

Cryptographie post-quantique à base de réseaux

Une implémentation d'un *Ciphertext-Policy Attribute-Based Encryption*

Quentin BODINI--LEFRANC
Théodore HALLEY

Damya BOUIZEGARENE
Manan KAILA

Rémi GERME

PSC INF04

avril 2024

Résumé

Dans le cadre de notre *projet scientifique collectif* (PSC), nous avons cherché à étudier et à implémenter un schéma de *chiffrement par attributs*. Il s'agit d'un protocole cryptographique permettant de mettre en place une politique de contrôle d'accès sur les données. Dès lors, il n'est possible pour un utilisateur de déchiffrer le message que s'il a les *attributs* nécessaires.

On propose dans un premier temps une étude détaillée du schéma considéré introduit par Z. BRAKERSKI et V. VAIKUNTANATHAN [BV22], avant d'aborder plus en détail l'implémentation que l'on en propose. L'apport de notre travail est avant tout de fournir une implémentation fidèle de ce schéma, jusqu'alors jamais réalisée.

Remerciements

Avant de commencer ce rapport, nous tenons à adresser nos remerciements à toutes les personnes qui nous ont aidés de près ou de loin pour notre PSC.

Nous voulons tout d'abord remercier Madame Adeline ROUX-LANGLOIS pour avoir accepté d'être notre tutrice de projet pendant toute cette année. Ses conseils nous ont souvent permis d'avancer dans les moments clefs du projet. Nous souhaitons également remercier Monsieur Lucas PRABEL qui a mis à notre disposition tout le travail réalisé pour son mémoire sur les KP-ABE, ce qui nous a notamment permis de nous lancer dans l'implémentation plus rapidement que si nous étions partis de rien. Nos remerciements vont également à Monsieur Gilles SCHAEFFER pour son travail de coordinateur au sein de l'École polytechnique. Et enfin, nous remercions tous nos amis et proches qui nous ont soutenus dans ce projet parfois difficile.

Table des matières

1 Introduction	3
1.1 Enjeux et motivation	3
1.1.1 Les enjeux de la cryptographie au XXI ^{ème} siècle	3
1.1.2 Cryptographie et réseaux euclidiens	4
1.2 État de l'art	4
1.2.1 Des problèmes difficiles sur les réseaux euclidiens	4
1.2.2 Protocoles de chiffrement pour le <i>cloud</i>	5
1.2.3 Autour des formes d'ABE	7
1.3 Notre PSC	8
1.3.1 Contribution à la littérature	8
1.3.2 Organisation du travail	9
2 Description du schéma [BV22]	10
2.1 Initialisation : fonction <i>Setup</i>	11
2.1.1 Présentation	11
2.1.2 Contenu détaillé	11
2.2 Génération de clefs : fonction <i>KeyGen</i>	11
2.2.1 Présentation	11
2.2.2 Contenu détaillé	12
2.3 Chiffrement : la fonction <i>Enc</i>	12
2.3.1 Présentation	12
2.3.2 Un bref détour par <i>BGG⁺-lite</i>	12
2.3.3 Contenu détaillé	13
2.4 Déchiffrement : la fonction <i>Dec</i>	14
2.4.1 Présentation	14
2.4.2 Contenu détaillé	14
3 Une implémentation de [BV22]	14
3.1 Architecture	15
3.1.1 Objets	15
3.1.2 Structure	15
3.2 Algorithmes	16
3.2.1 Fonctions auxiliaires importantes	16
3.2.2 Fonctions de <i>BGG⁺-lite</i>	19
3.2.3 Fonctions du CP-ABE	20
3.3 Tests	21
3.3.1 Correction	21
3.3.2 Choix des paramètres	22
3.3.3 Analyse de performances	23
4 Conclusion	24
Bibliographie	26

Prolégomènes

Avant de débuter cet article, on se propose de dresser quelques conventions et notations afin d'en faciliter la lecture :

- \mathbb{Z} représente l'ensemble des entiers relatifs. \mathbb{Z}_q représente l'ensemble \mathbb{Z} modulo l'entier q . On note $\llbracket \dots \rrbracket$ les intervalles d'entiers.
- On note par une lettre majuscule grasse les matrices d'éléments de \mathbb{Z} ou \mathbb{Z}_q . On note par une lettre minuscule grasse les vecteurs d'éléments de \mathbb{Z} ou \mathbb{Z}_q . On note par une minuscule italique les scalaires.

Exemples : $\mathbf{u} \in \mathbb{Z}_q^n$, $\mathbf{A} \in \mathbb{Z}^{n \times m}$, $b \in \mathbb{Z}$.

- On note $\mathcal{U}(E)$ la distribution uniforme sur un ensemble E .
- On note $\mathcal{D}_{\mathbb{E}, \alpha}$ la distribution gaussienne discrète¹ sur un réseau euclidien \mathbb{E} , de paramètre α . On note $\mathcal{D}_{\mathbb{E}, \alpha, c}$ lorsqu'elle est centrée sur l'élément c (si c est omis, on considère que $c = 0$).
- On note \mathbf{I}_n la matrice identité de taille n . L'indice pourra être omis lorsque la dimension est implicite.
- On note par un mot en linéales les éléments constitutifs du cryptosystème.
Exemples : le chiffré CT, la clef publique PK.
- On note par un mot en chasse fixe les fonctions et variables de l'implémentation.
Exemples : la fonction TrapSamp, la variable u.
- Les vecteurs seront toujours représentés sous forme de lignes, et considérés comme tels dans les produits matriciels.

De plus, on se donne q un nombre premier, et on note $[q]$ après une opération s'effectuant modulo q .

1 Introduction

1.1 Enjeux et motivation

1.1.1 Les enjeux de la cryptographie au xxI^{ème} siècle

Depuis ses origines, la cryptographie a été principalement conçue pour sécuriser des échanges entre deux interlocuteurs. Pourtant, depuis une dizaine d'années, les échanges tendent à se complexifier. De plus en plus, les informations sont stockées sur des serveurs distants, et accessibles à des groupes d'individus. Cette méthode de transmission de l'information constitue ce qui est communément appelé le *cloud*. Dès lors, afin de sécuriser les échanges par le *cloud*, il est nécessaire de développer de nouveaux protocoles cryptographiques pouvant gérer plus de deux individus. Les serveurs distants étant directement connectés au réseau internet, il est primordial de pouvoir protéger les données qu'ils renferment. On étudie donc des systèmes de chiffrement fondés sur les *attributs* de chaque individu. Ainsi, on peut laisser des groupes définis d'individus (potentiellement des singletons) accéder à l'information. Ces groupes sont précisément définis en connaissant leurs caractéristiques (ou attributs).

D'autre part, le monde de la cryptographie est également chamboulé par le développement de l'*ordinateur quantique*. La majorité des systèmes de chiffrement utilisés à l'heure actuelle repose sur le protocole RSA [RSA78]. La sécurité de celui-ci est fondée sur la difficulté d'un problème (à savoir son insolubilité en un temps raisonnable à l'échelle du transfert de l'information) de factorisation sur les entiers. Cependant, l'émergence de l'ordinateur quantique menace la sécurité de tels systèmes. Dans le cas de RSA, l'algorithme *quantique* de Shor [Sho97] permet notamment la factorisation de nombres entiers en temps $O(\log(n)^3)$, conduisant ainsi à la résolution du problème difficile sur lequel se fonde

1. Une *distribution gaussienne discrète* sur un réseau euclidien est obtenue en renormalisant les valeurs obtenues avec une gaussienne bi-dimensionnelle continue sur les points du réseau.

la sécurité de RSA en un temps raisonnable. Le développement de systèmes de chiffrement *post-quantiques*, résistants à des attaques par l'ordinateur quantique est donc primordial, dans un contexte où les progrès faits dans le domaine des calculateurs quantiques sont extrêmement rapides. Par ailleurs, avant même qu'un tel ordinateur ne soit fonctionnel, il est capital de protéger les échanges les plus sensibles avec des protocoles cryptographiques post-quantiques, étant donné que ceux-ci peuvent être stockés, puis potentiellement déchiffrés grâce à un calculateur quantique plus tard.

1.1.2 Cryptographie et réseaux euclidiens

Dans ce contexte de grands bouleversements, de nombreuses méthodes de chiffrement sont développées afin de répondre aux nouveaux enjeux de la cryptographie. Il s'agit à la fois d'être résistant aux attaques de l'ordinateur quantique, et de permettre des échanges à grands nombres d'interlocuteurs (dans des dynamiques de *cloud*). De tels nouveaux protocoles de chiffrement doivent à terme pouvoir remplacer ceux fondés sur la factorisation d'entiers (donc le très célèbre RSA), ou sur le logarithme discret, vulnérables à de telles attaques.

Dans le but de parvenir à un système consensuel et unifié (sur le modèle de RSA), le *National Institute of Standards and Technology* américain (NIST) a lancé en 2016 une compétition visant à sélectionner un protocole standardisé répondant aux problématiques citées dans le paragraphe ci-dessus [NIS16]. De très nombreux systèmes de chiffrement ont été proposés (un total de 69 ont pu participer au premier tour de sélection). La sécurité de ceux-ci est fondée sur des problèmes réputés difficiles (y compris pour l'ordinateur quantique) variés. En 2022, au troisième tour de sélection, les seuls domaines d'où sont tirés ces problèmes difficiles sont les suivants :

- les réseaux euclidiens,
- les fonctions de hachage,
- les codes correcteurs d'erreurs,
- les courbes elliptiques.

Actuellement, au quatrième tour de sélection, trois protocoles ont été sélectionnés dans le cadre de cette compétition du NIST, dans le but d'être standardisés. Il s'agit du protocole *Kyber* [BDK⁺18] permettant le chiffrement de messages et le partage de clefs, ainsi que des protocoles *Dilithium* [DKL⁺18] et *Falcon* [FHK⁺19] pour la signature de messages. Il est intéressant de remarquer que ces trois systèmes cryptographiques sont fondés sur l'étude des *réseaux euclidiens*.

C'est effectivement un sujet d'étude extrêmement prometteur pour la cryptographie. On développera plus en détail les aspects techniques ci-après, mais on peut d'ores et déjà noter l'apport des recherches sur l'échantillonnage gaussien (notamment grâce à [GPV07]), qui lèvent de nombreuses difficultés techniques, et permettent aujourd'hui de construire des cryptosystèmes complets et efficaces. Il est important de noter qu'il a même été possible de construire des schémas de chiffrement complètement *homomorphes* [Gen09]². C'est pour toutes ces raisons que nous avons choisi de cibler précisément le domaine de la cryptographie fondée sur les réseaux euclidiens dans le cadre de notre PSC.

