (\beta_1, \ldots, \beta_t)$
Sortie : $sk = \alpha \cdot \beta$

1: La nouvelle partie $\mathcal{P}_{k+i}$ génère aléatoirement $(a, b, c) \in \mathbb{F}^3$ tels que $a \cdot b = c$.
2: $\mathcal{P}_{k+i}$ génère $(a_1, \ldots, a_t), (b_1, \ldots, b_t), (c_1, \ldots, c_k) \in \mathbb{F}^t$ tels que : $\sum_{j=1}^{t} a_j = a, \quad \sum_{j=1}^{t} b_j = b$ et $\sum_{j=1}^{t} c_j = c$
3: $\mathcal{P}_{k+i}$ envoie le triplet (a_j, b_j, c_j) à la partie $\mathcal{P}_j$ pour tout $j \in E$
4: **pour tout** $j \in E, \mathcal{P}_j$: **faire**
5: calcule $[d]_j = [\alpha]_i - [a]_j$
6: calcule $[e]_j = [\beta]_j - [b]_j$
7: diffuse les $[d]_j$ et $[e]_j$ aux autres parties
8: **fin du pour**
9: **pour tout** $j \in E, \mathcal{P}_j$: **faire**
10: calculer $d = \sum_{j=1}^{t} [d]_j$, et $e = \sum_{j=1}^{t} [e]_j$
11: calculer $\sigma_j = \frac{de}{t} + d[b]_j + e[a]_j + [c]_j$
12: envoyer σ_j à $\mathcal{P}_{k+i}$
13: **fin du pour**
14: $\mathcal{P}_{k+i}$ calcule $\sum_{j=1}^{t} \sigma_j = \alpha \times \beta = sk$

Démonstration. (exactitude du protocole)
On a :

$$[d]_j = [\alpha]_j - [a]_j \Longrightarrow \sum_{j=1}^{t} [d]_j = \alpha - a$$

$[e]_j = [\beta]_{j=1} - [b]_j \Longrightarrow \sum_{j}^{t}[e]_j = \beta - b$ d'où :

$$\begin{aligned}\sum_{j=1}^{t}\sigma_j &= \sum_{j=1}^{t}\left(\frac{de}{t} + d[b]_j + e[a]_j + [c]_j\right)\\ &= de + db + ea + c\\ &= (\alpha - a)(\beta - b) + b(\alpha - a) + (\beta - b)a + c\\ &= \alpha\beta - ab + c\\ &= \alpha\beta\end{aligned}$$

□

Remarque 3.4.4. *Ce schéma permet le partage de la clé sans recourir au chiffrement de Paillier. Il est important de souligner que l'exactitude de ce protocole dépend du respect rigoureux des procédures par toutes les parties impliquées. Étant donné que ce respect est difficile à garantir en pratique, limiter la valeur de t (le nombre de parties) permet de réduire le risque de comportements malhonnêtes. En revanche, pour la confidentialité, un attaquant ayant le contrôle sur toutes les parties impliquées dans le protocole, à l'exception d'une seule, ne parviendra pas à obtenir d'information sur la clé.*

3.4.4 Protocoles : Calcul de la moyenne et de la variance sur des données privées avec Paillier

On considère N parties $\mathcal{P}_1, \ldots, \mathcal{P}_N$ et un serveur $\mathbb{S}$. Les parties disposent chacune de données privées x_i. Nous présentons ici deux protocoles simples et confidentiels pour calculer la moyenne et la variance de ces données en utilisant le chiffrement de Paillier.
Nous notons respectivement par Enc et Dec les fonctions de chiffrement et de déchiffrement de Paillier. Nous supposons que l'espace des textes en clair est défini par un module supérieur à la somme des carrés des x_i. De plus, nous supposons que seul le serveur détient la clé privée

Calcul de la moyenne

Algorithm 34 : Calcul de la moyenne

Entrée : $x_1, \dots x_N$, une clé publique de Paillier
Sortie : La moyenne des x_i

