Chiffrement fonctionnel pour le produit scalaire

Préparé par :

Lopez Angélique
angelique.lopez@etu.u-bordeaux.fr

Reyre Fivos
fivos.reyre@etu.u-bordeaux.fr

et

Zitouni Terki Sid Ali
sid-ali.zitouni-terki@etu.u-bordeaux.fr

Encadré par :

Castagnos Guilhem

Master 2 CSI

Janvier - Février 2021

Table des matières

1 Introduction

Le chiffrement fonctionnel est une technique récente généralisant le chiffrement asymétrique mais se différenciant de ce dernier en proposant un déchiffrement non binaire de l'information. Plus formellement, une autorité détient les clefs maîtres publique mpk et secrète msk. Le chiffrement d'un message m se fait classiquement à partir de la clé mpk. Le déchiffrement, quant à lui, dépend d'une clé k à partir de laquelle l'autorité dérive une clef secrète sk_k et ne permet pas de retrouver entièrement m mais l'image de k et m par une fonction F appelée fonctionnalité.

En particulier, nous nous intéressons au chiffrement fonctionnel utilisant comme fonctionnalité le produit scalaire. Dans ce cas, les messages sont des vecteurs et le déchiffrement peut-être vu comme une somme pondérée par les poids définis par k [**1**].

Dans le cadre de notre projet, nous étudions certaines constructions de chiffrement fonctionnel [**2**], [**3**], leur sécurité en termes d'indistinguabilité et nous proposons une implémentation de ces schémas à l'aide de Sage.

Nous nous focalisons tout d'abord sur les bases nécessaires à la compréhension de la suite. Puis, nous détaillons un schéma de chiffrement fonctionnel construit à partir d'un schéma de chiffrement à clef publique quelconque [**2**]. Ce schéma est ensuite explicité dans le cas où le schéma de chiffrement asymétrique de base est le cryptosystème de ElGamal. Cette construction présentant la contrainte d'une sécurité sémantique sélective, nous nous intéressons ensuite à une amélioration de ce schéma qui lui est IND-CPA [**3**]. Pour finir, nous étudions un schéma basé sur le chiffrement de Paillier qui, outre le fait d'être sémantiquement sûr, ne présente pas de coût lié au calcul d'un logarithme discret [**3**].

2 Prérequis

Dans cette partie, nous présentons les prérequis nécessaires à la compréhension du reste du rapport.

2.1 Chiffrement à clef publique (PKE)

2.1.1 Schéma [2]

Un schéma de **chiffrement à clef publique**, ou **PKE**, est un tuple de trois algorithmes $\mathcal{E} = (Setup, Encrypt, Decrypt)$:

1. $Setup(1^\lambda)$ renvoie les clefs publique p_k et secrète s_k étant donné le paramètre de sécurité λ.
2. $Encrypt(p_k, m)$ renvoie un chiffré c de m pour la clef publique p_k.
3. $Decrypt(s_k, c)$ renvoie le message m' correspondant au déchiffrement de c par la clef secrète s_k.

Le schéma est correct si pour tout $(p_k, s_k) \leftarrow Setup(1^\lambda)$, tout message m et chiffré $c = Encrypt(p_k, m)$, on a $Decrypt(s_k, c) = m$.

2.1.2 Sécurité [2], [3]

L'indistinguabilité est le critère utilisé pour évaluer la sécurité des schémas de ce rapport. Elle correspond à la capacité d'un adversaire $\mathcal{A}$ à faire la distinction entre deux chiffrés provenant de deux messages choisis. Plus formellement, un schéma de chiffrement est dit **sémantiquement sûr** ou **IND-CPA** si pour tout algorithme probabiliste polynômial $\mathcal{A}$ qui joue l'expérience $Exp_\lambda^{IND-CPA}(\mathcal{A})$, l'avantage de $\mathcal{A}$, $Adv_\lambda^{IND-CPA}(\mathcal{A}) = |1/2 - Pr(Exp_\lambda^{IND-CPA}(\mathcal{A}) = 1)|$ est négligeable.
Une **sécurité sémantique sélective** ou **s-IND-CPA** est aussi utilisée. Dans ce cadre, l'adversaire $\mathcal{A}$ ne dispose pas de la clef publique pour choisir les messages à distinguer. Elle correspond à une notion de sécurité moins forte que la sécurité IND-CPA de base.
Les expériences permettant de tester la sécurité d'un schéma de chiffrement à clef publique sont les suivantes :

$\text{Exp}_{\epsilon,\lambda}^{IND-CPA}(\mathcal{A})$:
$(pk, sk) \xleftarrow{\$} Setup(1^\lambda)$
$(m_0, m_1, s) \leftarrow \mathcal{A}_1(pk)$
$b^* \xleftarrow{\$} \{0, 1\}$
$Ct^* \xleftarrow{\$} Encrypt(pk, m_{b^*})$
$b \leftarrow \mathcal{A}_2(s, Ct^*)$
Si $b = b^*$:
Retourner 1
Retourner 0

$\text{Exp}_{\epsilon,\lambda}^{s-IND-CPA}(\mathcal{A})$:

$(m_0, m_1, s) \leftarrow \mathcal{A}_1()$

$(pk, sk) \xleftarrow{\$} Setup(1^\lambda)$

$b^* \xleftarrow{\$} \{0,1\}$

$Ct^* \xleftarrow{\$} Encrypt(pk, m_{b^*})$

$b \leftarrow \mathcal{A}_2(s, Ct^*)$

Si $b = b^*$:

Retourner 1

Retourner 0

2.2 Schémas de chiffrement à clef publique utiles

Nous donnons deux exemples de chiffrements à clef publique utilisés dans la suite.

2.2.1 ElGamal

Le schéma de chiffrement à clef publique de **ElGamal** dans l'exposant est défini par $\mathcal{E} = (Setup, Encrypt, Decrypt)$:

1. $Setup(1^\lambda)$ retourne (pk, sk) avec $pk = (G, q, g, h)$ et $sk = x \in \mathbb{Z}_q$ où G est un groupe d'ordre q, g est un générateur de G et $h = g^x$.
2. $Encrypt(pk, m)$ tire un entier aléatoire $r \in \mathbb{Z}_q$ et retourne c chiffré de m défini par $c = (c_0, c_1)$ où $c_0 = g^r$ et $c_1 = g^m h^r$.
3. $Decrypt(sk, c)$ retourne m' en calculant $log_g(\frac{c_1}{c_0^{sk}})$.

Le schéma est correct puisque pour tout $(p_k, s_k) \leftarrow Setup(1^\lambda)$, tout message m et chiffré $c = Encrypt(p_k,\ m)$, on a : $Decrypt(sk, c) = log_g(\frac{c_1}{c_0^{sk}}) = log_g(\frac{g^m h^r}{(g^r)^x}) = log_g(g^m) = m$.

2.2.2 Paillier

Le schéma de chiffrement à clef publique de **Paillier** est défini par $\mathcal{E} = (Setup, Encrypt, Decrypt)$:

1. $Setup(1^\lambda)$ génère un module RSA $N = pq$. L'algorithme retourne les clefs publique $p_k = N$ et secrète $s_k = \varphi(N)$.
2. $Encrypt(p_k,\ m)$ tire de manière aléatoire et uniforme un $r \in \mathbb{Z}_{N^2}^*$ et renvoie un chiffré c de $m \in \mathbb{Z}_N$ avec $c = (1+N)^m r^N \mod N^2$.
3. $Decrypt(s_k,\ c)$ renvoie le message déchiffré $m' = \frac{((c^{s_k} \mod N^2)-1)s_k^{-1}}{N}$.

Le schéma est correct puisque pour tout $(p_k, s_k) \leftarrow Setup(1^\lambda)$, tout message m et chiffré $c = Encrypt(p_k, m)$, on a :
$c^{s_k} \bmod N^2 = (1+N)^{m\varphi(N)} \times r^{N\varphi(N)} \bmod N^2 = (1 + Nm\varphi(N)) \times 1 \bmod N^2$ par le binôme de Newton.
Ainsi : $Decrypt(s_k, c) = \frac{((c^{s_k} \bmod N^2)-1)s_k^{-1}}{N} = \frac{(1+Nm\varphi(N)-1)\times\varphi(N)^{-1} \bmod N^2}{N} = m$.

2.3 Chiffrement fonctionnel (FE)

Le chiffrement fonctionnel consiste à chiffrer des messages avec une clef publique maître mpk mais contrairement au chiffrement à clef publique, le déchiffrement se fait avec une clef secrète sk_k dérivée correspondant à une entrée k. Le but étant de ne pas déchiffrer totalement l'information mais une partie correspondant à une fonction F du message chiffré et de k.

2.3.1 Fonctionnalité [1], [2]

On appelle **fonctionnalité**, une fonction F définie de la manière suivante : $F : K \times X \to \Sigma \cup \{\perp\}$, où K est l'espace des clés, X l'espace des messages, Σ l'espace des sorties et $\perp$ est une chaîne spéciale non contenue dans Σ, représentant une erreur.

La **fonctionnalité du produit scalaire**, où **IP**, est une fonctionnalité $F_{IP}(x, y) =< x, y >$, où x et y sont des vecteurs de $K \times X$ de même longueur.

2.3.2 Schéma [2]

Un schéma de **chiffrement fonctionnel** pour une fonctionnalité F est un tuple de quatre algorithmes $\mathcal{F} = (Setup, KeyDer, Encrypt, Decrypt)$:

1. $Setup(1^\lambda)$ renvoie les clefs maîtres publique mpk et secrète msk étant donné le paramètre de sécurité λ.
2. $KeyDer(mpk, k)$ renvoie la clef secrète sk_k correspondant à la clef publique maître mpk et à la clef k.
3. $Encrypt(mpk, x)$ renvoie un chiffré c de x pour la clef publique mpk.
4. $Decrypt(mpk, c, sk_k)$ renvoie y.

Le schéma est correct si pour tout $(mpk, msk) \leftarrow Setup(1^\lambda)$, toute clef k, tout message x, pour $sk_k \leftarrow KeyDer(msk, k)$ et $c \leftarrow Encrypt(mpk, x)$, on a $Decrypt(mpk, c, sk_k) = F(k, x)$ où $F(k, x) \neq \perp$.

