Université Paris 7

Interrogation Confidentielle de Bases de Données

Malika Izabachène

Rapport de Stage de Master MPRI2

Laboratoire d'accueil : Département d'Informatique de l'École Normale Supérieure de Paris

Sous la direction de : M. Michel Abdalla et de M. David Pointcheval

Date du stage : Mars-Juillet 2006

Résumé du Stage :

L'interrogation confidentielle de bases de données est une problématique qui peut être introduite de la manière suivante : un utilisateur possède un indice i, il souhaite retrouver la valeur de x_i dans une base de données sans transmettre la moindre information sur ce qu'il cherche. On s'intéresse au coût de communication entre les deux participants : l'utilisateur et le serveur. On cherche à construire un protocole interactif entre les deux joueurs qui consiste en une succession de requêtes et réponses : l'utilisateur envoie des requêtes à la base de données. Celle-ci renvoie la réponse correspondante qui doit permettre à l'utilisateur de reconstituer la valeur de x_i, tout en préservant la confidentialité de sa requête.
La première solution qui vient à l'esprit est que la base de données ion transmet tout ce qu'elle contient. On cherche bien évidemment une meilleure solution. Le but des divers travaux réalisés ces dernières années consistait à optimiser le nombre de bits transmis entre les deux parties.
Nous étudierons une primitive introduite pour l'étude de cette problématique appelée, une *Private Information Retrieval, PIR*. L'intérêt porté à cette primitive se justifie d'autant plus d'un point de vue pratique. En effet, il peut être très commode pour un utilisateur d'avoir accès à des données confidentielles stockées dans un serveur. Ce rapport se divise en deux parties une première qui présente quelques protocoles d' Information theoretic PIR et une deuxième partie qui étudie les *Computational PIR*, l'objectif étant de généraliser l'usage des protocoles de PIR aussi bien d'un point de vue théorique que pratique.

Remerciements :

Avant tout, j'aimerais remercier Michel et David, mes deux encandrants qui ont su se montrer patient, agréable et sympatique. Tout d'eux ont réfléchi aux différentes questions qui se posaient et ont apporté des suggestions intéressantes qui sont la source des principales idées développées dans ce rapport. De plus, je dois reconnaître qu'ils m'ont accompagné et soutenu dans mes moments de doute tout au long de mon stage.

D'autre part, je suis particulièrement reconnaissante aux directeurs d'équipe et de laboratoire, M. David Pointcheval et M. Jacques Stern qui m'ont permis de faire ce stage au sein d'une équipe exceptionnelle. Je me vois d'ailleurs ravie de pouvoir continuer ma thèse sous la direction de M. Pointcheval.

Pour terminer, un grand merci aux compétences et au sourir de chacun des membres, merci pour l'aide précieuse des thésards, merci aux autres stagiaires pour leur accompagnement et leur sympathie et enfin merci au personnel administratif qui a su se montrer aimable et serviable.

Table des matières

Chapitre 1

Introduction aux PIR

Dans le modèle des PIR, le niveau de sécurité que l'utilisateur souhaite atteindre doit être précisé : il ne veut transmettre aucune information sur ce qu'il cherche mais aussi sur ce qu'il ne cherche pas. Par exemple, la base de données ne doit pas connaître la valeur de i mais ne doit pas non plus apprendre une information du type $i \neq 59$. On pourra aussi envisager le cas où l'utilisateur n'apprend rien d'autres de plus que le bit demandé. On parlera alors de *Symmetric PIR*, noté $SPIR$. On distinguera dans ce cas :

- le cas où l'utilisateur est honnête mais curieux, i.e., qu'il suit le protocole mais qu'il cherche à récupérer des informations supplémentaires.
- puis, le cas où l'utilisateur est malhonnête.

On précisera par la suite (suivant le modèle et les moyens donnés aux deux participants) le niveau de sécurité que l'on souhaite atteindre pour chacun d'eux.
On pourra supposer que l'information contenue dans la base de données est une chaîne de bits de la forme x= $x_1, \ldots, x_n$. On distingue deux catégories de PIR selon les moyens donnés à un attaquant potentiel : ceux connus sous le nom de *Information-theoretic PIR*, que l'on notera $ITPIR$ ou simplement PIR pour lesquels les bases de données ont une puissance de calcul illimitée ; et les *Computational PIR*, qu'on notera $CPIR$ pour lesquelles la confidentialité de la requête de l'utilisateur repose sur des hypothèses calculatoires. Pour ces derniers schémas, la puissance de calcul des bases de données est polynomiale.

1.1 Notations utilisées

n	taille de la base de données
$\log_2 n$	logarithme en base 2 de n.
U	utilisateur
DB	base de données
x	contenu de la base de données
i	indice secret de l'utilisateur
x_i	élément recherché par U dans x.

1.2 Résultats généraux

- Dans la cas des $ITPIR$, il a été démontré dans [9], que si on se restreint à une seule base de données, la complexité en terme de bits transmis est forcément linéaire. Ce qui a amené Chor. et al. à envisager un schéma avec réplication des bases de données. C'est le seul moyen pour cette catégorie de PIR, d'obtenir une complexité sous-linéaire. Dans

ce cas, on supposera que les bases de données ne communiquent pas entre elles, afin de préserver la confidentialité de la requête de l'utilisateur. Par la suite, le nombre de bases de données sera noté k avec $k \geqslant 2$.

– Un schéma de PIR est un algorithme probabiliste où seul l'utilisateur a accès à de l'aléa. On supposera qu'il obtient toujours la bonne réponse à sa requête. Pour les $SPIR$, l'algorithme de réponse des bases de données est aussi probabiliste. Différents travaux réalisés par Chor et al., Ambainis, Ashai et Kushilevitz, Beimel et al. conduisent aux résultats de la figure 1.1.

Outils utilisés	Référence	2 BD	Complexité
Racine $k_{\text{ième}}, k \geqslant 4$	[9]	pas de PIR	$k \cdot n^{\frac{1}{\log k}}$
Codes couvrants	[9]	$n^{\frac{1}{3}}$	$(k \cdot \log_2 k) n^{\frac{1}{\log_2 k + \log_2 \log_2 k}}$
Interpolation poly	[9]	$n^{\frac{1}{3}}$	$(k^2 \cdot \log_2 k) \cdot n^{\frac{1}{k}}$
Récurrence	[2]	$n^{\frac{1}{3}}$	$2^{k^2} \cdot n^{\frac{1}{2k-1}}$
Alg. linéaire	[3], [12]	$n^{\frac{1}{3}}$	$k^3 \cdot n^{\frac{1}{2k-1}}$
Poly-heavy	[5]	$n^{\frac{1}{3}}$	$n^{O(\frac{\log_2 \log_2 k}{k \log_2 k})}$

FIG. 1.1 – Résultat généraux pour les $ITPIR$: ici, n est la longueur de la base de données et k le nombre de bases de données.

– On peux améliorer la complexité si on limite la puissance de calcul des bases de données. On restreint le temps de calcul des bases de données qui devient alors polynomial. Dans ce cas, la confidentialité de la requête de l'utilisateur repose sur une hypothèse calculatoire qui détermine la construction du schéma. Dans ce cas, on veut s'assurer qu'après avoir répondu à la requête de l'utilisateur, les bases de données ne peuvent rien calculer sur i. On parle alors de *Computational PIR*. Cette restriction a tout de même l'avantage de permettre la construction de schémas avec une seule base de données. En effet, Kushilevitz et Ostrovsky [13] ont proposé un schéma avec une base de données et une complexité en $O(n^\epsilon)$ pour tout $\epsilon > 0$. Ce schéma repose sur la difficulté de déterminer la résiduosité quadratique. D'autres schémas de $CPIR$ ont été proposés, tous ayant toujours pour objectif d'optimiser le nombre de bits transmis. On peux résumer les résultats généraux pour ce type de schémas par le tableau suivant :

Hypothèse	Référence	Nombres DB	Complexité
Résiduosité quadratique	[13]	1	$O(n^\epsilon)$ avec ϵ petit
Problème ϕ-Hiding	[6]	1	$O((log_2 n)^a)$ a paramètre de sécurité
Existence d'une permutation à trappe	[14]	1	$n - O(n)$
Haute résiduosité	[7]	1	$O(n^\epsilon \log_2 n)$ avec ϵ petit

FIG. 1.2 – Résultat généraux pour les $CPIR$.

1.3 Définitions générales

Il existe plusieurs variantes de PIR dans la littérature. On donnera les principales définitions et notations dont on se servira.

Un schéma de PIR à un tour avec $x \in \{0,1\}^n$ et k bases de données est un schéma de récupération confidentielle d'information tel que, après que la requête ait été effectuée et traitée, aucune des bases de données n'obtient la moindre information sur i. On suppose que la puissance de calcul des bases de données est illimitée. Pour cette catégorie de PIR, on supposera qu'il y a des copies multiples des bases de données qui ne communiquent pas entre elles afin de garantir la confidentialité de l'utilisateur.

Définition 1.1 ($k-DB$ Private Information Retrieval à un tour) *Pour $x \in \{0,1\}^n$ et k BD, un schéma de k-BD à un tour a la forme suivante :*

1. *U veut connaître x_i. Il y a k copies des bases de données DB. On suppose qu'elles ne peuvent pas communiquer entre elles.*
2. *U choisit N aléatoirement, $N \in \{0,1\}^*$. U calcule la requête $q_1, \ldots, q_k$ à l'aide de N. Il envoie q_j à DB_j, pour $j = 1, \ldots, k$.*
3. *$\forall j \in [k]$, DB_j renvoie la réponse correspondant à la requête q_j, $ANS_j(q_j)$.*
4. *U calcule x_i à partir de i, N et de $ANS_j(q_j)$.*

La complexité en terme de communication d'un tel protocole est définie par :

$$\sum_{j=1}^{k} |q_j| + |ANS_j(q_j)|.$$

Un schéma de $CPIR$ à un tour avec $x \in \{0,1\}^n$ et k, le nombre de bases de données est un schéma de récupération confidentielle d'information tel que, après que la requête ait été effectuée et traitée, aucune base de données n'obtient la moindre information sur i. On suppose, en revanche que la puissance de calcul des bases de données est limitée. On doit s'assurer pour cela que les limites fixées pour chaque base de données ne leur permet pas de calculer la moindre information sur i.

Un *Symmetric PIR* est un schéma qui satisfait les propriétés suivantes pour lequel la confidentialité est assurée pour les deux parties, i.e., l'utilisateur fait une requête à la base de données de telle sort que celle-ci n'apprend pas le bit demandé comme pour les PIR. De plus, l'utilisateur n'apprend pas plus d'un bit d'information.

Chapitre 2

Information-Theoretic PIR

2.1 Schémas de sommations linéaires

Les premiers schémas que nous présenterons sont pour le moment les plus efficaces pour des petites valeurs de k, particulièrement pour le cas où $k = 4$. On peut noter que le nombre de bits transmis dans ce protocole pour chacun des deux participants n'est pas équilibré. Le schéma qui suit permet d'équilibrer ce nombre de bits, ce qui aura pour effet de diminuer la complexité de communication. Une technique plus générique sera présentée plus loin.

2.1.1 Un schéma basique multi-bases de données

On présente dans cette partie un schéma avec $k = 2^d$ bases de données aboutissant à une complexité de communication égale à $O(n^{\frac{1}{d}})$ bits. L'idée clé de la preuve d'existence d'un tel schéma consiste à représenter $x = x_1, \ldots, x_n$ comme un hypercube de dimension d et d'exploiter les propriétés de linéarité du XOR. Plus précisément, chaque élément est indicé par $(i_1, \ldots, i_d)$ avec $i_j \in \{0, 1\}$

Le cas trivial

On commencera par présenter le cas trivial où $d = 1$, i.e., $k = 2^1$ bases de données. C'est un schéma trivial, i.e., que la complexité est linéaire. Cela dit, il nous permettra de mieux comprendre des schémas sous-linéaires pour $d \geqslant 3$.

Notations :

- On note pour x une chaîne de bits et i un indice tel que $i \leqslant |x|$, $x \oplus \mathbb{1}_i$ la chaîne de bits flippée à la position i, où $\mathbb{1}_i$ est la chaîne comportant exactement un seul 1 à la position i.
- On note respectivement DB_0 et DB_1 les 2 bases de données.

Description du protocole

1. U veut retrouver le bit x_i.
2. U génère aléatoirement une chaîne de bits $\mathbf{S} \in \{0, 1\}^n$.
3. U génère une chaîne additionnelle **S'**définie par $\mathbf{S'} = \mathbf{S} \oplus \mathbb{1}_i$.
4. U envoie une chaîne différente à chacune des bases de données : Par exemple DB_0 reçoit $\mathbf{S}$ et DB_1 reçoit **S'**. À cette étape, l'utilisateur envoie n bits à chaque base de données.

5. Chacune des bases de données répond par le ou-exclusif de tous les bits de x, indicés par les indices des occurrences de 1 de la séquence aléatoire qu'elle a reçue. Plus précisément, DB_0 envoie $\oplus_{S_j=1} x_j$, ce qui correspond à envoyer $x \cdot \mathbf{S}$ sur $GF(2)$. Et DB_1 envoie $\oplus_{S'_j=1} x_j$, i.e., $x \cdot \mathbf{S'}$. Les bases de données envoient ici 2 bits, i.e., 1 chacune.
6. U retrouve x_i, en faisant le ou-exclusif des deux bits qu'elle a reçus. En effet, les chaînes aléatoires $\mathbf{S}$ et $\mathbf{S'}$ ne diffèrent qu'à la position i. Donc les bits s'annulent deux à deux et il ne reste plus que x_i.

Complexité

La complexité de communication est donc linéaire en n ; ce qui n'est pas mieux que la solution triviale au problème introduit. Mais sa représentation dans $\mathbb{F}_2$ est assez naturelle et son fonctionnement, assez simple à comprendre. D'autre part, ce schéma se rapproche plus d'un $SPIR$ que le schéma trivial qui consiste à envoyer toute la base de données.

Pour la complexité calculatoire, les opérations sont uniquement des additions modulo 2 : les bases de données doivent parcourir la totalité de leur contenu, elles XORent certaines valeurs de bits de x donc au plus n. L'utilisateur, quant à lui fait 2 additions modulo 2.

un schéma en $O(\sqrt{n})$

À présent, nous allons présenter une solution dans le même esprit pour le cas $d = 2$. On obtient pour ce schéma une complexité en $O(\sqrt{n})$; ce qui est proche du meilleur résultat connu pour 2 bases de données en $O(n^{\frac{1}{3}})$. Cette solution sera présentée juste après.

Notations :
- La base de données de taille n est vue comme une matrice de taille $\sqrt{n} \times \sqrt{n}$. On notera X cette matrice. Chaque indice $j \in [n]$ sera représenté par $< j_1, j_2 >$, où $j_1, j_2 \in [\sqrt{n}]$. Cela peut être fait de manière naturelle par une fonction bijective de $\{0,1\}^n \mapsto (\{0,1\}^{\sqrt{n}})^2$.
- On note respectivement $DB_{00}, DB_{01}, DB_{10}, DB_{11}$ les 4 bases de données.
- On note $\mathbf{S_0}, \mathbf{S_1} \in \{0,1\}^{\sqrt{n}}$ deux vecteurs colonnes de dimension $\sqrt{n}$. avec cette notation, $S_{0,j}$ correspond à la $j_{\text{ième}}$ composante du vecteur $\mathbf{S_0}$.
  La base de données X est vue comme une matrice carré X de taille $\sqrt{n}$. Autrement dit, pour tout $j = 1, \ldots, \sqrt{n}$, $X_{*,j}$ correspond à la $j_{\text{ième}}$ colonne de la matrice, i.e., $X_{*,j}$ est un vecteur colonne de taille $\sqrt{n}$.
  La notation $\oplus_{S_0(j_1)=1, S_1(j_2)=1} X_{j_1,j_2}$ est la somme dans $GF(2)$ des bits de X dont les deux positions correspondent aux indices pour lesquels $S_0(j_1) = 1$ et $S_1(j_2) = 1$. Cette notation étant assez difficile à manipuler, on préférera utiliser le produit scalaire dans $GF(2)$. Ce qui revient alors à calculer $\mathbf{S_0^T} \cdot X \cdot \mathbf{S_1}$.

Description du protocole

1. On suppose que U veut retrouver le bit X_{i_1,i_2}.
2. U génère aléatoirement deux chaînes de bits $S_0, S_1 \in \{0,1\}^{\sqrt{n}}$.
3. U génère deux chaînes additionnelles S'_0, S'_1 définies par :
$$\mathbf{S'_0} = \mathbf{S_0} \oplus \mathbb{1}_{i_1} \text{ et } \mathbf{S'_1} = \mathbf{S_1} \oplus \mathbb{1}_{i_2}.$$
4. U envoie deux chaînes à chaque base de données :
   DB_{00} reçoit $\mathbf{S_0}$ et $\mathbf{S_1}$.

DB_{01} reçoit $\mathbf{S_0}, \mathbf{S'_1}$.
DB_{10} reçoit $\mathbf{S'_0}, \mathbf{S_1}$.
DB_{11} reçoit $\mathbf{S'_0}, \mathbf{S'_1}$.
Ici, U envoie donc $2 \cdot \sqrt{n}$ bits à chaque base de données.

5. DB_{00} renvoie $r_1 \stackrel{\text{déf}}{=} \oplus_{S_0(j_1)=1, S_1(j_2)=1} X_{j_1,j_2} = \mathbf{S_0^T} \cdot X \cdot \mathbf{S_1}$.
   DB_{01} renvoie $r_2 \stackrel{\text{déf}}{=} \oplus_{S_0(j_1)=1, S'_1(j_2)=1} X_{j_1,j_2} = \mathbf{S_0^T} \cdot X \cdot \mathbf{S'_1}$.
   DB_{10} renvoie $r_3 \stackrel{\text{déf}}{=} \oplus_{S'_0(j_1)=1, S_1(j_2)=1} X_{j_1,j_2} = \mathbf{S_0'^T} \cdot X \cdot \mathbf{S_1}$.
   Et enfin, DB_{11} renvoie $r_4 \stackrel{\text{déf}}{=} \oplus_{S'_0(j_1)=1, S'_1(j_2)=1} X_{j_1,j_2} = \mathbf{S_0'^T} \cdot X \cdot \mathbf{S'_1}$.

   À cette étape, les bases de données envoient 4 bits i.e., un bit chacune.

6. U calcule $\oplus_{j=1}^{4} r_j$. Et il retrouve le bit X_{i_1,i_2}.

Vérifions d'abord sur un exemple que l'utilisateur reçoit bien ce qu'il veut. Afin de mieux voir ce qu'il se passe, prenons un exemple . Si on prend $i = (2,3)$ et $n = 16$, on a alors $\mathbf{S_0}, \mathbf{S_1}, \mathbf{S'_0}$ et $\mathbf{S'_1} \in \{0,1\}^4$.

Par exemple, $S_0 = \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline 1 \\ \hline 1 \\ \hline \end{array}$ et $S_1 = \begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline 1 \\ \hline 0 \\ \hline \end{array}$.

On a alors $S'_0 = \begin{array}{|c|}\hline 0 \\ \hline 1 \\ \hline 1 \\ \hline 1 \\ \hline \end{array}$ et $S'_1 = \begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline 0 \\ \hline 0 \\ \hline \end{array}$.

DB_{00} envoie donc $B_{00} \stackrel{\text{déf}}{=} X_{1,3} \oplus X_{1,4} \oplus X_{2,3} \oplus X_{2,4} \oplus X_{3,3} \oplus X_{3,4}$,
DB_{01} envoie $B_{01} \stackrel{\text{déf}}{=} X_{1,3} \oplus X_{1,4} \oplus X_{2,3} \oplus X_{2,4}$,
DB_{10} envoie $B_{10} \stackrel{\text{déf}}{=} X_{1,2} \oplus X_{1,3} \oplus X_{1,4} \oplus X_{2,2} \oplus X_{2,3} \oplus X_{2,4} \oplus X_{3,2} \oplus X_{3,3} \oplus X_{3,4}$,
et DB_{11} envoie $B_{11} \stackrel{\text{déf}}{=} X_{1,2} \oplus X_{1,3} \oplus X_{1,4} \oplus X_{1,3} \oplus X_{2,2} \oplus X_{2,3} \oplus X_{2,4}$.
On remarque que chaque bit apparaît deux ou quatre fois sauf $X_{2,3}$ qui apparaît un nombre impair de fois. Lorsque U XORe les quatre bits qu'il reçoit, il obtient alors $X_{2,3}$.
De manière générale, on peut remarquer que tous les bits apparaissent dans le XOR final de l'utilisateur un nombre pair de fois sauf le bit désiré.

Correction

Il s'agit de montrer que l'utilisateur retrouve le bit désiré en calculant $r_1 \oplus r_2 \oplus r_3 \oplus r_4$:

$$\begin{aligned}
\oplus_{k=1}^{4} r_k &= S_0^T \cdot X \cdot S_1 \oplus\ S_0^T \cdot X \cdot S'_1\ \oplus\ S_0'^T \cdot X \cdot S_1\ \oplus\ S_0'^T \cdot X \cdot S'_1 \\
&= S_0^T \cdot X \cdot S_1\ \oplus\ (S_0^T \cdot X \cdot (S_1 \oplus \mathbb{1}_{i_2}))\ \oplus\ ((S_0 \oplus \mathbb{1}_{i_1})^T \cdot X \cdot S_1)\ \oplus\ ((S_0 \oplus \mathbb{1}_{i_1})^T \cdot X \cdot (S_1 \oplus \mathbb{1}_{i_2})) \\
&= S_0^T \cdot X \cdot S_1\ \oplus\ (S_0^T \cdot X \cdot S_1\ \oplus\ S_0^T \cdot X \cdot \mathbb{1}_{i_2})\ \oplus\ (S_0^T \cdot X \cdot S_1\ \oplus\ \mathbb{1}_{i_1}^T \cdot X \cdot S_1) \\
&\oplus (S_0^T \cdot X \cdot S_1\ \oplus\ S_0^T \cdot X \cdot \mathbb{1}_{i_2}\ \oplus\ \mathbb{1}_{i_1}^T \cdot X \cdot S_1\ \oplus\ \mathbb{1}_{i_1}^T \cdot X \cdot \mathbb{1}_{i_2}) \\
&= \mathbb{1}_{i_1}^T \cdot X \cdot \mathbb{1}_{i_2} = X_{i_1,i_2}.
\end{aligned}$$

Complexité

Le nombre de bits envoyés dans ce protocole est $8 \cdot \sqrt{n} + 4$. Si on s'intéresse à la complexité calculatoire, on remarque que chaque base de données fait le ou-exclusif d'au plus n valeurs en parcourant conjointement la matrice X, (vue composante par composante) et les deux séquences aléatoires de longueur $\sqrt{n}$ que la base de données a reçues. L'utilisateur, quant à lui fait 4 additions modulo 2.

remarque 2.1 *Dans ce protocole, on voit que* $\mathbf{S_0}, \mathbf{S_1}, \mathbf{S'_0}$ *et* $\mathbf{S'_1}$ *permettent de masquer l'information que U souhaite récupérer. La valeur du bit* X_{i_1,i_2} *est indépendante de chacun d'eux. Mais si les bases de données viennent à communiquer entre elles, elles obtiennent immédiatement plus d'informations sur i, en sachant que chacun des deux couples ne diffèrent seulement d'un bit. Quelle que soit la valeur de* X_{i_1,i_2}*, U calcule la même position du bit grâce au codage utilisé, i.e.,* $\oplus B_{i,j}$*, pour* $i, j \in \mathbb{F}_2$ *:*

$$\begin{aligned} B_{00} &= \oplus_{S_0(j_1)=1, S_1(j_2)=1} X_{j_1,j_2}, \\ B_{01} &= B_{00} \oplus (\oplus_{S_0(j_1)=1} X_{i_1,j_2}), \\ B_{10} &= B_{00} \oplus (\oplus_{S_1(j_2)=1} X_{j_1,i_2}), \\ B_{11} &= B_{00} \oplus B_{01} \oplus B_{10} \oplus X_{i_1,i_2}. \end{aligned}$$

remarque 2.2 *La preuve de correction de ce schéma se vérifie avec la dernière remarque de l'étape* 5*. Pour la preuve de sécurité, il suffit d'observer que chaque base de données reçoit une séquence de bits choisie indépendemment et uniformément dans* $\in \{0,1\}^{\sqrt{n}}$*. De ce fait, les requêtes effectuées aux bases de données ont la même distribution pour tous les couples* (i, j)*.*

2.1.2 Schéma de *Codes Couvrants*

Cette méthode aboutit à la même complexité que le schéma précédent mais réduit le nombre de bases de données impliquées ; il est intéressant pour des petites valeurs de k.
Considèrons le schéma précédent pour $k = 8$ noté **P**. Les 8 bases de données sont indicés par les mots de longueur 3 dans {0,1}, i.e., $DB_{000}, DB_{001}, DB_{010}, DB_{100}, DB_{011}, DB_{110}, DB_{101}, DB_{111}$. On construit un protocole **P'** à partir du protocole **P** ayant la même complexité avec seulement deux bases de données. On va commencer par introduire quelques notations supplémentaires puis nous décrirons le protocole **P** à partir duquel on construira ensuite **P'**.