1: **pour tout** $j \in E, \mathcal{P}_j$: **faire**
2: calcule $y_i = Enc(x_i)$ et l'envoie à $\mathbb{S}$
3: **fin du pour**
4: Le serveur calcule et retourne $m = \frac{1}{N} Dec(\prod_{i=1}^{N} y_i)$

Démonstration.
Comme la fonction de chiffrement est additivement homomorphe, on a :
$$\prod_{i=1}^{N} y_i = \sum_{i=1}^{N} Enc\,(x_i) \Longrightarrow \frac{1}{N} Dec(\prod_{i=1}^{N} y_i) = \frac{1}{N} \sum_{i=1}^{N} x_i$$
□

Calcul de la variance

On notant $\mathbb{V}$ la variance, on a la formule suivante :

$$\mathbb{V} = \frac{1}{N} \sum_{i=1}^{N} x_i^2 - \overline{x}^2 \textit{ avec } \overline{x} \textit{ désignant la moyenne.}$$

Algorithm 35 : Calcul de la variance

Entrée : $x_1, \dots x_N$, une clé publique de Paillier
Sortie : La variance des x_i

1: **pour tout** $j \in E, \mathcal{P}_j$: **faire**
2: calcule $y_i = Enc(x_i)$ et $z_i = Enc(x_i^2)$ et l'envoie le couple (y_i, z_i) à $\mathbb{S}$
3: **fin du pour**
4: $\mathcal{S}_1 = Dec(\prod_{i=1}^{N} y_i)$ et $Dec(\prod_{i=1}^{N} z_i)$
5: Le serveur retourne $\frac{1}{N}\mathcal{S}_2 - \left(\frac{1}{N}\mathcal{S}_1\right)^2$

Démonstration.

Nous avons :

$$\mathcal{S}_1 = \sum_{i=1}^{N} x_i \; et \; \mathcal{S}_2 = \sum_{i=1}^{N} x_i^2 \Longrightarrow \frac{1}{N}\mathcal{S}_2 - \left(\frac{1}{N}\mathcal{S}_1\right)^2 = \mathbb{V}$$

□

Ces protocoles simples présentent de nombreuses applications dans la vie réelle, notamment en permettant la collecte de statistiques sur des données privées de manière confidentielle.

3.5 Évaluation et résultats

Les résultats de nos implémentations sont présentés à l'annexe H. Nous avons réalisé une implémentation du schéma de chiffrement de Paillier ainsi que de sa version modifiée par Damgård-Jurik, en procédant à une comparaison des algorithmes de chiffrement et de déchiffrement correspondants. En outre, nous avons développé une implémentation des divers algorithmes associés au schéma de déchiffrement à seuil de Paillier. Les conclusions que nous en tirons sont les suivantes :

- ▷ Comparaison des schémas Paillier et Damgård-Jurik :
  Le schéma de Damgård-Jurik est deux fois plus rapide que celui de Paillier pour le chiffrement. En revanche, pour le déchiffrement, Paillier est presque aussi rapide que Damgård-Jurik. Les deux algorithmes se montrent également très efficaces ; par exemple, pour un module de 1535 bits, le temps de déchiffrement est inférieur à 0,03 seconde pour chacun d'eux. Il est important de noter que ces comparaisons ont été effectuées avec 8 modules, et l'on peut espérer que pour des modules beaucoup plus grands, le déchiffrement de Damgård pourrait devenir plus rapide que celui de Paillier.
- ▷ Comparaison des temps de génération des clés de déchiffrement et de vérification :
  Lors du déchiffrement partagé, la génération des clés secrètes de déchiffrement est beaucoup plus rapide que celle des clés de vérification. Par exemple, avec 1000 parties avec un seuil fixé à 600 et un module de 1535 bits, la génération des clés secrètes ne prend que 0,3 seconde, tandis que la génération des clés de vérification nécessite 130 secondes. Cette différence montre que la génération des clés de vérification est beaucoup plus coûteux en temps.