2.3.3 Sécurité [2]

Comme pour le chiffrement à clef publique, des expériences permettent d'évaluer la sécurité IND-CPA d'un schéma de chiffrement fonctionnel. Nous donnons l'expérience correspondant à l'IND-FE-CPA classique puis à l'IND-FE-CPA sélective, nommée s-IND-FE-CPA.

Il est intéressant de remarquer que la condition $F(k, m_0) = F(k, m_1)$ est nécessaire, sinon $\mathcal{A}$ peut réussir l'expérience par ce biais.

$\text{Exp}^{IND-FE-CPA}_{\epsilon,\lambda}(\mathcal{A})$:
$(mpk, msk) \xleftarrow{\$} Setup(1^\lambda)$
$V \leftarrow \emptyset$
$(m_0, m_1, s) \leftarrow \mathcal{A}_1(mpk)$
$b^* \xleftarrow{\$} \{0, 1\}$
$Ct^* \xleftarrow{\$} Encrypt(mpk, m_{b^*})$
$b \leftarrow \mathcal{A}_2(s, Ct^*)$
Si $b \neq b^*$ ou s'il existe k dans V tel que $F(k, m_0) \neq F(k, m_1)$:
Retourner 0
Retourner 1

Pendant son exécution, $\mathcal{A}$ (composé de $\mathcal{A}_1$ et $\mathcal{A}_2$) peut faire appel à l'oracle O :

$O(k)$:
$V \leftarrow V \cup \{k\}$
$sk_k \xleftarrow{\$} KeyDer(msk, k)$
Retourner sk_k

On défini également l'expérience s-IND-FE-CPA :

$\text{Exp}^{s-IND-FE-CPA}_{\epsilon,\lambda}(\mathcal{A})$:
$(m_0, m_1, s) \leftarrow \mathcal{A}_1()$
$(mpk, msk) \xleftarrow{\$} Setup(1^\lambda)$
$V \leftarrow \emptyset$
$b^* \xleftarrow{\$} \{0, 1\}$
$Ct^* \xleftarrow{\$} Encrypt(mpk, m_{b^*})$
$b \leftarrow \mathcal{A}_2(s, Ct^*)$
Si $b \neq b^*$ ou s'il existe k dans V tel que $F(k, m_0) \neq F(k, m_1)$:
Retourner 0
Retourner 1

Pendant son exécution, $\mathcal{A}$ (composé de $\mathcal{A}_1$ et $\mathcal{A}_2$) peut faire appel à l'oracle O :

$O(k)$:
$V \leftarrow V \cup \{k\}$
$sk_k \xleftarrow{\$} KeyDer(msk, k)$
Retourner sk_k

2.4 Hypothèses calculatoires

Nous rappelons deux hypothèses calculatoires sur lesquelles nous nous appuierons dans la suite pour prouver la sécurité des schémas étudiés.

2.4.1 DDH

Soit G un groupe cyclique d'ordre q, g un générateur et les distributions $D_0 = \{(g^a, g^b, g^{ab}) \mid a, b \in \mathbb{Z}_q\}$ et $D_1 = \{(g^a, g^b, g^c) \mid a, b, c \in \mathbb{Z}_q\}$. L'**hypothèse Décisionnelle de Diffie-Hellman** ou **DDH** suppose que ces deux distributions sont indistinguables.

2.4.2 DCR

Soit $N = pq$ avec p, q. L'**hypothèse décisionnelle de résiduosité quadratique**, ou **DCR**, suppose que les distributions $D_0 = \{z = z_0^N \bmod N^2 | z_0 \xleftarrow{\$} Z_N^*\}$ et $D_1 = \{z \xleftarrow{\$} Z_{N^2}^*\}$ sont indistinguables.

3 Construction d'un schéma FE-IP à partir d'un schéma PKE

Dans cette partie, nous nous intéressons à la construction et à la sécurité d'un schéma FE-IP construit à partir d'un schéma PKE quelconque mais possédant certaines propriétés. Ce schéma est s-IND-FE-CPA et nous en détaillons une preuve. Pour finir, nous construisons ce schéma à partir du schéma PKE ElGamal et nous retrouvons le schéma DDH-IP donné par les mêmes auteurs [**2**].

3.1 Hypothèses sur le schéma PKE

Soit $\mathcal{E} = (Setup, Encrypt, Decrypt)$ un schéma PKE. Pour construire le schéma FE-IP, le schéma PKE doit satisfaire les propriétés décrites dans cette section.

3.1.1 Structure

Les clés publiques de $\mathcal{E}$ sont des éléments d'un groupe multiplicatif G, les clés privées sont des éléments d'un groupe additif H et l'espace des messages est $\mathbb{Z}_q$ pour un premier q. Le chiffré d'un message x est de la forme $Ct = (ct_0, ct_1)$. *Encrypt* peut être vu comme un couple d'algorithmes (C,E) où $ct_0 = C(r)$ et $ct_1 = E(x, pk; r)$ avec r un aléa et pk une clé publique.

3.1.2 LKH

$\mathcal{E}$ respecte la propriété LKH : pour tous couple de clefs $(pk_1, sk_1), (pk_2, sk_2) \in G \times H$ et tous $y_1, y_2 \in \mathbb{Z}_q$, $(pk_1^{y_1} pk_2^{y_2},\ y_1 sk_1 + y_2 sk_2) \in G \times H$ est un couple de clefs pour $\mathcal{E}$.

3.1.3 LCH

$\mathcal{E}$ respecte la propriété LCH : $E(x_1 + x_2, pk_1 pk_2; r) = E(x_1, pk_1; r) \times E(x_2, pk_2; r)$ pour tous messages x_1, x_2 et clés publiques pk_1, pk_2.

3.1.4 Réutilisation de l'aléa

$\mathcal{E}$ respecte la propriété de réutilisation de l'aléa : deux chiffrés différents Ct et Ct' de m tels que $ct_0 = ct'_0 = C(r)$ ne révèlent pas plus d'information sur m que deux chiffrés qui n'utilisent pas le même aléa.

3.2 Construction PKE-IP

Soit $\mathcal{E} = (Setup, Encrypt, Decrypt)$ un schéma de chiffrement à clef publique respectant les propriétés de la partie 3.1.

Le schéma **PKE-IP** se construit à partir de $\mathcal{E}$ par le tuple d'algorithmes $\mathcal{F} = (Setup, KeyDer, Encrypt, Decrypt)$:

1. $Setup(1^\lambda, 1^l, B)$ appelle l'algorithme de génération de clefs de $\mathcal{E}$ pour créer l couples de clefs (sk_1, pk_1), ..., (sk_l, pk_l) partageant les mêmes paramètres publics params. L'algorithme génère aussi l'espace des clefs de la fonctionnalités K_l et l'espace des messages $X_l = M = \{0, ..., B-1\}^l \subseteq \mathbb{Z}_q$. Il renvoie les clefs maîtres $mpk =$ (params, pk_1, ..., pk_l) et $msk = (sk_1, ..., sk_l)$.
2. $KeyDer(msk, y)$ retourne la clef secrète correspondant à $msk = (sk_1, ..., sk_l)$ et à la clef $y = (y_1, ..., y_l)$: $sk_y = \sum_i y_i sk_i$.
3. $Encrypt(mpk, x)$ retourne le chiffré c du message x par la clef publique maître mpk. Pour cela, il prend un aléatoire dans l'espace des aléatoires de $\mathcal{E}$ et calcule $c_0 = \mathcal{E}.C(r)$ et pour tout $i \in \{1, ..., l\}$, $c_i = \mathcal{E}.E(pk_i, x_i; r)$. Le chiffré c vaut $c = (c_0, (c_i)_i)$.
4. $Decrypt(mpk, c, sk_y)$ prenant en entrée une clef secrète sk_y liée à $y = (y_1, ..., y_l)$, retourne $\mathcal{E}.Decrypt(sk_y, (c_0, \prod_i c_i^{y_i})) = < x, y >$.

Le schéma est correct puisque pour tout $(mpk, msk) \leftarrow Setup(1^\lambda, 1^l)$, toute clef y, tout message x, pour $sk_y \leftarrow KeyDer(msk, y)$ et $c \leftarrow Encrypt(mpk, x)$, on a :

$$\begin{aligned} Decrypt(mpk, c, sk_k) &= \mathcal{E}.Decrypt(sk_y, (c_0, \textstyle\prod_i c_i^{y_i})) \\ &= \mathcal{E}.Decrypt(sk_y, (c_0, \textstyle\prod_i \mathcal{E}.E(pk_i, x_i; r)^{y_i})) \\ &= \mathcal{E}.Decrypt(sk_y, (c_0, \mathcal{E}.Encrypt(\textstyle\prod_i pk_i^{y_i}, \sum_i y_i x_i; r))) \\ &\quad \text{par la propriété LCH} \\ &= \textstyle\sum_i y_i x_i \text{ par la propriété LKH.} \end{aligned}$$

3.3 Sécurité

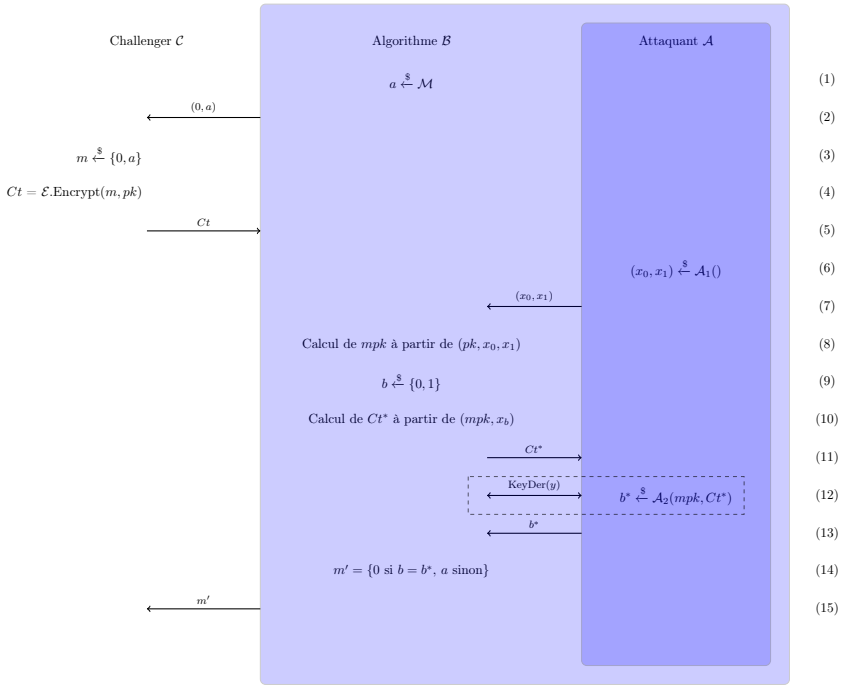
Le schéma PKE-IP détaillé dans la partie précédente 3.2 est s-IND-FE-CPA sous les hypothèses suivantes :

— le schéma PKE $\mathcal{E}$ servant à la construction est s-IND-CPA.
— les paramètres de $\mathcal{E}$ l, q et B sont tels que $q > lB^2$.