Notations

- On suppose qu'il existe un entier m tel que $n = m^3$, on considérera la fonction bijective $\{0,1\}^{m^3} \mapsto (\{0,1\}^m)^3$. On représente le contenu des bases de données $x = x_1, \ldots, x_n$, comme un cube de taille $m \times m \times m$. On notera X ce cube. L'utilisateur U veut retrouver l'indice $i =< i_1, i_2, i_3 >$ avec $i_r \in [m]$ pour $r = 1, 2, 3$.
- La notation qui va être introduite ici va nous permettre d'étendre le produit scalaire en dimension 3. Soient S_1, S_2, S_3, trois vecteurs $\in \{0,1\}^m$. On définit le cube $\overline{S} \stackrel{\text{déf}}{=} S_1 \times S_2 \times S_3 = (\overline{S})_{i,j,k}$ avec $i, j, k \in [m]$ tel que :

  $$\overline{S}_{i,j,k} = 1 \text{ si et seulement si } S_{1,i} = S_{2,j} = S_{3,k} = 1.$$

  Avec ces notations, on peut alors définir le produit de X et de $\overline{S}$ dans $GF(2)$ qu'on notera

  $$< X, \overline{S} > \stackrel{\text{déf}}{=} \sum_{i \in [m]} \sum_{j \in [m]} \sum_{k \in [m]} S_{i,j,k} \times X_{i,j,k} \mod 2.$$

- X_{i_1,i_2,i_3} représente le bit du cube que U veut retrouver.

Description du protocole P

1. U génère aléatoirement trois chaînes de bits $S_1, S_2, S_3 \in \{0,1\}^m$.
2. U génère trois chaînes additionnelles S'_1, S'_2, S'_3 définies par :

$$S'_1 = S_1 \oplus \mathbb{1}_{i_1}, S'_2 = S_2 \oplus \mathbb{1}_{i_2} \text{ et } S'_3 = S_3 \oplus \mathbb{1}_{i_3}.$$

On pose :

$$\begin{aligned}
\overline{S}_1 &\stackrel{\text{déf}}{=} S_1 \times S_2 \times S_3,\\
\overline{S}_2 &\stackrel{\text{déf}}{=} S_1 \times S_2 \times S'_3,\\
\overline{S}_3 &\stackrel{\text{déf}}{=} S_1 \times S'_2 \times S_3,\\
\overline{S}_4 &\stackrel{\text{déf}}{=} S'_1 \times S_2 \times S_3,\\
\overline{S}_5 &\stackrel{\text{déf}}{=} S_1 \times S'_2 \times S'_3,\\
\overline{S}_6 &\stackrel{\text{déf}}{=} S'_1 \times S_2 \times S'_3,\\
\overline{S}_7 &\stackrel{\text{déf}}{=} S'_1 \times S'_2 \times S_3,\\
\overline{S}_8 &\stackrel{\text{déf}}{=} S'_1 \times S'_2 \times S'_3.
\end{aligned}$$

U envoie trois des chaînes engendrées à chacune des bases de données :
DB_{000} reçoit (S_1, S_2, S_3), DB_{001} reçoit (S_1, S_2, S'_3),
DB_{010} reçoit (S_1, S'_2, S_3), DB_{100} reçoit (S'_1, S_2, S_3),
DB_{011} reçoit (S_1, S'_2, S'_3), DB_{101} reçoit (S'_1, S_2, S'_3),
DB_{110} reçoit (S'_1, S'_2, S_3) et DB_{111} reçoit (S'_1, S'_2, S'_3).
L'utilisateur envoie à cette étape $8 \cdot 3 \cdot m$.

3. Chaque base de données répond en renvoyant un bit correspondant au XOR des bits de X, indicé par les indices correspondant aux occurrences de 1 dans les trois chaînes envoyées par l'utilisateur. Plus précisément, pour $a, b, c \in [m]$:

$$DB_j \text{ calcule } a_j =< \overline{S}_j, X > .$$

Chaque base de données envoie donc un bit i.e., 8 bits au total.

4. U XORe les 8 bits qu'il reçoit i.e., $\oplus_{j\in[8]} a_j$, il s'agit de la réponse attendue.

Correction

similaire à la section 2.1.1.

Définition 2.1 (Covering Codes) *Un Covering Code, C_d est une collection d'éléments $C_d = \{c_1, \ldots, c_k\} \subseteq \{0,1\}^d$ telle que les boules de rayons 1 du code couvrent l'espace tout entier, i.e., $\{0,1\}^d \subseteq \cup_{c_j \in C_d} B(c_j, 1)$, où $B(c,1)$ est l'ensemble contenant tous les mots de longueur d qui diffèrent d'un bit avec c.*

Soit d la dimension utilisée pour la longueur des indices des bases de données. Ici, $d = 3$. La propriété ci-dessus repose sur la capacité des élément w à couvrir l'espace $\{0,1\}^d$, où w parcourt tout le code. Autrement dit, tout les mots ne différant au plus que d'une position avec w appartiennent à la même boule que w. Si l'union de toutes les boules est disjointe, alors le code ainsi

construit est dit parfait. Ce problème est équivalent à celui de la théorie des codes qui consiste à couvrir l'espace $\{0,1\}^d$ par des boules de rayon 1. Pour notre exemple, on peut déjà remarquer que l'espace $\{0,1\}^3$ est engendré par les deux mots 000 et 111. Il est facile de voir que $\{000, 111\}$ est un *Covering Code*. Les 8 mots de l'espace sont :$\{000, 001, 010, 100, 011, 101, 110, 111\}$. Les 4 premiers appartiennent à la même boule de centre 000 et de rayon 1. Et les 4 derniers appartiennent à la boule de centre 111 et de rayon 1. Les deux boules couvrent bien l'espace $\{0,1\}^3$. À partir de ces remarques, on va construire un PIR à partir du schéma de base à $2^3 = 8$ bases de données. On diminuera ainsi le nombre de bases de données.

On utilisera les deux bases de données indicées par les éléments de poids « extrême » i.e., DB_{000} et DB_{111} pour simuler le calcul des 6 autres bases de données à distance exactement 1 de l'une de ces deux bases de données. DB_{000} connaît deux des chaînes parmi les trois que les bases de données à distance 1 de $(0,0,0)$ ont reçu dans $\mathbf{P}$. Et de même DB_{111} connaît deux des chaînes connues par les trois bases de données à distance 1 de $(1,1,1)$. DB_{000} simule donc les trois bases de données en question en générant pour chacune les m possibilités pour chacune des trois ; chaque simulation consiste donc à exécuter le protocole $\mathbf{P}$avec trois chaînes, où l'une d'entre elles possède un bit flippé différent pour chaque simulation.

Notations : On note $S_1(j) \stackrel{\text{déf}}{=} S_1 \oplus \mathbb{1}_j$. $S_2(j)$ et $S_3(j)$ sont définis de manière similaire. On notera de la même manière $a_{r,j} \stackrel{\text{déf}}{=} < \overline{S}_r(j), X >$, pour $r = 1, 2, 3$.
En supposant que U veut toujours retrouver le bit X_{i_1,i_2,i_3} et en adoptant les mêmes notations que celles introduites pour le schéma précédent, on obtient le protocole **P'** suivant :

Description du protocole P'

1. U génère aléatoirement trois chaînes de bits $S_1, S_2, S_3 \in \{0,1\}^m$.
2. U génère trois chaînes additionnelles S'_1, S'_2, S'_3 définie par :

$$S'_r = S_r \oplus i_r \text{ pour } r = 1, 2, 3.$$

3. U envoie trois des chaînes engendrées à chacune des bases de données :

$$DB_{000} \text{ reçoit } (S_1, S_2, S_3) \text{ et } DB_{111} \text{ reçoit } (S'_1, S'_2, S'_3).$$

4. Chacune des bases de données simule trois bases appartenant à la même boule :
   - DB_{000} calcule $a_1 =< \overline{S}_1, X >$ et simule DB_{001}, DB_{010} et DB_{100}. Plus précisément, DB_{000} calcule $S_{r,j}$ pour $j \in [m]$ et $r = 1, 2, 3$ et simule les calculs pour chacune des chaînes possibles. Pour $a, b, c \in [m]$, DB_{000} calcule donc pour chacune des 3 bases de données :

$$\begin{array}{ll} a_{2,j} =< \overline{S}_2(j), X > & \text{mod 2 pour } DB_{001} \text{ et } j \in [m] \\ a_{3,j} =< \overline{S}_3(j), X > & \text{mod 2 pour } DB_{010} \text{ et } j \in [m]. \\ a_{4,j} =< \overline{S}_4(j), X > & \text{mod 2 pour } DB_{100} \text{ et } j \in [m]. \end{array}$$

   - De même DB_{111} calcule $a_8 =< \overline{S}_8, X >$ et simule DB_{011}, DB_{101} et DB_{110} en calculant :

$$\begin{array}{ll} a_{5,j} = & < \overline{S}_5(j), X > \quad \text{mod 2 pour } DB_{011}. \\ a_{6,j} = & < \overline{S}_6(j), X > \quad \text{mod 2 pour } DB_{101}. \\ a_{7,j} = & < \overline{S}_7(j), X > \quad \text{mod 2 pour } DB_{110}. \end{array}$$

   Pour chacune des bases de données simulées, il y a m possibilités. DB_{000} et DB_{111} envoient donc $1 + 3 \cdot m$ bits.

5. Enfin, U retrouve le bit X_{i_1,i_2,i_3} comme il aurait fait dans le protocole **P**, en faisant la somme dans $GF(2)$ des bits envoyés par les bases de données et en utilisant les $a_{i,j}$ convenables.

Complexité

On s'intéressera d'abord à la complexité dans le cas particulier où $d = 3$. L'utilisateur envoie 6 chaînes de bits, chacune de longueur m, soit au total $6 \cdot m$ bits. Chacune des deux bases de données répond en renvoyant $1 + 3 \cdot m$ bits, ce qui donne une complexité globale de $2 + 12 \cdot m$. Plus généralement, on note d la dimension de l'espace considéré et k le nombre de bases de données impliquées. Dans le protocole, celles-ci correspondent aux vecteurs engendrant le code considéré. Ici, il y en a k. L'utilisateur envoie $d \cdot n^{\frac{1}{d}}$ à chaque base de données. Chacune simule le calcul des autres vecteurs présents dans l'espace considéré : il y a 2^d éléments au total. Le nombre de bits renvoyé est donc $k + (2^d - k)n^{\frac{1}{d}}$.
En ce qui concerne la complexité calculatoire, chaque base de données renvoie sa réponse comme dans **P**, mais elle calcule aussi pour chacune des trois bases qu'elle simule le ou-exclusif de certains bits de x pour toutes les possibilités ,i.e., $\sqrt[d]{n}$ possibilités pour chacune des bases simulées. Donc chaque base fait au total au plus $n + d \cdot n\sqrt[d]{n}$ ou-exclusifs. L'utilisateur sait localiser le bit désiré. Il XORe les 2^d bits choisis, un pour chaque base de données comme dans **P**.

Théorème 2.1 *Soient d et k deux entiers tels qu'il existe un Covering Codes de distance* 1 *dans $\in \{0,1\}^d$. Alors il existe un schéma de PIR avec k bases de données, chacune possédant la chaîne x et telle que la complexité de la communication est $k + (2^d + (d-1)k)n^{\frac{1}{d}}$.*

remarque 2.3 *Les résultats obtenus avec cette méthode sont pertinents pour des petites valeurs de k : pour $k = 2$, la complexité de communication obtenue est $12 \cdot m + 2$ et pour $k = 4$, elle est de $28 \cdot m + 4$.*

2.2 Shamir's Secret Sharing, « SSS »

Dans la construction des protocoles de PIR, les schémas de partage de secret « à la Shamir », qu'on notera SSS *scheme* sont très importants. Leur rôle est exposé de manière détaillée dans [4]. On fera dans cette partie un bref rappel sur le protocole général de partage secret qui permettra aux bases de données de partager la requête et à l'utilisateur de retrouver l'élément possédé par les bases de données. De manière générale, un tel schéma permet à un utilisateur de partager un secret s entre l joueurs qui seront assimilés dans notre cas aux bases de données, de telle sorte qu'au moins t joueurs puissent reconstruire le secret s. En dessous de ce seuil, ce n'est pas possible.

2.2.1 Description générale de SSS scheme

Soit $\mathbb{F}$ un groupe fini à au moins l éléments, i.e., $|\mathbb{F}| = q > l$ et soient $\omega_1, \ldots, \omega_l$ l éléments de $\mathbb{F}$ non nuls. Un utilisateur U veut partager un secret s en utilisant un schéma avec l joueurs. À la fin, tout sous-ensemble de t joueurs suffit à reconstruire le secret. Selon les applications, on pourra préférer avoir un nombre supérieur de serveurs, à celui nécessaire pour reconstruire le secret. On parlera alors de t-out-of-l Sharing Shamir Scheme, SSS. Dans ce cas, on peut vérifier la réponse renvoyée par les serveurs et savoir lesquels d'entre eux sont malhonnêtes.

– U choisit t éléments aléatoires $a_0, \ldots, a_{t-1}$ et définit un polynôme uni-varié P en la variable Y défini par :
$$P(Y) = a_{t-1}Y^{t-1} + a_{t-2}Y^{t-2} + \ldots, a_1 Y + s.$$
On remarque que le secret est $P(0) = s$.
– Pour $j = 1, \ldots, l$, chaque DB_j possède une partie du secret s, $P(\omega_j)$.
– Par interpolation, tout sous-ensemble de t joueurs parmi les l joueurs peut retrouver le polynôme P de degré $t-1$.

Plus précisément, pour tout ensemble $\{j_1, \ldots, j_t\}$, il existe des constantes $\alpha_{j_1}, \ldots, \alpha_{j_t}$ appelées coefficients de Lagrange indépendantes du polynôme $P(Y)$ et du secret s, tel que pour tout $h = 1, \ldots, t$, on a :

$$\alpha_{j_h} = \prod_{d \neq h} \frac{\omega_{j_d}}{\omega_{j_d} - \omega_{j_h}} \text{ tel que } s = P(0) = \sum_{h=1}^{T} \alpha_{j_h} P(\omega_{j_h}).$$

De plus, tout sous-ensemble de $t-1$ joueurs n'apprend rien sur s.

2.3 Schémas d'Interpolation Polynomiale

Nous allons décrire dans cette partie un schéma proposé par Chor, Goldreich, Kushilevitz et Sudan en 1995, dans [9] qui permet de résoudre le problème des PIR. Le premier schéma est un cas particulier des schémas qui suivent avec $k = \log_2 n + 1$ bases de données. Ils obtiennent les résultats suivants :

– Un schéma pour $k = \frac{1}{3} * \log_2 n + 1$ bases de données et une complexité en $\frac{1}{3} * (1 + o(1)) \cdot (\log_2^2 n + 1)$ bits.
– Un schéma avec un nombre constant de bases de données k et une complexité en $O(n^{\frac{1}{k}})$ bits. Ce résultat est assez proche de la meilleure borne asymptotique obtenue à ce jour pour les *Private Information Theoretic Retrieval*, en $O(n^{\frac{\log_2 \log_2 k}{k \log_2 k}})$, [5].

2.3.1 Un premier schéma en $(1 + o(1)) \log_2^2 n \log_2 \log_2 (2n)$

Notations : on introduira toutes les notations qui nous seront utiles avant de décrire le protocole.

– On suppose que $n = 2^s$ et on identifiera l'ensemble $[n] = 1, \ldots, n$ à l'ensemble des fonctions de $[s]$ dans $\{0, 1\}$. Autrement dit, si $j \in [n]$, alors $j(l)$ sera la valeur du $l_{\text{ième}}$ bit de poids fort de l'expansion binaire de j. Cette notation permettra d'énumérer les n éléments de l'ensemble $[n]$.
– On note $\delta_{i,j}$ la fonction de Kronecker, i.e., $\delta_{i,j} = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{sinon} \end{cases}$.
– Ici, la chaîne de bits $x = x_1, \ldots, x_n$ sera codée par un polynôme uni-varié en z défini sur un groupe fini $GF(q)$ à au moins $s + 2$ éléments.
– Pour tout $p = 1, \ldots, s + 1$, DB_p calcule la valeur de $f_j^i(p)$, pour chaque $j \in [n]$. Elle calcule d'abord pour tout $l = 1, \ldots, s$:
$$f_{j,l}^i(p) = \begin{cases} g_l(p) & \text{si } j(l) = 1 \\ (1 - g_l(p)) & \text{sinon} \end{cases}$$

Les fonctions f_j^i sont définies pour tout $l \in [s]$ et pour tout $j \in [n]$ de la manière suivante :

$$f_j^i(z) = \prod_{l=1}^{s} f_{j,l}^i(z) = \prod_{j(l)=1} g_l(z) \prod_{j(l)=0} (1 - g_l(z)).$$

– Enfin on définit F une fonction en la variable z sur $GF(q)$ tel que :

$$F^{i,x}(z) = \sum_{j \in [n]} f_j^i(z) x_j \in GF(2).$$

Description du protocole

1. U veut retrouver le bit x_i.
2. Il choisit aléatoirement s éléments dans $GF(q)$ notés $r_1, \ldots, r_s$ et il définit s fonctions : $g_l(z) = r_l \cdot z + i(l)$, pour $l = 1, \ldots, s$. Il envoie $g_1(p), \ldots, g_s(p)$ à DB_p pour $p = 1 \ldots, s+1$. U envoie donc :

   $$g_1(1) = r_1 * 1 + i(1), \ldots, g_s(1) = r_s * 1 + i(s) \text{ à } DB_1.$$
   $$g_1(2) = r_2 * 1 + i(2), \ldots, g_s(2) = r_s * 2 + i(s) \text{ à } DB_2.$$
   $$\vdots$$
   $$g_1(s+1) = r_1 * (s+1) + i(1), \ldots, g_s(s+1) = r_s * (s+1) + i(s) \text{ à } DB_{s+1}.$$

   À cette étape, il envoie s éléments du groupe à chaque base de données, i.e., $s \cdot (s+1) \log_2 q$ éléments.
3. Pour tout $p = 1, \ldots, s+1$, DB_p calcule la valeur de $f_j^i(p)$, pour chaque $j \in [n]$. Et DB_p détermine $f_j^i(p)$ en vérifiant les conditions P_1 et P_2 :
   – P_1 : les f_j^i sont des polynômes en z de degré au plus s. Pour i fixé, chaque base de données aura à calculer n fonctions f_j^i. Ces fonctions seront définies plus loin par l'utilisateur afin de masquer i.
   – P_2 : $f_j^i(0) = \delta_{i,j}$, pour tout $j \in [n]$. C'est cette fonction qui permettra à l'utilisateur de retrouver la valeur de x_i.
4. Enfin, chaque base de données DB_p envoie sa valeur de $F^{i,x}(p)$ à l'utilisateur, avec :

   $$F^{i,x}(p) = f_1^i(p)x_1 + \cdots + f_n^i(p)x_n.$$

   Chaque base de données doit alors envoyer un élément du groupe soit au total $(s+1) \log_2 q$ bits.
5. À présent, U possède $F^{i,x}(1), \ldots, F^{i,x}(s+1)$. Par interpolation, il retrouve $x_i = F^{i,x}(0)$. En effet, il existe un unique polynôme de degré s, P tel que pour tout $p \in [s+1]$, $P(p) = F^{i,x}(p)$. On définit ce polynôme de la manière suivante : $P(z) \stackrel{\text{déf}}{=} a_s \cdot z^s + \cdots + a_1 z + x_i$. U possède $s+1$ couples $(1, P(1)), \ldots, (P(s+1))$ et veut retrouver ce polynôme ou plus précisément $P(0)$. Il écrit donc :

   $$a_0 + a_1 * 1 + \cdots + a_s * 1^s = P(1)$$
   $$a_0 + a_1 * 2 + \cdots + a_s * 2^s = P(2)$$
   $$\vdots$$
   $$a_0 + a_1 * (s+1) + \cdots + a_s * (s+1)^s = P(s+1)$$

On pose

$$V = \begin{pmatrix} 1 & 1^2 & \cdots & 1^s \\ 2 & 2^2 & \cdots & 2^s \\ \vdots & \vdots & \ddots & \vdots \\ s & s^2 & \cdots & s^s \\ s+1 & (s+1)^2 & \cdots & (s+1)^s \end{pmatrix}$$

On a alors :

$$V * \begin{pmatrix} a_0 \\ \vdots \\ a_s \end{pmatrix} = \begin{pmatrix} F^{i,x}(1) \\ \vdots \\ F^{i,x}(s+1) \end{pmatrix},$$

où $a_0 = x_i$. L'utilisateur retrouve la suite $(a_s, \ldots, a_1, x_i)$ puisque V est de déterminant non nul.

remarque 2.4 *Pour la preuve de sécurité, il suffit d'observer que les valeurs $g_1(p), \ldots, g_s(p)$ envoyées par l'utilisateur sont indépendantes et uniformément distribuées dans le corps par rapport à i.*

Correction

Pour la preuve de correction du schéma, il suffit de vérifier les propriétés P_1 et P_2 : en effet, si on suppose ces deux conditions vraies, alors d'après P_2, on a : $F^{i,x}(0) = \sum_{j\in[n]} f_j^i(0)x_j = \sum_{j\in[n]} \delta_{i,j}x_j = x_i$. D'autre part d'après P_1, $F^{i,x}$ est un polynôme en z de degré au plus s. Donc si l'utilisateur U possède la valeur de $F^{i,x}$ en $s+1$ points, il peut retrouver $F^{i,x}(0)$ par interpolation.

- Pour P_1, on peut noter que f_j^i est le produit de s polynômes linéaires en z.
- Pour P_2, on a :

$$\begin{aligned} f_j^i(0) &= \prod_{l=1}^{s} f_{j,l}^i(0) \\ &= \prod_{l=1}^{s} (j(l)\cdot i(l) + (1-j(l))\cdot(1-i(l))) \\ &= \delta_{i,j}. \end{aligned}$$

Complexité

On considère toujours la complexité en termes de nombre de bits transmis. L'utilisateur envoie s éléments du groupe à chaque base de donnée soit $s*(s+1)\cdot\log_2 q$ bits envoyés, avec q qui est le nombre d'éléments du groupe et chaque base de données répond en envoyant un élément de $GF(q)$, soit $(s+1)\cdot\log_2 q$ soit un total de $(s+1)^2\cdot\log_2 q$ bits communiqués. On doit avoir $q \geqslant s+2$, de manière à avoir au moins $s+1$ points non nuls ; ce qui permettra à l'utilisateur de retrouver x_i. En pratique, il se trouve qu'on peut toujours trouver un entier premier q tel que $q \in [s+2, 2s]$. Dans le protocole précédent, $s = \log_2 n$ donc avec $\log_2 n + 1$ bases de données, on obtient une complexité de $(1+o(1))\log_2^2 n \log_2\log_2(2n)$.

Pour la complexité en termes de nombre d'opérations dans le groupe, chaque base de données doit calculer $F^{i,x}(p)$; U doit pour cela calculer n produits, chacun étant des produits de s éléments de $GF(q)$, chacun égale à $g_l(p)$ ou $1-g_l(p)$, selon la valeur du bit $j(l)$ pour $l \in [s]$ et $j \in [n]$. Les bases de données doivent donc chacune faire $n\cdot(s+1)$ multiplications et n

additions. L'utilisateur retrouve le polynôme par interpolation. Ce calcul est cubique en la taille de la matrice V, i.e., s^3. Puis il retrouve le bit x_i en calculant $F^{i,x}(0)$, ce qui demande n multiplications et n additions.

remarque 2.5 *On peut remarquer qu'il y a un déséquilibre dans la répartition du nombre de bits envoyés : l'utilisateur envoie s fois plus d'éléments que les bases de données. Une technique très générale permet d'équilibrer ce nombre de bits ; elle sera présentée plus loin.*

2.3.2 Schéma général d'interpolation polynomiale

Dans cette partie, on considère le cas général où $k \leqslant \log_2 n$. L'idée reste la même mais on utilisera une représentation différente pour énumérer les entiers.

Notation : On précisera juste les notations qui changent. Chaque entier $j \in [n]$ sera représenté par la $j_{\text{ième}}$ séquence de s bits avec $k-1$ occurrences de 1. On ordonne les séquences par ordre lexicographique. On prend alors le plus petit s vérifiant $\binom{s}{k-1} \geqslant n$, de manière à pouvoir énumérer tous les éléments de l'ensemble $[n]$. On identifie alors le $j_{\text{ième}}$ éléments de $[n]$ à la fonction $j : [n] \mapsto \{0,1\}$ tel que pour tout $l = 1, \ldots, s$, $j(l)$ est le $l_{\text{ième}}$ bit de la $j_{\text{ième}}$ séquence de s bits.

Description du protocole

1. L'utilisateur veut retrouver le bit x_i.
2. Il choisit aléatoirement s éléments dans $GF(q)$, notés $r_1, \ldots, r_s$, et définit s fonctions : $g_l(z) = r_l \cdot z + i(l)$, pour $l = 1, \ldots, s$. Il envoie $g_1(p), \ldots, g_s(p)$ à DB_p pour $p = 1 \ldots, k$. Il envoie donc :

$$\begin{aligned} &g_1(1) = r_1 \cdot 1 + i(1), \ldots, g_s(1) = r_s \cdot 1 + i(s) \text{ à } DB_1. \\ &g_1(2) = r_2 \cdot 1 + i(2), \ldots, g_s(2) = r_s \cdot 2 + i(s) \text{ à } DB_2. \\ &\vdots \\ &g_1(k) = r_1 \cdot k + i(1), \ldots, g_s(k) = r_s \cdot k + i(s) \text{ à } DB_k. \end{aligned}$$

   À cette étape, U envoie s éléments du groupe à chaque base de données, i.e., $k \cdot s \log_2 q$ éléments.
3. Pour tout $p = 1, \ldots, k$, DB_p construit à partir des valeurs reçues $f_j^i(p)$ vérifiant les conditions P_1 et P_2 :
   - P_1 : les fonctions f_j^i sont des polynômes en z de degré au plus s. Pour i fixé, chaque base de données aura à calculer n fonctions f_j^i. Ces fonctions seront définies plus loin par l'utilisateur afin de masquer i.
   - P_2 : $f_j^i(0) = \delta_{i,j}$, pour tout $j \in [n]$. Cette fonction permettra à l'utilisateur de retrouver la valeur de x_i.