1.2 État de l'art

1.2.1 Des problèmes difficiles sur les réseaux euclidiens

La cryptographie se fonde avant tout sur des problèmes difficiles, et tout particulièrement sur la création de *fonctions à trappe*. Il s'agit de fonctions dont l'image est facile à calculer, mais dont l'inverse est difficilement évaluable, à moins d'avoir la *trappe*. Dès lors, on peut schématiquement chiffrer un

2. Le chiffrement homomorphe correspond à une particularité des systèmes de cryptographie asymétriques. Pour ceux-ci, les opérations arithmétiques peuvent être réalisées sur les chiffrés directement. On peut donc agir sur les clairs en travaillant sur les chiffrés. Un opérateur devant réaliser des opérations sur les messages n'a donc pas à les déchiffrer.

message en calculant l'image de la fonction, et le déchiffrer en en calculant l'inverse (ce qui n'est possible qu'avec la trappe, possédée uniquement par ceux ayant l'autorisation de déchiffrer le message). Dans le cadre de la cryptographie fondée sur les réseaux euclidiens, on note deux principaux problèmes difficiles : le problème *learning with errors* (LWE), et le problème *inhomogenous short integer solution* (ISIS). On procède ici à leur description détaillée.

Le problème ISIS. Ce premier problème a été introduit par M. AJTAI en 1996 [Ajt96]. Il se présente comme ci-dessous.

Définition 1 (Problème ISIS). Il s'agit de trouver une solution de petite norme euclidienne $\mathbf{x} \in \mathbb{Z}_q^n$ à un problème de la forme

$$\mathbf{x}\mathbf{A} = \mathbf{u} \quad [q],$$

pour une matrice $\mathbf{A} \leftarrow \mathcal{U}(\mathbb{Z}_q^{n \times m})$ et un vecteur $\mathbf{u} \leftarrow \mathcal{U}(\mathbb{Z}_q^m)$ tirés uniformément, avec n et m des entiers tels que $m > n$.

On notera qu'il est *a priori* envisageable de trouver un vecteur non nul vérifiant $\mathbf{x}\mathbf{A} = \mathbf{u} \quad [q]$, mais pour autant, il n'est pas évident qu'un tel vecteur soit bien de petite norme. Dès lors, lorsque la matrice \mathbf{A} n'est pas pathologique, il n'est pas possible de résoudre le problème en un temps polynomial sans la donnée d'une matrice \mathbf{T} qui soit une base *courte* (c'est à dire de petite norme euclidienne) de l'espace

$$\Lambda^\perp(\mathbf{A}) := \{\mathbf{x} \mid \mathbf{x}\mathbf{A} = 0 \quad [q]\}.$$

Une telle matrice \mathbf{T} constitue vraisemblablement la trappe d'une telle primitive cryptographique.

Le problème LWE. Ce second problème a été introduit par O. REGEV en 2005 [Reg05]. Il s'agit cette fois de retrouver un vecteur donné à partir d'un ensemble de produits scalaires bruités de celui-ci avec des vecteurs d'une distribution uniforme. On notera que tout comme le problème ISIS, il s'agit d'un *average-case-problem*. C'est à dire qu'il est intéressant (*i.e.* difficile) dans le cas moyen, et pas uniquement dans un ensemble de cas *non pathologiques*, puisque les données du problème sont tirées aléatoirement. Cependant, à la différence du problème ISIS, le problème LWE ne possède qu'une seule solution. Cela le rend donc tout à fait adapté aux protocoles de codages, là où ISIS tend plutôt à être une primitive utile dans les protocoles de signature. On présente plus précisément le problème LWE ci-dessous.

Définition 2 (Problème LWE). On se donne un vecteur $\mathbf{u} \in \mathbb{Z}_q^n$. L'objectif est de retrouver ce vecteur à partir de la distribution

$$\{(\mathbf{A}, \mathbf{u}\mathbf{A} + \mathbf{e}) \mid \mathbf{A} \leftarrow \mathcal{U}(\mathbb{Z}_q^{n \times m}), \mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z}^m, \alpha q}\},$$

pour un paramètre α donné.

Les deux problèmes présentés ci-dessus seront intensivement utilisés dans la suite de notre étude. Ce sont effectivement des primitives cryptographiques très fréquentes et utiles pour la construction de protocoles de chiffrement. Elles forment en quelque sorte les briques de base de la cryptographie fondée sur les réseaux euclidiens.

1.2.2 Protocoles de chiffrement pour le *cloud*

Chiffrement à clef publique. Les protocoles de *chiffrement à clef publique* sont particulièrement adaptés aux échanges à deux individus : l'un d'eux possède le message en clair à transmettre, et l'autre

une clef secrète pour le déchiffrer³. Il s'agit donc d'un protocole de chiffrement *asymétrique* : Alice et Bob n'ont pas des rôles interchangeables, et n'ont pas en possession les mêmes éléments. On notera que c'est sur de tels protocoles asymétriques que s'appuient les protocoles de chiffrement *symétriques*. Le premier est utilisé afin de partager la clef, avant que le deuxième (souvent plus sûr) ne puisse être utilisé. Un exemple notable de chiffrement à clef publique, dont la difficulté est fondée sur le problème LWE, est le *chiffrement de Regev* [Reg05], que l'on présente ci-dessous.

Définition 3 (Chiffrement de Regev). On se donne une instance du problème LWE pour le vecteur $\mathbf{s} \in \mathbb{Z}_q^n$ (qui fera office de *clef secrète*). La *clef publique* est alors un couple $(\mathbf{A}, \mathbf{b}) = (\mathbf{A}, \mathbf{s}\mathbf{A} + \mathbf{e})$ où $\mathbf{A} \leftarrow \mathcal{U}(\mathbb{Z}_q^{n \times m})$, et $\mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z}^m, \alpha q}$. L'encodage du bit μ se fait en calculant le chiffré CT tel que

$$\text{CT} := (\mathbf{x}, y) = (\mathbf{r}\mathbf{A}, \mathbf{r}\mathbf{b}^T + \lfloor q/2 \rfloor \mu),$$

où $\mathbf{r} \leftarrow \mathcal{U}(\{0, 1\}^n)$. Quant au déchiffrement, il est réalisé grâce à la clef secrète \mathbf{s} en calculant

$$y - \mathbf{x}\mathbf{s}^T \pmod{q}.$$

Si le résultat est plus proche de $\lfloor q/2 \rfloor$ que de 0, alors $\mu = 1$, sinon $\mu = 0$.

Chiffrement par l'identité. Cependant, un tel chiffrement à clef publique ne permet pas d'avoir de contrôle d'accès fin sur les données. Il n'existe qu'une seule clef privée, qui donne accès à toutes les données qui ont été chiffrées à l'aide de la clef publique associée. Dès lors, il n'est pas envisageable d'utiliser ce genre de protocoles cryptographiques dans le cadre du *cloud*. C'est pourquoi d'autres familles de cryptosystèmes ont été développées : par exemple le *chiffrement par l'identité* (*Identity-based encryption*, ou IBE). Dans le cas des IBE, les clefs publiques sont fonctions de l'*identité* de chacun. Celles-ci peuvent être calculées par tout le monde, dans le but de chiffrer un message qui ne sera lisible que par un individu donné, discriminé par son identité. Quant aux clefs privées, elles sont distribuées à chacun par une autorité de confiance⁴. Elles ne permettent naturellement que de déchiffrer exclusivement les messages chiffrés avec la clef publique associée à l'individu donné.

Chiffrement par attributs. Bien que les IBE permettent bien plus de flexibilité que les chiffrements à clefs publiques classiques, ils possèdent également des limites intrinsèques qu'il convient de dépasser dans le cas d'un usage à l'échelle du *cloud*. En effet, il est impossible de construire des groupes d'individus se chevauchant, ayant chacun accès à un éventail donné d'informations, à moins de multiplier les clefs privées pour chaque individu (ce que l'on préfère éviter). On ne peut donc pas construire des conditions complexes sur l'accès de chacun à des données. C'est donc dans ce contexte qu'a émergé le *chiffrement par attributs* (*Attribute-based encryption*, ou ABE). Celui-ci entend permettre une plus grande sélectivité sur l'accès aux données. Il s'agit de donner à chaque individu un ensemble d'attributs, puis de discriminer la possibilité de déchiffrer une information selon les *attributs* possédés : seuls les individus avec les attributs appropriés pourront déchiffrer les données. Dès lors, on prolonge le chiffrement par l'identité, en approfondissant la notion d'identité afin de la rendre plus complète. Le niveau de contrôle obtenu sur l'accès aux données est ainsi plus granulaire. Une telle propriété est particulièrement recherchée dans les applications liées au *cloud*.

Dans le modèle de l'ABE, les clefs publiques sont construites en fonction d'un ensemble d'attributs cibles (que l'on peut voir comme une suite de booléens). Elles permettent de chiffrer une information de telle sorte qu'elle ne soit lisible que par les individus possédant ces attributs cibles. Les clefs privées contiennent les informations sur les attributs de celui qui possède la clef (le receveur des chiffrés).

3. On n'oublie pas également la clef publique, clef de voûte du système cryptographique, qui contient toute l'information, sans pour autant que celle-ci ne soit directement accessible (notamment pour le déchiffrement).

4. En effet, s'il était possible à chacun de construire lui-même sa clef privée, celles-ci pourraient être construites par n'importe qui (les identités étant publiques), ce qui mettrait à mal toute la structure du système cryptographique...

Dès lors, la situation est similaire au cas de l'IBE, à savoir que les clefs privées doivent être générées par une autorité de confiance, selon une méthode qui soit secrète, à partir d'une sélection d'attributs caractérisant son possesseur. Dans le cas contraire, n'importe qui pourrait construire des clefs privées, et la sécurité du protocole cryptographique s'effondrerait.

L'une des premières propositions d'ABE a été formulée en 2013 [GVW13]. Elle représente effectivement les attributs comme un ensemble de booléens. Quant au test des attributs, afin de savoir si un individu (*i.e.* une clef secrète) peut déchiffrer un message chiffré, il s'effectue en évaluant un circuit booléen en l'ensemble des attributs. Dès lors que le circuit booléen s'évalue à 0 sur les attributs d'un receveur, il peut lire le chiffré. Ainsi, un tel ABE permet une grande finesse dans la gestion de l'accès aux données, avec une immense liberté permise par l'usage de circuits booléens. Les nombreuses qualités des ABE nous ont donc poussés à nous y intéresser plus en détail, et à choisir un ABE dans le cadre de notre implémentation.

1.2.3 Autour des formes d'ABE

Les protocoles de chiffrement par attributs ne sont cependant pas tous semblables. Ils peuvent notamment être regroupés en deux grandes catégories. Il y a en effet deux types d'informations à stocker et à transmettre dans un cryptosystème ABE : les attributs relatifs aux individus receveurs, contenus dans les clefs privées, et les attributs relatifs à la possibilité de déchiffrer des messages, contenus dans les chiffrés. Ces deux informations peuvent être stockées de deux manières : ou bien sous forme d'un ensemble de booléens, ou bien sous la forme d'une fonction booléenne. Naturellement, étant donné qu'il est nécessaire d'avoir d'un côté une fonction, et de l'autre des arguments, seules deux possibilités sont envisageables.