Soit $\mathcal{E}$ un schéma de chiffrement à clé publique vérifiant les hypothèses précédentes et $\mathcal{F}$ le schéma PKE-IP construit à partir de $\mathcal{E}$. Montrons que $\mathcal{F}$ est s-IND-FE-CPA.

Pour cela, supposons qu'il existe un algorithme $\mathcal{A}$ cassant la sécurité de $\mathcal{F}$ i.e. qui a un avantage non négligeable pour l'expérience s-IND-FE-CPA et construisons $\mathcal{B}$ qui casse la sécurité s-IND-CPA de $\mathcal{E}$.

3.3.1 Schéma général



3.3.2 Explications détaillées

(1) Choix des messages par $\mathcal{B}$

$\mathcal{B}$ commence par choisir deux messages dans $\mathcal{M} = \mathbb{Z}_q$: a aléatoire dans $\mathcal{M}$ et 0.

(2-7) Obtention du chiffré challenge et des messages de la part de $\mathcal{A}$

$\mathcal{B}$ donne ces deux messages à $\mathcal{C}$ qui lui renvoie une clé publique pk et un chiffré C_t de 0 ou a.
Il demande aussi à $\mathcal{A}$ deux messages x_0 et x_1 dans $\mathbb{Z}_q^l$.

(8) Fabrication de la clé publique mpk

$\mathcal{B}$ trouve une base $(z_1, ..., z_{l-1})$ de $(x_1 - x_0)^\perp$. Cette base est utile pour dériver les clefs secrètes. Ainsi, $(x_1 - x_0, z_1, ..., z_{l-1})$ est une base de $\mathbb{Z}_q^l$ et les vecteurs de la base canonique e_i peuvent être écrits dans cette base. Il existe donc pour tout $i \in \{1, ..., l\}$ et $j \in \{1, ..., l-1\}$, $\lambda_{i,j}$ et $\alpha_i \in \mathbb{Z}_q$ tels que :

$e_i = \alpha_i(x_1 - x_0) + \sum_j \lambda_{i,j} z_j$.

On a $x_{1,i} - x_{0,i} =< x_1 - x_0, e_i >=< x_1 - x_0, \alpha_i(x_1 - x_0) + \sum_j \lambda_{i,j} z_j >$
$= \alpha_i < x_1 - x_0, x_1 - x_0 > + \sum_j \lambda_{i,j} < x_1 - x_0, z_j >$
$= \alpha_i ||x_1 - x_0||^2$

Donc $\alpha_i = \frac{(x_{1,i} - x_{0,i})}{||x_1 - x_0||^2}$ où $||x_1 - x_0||^2 \neq 0$ modulo q pour tout i, puisque $q > l \times B^2$.
Ensuite, $\mathcal{B}$ crée un couple de clefs pour chaque $j \in \{1, ..., l-1\} : (pk_{z_j}, sk_{z_j}) \leftarrow Setup(1^\lambda)$. À ce stade, le tuple de l clés $(pk, pk_{z_1}, ..., pk_{z_{l-1}})$ forme une clé publique maître conforme pour $\mathcal{A}$. Il calcule enfin pour tout $i \in \{1, ..., l\}, pk_i = pk^{\alpha_i} \prod_j pk_{z_j}^{\lambda_{i,j}}$.
On applique donc juste un changement de base sur les pk_{z_j} pour obtenir $mpk = (pk_i)_i$. $\mathcal{B}$ construit donc bien une clé maître publique conforme à l'environnement de $\mathcal{A}$.

(9-11) Fabrication du chiffré challenge

$\mathcal{B}$ prend un bit b aléatoire.
Puis, il pose :

- $ct_0^* = ct_0$
  et
- $(ct_i^* = ct_1^{\alpha_i} \times \mathcal{E}.E(\gamma_i, 0; r) \times \mathcal{E}.E(1_H, x_{b,i}; r))_i$

où $C_t = (ct_0, (ct_i)_i)$ est le chiffré de 0 ou a fourni par $\mathcal{C}$.
$\mathcal{B}$ donne $C_t^* = (ct_0^*, (ct_i^*)_i)$ à $\mathcal{A}$.

(12) Fabrication des clefs secrètes dérivées

$\mathcal{B}$ fabrique les clefs secrètes dérivées que lui demande $\mathcal{A}$. Par hypothèse, $\mathcal{A}$ ne peut demander des clefs secrètes que pour des vecteurs y tels que $< x_0, y >=< x_1, y >$ soit $< x_1 - x_0, y >= 0$ donc $y \in (x_1 - x_0)^\perp$.

Pour fabriquer la clef correspondant à un vecteur y, $\mathcal{B}$ calcule :

$sk_y =< y, msk >= \sum_j y_i sk_i = \sum_j y_i \alpha_i sk + \sum_i \sum_j y_i \lambda_{i,j} sk_{z_j}$
$= \sum_j y_i \frac{(x_{1,i} - x_{0,i})}{||x_1 - x_0||^2} sk + \sum_i \sum_j y_i \lambda_{i,j} sk_{z_j}$

Comme $y \in (x_1 - x_0)^\perp$, le premier terme de la somme est nul, $\mathcal{B}$ n'a donc pas besoin de connaître sk pour pouvoir dériver la clé secrète. On a donc : $sk_y = \sum_j (\sum_i y_i \lambda_{i,j}) sk_{z_j}$.

De cette façon, $\mathcal{B}$ simule bien l'environnement de $\mathcal{A}$.

(13-15) Réponse de $\mathcal{B}$ au challenge

Si $\mathcal{A}$ trouve la valeur de b, $\mathcal{B}$ répond 0 à $\mathcal{C}$. Sinon, $\mathcal{B}$ répond 1.

3.3.3 Avantage de $\mathcal{B}$

1. Supposons que $\mathcal{C}$ a chiffré 0. Dans ce cas, on a pour tout i :
   $ct_i^* = ct_1^{\alpha_i} \times \mathcal{E}.E(\gamma_i, 0; r) \times \mathcal{E}.E(1_H, x_{b,i}; r)$
   $= \mathcal{E}.E(pk, 0; r)^{\alpha_i} \times \mathcal{E}.E(\gamma_i, 0; r) \times \mathcal{E}.E(1_H, x_{b,i}; r)$
   $= \mathcal{E}.E(pk, 0; r)^{\alpha_i} \times \mathcal{E}.E(\gamma_i, x_{b,i}; r)$
   $= (\prod_{k=1}^{\alpha_i} \mathcal{E}.E(pk, 0; r)) \times \mathcal{E}.E(\gamma_i, x_{b,i}; r)$
   $= \mathcal{E}.E(pk^{\alpha_i}\gamma_i, 0 + x_{b,i}; r)$ grâce à la propriété LCH
   $= \mathcal{E}.E(pk_i, x_{b,i}; r)$

   Ainsi, Ct^* dans ce cas correspond à un chiffré de la forme :

   $$Ct^* = \mathcal{F}.Encrypt(mpk, x_b)$$

   $\mathcal{A}$ réussit alors à distinguer x_b avec un avantage ϵ.

2. Supposons que $\mathcal{C}$ a chiffré a. Dans ce cas, on a pour tout i :
   $ct_i^* = ct_1^{\alpha_i} \times \mathcal{E}.E(\gamma_i, 0; r) \times \mathcal{E}.E(1_H, x_{b,i}; r)$
   $= \mathcal{E}.E(pk, a; r)^{\alpha_i} \times \mathcal{E}.E(\gamma_i, 0; r) \times \mathcal{E}.E(1_H, x_{b,i}; r)$
   $= \mathcal{E}.E(pk, a; r)^{\alpha_i} \times \mathcal{E}.E(\gamma_i, x_{b,i}; r)$
   $= (\prod_{k=1}^{\alpha_i} \mathcal{E}.E(pk, a; r)) \times \mathcal{E}.E(\gamma_i, x_{b,i}; r)$
   $= \mathcal{E}.E(pk^{\alpha_i}\gamma_i, a \times \alpha_i + x_{b,i}; r)$ grâce à la propriété LCH
   $= \mathcal{E}.E(pk_i, s_i; r)$

   Développons s_i :
   $s_i = a \times \alpha_i + x_{b,i} = a \times \frac{(x_{1,i} - x_{0,i})}{||x_1 - x_0||^2} + x_{b,i}$
   $= a \times \frac{(x_{1,i} - x_{0,i})}{||x_1 - x_0||^2} + bx_{1,i} + (1-b)x_{0,i}$
   $= (x_{1,i} - x_{0,i}) \times \frac{a}{||x_1 - x_0||^2} + b(x_{1,i} - x_{0,i}) + x_{0,i}$
   $= (x_{1,i} - x_{0,i}) \times (\frac{a}{||x_1 - x_0||^2} + b) + x_{0,i}$
   $= (x_{1,i} - x_{0,i}) \times u + x_{0,i}$ où $u = (\frac{a}{||x_1 - x_0||^2} + b)$
   $= ux_{1,i} + (1-u)x_{0,i}$

   Ainsi, Ct^* dans ce cas correspond à un chiffré de la forme :

   $$Ct^* = \mathcal{F}.Encrypt(mpk, s)$$

   où $s = ux_1 + (1-u)x_0$. Puisque a est choisi aléatoirement, u est donc également un élément aléatoire de $\mathcal{M}$. $s = ux_1 + (1-u)x_0$ est alors une combinaison linéaire aléatoire de x_0 et x_1, qui cache donc b. L'avantage de $\mathcal{A}$ sur le chiffré de ce message est alors 0.