   Chaque DB_p calcule pour tout $j \in [n]$, $f_j^i(p)$ qui doit être ici de degré au plus $k-1$: pour tout $l = 1, \ldots, s$, $f_{j,l}^i(z)$ est définie par :

$$f_{j,l}^i(z) = j(l) \cdot g_l(z) + (1 - j(l))(1 - g_l(z)),$$

   où $j(l)$ est le $l_{\text{ième}}$ bit de j. Et le polynôme f_j^i est alors défini par :

$$f_j^i(z) = \prod_{l : j(l)=1} f_{j,l}^i(z).$$

Il se trouve qu'il y a exactement $k-1$ occurrences de 1 dans la $j_{\text{ième}}$ séquence de s bits. f_j^i est donc de degré exactement $k-1$.

4. Enfin, chaque base de données DB_p envoie sa réponse à l'utilisateur : $F^{i,x}(p) = f_0^i(p)x_0 + f_1^i(p)x_1 + \cdots + f_n^i(p)x_n$. Chaque base de données envoie un élément du groupe, soit au total $(s+1)\cdot\log_2 q$ bits.
5. À présent, U possède $F^{i,x}(1), \ldots, F^{i,x}(k)$. Il peut donc déterminer le polynôme $F^{i,x}$, qui est de degré au plus $k-1$. Par interpolation, il retrouve comme précédemment l'élément $x_i = F^{i,x}(0)$.

Correction

Il s'agit de vérifier les conditions P_1 et P_2. La propriété P_1 assure que l'utilisateur peut reconstruire le polynôme souhaité à partir des valeurs reçues ; ce qui a déjà été vérifié dans le protocole. Si la propriété P_2 est vérifié alors l'utilisateur retrouve bien le bit x_i, et pas une autre valeur, puisque : $F^{i,x}(0) = \sum_{j\in[n]} f_j^i(0)x_j = \sum_{j\in[n]} \delta_{i,j}x_j = x_i$. Vérifions P_2 :

$$\begin{aligned} f_{j,l}^i(0) &= \prod_{l:j(l)=1} f_{j,l}^i(0) \\ &. = \prod_{l:j(l)=1} (j(l) * i(l) + (1-j(l)) * (1-i(l)) \\ &= \begin{cases} \prod_{l=1}^{s} (i^2(l) + (1-i(l))^2) & \text{si } i = j \\ 0 & \text{sinon} \end{cases} \end{aligned}$$

Si $i = j$, on remarque que l'un des éléments de la somme est toujours égal à 1. Si $i \neq j$, il existe un indice l pour lequel on a $i(l) = 1$ et $j(l) = 0$, alors un élément du produit sera égal à 0, d'où le résultat. Et on a donc : $f_{j,l}^i(0) = \delta_{i,j}$.

Complexité

La répartition du nombre de bits communiqués dans ce protocole est la même que la précédente. Cela dit, la complexité globale dépend de k, le nombre de bases de données, de s, le nombre d'éléments du groupe envoyés par l'utilisateur et de q la taille du groupe. La complexité globale est ici $k\cdot(s+1)\cdot\log_2 q$. On doit toujours avoir $\binom{s}{k-1} \geqslant n$ et $q \geqslant k+1$, de manière à pouvoir énumérer tous les n éléments et retrouver x_i par interpolation polynomiale. Si on prend la plus petite valeur de q satisfaisant ces conditions, on obtient : $k*(s+1)\log_2 q \leqslant k*(s+1)\log_2(k+1)$. Dans [13], une analyse de complexité montre qu'il existe un PIR pour $s = \log_2 n + \log_2\log_2 n$ et $k = \frac{s}{2} + 1$:

Théorème 2.2 *Il existe un schéma de « Private Information Retrieval » pour* $\log_2 n + \log_2(\log_2 n + 1)$ *bases de données, chacune possédant une chaîne caractère de longueur n avec une complexité en* $\frac{1}{2}(1+o(1))\log_2^2 n \cdot \log_2(\log_2 2n)$ *bits.*

remarque 2.6 *Ce résultat est une amélioration du résultat précédent. Nous l'obtenons en diminuant de moitié le nombre de bases de données impliquées. En prenant k constant, s est de l'ordre de* $O(n^{\frac{1}{k-1}})$ *et on obtient alors une complexité en* $O(n^{\frac{1}{k-1}})$*. Le résultat qui suit améliore cette borne à* $O(n^{\frac{1}{k}})$*, en équilibrant la répartition des bits envoyés dans le protocole.*

2.3.3 Amélioration du schéma général d'Interpolation polynomiale

Cette méthode n'améliore pas la complexité globale du schéma général. Cela dit, en modifiant la représentation des entiers dans l'ensemble $[n]$, la dépendance entre s, k et n permet de prendre un nombre plus petit de bases de données. On rappelle que $n = 2^s$ et que k est le nombre de bases de données. Dans ce cas, le nombre de bits envoyés par les bases de données diminuent. Cette amélioration est pertinente pour $k = \Theta(\log_2 n)$.

Notations :

- On considère désormais les séquences de longueur s à valeur dans $\{0, \ldots, k-1\}$ tel que $\sum_{l=1}^{s} j(l) = k-1$, où $j(l)$ est la $j_{\text{ième}}$ valeur de la séquence. Le nombre de séquences vérifiant cette condition est $\binom{s+k-2}{k-1}$. On doit donc avoir $\binom{s+k-2}{k-1} \geqslant n$.
- $GF(q)$ est un corps fini à au moins $k+1$ éléments.
- L'utilisateur veut retrouver le $i_{\text{ième}}$ élément de la séquence. i sera représenté comme un vecteur sur $GF(q)$ de dimension s, noté $i = (i(1), \ldots, i(s))$.
- $\vec{w}$ est un vecteur choisi aléatoirement et uniformément dans $GF(q)^s$.
- On considère deux polynômes G et F. G est un polynôme multi-varié en les variables $y_1, cdots, y_s$ et F est un polynôme uni-varié définit sur $GF(q)$ tel que : $F(\vec{z}) = G(\vec{\imath} + z * \vec{w})$.

Description du protocole

1. U choisit aléatoirement s éléments dans le groupe $GF(q)$, notés $w_1, \ldots, w_s$, tels que $\vec{w} = (w(1), \ldots, w(s)) \in GF(q)^s$. Il envoie $i_1 + pw(1), \ldots, i_2 + pw(2)$ à DB_p pour $p = 1 \ldots, k$. À cette étape, il envoie toujours $k * s * \log_2 q$ bits au total.
2. Chaque base de données DB_p pour $p = 1 \ldots, k$ calcule pour $j = 1, \ldots, n$ un polynôme f_k^j défini de $GF(q)^s$ dans $GF(q)$par :

$$f_k^j(\vec{y}) = \prod_{l=1}^{s} \prod_{p=0}^{j(l)-1} \frac{y(l) - p}{j(l) - p}.$$

   On remarque alors que f_k^j est de degré $\sum_{l=1}^{s} j(l) = k-1$. DB_p construit alors un polynôme G multi-varié en $(y_1, \ldots, y_s)$ défini par

$$G : GF(q)^s \mapsto GF(q), \text{tel que } G(\vec{y}) = \sum_{j \in [n]} f_k^j(\vec{y}) x_j,$$

   où G est de degré au plus $k-1$, puisque par définition f_k^j est de degré au plus $k-1$.
   Puis chaque DB_p construit un polynôme G vérifiant les conditions P_1 et P_2.
   - P_1 : G est un polynôme en s variables de degré total au plus $k-1$ et F est un polynôme uni-varié en z de degré $k-1$
   - P_2 : Pour tout $j \in [n]$, $G(\vec{j}) = x_j$. En particulier, on a $F(0) = G(\vec{\imath}) = x_i$.

   DB_p peut calculer $F(p) = G(\vec{\imath} + p\vec{w})$, elle envoie cette valeur à l'utilisateur i.e., $\log_2 q$ bits.
3. A partir des k évaluations du polynôme F que l'utilisateur reçoit, il retrouve la valeur de $F(0)$ par interpolation, puisque F est la somme de n polynômes de degré $k-1$.

Correction

Vérifions d'abord que pour tout j', on a : $f_k^j(\vec{j'}) = \delta_{j,j'}$.
Si $j = j'$, tous les termes du produit sont égaux à 1. Si $j \neq j'$, il existe un indice l pour lequel

$j'(l) \leqslant j(l)$. Pour $p = j'(l)$, le numérateur du quotient sera nul, d'où le résultat. Si la propriété P_2 est vérifiée, l'utilisateur retrouve bien la valeur de x_i. Il s'agit donc de vérifier P_2 :

$$\begin{aligned} f^i_{j,l}(0) &= \prod_{l:j(l)=1} f^i_{j,l}(0) \\ &. = \prod_{l:j(l)=1} (j(l)\cdot i(l) + (1-j(l))\cdot(1-i(l)) \\ &= \begin{cases} \prod_{l=1}^{s}(i^2(l) + (1-i(l))^2) & \text{si } i = j \\ 0 & \text{sinon} \end{cases} \end{aligned}$$

Et alors

$$\begin{aligned} F(0) &= G(\vec{i}) \\ &= \sum_{j\in[n]} f^j_k(\vec{i})x_j \\ &= \sum_{j\in[n]} \delta_{j,i}x_j = x_i \end{aligned}$$

Complexité

La répartition du nombre de bits communiqués est encore la même que dans le protocole précédent. Cela dit, la complexité globale dépend de k, le nombre de bases de données, de s, le nombre d'éléments du groupe envoyés par l'utilisateur et de q. L'utilisateur envoie $s\log_2 q$ bits à chaque base de données et reçoit $\log_2 q$ bits de chacune d'elles. La complexité globale est ici $k * (s+1)\log_2 q$. On doit toujours avoir $\binom{s+k-2}{k-1} \geqslant n$ et $q \geqslant k+1$, de manière à pouvoir énumérer tous les n éléments et retrouver x_i par interpolation polynomiale.

remarque 2.7 *En réalité, cette amélioration est pertinente pour des valeurs de k de l'ordre de $\Theta(\log_2 n)$. Si on prend $k = 1/3 * \log_2 n$ et $s = 2+\log_2 n$, on a $\binom{s+k-2}{k-1} \approx 2^{H_2(1/4)4/3\log_2 n} > n^{1.081}$, où $H_2()$ est la fonction binaire d'entropie.*

Théorème 2.3 *Il existe un schéma de Private Information Retrieval pour $1 + \frac{1}{3}\log_2 n$ bases de données, chacune possédant une chaîne de bits de longueur n avec une complexité en $\frac{1}{3}(1 + o(1))\log_2^2 n \log_2\log_2 2n$.*

2.4 Private Information Retrieval of Blocks

On peut généraliser le problème des PIR à celui des *PIR par bloc*. On note alors $PIR_k(n,l)$ un schéma de PIR, où k est le nombre de bases de données, n la taille de x et l le nombre de bits qu'on veut récupérer. On suppose que la base de données est une partition de blocs. Pour simplifier, on supposera que chaque bloc contient le même nombre de bits l. Une manière triviale de résoudre ce problème est d'invoquer l fois un schéma de PIR pour un seul bit, $\mathrm{PIR}_k(n,1)$. Tout d'abord, nous présenterons une méthode générique plus efficace qui aura pour effet d'équilibrer le nombre de bits transmis. Puis nous appliquerons le schéma au protocole précédent. Cette procédure augmentera le nombre de bits envoyés par les bases de données ; ce qui conduit à une meilleure complexité asymptotique. Et enfin on donnera des résultats généraux.

2.4.1 Construction de P'

À partir d'un schéma de $PIR_k(n,1)$, **P**, pour lequel les bases de données contiennent $n = m \cdot l$ bits, nous allons construire un schéma de $PIR_k(n,l)$, **P'**, pour lequel l'utilisateur récupère l bits. Ces $m \cdot l$ bits seront vus comme une matrice de taille $l \times m$. L'utilisateur veut retrouver la $i_{\text{ème}}$ colonne. Il agit comme dans le protocole **P** avec pour indice secret i. Chaque base de données exécute le protocole **P** l fois en parallèle, en considérant une ligne différente à chaque exécution du protocole. L'utilisateur retrouve ainsi toute la $i_{\text{ième}}$ colonne.

Complexité du protocole P'

Le nombre de bit envoyés par l'utilisateur est identique au protocole **P**. Mais chaque base de données envoie l fois plus d'éléments dans **P'**que dans **P**.

2.4.2 Application au schéma 2.3.2 et résultat

En appliquant la technique présentée pour équilibrer le nombre de bits envoyés au schéma général d'interpolation polynomiale **P**, on se ramène au résultat suivant :

Théorème 2.4 *Soit k, m, l et s tel que $\binom{s}{k-1} l \geqslant m$, et $q \geqslant k+1$ une puissance d'un nombre premier, alors il existe un schéma de PIR avec une complexité en $k * (s+l) \log_2 q$.*

preuve 2.1 *Le contenu de la base de données est vu comme une matrice de taille $l \times m$ et on considère le protocole par interpolation polynomiale dans le cas général. On suppose que les entiers k, n, l et s sont tels que $\binom{s}{k-1} \geqslant l$ et $q \geqslant k+1$ afin de pouvoir appliquer notre schéma d'interpolation polynomiale. On représente la base de données comme une matrice de taille $l \times m$. L'utilisateur veut retrouver le bit à la position $i_{ième}$ colonne. Il agit comme de la partie **1.3.3** avec pour indice secret i. Chaque base de données exécute le protocole **P** l fois en parallèle, comme décrit précédemment. Seulement, plutôt que d'envoyer un bit comme dans **P**, les bases de données envoient l bits à chaque fois, comme si l'utilisateur demandait un bloc entier de taille l. L'utilisateur envoie donc le même nombre de bit que dans **P**, mais les bases de données envoient au total $k \cdot l \cdot \log_2 q$ bits.*

Résultat

Dans [9], en posant $s = \sqrt[k-1]{(k-1)!(n/m+k)}$, on a bien $\binom{s}{k-1} \geqslant n/m$. En prenant $m = \sqrt[k]{n}$ et $s = \sqrt[k-1]{(k-1)!(n^{\frac{k-1}{k}}+k)}$, ils obtiennent une complexité en $k * (s + \sqrt[k]{n+k}) \log_2 q$. Ce résultat est asymptotiquement meilleur que la méthode des codes couvrants sauf pour les cas où $k = 2$ et $k = 4$, avec lesquels on obtient ici respectivement une complexité en $O(n^{\frac{1}{2}})$ et $O(n^{\frac{1}{3}})$ (avec une constante d'ordre comparable dans les deux méthodes pour le dernier résultat).

2.4.3 Application au schéma 2.3.3 et résultat

À présent, si on applique cette transformation au dernier schéma d'interpolation polynomiale présenté, nous obtenons le résultat suivant :

Théorème 2.5 *Soit k, le nombre de bases de données et n, l et s tel que $\binom{s+k-2}{k-1} l \geqslant n$ et $q \geqslant k+1$ une puissance d'un nombre premier, alors il existe un schéma de PIR, où les bases de données possèdent une chaîne de caractères de longueur n et pour lequel chacune reçoit $s \log_2 q$ bits et renvoie $l \cdot \log_2 q$ bits.*

remarque 2.8 *Pour k fixé, la complexité asymptotique reste la même avec ou sans l'amélioration ; cela dit, pour des grandes valeurs de k (par exemple* $k = 16$ *et* $n = 2^{40}$*), l'amélioration est pertinente :* 3205 *bits sont transmis dans le premier cas et* 2289 *dans le second avec ces valeurs pour k et n.*

2.5 Application

2.5.1 Retrouver des éléments dans $GF(q)$

Nous voulons à présent retrouver des éléments appartenant à un groupe $GF(q)$, avec q premier ou puissance d'un entier premier p. Le protocole d'interpolation polynomiale peut être une solution pour des valeurs de q pas trop grandes. Il s'avère cependant que si q est trop grand, cette solution devient difficile en pratique. En effet, la complexité augmente des deux côtés, i.e., pour l' utilisateur et pour les bases de données puisqu'un facteur $\log_2 q$ qui apparaît dans la communication des deux participants.

Dans cette partie, on cherche à manipuler des éléments dans un groupe d'ordre q avec q assez grand. Une première solution qui permettrait de retrouver $\log_2 q$ bits est d'appliquer la méthode de parallélisation des bases de données. Dans ce cas, un facteur $\log_2 q$ apparaît seulement du côté des bases de données. Ici, nous allons construire un protocole permettant à l'utilisateur de retrouver directement $\log_2 q$ bits tout en préservant sa confidentialité. En réalité, il s'agit d'une extension du protocole de sommation linéaire. Le but de cette construction est de permettre l'intégration d'un tel PIR dans un protocole cryptographique qui utilise des objets de $GF(q)$.

Nous allons reprendre le protocole de sommation linéaire afin de voir comment il serait possible de manipuler des éléments dans $GF(q)$, le but étant d'intégrer ce schéma de PIR à un protocole d'établissement de clé de session par mot de passe $\in GF(q)$ stocké dans un serveur.

2.5.2 Schéma de sommation linéaire dans $GF(q)$

Dans la construction de PIR présentée à la section 2.1.1, il y a 4 bases de données, donc 4 réponses à combiner pour retrouver l'élément désiré. L'utilisateur fait une requête à chaque base de données avec pour indice secret (i, j) qui correspond au mot de passe $X_{i,j}$ d'un client C. La base de données $X_{k,l}$ avec $k \in [m]$ et $l \in [p]$ possédées par chacune des bases contient donc n éléments de $\mathbb{Z}_q$. Si les éléments de la base de données sont dans $\mathbb{Z}_q$, alors la somme n'est plus dans $GF(2)$ mais les calculs se font modulo q. On suppose que G connaît la taille de la base de données n. La base de données est vue comme une matrice de taille $m \times [p]$. La première base de données envoie B_{00}, la seconde B_{01}, la troisième B_{10} et la quatrième B_{11}. Les chaînes aléatoires S_0 et S_1 sont respectivement dans $\{0,1\}^m$ et $\{0,1\}^p$. Auparavant nous avions :

$$\begin{aligned} B_{00} &= \oplus_{S_0(j_1)=1, S_1(j_2)=1} X_{j_1,j_2}, \\ B_{01} &= B_{00} \oplus (\oplus_{S_0(j_1)=1} X_{j_1}, j), \\ B_{10} &= B_{00} \oplus (\oplus_{S_1(j_2)=1} X_{i,j_2}), \\ B_{11} &= B_{00} \oplus B_{01} \oplus B_{10} \oplus X_{i,j}. \end{aligned}$$

Voici le nouveau protocole obtenu en adaptant le protocole de 2.1.1 à un nouveau protocole où les éléments de la base de données sont dans $\mathbb{Z}_q$.

Protocole

1. G possède l'indice secret (i, j). G choisit deux chaînes aléatoires S_0 et S_1 respectivement dans $\{0,1\}^m$ et dans $\{0,1\}^p$. Elle calcule $S'_0 = S_0 \oplus \mathbb{1}_i$ et $S'_1 = S_1 \oplus \mathbb{1}_j$. G envoie S_0 et S_1 à DB_{00}, S'_0 et S_1 à DB_{01}, S_0 et S'_1 à DB_{10} et enfin S'_0 et S'_1 à DB_{11}.

2. DB_{00} calcule et envoie $B_{00} = S_0^T.X.S_1 \mod q$,
   DB_{01} calcule et envoie $B_{01} = S_0'^T.X.S_1 \mod q$,
   DB_{10} calcule et envoie $B_{10} = S_0^T.X.S_1' \mod q$,
   DB_{11} calcule et envoie $B_{11} = S_0'^T.X.S_1' \mod q$.
   Voyons comment les calculs se transposent dans $\mathbb{Z}_q$. On définit $S_{0,i} \overset{\text{déf}}{=} b_1$ et $S_{1,j} \overset{\text{déf}}{=} b_2$. On ne donnera pas le détail de tous les cas mais seulement le résultat final de tous les cas. On notera $X_{i,*}$ le vecteur de la $i_{\text{ième}}$ ligne de la matrice X et $X_{*,j}$ représentant le vecteur de la $j_{\text{ième}}$ colonne de la matrice X.
   – Si $b_1 = 1$,

$$\begin{aligned} B_{01} &= S_0'^T \cdot X \cdot S_1 \mod q, \\ &= (S_0 \oplus \mathbb{1}_i)^T \cdot X \cdot S_1 \mod q, \\ &= B_{00} - \mathbb{1}_i^T \cdot X \cdot S_1 \mod q \\ &= B_{00} - X_{i,*} \cdot S_1 \mod q. \end{aligned}$$

   – Si $b_1 = 0$,

$$\begin{aligned} B_{01} &= S_0'^T \cdot X \cdot S_1 \mod q, \\ &= (S_0 \oplus \mathbb{1}_i)^T \cdot X \cdot S_1 \mod q, \\ &= B_{00} + {\mathbb{1}_i}^T \cdot X \cdot S_1 \mod q \\ &= B_{00} + X_{i,*} \cdot S_1 \mod q. \end{aligned}$$

   – Si $b_2 = 1$,

$$\begin{aligned} B_{10} &= S_0^T \cdot X \cdot S_1' \mod q, \\ &= S_0^T \cdot X \cdot (S_1 + \mathbb{1}_j) \mod q, \\ &= B_{00} - S_0^T \cdot X \cdot \mathbb{1}_j \mod q \\ &= B_{00} - S_0^T \cdot X_{*,j} \mod q. \end{aligned}$$

   – Si $b_2 = 0$,

$$\begin{aligned} B_{10} &= S_0^T \cdot X \cdot S_1' \mod q, \\ &= S_0^T \cdot X \cdot (S_1 \oplus \mathbb{1}_j) \mod q, \\ &= B_{00} + S_0^T.X \cdot \mathbb{1}_j \mod q \\ &= B_{00} + S_0^T \cdot X_{*,j} \mod q. \end{aligned}$$

   – Si $b_1 = b_2$, par exemple si $b_1 = b_2 = 1$,

$$\begin{aligned} B_{11} &= S_0'^T \cdot X \cdot S_1' \mod q, \\ &= (S_0 \oplus \mathbb{1}_i)^T \cdot X \cdot (S_1 \oplus \mathbb{1}_j) \mod q \\ &= B_{00} - ({\mathbb{1}_i}^T \cdot X \cdot S_1) - (S_0^T \cdot X \cdot \mathbb{1}_j) + X_{i,j} \mod q \\ &= B_{00} - X_{i,*} \cdot S_1 - S_0^T \cdot X_{*,j} + X_{i,j} \mod q. \end{aligned}$$

   – Le cas $(b_1, b_2) = 0$ est similaire.