Politique d'accès sur la clef. Les protocoles de la forme *Key-policy Attributed-based encryption* (KP-ABE) sont tels que les chiffrés contiennent les informations nécessaires au décodage sous la forme d'un ensemble de booléens. D'autre part, les clefs privées des receveurs contiennent les informations sur leurs propres attributs sous la forme d'une fonction booléenne. En conséquence, lors d'une tentative de déchiffrement par un individu, la fonction booléenne de sa clef privée s'évalue en les informations contenues dans le chiffré. Métaphoriquement, on peut imaginer que chaque individu possède dans sa clef privée un cadenas sur lequel on vient renseigner le code contenu dans le chiffré pour tenter de l'ouvrir. Le principal inconvénient des KP-ABE est de laisser peu de souplesse aux émetteurs (comme expliqué par [JTC22]). En effet, les individus ont déjà leurs clefs avec la structure d'accès globale, et il n'est donc pas possible de proposer des politiques d'accès autres que celles envisagées au moment de la création et de l'envoi des clefs privées.

Politique d'accès sur le chiffré. D'autre part existent les protocoles de la forme *Ciphertext-policy Attributed-based encryption* (CP-ABE). Ceux-ci sont tels que les chiffrés contiennent les informations nécessaires au décodage sous la forme d'une fonction booléenne. Par ailleurs, les clefs privées des receveurs contiennent les informations sur leurs propres attributs sous la forme d'un ensemble de booléens. De ce fait, lors d'une tentative de déchiffrement par un individu, la fonction booléenne du chiffré s'évalue en les informations contenues dans sa clef privée. Métaphoriquement, on peut imaginer que chaque individu possède dans sa clef privée un code qu'il vient renseigner sur un cadenas contenu dans le chiffré pour tenter de l'ouvrir. Ainsi, les CP-ABE offrent une plus grande flexibilité du système, chaque message possédant en effet sa propre politique d'accès (donc moins bridée que pour un KP-ABE). Cependant, on observe que la taille des chiffrés croît linéairement en le nombre d'attributs, d'où une problématique d'expédition.

On propose les définitions détaillées d'un CP-ABE et de sa sécurité (de telles définitions sont analogues à celles énoncées dans [LLS14]).

Définition 4 (CP-ABE). Un *protocole de chiffrement par attributs à politique d'accès sur le chiffré* est la donnée de quatre algorithmes probabilistes polynomiaux :

- $\text{Setup}(1^\lambda, \text{des}) \rightarrow (\text{MPK}, \text{MSK})$: à partir du *paramètre de sécurité* λ et d'une description des , Setup renvoie la *master public key* (MPK) et la *master secret key* (MSK).
- $\text{KeyGen}(1^\lambda, \text{MPK}, \text{MSK}, x) \rightarrow \text{SK}_x$: à partir de la *master public key*, de la *master secret key*, et de la suite d'attributs x , KeyGen renvoie la clef secrète associée aux attributs x (SK_x).
- $\text{Enc}(1^\lambda, \text{MPK}, f, \mu) \rightarrow \text{CT}_f$: à partir de la *master public key*, de la fonction booléenne de politique d'accès f et du bit à chiffrer μ , Enc renvoie le chiffré associé CT_f .
- $\text{Dec}(1^\lambda, \text{MPK}, \text{SK}_x, x, \text{CT}_f) \rightarrow \mu$: à partir de la *master public key*, de la clef privée de l'individu, de ses attributs et du chiffré, Dec renvoie le bit chiffré.

Le protocole de chiffrement doit être *correct*, c'est à dire que pour tout λ assez grand, et pour f et x tel que $f(x) = 0$, on a

$$\forall \mu \in \{0, 1\}, \text{Dec}(1^\lambda, \text{MPK}, \text{KeyGen}(1^\lambda, \text{MPK}, \text{MSK}, x), x, \text{Enc}(1^\lambda, \text{MPK}, f, \mu)) = \mu,$$

où $\text{MPK}, \text{MSP} := \text{Setup}(1^\lambda, \text{des})$.

Définition 5 (Sécurité d'un CP-ABE). On définit la sécurité d'un CP-ABE du point de vue de l'*indistinguabilité des chiffrés* par le jeu suivant.

Un challenger lance le Setup et génère une instance de (MPK, MSK) . Un attaquant a accès à la MPK . Il propose deux messages M_0 et M_1 , ainsi qu'une politique de déchiffrement f ⁵. L'un des deux messages est chiffré par le challenger pour la politique f , et renvoyé à l'attaquant. Ce dernier doit alors déterminer en temps polynomial si le chiffré est celui de M_0 ou de M_1 .

Le protocole est considéré comme sûr si l'attaquant ne réussit en moyenne pas mieux que le hasard⁶. Afin de faire son choix, l'attaquant peut obtenir autant de clefs secrètes SK_x vérifiant $f(x) = 1$ que voulu. On parle alors d'*indistinguabilité des chiffrés* dans une attaque à *clairs choisis* (IND-CPA). Pour obtenir l'*indistinguabilité des chiffrés* dans une attaque à *chiffrés choisis* (IND-CCA), plus forte que l'IND-CPA, il faut que l'attaquant ait également accès à un oracle de déchiffrement qui lui permet de déchiffrer tout chiffrés différents de M_0 et M_1 , associés à la politique f , avant de faire son choix.

Dans le cadre de notre étude et de notre implémentation, nous avons choisi de travailler sur un protocole de CP-ABE plutôt que de KP-ABE. Cette décision est principalement motivée par l'état de l'art, et notamment l'existence de CP-ABE dont aucune implémentation n'a encore été proposée. CP-ABE et KP-ABE étant de proches cousins, possédant chacun leurs avantages et inconvénients respectifs, il n'était pas possible de les discriminer sur un argument purement technique.

1.3 Notre PSC

1.3.1 Contribution à la littérature

On se propose, dans le cadre de notre *projet scientifique collectif* (PSC), d'étudier un système de chiffrement de type CP-ABE, fondé sur l'utilisation de réseaux euclidiens, et tout particulièrement autour du problème LWE et de ses dérivés. Nous avons choisi de travailler sur le système de chiffrement de Z. BRAKERSKI et V. VAIKUNTANATHAN [BV22].

5. Le scénario est donc dit *adaptatif*, l'attaquant choisit f en même temps que M_0 et M_1 . Dans un cas *sélectif*, l'attaquant choisit f avant que ne soit lancé le Setup par le challenger.

6. On peut naturellement définir avec plus de précision ce que signifie *faire mieux que le hasard*, à l'aide de *distingueurs*. Cette théorie est présentée dans [LLS14], et ne fait pas l'objet d'une ré-exposition ici.

Notre contribution porte notamment sur l'explication dans le détail de ce système de chiffrement, dont nous avons explicité chacune des étapes. Il s'agit en particulier d'approfondir le lien entre ce CP-ABE et le KP-ABE sur lequel il se base (le protocole BGG⁺-lite [BGG⁺¹⁴]). Le détail des dimensions et des propriétés de chaque objet est primordial pour que le cryptosystème soit implantable en pratique, et l'appel à certaines fonctions et éléments d'un autre système de chiffrement nécessite un regard particulièrement attentif sur ces points.

L'essentiel de notre apport est cependant l'implémentation en elle-même du protocole décrit par Z. BRAKERSKI et V. VAIKUNTANATHAN. Celle-ci n'avait pas encore été réalisée, et a mobilisé une part conséquente de notre travail. Si le protocole BGG⁺ a déjà été implanté par W. DAI *et al.* [DDP⁺¹⁸], il reste beaucoup à faire pour l'adapter dans le cadre de notre construction, sans oublier les éléments propres au CP-ABE, pour certains vierges d'implémentation.

Dans le cadre de notre travail d'implémentation du schéma en langage C, nous avons également pu réutiliser des éléments déjà construits par L. PRABEL, notamment au cours de son étude sur les cryptosystèmes fondés sur les modules de polynômes [BEP⁺²¹]. Là encore, le travail d'adaptation est primordial, dans la mesure où les objets manipulés, s'ils sont semblables, n'en restent pas moins différents.

1.3.2 Organisation du travail

Au cours de notre PSC, nous avons eu l'occasion de travailler en groupe sur une période d'une dizaine de mois. Dès lors, il nous a été nécessaire d'organiser notre travail avec précision.

Échéancier. Dans un premier temps, nous avons dressé un échéancier que nous avons tenu tout au long de notre projet. Celui-ci est présenté ci-dessous⁷ :

- **Septembre 2023** : Documentation sur l'état de l'art de la cryptographie à base de réseaux.
- **Octobre et novembre 2023** : Documentation sur les systèmes de chiffrement fondés sur les attributs, choix du modèle à implémenter.
- **Décembre 2023** : Étude de la stratégie d'implémentation, détermination des éléments à construire et des éléments à récupérer de la littérature.
- **Janvier à mars 2024** : Implémentation de la solution choisie.
- **Avril 2024** : Finalisation de l'implémentation, tests finaux et complétion du rapport.
- **Mai 2024** : Fin du projet et rendu de l'implémentation et du rapport.

Peu de retard a été pris sur notre échéancier, excepté pour le début de la phase d'implémentation, qui a vraisemblablement débuté au début du mois de février 2024. Il est important de noter que la période de documentation du projet a duré près de quatre mois.

L'étude théorique de l'état de l'art de la cryptographie post-quantique a été longue et relativement laborieuse (comme attendu lors de la rédaction de l'échéancier en septembre 2023). Ce domaine, nouveau pour nous tous, peut être particulièrement riche pour un néophyte. Cela explique le long processus d'appropriation, ainsi que d'étude théorique. Nous avons également consacré du temps à la compréhension en profondeur du schéma [BV22] que nous cherchons à implémenter.

Dans un second temps, la phase d'implémentation a été plus courte, mais également plus intense. Elle a été l'occasion de davantage répartir le travail au sein du groupe, comme on le détaillera dans le paragraphe suivant. Cette seconde période nous a mis face à de nombreuses difficultés inopinées, sur des points que nous ne considérons pas comme délicats lors de l'étude théorique. Un aller-retour continu entre théorie et pratique s'est avéré nécessaire afin d'avancer dans l'implémentation.

7. Cet échéancier est directement tiré de notre *Proposition détaillée* de PSC de septembre 2023.

Répartition du travail. Comme on a pu l'évoquer, une répartition du travail au sein du groupe s'est avérée nécessaire pour le respect de l'échéancier que l'on s'était fixé. Elle s'est principalement articulée autour de deux sous-groupes :

- D'un côté, Manan et Quentin se sont principalement intéressés aux aspects les plus théoriques.
- D'un autre, Damya, Théodore et Rémi, plus à l'aise avec le langage C, se sont focalisés sur l'implémentation dès lors que celle-ci a été entamée.

L'articulation du travail au cours du temps et au sein des deux sous-groupes est détaillée en figure 1. Avant le mois d'octobre 2023, tout le groupe travaillait simultanément sur l'état de l'art et l'initiation à la cryprographie post-quantique.

Cette répartition en un binôme et un trinôme nous a permis d'avancer bien plus vite pendant la période consacrée à l'implémentation. Il a notamment été possible de régler des difficultés théoriques observées au cours de l'implémentation, pendant que le reste du groupe continuait d'autres parties de la programmation.

Par ailleurs, le binôme théoricien a pu avancer sur la rédaction du rapport au cours des avancées fluides de l'implémentation. Ceci nous a permis de fournir le rapport en temps voulu, tout en continuant à avancer le programme jusqu'à l'échéance finale du mois d'avril.