- ▷ Comparaison des temps de génération des preuves, des vérifications, des déchiffrements partiels et de combinaison :
  Le temps nécessaire pour générer les preuves est presque identique à celui requis pour leur vérification. De plus, le temps de déchiffrement partiel est comparable à celui nécessaire pour la combinaison des parts, avec une légère avance en termes de rapidité pour la combinaison.

Nous affirmons que, sur la base de ces résultats, les algorithmes du schéma de déchiffrement partagé de Paillier offrent des performances suffisantes pour une utilisation pratique, notamment dans le cadre du projet BI4people. Nous soulignons également que, compte tenu de la puissance de calcul dont nous disposons (voir annexe H), la génération de grands nombres premiers sûrs a constitué une difficulté. Cela a limité la possibilité d'élargir les comparaisons et d'obtenir de meilleurs résultats avec une puissance de calcul plus élevée.

Conclusion

Dans ce rapport, nous avons présenté le travail réalisé au sein du laboratoire ERIC.

Dans un premier temps, j'ai abordé les problèmes mathématiques qui sous-tendent la sécurité des systèmes cryptographiques explorés au cours de ce stage, en établissant des relations hiérarchiques pour certains d'entre eux. Ensuite, nous avons introduit le cadre général des calculs multipartites et exploré les différentes familles de chiffrement homomorphe, en étudiant en détail les systèmes **RSA**, **ElGamal**, **Paillier**, **BFV**, **BGV** et **CKKS**, avec un accent particulier sur les aspects techniques et mathématiques. Cette attention portée aux détails mathématiques des systèmes homomorphes mentionnés a été motivée par le fait que ces derniers avaient été sélectionnés lors d'une étude préliminaire dans le cadre du projet. Cependant, ce travail antérieur visait à fournir les principes fondamentaux des schémas de chiffrement homomorphe sans approfondir les aspects mathématiques.

La dernière partie, dédiée aux protocoles de calcul multipartite, a commencé par une étude des schémas de déchiffrement partagé à seuil, en présentant un schéma générique basé sur **RLWE**, pouvant être instancié avec les systèmes **BFV**, **BGV** et **CKKS**. La majeure partie du travail accompli au cours de ce stage a été consacrée à l'étude du schéma de déchiffrement partagé à seuil de Paillier, depuis les aspects théoriques jusqu'à l'implémentation des différents algorithmes associés. Les résultats de ces implémentations témoignent de l'efficacité du schéma et montrent qu'il peut être utilisé en pratique, notamment dans le cadre du projet **BI4people**.

En tant que contributions, nous avons proposé deux protocoles pour la gestion des clés, dont l'un fait appel au chiffrement et l'autre n'utilise que des opérations linéaires dans un corps. Nous avons également proposé deux protocoles pour la collecte de données statistiques, spécifiquement pour le calcul de la moyenne et de la variance, en utilisant le chiffrement

de Paillier.

Ce stage m'a permis non seulement d'approfondir mes connaissances théoriques, mais aussi de renforcer mes compétences techniques en cryptographie, tout en contribuant aux travaux du projet BI4people.
Cette expérience a renforcé ma passion pour les mathématiques en général et pour la cryptographie en particulier, et me motive grandement à poursuivre la recherche dans ce domaine.

Bibliographie

[BD99] Dan BONEH et Glenn DURFEE. "Cryptanalysis of RSA with private key d less than N 0.292". In : *Advances in Cryptology—EUROCRYPT'99 : International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*. Springer. 1999, p. 1-11.

[BGV12] Zvika BRAKERSKI, Craig GENTRY et Vinod VAIKUNTANATHAN. "(Leveled) fully homomorphic encryption without bootstrapping". In : *3rd Conference on Innovations in Theoretical Computer Science, ITCS 2012*. 2012, p. 309-325.

[Bon+99] Dan BONEH et al. "Twenty years of attacks on the RSA cryptosystem". In : *Notices of the AMS* 46.2 (1999), p. 203-213.

[Bra12] Zvika BRAKERSKI. "Fully homomorphic encryption without modulus switching from classical GapSVP". In : *Annual cryptology conference*. Springer. 2012, p. 868-886.