– Si $b_1 \neq b_2$, par exemple si $b_1 = 1$ et $b_2 = 0$,

$$\begin{aligned} B_{11} &= S_0'^T \cdot X \cdot S_1' \mod q \\ &= (S_0 - \mathbb{1}_i)^T \cdot X \cdot (S_1 + \mathbb{1}_j) \mod q \\ &= B_{00} - ({\mathbb{1}_i}^T \cdot X \cdot S_1) + (S_0^T \cdot X \cdot \mathbb{1}_j) - (\mathbb{1}_i \cdot X \cdot \mathbb{1}_j) \mod q \\ &= B_{00} - X_{i,*} \cdot S_1 + S_0^T \cdot X_{*,j} - X_{i,j} \mod q. \end{aligned}$$

Voici le résultat final pour les 4 cas qu'on pourra vérifier en suivant la démarche ci-dessus :
– Si $(b_1, b_2) = (0, 0)$, on a :

$$\begin{aligned} B_{01} &= B_{00} + {\mathbb{1}_i}^T \cdot X \cdot S_1 \mod q \\ B_{10} &= B_{00} + (S_0^T \cdot X \cdot \mathbb{1}_j) \mod q \\ B_{11} &= B_{00} + ({\mathbb{1}_i}^T \cdot X \cdot S_1 + S_0^T \cdot X \cdot \mathbb{1}_j + X_{i,j}) \mod q \end{aligned}$$

Dans ce cas l'utilisateur calcule :

$$\begin{aligned} +B_{00} - B_{01} - B_{10} + B_{11} \mod q &= +B_{00} - (B_{00} + X_{i,*} \cdot S_1) - (B_{00} + S_0^T \cdot X_{*,j}) \\ &\quad + (B_{00} + X_{i,*} \cdot S_1 + S_0^T \cdot X_{*,j} + X_{i,j}) \mod q \\ &= X_{i,j} \mod q. \end{aligned}$$

– Si $(b_1, b_2) = (1, 0)$, on a :

$$\begin{aligned} B_{01} &= B_{00} - {\mathbb{1}_i}^T \cdot X \cdot S_1 \mod q \\ B_{10} &= B_{00} + S_0^T.X.\mathbb{1}_j \mod q \\ B_{11} &= B_{00} - ({\mathbb{1}_i}^T \cdot X \cdot S_1) + (S_0^T \cdot X \cdot \mathbb{1}_j) - X_{i,j} \mod q \end{aligned}$$

Dans ce cas l'utilisateur calcule :

$$\begin{aligned} +B_{00} - B_{01} - B_{10} + B_{11} \mod q &= +B_{00} - (B_{00} - X_{i,*} \cdot S_1) - (B_{00} + S_0^T \cdot X_{*,j}) \\ &\quad + (B_{00} - X_{i,*}.S_1 + S_0^T.X_{*,j} - X_{i,j}) \mod q \\ &= -X_{i,j} \mod q. \end{aligned}$$

– Si $(b_1, b_2) = (0, 1)$, on a :

$$\begin{aligned} B_{01} &= B_{00} + {\mathbb{1}_i}^T \cdot X \cdot S_1 \mod q \\ B_{10} &= B_{00} - S_0^T \cdot X \cdot \mathbb{1}_j \mod q \\ B_{11} &= B_{00} + ({\mathbb{1}_i}^T \cdot X \cdot S_1) - (S_0^T \cdot X \cdot \mathbb{1}_j) - X_{i,j} \mod q. \end{aligned}$$

Dans ce cas l'utilisateur calcule :

$$\begin{aligned} +B_{00} - B_{01} - B_{10} + B_{11} \mod q &= +B_{00} - (B_{00} + X_{i,*} \cdot S_1) - (B_{00} - S_0^T \cdot X_{*,j}) \\ &\quad + (B_{00} + X_{i,*} \cdot S_1 - S_0^T \cdot X_{*,j} - X_{i,j}) \mod q \\ &= -X_{i,j} \mod q. \end{aligned}$$

– Si $(b_1, b_2) = (1, 1)$, on a :

$$\begin{aligned} B_{01} &= B_{00} - ({\mathbb{1}_i}^T \cdot X \cdot S_1) \mod q \\ B_{10} &= B_{00} - (S_0^T \cdot X \cdot \mathbb{1}_j) \mod q \\ B_{11} &= B_{00} - ({\mathbb{1}_i}^T \cdot X \cdot S_1) - (S_0^T \cdot X \cdot \mathbb{1}_j) + X_{i,j} \mod q. \end{aligned}$$

Dans ce cas l'utilisateur calcule :

$$\begin{aligned} +B_{00} - B_{10} - B_{01} + B_{11} \mod q &= +B_{00} - (B_{00} - X_{i,*} \cdot S_1) - (B_{00} - S_0^T \cdot X_{*,j}) \\ &\quad + (B_{00} - X_{i,*} \cdot S_1 - S_0^T \cdot X_{*,j} + X_{i,j}) \mod q \\ &= X_{i,j} \mod q. \end{aligned}$$

Conclusion

En considérant toutes les possibilités, on a montré que la seule combinaison de signes devant les 4 sommes B_{00}, B_{10}, B_{01} et B_{11} est « +, -, -, + ». La dernière condition annule une des deux sommes et on obtient ainsi la solution qui convient.

Correction

D'après l'analyse de cas qui précède, G doit être capable de savoir s'il reçoit $X_{i,j} \mod q$ ou bien $-X_{i,j} \mod q$. On voit que si b_1 et b_2 sont de même signe, il reçoit directement $X_{i,j}$ et si b_1 et b_2 sont de signes opposés, G reçoit $-X_{i,j} \mod q$.

Complexité

Du côté utilisateur, la complexité de communication est la même. Mais les 4 bases de données envoient désormais chacune 4 éléments de $GF(q)$. On a donc un facteur $\log_2 q$ qui apparaît devant chaque élément envoyé.

2.5.3 Intégration au protocole de « Key Exchange » authentifié par mot de passe

Reprenons le protocole $GPAKE$, de l'article [1], où $\mathbb{G} = (G, g, q)$ est un groupe d'ordre q, engendré par l'élément g ; l est un paramètre de sécurité. On définit les fonctions $\mathcal{G}$, Hash_1 et Hash_2 par :

$$\mathcal{G} : \mathcal{U}^2 \times Dic \mapsto \mathbb{G},\ \mathrm{Hash}_1 : U^2 \times \mathbb{G} \times \mathbb{G} \mapsto \{0,1\}^l \text{ et } \mathrm{Hash}_2 : U^2 \times \mathbb{G} \times \mathbb{G} \mapsto \{0,1\}^l,$$

où $Hash_1$ et $Hash_2$ sont des oracles aléatoires. Le protocole initial de [1] permet d'établir une clé de session entre C, le client et G, la passerelle. On cherche à modifier ce schéma afin de permettre à C de préserver son anonymat auprès du serveur S. Dans le schéma d'origine, l'utilisateur choisit x aléatoire dans $\mathbb{Z}_q$ et envoie son identité C, assimilable à l'indice désiré et le chiffré de g^x, X^*. G choisit y aléatoire dans $\mathbb{Z}_q$ et calcule g^y. G envoie le couple (C, X^*) au serveur qui choisit alors $s \in \mathbb{Z}_q$ aléatoire dans $\mathbb{Z}_q$; il retrouve PW à partir de l'identité du client, C. Et enfin, il peut retrouver X. Il envoie $\overline{X} \stackrel{\text{déf}}{=} X^s$ et $\overline{g} \stackrel{\text{déf}}{=} g^s$ à la passerelle, qui calcule sa clé de session provisoire K avec son y et $\overline{Y} \stackrel{\text{déf}}{=} \overline{g}^y$.
G envoie $\overline{Y}$ et son authentificateur $\mathrm{AuthG} \leftarrow Hash_2(C, G, X^*, \overline{Y}, K)$ au client. Ce dernier calcule sa clé de session provisoire résultant du message reçu et vérifie sa validité par comparaison avec son authentificateur qu'il calcule au préalable. On peut se reporter au schéma 2.1, $GPAKE$ pour plus de détails.

Désormais le client veut préserver son anonymat auprès du serveur S. Mais le serveur doit pouvoir authentifier le client, i.e., vérifier que C connaît le bon mot de passe, PW. Et G doit pouvoir retrouver $\overline{X}$ sans transmettre C à S. On peut alors utiliser une construction de PIR où la passerelle fait une requête et le serveur lui renvoie une réponse qui lui permet de calculer $\overline{X}$ et de poursuivre le protocole $GPAKE$. Un tel schéma posséderait les mêmes propriétés que le protocole $GPAKE$, mais il aurait l'avantage de ne pas transmettre l'identité, C du client à S.

Spécifions les paramètres du nouveau protocole **PP** que l'on veut construire. Il s'agit d'intégrer dans $GPAKE$ un protocole interactif de PIR entre G, l'utilisateur et S, le serveur. On s'intéressera pour cela au protocole présenté à la section2.5.2. On considère que le mot de passe PW n'est plus égal à $\mathcal{G}(C, G, pw)$, avec $pw \in \mathbb{Z}_q$. La fonction $\mathcal{G}$ est définie différemment. À présent, on a : $PW \stackrel{\text{déf}}{=} h^{pw}$, où $h \in \mathbb{G}$ et $pw = \mathcal{G}(C, G, ppw) \in \mathbb{Z}_q$, avec $ppw \in Dic$. La base de données pw est toujours vue comme une matrice de taille $n = m \times p$. Elle contient n mots de passe $pw_{k,l}$ pour tout $k \in [m]$ et $l \in [p]$. G possède l'indice secret (i, j) correspondant à l'identité C du client. Le mot de passe de C est donc $PW \stackrel{\text{déf}}{=} h^{pw_{i,j}} \in \mathbb{G}$. Le mot de passe PW est partagé en quatre éléments de $\mathbb{G}, PW_{00}, PW_{01}, PW_{10}$ et PW_{11}. La connaissance de ces 4 éléments permet de retrouver PW. De manière similaire, la connaissance de $pw_{i,j}$ est partagé en la connaissance de 4 éléments $pw_{00}, pw_{01}, pw_{10}$ et $pw_{11} \in \mathbb{Z}_q$. Les 4 bases de données sont $DB_{00}, DB_{01}, DB_{10}, DB_{11}$. On suppose qu'elles ont toutes les 4 accès à un aléa commun s.

client C		passerelle G		serveur S
$\mathcal{G}, Hash1, Hash2$		$\mathcal{G}, Hash1, Hash2$		$\mathcal{G}, Hash1, Hash2$
$pw \in Dic$				
$PW = \mathcal{G}(C, G, pw) \in \mathbb{G}$				$PW = \mathcal{G}(C, G, pw) \in \mathbb{G}$
	canal non authentifié		canal privé authentifié	
$accept \leftarrow false$		$accept \leftarrow false$		
$x \xleftarrow{R} \mathbb{Z}_q, X \leftarrow g^x$				
$X^* \leftarrow X \times PW$	$\xrightarrow{C, X^*}$			
		$y \xleftarrow{R} \mathbb{Z}_q, Y \leftarrow g^y$	$\xrightarrow{C, X^*}$	
				$s \xleftarrow{R} \mathbb{Z}_q$
				$X \leftarrow X^*/PW$
			$\xleftarrow{\overline{X}, \overline{g}}$	$\overline{X} \leftarrow X^s, \overline{g} \leftarrow g^s$
		$K \leftarrow \overline{X}^y, \overline{Y} \leftarrow \overline{g}^y$		
		$AuthG \leftarrow Hash_2(C, G, X^*, \overline{Y}, K)$		
	$\xleftarrow{G, \overline{Y}, AuthG}$			
$K \leftarrow \overline{Y}^x$				
$AuthG' \leftarrow Hash_2(C, G, X^*, \overline{Y}, K)$				
$AuthG' \stackrel{?}{=} AuthG$				
$SK \leftarrow Hash_1(C, G, X^*, \overline{Y}, K)$		$SK \leftarrow Hash_1(C, G, X^*, \overline{Y}, K)$		
$accept \leftarrow true$		$accept \leftarrow true$		

FIG. 2.1 – $GPAKE$: Un protocole authentifié d'échange de clé par mot de passe

Protocole

1. Après avoir reçu C et X^*, la passerelle G retrouve l'indice (i, j) correspondant. Elle choisit 2 chaînes aléatoires S_0 et S_1 respectivement $\in \{0,1\}^m$ et dans $\in \{0,1\}^p$. Elle définit S'_0 et S'_1 par :
$$S'_0 = S_0 + \mathbb{1}_i \text{ et } S'_1 = S_1 + \mathbb{1}_j.$$
G envoie alors le triplé (X^*, S_0, S_1) à DB_{00}, (X^*, S'_0, S_1) à DB_{01}, (X^*, S_0, S'_1) à DB_{10} et enfin (X^*, S'_0, S'_1) à DB_{11}.
2. Les 4 bases de données choisissent un élément s aléatoire dans $\mathbb{Z}_q$. Cet élément est commun aux 4 bases de données. On rappelle que PW $\stackrel{\text{déf}}{=} h^{pw_{i,j}} \in \mathbb{G}$, avec $h \in \mathbb{G}$. Chaque base de données répond en envoyant un élément de $\mathbb{G}$. Pour notre protocole, une des bases de données utilise X^* pour ces calculs. On choisit par convention qu'il s'agit de DB_{00}.

DB_{00} calcule $pw_{00} \stackrel{\text{déf}}{=} S_0^T \cdot pw \cdot S_1 \mod q$. Et elle envoie $PW_{00} \stackrel{\text{déf}}{=} (\frac{X*}{h^{pw_{00}}})^s$.
DB_{01} calcule $pw_{01} \stackrel{\text{déf}}{=} S_0'^T.pw.S_1 \mod q$. Et elle envoie $PW_{01} \stackrel{\text{déf}}{=} (h^{pw_{01}})^s$.
DB_{10} calcule $pw_{10} \stackrel{\text{déf}}{=} S_0^T.pw.S'_1 \mod q$. Et elle envoie $PW_{10} \stackrel{\text{déf}}{=} (h^{pw_{10}})^s$.
DB_{11} calcule $pw_{11} \stackrel{\text{déf}}{=} S_0'^T.pw.S'_1 \mod q$. Et elle envoie $PW_{11} \stackrel{\text{déf}}{=} (h^{pw_{11}})^s$.

3. On rappelle que G veut retrouver $\overline{X}$, où $\overline{X} = X^s$. On remarque que X peut être calculé comme $X \leftarrow X^*/PW$. Dans ce protocole, S ne transmet pas X à G. Mais G reçoit $PW_{00}, PW_{01}, PW_{10}$ et PW_{11}. En regardant la valeur des bits $S_{0,i} = b_1$ et $S_{1,j} = b_2$, G retrouve la combinaison de division ou de multiplication correspondant à sa requête. Par convention, on admet que G calcule toujours la valeur suivante :
$$\overline{X} = \frac{PW_{00} \cdot PW_{01} \cdot PW_{10}}{PW_{11}}.$$
On considérera que G adapte le protocole selon la valeur de b_1 et de b_2.

Correction

En se rapportant à la sous-section 2.5.2, G retrouve bien la solution dans $\mathbb{G}$.

Protocole PP

En modifiant $GPAKE$, nous proposons le protocole final suivant :

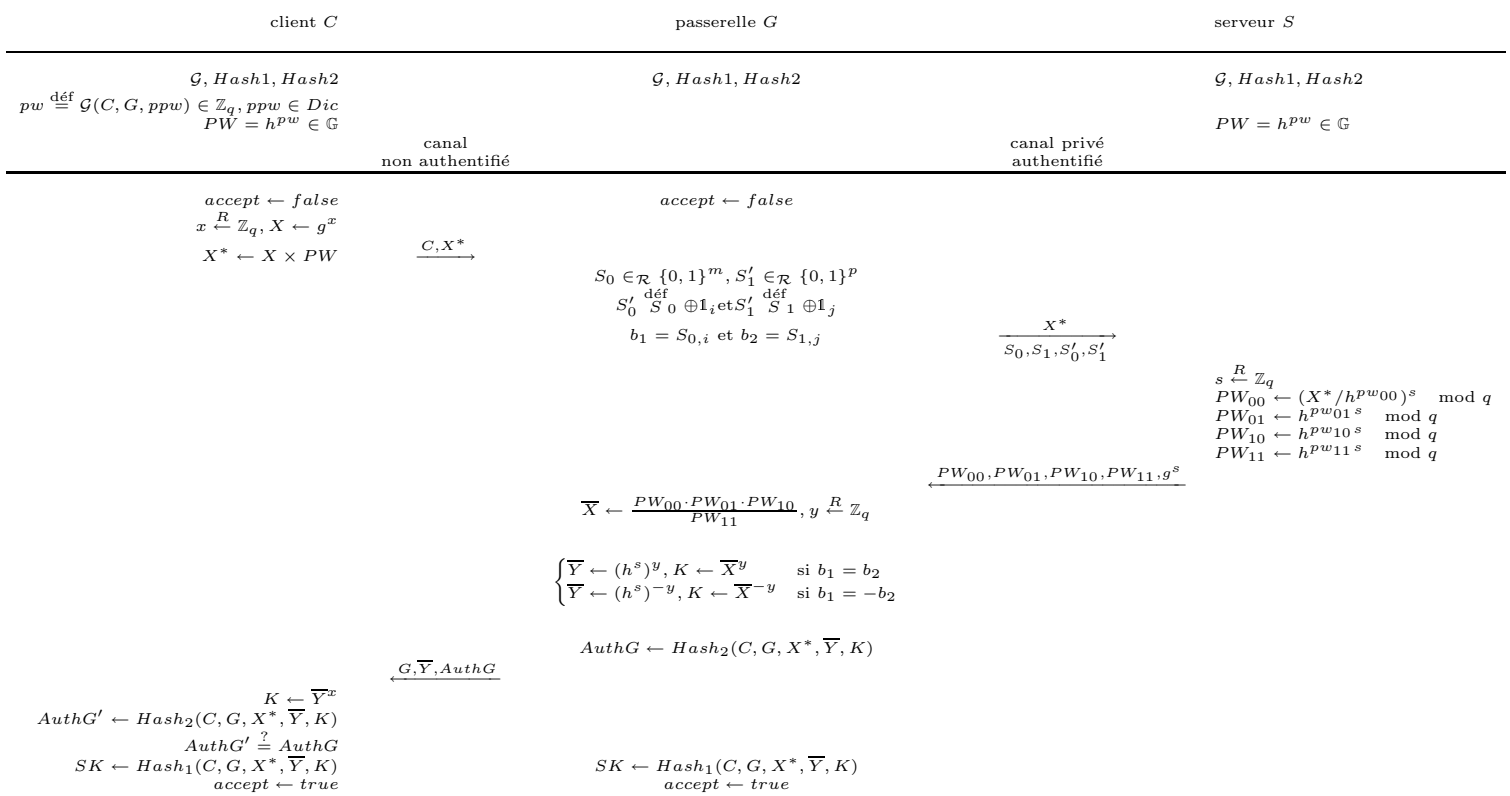

FIG. 2.2 – $GPAKE - PIR$: Protocole authentifié et anonyme d'échange de clé par mot de passe

Chapitre 3

Computational Private Information Retrieval, CPIR

Nous nous sommes intéressés jusqu'ici à des schémas parfaits au sens de la théorie de l'information i.e., que la puissance de calcul des bases de données n'était soumise à aucune contrainte calculatoire. On a vu dans ce cas que si on se limite à une seule base de données, la complexité de communication est nécessairement linéaire. Le seul moyen d'obtenir une complexité sous-linéaire est d'envisager la réplication des bases de données.

En 1997, plusieurs auteurs Chor, Gilboa, [8], Ostrovsky et Shoup [17] se sont intéressés à une nouvelle notion, celle de *Computationally-Private Information Retrieval*, pour lesquels les calculs des bases de données sont polynomiaux en la taille de leur contenu, n et la confidentialité de l'utilisateur est assurée sous réserve d'une hypothèse calculatoire. Les $CPIR$ ont plusieurs avantages : tout d'abord, cette primitive permet de réduire le nombre de bases de données. La plupart des protocoles n'utilisent qu'une seule base de données. Mais elle permet également de réduire considérablement la complexité de communication. Certains protocoles [7] aboutissent à une complexité logarithmique en n.

On donnera tout d'abord les définitions générales pour les $CPIR$ puis nous présenterons des exemples de chiffrements homomorphiques pour lesquels la sécurité repose sur une hypothèse calculatoire propre au cryptosystème. On présentera ensuite les protocoles correspondant à ces différents schémas de chiffrement et enfin nous proposerons une généralisation permettant de construire un PIR à partir d'un schéma de chiffrement homomorphe probabiliste pour lequel les paramètres généraux varient selon l'instance choisie. Nous précisons tout de même que les calculs restent malheureusement polynomiaux en n dans la plupart des protocoles.

3.1 Définitions

On se place ici dans le cas d'un protocole avec deux participants : un utilisateur, U et une base de données DB, tous deux limités à des calculs probabilistes et polynomiaux.

Protocole de Computational PIR

DB possède en entrée un paramètre de sécurité 1^k et une chaîne de caractères $x = x_1, \ldots, x_n$. U veut retrouver le bit x_i. Il possède en entrée le même paramètre de sécurité que DB.

1. U génère une requête $q(i)$ correspondant à l'élément qu'il cherche et lui permettant de masquer l'indice de cet élément.
2. DB génère une réponse $a(x, i)$ par des calculs uniquement polynomiaux.

3. À partir de *a(x,i)* et de calculs polynômiaux, U retrouve x_i.

Définition 3.1 *Pour les $CPIR$, la complexité de communication est mesurée en fonction de la taille de n et du paramètre de sécurité. On note $C(n,k)$ cette complexité. On dit qu'un schéma de $CPIR$ a une complexité égale à $C(n,k)$, si avec les notations précédentes, on a : $|q(i)| + |a(x,q(i))| < C(n,k)$.*

Pour un schéma de $CPIR$, un utilisateur doit aussi pouvoir retrouver le bit qu'il désire et la confidentialité de sa requête doit être préservée, ce qu'on formalise de la manière suivante :

Définition 3.2 *Un schéma de $CPIR$ à une seule base de données est un protocole à deux partis : un serveur qui contient n éléments et un utilisateur U qui possède un indice $i \in [n]$ où l'utilisateur retrouve l'élément x_i et qui garantit la propriété de sécurité ci-dessous.*

Sécurité

Soit P un protocole de $CPIR$ et soit A un adversaire qui joue le rôle de la base de données. Soit k un paramètre de sécurité et n la taille de la base de données. Dans un premier temps, on suppose que A donne en sortie deux indices distincts i_0 et $i_1 \in [n]$, indices pour lesquels A estime pouvoir distinguer les requêtes $q(i_0)$ et $q(i_1)$. On définit l'avantage de A à compromettre la sécurité de P par :

$$Adv^A_{P,n,k} = 2 * \max_{q(i_0),q(i_1)} |Pr[b \leftarrow \{0,1\}, r_Q \leftarrow \mathcal{R}_Q, A(1^k, q(i_b)) = b] - \frac{1}{2}|.$$

On dit que le protocole P préserve la confidentialité de l'utilisateur U ou que P est (ϵ, τ) sûr contre tout attaquant polynomial si et seulement si $Adv^A_{P,n,k} \leqslant \epsilon(k,n)$, avec $\epsilon(k,n)$ négligeable en la taille des paramètres k et n.

3.2 Hypothèses calculatoires

3.2.1 Le problème de la résiduosité quadratique

Ce problème a été introduit en 1984 par Goldwasser et Micali dans [11]. La partie qui suit rappelle le problème.

Définitions

Soit N un entier naturel. On définit $\mathbb{Z}_N^* \stackrel{\text{déf}}{=} \{x | 1 \leqslant x \leqslant N, \gcd(N,x) = 1\}$.

Définition 3.3 *On note $\mathcal{Q}_N(y) = 0$ s'il existe $w \in \mathbb{Z}_N^*$ tel que $w^2 = y \mod n$. On dira alors que y un résidu quadratique modulo n. Et si $\mathcal{Q}_N(y) = 1$, on dira alors que y n'est pas un résidu quadratique modulo n.*

Soit H_k l'ensemble défini par :

$$H_k \stackrel{\text{déf}}{=} \{N | N = p_1 p_2, \text{ où } p_1 \text{ et } p_2 \text{ sont deux entiers premiers de taille } \frac{k}{2} \text{ bits}\}.$$

Définition 3.4 *On considère uniquement les entiers $N \in H_k$ et $y \in \mathbb{Z}_N^*$ tel que $(\frac{y}{N}) = 1$, où $(.)$ est le symbole de Jacobi de y modulo N. On note alors $\mathbb{Z}_N^{+1} \stackrel{déf}{=} \{y \in \mathbb{Z}_N^* | (\frac{y}{N}) = 1\}$.*

Étant donnés un entier $N \in H_k$ et un élément $y \in \mathbb{Z}_N^{+1}$, le problème de décider la résiduosité quadratique est considéré comme un problème difficile

Si la factorisation de N est connue, on peut calculer $\mathcal{Q}_N(y)$ en temps $O(|N|^3)$. En utilisant le crible algébrique, on peut factoriser N en temps $2^{O(1)|N|^{\frac{1}{3}}(\log_2|N|)^{\frac{2}{3}}}$.

Soit $\mathcal{R}$ un espace de probabilité. À partir de l'hypothèse introduite, on définit la fonction de chiffrement suivante :

$$\mathcal{E} : \mathbb{Z}_2 \times \mathcal{R} \mapsto \mathbb{Z}_N^*.$$

Soit $g \in \mathbb{Z}_N^{+1}$ tel que $\mathcal{Q}_N(y) = 1$. On a pour tout $m \in \mathbb{Z}_2, r \in_{\mathcal{R}} \mathbb{Z}_N^*$ et $g \in \mathbb{Z}_N^*$, $\mathcal{E}(m,r) = g^m \cdot r^2$. La fonction $\mathcal{E}$ possède les propriétés suivantes :

- Tout d'abord pour m et m' deux éléments de $\mathbb{Z}_2$ et r, r', r'' des éléments aléatoires de $\mathbb{Z}_N^*$, on a :
$$\mathcal{E}(m,r) \cdot \mathcal{E}(m',r')) = \mathcal{E}(m \oplus m', r'').$$
- Et de plus,
$$\mathcal{E}(m,r)^c = \mathcal{E}(m \cdot c \mod 2, r).$$

remarque 3.1 *Pour choisir un élément aléatoire de $\mathbb{Z}_N^{+1}$, il suffit de choisir aléatoirement un bit aléatoires b, un élément r dans $\mathbb{Z}_N^*$ et de calculer $r^2 \cdot (-1)^b$.*

remarque 3.2 *$(\frac{y}{N})$ peut être calculer en temps polynomiale en la taille de N, pour N un entier quelconque, même sans connaître la factorisation de N. En effet, si $(\frac{y}{N}) = -1$, y n'est jamais un résidu quadratique, en revanche parmi les y tel que $(\frac{y}{N}) = 1$, la moitié sont des résidus quadratiques modulo N et les autres n'en sont pas. Et pour tout $x, y \in \mathbb{Z}_N^*$, on a $\mathcal{Q}_N(x \cdot y) = \mathcal{Q}_N(x) \oplus \mathcal{Q}_N(y)$.*

Difficulté du problème RQ

Avec les notations qui viennent d'être introduites, on peut à présent formaliser, l'hypothèse relative à la difficulté du problème de déterminer la résiduosité quadratique dans un groupe $\mathbb{Z}_N^*$.

Hypothèse 3.1 *RQ Pour toute constante c, et tout algorithme polynomiale et probabiliste $\mathcal{A}$, il existe un entier K tel que pour tout $k > K$,*

$$Prob_{N \in_{\mathcal{R}} H_k; y \in_{\mathcal{R}} \mathbb{Z}_N^{+1}}(C_k(N,y) = Q_N(y)) < \frac{1}{2} + \frac{1}{k^c}, \text{ où } N \in_{\mathcal{R}} H_k, y \in_{\mathcal{R}} \mathbb{Z}_N^{+1}.$$

Cette hypothèse signifie donc qu'étant donné un $N \in_{\mathcal{R}} H_k, y \in_{\mathcal{R}} \mathbb{Z}_N^{+1}$ choisi aléatoirement, tout attaquant polynomial contre le problème de la résiduosité quadratique a un avantage négligeable.