	D'octobre à janvier	De janvier à mars	D'avril à mai
Manan Quentin	<ul style="list-style-type: none"> • Approfondissement théorique • Choix du schéma à implémenter 	<ul style="list-style-type: none"> • Suivi théorique • Rédaction du rapport 	<ul style="list-style-type: none"> • Finalisation du rapport
Damya Théodore	<ul style="list-style-type: none"> • Documentation théorique • Apprentissage du langage C 	<ul style="list-style-type: none"> • Implémentation 	<ul style="list-style-type: none"> • Rédaction du rapport
Rémi	<ul style="list-style-type: none"> • Documentation théorique • Étude des implantations existantes • Apprentissage et mise en place de l'environnement en C 	<ul style="list-style-type: none"> • Implémentation 	<ul style="list-style-type: none"> • Finalisation du rapport • Test de l'implémentation

FIGURE 1 – Organisation et répartition des tâches

2 Description du schéma [BV22]

Après consultation de différents schémas de chiffrement, notre regard s'est porté sur celui proposé par Z. BRAKERSKI et V. VAIKUNTANATHAN. Il s'agit bien d'un CP-ABE, que l'on se propose dans cette section d'étudier. On découpe son analyse selon les quatre algorithmes de la définition 4, correspondant aux phases d'**initialisation**, de **génération de clefs**, de **chiffrement** et de **déchiffrement**.

Avant de commencer le détail des phases du schéma, on introduit quelques paramètres entiers qui seront utilisés au cours de notre exposé :

- le paramètre de sécurité $\lambda \in \mathbb{N}$,
- la longueur $r \in \mathbb{N}$ des attributs,
- l'entier premier $q \in \mathbb{N}$, et l'entier $k := \lceil \log q \rceil$,
- la profondeur maximale $d \in \mathbb{N}$ du circuit booléen définissant la politique d'accès aux données,
- le degré $n \in \mathbb{N}$ des polynômes sur lesquels on travaille,

- la matrice gadget $\mathbf{G} \in \mathbb{Z}_q^{n \times kn}$, telle que

$$\mathbf{G} = \mathbf{g} \otimes \mathbf{I}_n,$$

avec $\mathbf{g} = (1, 2, \dots, 2^{k-1}) \in \mathbb{Z}_q^k$,

- on pose $m \in \mathbb{N}$, un paramètre pour le CP-ABE, tel que $m > n$,
- on pose $\ell \in \mathbb{N}$, un paramètre pour le KP-ABE BGG⁺-lite, tel que $\ell = kn$.

2.1 Initialisation : fonction **Setup**

2.1.1 Présentation

Le *setup* est la première phase du cryptosystème, qui vise à construire la **MPK** et la **MSK** : respectivement la *master public key* et la *master secret key*. Celles-ci seront utilisées tout au long du protocole. On notera que la **MSK** est bien une trappe pour des extraits de la **MPK**, comme on le présentera ci-après. Cette phase sera intégralement implémentée dans la fonction TrapGen (directement appelée par **Setup**).

2.1.2 Contenu détaillé

Comme expliqué, cette phase conduit à la production de **MPK** (la matrice \mathbf{B} définie ci-dessous) et **MSK** (la matrice \mathbf{T} définie ci-dessous). Concrètement, on utilise une fonction TrapGen définie comme suit :

$$\boxed{\text{TrapGen}(1^{2rn}, 1^m, q) \longrightarrow (\mathbf{B}, \mathbf{T})}.$$

La matrice $\mathbf{B} \in \mathbb{Z}_q^{m \times (2r+1)n}$ peut être décomposée par bloc en $2r + 1$ matrices de taille $m \times n$:

$$\mathbf{B} = (\mathbf{B}_0 \parallel \mathbf{B}_{1,0} \parallel \mathbf{B}_{1,1} \parallel \dots \parallel \mathbf{B}_{r,0} \parallel \mathbf{B}_{r,1}),$$

et la matrice $\mathbf{T} \in \mathbb{Z}^{m \times m}$ est à petits coefficients, et vérifie la propriété

$$\left\{ \begin{array}{l} \mathbf{T}\mathbf{B}_0 = 0 \quad [q], \\ \forall i \in \llbracket 1, \dots, r \rrbracket, \forall b \in \{0, 1\}, \quad \mathbf{T}\mathbf{B}_{i,b} = 0 \quad [q]. \end{array} \right. \quad (1)$$

On notera que \mathbf{T} peut-être considérée comme une trappe *jointe* pour l'ensemble des $\mathbf{B}_{i,b}$. Ces dernières matrices peuvent se combiner par la donnée de $\{x_i\}_{i \in \llbracket 1, \dots, r \rrbracket} \in \{0, 1\}^r$, qui correspond précisément à la donnée des attributs d'un utilisateur - on retient alors uniquement les matrices \mathbf{B}_{i,x_i} . Par ailleurs, la génération de trappe est un thème cryptographique très récurrent, et notamment abordé par [BEP⁺21], qui en propose une implémentation dont on s'inspirera dans la partie suivante.

2.2 Génération de clefs : fonction **KeyGen**

2.2.1 Présentation

La *key generation* est la seconde étape de notre protocole. Elle conduit à la fabrication de clefs secrètes contenant les informations sur son propriétaire. Elle est réalisée à partir de la **MSK** en spécifiant cette dernière afin qu'elle laisse apparaître les informations sur les attributs de l'individu. Cette phase sera implémentée dans la fonction TrapSamp (directement appelée par **KeyGen**).

2.2.2 Contenu détaillé

Après avoir décrit la génération des clefs maîtresses, il est nécessaire d'aborder la génération des clefs individuelles $\{\text{SK}_x\}_{x \in \mathbb{I}}$, où x parcourt l'ensemble \mathbb{I} des individus. Les SK_x sont des vecteurs $\mathbf{t}_x \in \mathbb{Z}^m$. On les génère en appelant la fonction `TrapSamp` définie comme suit :

$$\boxed{\text{TrapSamp}(\mathbf{B}, \mathbf{T}, x) \longrightarrow \mathbf{t}_x}.$$

Les vecteurs \mathbf{t}_x vérifient :

$$\begin{cases} \mathbf{t}_x \mathbf{B}_0 = 0 & [q], \\ \mathbf{t}_x \mathbf{B}_{i,x_i} = 0 & [q], \\ \mathbf{t}_x \mathbf{B}_{i,1-x_i} \leftarrow \mathcal{U}(\mathbb{Z}_q^n) \text{ indépendantes.} \end{cases} \quad (2)$$

Dès lors, la clef secrète est une trappe pour toutes les matrices \mathbf{B}_{i,x_i} , mais pas pour les matrices $\mathbf{B}_{i,1-x_i}$. On est donc bien parvenu à ce que SK_x contienne les informations sur les attributs de x . Cette propriété sera capitale dans la suite du schéma.

2.3 Chiffrement : la fonction `Enc`

2.3.1 Présentation

L'*encoding* est la troisième partie⁸ de notre protocole. Cette phase sera implémentée dans la fonction `Enc`. Dans le cadre du schéma [BV22], cette phase fait appel au protocole BGG+-Lite [BGG+14]. Dès lors, tous les éléments de ce sous-système seront notés sous la forme $\text{BGG}.x$.

Le protocole BGG+-Lite est en réalité un KP-ABE, dont on réutilise certaines fonctions (à savoir `BGG.KeyGen` et `BGG.OfflineEnc`). Dès lors, cela évite d'avoir à proposer une nouvelle implémentation de ces points techniques principalement fondés sur des instances de ISIS (pour `BGG.KeyGen`) et LWE (pour `BGG.OfflineEnc`). En effet, une implémentation de BGG+-Lite a déjà été proposée en 2018 [DDP+18]. Dans l'ensemble, cette phase de chiffrement permet d'obtenir le message chiffré CT .

2.3.2 Un bref détour par BGG+-lite

On s'intéresse uniquement au cas d'un bit $\mu \in \{0, 1\}$, qui sera bien sûr aisément extensible à un message $M \in \{0, 1\}^N$. Il s'agit tout d'abord de créer des clefs publiques et privées pour BGG. Nous construisons donc BGG.MPK (la matrice \mathbf{A} définie ci-dessous) et BGG.SK_f (la matrice \mathbf{T}_f définie ci-dessous). On notera qu'ici, l'attribut considéré est la fonction booléenne f (ne comportant que des portes NAND), qui contient la politique de déchiffrement du CP-ABE. On fait donc appel à la fonction `BGG.KeyGen`, qui donne :

$$\boxed{\text{BGG.KeyGen}(1^\lambda, 1^r, f) \longrightarrow (\mathbf{A}, \mathbf{T}_f)}.$$

La matrice $\mathbf{A} \in \mathbb{Z}^{n \times (r+1)\ell}$ peut être décomposée par blocs en $r + 1$ matrices de taille $n \times \ell$:

$$\mathbf{A} = (\mathbf{A}_0 \parallel \mathbf{A}_1 \parallel \dots \parallel \mathbf{A}_r).$$

Les matrices $\mathbf{A}_1, \dots, \mathbf{A}_r$ sont des instances de LWE (donc de distribution $\mathcal{U}(\mathbb{Z}_q^{n \times \ell})$), associées à $\mathbf{s} \in \mathbb{Z}^n$ et aux erreurs $\mathbf{e}_1, \dots, \mathbf{e}_r \in \mathbb{Z}^\ell$. Une matrice $\mathbf{A}_f \in \mathbb{Z}^{n \times \ell}$ est construite en même temps que la matrice $\mathbf{H}_{f,x,\mathbf{A}_1, \dots, \mathbf{A}_r} \in \mathbb{Z}^{r\ell \times \ell}$ par la *propriété homomorphe* :

$$(\mathbf{A}_1 + x_1 \mathbf{G} \parallel \dots \parallel \mathbf{A}_r + x_r \mathbf{G}) \cdot \mathbf{H}_{f,x,\mathbf{A}_1, \dots, \mathbf{A}_r} = \mathbf{A}_f + f(x_1, \dots, x_r) \mathbf{G}. \quad (3)$$

8. On note cependant que les phases de chiffrement et de génération de clef sont en réalité interchangeables, et même réalisables simultanément.