[CDN01] Ronald CRAMER, Ivan DAMGÅRD et Jesper B NIELSEN. "Multiparty computation from threshold homomorphic encryption". In : *Advances in Cryptology—EUROCRYPT 2001 : International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6–10, 2001 Proceedings 20*. Springer. 2001, p. 280-300.

[Che+17] Jung Hee CHEON et al. "Homomorphic encryption for arithmetic of approximate numbers". In : *Advances in Cryptology–ASIACRYPT 2017 : 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer. 2017, p. 409-437.

[CP92] David CHAUM et Torben Pryds PEDERSEN. "Wallet databases with observers". In : *Annual international cryptology conference*. Springer. 1992, p. 89-105.

[DJ00] Ivan B DAMGÅRD et Mads J JURIK. "Efficient protocols based on probabilistic encryption using composite degree residue classes". In : *BRICS Report Series* 7.5 (2000).

[DJN10] Ivan DAMGÅRD, Mads JURIK et Jesper Buus NIELSEN. "A generalization of Paillier's public-key system with applications to electronic voting". In : *International Journal of Information Security* 9 (2010), p. 371-385.

[Doa+23] Thi Van Thao DOAN et al. "A survey on implementations of homomorphic encryption schemes". In : *The Journal of Supercomputing* 79.13 (2023), p. 15098-15139.

[ElG85] Taher ELGAMAL. "A public key cryptosystem and a signature scheme based on discrete logarithms". In : *IEEE transactions on information theory* 31.4 (1985), p. 469-472.

[FV12] Junfeng FAN et Frederik VERCAUTEREN. "Somewhat practical fully homomorphic encryption". In : *Cryptology ePrint Archive* (2012).

[Gen09] Craig GENTRY. *A fully homomorphic encryption scheme*. Stanford university, 2009.

[LPR13] Vadim LYUBASHEVSKY, Chris PEIKERT et Oded REGEV. "A toolkit for ring-LWE cryptography". In : *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2013, p. 35-54.

[MBH23] Christian MOUCHET, Elliott BERTRAND et Jean-Pierre HUBAUX. "An efficient threshold access-structure for rlwe-based multiparty homomorphic encryption". In : *Journal of Cryptology* 36.2 (2023), p. 10.

[Pai99] Pascal PAILLIER. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In : *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques*. T. 1592. Lecture Notes in Computer Science. Springer, 1999, p. 223-238. DOI : 10.1007/3-540-48910-X_16.

[RAD+78] Ronald L RIVEST, Len ADLEMAN, Michael L DERTOUZOS et al. "On data banks and privacy homomorphisms". In : *Foundations of secure computation* 4.11 (1978), p. 169-180.

[RSA78] Ronald L RIVEST, Adi SHAMIR et Leonard ADLEMAN. "A method for obtaining digital signatures and public-key cryptosystems". In : *Communications of the ACM* 21.2 (1978), p. 120-126.

[Sha79] Adi SHAMIR. "How to share a secret". In : *Communications of the ACM* 22.11 (1979), p. 612-613.

[Sho00] Victor SHOUP. "Practical threshold signatures". In : *Advances in Cryptology—EUROCRYPT 2000 : International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19*. Springer. 2000, p. 207-220.

[Sti05] Douglas R STINSON. *Cryptography : theory and practice*. Chapman et Hall/CRC, 2005.

[TY98] Yiannis TSIOUNIS et Moti YUNG. "On the security of ElGamal based encryption". In : *International Workshop on Public Key Cryptography*. Springer. 1998, p. 117-134.

[Wie90] Michael J WIENER. "Cryptanalysis of short RSA secret exponents". In : *IEEE Transactions on Information theory* 36.3 (1990), p. 553-558.

[Yao82] Andrew C YAO. "Protocols for secure computations". In : *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE. 1982, p. 160-164.