3.2.2 Problème de la haute résiduosité

Cryptosystème de Paillier, [18]

Ce schéma a été introduit par Pascal Paillier en 1999 dans [18]. Nous allons tout d'abord faire quelques rappels mathématiques utiles à la compréhension du cryptosystème de Paillier. $N = p * q$ est un module RSA, avec $p < q$. On suppose que p ne divise pas $q - 1$, autrement dit $\gcd(N, \phi(N)) = 1$.

Définition 3.5 *Un élément z est un $N_{ième}$ résidu modulo N^2 s'il existe un élément $y \in \mathbb{Z}_{N^2}^*$ tel que $z = y^N \mod N^2$.*

Définition 3.6 *Pour N produit de deux entiers premiers p et q, on définit la fonction d'Euler que l'on note $\phi(N)$, où $\phi(N) = (p-1)(q-1)$.*

Dans l'article de Paillier, on définit également la fonction de Carmichaël :

Définition 3.7 *Pour N produit de deux entiers premiers p et q, on définit la fonction de Carmichaël que l'on note $\lambda(N)$ définie par $\lambda(N) = ppcm(p-1, q-1)$ et telle que : pour tout $w \in \mathbb{Z}^*_{N^2}, w^{\lambda(N)} = 1 \mod n$ et $w^{\lambda(N)N} = 1 \mod N^2$.*

Pour le déchiffrement, on définit également l'ensemble S_N et la fonction L :

Définition 3.8

$$S_N = \{u < N^2 | u = 1 \mod N\}.$$

Définition 3.9 *Pour $u \in \mathbb{Z}^*_{N^2}$, on définit :*

$$L(u) = \frac{u-1}{N} \mod N^2.$$

Afin de comprendre le cryptosystème, on va d'abord montrer que $\mathbb{Z}^*_{N^2} \simeq \mathbb{Z}_N \times \mathbb{Z}^*_N$. Soit g un élément d'ordre N dans $\mathbb{Z}^*_{N^2}$ et H un groupe tel que $H \simeq \mathbb{Z}^*_N$. Donc H est d'ordre $\phi(N)$. On définit le groupe quotient $G \simeq \mathbb{Z}^*_{N^2}/H$. Alors G est cyclique d'ordre N et $G \simeq \mathbb{Z}_N$. On notera $< gH >$ le groupe $\{g, gh, gh^2, \ldots, gh^{\phi(N)-1}\}$. Les preuves qui suivent pourront être trouvées dans l'article [16].

Lemme 1 *Soit p un entier premier. Alors $\mathbb{Z}^*_{p^2} \simeq \mathbb{Z}_p \times \mathbb{Z}^*_p$.*

preuve 3.1 *On peut déjà remarquer que $|\mathbb{Z}^*_{p^2}| = \phi(p^2) = p(p-1)$. Il existe donc un élément d'ordre p dans $\mathbb{Z}^*_{p^2}$, par exemple $p+1$. Donc il existe un sous-groupe de $\mathbb{Z}^*_{p^2}$ d'ordre p, que l'on notera H_p. Soit g un générateur du groupe cyclique $\mathbb{Z}^*_p$ d'ordre $p-1$. g est donc un générateur de $\mathbb{Z}^*_p$. L'ordre de g est au moins $p-1$ dans $\mathbb{Z}^*_{p^2}$. Donc $|g| = p-1$ ou $|g| = p(p-1)$.*

- *Supposons que $|g| = p-1$. Alors il y a un sous-groupe cyclique $< g >$ dans $\mathbb{Z}^*_{p^2}$ d'ordre $p-1$. Comme $\gcd(p, p-1) = 1$, on a $H_p \cap < g >= 1$. Et donc $\mathbb{Z}^*_{p^2} \simeq H_p \times < g > \simeq \mathbb{Z}_p \times \mathbb{Z}^*_p$.*
- *Dans l'autre cas, si $|g| = p(p-1)$, alors $\mathbb{Z}^*_{p^2}$ est cyclique et alors,*

$$\mathbb{Z}^*_{p^2} \simeq \mathbb{Z}_{p(p-1)} \simeq \mathbb{Z}_p \times \mathbb{Z}_{p-1} \simeq \mathbb{Z}_p \times \mathbb{Z}^*_p, \text{ d'où le résultat .}$$

Lemme 2 *Soit N un entier tel que $N = pq$, où p et q sont premiers. Alors $\mathbb{Z}^*_{N^2} \simeq \mathbb{Z}_N \times \mathbb{Z}^*_N$.*

preuve 3.2 *On peut déjà noter que $\mathbb{Z}_{N^2} \simeq \mathbb{Z}_{p^2} \times \mathbb{Z}_{q^2}$ et que $\mathbb{Z}^*_{N^2} \simeq \mathbb{Z}^*_{p^2} \times \mathbb{Z}^*_{q^2}$. Si on applique le lemme 1, on obtient : $\mathbb{Z}^*_{N^2} \simeq \mathbb{Z}_p \times \mathbb{Z}^*_p \times \mathbb{Z}_q \times \mathbb{Z}^*_q$, i.e., :*

$$\mathbb{Z}^*_{N^2} \simeq (\mathbb{Z}_p \times \mathbb{Z}_q) \times (\mathbb{Z}^*_p \times \mathbb{Z}^*_q) \simeq \mathbb{Z}_N \times \mathbb{Z}^*_N.$$

On remarque alors que le sous-groupe du produit direct $\mathbb{Z}_N \times \mathbb{Z}^*_N$ isomorphe à $\mathbb{Z}^*_N$ est l'unique sous-groupe de $\mathbb{Z}^*_{N^2}$ d'ordre divisible par $(p-1)(q-1)$. nous avons noté H ce groupe.

Lemme 3 *Soit N un entier tel que $N = pq$, où p et q sont premiers. Les $N_{ième}$ résidus modulo N^2 sont exactement les éléments de H.*

preuve 3.3 *Il s'agit de montrer qu'un élément $h \in \mathbb{Z}_{N^2}^*$ possède une racine $N_{ième}$ modulo N si et seulement si $h \in H$. On définit l'application $\psi : \mathbb{Z}_{N^2}^* \mapsto \mathbb{Z}_{N^2}^*$ tel que $\psi(x) = x^N$. Montrons que ψ est un homomorphisme de noyau isomorphe à $\mathbb{Z}_N$ et ayant pour image le sous-groupe H i.e., $Ker(\psi) \simeq \mathbb{Z}_N$ et donc $|Im(\psi)| = |H|$. De manière évidente, $Im(\psi)$ est exactement le groupe des $N_{ième}$ résidus modulo N^2. Un élément est dans le noyau si et seulement si son ordre est divisible par N , i.e., d'ordre $1, p, q$ ou N. D'après ce qui précède, on a $\mathbb{Z}_{N^2}^* \simeq \mathbb{Z}_N \times \mathbb{Z}_N^*$. On rappelle que $\gcd(N, \phi(N)) = 1$, donc $\mathbb{Z}_N$ contient les éléments de $\mathbb{Z}_{N^2}^*$ d'ordre $1, p, q$ ou N. Comme H est l'unique sous-groupe de $\mathbb{Z}_{N^2}^*$ d'ordre $(p-1)(q-1)$ et que $Im(\psi)$ est précisément le sous-groupe de $\mathbb{Z}_{N^2}^*$ des $N_{ième}$ résidus modulo N, on a donc $Ker(\psi) \simeq \mathbb{Z}_N$ et $|Im(\psi)| = |H|$.*

Description du cryptosystème de Paillier

Soit g un élément non nul de $\mathbb{Z}_{N^2}^*$ d'ordre un multiple de N.

Définition 3.10 *Pour $g \in \mathbb{Z}_{N^2}^*$, on note $\mathcal{E}_g$ la fonction définie par : $\mathcal{E}_g(m, r) = g^m \cdot r^N \mod N^2$, où $m \in \mathbb{Z}_N$ et $r \in \mathbb{Z}_N^*$.*

On cherche à trouver des fonctions $\mathcal{E}_g$ pour lesquels à partir de $\mathcal{E}_g(m, r)$, il est difficile de déchiffrer m.

Lemme 4 *Si l'ordre de g est un multiple non nul de N, alors la fonction $\mathcal{E}_g$ est bijective.*

On pourra trouver la preuve dans l'article de Paillier [18]. Cette propriété est importante si on veut pouvoir déchiffrer un message. En effet, comme dans tout cryptosystème pour pouvoir déchiffrer un message, tout chiffré ne doit posséder qu'un unique antécédent par la fonction de chiffrement, ce qui est le cas ici si g vérifie les conditions du lemme ci-dessus.

Il s'agit à présent de trouver un élément g qui convient. L'ordre de g dans $\mathbb{Z}_{N^2}^*$ est le plus petit l tel que $g^l = 1 \mod N$. On sait que l'ordre de g dans $\mathbb{Z}_{N^2}^*$est un multiple de N i.e., $l = N\alpha \mod N^2$ pour certains α, ce qui signifie que $g^{N\alpha} = 1 \mod N^2$. Par définition de la fonction de Carmichaël, on sait que l'ordre maximal de g est $N\lambda$, on a donc $1 \leqslant \alpha \leqslant \lambda(N)$. Dans son article, Paillier note β les éléments de $\mathbb{Z}_{N^2}^*$ d'ordre $N \cdot \alpha$, avec $1 \leqslant \alpha \leqslant \lambda(N)$, ce qui permet de sélectionner les éléments g qui convienne. Pour trouver g, Paillier suggère de prendre un élément aléatoire de $\mathbb{Z}_{N^2}^*$ et de vérifier que :

$$\gcd(L(g^\lambda \mod N^2), N) = 1.$$

Mais il s'avère qu'en pratique cela n'est pas si simple. C'est pourquoi plusieurs auteurs prennent $g = 1 + N \in \beta$, d'ordre exactement N ce qui simplifie énormément les démonstrations et les calculs. Comme tout élément $g \in \beta$ convient, il suffit de considérer le cas où $g = 1 + N$. En effet, la sécurité sémantique du cryptosystème ne dépend pas du choix de g. Ce résultat est démontré dans la thèse de Jurik(Chap2 p13-16).

Définition 3.11 *Soit $g \in \beta$. Pour $w \in \mathbb{Z}_{Nr}^*$ et g fixé, on appelle la $N_{ième}$ classe de résiduosité de w l'unique $m \in \mathbb{Z}_N$ pour lequel il existe $r \in \mathbb{Z}_N^*$ tels que $\mathcal{E}_g(m, r) = w$.*

Définition 3.12 *Si $w \in \mathbb{Z}_{N^2}^*$, on définit w_g comme le plus petit entier m non nul tel que $w = g^m \cdot r^N$, pour $g \in \mathbb{Z}_{N^2}^*$ et $r \in \mathbb{Z}_N^*$.*

On note $\mathcal{D}_g$ la fonction de déchiffrement associé à $\mathcal{E}_g$. On remarque que la fonction $\mathcal{E}_g$ possède les propriétés suivantes :

– Pour m et m' deux messages de $\mathbb{Z}_N$ et r, r' des éléments aléatoires de $\mathbb{Z}_N^*$, on a :

$$\mathcal{E}_g(m; r) \cdot \mathcal{E}_g(m'; r')) = \mathcal{E}_g(m + m' \mod N; r \cdot r') \mod N^2.$$

– Et de plus :

$$\mathcal{E}_g(m, r)^c = \mathcal{E}_g(c \cdot m \mod N; r) \mod N^2.$$

Hypothèse HR : détermination la classe de résiduosité.

Hypothèse 3.2 *(HR) Si la factorisation de N est inconnue, il n'y a pas d'algorithme polynomial qui permette de distinguer les $N_{ième}$ résidus modulo N^2.*

Comme nous l'avons précisé auparavant, on pose $g = 1 + N$. Avec les notations précédentes pour $w \in \mathbb{Z}^*_{N^2}$, on a $w = (1+N)^m \cdot r^N \mod N^2$. On cherche donc l'exposant de w_g en base $1 + N$, i.e., m.

Lemme 5 *Pour tout $w \in \mathbb{Z}^*_{N^2}$, on a $L(w^\lambda \mod N^2) = \lambda w_g \mod N$.*

preuve 3.4 *[18] :*

$$\begin{aligned} w^\lambda &= (1+N)^{m\lambda} r^{N\lambda} \\ &= (1+N)^{m\lambda} \\ &= 1 + m\lambda N \mod N^2. \end{aligned}$$

Ce qui permet d'appliquer la fonction L et on obtient :

$$L(w^\lambda \mod N^2) = L(1 + m\lambda N) = \lambda m, \text{ où } m = \overline{w}^g.$$

3.2.3 Extension du problème de la haute résiduosité

Ce problème a été introduit par Damgard-Jurik peu après le cryptosystème de Paillier, en 2000 dans [10]. Il s'agit d'une généralisation du schéma de Pailler pour laquelle on peut chiffrer des messages de longueur quelconque sans modifier les paramètres publics de départ et tout en ayant la même clé privée que dans le cryptosystème de Paillier. De plus, la sécurité de ce cryptosystème repose sur la même hypothèse calculatoire. On choisit toujours les mêmes notations et définitions pour le module N.

Chiffrement

Définition 3.13 *Pour $g \in \mathbb{Z}^*_{N^{s+1}}$, et $s < p, q$, on définit $\mathcal{E}_g = g^m \cdot r^N \mod N^{s+1}$, où $m \in \mathbb{Z}_{N^s}$ et $r \in \mathbb{Z}_{N^*}$.*

En généralisant ce qui précède, on obtient les résultats suivant : On a $\mathbb{Z}_{N^s} \times \mathbb{Z}_{N^*} \simeq \mathbb{Z}^*_{N^{s+1}}$. Le cas $s = 1$ correspond au cryptosystème de Pailler. Dans [5], on montre que $\mathbb{Z}^*_{N^{s+1}}$ est un groupe multiplicatif produit direct de $G \times H$, avec H isomorphe à $\mathbb{Z}^*_N$ et G un sous groupe cyclique de $\mathbb{Z}^*_{N^{s+1}}$ d'ordre N^s. On a donc $|\mathbb{Z}^*_{N^{s+1}}| = \phi(N^{s+1}) = N^s\phi(N)$. Soit g un élément d'ordre N^s dans $\mathbb{Z}^*_{N^{s+1}}$ non nul. Alors $H, gH, g^2H, \ldots, g^{N^s-1}H$ engendre H dans $\mathbb{Z}^*_{N^{s+1}}$. Dans [5], on montre que $g \in (1+N)^j \cdot h$, pour un certain entier $j < n$ et $h \in \mathbb{Z}_{N^*}$. Pour les mêmes raisons que celles évoquées dans la section précédentes, on considère que $g = (1 + N)$.

Notre objectif étant d'utiliser ce cryptosystème pour la construction de protocole de PIR, nous adapterons les définitions des fonctions de chiffrement et de déchiffrement.

Définition 3.14 *Pour $g \in \mathbb{Z}^*_{N^{s+j+1}}$ et s un entier fixé, on considérera ici la fonction de chiffrement $\mathcal{E}_s$ définie par :*

$$\mathcal{E}_s : \mathbb{Z}_{N^s} \times \mathbb{Z}^*_N \mapsto \mathbb{Z}^*_{N^{s+1}} \text{ tel que } \mathcal{E}_j(m, r) = c = g^m \cdot r^{N^s} \mod N^{s+1} \text{ avec } m \in \mathbb{Z}_{N^s} \text{ et } r \in \mathbb{Z}^*_N.$$

Les fonctions de chiffrement $\mathcal{E}_s$ possèdent les propriétés suivantes :

– Pour tout $m, m' \in \mathbb{Z}_{|\mathcal{M}_s|}$ et pour tout $r, r' \in \mathcal{R}$, on a :

$$\mathcal{E}_s(m, r) \cdot \mathcal{E}_s(m', r') = \mathcal{E}_j(m + m', r \cdot r'),$$

où $\cdot$ est une opération multiplicative dans $\mathcal{C}_{s+j}$, $+$ est une opération additive dans $\mathbb{Z}_{|\mathcal{M}_s|}$.

– Pour tout entier $c \in \phi(N^2)$, on a :

$$\mathcal{E}_s(m, r)^c = \mathcal{E}_s(m \cdot c, r) \mod N^2.$$

Déchiffrement

Pour tout chiffré c, on a alors $c = (1+N)^m \cdot r^{N^s} \mod N^{s+1}$. Afin de déchiffrer m, on cherche à éliminer la partie de l'expression de droite appartenant à H. Soit d un multiple de $\lambda(N)$ non nul. Alors :

$$\begin{aligned} c^d &= (1+N)^{m \cdot d}(r^{N^s})^d \mod N^{s+1} \\ &= (1+N)^{m \cdot d} r^{N^s \cdot d} \mod N^{s+1} \\ &= (1+N)^{m \cdot d} \mod N^{s+1}. \end{aligned}$$

Par définition de $\lambda(N)$, $r^{\lambda(N)} = 1 \mod N$. D'autre part, comme N^{s+1} est un multiple de $N, r^{N^s \cdot d} = r^{N^s(p-1)(q-1)} = 1 \mod N^{s+j+1}$ puisque $N^s \cdot d$ est un multiple de $\phi(N)$. Pour déchiffrer m, il s'agit donc de trouver le logarithme discret de c en base $1 + N$ dans $\mathbb{Z}_{N^s}$. On a vu que $(1+N)^m = 1 + nm \mod N^2$. Nous allons donner les étapes de l'algorithme de déchiffrement présenté dans [5] :

$$L((1+N)^i \mod N^{s+1}) = (i + \binom{i}{2} N + \cdots + \binom{i}{s} \cdot N^{s-1}) \mod N^s \qquad (1).$$

Il s'agit d'extraire pas à pas les résidus modulo les puissances successives de N. Plus précisément, on définit :

$$\begin{aligned} i_1 &= i \mod N \\ i_2 &= i \mod N^2 \\ &\vdots \\ i_j &= i \mod N^j. \end{aligned}$$

On sait déjà extraire i_1 grâce au déchiffrement de Pailler. À une étape donnée, on suppose avoir i_s, il s'agit de trouver i_{s+1}.

$$i_j = i_{j-1} + k * N^{j-1} \qquad (2).$$

$$L((1+N)^i \mod N^{j+1}) = (i_j + \binom{i_j}{2} N + \cdots + \binom{i_j}{j} N^{j-1} \mod N^j \qquad (3).$$

$$\text{Pour } j > t > 0, \text{ on a, } \binom{i_j}{t+1} N^t = \binom{i_{j-1}}{t+1} N^t \mod N^j \qquad (4).$$

En réécrivant i_s à l'aide de l'équation 2, on a :

$$\binom{i_j}{t+1} N^t = \binom{i_{j-1} + k * N^{j-1}}{t+1} N^t \mod N^j.$$

Lorsqu'on évalue le produit de $k * N^{j-1}$ et N^T modulo N^j, ce terme s'annule. Ce qui nous donne l'équation suivante :

$$L((1+N)^i \mod N^{j+1}) = (i_{j-1}+k*N^{j-1}+\binom{i_{j-1}}{2}N+\cdots+\binom{i_{j-1}}{j}N^{j-1}) \mod N^j \quad (5).$$

En combinant les équations (2) et (4), on obtient l'équation finale qui permet de récupérer i_j :

$$i_j = L((1+N)^i \mod N^{j+1}) - (i_{j-1} + \binom{i_{j-1}}{2}N + \cdots + \binom{i_{j-1}}{j}N^{j-1}) \mod N^j \quad (6).$$

On pourra trouver dans un algorithme permettant de calculer i_j pour toute valeur de j. Désormais on peut calculer m tel que $i = m \cdot d \mod N^s$ en calculant $d^{-1} \mod N^s$, puisque $m \cdot d \cdot d^{-1} = m \mod N^s$.

3.3 Protocoles de « PIR homomorphes »

3.3.1 Protocole RQ, [13]

Le paramètre de sécurité sera ici défini par la taille de l'entier N, noté $|N|$. Dans ce protocole, la base de données x est vue comme une matrice de taille $n = s*t$. L'utilisateur U veut retrouver le bit x_i, avec $i = (a, b)$.

1. U choisit aléatoirement deux nombres premiers p_1 et p_2 tels que $N = p_1p_2 \in H_k$. Il garde la factorisation de N secrète.
2. U choisit aléatoirement t nombres premiers $y_1, \ldots, y_t \in \mathbb{Z}_N^{+1}$ tel que $\mathcal{Q}_N(y_b) = 1$ et $\mathcal{Q}_N(y_j) = 0$, pour $j \neq b$. U envoie ces t entiers à la base de données, i.e., $t * k$ bits.
3. Pour toute ligne $r \in [s]$, la base de données calcule un entier $z_r \in \mathbb{Z}_N^*$, en calculant tout d'abord :
$$w_{r,j} = \begin{cases} {y_j}^2 \cdot h_{r,j}^2 & \text{si } x_{r,j} = 0 \\ y_j \cdot h_{r,j}^2 & \text{si } x_{r,j} = 1 \end{cases},$$
où $h_{r,j}$ est un élément aléatoire de $\mathbb{Z}_N^*$ DB calcule alors $z_r \overset{\text{déf}}{=} \prod_{j=1}^t w_{r,j}$. DB envoie $z_1, \ldots, z_s$ à U, i.e., $s \cdot k$ bits.
4. U considère uniquement le z_a correspondant à la ligne contenant le bit qui l'intéresse, $x_{a,b}$. On remarque que si $j \neq b, w_{r,j}$ est toujours un résidu quadratique et si $j = b$, alors $\mathcal{Q}_N(w_{r,b}) = 0$ si et seulement si $x_{a,b} = 0$, d'après la remarque 3.2. Donc z_r est un résidu quadratique si et seulement si $x_{a,b} = 0$. De ce fait, à partir de z_a et de la factorisation de N, que lui seul connaît, U retrouve le bit $x_{a,b}$.

3.3.2 Complexité

Les éléments envoyés sont successivement $(N, y_1, \ldots, y_t, z_1, \ldots, z_s)$. U envoie $1 + t$ éléments du groupe et la base de données envoie s éléments du groupe. La complexité globale est donc en $(1 + s + t) \cdot k$ bits de communication si k est la taille des éléments du groupe. En prenant $s = t = \sqrt{n}$, on obtient une complexité globale égale à $(2\sqrt{n} + 1)k$ bits, i.e., en $O(n^{\frac{1}{2}+c})$, avec pour paramètre de sécurité k égale à n^c, pour c une constante > 0.

Pour la complexité calculatoire, on considérera l'algorithme naïf pour la multiplication modulo N, en $O(|N|^2)$, même si on peut faire mieux avec la transformée de Fourrier. La base de

données doit faire en moyenne n multiplications, car elle calcule pour chaque ligne, i.e., s un produit de t éléments de $\mathbb{Z}_N^{+1}$ qui ont éventuellement été calculés auparavant. Quant à l'utilisateur, il veut savoir si z_a est un carré modulo N, connaissant la factorisation de N. Ce qui peut être fait en un temps $O(|N|^3)$. La complexité calculatoire est donc cubique en la taille de N, i.e., en $k = n^c$.

remarque 3.3 *On peut remarquer que c'est la représentation de la base de données sous la forme d'une matrice qui permet d'équilibrer la répartition des bits envoyés des deux côtés. Cette méthode est d'ailleurs générique. L'utilisateur envoie $s + 1 = \sqrt{n} + 1$ éléments du groupe et la base de données envoie $t = \sqrt{n}$ éléments du groupe. Ce qui revient à appliquer la transformation par bloc de la section 2.4 à ce même schéma avec pour cas particulier $t = 1$. Le protocole de la section suivante est construit sur cette idée. On construit d'abord un protocole où n éléments sont envoyés du côté utilisateur puis on équilibre par cette transformation qui lui permet d'en envoyer moins.*

3.3.3 Preuve de sécurité

Théorème 3.1 *Sous l'hypothèse **RQ**, pour tout $c > 0$, il existe un CPIR avec une seule base de données et une complexité en $O(n^{0.5+c})$.*

On pourra trouver cette preuve dans l'article [13]. C'est une preuve par l'absurde. On suppose d'abord qu'il existe un distingueur **B** capable de distinguer deux requêtes différentes correspondant aux indices $i_1 = (a, b)$ et $i_2 = (a', b')$. On va construire un algorithme $A_k(N, y)$ suivant le modèle standard, ce qui contredira l'hypothèse QR. On notera Q l'algorithme de requête.
Voyons tout d'abord comment on se ramène à ne considérer que le cas où $b \neq b'$. On peut déjà remarquer que le protocole est indépendant de a.