Conclusion

Calculons l'avantage de $\mathcal{B}$:

$$\mathbf{Adv}_{\mathcal{E},\lambda}^{s-IND-CPA}(\mathcal{B}) = |Pr[\mathbf{Exp}_{\mathcal{E},\lambda}^{s-IND-CPA}(\mathcal{B}) = 1] - \tfrac{1}{2}|$$

$$
\begin{aligned}
&= |Pr[\mathbf{Exp}^{s-IND-CPA}_{\mathcal{E},\lambda}(\mathcal{B}) = 1 \ \cap \ m = 0] \\
&\qquad + Pr[\mathbf{Exp}^{s-IND-CPA}_{\mathcal{E},\lambda}(\mathcal{B}) = 1 \ \cap \ m = a] - \tfrac{1}{2}| \\
&= |\tfrac{1}{2} Pr[\mathbf{Exp}^{s-IND-CPA}_{\mathcal{E},\lambda}(\mathcal{B}) = 1 \mid m = 0] \\
&\qquad + \tfrac{1}{2} Pr[\mathbf{Exp}^{s-IND-CPA}_{\mathcal{E},\lambda}(\mathcal{B}) = 1 \mid m = a] - \tfrac{1}{2}| \\
&= |\tfrac{1}{2} Pr[\mathbf{Exp}^{s-IND-FE-CPA}_{\mathcal{F},\lambda}(\mathcal{A}) = 1 \mid m = 0] \\
&\qquad + \tfrac{1}{2} \times \tfrac{1}{2} - \tfrac{1}{2}| \\
&= |\tfrac{1}{2} Pr[\mathbf{Exp}^{s-IND-FE-CPA}_{\mathcal{F},\lambda}(\mathcal{A}) = 1 \mid m = 0] - \tfrac{1}{4}| \\
&= \tfrac{1}{2} |Pr[\mathbf{Exp}^{s-IND-FE-CPA}_{\mathcal{F},\lambda}(\mathcal{A}) = 1 \mid m = 0] - \tfrac{1}{2}| \\
&= \tfrac{1}{2} \mathbf{Adv}^{s-IND-FE-CPA}_{\mathcal{F},\lambda}(\mathcal{A})
\end{aligned}
$$

S'il existe un algorithme polynomial probabiliste $\mathcal{A}$ attaquant la sécurité s-IND-FE-CPA du schéma $\mathcal{F}$ ayant un avantage non négligeable ϵ, alors on peut construire un algorithme $\mathcal{B}$ polynomial probabiliste attaquant la notion s-IND-CPA du schéma $\mathcal{E}$ ayant un avantage non négligeable $\frac{1}{2}\epsilon$. Dans ce cas, le $\mathcal{E}$ n'est pas s-IND-CPA (contradiction).

Le schéma $\mathcal{F}$ est donc s-IND-FE-CPA.

3.4 Construction à partir du schéma PKE ElGamal

3.4.1 Vérification des hypothèses

Vérifions que le schéma ElGamal (partie 2.2.1) possède les propriétés nécessaires pour la construction d'un schéma PKE-IP. Soient G, q, g des paramètres publiques correspondant à ce schéma ElGamal.

Structure
Par construction, les clés publiques appartiennent à $(G, \times, 1)$, g étant un générateur de G et d'ordre $q-1$, les clés privées étant des exposants de g, ce sont des éléments de $(\mathbb{Z}_q, +, 0)$.

LKH
Soient deux clés publiques $pk_1 = h_1, pk_2 = h_2$ de clés privées correspondantes respectives $sk_1 = x_1, sk_2 = x_2$ où $h_i = g^{x_i}$ pour $i \in \{1, 2\}$. Soient $S = y_1 sk_1 + y_2 sk_2$ et $P = pk_1^{y_1} pk_2^{y_2}$ où $y_1, y_2 \in \mathbb{Z}_q$. On a :

$$P = (g^{x_1})^{y_1} \times (g^{x_2})^{y_2} = g^{y_1 x_1 + y_2 x_2} = g^{y_1 sk_1 + y_2 sk_2} = g^S$$

où $S \in \mathbb{Z}_q$ donc (P, S) forme bien une paire de clés valide du schéma.

LCH
Soient $m_1, m_2 \in G$ deux messages, $(pk_1, sk_1), (pk_2, sk_2), (P, S)$ trois paires de clés et comme précédemment, où $y_1 = y_2 = 1$. On a :
$E(m_1 + m_2, pk_1 pk_2; r) = g^{(m_1+m_2)}(g^{x_1+x_2})^r = g^{m_1} g^{m_2} g^{rx_1} g^{rx_1}$
$= g^{m_1}(g^{x_1})^r \times g^{m_2}(g^{x_2})^r = E(m_1, pk_1; r)E(m_2, pk_2; r)$

Réutilisation de l'aléa
On admet ce critère.

3.4.2 Schéma

Le schéma $\mathcal{F} = (Setup, KeyDer, Encrypt, Decrypt)$ PKE-IP construit à partir de $\mathcal{E}$ est le suivant :

1. $Setup(1^\lambda, 1^l)$ fait appel à $GroupGen$ pour générer $params = (G, q, g)$. Il fait appel l fois à $\mathcal{E}.Setup(1^\lambda)$ pour générer $\{(pk_i = h_i = g^{x_i}, sk_i = x_i)\}_{i=1}^l$. Il retourne $(mpk = (params, pk_1, .., pk_2))$ et $msk = (sk_1, ..., sk_l)$.
2. $KeyDer(msk, y)$ retourne $sk_y =< msk, y >= \sum_{i\in[l]} x_i y_i$.
3. $Encrypt(mpk, m)$ prend un message $m = (m_1..., m_l) \in G^l$, tire r aléatoirement dans $\mathbb{Z}_q$. Il pose $ct_0 = g^r = C(r)$ et fait appel l fois à $\mathcal{E}.E(m_i, pk_i; r)$ pour générer $ct_1 = \{ct_i = g^{m_i} h_i^r\}_{i=1}^l$. Il retourne $Ct = (ct_0, ct_1)$.
4. $Decrypt(mpk, sk_y, Ct)$ calcule $\frac{\prod_{i\in[l]} ct_i^{y_i}}{ct_0^{sk_y}} = g^{<m,y>}$. Il retourne $log_g(g^{<m,y>}) =< m, y >$

Le schéma est correct puisque :

$$\frac{\prod_{i\in[l]} ct_i^{y_i}}{ct_0^{sk_y}} = \frac{\prod_{i\in[l]}(g^{m_i} h_i^r)^{y_i}}{(g^r)^{sk_y}} = \frac{\prod_{i\in[l]}(g^{m_i y_i} g^{r x_i y_i})}{g^{r\sum_{i\in[l]} x_i y_i}} = g^{\sum_{i\in[l]} m_i y_i} = g^{<m,y>}$$

Le schéma PKE-IP détaillé dans cette partie présente l'avantage de pouvoir s'appliquer à plusieurs schémas PKE. Sa limitation réside dans le fait que sa sécurité est seulement sélective. Pour remédier à ce problème, l'article [**3**] ont proposé un schéma IND-CPA basé sur DDH. Il est une extension du schéma DDH-IP de [**2**].

4 Construction d'un schéma FE-IP basée sur DDH

Le schéma basé sur DDH de [**3**] propose une amélioration du schéma de la partie précédente. Il est IND-CPA et nous en détaillons une preuve reposant sur une séquence de jeux.

4.1 Schéma

Le schéma de chiffrement fonctionnel basé sur DDH et sémantiquement sûr se définit par $\mathcal{F} = (Setup, KeyDer, Encrypt, Decrypt)$:

1. $Setup(1^\lambda, 1^l)$ prend un groupe cyclique G d'ordre premier $q > 2^\lambda$ et deux générateurs g et h de G. L'algorithme tire aussi de manière aléatoire pour $i \in \{1, ..., l\}$, $s_i, t_i \in \mathbb{Z}_q$ et pose $h_i = g^{s_i} h^{t_i}$. Il renvoie enfin les clefs maîtres : $mpk = (G, g, h, \{h_i\}_{i=1}^l)$ et $msk = \{(s_i, t_i)\}_{i=1}^l$.
2. $KeyDer(msk, x)$ dérive la clef secrète correspondant à $x = (x_1, ..., x_l) \in \mathbb{Z}_q^l$ à partir de la clef maître secrète $msk = \{(s_i, t_i)\}_{i=1}^l$ en calculant : $sk_x = (s_x, t_x) = (< s, x >, < t, x >) = (\sum_{i=1}^{l} s_i x_i, \sum_{i=1}^{l} t_i x_i)$.
3. $Encrypt(mpk, y)$ chiffre le vecteur $y = (y_1, ..., y_l) \in \mathbb{Z}_q^l$ avec la clef maître publique mpk. Pour cela, l'algorithme prend un entier $r \in \mathbb{Z}_q$ de manière aléatoire et pose : $C = g^r, D = h^r$ et $\{E_i = g^{y_i} h_i^r\}_{i=1}^l$. Le chiffré retourné est alors : $C_y = (C, D, E_1, ..., E_l)$.
4. $Decrypt(mpk, sk_x, C_y)$ permet le déchiffrement de $C_y = (C, D, E_1, ..., E_l)$ par la clef dérivée secrète $sk_x = (s_x, t_x)$. Pour cela, l'algorithme calcule $E_x = \frac{\prod_{i=1}^{l} E_i^{x_i}}{C^{s_x} D^{t_x}}$ et retourne $log_g(E_x)$.