ANNEXES

A Preuve propriétés de la fonction λ [1.1.1]

Démonstration. D'abord w appartient au groupe $\mathbb{Z}_n^\times$, et dans ce dernier $\lambda(n)$ est multiple de l'ordre de w, donc $w^{\lambda(n)} \equiv 1 \mod n$.
D'autre part, la congruence $w^{\lambda(n)} \equiv 1 \mod n$ implique qu'il existe $k \in \mathbb{Z}$ tel que $w^{\lambda(n)} = 1 + kn$ et donc $w^{n\lambda(n)} = (1+kn)^n \equiv 1 \mod n^2$. □

B Démonstration de la proposition 1.2.1

Démonstration. Soient $\mathcal{S}$, $\mathcal{S}_p$ et $\mathcal{S}_q$ les sous-groupes des carrés dans $\mathbb{Z}_N^\times$, $\mathbb{Z}_p^\times$ et $\mathbb{Z}_q^\times$ respectivement. D'après le Théorème des Restes Chinois, il existe un isomorphisme entre $\mathbb{Z}_N^\times$ et $\mathbb{Z}_p^\times \times \mathbb{Z}_q^\times$ qui induit un isomorphisme entre les groupes $\mathcal{S}$ et $\mathcal{S}_p \times \mathcal{S}_q$. D'après les lemmes [1.2.1] et [1.2.2], le groupe $\mathcal{S}_p \times \mathcal{S}_q$ est cyclique d'ordre $p' \times q' = \frac{\phi(N)}{4}$. □

C Schéma de McEliece et Sarwate

Ce schéma utilise une propriété des codes MDS (Maximum Distance Separable). En effet, on a la proposition suivante :

Proposition C.1. *pour un code MDS $\mathcal{C}$ de longueur $n+1$, de dimension t et de distance minimale $n-t+2$ défini sur un corps $\mathbb{F}_q$, que l'on note simplement par $\mathcal{C} = [n+1, t, n-t+2]_q$. Soit $a_0 \in \mathbb{F}_q$, un secret, et $a_1, a_2, \ldots, a_{t-1} \in \mathbb{F}_q$, il existe un unique mot de code $c = (c_0, c_1, \ldots, c_n) \in \mathcal{C}$ tel que $c_i = a_i$ pour $0 \leq i \leq t-1$.*

Formellement, avec ce schéma, on procède comme suit :

1. On choisit un secret $s = a_0 \in \mathbb{F}_q$.
2. On choisit $a_1, a_2, \ldots, a_{t-1} \in \mathbb{F}_q$.
3. On calcule $c \in \mathcal{C}$ tel que $c_i = a_i$ pour $0 \leq i \leq t-1$. Par exemple, on peut considérer une matrice génératrice $G = [I_t \mid A]$ de $\mathcal{C}$ sous forme systématique, de sorte que $c = (a_0, a_1, \ldots, a_{t-1})G$.

Remarque C.1. *Si l'on considère un code $GRS_{n+1,t}(\alpha, \beta)$ avec $\alpha = (0, 1, \ldots, n)$ et $\beta = (1, \ldots, 1)$, alors le schéma de McEliece et Sarwate correspond au schéma de Shamir.*

D Preuve déchiffrement RSA [3]

Démonstration. Nous distinguons les trois cas exclusifs suivants :

1. si $M \equiv 0 \mod N$, alors $c^d \equiv M^{ed} \equiv M$
2. si $pgcd(M, N) = 1$, alors $M \in \mathbb{Z}_N^\times$, donc

$$c^d \equiv M^{ed} \equiv M \mod N$$

3. si $pgcd(M, N) \neq 1$, alors p ou q divise M. Supposons sans perte de généralité que p divise M. Alors $pgcd(q, M) = 1$ et donc dans le groupe $\mathbb{Z}_q^\times$, on a

$$M^{q-1} \equiv 1 \mod q \Longrightarrow M^{\phi(N)} \equiv 1 \mod q \; car \; (q-1) \; divise \; \phi(N)$$