- Supposons avoir à notre disposition un algorithme **B** qui renvoie les deux indices i_1 et i_2, pour lesquels il estime pouvoir distinguer $Q(i_1)$ et $Q(i_2)$. **B** reçoit en entrée une requête, $Q(i_c)$ comme dans le protocole. **B** renvoie une réponse c'. La probabilité de succès de **B** est par définition Prob$[c = c']$.
  1. Supposons que $b = b'$. Dans ce cas, **B** est incapable de distinguer $Q(i_1)$ et $Q(i_2)$. En effet, les deux distributions sont parfaitement aléatoires et indistinguables pour **B**.
  2. Si $b \neq b'$,
- dans ce cas, on peut construire un attaquant $A(N, y)$ contre l'hypothèse RQ : étant donnés $N \in_{\mathcal{R}} H_k, y \in_{\mathcal{R}} \mathbb{Z}_N^{+1}$, $A_k(N, y)$ fabrique une requête pour **B** à partir de ce qu'il possède en entrée et simule **B**. $A_k(N, y)$ renvoie 1 si **B** a bien deviné et 0 dans l'autre cas.

preuve 3.5
- ***B** a en entrée une séquence $(N, (y_1, \ldots, y_t))$ comme dans le protocole. **B** renvoie 1 avec probabilité p si la position choisie est b i.e., si U veut retrouver le bit $x_{*,b}$ et **B** renvoie 1 avec probabilité $p + \epsilon$ si la position choisie est b'.*
- *La construction qui suit montre qu'à l'aide de **B**, on peut alors construire un attaquant polynomial $A_k(N, y)$ qui résout le problème de la résiduosité quadratique avec probabilité non négligeable, ce qui est contraire à l'hypothèse QR.*
  A_k possède en entrée un couple (N, y), comme décrit précédemment.
  1. *On choisit aléatoirement $t - 2$ résidus quadratiques dans $\mathbb{Z}_N^*$. On les place à une position quelconque sauf en b ou b'.*
  2. *On place y, l'élément dont on veut déterminer la résiduosité quadratique soit à la position b ou b'. Et on choisit un autre résidu quadratique qu'on place à la position restante.*

3. *On fait ensuite appel à l'algorithme* $\boldsymbol{B}$ *avec en entrée la séquence construite : si la position choisie est* b'*, on renvoie la même réponse que* $\boldsymbol{B}$*. Sinon, on renvoie la réponse flippée de* $\boldsymbol{B}$*.*

Évaluons la probabilité que $A_k(N, y)$ *renvoie* 1 *sachant que* y *est un résidu quadratique modulo* N*. Dans ce cas, quelle que soit la position choisie* b *ou* b'*,* $\boldsymbol{B}$ *possède alors en entrée* t *éléments aléatoires qui sont tous des résidus quadratiques.*
Soit $q \stackrel{déf}{=} Prob(B = 1|\mathcal{Q}_N(y) = 0)$*. Alors :*

$$\begin{aligned} Prob_{N\in_{\mathcal{R}}H_k;y\in_{\mathcal{R}}\mathbb{Z}_N^{+1}}(C_k(N,y) = 1|\mathcal{Q}_N(y) = 0) \\ &= Prob(B = 1|\mathcal{Q}_N(y) = 0) * [Prob(y = y_b) + Prob(y = y'_b)] \\ &= q * \frac{1}{2} + (1-q)\frac{1}{2} = \frac{1}{2}. \end{aligned}$$

Dans ce cas, l'avantage d'un attaquant est nul puisque la position c *est choisie aléatoirement par* A*. A présent si* y *n'est pas un résidu quadratique modulo* N*, la séquence possédée par* $\boldsymbol{B}$ *en entrée contient exactement un élément qui n'est pas un résidu quadratique modulo* N*. On a :*

$$\begin{aligned} &Prob_{N\in_{\mathcal{R}}H_k;y\in_{\mathcal{R}}\mathbb{Z}_N^{+1}}(C_k(N,y) = 1|\mathcal{Q}_N(y) = 1) \\ &= Prob(B = 1|\mathcal{Q}_N(y) = 1) * Prob(y = y_b) + Prob(B = 1|\mathcal{Q}_N(y) = 1) * Prob(y = y'_b) \\ &= Prob(B = 1|y = y_b) * Prob(y = y_b) + Prob(B = 1|y = y'_b) * Prob(y = y'_b) \\ &= (p+\epsilon)\frac{1}{2} + (1-p)\frac{1}{2} = \frac{1}{2} + \frac{\epsilon}{2} \end{aligned} .$$

Dans ce cas, la probabilité de succès de $\boldsymbol{B}$ *est* $\frac{1}{2} + \frac{\epsilon}{2}$ *et l'avantage de l'attaquant est* $\epsilon/2$*.*
On a construit un algorithme polynomial avec en entrée y *et* N *qui détermine la résiduosité quadratique avec probabilité au moins* $\frac{1}{2} + \frac{\epsilon}{2}$*. Ce qui signifie que notre hypothèse de départ est fausse, cet attaquant ne peut donc pas exister.*

remarque 3.4 *Dans [13], Kushilevitz et Ostrovsky montrent qu'il existe un schéma avec une seule base de données et une communication en* $2^{\sqrt{\log_2 n.\log_2(\log_2 n)}}$*. Ils prennent pour paramètre de sécurité* $k = \Omega(\log^{3-o(1)} n)$*.*

3.3.4 Schéma récursif en $O(n^\epsilon)$

En appliquant la méthode récursive de Kushilevitz et Ostrovsky, on peut améliorer la complexité du protocole précédent. Il est important de remarquer que cette méthode ne peut être appliquée que dans le cas où le protocole d'origine est interactif et consiste en un seul échange d'informations entre les deux joueurs.
Dans le schéma précédent, l'utilisateur veut retrouver au départ un seul élément de $\mathbb{Z}_N^*$. Mais il reçoit plusieurs valeurs $z_1, z_2, \ldots, z_s$ de la base de données. Il suffit alors de faire en sorte que l'utilisateur ne reçoive au final exactement que ce qui l'intéresse, i.e., l'élément de z_a si l'indice secret est (a, b) ; ce qui revient en quelque sorte à équilibrer le nombre de bits des deux côtés. Le problème peut être vu alors de la manière suivante : la base de données x est vue comme une chaîne d'éléments de longueur $s \cdot k$ découpée en blocs de taille égale k. U cherche à récupérer un bloc entier.

z_1	$\ldots$	z_a	$\ldots$	z_s

Dans le protocole d'origine, la base de données S a pour taille n. On construit une suite de schémas S_j pour lesquels la taille de la base de données diminue à chaque niveau de la récursion. On suppose qu'il y a L étapes. On note n_j le nombre d'éléments secrets que possèdent la base de données à l'étape j de la récursion. Chacun des éléments est de longueur k bits. On note S_j le $j_{\text{ième}}$ protocole et t_j le nombre de colonnes virtuelles de la matrice utilisée pour représenter la base de données à cette étape. On a donc $t_j = \frac{n_j}{s_j}$.

L'idée du protocole récursif est la suivante : pour chaque protocole S_j, on exécute k fois le protocole S_{j+1} qui permet à U de récupérer de manière virtuelle les k bits de l'élément x_i. Il utilise à chaque étape une chaîne de longueur plus petite n_j. On obtient le protocole ci-dessous en faisant une seule modification à l'étape du protocole S à laquelle l'utilisateur reçoit plus d'informations que nécessaire. Il s'agit de l'étape 4, l'étape à laquelle la base de données envoient les éléments à l'utilisateur.

Protocole récursif

– Au niveau $j = 1$ de la récursion, la base de données est de taille $n = s \cdot k$. Le protocole de départ S_1 correspond au protocole décrit à la section 3.3.1.

1. U choisit aléatoirement deux nombres premiers p_1 et p_2 tel que $N = p_1 p_2 \in H_k$. Il garde la factorisation de N secrète. U choisit aléatoirement t nombres premiers $y_1, \ldots, y_t \in \mathbb{Z}_N^{+1}$ tel que $\mathcal{Q}_N(y_b) = 1$ et $\mathcal{Q}_N(y_j) = 0$, pour $j \neq b$. U envoie ces t entiers à la base de données i.e., $t \cdot k$ bits.

2. On rappelle que U veut récupérer k bits. U exécute k fois le protocole S_2 de manière indépendante. Chacune des k exécutions correspond à la récupération d'un bit.

3. À l'étape j de la récursion, U exécute k fois le protocole S_{j+1} sur une base de données plus petite de taille $n_{j+1} = k \cdot s_j$. À chaque étape, la base de données n'envoie aucun élément, mais U retrouve de manière virtuelle les k bits en réitérant les exécutions pour chacune des k branches de l'étape précédente.

– Pour une étape donnée j, on choisit le même N et les k bases de données utilisent à chaque fois la même requête $y_1, \ldots, y_t \in \mathbb{Z}_N^{+1}$ tel que $\mathcal{Q}_N(y_b) = 1$. À la fin des L étapes, U retrouve z_a.

remarque 3.5 – *Quelle que soit l'étape j de la récursion, l'indice de l'élément recherché par l'utilisateur est toujours le même. Cet indice ne dépend que de i, l'indice retrouvé et de j, le numéro niveau d'exécution. Dans le cas d'un protocole interactif, il suffit donc pour chaque niveau j de la récursion de toujours utiliser la même requête qui servira à chacune des k exécutions de S_{j+1}. On utilise également le même paramètre de sécurité 1^k pour chaque protocole S_j, ainsi que la même clé publique N. Cependant, la taille de la chaîne sur laquelle on effectue la recherche devient de plus en plus petite.*

– *En effet, on remarque qu'à une étape j donnée, la première des k exécution permet de retrouver le bit de poids fort, la seconde des k exécutions permet de retrouver le second bit de poids fort et ainsi de suite. Alors à la $d_{ième}$ exécution, U retrouve le $d_{ième}$ bit de poids le plus fort de z_a. Donc, à une étape j donnée, le protocole peut être exécuté avec une base de données plus petite d'un facteur k à chacune des k exécutions.*

Complexité du protocole récursif

À la première étape, U envoie une requête à la base de données. DB n'envoie rien à U mais prépare k chaînes. À la seconde étape U envoie une nouvelle requête à DB, qui prépare à son tour k chaînes pour chacune des k chaînes qu'il possède déjà et ainsi de suite. À chaque étape j, U envoie une requête à DB qui a pour taille $t_j * k$. Si on prend pour tout j, $t_j = n^{\frac{1}{L+1}}$, U envoie au total $n^{\frac{1}{L+1}} * L * k$, puisqu'on utilise pour chaque étape la même requête. Au dernier pas de la récursion, la base de données envoie sa réponse de taille $t_j \cdot k^L$ bits, soit $n^{\frac{1}{L+1}} k^L$ bits. En prenant la valeur de L, tel que $k^L = t_j$, on obtient $L = O(\sqrt{\frac{\log_2 n}{\log_2 k}})$. La complexité de communication totale est alors $2^{O(\sqrt{\log_2 n . \log_2 k})}$. Et si $k = log_2^c\, n$, on obtient $2^{O(\sqrt{\log_2 n . \log_2(\log_2 n)})}$.

3.3.5 Description du protocole linéaire HR

Le paramètre de sécurité sera toujours défini par la taille du module N, i.e., $|N|$. La base de données est de taille n. L'utilisateur U veut retrouver le bit x_i.

1. U choisit aléatoirement deux nombres premiers p et q tel que p ne divise pas $q-1$. Il envoie un module RSA, N de taille k bits tel $N = pq$. Il garde la factorisation de N secrète.
2. U choisit aléatoirement $g \in \mathbb{Z}_{N^2}^*$ tel que $< gH >$ soit un générateur de G et tel que le logarithme discret en base g soit facile à calculer modulo N, par exemple $1 + N$. Il pose pour tout $j \in [n]$, $g_j = g^{\beta N}$, où $\beta \in \mathbb{Z}_N$, pour $j \neq i$ et $g_i = g^\alpha$ où $\alpha \in \mathbb{Z}_N^*$. Il garde α pour pour pouvoir l'utiliser ultérieurement. Il envoie les g_j à la base de données, DB, soit n bits.
3. DB choisit un élément aléatoire h dans $\mathbb{Z}_N^*$ et calcule $g_j^{x_j} h^N \in \mathbb{Z}_{N^2}^*$, pour $j = 1, \ldots, n$ et pour $x_j \in \mathbb{Z}_N$. DB envoie $z \stackrel{\text{déf}}{=} \prod_{j=1}^n g_j^{x_j} h^N$ à l'utilisateur.
4. U reçoit donc $z = \prod_{j=1}^n g_j^{x_j} h_j^N$, pour un certain $h \in \mathbb{Z}_N^*$. Il possède la factorisation de N et peut donc retrouver l'élément x_i de $\mathbb{Z}_N$.

Complexité

À la première étape, l'utilisateur envoie $n+1$ éléments : le module N de taille k, le paramètre de sécurité et n éléments du groupe $\mathbb{Z}_{N^2}^*$. Ensuite, la base de données répond en renvoyant un élément du groupe, z. Ce qui donne une complexité de $2(n+1)\log_2 N + k$ bits transmis. En prenant $k = n^c$, pour une constante c arbitrairement petite, on obtient une complexité clairement linéaire.

Pour la complexité calculatoire, l'utilisateur fait N exponentiations et n multiplications dans $\mathbb{Z}_{N^2}^*$. Quant à la base de données, elle fait $2N$ exponentiations et $2N$ multiplications, ce qui donne une complexité en $O(C \cdot n(\log_2 N)^2)$.

Correction

L'élément z que l'utilisateur reçoit est dans $\mathbb{Z}_{N^2}^*$. Considérons d'abord le cas simple où $g = 1 + N$. On peut vérifier que $1 + N$ est d'ordre N dans $\mathbb{Z}_{N^2}^*$. Notons que

$$(1+N)^x = 1 + \binom{x}{1} N + \binom{x}{2} N^2 + \cdots + \binom{x}{x} N^x.$$

Donc $g^{x_i} = 1 + N \cdot x_i \mod N^2$. Autrement dit, le logarithme discret d'un élément $\mu = (1 + N)^{x_i} = 1 + x_i \cdot N$ en base $1 + N$ modulo N^2 est x_i. On définit l'application L par :

$$L_g(\mu) = \frac{\mu - 1}{N} \text{ pour } \mu \in \mathbb{Z}^*_{N^2} \text{ et } g \text{ un élément de } \mathbb{Z}^*_{N^2} \text{ d'ordre } N.$$

Et donc si $\mu = g^{x_i} \mod N^2$, $x_i = \frac{L_g(\mu \mod N^2)}{L_g(g^{\phi(N)} \mod N^2)} \mod N$.
Pour retrouver x_i, on s'intéresse à l'équation $z = \prod_{j=1}^{n} g_j^{x_j} h^N \mod N^2$. L'utilisateur connaît la factorisation du module N et connaît donc $\phi(N)$. Il calcule d'abord $g^{\phi(N)} = 1 + k\phi(N)N$, il en déduit k puisque $k = \frac{L(g^{\phi(N)} \mod N^2)}{\phi(N)} \mod N$. Et comme $z^{\phi(N)} = 1 + k \cdot x_i\phi(N)N$, il en déduit x_i.

Preuve de sécurité

À l'étape 3, on choisit n éléments aléatoires dans $\mathbb{Z}^*_N$, plutôt que dans $\mathbb{Z}^*_{N^2}$. On va montrer que choisir un élément aléatoire dans $\mathbb{Z}^*_{N^2}$ équivaut à choisir un élément aléatoire dans $\mathbb{Z}^*_N$.

Lemme 6 *Soit $N = pq$ un module RSA tel que $\gcd(N, \phi(N)) = 1$. Soit H le sous-groupe de $\mathbb{Z}^*_{n^2}$ contenant les $N_{ième}$ résidus modulo N. Si $r \in \mathbb{Z}^*_N$ est choisi aléatoirement et uniformément, alors $r^N \mod N^2$ semble aléatoire et uniforme dans H.*

preuve 3.6 *On considère l'application injective suivante : $\psi : \mathbb{Z}^*_N \mapsto \mathbb{Z}^*_{N^2}$ tel $\psi(x) = x^N$. Alors, on a $Im(\psi) \subset H$ et par injectivité, on a l'égalité, i.e., $Im(\psi) = H$. De plus comme $|\mathbb{Z}^*_N| = |H|$ et $\gcd(N, \phi(N)) = 1$, ψ est donc une bijection. D'autre part, l'équivalence modulo N^2 implique l'équivalence modulo N.*

preuve 3.7 *Il s'agit d'une preuve similaire à la preuve 3.3.3. On suppose qu'il existe un distingueur, $\boldsymbol{B}$ capable de distinguer deux requêtes $q(i_0)$ et $q(i_1)$ pour deux indices différents de son choix. choix i_0 et i_1 B reçoit en entrée une requête $q_{i_b} = (N, g_1, \ldots, g_N)$ pour un bit b aléatoire dans $\{0, 1\}$. Il s'agit soit de $q(i_0)$ ou soit de $q(i_1)$. Et $\boldsymbol{B}$ renvoie sa réponse b'. Plus précisément $\boldsymbol{B}$ renvoie 1 avec probabilité p si la position choisie est i_0 i.e., $b = 0$ et renvoie 1 avec probabilité $p+\epsilon$, pour ϵ non négligeable si la position choisie est i_1 i.e., $b = 1$. On construit un attaquant $A(i_0, i_1)$, noté A contre l'hypothèse RC de la manière suivante : A possède en entrée une instance du problème RC, (N, y). On suppose également que n, la taille de la base données est publique.*

1. *On choisit $n - 2$ éléments de $\mathbb{Z}^*_N$ dans $\mathbb{Z}^*_{N^2}$, on les place à une position quelconque des indices i_0, i_1, indices pour lesquelles on a le distingueur B.*
2. *On place y à la position i_0 ou i_1. On notera i_c cette position. Puis on place une autre puissance $N_{ième}$ modulo N^2, choisie aléatoirement dans $\mathbb{Z}^*_{N^2}$ à la position restante.*
3. *On fait ensuite appel à l'algorithme B, en lui donnant en entrée la séquence $(N, (g_1, \ldots, g_n))$ construite. $\boldsymbol{B}$ renvoie sa réponse b'.*
4. *A renvoie la même réponse que $\boldsymbol{B}$ i.e., b' si $i_c = i_1$. Et A renvoie $1 - b'$ si $i_c = i_0$.*

Évaluons la probabilité que A renvoie 1, (y, N), en sachant que y est une puissance $N_{ième}$ modulo N^2. Si $q \stackrel{déf}{=} Prob(b' = 1 | y \in \mathbb{Z}^*_N)$, alors :

$$\begin{aligned} Prob(b'' = 1 | y \in \mathbb{Z}^*_N) &= Prob(b'' = 1 | y \in \mathbb{Z}^*_N) * [Prob(y = g_{i_0}) + Prob(y = g_{i_1})] \\ &= Prob(b' = 0 | y \in \mathbb{Z}^*_N) * Prob(y = g_{i_0}) + Prob(b' = 1 | y \in \mathbb{Z}^*_N) * Prob(y = g_{i_1}) \\ &= (1 - q) * \frac{1}{2} + q\frac{1}{2} = \frac{1}{2}. \end{aligned}$$

*Dans ce cas, quelle que soit la position choisie i_0 ou i_1, **B** possède alors en entrée n éléments qui sont tous des puissances $N_{ième}$ modulo N^2. Et alors l'avantage d'un attaquant est nul puisque la position i_c est choisie aléatoirement.*

Si y n'est pas une puissance $N_{ième}$ modulo N^2, la séquence que B possède en entrée contient exactement un élément qui n'est pas une puissance $N_{ième}$ modulo N^2. Dans ce cas, on a :

$$\begin{aligned} Prob(b''=1||y\notin\mathbb{Z}_N^*) &= Prob(b''=1|y\notin\mathbb{Z}_N^*|)*Prob(y=g_{i_0})+Prob(b''=1||y\notin\mathbb{Z}_N^*)*Prob(y=g_{i_1}) \\ &= Prob(b'=0|y\notin\mathbb{Z}_N^*|)*Prob(y=g_{i_0})+Prob(b'=1|y\notin\mathbb{Z}_N^*|)*Prob(y=g_{i_1}) \\ &= (1-p)\frac{1}{2}+(p+\epsilon)\frac{1}{2}=\frac{1}{2}+\frac{\epsilon}{2}. \end{aligned}$$

*Dans ce cas, la probabilité de succès de **B** est $\frac{1}{2}+\frac{\epsilon}{2}$ et l'avantage de l'attaquant est $\epsilon/2$. On a construit un algorithme polynomial avec en entrée y et N qui détermine la résiduosité quadratique avec probabilité au moins $\frac{1}{2}+\frac{\epsilon}{2}$, ce qui est contraire à l'hypothèse RC, d'où le résultat.*

3.3.6 Protocole de Chang, [7]

Ce protocole repose sur la même hypothèse que le protocole précédent. Il est construit à partir de la remarque suivante : on rappelle que la fonction de chiffrement du cryptosystème de Pailler est :

$$\mathcal{E}(m,r)=g^m\cdot r^N \mod N^2, \text{ avec } m\in\mathbb{Z}_N \text{ et } r\in\mathbb{Z}_N^*.$$

On a $\mathbb{Z}_{N^2}^*\neq\mathbb{Z}_N$, un message chiffré ne peut donc être chiffré une nouvelle fois.
En revanche, on remarque $\mathbb{Z}_{N^2}^*\cup\{0\}\subset\mathbb{Z}_{N^2}$, qui est est un groupe additif cyclique. Et on a :

$$c=m_0N+m_1 \text{ avec } m_0 \text{ et } m_1\in\mathbb{Z}_N \text{ et } c\in\mathbb{Z}_{N^2}.$$

Et alors m_0 et m_1 peuvent chacun être chiffré avec la même fonction de chiffrement $\mathcal{E}$. Autrement dit, si on veut chiffrer un message $m\in\mathbb{Z}_{N^2}^*$ deux fois, on chiffre d'abord m puis la paire $(m_0,m_1)\mapsto(\mathcal{E}(m_0,r_0),\mathcal{E}(m_1,r_1))$. Et on peut retrouver le message initial m en calculant :

$$m=\mathcal{D}(\mathcal{D}(\mathcal{E}(m_0,r_0))N+\mathcal{D}(\mathcal{E}(m_1,r_1))) \mod N.$$

Le protocole permet de faire diminuer la complexité du côté serveur par rapport au protocole précédent. Pour ce protocole, la complexité de communication est de $2k$ pour le serveur et $2k$ pour l'utilisateur, où $k=2\log_2 N$ qui est le paramètre de sécurité.

La base de données est représentée ici comme une matrice : on choisit m un entier tel que la base de données est de taille $n=m^2$. $x[k,l]$ est l'élément $x[(k-1)m+(l-1)+1]$ de la base de données linéaire. On suppose que U veut retrouver l'élément $x[i,j]$. On définit $I(t,t')$ une fonction telle que $I(t,t')=0$ si $t\neq t'$ et 1 sinon. Il envoie $\alpha_t=\mathcal{E}(I(t,i),r_t)$ et $\beta_t=\mathcal{E}(I(t,j),s_t)$, soit 2 fois m éléments de $\mathbb{Z}_N^*$.

Description du protocole

1. L'utilisateur choisit aléatoirement et uniformément s_t et r_t dans $\mathbb{Z}_N^*$ pour tout $t\in[m]$. Il envoie $\alpha_t=\mathcal{E}(I(t,i),r_t)$ et $\beta_t=\mathcal{E}(I(t,j),s_t)$, soit 2 fois m éléments de $\mathbb{Z}_N^*$.
2. *DB* calcule :
$$\sigma_k=\prod_{t\in[m]}\beta_t^{x(k,t)} \mod N^2 \text{ pour } k\in[m].$$

La base de données sépare chaque $\sigma_k \in \mathbb{Z}_{N^2}^*$ de la manière suivante :

$$\sigma_k = u_k \cdot N + v_k \text{ avec } u_k, v_k \in \mathbb{Z}_N.$$

Elle envoie $u = \prod_{t\in[m]} {\alpha_t}^{u_t} \mod N^2$ et $v = \prod_{t\in[m]} {\alpha_t}^{v_t} \mod N^2$ à U.

3. U retrouve l'élément qu'il cherche en calculant :

$$\mathcal{D}(\mathcal{D}(u)N + \mathcal{D}(v)) = x[i, j].$$

Complexité

U envoie, pour tout $t \in [m], \alpha_t$ et $\beta_t \in \mathbb{Z}_N^*$, soit $2k.\sqrt{n}$ bits, où k est le paramètre de sécurité ici égal à $[2\log_2 N]$. La base de données envoie 2 éléments de $\mathbb{Z}_{N^2}^*$ i.e., $2k$ bits.

Correction

On remarque que pour tout $k \in [m]$, $\sigma_k = \mathcal{E}(x_{k,j}, \tau_k)$, avec τ_k choisi aléatoirement et uniformément dans $\mathbb{Z}_N^*$. L'utilisateur retrouve l'élément $x[i, j]$ en deux étapes. Il applique une première fois la fonction de déchiffrement à u et v puisque $u = \mathcal{E}(u_i, \tau_u)$ et $v = \mathcal{E}(v_i, \tau_v)$, pour τ_u et $\tau_v \in \mathbb{Z}_N^*$. Puis U retrouve le chiffré de $x[i, j]$, en calculant $\sigma_i = u_i N + v_i$. Il retrouve enfin $x[i, j]$, en calculant $\mathcal{D}(\sigma_i)$: il calcule d'abord $v_i = \mathcal{D}(\sigma_i) \mod N$, puis $u_i = \sigma_i - \frac{v_i}{N} \mod N$.

3.3.7 Protocole de Lipmaa

Le cryptosystème construit à la section 3.2.3 va permettre de construire le protocole de l'article [15]. Ce cryptosystème a l'avantage d'être « length-flexible » i.e., qu'étant donné un entier s, on peut chiffrer et déchiffrer des messages de longueur quelconque ; ce qui nous amène à nous intéresser à un nouveau protocole où cette fois pour rechiffrer un message déjà chiffré, il ne sera pas utile de diviser le chiffré en deux parties afin de pouvoir le rechiffrer, comme pour le protocole de Chang.