Le schéma est correct puisque dans l'algorithme de déchiffrement $Decrypt$:

$$E_x = \frac{\prod_{i=1}^{l} E_i}{C^{s_x} D^{t_x}} = \frac{\prod_{i=1}^{l} g^{y_i + r s_i x_i} h^{r t_i x_i}}{\prod_{i=1}^{l} g^{r s_i x_i} h^{r t_i x_i}} = g^{<x,y>}.$$

Ce schéma présente la contrainte de nécessiter le calcul d'un logarithme discret dans l'algorithme $Decrypt$, ce qui est coûteux. Ainsi, dans cette construction la valeur du produit scalaire $< x, y >$ est comprise entre 0 et un entier L. Le calcul du logarithme discret peut alors se faire en $O(L^{\frac{1}{2}})$ (avec l'algorithme de Pollard par exemple).

4.2 Sécurité

Dans cette section, nous montrons que le schéma précédent est IND-FE-CPA. Pour cela, nous prenons un adversaire $\mathcal{A}$ contre la sécurité IND-FE-CPA et nous utilisons une séquence de jeux permettant d'obtenir une majoration, négligeable, de l'avantage de $\mathcal{A}$.

Dans cette preuve et dans la suite, on note S_i l'événement : "$\mathcal{A}$ réussit l'expérience i" .

Jeu 0 :

Ce jeu correspond à l'expérience IND-FE-CPA classique (partie 2.3.3).
$\mathcal{B}$ envoie mpk à $\mathcal{A}$ et $\mathcal{A}$ choisit deux vecteurs distincts y_0, $y_1 \in \mathbb{Z}_q^l$. Il obtient de $\mathcal{B}$ ensuite un chiffré C_{y_β} de $y_\beta = (y_{\beta,1}, ..., y_{\beta,l})$, pour $\beta \xleftarrow{\$} \{0,1\}$.
A la fin du jeu, $\mathcal{A}$ renvoie $\beta^{'} \in \{0,1\}$. Donc $S_1 = 1$ si et seulement si $\beta' = \beta$.
Pour tout vecteur $x \in \mathbb{Z}_q^l$ soumis à l'oracle d'extraction de clé secrète, x doit vérifier $< x, y_0 >=< x, y_1 > \mod q$.

Jeu 1 :

Dans ce jeu, on modifie la génération du challenge $C_{y_\beta} = (C, D, E_1, \cdots, E_l)$. À savoir, le challenger $\mathcal{B}$ calcule d'abord :

$$C = g^r \ et \ D = h^r$$

pour un $r \xleftarrow{\$} \mathbb{Z}_q$ pris aléatoirement. Ensuite, il utilise $msk := \{(s_i, t_i)\}_{i=1}^l$ pour calculer

$$E_i = g^{y_{\beta,i}} C^{s_i} D^{t_i}$$

On peut observer que $C_{y_\beta} = (C, D, E_1, \cdots, E_l)$ a la même distribution que dans le jeu 0, car on a : $\{C^{s_i} D^{t_i} \mid r \xleftarrow{\$} \mathbb{Z}_q\} = \{(g^r)^{s_i} (h^r)^{t_i} \mid r \xleftarrow{\$} \mathbb{Z}_q\} = \{(g^{s_i} h^{t_i})^r \mid r \xleftarrow{\$} \mathbb{Z}_q\} = \{h_i^r \mid r \xleftarrow{\$} \mathbb{Z}_q\}$.
On a donc $Pr[S_1] = Pr[S_0]$.

Jeu 2 :

Dans ce jeu, la génération du chiffré challenge est encore modifiée. Plus spécifiquement, $\mathcal{B}$ prend cette fois deux aléatoires $r, r' \in \mathbb{Z}_q$ et calcule $C = g^r$ et $D = h^{r+r'}$. Les éléments E_i sont calculés comme dans le jeu 1 soit, pour tout $i \in \{1, ..., l\}$, $E_i = g^{y_{\beta,i}} C^{s_i} D^{t_i}$.

La différence de probabilités $|Pr[S_2] - Pr[S_1]|$ de succès de l'attaquant $\mathcal{A}$ entre le jeu 1 et le jeu 2 est négligeable.
En effet, le seul changement entre les jeux 1 et 2 réside dans le fait que (h, C, D) est un triplet DDH dans le jeu 1 et un triplet aléatoire dans le jeu 2. Ainsi, on peut construire un algorithme $\mathcal{C}$ qui prend en entrée un triplet (X, Y, Z) et qui attaque DDH en utilisant $\mathcal{A}$. Pour cela, il suffit de prendre : $(h, C, D) = (X, Y, Z)$ et de construire un chiffré challenge comme dans les jeux 1 et 2. Plus spécifiquement, $\mathcal{C}$ crée deux clefs mpk et msk, demande deux messages clairs y_0, y_1 à $\mathcal{A}$, tire β aléatoire dans $\{0,1\}$ et fabrique un chiffré de y_β comme ce qui suit. $\mathcal{C}$ pose $C = Y$, $D = Z$ et calcule pour tout $i \in \{1, ..., l\}$, $E_i = X^{y_{\beta,i}} C^{s_i} D^{t_i}$. Le chiffré challenge est alors : $C_{y\beta} = (C, D, E_1, \cdots, E_l)$, comme précédemment. $\mathcal{A}$ répond β' et $\mathcal{C}$ retourne 1 si $\beta = \beta'$ et 0 sinon.
Dans ce cas, $Adv_{\mathcal{C}}^{DDH}(\lambda) = |Pr(\text{C retourne 1 et le triplet est un triplet DDH}) -$

$Pr(\text{C retourne 1 et le triplet est un triplet aléatoire})| = |Pr[S_1] - Pr[S_2]|$.
Ainsi, comme sous l'hypothèse DDH $Adv_{\mathcal{C}}^{DDH}(\lambda)$ est négligeable, $|Pr[S_2] - Pr[S_1]|$ est aussi négligeable.

Montrons désormais que $Pr[S_2] = \frac{1}{2}$. Pour cela, montrons que $\beta \in \{0,1\}$ est totalement indistinguable dans le chiffré challenge $C_{y_\beta} = (C, D, \{E_i\}_i)$. On remarque tout d'abord que pour tout $i \in \{1, ..., l\}$, $E_i = g^{y_{\beta,i}} C^{s_i} D^{t_i} = g^{y_{\beta,i} + \omega r' t_i} h_i^r$ où $\omega = log_g(h)$. Donc $\mathcal{A}$ peut reconnaître un chiffré de $z_\beta = (y_{\beta,1} + \omega \cdot r' \cdot t_1, ..., y_{\beta,l} + \omega \cdot r' \cdot t_l) = y_\beta + \omega \cdot r' \cdot t \in \mathbb{Z}_q^l$.

Il suffit donc de montrer que z_β ne révèle rien sur β. Pour cela, on pose $y = y_0 - y_1$ la différence entre les deux messages que $\mathcal{A}$ donne au début de l'expérience et on prend une base de $y^\perp = \{x \in \mathbb{Z}_q^l | < x, y >= 0 \bmod q \}$. On pose alors $X_{top} \in \mathbb{Z}_q^{(l-1)\times l}$, la matrice dont les lignes sont les vecteurs de cette base. Soit $y' \in \mathbb{Z}_q^l \backslash y^\perp$. On construit alors X la matrice suivante :

$$X = \left[\frac{X_{top}}{y'^T} \right] \in \mathbb{Z}_q^{l \times l}$$

Par construction, les lignes de X forment alors une base de $\mathbb{Z}_q^l$. Celles-ci sont créées de manière déterministe à partir de y auquel $\mathcal{A}$ a accès, $\mathcal{A}$ connaît donc X. Ainsi, le fait que z_β ne divulgue aucune information sur β est équivalent au fait que $X \cdot z_\beta$ ne divulgue aucune information sur β.

Par construction, $X_{top} \cdot y = 0_{\mathbb{Z}^l}$ donc $X_{top} \cdot y_0 = X_{top} \cdot y_1$ et $X_{top} \cdot z_\beta$ est alors indépendant de β et n'en révèle aucune information. La dernière ligne de $X_{top} \cdot z_\beta$ est la suivante, dans $\mathbb{Z}_q$:

$$< y', z_\beta >=< y', y_\beta + \omega \cdot r' \cdot t >=< y', u > + \omega \cdot r' \cdot < y', t >$$

L'idée est alors de montrer que le deuxième terme masque complètement le premier grâce à son uniformité.

Soit une clé maître privée $msk_0 = (s_0, t_0) = ((s_{0,1}, ..., s_{0,l}), (t_{0,1}, ..., t_{0,l})) \in \mathbb{Z}_q^l \cdot \mathbb{Z}_q^l$. Soit une requête de clé privée, le vecteur x. On a alors $sk_x = (< s_0, x > , < t_0, x >)$. Construisons l'ensemble de tous les couples (s, t) pour lesquels on a des clés secrètes égales à sk_x (toujours pour la requête x). En réalité, on s'intéresse principalement à t (s dépendra de t) :
$x \in y^\perp$ donc $< x, y >= 0$ et on a $< t, y >=< t_0, y > + \mu < x, y >$ pour tout μ dans $\mathbb{Z}_q$ donc on obtient l'ensemble $\{t \in \mathbb{Z}_q^l \mid t = t_0 + \mu y \bmod q\}$.

$$\begin{aligned} \text{Ainsi, } \omega r' t < y', t > \bmod q &= \omega r' < y', t_0 + \mu y > \bmod q \\ &= \omega r' (< y', t_0 > + \mu < y', y >) \bmod q \end{aligned}$$

Par construction, $< y', y > \neq 0$ donc la distribution de $\mu < y', y >$ est uniforme. r' est choisi uniformément dans $\mathbb{Z}_q$ donc il est non nul avec probabilité

valant $1 - \frac{1}{q}$. Finalement, $\omega \cdot r' \cdot t < y', t >$ mod q est donc uniforme.
Donc $Pr(S_2) = \frac{1}{2}$.