D'où $M^{ed-1} \equiv 1 \mod q \Longrightarrow M^{ed} \equiv M \mod q$. D'autre part, p divise M donc $M^{ed} \equiv M \mod p$. Finalement, $M^{ed} \equiv M \mod N$

□

E Preuve déchiffrement Paillier [9]

Démonstration. Comme $pgcd(N, \phi(N)) = 1$, donc il existe $a \in \mathbb{Z}_N$, $b \in \mathbb{Z}_N^*$ tels que $g = (1+N)^a b^N$ (d'après l'isomorphisme de la démonstration du théorème 1.2.1)
On a :

$$\begin{aligned} L(c^{\lambda(N)} \mod N^2) &= L((g^M x^N)^{\lambda(N)} \mod N^2) \\ &= L((1+N)^{aM\lambda(N)}(xb^M)^{N\lambda(N)}) \mod N^2) \end{aligned}$$

comme $xb^M \in \mathbb{Z}_{N^2}^*$, donc on a

$$(xb^M)^{\lambda(N)N} \equiv 1 \mod N^2 \; propriété \; de \; la \; fonction \; \lambda$$

Il suit,

$$\begin{aligned} L(c^{\lambda(N)} \mod N^2) &= L((1+N)^{aM\lambda(N)} \mod N^2) \\ &= aM\lambda(N) \mod N \end{aligned}$$

d'autre part,

$$\begin{aligned} L(g^{\lambda(N)} \mod N^2) &= L((1+N)^{a\lambda(N)} b^{N\lambda(N)} \mod N^2) \\ &= a\lambda(N) \end{aligned}$$

d'où

$$\frac{L(c^{\lambda(N)} \mod N^2)}{L(g^{\lambda(N)} \mod N^2)} = \frac{aM\lambda(N)}{a\lambda(N)} = M$$

□

F Preuve de validité de l'algorithme de combinaison [24]

Démonstration. (Validité de l'algorithme de combinaison)

Soit S un ensemble de $t+1$ parties participant au déchiffrement. Chaque partie $\mathcal{P}_i$ détient donc un couple $(\delta_i, \text{preuve}_i)$, constitué d'un déchiffrement partiel δ_i et d'une preuve de validité preuve_i.
D'une part, nous avons :

$$c^{4\Delta^2 m\beta} = (1+NM)^{4\Delta^2 m\beta} x^{4\Delta^2 mN\beta} \mod N^2$$

comme $\mathbb{Z}_N^*$ est d'ordre $\phi(N) = 4m$, donc $x^{4\Delta^2 mN\beta} = 1$.
Ainsi,

$$c^{4\Delta^2 m\beta} = 1 + 4NM\Delta^2 m\beta \mod N^2$$

d'autre part,

$$c^{4\Delta^2 m\beta} \overset{(1)}{=} c^{4\Delta \sum_{j\in S} \lambda_0^S(j) s_{k_j}} \overset{(2)}{=} \prod_{j\in s} c^{4\Delta\lambda_0^S(j) s_{k_j}} \overset{(3)}{=} \prod_{j\in S} \delta_j^{2\lambda_0^S(j)} \mod N^2$$

et donc :

$$L\left(\prod_{j\in S} \delta_j^{2\lambda_0^S(j)} \mod N^2\right) = L\left(1 + 4NM\Delta^2 m\beta \mod N^2\right) = 4M\Delta^2 m\beta$$

enfin :

$$L\left(\prod_{j\in S} \delta_j^{2\lambda_0^S(j)} \mod N^2\right) \times \left(4\Delta^2 m\beta\right)^{-1} \mod N = M$$

$(1) : \Delta m\beta = f(0) = \sum_{j\in S} \lambda_0^S(j) s_{k_j}$, $\quad (2) : \textit{triviale}$, $\quad (3) : \delta_j = c^{2\Delta s_{k_j}}$ □

G La probabilité que la condition $z < 2^{l_2+2l_1}$ soit vraie.