Définitions

Soit s un entier fixé. Soient $\mathcal{M}_s$ l'ensemble des messages et $\mathcal{C}_s$ l'ensemble des chiffrés. Soit $\mathcal{R}$ l'espace des aléas. On demande que pour un certain entier positif a, $\mathcal{C}_{s+a} \subseteq \mathcal{M}_s$. On note ξ le plus petit entier vérifiant cette condition. Pour le cryptosystème de Dämgard-Jurik de PKC 2001 [10], $\frac{\log_2 \mathcal{C}_s}{\log_2 \mathcal{M}_s} \approx 1 + \frac{1}{s}$. Autrement dit le facteur d'expansion ξ est d'autant plus petit que la valeur de s est élevée. On suppose que le paramètre de sécurité est $k = \log_2 |\mathcal{M}_1|$. Afin de simplifier la compréhension du schéma, on prendra $|\mathcal{M}_s| = |\mathcal{M}_1|^s$, où $\log_2 |\mathcal{M}_s| = s \cdot k$ et $|\mathcal{C}_s| = |\mathcal{M}_{s+\xi}|$. Pour ce protocole, on prend $s \overset{\text{déf}}{=} \frac{l}{k}$, où l est le nombre de bits que l'utilisateur veut récupérer.

Représentation de la base de données

Au départ, la base de données x contient n entrées dans $\mathbb{Z}_N \simeq \{0, 1\}^l$, pour un certain entier l. On représente la base de données comme un hyper-cube de dimension α. Plus précisément, $n = \prod_{j=1}^{\alpha} \lambda_j$, où les λ_j sont des entiers distincts. On suppose que l'utilisateur veut retrouver

l'élément ayant pour indice $i = (q_1, \ldots, q_\alpha)$ dans x, tel que pour tout $j \in [\alpha]$, on a $q_j \in \mathbb{Z}_{\lambda_j}$. Autrement dit, l'élément $S(i_1, \ldots, i_\alpha)$ est défini par l'élément

$$S\ [\ i_1.\prod_{j=2}^{\alpha}\lambda_j + i_2.\prod_{j=3}^{\alpha}\lambda_j + \cdots + i_{\alpha-1}.\lambda_\alpha + i_\alpha + 1\ ]\ \text{de la base de données .}$$

Principe du protocole

Tout message chiffré pouvait être rechiffré avec les mêmes paramètres publics, mais un message plus long. De manière générale, on a :

$$m_n = \mathcal{E}_{s+n-1}((\ldots\mathcal{E}_{s+1}(\mathcal{E}_s(m_0; r_0); r_1); \ldots); r_{n-1}) \in \mathbb{Z}_{N^{s+n}}.$$

Et on retrouve m_0 en calculant :

$$m_0 = \mathcal{D}_s(\ldots\mathcal{D}_{s+n-2}(\mathcal{D}_{s+n-1}(m_n))\ldots).$$

L'utilisateur parcourt l'hyper-cube et chiffre au fur et à mesure sa requête suivant la valeur de i_j.
U parcourt tous les indices $(b_{j,t})_{j\in[\alpha],t\in[\lambda_j]}$ et chiffre 1 si $i_{j,t} = 1$ pour $j \in [\alpha]$ et $t \in [\lambda_j]$ et 0 si $i_{j,t} = 0$. Il fabrique de cette manière une matrice $(\beta_{j,t})_{j\in[\alpha],t\in[\lambda_j]}$. Le serveur fabrique de manière récursive à chaque étape j du parcours une base de données $S_j(i_j, \ldots, i_\alpha)$ de dimension $\alpha - j$ qui est le $j_{\text{ième}}$ chiffrement de l'élément $S(q_1, \ldots, q_{j-1}, i_j, \ldots, i_\alpha)$.
Pour la dimension j, le serveur calcule pour $i_{j+1} \in [\lambda_{j+1}], \ldots, i_\alpha \in [\lambda_\alpha]$:

$$\prod_{t\in\mathbb{Z}_{\lambda_j}} \mathcal{E}_{s+j-1}(x_{j,t}; r_{j,t})^{S_{j-1}(t,i_{j+1},\ldots,i_\alpha)} = \prod_{t\in\mathbb{Z}_{\lambda_j}} \mathcal{E}_{s+j-1}(x_{j,t} \cdot S_{j-1}(t, i_{j+1}, \ldots, i_\alpha); r'_{j,t}).$$

En effet, d'après la seconde propriété de la fonction de chiffrement de la section 3.2.3, on a :

$$\mathcal{E}_j(m_j; r_j)^{\mathcal{E}_{j-1}(m_{j-1};r_{j-1})} = \mathcal{E}_j(m_j \cdot \mathcal{E}_{j-1}(m_{j-1}; r_{j-1}); r'_j).$$

Si $x_{j,t} = 0$ alors on chiffre 0 i.e.,. qu'on obtient une puissance $N_{\text{ième}}$ sinon on rechiffre le message déjà chiffré par S_{j-1}.
À la $\alpha_{\text{ième}}$ itération, , le serveur se retrouve avec un seul élément de taille $(s + \alpha\xi)k$ bits qui est le $\alpha_{\text{ième}}$ chiffré de $S(q_1, \ldots, q_\alpha)$.

Protocole

1. L'utilisateur possède en entrée le uplet $(1^k, q, n; \mathcal{R}_Q)$. Sa requête est $(q_1, \ldots, q_\alpha)$. Il calcule pour tout $j \in [\alpha]$ et pour tout $t \in \mathbb{Z}_{\lambda_j}$:

   $\beta_{j,t} \overset{\text{déf}}{=} \mathcal{E}_{s+j-1}(b_{j,t}, r_{j,t})$ où $r_{j,t}$ est un élément aléatoire généré selon la distribution de probabilité $\mathcal{R}$.

   Il envoie $req = (pk, (\beta_{j,t})_{j\in[\alpha],t\in\mathbb{Z}_{\lambda_j}})$ au serveur.

2. Le serveur possède en entrée $(1^k, S_0 = x, req; \mathcal{R}')$. Il calcule à chaque étape $j \in [\alpha]$, pour $i_{j+1} \in [\lambda_{j+1}], \ldots, i_\alpha \in [\lambda_\alpha]$, une nouvelle base de données :

   $$S_j(i_{j+1}, \ldots, i_\alpha) \overset{\text{déf}}{=} \prod_{t\in\mathbb{Z}_{\lambda_j}} \beta_{j,t}^{S_{j-1}(t,i_{j+1},\ldots,i_\alpha)}.$$

   Le serveur envoie S_α à l'utilisateur.

3. U retrouve l'élément x_i en calculant pour j allant de α à 1 :

$$S'_{j-1} \stackrel{\text{déf}}{=} \mathcal{D}_{s+j-1}(S'_j), \text{ et en posant } S'_\alpha = S_\alpha.$$

Il retrouve alors $S'_0 = S(q_1, \ldots, q_\alpha)$.

Correction

U calcule d'abord :

$$S'_{\alpha-1} = \mathcal{D}_{s+\alpha-1}(S_\alpha) = \mathcal{D}_{s+\alpha-1}(\prod_{t\in[\lambda_\alpha]} \beta_{\alpha,t}^{S_{\alpha-1}(t)}).$$

Mais par définition des $\beta_{\alpha,t}$, on a :

$$S'_{\alpha-1} = \mathcal{D}_{s+\alpha-1}(\prod_{t\in[\lambda_\alpha]} (\mathcal{E}_{s+\alpha-1}(b_{\alpha,t})))^{S_{\alpha-1}(t)}) \tag{3.1}$$

$$= \mathcal{D}_{s+\alpha-1}(\mathcal{E}_{s+\alpha-1}(\sum_{t\in[\lambda_\alpha]} b_{\alpha,t} \cdot S_{\alpha-1}(t))) \tag{3.2}$$

$$= \mathcal{D}_{s+\alpha-1}(\mathcal{E}_{s+\alpha-1}(S_{\alpha-1}(q_\alpha))) \tag{3.3}$$

$$= S_{\alpha-1}(q_\alpha). \tag{3.4}$$

L'équation 3.2 est obtenue en utilisant les propriétés homomorphiques de la fonction $\mathcal{E}_{s+\alpha-1}$, l'équation 3.3 est une conséquence de la construction de la requête et par correction du schéma de chiffrement, on obtient l'équation 3.4.
De manière générale, on suppose qu'à l'étape j, on a : $S'_j = S_j(q_j, \ldots, q_\alpha)$, on veut montrer que $S'_{j-1} = S_{j-1}(q_{j-1}, \ldots, q_\alpha)$. Par définition des S'_j, on a $S'_{j-1} = \mathcal{D}(S'_j)$.

$$\mathcal{D}_{s+j-1}(S'_j) = \mathcal{D}_{s+j-1}(S_j(q_j, \ldots, q_\alpha)) = \mathcal{D}_{s+j-1}(\prod_{t\in[\lambda_{j-1}]} (\mathcal{E}_{s+j-1}(b_{j,t}))^{S_{j-1}(t,q_j,\ldots,q_\alpha)}) \tag{3.5}$$

$$= \mathcal{D}_{s+j-1}(\mathcal{E}_{s+j-1}(\sum_{t\in[\lambda_{j-1}]} b_{j,t} \cdot S_{j-1}(t, q_j, \ldots, q_\alpha))) \tag{3.6}$$

$$= \mathcal{D}_{s+j-1}(\mathcal{E}_{s+j-1}(S_{j-1}(q_{j-1}, \ldots, q_\alpha))) \tag{3.7}$$

$$= S'_{j-1}(q_{j-1}, \ldots, q_\alpha). \tag{3.8}$$

Le passage à l'équation 3.6 provient des propriétés homomorphiques de $\mathcal{E}_{s+j-1}$, le passage à l'équation 3.7 provient de la correction du schéma de chiffrement introduit à la section 3.2.3. Et à la fin, on calcule bien $S'_0 = S_0(q_1, \ldots, q_\alpha)$.

Complexité

Au départ, on supposera pour simplifier que tous les λ_j sont égaux à $n^{\frac{1}{\alpha}}$. On verra comment choisir les λ_j de manière à optimiser la complexité. On rappelle que k est le paramètre de sécurité où $k = \log_2 |\mathbb{Z}_{N^s}|$ et ξ est le facteur additif d'expansion de la taille des chiffrés. Du côté utilisateur, U envoie pour chaque dimension $j \in \alpha$, λ_j chiffrés chacun de taille $(s + j\xi)k$. Si on prend $\lambda_j = n^{\frac{1}{\alpha}}$, on obtient :

$$\sum_{j=1}^{\alpha} \sum_{t=1}^{\lambda_j - 1} (s + j\xi)k = \sum_{j=1}^{\alpha} (s + j\xi).(n^{\frac{1}{\alpha}} - 1).k$$

$$= \alpha.(s + (\alpha + 1)\frac{\xi}{2}).(n^{\frac{1}{\alpha}} - 1).k$$

La base de données envoie un seul élément S_α de longueur $(s + \alpha\xi)k$ bits.

Dans l'article de Lipmaa [15], il recommande les valeurs de λ_j pour lesquels on peut optimiser la communication pour des petites valeurs de s. Il redéfinit les λ_j par :

$$\lambda_j \leftarrow \left(\frac{(s+\alpha)!}{s!}\right)^{\frac{1}{\alpha}} .(s+j)^{-1}.n^{\frac{1}{\alpha}}.$$

Ce sont les valeurs qui permettent de minimiser la communication :

$$\sum_{j=1}^{\alpha}(\lambda_j - 1)(s+j) = \sum_{j=1}^{\alpha}\lambda_j(s+j) - \alpha(s + \frac{(\alpha+1)}{2}),$$

tout en respectant la condition $n = \prod_{j=1}^{\alpha}\lambda_j$.

3.4 Protocole générique

Cette partie consiste à présenter une manière générale de construire un $CPIR$ à partir d'une hypothèse calculatoire. Nous allons construire un schéma de chiffrement probabiliste possédant certaines caractéristiques en rappelant les définitions générales, puis nous allons montrer que les différents protocoles de $CPIR$ présentées jusqu'ici sont construits suivant ce modèle générique.

Dans un premier temps, nous allons décrire cette primitive générique. Et dans une deuxième partie, nous exhiberons différentes instances. Tous ces schémas ont été étudiés dans les sections précédentes.

On notera n, la taille de la base données, α le nombre de dimensions choisi pour représenter les indices des éléments dans la base de données x. k sera le paramètre de sécurité des deux participants, qui sera égale à la taille des chiffrés. On notera $\mathcal{E}_{pk}$ la fonction de chiffrement et $\mathcal{D}_{sk}$ la fonction de déchiffrement. Si ces deux clés sont clairement précisées par le contexte, on notera respectivement $\mathcal{E}$ et $\mathcal{D}$ ces deux algorithme.

3.4.1 Chiffrement

Définition 3.15 *Un chiffrement à clé publique utilise deux clés, une clé publique pk et une clé secrète sk. La clé publique, les algorithmes de chiffrement et de déchiffrement sont publiés de telle sorte que n'importe qui peut chiffrer des messages. La clé privée est gardée par la personne qui reçoit les messages, ici l'utilisateur, afin de pouvoir déchiffrer les messages reçus, ceux-ci ayant été chiffrés avec la clé publique correspondante connue de la base de données. La clé sk ne peut être facilement découverte avec la connaissance de pk et de l'algorithme de chiffrement, cette propriété caractérise un chiffrement asymétrique.*

Nous allons faire quelques rappels sur la procédure de chiffrement.

- **Algorithme de génération des clés :** Étant donné le paramètre de sécurité k en entrée, on choisit une paire de clé (pk, sk).
- **Algorithme de chiffrement :** Soit $\mathcal{R}$ un espace de probabilité fixé. $\mathcal{E}$ est définie par :

$$\mathcal{E}_{pk} : \mathcal{M} \times \mathcal{R} \mapsto \mathcal{C} \text{ tel que } \mathcal{E}_{pk}(m; r) \in \mathcal{C} \text{ est le chiffré du message } m \in \mathcal{M}.$$

Afin de simplifier la notation, on préférera noter le chiffré d'un message $\mathcal{E}(m)$.

– **Algorithme de déchiffrement :** La fonction de déchiffrement $\mathcal{D}_{sk}$ est définie par :

$$\mathcal{D}_{sk} : \mathcal{C} \mapsto \mathcal{M} \cup \{\perp\}, \text{ tel que,}$$

$$\mathcal{D}_{sk}(c) = \begin{cases} m & \text{si } c \text{ est un chiffré valide} \\ \perp & \text{sinon} \end{cases}$$

Sécurité

On demande que notre fonction de chiffrement $\mathcal{E}$ soit sémantiquement sûre i.e. :

Soit A un attaquant générant deux messages m_0 et m_1 de taille égale. On suppose que l'attaquant peut faire appel à un oracle de chiffrement qui renvoie le chiffré c_b de l'un des deux messages m_0 et m_1 choisi aléatoirement. On dit que l'algorithme de chiffrement utilisé par l'oracle de chiffrement est sémantiquement sûr si l'attaquant ne peut deviner b avec probabilité significativement plus grande que $\frac{1}{2}$. De manière plus formelle, on considère la définitoin suivante :

Définition 3.16 *Soit $\mathcal{AE}$ un schéma de chiffrement construit à partir de la primitive* 3.4.1 *et k un paramètre de sécurité. Soit $b \leftarrow \{0,1\}$ et $\mathcal{A}$ un attaquant contre le schéma $\mathcal{AE}$. On considère l'expérience suivante où l'attaquant $\mathcal{A}$ renvoie deux messages et a accès à un oracle :*

$$\begin{array}{l} \textit{Expérience } \mathbf{Exp}^{\textit{IND-CPA-b}}_{\mathcal{AE},\mathcal{A}} \\ (sk, pk) \leftarrow \mathit{KeyGen}(1^k) \\ (m_0, m_1) \leftarrow \mathcal{A}(\textsc{find}, pk) \\ \quad C \leftarrow \mathcal{E}_{pk}(m_b) \\ b' \leftarrow \mathcal{A}(\textsc{guess}, C) \\ \textit{return } b' \end{array} .$$

On définit l'avantage de l'attaquant $\mathcal{A}$ dans l'expérience $\mathbf{Exp}^{\textit{IND-CPA-b}}_{\mathcal{AE},\mathcal{A}}$ par :

$$\mathbf{Adv}^{\textit{IND-CPA}}_{\mathcal{AE},\mathcal{A}}(t) = Pr[\mathbf{Exp}^{\textit{IND-CPA-1}}_{\mathcal{AE},\mathcal{A}} = 1] - Pr[\mathbf{Exp}^{\textit{IND-CPA-0}}_{\mathcal{AE},\mathcal{A}} = 1],$$

où t est le temps maximum de l'attaquant $\mathcal{A}$ dans l'execution de l'expérience $\mathbf{Exp}^{\textit{IND-CPA-b}}_{\mathcal{AE},\mathcal{A}}$.

Propriétés homomorphiques de $\mathcal{E}$

On demande que la fonction $\mathcal{E}_{pk}$ soit additivement homomorphes i.e. :

$$\mathcal{E}_{pk}(m_1 \oplus m_2; r_1 \otimes r_2) = \mathcal{E}_{pk}(m_1; r_1) \cdot \mathcal{E}_{pk}(m_2; r_2),$$

où $\otimes$ est une loi de groupoïde dans $\mathcal{R}$ et $\oplus$ une loi de groupe dans $\mathcal{M}$.

3.4.2 Protocole générique linéaire

On rappelle que n est la taille de la base de données. L'ensemble des messages $\mathcal{M}$ de départ possède n éléments du groupe. On pose $\log_2 |M| \stackrel{\text{déf}}{=} l$. Étant donné k le paramètre de sécurité, l'algorithme de génération de clé renvoie la paire (pk, sk), où pk est la clé publique i.e., le module N et la clé secrète i.e., la factorisation de N. On suppose que l'utilisateur veut retrouver x_i. Au départ, il possède le paramètre k en entrée.

Protocole linéaire.

– L'utilisateur définit d'abord $(b_j)_{j\in[n]}$ tel que :

$$b_j = \begin{cases} 0 & \text{si } j \neq i \\ 1 & \text{si } j = i. \end{cases}$$

U génère sa requête $q = (q_1, \cdots, q_n)$, où $q_j = \mathcal{E}(b_j)$. Il envoie q à la base de données.

– Aprés avoir reçu q, la base de données utilise les propriétés homomorphiques de $\mathcal{E}$ pour chiffrer la somme des $b_j \cdot x_j$ pour tout $j \in [n]$. La base de données calcule grâce à la fonction F définie ci-dessus :

$$R \stackrel{\text{déf}}{=} = \mathcal{E}(\sum_{j\in[n]} b_j \cdot x_j) = \prod_{j\in[n]} \mathcal{E}(b_j \cdot x_j).$$

L'égalité ci-dessus est claire compte tenu des remarques précédentes.

– L'utilisateur déchiffre en calculant :

$$x_i = \mathcal{D}(R).$$

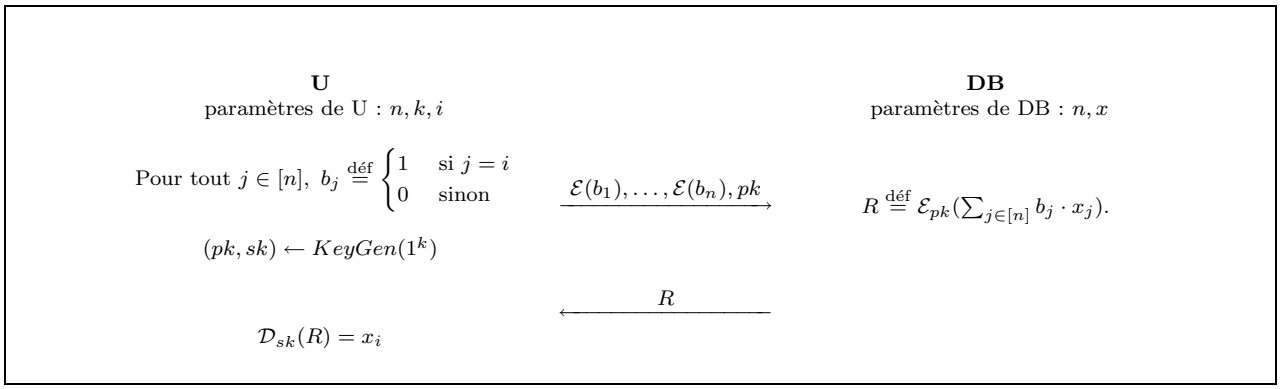

FIG. 3.1 – Protocole générique linéaire.

Correction et complexité

L'utilisateur calcule la bonne réponse par construction du schéma de chiffrement utilisé :

$$x_i = \mathcal{D}(R) = \mathcal{D}(\mathcal{E}(\sum_{j\in[n]} x_j \cdot b_j)).$$

La complexité de communication est bien entendu linéaire.

Preuve de sécurité

Supposons qu'il existe un attaquant $\mathcal{A} \stackrel{\text{déf}}{=} \mathbf{Exp}^b_{\mathcal{A},P}$ contre P, le protocole de PIR linéaire de la section 3.4.2. On va alors construire un attaquant B contre le shéma de chiffrement $\mathcal{AE}$. Par hypothèse de sécurité cet attaquant a un avantage négligeable dans l'expèrience $\mathbf{Exp}^{\text{IND-CPA-b}}_{\mathcal{AE}}$. Et donc l' attaquant $\mathcal{A}$ aura aussi un avantage négligeable contre le protocole de PIR linéaire. On définit l'attaquant B par l'algorithme suivant :

$$
\begin{array}{r|l}
pk \rightarrow & i_0, i_1 \leftarrow \mathcal{A} \\
0, 1 \leftarrow & \quad \text{pour } j \neq i_0, i_1, \\
& \qquad C_j = \mathcal{AE}_{pk}(0) \\
C = \mathcal{AE}(b) \rightarrow & \quad b' \leftarrow \{0,1\} \\
& \qquad \text{si } b' = 0 \quad C_{i_0} = C,\ C_{i_1} = \mathcal{AE}(0) \\
& \qquad \text{si } b' = 1 \quad C_{i_1} = C,\ C_{i_0} = \mathcal{AE}(0) \\
& \quad b'' \leftarrow \mathcal{A} \\
& \quad \text{Si } b'' = b' \qquad \text{renvoyer } 1 \\
& \quad \text{sinon} \qquad \text{renvoyer } 0
\end{array}
\quad .
$$

$$
\begin{aligned}
\text{Par définition, } \mathbf{Adv}_B &= Pr[\mathbf{B} = 1 | b = 1] - Pr[\mathbf{B} = 1 | b = 0] \\
&= Pr[b'' = b' | b = 1] - Pr[b'' = b' | b = 0].
\end{aligned}
$$

– Dans le cas où $b = 0$, $\mathcal{A}$ ne reçoit que des chiffrements de 0. Il a donc un avantage nul de deviner b' par rapport au cas aléatoire puisque les chiffrements sont indistingables.
– Dans le cas où $b = 1$, on a :

$$
\begin{aligned}
Pr[b'' = b' | b = 1] &= Pr[b'' = b' | b = 1 \cap b' = 0] \cdot \frac{1}{2} + Pr[b'' = b' | b = 1 \cap b' = 1] \cdot \frac{1}{2} \\
&= \frac{1}{2}(1 - Pr[b'' = 1 | b = 1 \cap b' = 0] + Pr[b'' = b' | b = 1 \cap b' = 1]) \\
&= \frac{1}{2}(1 + \mathbf{Adv}_A(t))
\end{aligned}
$$

D'après ce qui précède, on a donc :

$$
\begin{aligned}
\mathbf{Adv}_B(t) &= Pr[\mathbf{B} = 1 | b = 1] - Pr[\mathbf{B} = 1 | b = 0] \\
&= \frac{1}{2}(1 + \mathbf{Adv}_A(t)) - \frac{1}{2} \\
&= \frac{\mathbf{Adv}_A(t)}{2}
\end{aligned}
$$

Par définition de la sécurité du chiffrement $\mathcal{AE}$:

$$
\mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{AE}}(t) \stackrel{\text{déf}}{=} \max_{\mathcal{A}} \mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{AE},\mathcal{A}}(t).
$$

On a alors :

$$
\mathbf{Adv}_B(t) \leqslant \mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{AE}}(t).
$$

Et donc :

$$
\mathbf{Adv}_A(t) = 2 \cdot \mathbf{Adv}_B(t) \leqslant 2 \cdot \mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{AE}}(t).
$$

D'après la définition de sécutité 3.16, l'attaquant contre le protocole de PIR linéaire ne peut avoir un avantage significatif.

Amélioration du protocole linéaire

Dans le protocole linéaire, l'utilisateur envoie n éléments et la base de données n'envoie qu'un élément. On peut référencer la base de données de manière différente en s'inspirant des techniques utilisées par les PIR par bloc. On arrive alors à équilibrer la complexité de communication des deux participants. Si on choisit de représenter la base de données par une matrice de taille λ_1 blocs chacun de taille λ_2, l'utilisateur n'envoie plus que λ_1 éléments et la base de données envoie λ_2 éléments. Mais l'utilisateur n'est intéressé qu'à un des λ_2 éléments. On peut alors aller plus loin en utilisant un protocole de PIR avec une nouvelle base de données de taille λ_2. Dans ce cas, l'utilisateur envoie $\lambda_1 + \lambda_2$ éléments et la base de données n'envoie plus qu'un élément.