En cumulant les majorations obtenues dans les différents jeux, on obtient : $Adv_{\mathcal{A}}^{IND-FE-CPA} = |Pr(S_0) - 1/2| = |Pr(S_1) - 1/2|$ donc
$Adv_{\mathcal{A}}^{IND-FE-CPA} = |Pr(S_1) - Pr(S_2)| \leq Adv_{\mathcal{C}}^{DDH}(\lambda)$ qui est négligeable sous l'hypothèse DDH.

Donc $Adv_{\mathcal{A}}^{IND-FE-CPA}$ est négligeable.

Le schéma décrit dans cette partie propose donc une construction IND-FE-CPA basée sur DDH. Sa limite se situe au niveau du déchiffrement puisqu'en effet, il nécessite le calcul coûteux d'un logarithme discret. Dans la partie suivante, nous présentons une construction basée sur le chiffrement de Paillier, toujours IND-FE-CPA, qui ne présente pas une limitation de ce type.

5 Construction d'un schéma FE-IP basée sur le chiffrement de Paillier

Dans cette partie, nous étudions en termes de construction et de sécurité, un schéma basé sur le chiffrement de Paillier proposé par **[3]**.

5.1 Schéma

Pour ce schéma, nous posons X, Y, N des entiers tels que $XY < N$ où N est un module RSA. Pour de tels paramètres X et Y, les messages y et les clés x vérifieront les critères : $||x||_\infty < X$, $||y||_\infty < Y$ et $X, Y < (N/l)^{\frac{1}{2}}$.

Nous décrivons les quatre algorithmes usuels, de la manière suivante :

1. $Setup(1^\lambda, 1^l, X, Y)$ choisit aléatoirement p', q' deux premiers strictement supérieurs à 2^λ tels que $p = 2p' + 1$ et $q = 2q' + 1$ soient premiers. Il pose $N = pq$ et $g = g'^{(2N)} \bmod N^2$ où g' est un élément tiré aléatoirement dans $\mathbb{Z}^*_{N^2}$. Il choisit un vecteur $s = (s_1, ..., s_l)$ dans la distribution gaussienne $D_{\mathbb{Z}^l,\sigma}$ où $\sigma > \sqrt{\lambda}.N^{\frac{5}{2}}$ et pose $h = (h_1, ..., h_l) = (g^{s_1}, ..., g^{s_l})$. Il retourne $mpk = (N, g, \{h_i\}_{i=1}^l, Y)$ et $msk = (\{s_i\}_{i=1}^l, X)$.
2. $KeyDer(msk, x)$ retourne $sk_x =< s, x >$.
3. $Encrypt(mpk, y)$ choisit r aléatoirement dans $\{0, ..., \lfloor N/4 \rfloor\}$ et construit $C_0 = g^r \bmod N^2$ et $\{C_i = (1 + y_i N)h_i^r \bmod N^2\}_{i=1}^l$. Il retourne $C_y = (C_0, C_1, ..., C_l)$.
4. $Decrypt(mpk, sk_x, C_y)$ calcule $C_x = C_0^{-sk_x} \prod_{i=1}^l C_i^{x_i}$ et retourne $< x, y >= \log_{(1+N)}(C_x) = \frac{C_x - 1 \bmod N^2}{N}$.

Pour vérifier l'égalité de l'algorithme *Decrypt*, nous utilisons la propriété suivante :

$$\begin{aligned}(1+N)^t \bmod N^2 &= \textstyle\sum_{k=0}^t \binom{t}{k}(1^{t-k}N^k \bmod N^2) \\ &= \binom{t}{0}N^0 + \binom{t}{1}N^1 + \textstyle\sum_{k=2}^t (1^{t-k}N^k \bmod N^2) \bmod N^2 \\ &= (1 + tN) \bmod N^2 \\ (1+N)^t \bmod N^2 &= (1 + tN) \bmod N^2\end{aligned}$$

Le schéma est correct puisque l'on a :

$$\begin{aligned}C_x &= C_0^{-sk_x} \textstyle\prod_{i=1}^l C_i^{x_i} = \frac{1}{g^{r<s,x>}} \prod_{i=1}^l (1 + y_i N)^{x_i} h_i^{rx_i} \bmod N^2 \\ &= \frac{1}{g^{r<s,x>}} \textstyle\prod_{i=1}^l (1+N)^{x_i y_i} g^{r x_i y_i} \bmod N^2 \\ &= \frac{1}{g^{r \sum_{i=1}^l x_i y_i}} (1+N)^{\sum_{i=1}^l x_i y_i} g^{r \sum_{i=1}^l x_i y_i} \bmod N^2 \\ C_x &= (1+N)^{<x,y>} \bmod N^2\end{aligned}$$

d'où $< x, y >= \log_{(1+N)}(C_x)$ ou encore :

$C_x = (1+N)^{<x,y>} = (1+ < x, y > N)$ donc $< x, y >= \frac{C_x - 1 \bmod N^2}{N}$

5.2 Sécurité

Dans cette section, nous montrons que le schéma précédent est IND-FE-CPA. Pour cela, nous prenons un adversaire $\mathcal{A}$ contre la sécurité IND-FE-CPA et nous utilisons une séquence de jeux permettant d'obtenir une majoration, négligeable, de l'avantage de $\mathcal{A}$.

Jeu 0 :
C'est le jeu original de sécurité IND-FE-CPA.
On donne à l'adversaire $\mathcal{A}$ la clé maître publique $mpk = (N, g, \{h_i\}_{i=1}^{l}, Y)$.
Dans la phase de challenge, $\mathcal{A}$ choisit deux vecteurs y_0 et y_1 dans $\mathbb{Z}^l$ et obtient de $\mathcal{B}$ un chiffré C_{y_β} de y_β pour un aléatoire $\beta \in \{0, 1\}$. Puisque r est tiré uniformément aléatoirement dans $\{0, ..., \lfloor N/4 \rfloor\}$, certains éléments de $\mathbb{Z}_{N^2}^*$ ont $\frac{1}{1+\lfloor \frac{N}{4} \rfloor}$ chances d'être tiré, et certains 0. Ainsi, la distribution des C_0 est une distribution statistique.
A la fin de ce challenge, $\mathcal{A}$ sort $\beta' \in \{0, 1\}$.
Pour tout vecteur $x \in \mathbb{Z}^l$ soumis à l'oracle d'extraction de clé secrète, x doit vérifier $< x, y_0 >=< x, y_1 >$ dans $\mathbb{Z}$.

Jeu 1 :
Dans cette expérience, le chiffré challenge $C_{y_\beta} = (C_0, C_1, ..., C_l)$ est modifié par rapport au jeu 0.

Pour générer C_0 ici, $\mathcal{B}$ prend un z_0 aléatoire dans $\mathbb{Z}_N^*$ puis pose $C_0 = z_0^{2N} \bmod N^2$.
Pour générer les C_i ensuite, $\mathcal{B}$ utilise la clef maître $msk = (\{s_i\}_i, X)$. Pour tout $i \in \{1, ..., l\}$, il pose $C_i = (1+N)^{y_{\beta,i}} C_0^{s_i}$.

Le chiffré C_{y_β} a pratiquement la même distribution que dans le jeu 0, la seule différence est que C_0 est maintenant distribué de manière parfaitement uniforme (et non plus de manière statistique) dans le sous-groupe des résidus $N^{\text{ème}}$ modulo N^2. La différence entre les deux distributions est négligeable.
On a donc $|Pr(S_1) - Pr(S_0)| \leq 2^{-\lambda}$.

Jeu 2 :
Dans cette expérience, le chiffré challenge est à nouveau modifié. Au lieu de former C_0 comme dans le jeu 1, le challenger choisi z aléatoire dans $\mathbb{Z}_{N^2}^*$ et pose $C_0 = z^2 \bmod N^2$. Le reste des étapes est le même. La différence entre les jeux 1 et 2 est alors que dans le premier cas C_0 est un $N^{\text{ème}}$ résidu modulo N^2 et que dans le deuxième cas ce n'en est pas (C_0 est juste un carré dans $\mathbb{Z}_{N^2}^*$).

La différence de probabilités $|Pr[S_2] - Pr[S_1]|$ de succès de l'attaquant $\mathcal{A}$ entre le jeu 1 et le jeu 2 est négligeable.
En effet, le seul changement entre les jeux 1 et 2 est dans le fait que z est un $N^{\text{ème}}$ résidu modulo N^2 dans le jeu 1 et un carré aléatoire dans le jeu 2. Ainsi, on peut construire un algorithme $\mathcal{C}$ qui prend en entrée élément ζ de $\mathbb{Z}^*_{N^2}$ et qui attaque DCR en utilisant $\mathcal{A}$. Pour cela, il suffit de prendre : $z = \zeta$ et de construire un chiffré challenge comme dans les jeux 1 et 2. Plus spécifiquement, $\mathcal{C}$ crée deux clefs mpk et msk, demande deux messages clairs y_0, y_1 à $\mathcal{A}$, tire β aléatoire dans $\{0,1\}$ et fabrique un chiffré de y_β comme ce qui suit. $\mathcal{C}$ pose $C_0 = z$ et calcule pour tout $i \in \{1, ..., l\}$, $C_i = (1+N)^{y_{\beta,i}} \cdot C_0^{s_i}$. Le chiffré challenge est alors : $C_{y_\beta} = (C_0, C_1, \cdots, C_l)$, comme précédemment. $\mathcal{A}$ répond β' et $\mathcal{C}$ retourne 1 si $\beta = \beta'$ et 0 sinon.
Dans ce cas, $Adv_{\mathcal{C}}^{DCR}(\lambda) = |Pr(\text{C retourne 1 et z est un } N^{\text{ème}} \text{ résidu modulo } N^2) - Pr(\text{C retourne 1 et z est un carré aléatoire})| = |Pr[S_1] - Pr[S_2]|$.
Ainsi, comme sous l'hypothèse DCR, $Adv_{\mathcal{C}}^{DCR}(\lambda)$ est négligeable, $|Pr[S_2] - Pr[S_1]|$ est aussi négligeable.