On obtient une telle construction en itérant la relation

$$(\mathbf{A}_1 + x_1 \mathbf{G} \parallel \mathbf{A}_2 + x_2 \mathbf{G}) \begin{pmatrix} \mathbf{G}^{-1}(\mathbf{A}_2) \\ -x_1 \mathbf{I} \end{pmatrix} = \mathbf{A}_{\text{NAND}} + \text{NAND}(x_1, x_2) \mathbf{G},$$

où $\mathbf{A}_{\text{NAND}} := \mathbf{A}_1 \mathbf{G}^{-1}(\mathbf{A}_2) - \mathbf{G} \in \mathbb{Z}_q^{n \times kn}$, et \mathbf{G}^{-1} une fonction de $\mathbb{Z}_q^{n \times m}$ dans $\{0, 1\}^{kn \times m}$ vérifiant

$$\mathbf{G}\mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A}. \quad (4)$$

On observe dans le même temps que la matrice \mathbf{A}_f vérifie

$$\mathbf{A}_f = f(\mathbf{A}_1, \dots, \mathbf{A}_r).$$

La matrice \mathbf{T}_f est alors générée selon une distribution $\mathcal{D}_{\mathbb{Z}^{\ell \times \ell}, \alpha}$ (pour un α donné), et la matrice $\mathbf{A}_0 \in \mathbb{Z}^{n \times \ell}$ est définie par

$$\mathbf{A}_0 = \mathbf{A}_f \mathbf{T}_f \quad [q].$$

On peut alors utiliser les fonctions de chiffrement provenant de BGG, directement fondées sur des instances de LWE. On utilise notamment les fonctions BGG.OfflineEnc et BGG.Dec définies telles que

$$\boxed{\text{BGG.OfflineEnc}(\mathbf{A}, \mu) \longrightarrow \text{BGG.CT}},$$

$$\boxed{\text{BGG.Dec}(\mathbf{T}_f, \text{BGG.CT}) \longrightarrow \mu}.$$

Le chiffré *offline* BGG.CT est défini comme

$$\text{BGG.CT} = \left(\mathbf{u}, \begin{pmatrix} \mathbf{s}\mathbf{A}_1 & \parallel & \dots & \parallel & \mathbf{s}\mathbf{A}_r \\ \mathbf{s}(\mathbf{A}_1 + \mathbf{G}) & \parallel & \dots & \parallel & \mathbf{s}(\mathbf{A}_r + \mathbf{G}) \end{pmatrix} + \begin{pmatrix} \mathbf{e} \\ \mathbf{e} \end{pmatrix} \right), \quad (5)$$

où $\mathbf{u} = \mathbf{s}\mathbf{A}_0 + \mathbf{e}_0$ si $\mu = 0$, ou $\mathbf{u} \leftarrow \mathcal{U}(\mathbb{Z}_q^\ell)$ si $\mu = 1$. On note qu'il permet de calculer le chiffré pour n'importe quel x :

$$\text{BGG.CT}_x = (\mathbf{u}, \mathbf{s}(\mathbf{A}_1 + x_1 \mathbf{G} \parallel \dots \parallel \mathbf{A}_r + x_r \mathbf{G}) + \mathbf{e}).$$

Quant au déchiffrement, il est réalisé grâce à la propriété homomorphique (3), et consiste en le calcul de

$$(\mathbf{s}(\mathbf{A}_1 + x_1 \mathbf{G} \parallel \dots \parallel \mathbf{A}_r + x_r \mathbf{G}) + \mathbf{e}) \mathbf{H}_{f,x,\mathbf{A}_1, \dots, \mathbf{A}_r} \approx \mathbf{s}(\mathbf{A}_f + f(x) \mathbf{G}).$$

Dans le cas où $f(x)$ s'annule, il est possible de multiplier par \mathbf{T}_f à droite pour retrouver μ (selon la proximité du résultat à \mathbf{u} , comme pour LWE).

2.3.3 Contenu détaillé

On construit une fonction Enc telle que

$$\boxed{\text{Enc}(\mathbf{B}, f, \mu) \longrightarrow \text{CT}_f},$$

qui fait appel à BGG.KeyGen , BGG.OfflineEnc et BGG.Dec .

En premier lieu, un appel à BGG.KeyGen initialise une instance du protocole BGG. Puis un appel à BGG.OfflineEnc permet de construire

$$\mathbf{C}_0 = \mathbf{S}\mathbf{A}_0 + \mathbf{E}_0 \in \mathbb{Z}^{m \times \ell},$$

et les

$$\mathbf{C}_{i,b} = \mathbf{S}(\mathbf{A}_i + b \mathbf{G}) + \mathbf{E}_{i,b} \in \mathbb{Z}^{m \times \ell},$$

grâce à la propriété du chiffré présentée en (5). On notera que l'on travaille ici avec des *matrices* $\mathbf{S} \leftarrow \mathcal{U}(\mathbb{Z}_q^{m \times n})$ et $\mathbf{E}_\bullet \leftarrow \mathcal{D}_{\mathbb{Z}^{m \times \ell}, \alpha}$ (pour un α donné).

Enfin, le chiffré renvoyé est

$$\begin{aligned}\mathsf{CT}_f &= (\mathbf{T}_f, \tilde{\mathbf{C}}_0, \{\tilde{\mathbf{C}}_{i,b}\}_{i,b}) \\ &= (\mathbf{T}_f, \mathbf{B}_0 \mathbf{U}_0 + \mathbf{C}_0, \{\mathbf{B}_{i,b} \mathbf{U}_{i,b} + \mathbf{C}_{i,b}\}_{i,b}),\end{aligned}$$

avec les matrices $\mathbf{U}_\bullet \leftarrow \mathcal{U}(\mathbb{Z}_q^{n \times \ell})$ indépendantes.

2.4 Déchiffrement : la fonction Dec

2.4.1 Présentation

Le *decoding* est la dernière partie de notre schéma. On évalue la fonction booléenne contenue dans le schéma en les attributs du destinataire qui sont contenus dans la clef secrète. Si cette évaluation donne 0, le déchiffrement a lieu et le destinataire peut lire le message en clair. Là encore, on la présente dans le cas où le message se résume à un bit $\mu \in \{0, 1\}$, étant donné que la généralisation est ensuite évidente. Cette phase sera implémentée dans la fonction Dec .

2.4.2 Contenu détaillé

Le déchiffrement est réalisé grâce à la fonction Dec :

$$\boxed{\mathsf{Dec}(x, \mathbf{t}_x, f, \mathsf{CT}_f) \longrightarrow \mu}.$$

Elle consiste en le calcul de

$$\mathbf{t}_x \left((\tilde{\mathbf{C}}_0 \parallel \tilde{\mathbf{C}}_{1,x_1} \parallel \dots \parallel \tilde{\mathbf{C}}_{r,x_r}) \begin{pmatrix} -\mathbf{I}_\ell \\ \mathbf{H}_{f,x, \mathbf{A}_1, \dots, \mathbf{A}_r} \mathbf{T}_f \end{pmatrix} \right), \quad (6)$$

et renvoie 1 si le vecteur obtenu est à petites composantes, et 0 sinon.

On notera que la connaissance de $\mathbf{A}_1, \dots, \mathbf{A}_r$ est nécessaire au déchiffrement. Ces matrices doivent donc être publiques.

3 Une implémentation de [BV22]

On propose dans cette partie notre implémentation à proprement parler. En effet, la partie précédente a été l'occasion de détailler le protocole [BV22], mais est restée purement théorique. Notre principal apport est de proposer une réalisation pratique, sous la forme d'une implémentation du système de chiffrement. Sa réalisation, dont le but est de fournir une implémentation des quatre grandes fonctions du CP-ABE : à savoir `Setup` (qui appelle directement la fonction `TrapGen`), `KeyGen` (qui appelle directement la fonction `TrapSamp`), `Enc` et `Dec`, a également nécessité la construction de nombreuses autres fonctions auxiliaires, indispensables à son fonctionnement. On détaillera dans les paragraphes à suivre ces algorithmes.

Nous avons choisi le langage C pour réaliser notre implémentation, notamment en raison de la littérature existante et des fonctions déjà réalisées dans ce langage. En effet, c'est le langage qui permet l'exécution la plus rapide, ce qui est privilégié en cryptographie - notons cependant que nous n'avons pas visé la performance dans notre implémentation. Dans le cadre de cet article, on présente les objets sous forme de code en langage C, et les algorithmes sous forme de pseudo-code en langage naturel. Ceci dans le but d'une plus grande lisibilité. L'intégralité du code est cependant présent sur un *repository GitHub* : <https://github.com/remigerme/CP-ABE>.

Pour réaliser concrètement notre implémentation, il a fallu résoudre de nombreux problèmes sur les dimensions des objets manipulés. Spécifions quelques paramètres précédemment introduits. On pose $p := m - 2$, et on prend $r = k$, afin de pouvoir réaliser concrètement l'implémentation — les calculs matriciels proposés dans le schéma ci-dessus n'étant pas possibles autrement. Cela implique que l'entier premier q croît exponentiellement avec le nombre d'attributs. On retrouve bien le problème soulevé pour les CP-ABE, à savoir que la taille des chiffrés (en nombre de bits) croît (au moins) linéairement avec le nombre d'attributs. De plus, la taille de certaines matrices (les $\mathbf{H}_{f,x,\mathbf{A}_1,\dots,\mathbf{A}_k}$) atteint $O(k\ell^2) = O(n^2k^3)$, ce qui est en pratique limitant sur la valeur des paramètres n et k utilisés.

3.1 Architecture

3.1.1 Objets

Nous avons fait le choix de représenter certains objets par des structures pour faciliter l'accès à certaines données. On présente ci-dessous une définition type de structure, pour les matrices.

```
typedef struct _matrix {
    unsigned int rows;
    unsigned int columns;
    scalar* data;
} matrix;
```

3.1.2 Structure

On présente ci-dessous la structure de notre implémentation. Celle-ci est résumée sous la forme d'un organigramme, en figure 2.

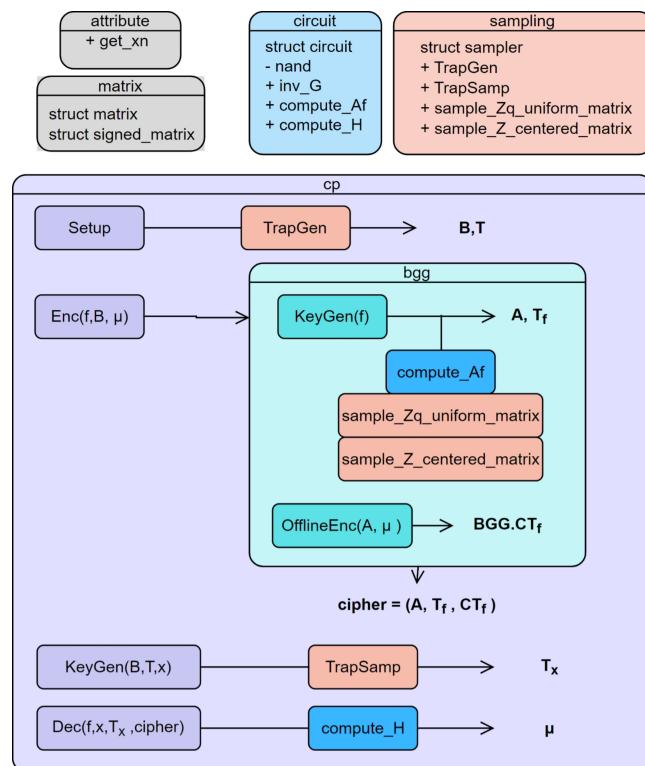


FIGURE 2 – Diagramme de dépendance entre les fichiers

3.2 Algorithmes

Table des algorithmes

1	TrapGen	17
2	TrapSamp	17
3	nand	17
4	inv_G	18
5	compute_Af	18
6	compute_H	19
7	BGG_KeyGen	19
8	BGG_OfflineEnc	20
9	Enc	20
10	Dec	21

3.2.1 Fonctions auxiliaires importantes

Génération d'entiers suivant une loi uniforme et une gaussienne discrète. Nous utilisons dans nos programmes des matrices aléatoires, dont les coefficients sont tantôt distribués uniformément sur \mathbb{Z}_q , tantôt distribués selon une gaussienne discrète $\mathcal{D}_{\mathbb{Z},\sigma}$. Pour ce faire, nous utilisons une implémentation existante de Lucas Prabel [Pra21], qui fournit une fonction permettant de générer des entiers aléatoires dans \mathbb{Z}_q , ainsi qu'une deuxième permettant de générer des entiers aléatoires distribués suivant une gaussienne discrète.