Nous avons : $r \in \left[0, 2^{l_2+2l_1}\right[$ et $(sk_i, h) \in [0, Nm[\times \left[0, 2^{l_1}\right[$. On a donc :

$$sk_i \times h \in \left[0, m \times 2^{l_2+l_1}\right[\Longrightarrow r + sk_i \times h \in \left[0, 2^{l_2+2l_1} + m \times 2^{l_2+l_1}\right[.$$

Ainsi, le nombre d'entiers z dans l'intervalle $\left[0, 2^{l_2+2l_1} + m \times 2^{l_2+l_1}\right[$ qui sont supérieurs à $2^{l_2+2l_1}$ est égal à $2^{l_2+2l_1} + m \times 2^{l_2+l_1} - 2^{l_2+2l_1} = m \times 2^{l_2+l_1}$. Par conséquent, la probabilité qu'un élément $z \in \left[0, 2^{l_2+2l_1} + m \times 2^{l_2+l_1}\right[$ soit supérieur à $2^{l_2+2l_1}$ est :

$$p = \frac{m \times 2^{l_2+l_1}}{2^{l_2+2l_1} + m \times 2^{l_2+l_1}} = \frac{1}{m^{-1} \times 2^{l_1} + 1} \leq \frac{1}{m^{-1} \times 2^{l_1}} = \frac{m}{2^{l_1}}.$$

Ainsi, avec une grande taille de la fonction de hachage, cette probabilité devient négligeable.

H Évaluation et résultats

Les résultats exposés dans cette section ont été obtenus à l'aide de Sagemath version *9.1*, exécuté sur un ordinateur avec les caractéristiques suivantes :

- ▷ Processeur : Intel(R) Celeron(R) N4120 CPU @ 1.10GHz 1.10 GHz
- ▷ Mémoire RAM installée : 8,00 Go (dont 7,82 Go utilisables)
- ▷ Type de système : Système d'exploitation 64 bits,

Les facteurs premiers sûrs des différents modules utilisés sont rapportés à l'annexe I.

H.1 Comparaison du schémas de Paillier avec le celui de Damgård et Jurik

Chiffrement Paillier	modules : N	temps (s)
	16	$5.76066.10^{-5}$
	32	$6.83522.10^{-5}$
	64	$7.29537.10^{-5}$
	128	0.00018
	256	0.00051
	512	0.00128
	1024	0.00832
	1535	0.02836

TABLE 1 – Paillier

Chiffrement Damgård.	modules : N	temps (s)
	16	$7.73978.10^{-5}$
	32	$7.91382.10^{-5}$
	64	$8.22663.10^{-5}$
	128	0.00016
	256	0.00032
	512	0.00064
	1024	0.00424
	1535	0.01434

TABLE 2 – Damgård et Jurik

Déchiffrement Paillier	modules : N	temps (s)
	16	$7.58171.10^{-5}$
	32	$3.82661.10^{-5}$
	64	$5.29313.10^{-5}$
	128	0.00010
	256	0.00048
	512	0.00116
	1024	0.00827
	1535	0.02864

TABLE 3 – Paillier

Déchiffrement Damgård.	modules : N	temps (s)
	16	$2.31313.10^{-5}$
	32	$1.15489.10^{-5}$
	64	$1.92475.10^{-5}$
	128	$7.37667.10^{-5}$
	256	0.00041
	512	0.00110
	1024	0.00806
	1535	0.02828

TABLE 4 – Damgård et Jurik

H.2 Schéma de déchiffrement partagé à seuil

Dans la suite, n désigne le nombre de parties, t représente le seuil, et N correspond à la taille du module en bits.

Algorithme de génération des clés

On note SK et VK les listes respectivement des clés de déchiffrement et des clés de vérification.