Jeu 3 :

Dans le chiffré challenge, on modifie la génération de C_0. Le challenger choisit $a_z \xleftarrow{\$} \mathbb{Z}^*_{N^2}$ et $r_z \xleftarrow{\$} \{1, ..., \lfloor \frac{N}{4} \rfloor\}$ et $\mathcal{B}$ calcule :

$$C_0 = (1+N)^{a_z} \cdot g^{r_z} \bmod N^2$$

$\{C_i\}_{i=1}^l$ reste inchangé. Ainsi, le choix de C_0 n'est plus uniforme dans l'ensemble des carrés de $\mathbb{Z}^*_{N^2}$ mais est de la forme donnée précédemment. Du point de vue de $\mathcal{A}$, la différence est donc indistinguable et $|Pr[S_3] - Pr[S_2]|$ est négligeable, soit $|Pr[S_3] - Pr[S_2]| \leq 2^{-\lambda}$.

On a donc $C_0 = (1+N)^{a_z} \cdot g^{r_z} \bmod N^2$ et on a : $(C_1, ..., C_l) = ((1+N)^{y_{\beta,1} + a_z \cdot s_1 \bmod N} \cdot h_1^{r_z} \bmod N^2, ..., (1+N)^{y_{\beta,l} + a_z \cdot s_l \bmod N} \cdot h_l^{r_z} \bmod N^2)$

On va montrer que $\beta \in \{0,1\}$ est statistiquement caché par cette quantité, et que plus précisément on a $|Pr[S_3] - \frac{1}{2}| \leq 2^{-\lambda}$.
On considère le vecteur $y = (y_1, ..., y_l) = \frac{1}{g}(y_1 - y_0) \in \mathbb{Z}^l$ avec $g = pgcd(y_{0,1} - y_{1,1}, ..., y_{0,l} - y_{1,l})$. On sait que toutes les requêtes de clés $x \in \mathbb{Z}^l$ sont dans le réseau $\{x \in \mathbb{Z}^l \ : < x, y >= 0\}$. Soit $X_{top} \in \mathbb{Z}^{(l-1)\times l}$ la matrice dont les lignes forment une base de ce réseau. On sait aussi que $\|y\|^2 < N$ (car $||y||^2 < (2Y)^2 < (XY)^2 < \sqrt{N}^2 = N$) et on peut supposer que $pgcd(\|y\|^2, N) = 1$ puisque la réduction permettrait de calculer un facteur non trivial de N autrement.
On définit la matrice :

$$X = \begin{bmatrix} X_{top} \\ y^T \end{bmatrix}$$

On admet que X est inversible.

$\mathcal{A}$ peut reconnaître un chiffré de $z_\beta = (y_{\beta,1} + a_z \cdot s_1 \bmod N, ..., y_{\beta,l} + a_z \cdot s_l \bmod N)^T \in \mathbb{Z}_N^l = y_\beta + a_z \cdot s \bmod N$. Il suffit donc de montrer que z_β ne révèle rien sur β. Comme X est inversible mod N et ne dépend pas de β, il suffit de montrer que $X.z_\beta$ ne révèle rien sur β.

On a tout d'abord que $X_{top} \cdot y_0 = X_{top} \cdot y_1$ par définition de X_{top} donc $X_{top} \cdot z_\beta = X_{top}(y_\beta + a_z \cdot s) \in \mathbb{Z}_N^{l-1}$ est indépendant de β.

Il reste donc à montrer que pour la dernière ligne de X, le produit $X.z_\beta$ est bien indépendant de β i.e. que $y^T \cdot z_\beta \bmod N =< y_\beta, y > + a_z < s, y > \bmod N$ ne révèle rien sur β. L'idée est de montrer que le deuxième terme masque complètement le premier grâce à son uniformité. Déterminons donc la distribution de $< s, y > \bmod N$.

Soit un vecteur $s_0 = (s_{0,1}, ..., s_{0,l}) \in \mathbb{Z}^l$ représentant une clef maître secrète quelconque vérifiant : $X_{top} \cdot s_0 = X_{top} \cdot s$ et $h_i = g^{s_{0,i}} \bmod N^2$ pour tout $i \in \{1, ..., l\}$. On détermine à partir de s_0 la distribution de s.

La distribution de la clef secrète maître s est $s_0 + D_{\Lambda,\sigma,-s_0}$ avec $\Lambda = \{t \in \mathbb{Z}^l | X_{top} \cdot t = 0 \in \mathbb{Z}^{l-1}, t = 0 \bmod p'q'\}$. La distribution gaussienne discrète $D_{\Lambda,\sigma,-s_0}$ se justifie par le fait que les clefs maîtres secrètes sont tirées avec une loi gaussienne centrée de paramètre σ. De plus, les vecteurs t composants le réseau Λ sont tels que $X_{top} \cdot (s_0 + t) = X_{top} \cdot s$ et donc sachant que $X_{top} \cdot s_0 = X_{top} \cdot s$ par hypothèse, on a donc $X_{top} \cdot t = 0$. De plus, pour tout $i \in \{1, ..., l\}$, $g^{s_i} = g^{s_{0,i}+t_i} \bmod N^2$ et comme on a aussi, $g^{s_i} = g^{s_{0,i}} \bmod N^2$, on a donc pour tout i, $t_i = 0 \bmod ord(g) = p'q'$. Donc $t = 0 \bmod p'q'$.
Ce réseau se réécrit : $\Lambda = \mathbb{Z} \cdot y \cap (p'q' \cdot \mathbb{Z})^l$ puisque X_{top} représente la base des vecteurs orthogonaux à y, soit $\Lambda = (p'q') \cdot \mathbb{Z} \cdot y$. Conditionnellement aux $\{h_i\}_i$ et à $X_{top} \cdot s$, on admet que la distribution de $< s, y >$ est donc : $< s_0, y > + D_{(p'q')||y||^2.\mathbb{Z},||y||\sigma,-<s_0,y>}$.

Pour déterminer la distribution de $< s, y > \bmod N$, on réduit la distribution obtenue $D_{(p'q')||y||^2.\mathbb{Z},||y||\sigma,-<s_0,y>}$ sur le réseau $\Lambda_0 = (p'q')||y||^2.\mathbb{Z}$ modulo le sous-réseau $\Lambda_0' = (p'q')||y||^2.(N\mathbb{Z})$. On admet que la distribution de $< s, y > \bmod N$ a une distance de $2^{-\lambda}$ de la distribution uniforme sur $\Lambda_0/\Lambda_0' \simeq \mathbb{Z}_N$ lorsque les $\{h_i\}_i$ et $X_{top}.s$ sont donnés.

Ainsi, comme a_z est fixé et non nul, la distribution de $a_z \cdot < s, y > \bmod N$ est proche de l'uniforme et masque donc complètement le premier terme $< y_\beta, y > \bmod N$.

$X.z_\beta$ et par conséquent z_β sont donc statistiquement indépendants de β.

En cumulant les majorations obtenues dans les différents jeux, on obtient : $Adv_{\mathcal{A}}^{IND-FE-CPA} = |Pr(S_0) - 1/2| \leq Adv_{\mathcal{C}}^{DCR}(\lambda) + 2^{-\lambda+1}$ qui est négligeable sous l'hypothèse DCR.

Dans cette partie, nous avons étudié le schéma FE-IP basé sur le chiffrement de Paillier proposé par [**3**], totalement IND-FE-CPA et ne présentant pas de limitation dans les calculs. En effet, deux calculs sont possibles au sein de l'algorithme *Decrypt* de ce schéma : un calcul modulaire et un reposant sur le logarithme discret.

6 Conclusion

L'objectif de ce projet était d'étudier différents schémas de chiffrement fonctionnel pour le produit scalaire. Pour cela, nous nous sommes surtout focalisés sur l'analyse de schémas construits à partir de différents chiffrements PKE existants, notamment ceux de ElGamal et Paillier.

Néanmoins, ces schémas possèdent quelques inconvénients. Dans la plupart des constructions présentées dans ce rapport, l'algorithme *Decrypt* fait appel au logarithme discret. Il est alors nécessaire, pour des soucis d'efficacité et de rapidité calculatoire, d'établir certaines conditions sur le produit scalaire, et par conséquent sur l'ensemble des messages ainsi que celui des clés.

Deux pistes d'amélioration sont alors possibles. D'un côté, on peut s'interroger sur les paramètres de la construction pour permettre une plus grande liberté sur les espaces restreints. Dans le cas de la construction basée sur DDH (3.4.2 et 4.1), on peut par exemple choisir un élément q plus grand pour augmenter B de sorte que la condition $lB^2 < q$ soit vérifiée. Ceci consiste cependant à repousser le problème plutôt que de le résoudre.

D'un autre côté, on peut étudier d'autres constructions, ne faisant pas appel au logarithme discret ou à d'autres problèmes difficiles lors de l'algorithme *Decrypt*. La construction réalisée à partir du chiffrement de Paillier permet de contourner cet obstacle en retrouvant la valeur du logarithme sans le calculer directement. L'article **[2]** présente une construction PKE-IP à partir du **schéma de Regev** dont la sécurité est basée sur l'hypothèse LWE (Learning With Errors). Celle-ci suppose que, pour deux paramètres n entier et q premier, un vecteur secret $s \in \mathbb{Z}_q^n$, une distribution de bruit $\chi \in \mathbb{Z}_q^n$, il est difficile de retrouver s à partir de couples $(a, <a, s> + e)$ connus, où e est choisi aléatoirement dans χ. Ce schéma utilise, au sein de *Decrypt*, un calcul modulaire ne nécessitant pas de condition particulière sur l'espace des messages quant à son efficacité calculatoire.