À partir de ces deux fonctions, on peut créer des matrices aléatoires. Les principales fonctions utilisées dans les algorithmes qui vont suivre sont :

- sample_Zq_uniform_matrix qui renvoie une matrice dont les coefficients sont distribués uniformément dans \mathbb{Z}_q .
- sample_Z_centered_matrix qui renvoie une matrice dont les coefficients sont distribués suivant la gaussienne $\mathcal{D}_{\mathbb{Z},\sigma}$.

Génération de trappes. Trois procédures nous permettent de générer des trappes :

- TrapGen génère la MSK \mathbf{T} du CP-ABE.
- TrapSamp permet d'obtenir une trappe associée à un attribut x à partir de la trappe jointe \mathbf{T} et d'un attribut x .
- Pour le KP-ABE BGG⁺-Lite, la trappe \mathbf{T}_f intervient après la construction de \mathbf{A}_f .

Pour TrapGen, nous nous sommes inspirés de la procédure TrapGen proposée dans [DDP⁺18], en ordonnant astucieusement les coefficients constituant le cœur de la trappe pour construire une matrice \mathbf{T} puis des matrices \mathbf{B}_0 , $\mathbf{B}_{i,b}$ vérifiant les relations

$$\mathbf{T}\mathbf{B}_{i,b} = \mathbf{T}\mathbf{B}_0 = 0 \quad [q].$$

Les vecteurs rho et mu ci-dessous sont des colonnes dans \mathbb{Z}^p .

Algorithme 1 TrapGen

```

rho ← sample_Z_centered_matrix
T ← [rho | -g + mu | Ip]                                ▷ T is the concatenation of the matrixes
for i in range [0, 2k] do
    for j in range [1, n] do
        a ←  $\mathcal{U}(\mathbb{Z}_q)$ 
        Bi, j[0] ← a
        Bi, j[1] ← 1
        for d in range [0, p - 1] do
            Bi, j[d + 2] ← g[d] - (a * rho[d] + mu[d])
        end for
    end for
end for
return {T, B}

```

En ce qui concerne TrapSamp, nous n'avons pas réussi à l'implémenter, ni à trouver des éléments théoriques qui nous indiqueraient comment procéder. Nous savons cependant précisément ce que nous attendons de cette fonction :

Algorithme 2 TrapSamp

Require: B
Require: T
Require: x
Ensure: Tx such as $\forall i \in [1, k]$ $Tx \mathbf{B}_0 = Tx \mathbf{B}_{i,x_i} = 0$ and $Tx \mathbf{B}_{i,1-x_i}$ is uniformly distributed
compute Tx
return Tx

Circuit booléen. Le circuit booléen est donné par la description d'un arbre dont chaque noeud correspond à une porte logique NAND entre ses deux enfants, et les feuilles correspondent aux entrées du circuit. Nous représentons ce circuit par un arbre binaire.

```

typedef struct btree {
    struct btree* left;
    struct btree* right;
    int n;
} circuit;

```

Les noeuds sont tels que left et right sont différents de NULL (et qu'importe la valeur de n), et pour les feuilles left = right = NULL, et n correspond alors à l'indice du vecteur d'entrée à considérer. Par exemple, pour un attribut (*i.e.* un vecteur booléen) $x = (x_1, \dots, x_k)$, être sur une feuille telle que $n = i$ revient à considérer la valeur du bit x_i .

Définissons ensuite une porte NAND pour deux matrices A et B :

Algorithme 3 nand

Require: A, B matrices ▷ Computes $NAND(A, B)$

```

R ← A * inv_G(B) - G
return R

```

où G et inv_G sont respectivement la matrice gadget et la fonction inverse de la matrice gadget.

Intuitivement, la fonction inverse décompose chaque coefficient d'une matrice en un vecteur de taille k correspondant à son écriture binaire, et multiplier par la matrice gadget \mathbf{G} permet de recomposer l'entier précédemment décomposé (d'après la relation (4)). On donne le détail ci-dessous.

Algorithme 4 inv_G

Require: A matrix ▷ Computes $G^{-1}(A)$

```

for i in range [0,n-1] do
    for j in range [0,l-1] do
        for b in range [0,k-1] do
            R[i*k + b][j] ← (A[i][j] >> b) & 1           ▷ Retrieves b-th bit of A[i][j]
        end for
    end for
end for
return R

```

Nous implémentons alors trois fonctions :

- compute_f(f, x), qui permet d'évaluer le circuit booléen f en l'attribut $x = (x_1, \dots, x_k)$.
- compute_Af($f, \vec{\mathbf{A}}$), qui permet d'évaluer le circuit booléen f sur le vecteur de matrices

$$\vec{\mathbf{A}} := (\mathbf{A}_1, \dots, \mathbf{A}_k)^9,$$

la porte NAND pour la matrice étant définie ci-dessous, construisant ainsi $\mathbf{A}_f = f(\vec{\mathbf{A}})$.

- compute_H($f, \vec{\mathbf{A}}, x$) qui permet de construire la matrice $\mathbf{H}_{f,x,\vec{\mathbf{A}}}$ définie par la propriété (3).

Les deux premières fonctions ci-dessus sont calculées par une simple recherche en profondeur dans l'arbre, appliquant la porte NAND (binaire ou matricielle) aux sous-résultats obtenus dans les fils gauche et droit. En pratique, pour des raisons de performance, nous avons utilisé de la mémoïsation pour éviter de calculer inutilement les résultats intermédiaires de sous-circuits identiques. Ci-dessous l'algorithme utilisé pour calculer \mathbf{A}_f (l'étape de mémoïsation n'est pas explicitée) :

Algorithme 5 compute_Af

Require: A matrix vector

Require: f circuit

```

if f is a node then
    R_left ← compute_Af(A, f.left)
    R_right ← compute_Af(A, f.right)
    R ← nand(R_left, R_right)
    return R
else if f is a leaf then
    A_n ← A[f.n]
    return A_n
end if      ▷ f should either be a leaf or have two children, otherwise the tree does not represent a
                circuit

```

Calcul de la matrice $\mathbf{H}_{f,x,\vec{\mathbf{A}}}$. La matrice $\mathbf{H}_{f,x,\vec{\mathbf{A}}}$ est calculée de manière récursive, en parcourant le circuit f . Pour un circuit f et des entrées $\vec{\mathbf{A}}$ et x , on s'intéresse à $\mathbf{A}_f = f(\vec{\mathbf{A}})$, $f(x)$ et $\mathbf{H}_{f,x,\vec{\mathbf{A}}}$, la matrice permettant de vérifier la relation (3).

9. Contrairement à la matrice \mathbf{A} définie précédemment, le vecteur de matrice $\vec{\mathbf{A}}$ ne contient pas la matrice \mathbf{A}_0 .

Pour l'implémentation, on définit donc une structure `H_triplet` qui nous permettra de faire remonter les résultats calculés pour les enfants du noeud considéré :

```
typedef struct H_triplet {
    matrix A;
    bool x;
    matrix H;
} H_triplet;
```

Algorithme 6 `compute_H`

Require: f circuit

Require: x attribute

Require: A matrix vector

```
if  $f$  is a leaf then
     $H \leftarrow \begin{pmatrix} 0 \\ \vdots \\ I_\ell \\ \vdots \\ 0 \end{pmatrix}$   $\triangleright H$  seen as a column is empty except in  $n$ -th position which is the identity
    return  $\{A_{f,n}, x_{f,n}, H\}$ 
end if
{ $A_l, x_l, H_l$ }  $\leftarrow$  compute_H( $f.left, x, A$ )
{ $A_r, x_r, H_r$ }  $\leftarrow$  compute_H( $f.right, x, A$ )
 $A \leftarrow A_l * \text{inv}_G(A_r) - G$ 
 $x \leftarrow 1 - x_l * x_r$ 
 $H \leftarrow H_l * \text{inv}_G(A_r) - x_l * H_r$ 
return  $\{A, x, H\}$ 
```

3.2.2 Fonctions de BGG^+ -lite

La fonction `BGG_KeyGen` génère les clés du KP-ABE, c'est à dire les matrices $\mathbf{A}_0, \mathbf{A}_1, \dots, \mathbf{A}_k$ ainsi que la trappe \mathbf{T}_f et la matrice \mathbf{A}_f décrite ci-dessus. Elle renvoie ces données sous la forme d'une structure `bogg_keys` décrite par :

```
typedef struct {
    matrix* A;
    signed_matrix Tf;
} bogg_keys;
```

Algorithme 7 `BGG_KeyGen`

Require: f circuit

```
for  $i$  in range  $[1, n]$  do
     $A[i] \leftarrow \text{sample}_{Zq} \text{uniform\_matrix}$   $\triangleright$  Here we generate the matrix vector  $A$ 
end for
 $A_f \leftarrow \text{compute\_Af}(A, f)$ 
 $Tf \leftarrow \text{sample}_{Z} \text{centered\_matrix}$   $\triangleright$  Here we generate the trap matrix  $T_f$ 
 $A[0] \leftarrow A_f * Tf \bmod q$   $\triangleright$  Finally, we compute  $A[0]$ 
return  $\{A, Tf\}$   $\triangleright$  of type bogg_keys
```

La fonction `BGG_OfflineEnc` réalise le chiffrement *offline* de BGG^+ , c'est-à-dire qu'elle calcule

le message chiffré CT_f d'un booléen u pour toutes les valeurs possibles de l'attribut. Cette fonction ne fait donc pas appel à la valeur d'un attribut x particulier.

Algorithme 8 BGG_OfflineEnc

Require: u boolean, A matrix vector

```

 $S \leftarrow \text{sample\_Zq\_uniform\_matrix}$                                  $\triangleright S$  serves as the LWE secret
 $E \leftarrow \text{sample\_Z\_centered\_matrix}$                              $\triangleright E$  is a short gaussian error vector
if  $u$  then
     $\text{BGG\_CTf}[0] \leftarrow \text{sample\_Zq\_uniform\_matrix}$ 
else
     $\text{BGG\_CTf}[0] \leftarrow SA[0] + E$ 
end if
for  $i$  in range  $[0, k-1]$  do
     $E \leftarrow \text{sample\_Z\_centered\_matrix}$ 
     $\text{BGG\_CTf}[1+2*i] \leftarrow SA[1+i] + E$ 
     $E \leftarrow \text{sample\_Z\_centered\_matrix}$ 
     $\text{BGG\_CTf}[1+2*i+1] \leftarrow S(A[1+i] + G) + E$ 
end for
```

3.2.3 Fonctions du CP-ABE

La fonction TrapGen (éventuellement encapsulée dans une fonction `Setup`) initialise les matrices $B_{i,b}$ et la trappe associée T telles que décrites plus haut (voir la relation (1)). Nous avons défini une structure `cp_keys` contenant les matrices B et T ainsi générées.