Génération des clés				
n	t	N	temps pour SK	temps pour VK
10	6	16	0.00097	0.00043
		32	0.00088	0.00038
		64	0.00100	0.00046
50	30	32	0.00183	0.00182
		64	0.00249	0.00307
		128	0.00253	0.00556
100	60	64	0.00367	0.00778
		128	0.00438	0.01973
		256	0.00719	0.07283
300	180	128	0.01411	0.15489
		256	0.01637	0.56826
		512	0.02127	0.89082
500	300	256	0.03224	1.56300
		512	0.03736	2.29542
		1024	0.10419	13.03314
1000	600	512	0.11894	9.03050
		1024	0.21423	47.64457
		1535	0.37226	130.97276

TABLE 5 – Clés de déchiffrement et de vérification

Algorithme de génération de preuves, de vérification des déchiffrements partiels et de combinaison

Pour des paramètres n, t, et N fixés, PR représente la liste des preuves de validité, VER le temps de vérification des preuves, Δ la liste des déchiffrements partiels, et $COMB$ le résultat de l'algorithme de combinaison.

Génération des preuves, vérifications, déchiffrements et combinaison						
n	t	N	PR	VER	Temps pour Δ	Temps pour $COMB$
10	*6*	16	0.00343	0.00351	$6.66332.10^{-5}$	0.00031
		32	0.00513	0.00536	$6.84857.10^{-5}$	0.00032
		64	0.00963	0.01034	$9.91940.10^{-5}$	0.00037
50	*30*	32	0.02487	0.02514	0.00058	0.00391
		64	0.04639	0.04856	0.00107	0.00432
		128	0.09093	0.09801	0.00347	0.00651
100	*60*	64	0.09371	0.09814	0.00379	0.01642
		128	0.18590	0.20403	0.01117	0.02191
		256	0.40842	0.43216	0.04142	0.04162
300	*180*	128	0.65254	0.68946	0.09010	0.20810
		256	1.60390	1.73243	0.33819	0.42015
		512	2.89376	3.0652	0.53531	0.50656
500	*300*	256	3.39553	3.62246	0.93571	1.25264
		512	5.97214	6.17590	1.39219	1.57358
		1024	22.43264	23.11856	6.60688	7.15454
1000	*600*	512	16.87081	17.42978	5.56329	6.77517
		1024	59.28363	61.72692	25.28336	29.51431
		1535	141.50759	145.08716	74.27168	59.19997

TABLE 6 – Preuves de validité, déchiffrements partiels et combinaison

1 2

1. Pour les 1000 parties (dernier bloc), la moyenne a été calculée à partir de seulement 1 itération pour les preuves et les vérifications

2. Pour le module de 1535 bits, la moyenne a été calculée à partir de seulement 10 itérations pour les déchiffrements partiels et la combinaison

I Modules

Nous présentons les deux facteurs premiers sûrs p et q de chaque module $N = p \times q$ utilisés dans nos implémentations.

▷ Module de 16 bits : $p = 167$, $q = 227$

▷ Module de 32 bits : $p = 36923$, $q = 39779$

▷ Module de 64 bits : $p = 2781598187$, $q = 2768976227$

▷ Module de 128 bits : $p = 17186230932211742123$, $q = 11097589242371475659$

▷ Module de 256 bits :

$p = 32779355440777431096601891748998820 2667$,

$q = 22069815380386303938177083341683029 4047$

▷ Module de 512 bits :

$p = 66531039064013288633755153326125662211250217009400068263316801199327330024327$,

$q = 72372199050812386454706125814972027970581398794212172114336449672746928730847$

▷ Module de 1024 bits :

$p = 87470801890767688631610107749472610551050643790706350044113265628303697357191871241376742900690639427250863630763303135371193610544106754225257550833797 87$,

$q = 95968949749586266187368867612044575133323467600853575578922802713072927585965672393068254115986312112128711649142718655212746352594715635194671379767175 47$

▷ Module de 1535 bits :

$p = 99450320947364235415133761796420866395671193055687023466331087315803895151869831202101947873154153190017098396863651425498013 54$

$$85298226518293534165124985028233239856763992453516812988129823772974588359199772239682289311520746086187,$$

$$q = 97422940036091485231707428999284808218178213215510498338925117461989610928220736679570749286443476474866566512392982447629135617199674607818231112747642525220396676085623254969644329816825886228435393499088249210768083659400985035 9$$