Références

[1] Boneh D., Sahai A., Waters B. (2011) *Functional Encryption : Definitions and Challenges*. In : Ishai Y. (eds) Theory of Cryptography. TCC 2011. Lecture Notes in Computer Science, vol 6597. Springer, Berlin, Heidelberg. https ://doi.org/10.1007/978-3-642-19571-6_16

[2] Abdalla M., Bourse F., De Caro A., Pointcheval D. (2015) *Simple Functional Encryption Schemes for Inner Products*. In : Katz J. (eds) Public-Key Cryptography – PKC 2015. PKC 2015. Lecture Notes in Computer Science, vol 9020. Springer, Berlin, Heidelberg. https ://doi.org/10.1007/978-3-662-46447-2_33

[3] Agrawal S., Libert B., Stehlé D. (2016) *Fully Secure Functional Encryption for Inner Products, from Standard Assumptions*. In : Robshaw M., Katz J. (eds) Advances in Cryptology – CRYPTO 2016. CRYPTO 2016. Lecture Notes in Computer Science, vol 9816. Springer, Berlin, Heidelberg. https ://doi.org/10.1007/978-3-662-53015-3_12

7 Annexe

GroupGen

La fonction suivante implémente *GroupGen* qui retourne un groupe cyclique G d'ordre p premier de K bits (pour K entrée de la fonction) et g générateur de G. *GroupGen* est appelée dans les implémentations basées sur DDH.

```
def GroupGen(K):
# retourne G, g, p : G groupe cyclique d'ordre p premier de K bits et de générateur g

    p = random_prime(2^K, lbound = 2^(K-1))
    G = GF(p)
    g = G.random_element()

    while g == G(0) or g.multiplicative_order() != p-1:
        g = G.random_element()

    return(G, p, g)
```

Construction DDH-IP

L'implémentation suivante représente le schéma PKE-IP appliqué à ElGamal de l'article [**2**], détaillé dans la partie 3.4 de ce rapport.

```
load("GroupGen.sage")

print("Implémentation du schéma basé sur DDH du 2e article")

## Setup

def Setup(K, l):
# retourne une clef maître secrète msk et une clef maître publique mpk
# correspondantes aux paramètres K et l

    G, p, g = GroupGen(K)
    msk = []
    mpk = []
    pub = (G, p, g, l) # informations publiques

    for _ in range(l):
        s = randint(0,p)
        msk.append(s)
        mpk.append(g^s)

    return(pub, mpk, msk)
```

```
## KeyDer

def KeyDer(msk, y):
# retourne la clef secrète dérivée de y par la clef maître secrète msk

    return vector(msk).inner_product(vector(y))

## Encrypt

def Encrypt(pub, mpk, x):
# retourne un chiffré de x par la clé maître publique mpk

    G, p, g, l = pub
    r = randint(0, p)

    Ct = [] # chiffré
    Ct.append(g^r) # ct0

    for i in range(l):
        Ct.append((mpk[i]^r)*(g^x[i])) # cti

    return(Ct)


## Decrypt

def Decrypt(pub, mpk, Ct, sky, y):
# retourne le déchiffré de Ct par la clef secrète dérivée sky

    G, p, g, l = pub

    produit = 1
    for i in range(l):
        produit = produit*(Ct[i+1]^y[i])

    return(log((produit/(Ct[0]^sky)),g))

K = 50
l = 20
pub, mpk, msk = Setup(K, l)
G, p, g, l = pub

B = floor(sqrt(p/l)) # calcul de la borne B telle que l*B^2<p
```

```
x = [randint(0, B) for _ in range(l)]
Ct = Encrypt(pub, mpk, x)
y = [randint(0, p) for _ in range(l)]
sky = KeyDer(msk, y)
x_y = Decrypt(pub, mpk, Ct, sky, y)

print(x_y, "correct ?", x_y == (vector(x).inner_product(vector(y)))%(p-1))
```

Construction DDH

L'implémentation suivante représente le schéma basé sur DDH de l'article [**3**], détaillé dans la partie 4.1 de ce rapport.

```
load("GroupGen.sage")

print("Implémentation du schéma basé sur DDH du 3e article")

## Setup

def Setup(l, K):
# retourne une clef maître secrète msk et une clef maître publique mpk
# correspondantes aux paramètres K et l

    G, q, g = GroupGen(K)
    Zq = IntegerModRing(q)
    msk = []
    mpk = []

    while True:
        g = G.random_element()
        if (g != G(0)) and g.multiplicative_order() == q-1: break
    while True:
        h = G.random_element()
        if (h != G(0)) and h.multiplicative_order() == q-1: break

    for i in range(l):
        (s, t) = (Zq.random_element(), Zq.random_element())
        msk.append((s, t))
        mpk.append(g^s*h^t)

    pub = (G, q, g, h, l) # informations publiques

    return(pub, mpk, msk)


## KeyDer
```

```
def KeyDer(msk, y):
# retourne la clef secrète dérivée de y par la clef maître secrète msk

    s = vector([ZZ(i[0]) for i in msk])
    t = vector([ZZ(i[1]) for i in msk])

    sy = vector(y).inner_product(s)
    ty = vector(y).inner_product(t)

    sky = y, sy, ty

    return(sky)

## Encrypt

def Encrypt(pub, mpk, x):
# retourne un chiffré de x par la clé maître publique mpk

    # definition du contexte
    G, p, g, h, l = pub
    Zq = IntegerModRing(G.order())

    # chiffrement
    r = Zq.random_element()
    C = g^r; D = h^r; E = []
    for i in range(l):
        E.append(g^x[i]*mpk[i]^r)
    Ct = [C]+[D]+E

    return(Ct)


## Decrypt

def Decrypt(pub, mpk, sky, Ct):
# retourne le déchiffré de Ct par la clef secrète dérivée sky

    # mise du contexte
    G, p, g, h, l = pub
    y,sy,ty = sky
    C,D,E = Ct[0],Ct[1],Ct[2:]
    Ey = 1

    # dechiffrement
    for i in range(l):
```

```
        Ey *= E[i]^y[i]
    Ey /= C^(sy)*D^(ty)

    return log(Ey,g)


K = 50; l = 20
pub, mpk, msk = Setup(K, l)
G, p, g, h, l = pub

x = [randint(0, p) for _ in range(l)]
Ct = Encrypt(pub, mpk, x)
y = [randint(0, p) for _ in range(l)]
sky = KeyDer(msk, y)
x_y = Decrypt(pub, mpk, sky, Ct)

print(x_y, "correct ?", x_y == (vector(x).inner_product(vector(y)))%(p-1))
```

Construction Paillier

L'implémentation suivante représente le schéma basé sur le chiffement de Paillier de l'article [**3**], détaillé dans la partie 5.1 de ce rapport.

```
from sage.stats.distributions.discrete_gaussian_integer \
 import DiscreteGaussianDistributionIntegerSampler

print("Implémentation du schéma basé sur Paillier")

## Setup

def Setup(K, l, X, Y):
# retourne une clef maître secrète msk et une clef maître publique mpk
# correspondantes aux paramètres K, l, X et Y

    # pprime : premier nombre premier aléatoire
    pprime = random_prime(2**K,lbound = 2**(K-1), proof=false)


    p = 2*pprime+1

    while (not(p.is_prime())): # on veut p premier
        pprime = random_prime(2**K,lbound = 2**(K-1), proof=false)
        p = 2*pprime+1

    # qprime : premier nombre premier aléatoire
    qprime = random_prime(2**K,lbound = 2**(K-1), proof=false)
```

```
    q = 2*qprime+1

    while (not(q.is_prime())): # on veut q premier
        qprime = random_prime(2**K,lbound = 2**(K-1), proof=false)
        q = 2*qprime+1

    N = p*q # N est le module composite de Paillier

    N2 = N**2 # N2 = N**2

    gprime = randint(1,N2)

    while gcd(gprime,N2)!=1:
        gprime = randint(1,N2) # gprime aléatoire dans (Z/N2Z)*

    g = pow(gprime,2*N,N2) # calcul de g

    # calcul de sigma tel que sigma > sqrt(landa)*(N)**(5/2)
    sigma = integer_ceil(sqrt(K)*(N)**(5/2)) + 1


    D = DiscreteGaussianDistributionIntegerSampler(sigma=sigma)

    s = [D() for _ in range(l)] # s suit une distribution gausienne

    h = [g**(s[i]) for i in range(l)]

    mpk = (N, g, h, Y)
    msk = (s, X)

    return (mpk, msk)


## KeyDer

def KeyDer(msk, y):
# retourne la clef secrète dérivée de y par la clef maître secrète msk

    sky = vector(msk[0]).inner_product(vector(y))
    return sky

## Encrypt

def Encrypt(mpk, x):
```

```
# retourne un chiffré de x par la clé maître publique mpk

    N = mpk[0]
    g = mpk[1]
    h = mpk[2]
    N2 = N**2

    Nfloor = integer_floor(N/4)
    r = randint(0, Nfloor) # r aléatoire entre 0 et N/4

    C0 = pow(g,r,N2) # C0
    C = [mod((1+x[i]*N)*h[i]**r, N2) for i in range(len(x))] # reste de Ct
    Ct = [C0]+C

    return Ct

## Decrypt

def Decrypt(mpk, sky, Ct, y):
# retourne le déchiffré de Ct par la clef secrète dérivée sky

    N = mpk[0]
    N2 = N**2
    C0 = Ct[0]

    produit = 1;
    for i in range(l):
        produit = produit*(Ct[i+1]^y[i]);
    Cy = mod(C0**(-sky)*produit, N2)

    return log(Cy,1+N)


def test_paillier(K, l, X, Y, a):
# permet de faire des tests sur les algorithmes implémentés

    (N, g, h, Y),(s, X)= Setup(K, l, X, Y)
    x = [randint(0, a) for _ in range(l)]
    skx = KeyDer((s, X), x)
    y = [randint(0, a) for _ in range(l)]
    Cy = Encrypt((N, g, h, Y), y)
    print("correct ?", Decrypt((N, g, h, Y), skx, Cy,x) \
== vector(x).inner_product(vector(y)))
    return (Decrypt((N, g, h, Y), skx, Cy,x), vector(x).inner_product(vector(y)))

'''
```

```
K = 25
l = 10
X = 100
Y = 100
a = 10
test_paillier(K, l, X, Y, a)
'''

K = 50
l = 20
X = 100
Y = 100
a = 20
test_paillier(K, l, X, Y, a)

'''
K = 100
l = 40
X = 1000
Y = 1000
a = 50
test_paillier(K, l, X, Y, a)
'''
```