La fonction TrapSamp génère ensuite à partir de ces matrices un vecteur t_x qui constitue la clef du CP-ABE (voir la relation (2)).

La fonction Enc calcule le chiffré CT_f associé au booléen u . Nous avons défini une structure `cp_ciphertext` regroupant le message chiffré et tous les éléments qui sont utilisés au cours du déchiffrement.

```

typedef struct {
    matrix* CTf;
    signed_matrix Tf;
    matrix* A;
} cp_ciphertext;
```

Algorithme 9 Enc

Require: B matrix, f circuit, u boolean

```

 $keys \leftarrow \text{BGG\_KeyGen}(f)$                                  $\triangleright$  generates A matrix vector and Tf matrix
 $\text{BGG\_CTf} \leftarrow \text{BGG\_OfflineEnc}(A, u)$ 
 $S \leftarrow \text{sample\_Zq\_uniform\_matrix}$ 
 $\text{CTf}[0] \leftarrow B[0]*S + \text{BGG\_CTf}[0]$ 
for  $i$  in range  $[1, 2*k+1]$  do
     $\text{CTf}[i] \leftarrow B[i]*S + \text{BGG\_CTF}[i]$ 
end for
return { $\text{CTf}, keys.Tf, keys.A$ }                                 $\triangleright$  of type cp_ciphertext
```

La fonction Dec calcule le vecteur décrit en (6) et renvoie 0 s'il est court et 1 sinon.

Algorithme 10 Dec

```

Require: x binary vector, f circuit, tx matrix, cipher cp_ciphertext
H ← Compute_H(A, f, x)
CTf_x[0] ← ciper.CTF[0]
for i in range [1,k] do
    CTf_x[i] ← ciper.CTF[i*2+x[i]]
end for
res ← tx*(CTf_x*H*Tf-ciper.CTF[0])
if res is short then
    return 0
else
    return 1
end if

```

Nous n'avons pas de définition explicite de ce qu'est un vecteur court, mais il s'agit d'un critère qui doit normalement être évaluable en fonction des seuls paramètres k, n, ℓ, p du problème. Dans un premier temps, nous avons essayé d'estimer la taille caractéristique des coefficients des matrices considérées au doigt mouillé en fixant le seuil à $m k \ell^2 \sigma^3$, ce qui a bien fonctionné en pratique sur nos premiers tests. Une analyse empirique plus poussée est faite en 3.3.

Un problème pratique auquel nous avons fait face était le choix de la norme utilisée. En effet, utiliser une norme classique (euclidienne par exemple) était un mauvais choix puisqu'elle faisait artificiellement exploser la taille de nos objets. En effet, dans \mathbb{Z}_q , $0 - 1 = q - 1$. Pour pallier à ce problème, nous avons décidé d'utiliser plutôt la fonction suivante : $i \mapsto \min(i, q - i)$, qui correspond à la distance minimale à 0 dans \mathbb{Z}_q . Cette décision est purement intuitive et n'est à priori pas justifiée théoriquement, mais il s'avère que cela fonctionne très bien en pratique.

3.3 Tests

Différentes fonctions de l'implémentation ont été testées pour vérifier le bon fonctionnement du code sur quelques exemples, ainsi que pour évaluer l'influence du choix des paramètres. Quelques exemples de sortie des fichiers de tests sont disponibles directement sur le [repository GitHub](#) sans avoir à lancer soi-même le projet.

3.3.1 Correction

- `test_sampling` réalise des tests sur les fonctions d'échantillonnage uniforme et gaussien, calcule la moyenne et la variance empirique et les compare aux valeurs théoriques attendues (très proches).
- `test_circuit` s'assure que la relation $\mathbf{G} \times \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A}$ est vérifiée, se charge de calculer \mathbf{A}_f et $\mathbf{H}_{f,x,\tilde{\mathbf{A}}}$ (que nous avons vérifiées à la main sur quelques cas assez simples), et de vérifier la relation (3).
- `test_bgg` fait appel à `BGG.KeyGen` et `BGG.OfflineEnc`, s'assure que la relation $\mathbf{A}_f \mathbf{T}_f = \mathbf{A}_0$ est vérifiée par les clefs $\tilde{\mathbf{A}}$ et \mathbf{T}_f renvoyées et qu'il n'y a pas de problème à l'exécution de `BGG.OfflineEnc` pour différentes clés et valeurs du bit à chiffrer.
- `test_gen_circuit` s'assure de tester des fonctions auxiliaires qui permettent de construire des portes AND, OR et NOT à partir des portes NAND. Ces dernières sont les seules représentées directement par un circuit. On construit par exemple un circuit pour qui seul l'attribut x est autorisé à déchiffrer, et qui vérifie donc, étant donné un certain $x \in \llbracket 0, 2^k - 1 \rrbracket$,

$$\forall y \in \llbracket 0, 2^k - 1 \rrbracket \setminus \{x\}, f(y) = 1 \text{ et } f(x) = 0.$$

- `test_cp_bit` teste toutes les fonctions du CP-ABE pour un bit à chiffrer (puis à déchiffrer). Comme TrapSamp n'a pas été implementée en pratique, on utilise la trappe jointe \mathbf{T} (qui donne strictement plus d'informations). Il est alors possible de vérifier qu'il est bien possible de chiffrer un bit, puis pour un utilisateur ayant un attribut convenable de déchiffrer correctement ce bit. Le chiffrement et le déchiffrement fonctionnent sur des exemples simples. Les résultats montrent que, même en utilisant \mathbf{T} , et donc sans générer de trappe \mathbf{t}_x spécifique à l'attribut de l'utilisateur, un utilisateur non autorisé ne parvient pas à déchiffrer le message.

3.3.2 Choix des paramètres

Notre étude du choix des paramètres a été purement expérimentale, et nous ne nous sommes pas intéressés aux conséquences sur la sécurité du schéma.

Le premier objectif était de déterminer empiriquement la taille d'un vecteur court en fonction des paramètres. Les tests utilisés se trouvent dans le fichier `test_is_short`. Sauf indication contraire, les paramètres utilisés dans les tests qui suivent sont $n = 1$, $k = 30$, $q = 1073707009$, $\sigma = 7,00$ et des petits circuits comme $f(x_1, x_2, x_3) = 1 - (x_1 + x_2 x_3)$, où le *+* est à comprendre comme un *ou* binaire.

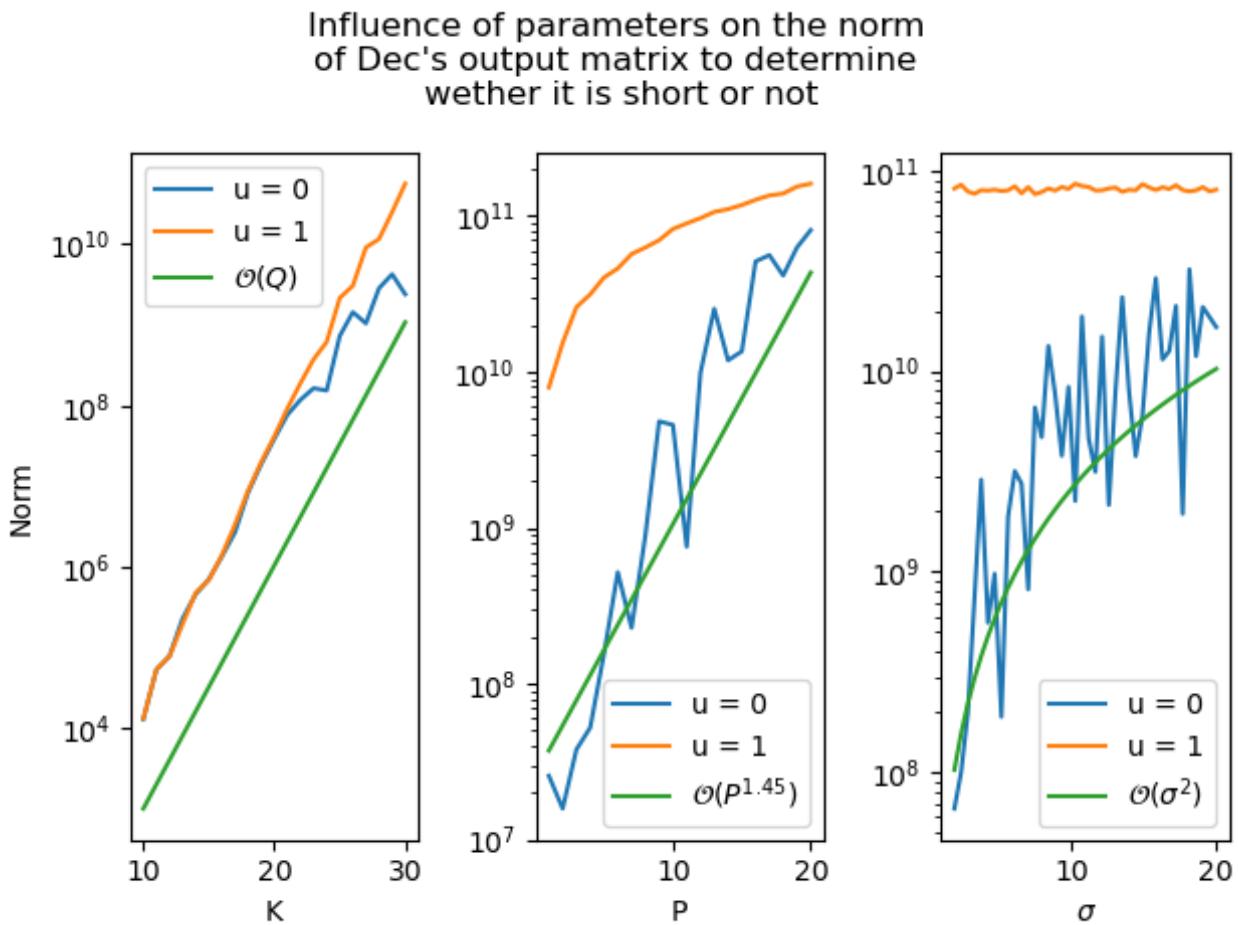


FIGURE 3 – Norme de la matrice renvoyée par Dec , selon la valeur des paramètres k , p et σ .

Les graphiques présentés en figure 3 conduisent donc à supposer que le seuil pour considérer qu'un vecteur est court est de l'ordre de $O(QP^{1.45}\sigma^2)$, ce qui fonctionne assez bien en pratique sur nos exemples simples. Nous avons observé des résultats optimaux sur une gamme variée de valeurs des

paramètres pour un seuil égal à $\frac{Q}{2} \times P^{1,45} \times \sigma^2 \times \frac{1}{10}$, le facteur $\frac{1}{10}$ ayant été obtenu par tâtonnement.

3.3.3 Analyse de performances

On teste les performances du schéma avec les paramètres utilisés précédemment et le circuit $f(x_1, x_2, x_3) = 1 - (x_1 + x_2 x_3)$. Si on imagine déployer le schéma dans une entreprise de développement informatique et qu'on introduit les attributs suivants :

- x_1 vaut 1 *ssi* l'utilisateur est administrateur,
- x_2 vaut 1 *ssi* l'utilisateur est développeur,
- x_3 vaut 1 *ssi* l'utilisateur travaille sur un projet fixé.

Le circuit s'interprète naturellement comme :

L'utilisateur a les droits *ssi* il est administrateur *ou* [est développeur et travaille sur le projet].

Les résultats ne sont pas éclatants : la fonction de chiffrement permet de chiffrer environ 75 octets par seconde, tandis que la fonction de déchiffrement traite seulement 3 octets par seconde. Le temps de déchiffrement ne dépend surprenamment pas de p . Il est en fait complètement dominé par les appels à `compute_H`.

Cette explosion du temps de calcul s'accompagne également d'une explosion de la norme de $\mathbf{H}_{f,x,\bar{\mathbf{A}}}$, ce qui empêche des utilisateurs ayant les droits de déchiffrer correctement le message. On remarque que les cas où l'utilisateur autorisé n'arrive pas à déchiffrer correctement correspondent aux cas où le circuit est le plus profond. Ci-dessous un exemple de sortie du programme de test :

```
Testing CP
Printing parameters...
N = 1
Q = 1073707009
K = 30
L = 30
P = 1
M = P + 2 = 3
SIGMA = 7.00
SHORT_THRESHOLD = 2.6e+09
attributes within range [0, 15]
Original message : Hello, world!
Cipher generated in 0.08839s
Message decrypted by an :
- unauthorized user : üÿÿÿÿÿÿÿÿÿÿÿÿ in 6.616s
- authorized user : Hello, world! in 6.6s
- unauthorized user : üÿÿÿÿÿÿÿÿÿÿÿÿ in 6.636s
- authorized user : Hello, world! in 6.678s
- unauthorized user : üÿÿÿÿÿÿÿÿÿÿÿÿ in 6.593s
- authorized user : Hello, world! in 6.599s
- authorized user : üÿÿÿÿÿÿÿÿÿÿÿÿ in 6.638s
- authorized user : Hello, world! in 6.647s
- unauthorized user : üÿÿÿÿÿÿÿÿÿÿÿÿ in 6.595s
- authorized user : Hello, world! in 6.61s
- authorized user : üÿÿÿÿÿÿÿÿÿÿÿÿ in 6.656s
- authorized user : Hello, world! in 6.651s
- unauthorized user : üÿÿÿÿÿÿÿÿÿÿÿÿ in 6.617s
- authorized user : Hello, world! in 6.614s
- authorized user : üÿÿÿÿÿÿÿÿÿÿÿÿ in 6.635s
- authorized user : Hello, world! in 6.643s
```

Cependant, le circuit utilisé précédemment est implémenté naïvement et il est donc possible d'utiliser un circuit optimisé moins profond équivalent. Nous avons adapté manuellement le circuit précédent, et cette version optimisée donne bien des résultats satisfaisants — et des performances déjà légèrement meilleures (*cf.* ci-dessous la sortie du programme de test avec le circuit optimisé manuellement).

```
Message decrypted by an :
- unauthorized user : yyyy-yyyy-yyyy-yyyy in 6.082s
- authorized user : Hello, world! in 6.104s
- unauthorized user : yyyy-yyyy-yyyy-yyyy in 6.115s
- authorized user : Hello, world! in 6.084s
- unauthorized user : yyyy-yyyy-yyyy-yyyy in 6.065s
- authorized user : Hello, world! in 6.08s
- authorized user : Hello, world! in 6.109s
- authorized user : Hello, world! in 6.095s
- unauthorized user : yyyy-yyyy-yyyy-yyyy in 6.074s
- authorized user : Hello, world! in 6.092s
- authorized user : Hello, world! in 6.103s
- authorized user : Hello, world! in 6.152s
- unauthorized user : yyyy-yyyy-yyyy-yyyy in 6.09s
- authorized user : Hello, world! in 6.097s
- authorized user : Hello, world! in 6.121s
- authorized user : Hello, world! in 6.11s
```

Ce travail d'optimisation de circuits représentés uniquement avec des portes NAND dépasse le cadre de notre PSC, mais constitue un enjeu majeur dans la réalisation pratique du schéma de chiffrement, aussi bien pour le bon fonctionnement que pour les performances.

Remarquons enfin que la taille des chiffrés en nombre de scalaires augmente en $O(mk\ell) = O(pnk^2)$, et les matrices $\mathbf{H}_{f,x,\tilde{\mathbf{A}}}$ utilisées voient leur norme augmenter en $O(k\ell^2) = O(n^2k^3)$ ce qui est en pratique très contraignant sur le choix de k et n .

4 Conclusion

Au cours de notre PSC, nous avons eu l'occasion de découvrir un nouveau domaine de l'informatique : la cryptographie. Il est indéniable que s'y initier nous a pris de longs mois, mais nous a également apporté une nouvelle maîtrise et de nouvelles connaissances dans cette discipline. Si le PSC a pour but d'obtenir une production finale, il a également pour but de nous faire découvrir le monde de la recherche, aussi bien dans sa dimension intellectuelle que dans sa dimension sociale. Ce projet a été l'occasion pour nous de travailler en groupe, et de chercher en groupe, chose que nous n'avions jamais encore eu l'occasion de faire dans le cadre d'un projet si vaste.

Par ailleurs, la production issue de notre PSC, présentée au cours de cet article, est également intéressante en elle-même. Bien que nous n'ayons pas atteint tous nos objectifs (qui étaient initialement de parvenir à terminer intégralement l'implémentation), nous sommes parvenus à construire une partie non négligeable du cryptosystème. Celui-ci a notamment été relativement bien compris, et décomposé afin de le rendre le plus intelligible possible. Ont été par exemple extraites les phases de *setup*, de *key generation*, d'*encrypting* et de *decrypting*. Chacune de ces étapes a également été détaillée et éclaircie, la littérature laissant quelquefois au lecteur la finalisation de certaines constructions.

Forts de cette compréhension (malgré des zones d'ombre persistantes), il a été possible d'implémenter une partie du cryptosystème. Nous sommes notamment parvenus à programmer la phase d'initialisation (*setup*) et de chiffrement (*encryption*) de notre CP-ABE, ainsi que l'ensemble des fonctions de BGG⁺-lite dont nous avions besoin. La phase de déchiffrement (*decryption*) est également presque complète.

Pour autant, notre cryptosystème n'est pas fonctionnel tel quel. Il ne nous a pas été possible d'implémenter la partie de génération de clef (la fonction `TrapSamp`), notamment en raison de difficultés théoriques rencontrées tardivement, qui nous ont fait manquer de temps. Dans la phase de déchiffrement, il nous manque également une définition théorique et précise de la fonction qui détermine si un vecteur est court (la fonction `is_short`). Celle-ci est nécessaire à la complémentation de la fonction `Dec`, et est donc manquante pour le bon fonctionnement de notre protocole de chiffrement.

De plus, bien que n'affectant pas le fonctionnement du cryptosystème, certains paramètres ont été choisis de telle sorte que le protocole de chiffrement fonctionne, mais pourraient être des freins à une utilisation pratique. Notre étude théorique n'a malheureusement pas pu être assez profonde pour déterminer les valeurs de paramètres qui soient utilisables dans le cadre d'un usage pratique. Les valeurs de ces paramètres peuvent également avoir une influence sur la sécurité du cryptosystème. Là encore, malgré une attention particulière, ainsi que des tests, tous les paramètres n'ont pas pu faire l'objet d'une étude suffisamment détaillée pour que leur valeur optimale soit déterminée.

Enfin, par delà l'implémentation du cryptosystème tel que présenté dans l'article de Z. BRAKERSKI et V. VAIKUNTANATHAN [BV22], nous avions initialement l'ambition d'en proposer une version fondée sur les modules de polynômes, et non sur les scalaires. Les progrès récents dans ce domaine (voir [BEP⁺21]), ainsi que le travail de L. PRABEL sur le sujet, fournissant dans le même temps des implémentations en C de certaines fonctions auxiliaires, nous ont fait espérer une possible adaptation du protocole [BV22] à un espace de modules de polynômes. Pour autant, nous avons fondamentalement manqué de temps pour nous y plonger et proposer une étude théorique complète, et une implémentation correspondante.

Dès lors, notre travail appelle à être continué. Si l'implémentation et l'étude sont bien avancées, il reste à finir le code, et notamment la partie de génération de clef. Par ailleurs, la perspective d'une adaptation à des modules de polynômes semble prometteuse, et mérirait que l'on s'y penche plus longuement pour en proposer une étude détaillée.

Références

- [Ajt96] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 99–108, New York, NY, USA, 1996. Association for Computing Machinery.
- [BDK⁺18] Joppe Bos, Leo Ducas, Eike Kiltz, T Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. Crystals - kyber : A cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 353–367, 2018.
- [BEP⁺21] Pauline Bert, Gautier Eberhart, Lucas Prabel, Adeline Roux-Langlois, and Mohamed Sabt. Implementation of lattice trapdoors on modules and applications. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography*, pages 195–214, Cham, 2021. Springer International Publishing.
- [BGG⁺14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit abe and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, pages 533–556, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
- [BV22] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-inspired broadcast encryption and succinct ciphertext-policy abe. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPICS*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- [DDP⁺18] Wei Dai, Yarkın Doröz, Yuriy Polyakov, Kurt Rohloff, Hadi Sajjadpour, Erkay Savaş, and Berk Sunar. Implementation and evaluation of a lattice-based key-policy abe scheme. *IEEE Transactions on Information Forensics and Security*, 13(5) :1169–1184, 2018.
- [DKL⁺18] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium : A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1) :238–268, Feb. 2018.
- [FHK⁺19] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Por-nin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon : Fast-fourier lattice-based compact signatures over NTRU. 2019.
- [Gen09] Craig Gentry. A fully homomorphic encryption scheme. 2009.
- [GPV07] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. 2007.
- [GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. *Journal of the ACM (JACM)*, 62 :1 – 33, 2013.
- [JTC22] Zulianie Binti Jemihin, Soo Fun Tan, and Gwo-Chin Chung. Attribute-based encryption in securing big data from post-quantum perspective : A survey. *Cryptography*, 6(3), 2022.
- [LLS14] Fabien Laguillaumie, Adeline Langlois, and Damien Stehlé. Chiffrement avancé à partir du problème learning with errors. 2014.
- [NIS16] Notice by the NIST. Announcing request for nominations for public-key post-quantum cryptographic algorithms. pages 92787–92788, 20/12/2016.
- [Pra21] Lucas Prabel. Module gaussian lattice. https://github.com/lucasprabel/module_gaussian_lattice/tree/main, 2021.

- [Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 84–93, New York, NY, USA, 2005. Association for Computing Machinery.
- [RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2) :120–126, feb 1978.
- [Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5) :1484–1509, 1997.