L'idée du protocole générique récursif s'inspire de ces remarques. Seulement, on utilise un multi-indexage pour représenter le contenu de la base de données. Plus précisément, on représente la base de données comme un hyper-rectangle de dimension α quelconque. Le scénario est le suivant : l'utilisateur chiffre sa requête, la base de données utilise les propriétés homomorphiques du schéma en parcourant les dimensions. Elle construit une nouvelle base de données de dimension plus petite à partir des propriétés homomorphiques de la fonction de chiffrement. Chaque entrée est vue comme un nouveau message de taille plus grande. L'étape suivante de la récursion se poursuit avec cette nouvelle base de données. Enfin, la base de données envoie à l'utilisateur les derniers chiffrés de la dernière étape. Pour retrouver l'élément qu'il cherche, l'utilisateur applique les fonctions de déchiffrement successivement aux chiffrés qu'il reçoit à l'aide des clés secrètes correspondant au schéma de chiffrement utilisé à l'étape en question.

3.4.3 Protocole récursif

Nous allons préciser les définition du chiffrement dans le cas du protocole récursif :

On définit $\alpha \geqslant 1$ un entier connu des deux participants tel que $n = \prod_{j=1}^{\alpha} \lambda_j$, avec $\lambda_j \leqslant n$ des entiers positifs pour tout $j \in [\alpha]$. Chaque élément contenu dans la base de données x sera noté $x_{j_1,\ldots,j_\alpha}$ avec $j_1 \in [\lambda_1], \ldots, j_\alpha \in [\lambda_\alpha]$. On définit une suite de α paires de clés de chiffrement et de déchiffrement, notées (pk_j, sk_j), pour $j \in \alpha$. Chaque paire de clés correspondra à une étape de la récursion. Dans certains protocoles, la première paire de clé pourra permettre de définir les autres paires de clé utilisées pour les autres étapes.

On définit une fonction de chiffrement $\mathcal{E}_j$ à une étape $j \in [\alpha]$ donnée :

$$\mathcal{E}_j : \mathcal{M}_j \times \mathcal{R}_j \mapsto \mathcal{C}_j,$$

où $\mathcal{M}_j$ est l'espace des messages et $\mathcal{C}_j$ l'espace des chiffrés. On définit $\mathcal{C}_0 = \mathcal{M}_1 \overset{\text{déf}}{=} \mathcal{M}$.

Pour $j \in \{0, \ldots, \alpha - 1\}$, on définit un entier l_j tel que :

$$\mathcal{C}_j \subseteq \mathcal{M}_{j+1}^{l_j}.$$

Et donc $l_j = \left\lceil \log_{|\mathcal{M}_{j+1}|} |\mathcal{C}_j| \right\rceil$. On suppose qu'il existe une fonction f_j injective tel que pour tout $j \in [\alpha]$:

$$f_j : \mathcal{C}_j \mapsto {\mathcal{M}_{j+1}}^{l_j}.$$

On définit alors une fonction bijective $\widetilde{f_j}$ définie par :

$$\widetilde{f_j} : \mathcal{C}_j \mapsto {\mathcal{M}_{j+1}}^{l_j} \cup \{\perp\}, \text{ tel que : } \begin{cases} \widetilde{f_j}(c_j) = f_j(c_j) & \text{si } \widetilde{f_j}(c_j) \in Im(f_j) \\ \widetilde{f_j}(c_j) = \perp & \text{sinon .} \end{cases}$$

Dans le protocole, on utilisera par commodité f_j plutôt que $\widetilde{f_j}$. On définit également la fonction $f_0(x) = x$ pour $x \in \mathcal{C}_0 = \mathcal{M}_1$. Et on a $f_0(x) = f_0^{-1}(x) = x$ pour $x \in \mathcal{M}_1$. Autrement dit, pour j fixé la fonction $\widetilde{f_j}$ détermine l'espace des messages de l'étape $j+1$. Elle permet donc de caractériser la construction de l'induction du chiffrement, comme nous le verrons par la suite.

Soit $\mathcal{D}_j$ pour $j \in \alpha$ la fonction de déchiffrement à l'étape j définie par :

$$\mathcal{D}_j : \mathcal{C}_j \mapsto \mathcal{M}_j \cup \{\perp\}.$$

Protocole

– On suppose que l'utilisateur veut récupérer l'élément indicé par $(q_1, \cdots, q_\alpha)$. En supposant que l'utilisateur possède en entrée le paramètre de sécurité k et qu'il veut récupérer l'élément indicé par $(q_1, \cdots, q_\alpha)$, il utilise l'algorithme de génération de clés appropriés pour générer $(pk_\alpha, sk_\alpha), \ldots, (pk_\alpha, sk_\alpha)$, α paires de clés correspondant chacune à une étape de la récursion.
U commence par construire α vecteurs, où le $j_{\text{ième}}$ vecteur est de dimension λ_j. On note $v_1, \ldots, v_\alpha$ ces vecteurs. Avec ces notations, U calcule :

$$v_j = (b_{j,1}, \ldots, b_{j,\lambda_j}) \text{ pour tout } j \in [\alpha],$$

où $b_{j,t}$ est défini par l'utilisateur en fonction de sa requête $(q_1, \ldots, q_\alpha)$ pour tout $t \in [\lambda_j]$. Plus précisément, U parcourt tous les indices $(j,t)_{j\in[\alpha],t\in[\lambda_j]}$ de la bases de données et définit :

$$b_{j,t} = \begin{cases} 1 & \text{si } t = q_j \\ 0 & \text{sinon .} \end{cases}$$

U calcule ensuite une séquence de requêtes :

$$\beta_{j,t} = \mathcal{E}_j(b_{j,t}), \text{ pour tout } j \in [\alpha], t \in [\lambda_j].$$

L'utilisateur envoie $(pk_1, \ldots, pk_\alpha)$ et $(\beta_{j,t})_{j\in[\alpha],t\in[\lambda_j]}$ à la base de données.

– La base de données parcourt récursivement son contenu et calcule sa réponse en utilisant les propriétés homomorphiques des fonctions $\mathcal{E}_j$:

1. Au départ, la base de données contient n éléments dans $\mathcal{M}_0$. La base de données de taille n est représentée sous forme d'une matrice de taille $\lambda_1 \times n_1$, où $n_1 = \prod_{j=2}^{\alpha} \lambda_j$. Pour tout $j_1 \in [\lambda_1], \ldots, j_\alpha \in [\lambda_\alpha]$, on définit

$$x_{j_1,\ldots,j_\alpha} = x^{(0)}_{j_1,\ldots,j_\alpha} = \overline{x}^{(0)}_{j_1,\ldots,j_\alpha}.$$

À la première étape de la récursion, DB construit une nouvelle base de données de taille n_1 en travaillant sur la première dimension. Plus précisément, la base de données calcule pour tout $j_2 \in [\lambda_2], \ldots, j_\alpha \in [\lambda_\alpha]$:

$$x^{(1)}_{j_2,\ldots,j_\alpha} = \mathcal{E}_1(\sum_{t=1}^{\lambda_1} (\overline{x}^{(0)}_{t,j_2,\ldots,j_\alpha} \cdot \beta_{1,t}).$$

À cette étape, la base de données n'envoie aucun élément. Elle obtient de manière virtuelle n_1 éléments dans $\mathcal{C}_1$.

2. À chaque entrée de la matrice, on fait correspondre l_1 messages dans $\mathcal{M}_2$. On applique la fonction f_1 à chaque élément obtenu. On définit :

$$\overline{x}^{(1)}_{j_2,\ldots,j_\alpha}[k_1] \stackrel{\text{déf}}{=} f_1(x^{(1)}_{j_2,\ldots,j_\alpha})[k_1], \text{ pour tout } k_1 \in [l_1].$$

Puis, la base de données utilise les propriétés homomorphiques de $\mathcal{E}_2$ pour construire une nouvelle base de données. Elle calcule pour tout $j_3 \in [\lambda_3], \ldots, j_\alpha \in [\lambda_\alpha]$ et pour tout $k_1 \in [l_1]$,

$$x^{(2)}_{j_3,\ldots,j_\alpha}[k_1] = \mathcal{E}_2(\sum_{t=1}^{\lambda_2} \overline{x}^{(1)}_{t,j_3,\ldots,j_\alpha}[k_1] \cdot \beta_{2,t}).$$

Comme à la première étape, la base de données n'envoie aucun élément, mais elle obtient de manière virtuelle $l_1 \cdot n_2$ éléments de $\mathcal{C}_2$, où $n_2 = \prod_{j=3}^{\alpha} \lambda_j$.

3. De manière générale à une étape $\nu \in [\alpha]$, on considère les deux ensembles $\mathcal{M}_\nu$ et $\mathcal{C}_\nu$. Au début de l'étape ν, la base de données se retrouve avec $(\prod_{j=1}^{\nu-2} l_j) \cdot n_{\nu-1}$ éléments de $\mathcal{C}_{\nu-1}$, où $n_\nu = \frac{n}{\lambda_{\nu+1}\cdots\lambda_\alpha}$. Elle applique la fonction $f_{\nu-1}$ à chaque élément et définit :

$$\overline{x}^{(\nu-1)}_{j_\nu,\ldots,j_\alpha}[k_1,\ldots,k_{\nu-1}] = f_{\nu-1}(x^{(\nu-1)}_{j_\nu,\ldots,j_\alpha}[k_1,\ldots,k_{\nu-2}])[k_{\nu-1}],$$

pour tout $k_1 \in [l_1], \ldots, k_{\nu-1} \in [l_{\nu-1}]$. Désormais, la base de données obtient $(\prod_{j=1}^{\nu-1] } l_j) \cdot n_{\nu-1}$ éléments dans $\mathcal{M}_\nu$. DB construit une nouvelle base de données de dimension $\alpha - \nu$ en utilisant les propriétés homomorphiques de le fonction $\mathcal{E}_\nu$. Plus précisément, DB calcule :

$$x^{(\nu)}_{j_{\nu+1},\ldots,j_\alpha}[k_1,\ldots,k_{\nu-1}] = \mathcal{E}_\nu(\sum_{t=1}^{\lambda_\nu} (\overline{x}^{(\nu-1)}_{t,j_{\nu+1},\ldots,j_\alpha} \cdot \beta_{\nu,t})[k_1,\ldots,k_{\nu-1}],$$

pour tout $k_1 \in [l_1], \ldots, k_{\nu-1} \in [l_{\nu-1}]$. La fonction f_ν de l'étape suivante permet d'obtenir des éléments dans $\mathcal{M}_{\nu+1}$ et de poursuivre la procédure récursive.

On choisit de représenter sur la figure 3.2, les deux étapes pour lesquelles les paramètres doivent être spécifiés, il s'agit de la première étape et de la dernière étape.

– À la fin, la base de données envoie $\prod_{j=1}^{\alpha-1} l_j$ éléments de $\mathcal{C}_\alpha$. Plus précisément pour tout $k_1 \in [l_1], \ldots, k_{\alpha-1} \in [l_{\alpha-1}]$, DB envoie :

$$x^{(\alpha)}[k_1;\ldots;k_{\alpha-1}] \in \mathcal{C}_\alpha.$$

– Aprés avoir reçu la réponse de la base de données, l'utilisateur retrouve l'élément qu'il cherche $x_{q_1,\ldots,q_\alpha}$ en utilisant les fonctions $\mathcal{D}_j$ et f^{-1}_{j-1}, pour $j \in [\alpha]$ successivement. Par définition, on a $f_0 = f_\alpha = Id$. Tout d'abord, U calcule pour tout $k_1 \in [l_1], \ldots, k_{\alpha-1} \in [l_{\alpha-1}]$:

$$\mathcal{D}_\alpha(x^{(\alpha)}[k_1;\ldots;k_{\alpha-1}]) \stackrel{\text{déf}}{=} \overline{x}^{(\alpha-1)}_{q_\alpha}[k_1;\ldots;k_{\alpha-1}].$$

Il obtient $\prod_{j=1}^{\alpha-1} l_j$ éléments dans $\mathcal{M}_\alpha$. Puis U applique $f^{-1}_{\alpha-1}$ à chaque élément obtenu. Il calcule pour tout $k_1 \in [l_1], \ldots, k_{\alpha-1} \in [l_{\alpha-1}]$:

$$f^{-1}_{\alpha-1}(\overline{x}^{(\alpha-1)}_{q_\alpha}[k_1;\ldots;k_{\alpha-1}]) \stackrel{\text{déf}}{=} x^{(\alpha-1)}_{q_\alpha}[k_1;\ldots;k_{\alpha-2}].$$

$$\left.\begin{array}{c} DB\ \ (1) \\ \begin{pmatrix} \overline{x}^{(0)}_{1,\ldots,1} & \cdots & \overline{x}^{(0)}_{1,\lambda_2,\ldots,\lambda_\alpha} \\ \vdots & \cdots & \vdots \\ \overline{x}^{(0)}_{\lambda_1,1,\ldots,1} & \cdots & \overline{x}^{(0)}_{\lambda_1,\lambda_2,\ldots,\lambda_\alpha} \end{pmatrix} \\ \text{requête}\ \ (1) \\ \begin{pmatrix} \beta_{1,1} & \cdots & \beta_{1,\lambda_1} \end{pmatrix} \end{array}\right\} \xrightarrow{\textbf{P.H}} \begin{array}{c} DB\ \ (2) \\ \begin{pmatrix} x^{(1)}_{1,\ldots,1} \\ \vdots \\ x^{(1)}_{\lambda_2,\ldots,\lambda_\alpha} \end{pmatrix} \end{array}$$

$$\xrightarrow{f_1} \left.\begin{array}{c} \text{parallélisation de } DB\ \ (2) \\ \begin{pmatrix} \overline{x}^{(1)}_{1,\ldots,1}[1] & \cdots & \overline{x}^{(1)}_{\lambda_2,\ldots,\lambda_\alpha}[1] \\ \overline{x}^{(1)}_{1,\ldots,1}[2] & \cdots & \overline{x}^{(1)}_{\lambda_2,\ldots,\lambda_\alpha}[2] \\ \vdots & \cdots & \vdots \\ \overline{x}^{(1)}_{1,\ldots,1}[l_1] & \cdots & \overline{x}^{(1)}_{\lambda_2,\ldots,\lambda_\alpha}[l_1] \end{pmatrix} \\ \text{requête}\ \ (2) \\ \begin{pmatrix} \beta_{2,1} & \cdots & \beta_{2,\lambda_1} \end{pmatrix} \end{array}\right\} \xrightarrow[\forall k_1 \in [l_1]]{\textbf{P.H}} \begin{array}{c} DB\ \ (3) \\ \begin{pmatrix} x^{(2)}_{1,\ldots,1}[k_1] & \cdots & x^{(2)}_{1,\lambda_4,\ldots,\lambda_\alpha}[k_1] \\ x^{(2)}_{2,\ldots,1}[k_1] & \cdots & x^{(2)}_{2,\lambda_4,\ldots\lambda_\alpha}[k_1] \\ \vdots & \cdots & \vdots \\ x^{(2)}_{\lambda_3,\ldots,1}[k_1] & \cdots & x^{(2)}_{\lambda_3,\ldots,\lambda_\alpha}[k_1] \end{pmatrix} \end{array}$$

$$\vdots \qquad \cdots \qquad \vdots \qquad \cdots$$

$$\xrightarrow[\forall k_1 \in [l_1],\ldots,k_{\alpha-2} \in [l_{\alpha-2}]]{f_{\alpha-1}} \left.\begin{array}{c} \text{parallélisation de } DB\ \ (\alpha-1) \\ \begin{pmatrix} \overline{x}^{(\alpha-1)}_{1,1}[k_1,\ldots,k_{\alpha-2}][1] & \cdots & \overline{x}^{(\alpha-1)}_{\lambda_{\alpha-1},\lambda_\alpha}[k_1,\ldots,k_{\alpha-2}][1] \\ \overline{x}^{(\alpha-1)}_{1,1}[k_1,\ldots,k_{\alpha-2}][2] & \cdots & \overline{x}^{(\alpha-1)}_{\lambda_{\alpha-1},\lambda_\alpha}[k_1,\ldots,k_{\alpha-2}][2] \\ \vdots & \cdots & \vdots \\ \overline{x}^{(\alpha-1)}_{1,1}[k_1,\ldots,k_{\alpha-2}][l_{\alpha-1}] & \cdots & \overline{x}^{(\alpha-1)}_{\lambda_{\alpha-1},\lambda_\alpha}[k_1,\ldots,k_{\alpha-2}][l_{\alpha-1}] \end{pmatrix} \\ \text{requête}\ \ (\alpha) \\ \begin{pmatrix} \beta_{\alpha,1},\ldots,\beta_{\alpha,\lambda_\alpha} & \cdots & \beta_{\alpha,1},\ldots,\beta_{\alpha,\lambda_\alpha} \end{pmatrix} \end{array}\right\}$$

$$\xrightarrow[\forall k_1 \in [l_1],\ldots,k_{\alpha-1} \in [l_{\alpha-1}]]{\textbf{P.H}} \begin{array}{c} DB\ \ (\alpha) \\ \begin{pmatrix} x^{(\alpha)}_1[k_1,\ldots,k_{\alpha-1}] \\ \vdots \\ x^{(\alpha)}_{\lambda_\alpha}[k_1,\ldots,k_{\alpha-1}] \end{pmatrix} \end{array}$$

FIG. 3.2 – Première étape et dernière étape du protocole générique récursif. À l'étape j, la base de données $DB\ \ (j)$ commence par appliquer la fonction f_{j-1}. Elle réorganise les éléments de la base de données afin d'exploiter les propriétés homomorphiques (**P.H**) de la fonction $\mathcal{E}_j$. À la fin de cette étape, on obtient $DB\ \ (j+1)$ qui contient $n_j = \lambda_{j+1} \cdot n_{j+1}$ éléments de $\mathcal{C}_j$. Chacun est de taille l_j dans $\mathcal{M}_{j+1}$. On parallélise $DB\ \ (j+1)$. On passe alors à l'étape $j+1$.

U obtient $(\prod_{j=1}^{\alpha-2} l_j) \cdot n_{\alpha-1}$ éléments dans $\mathcal{C}_{\alpha-1}$, où $n_{\alpha-1} = \lambda_\alpha$. De manière générale à une étape j du déchiffrement, U commence par appliquer la fonction $f_{\alpha-j}^{-1}$ aux éléments obtenus à l'étape $j-1$. Plus précisément pour tout $k_1 \in [l_1], \ldots, k_{\alpha-j} \in [l_{\alpha-j}]$, U calcule :

$$f_{\alpha-j}^{-1}(\overline{x}^{(\alpha-j)}_{q_{\alpha-j+1},\ldots,q_\alpha}[k_1;\ldots;k_{\alpha-j}]) \stackrel{\text{déf}}{=} x^{(\alpha-j)}_{q_{\alpha-j+1},\ldots,q_\alpha}[k_1;\ldots;k_{\alpha-(j+1)}].$$

U obtient $(\prod l_{\alpha-(j-1)}) \cdot n_{\alpha-j}$ éléments dans $\mathcal{C}_{\alpha-(j+1)}$, où $n_{\alpha-j} = \lambda_{\alpha-j+1} \cdot \ldots \lambda_\alpha$. Puis il applique la fonction de déchiffrement $\mathcal{D}_{\alpha-(j+1)}$ aux éléments obtenus :

$$\mathcal{D}_{\alpha-(j+1)}(x^{(\alpha-j)}_{q_{\alpha-j+1},\ldots,q_{\alpha-1}}[k_1;\ldots;k_{\alpha-(j+1)}]) = \overline{x}^{(\alpha-j-1)}_{q_{\alpha-j+1},\ldots,q_\alpha}[k_1;\ldots;k_{\alpha-(j+1)}].$$

À la fin du déchiffrement, U applique $\mathcal{D}_1$ et $f_0^{-1} = Id$. Pour plus de détails, on pourra se reporter à la preuve de correction ci-dessous.

Correction

Il s'agit de montrer le résultat suivant :

Pour toute étape $k = \alpha - j \in [\alpha]$, U calcule pour tout
$k_1 \in [l_1], \ldots, k_j \in [l_j], \overline{x}^{(j)}_{q_{j+1},\ldots,q_\alpha}[k_1;\ldots;k_j]$.

On suppose par récurrence qu'à une certaine étape $k-1$, U calcule bien pour tout $k_1 \in [l_1], \ldots, k_{j+1} \in [l_{j+1}]$, $\overline{x}^{(j+1)}_{q_{j+2},\ldots,q_\alpha}[k_1;\ldots;k_{j+1}]$. L'utilisateur commence par calculer pour tout $k_1 \in [l_1], \ldots, k_{j+1} \in [l_{j+1}]$:

$$f_{j+1}^{-1}(\overline{x}^{(j+1)}_{q_{j+2},\ldots,q_\alpha}[k_1;\ldots;k_{j+1}]) = x^{(j+1)}_{q_{j+2},\ldots,q_\alpha}[k_1;\ldots;k_j].$$

Puis il déchiffre en calculant :

$$\mathcal{D}_{j+1}(x^{(j+1)}[k_1;\ldots;k_j]) \stackrel{\text{déf}}{=} \overline{x}^{(j)}_{q_{j+1},\ldots,q_\alpha}[k_1;\ldots;k_j].$$

Il obtient $\prod_{j=1}^{j} l_j$ éléments dans $\mathcal{M}_{j+1}$. L'utilisateur réitère ce procédé jusqu'à $k = \alpha - 1$. Il se retrouve alors avec un élément de $\mathcal{M}_1 = \mathcal{C}_0$, par définition.

Complexité

U envoie d'abord sa requête $(\beta_{j,t})_{j\in[\alpha],t\in\lambda_j]}$. Il envoie donc $\sum_{j=1}^{\alpha} \lambda_j \cdot |\mathcal{C}_j|$ bits. Et la base de données envoie les $\prod_{j=1}^{\alpha-1} l_j$ chiffrés de $\mathcal{C}_\alpha$ i.e., $\prod_{j=1}^{\alpha-1} l_j \cdot |\mathcal{C}_\alpha|$ bits.

Preuve de sécurité

La preuve de sécurité du protocole générique récursif est construite sur le même modèle que la preuve de sécurité du protocole générique linéaire.
Soit $\mathcal{RE}$ un schéma de chiffrement construit à partir de la primitive de la section 3.4.3. Supposons qu'il existe un attaquant $\mathcal{A}$ contre le protocole générique récursif $\mathcal{RP}$ construit à partir du schéma $\mathcal{RE}$.
Soient $q_0 \stackrel{\text{déf}}{=} (q_{0,1}, \ldots, q_{0,\alpha})$ et $q_1 \stackrel{\text{déf}}{=} (q_{1,1}, \ldots, q_{1,\alpha})$ les deux indices pour lesquelles $\mathcal{A}$ estime pouvoir distinguer les deux requêtes q_0 et q_1 avec un avantage non négligeable. Alors il existe un indice $l \in [\alpha]$ tel que $q_{0,l} \neq q_{1,l}$. On va alors construire un attaquant contre le schéma de chiffrement $\mathcal{RE}_l$ utilisé à l'étape l de la récursion.

preuve 3.8 *On donnera juste l'idée de la preuve de sécurité, qui utilise un argument hybride standard. Tout d'abord, on définit $\alpha+1$ expériences pour $l \in \{0,\ldots,\alpha\}$, $H^l_{\mathcal{RP},\mathcal{A}}$ de la manière suivante :*

$$
\begin{array}{l}
q_0, q_1 \leftarrow \mathcal{A} \\
\textit{pour } i \in [\alpha] \\
\qquad (sk_i, pk_i) \leftarrow KeyGen_i(1^k) \\
\textit{pour } i = \{0,\ldots,\alpha\}, \\
\quad \textit{pour } j = \{0,\ldots,\lambda_i\}, \\
\qquad \textit{si } i \geqslant l, \\
\qquad\qquad \textit{si } j \neq q_{0,i}, \quad \beta i,j = \mathcal{RE}_i(0) \\
\qquad\qquad\quad \textit{sinon}, \quad \beta i,j = \mathcal{RE}_i(1) \\
\qquad \textit{si } i < l, \\
\qquad\qquad \textit{si } j \neq q_{1,i}, \quad \beta i,j = \mathcal{RE}_i(0) \\
\qquad\qquad\quad \textit{sinon}\ , \quad \beta i,j = \mathcal{RE}_i(1)
\end{array}
$$

Il est facile de voir que $H^{\alpha}_{\mathcal{RP},\mathcal{A}} = \mathbf{Exp}^1_{\mathcal{RP},\mathcal{A}}$ et que $H^0_{\mathcal{RP},\mathcal{A}} = \mathbf{Exp}^0_{\mathcal{RP},\mathcal{A}}$.
Et on a alors :

$$\mathbf{Adv}_{\mathcal{RP},\mathcal{A}} = \sum_{l=1}^{\alpha} Pr[H^l_{\mathcal{RP},\mathcal{A}}] - Pr[H^{l-1}_{\mathcal{RP},\mathcal{A}}] \qquad (1)$$

À présent, on considère l'adversaire ci-dessous contre le schéma de chiffrement $\mathcal{RE}_l$.

$$
\begin{array}{rl}
pk_t \rightarrow & q_0, q_1 \leftarrow \mathcal{A}_t \\
0,1 \leftarrow & \textit{pour } i \in [\alpha], \quad i \neq t, \\
& \qquad (sk_i, pk_i) \leftarrow KeyGen_i(1^k) \\
C \stackrel{déf}{=} \mathcal{E}_t(b) \rightarrow & \textit{pour } i = \{0,\ldots,\alpha\}, \\
& \quad \textit{pour } j = \{0,\ldots,\lambda_i\}, \\
& \qquad \textit{si } i > t, \\
& \qquad\quad \textit{si } j \neq q_{0,i}, \quad \beta i,j = \mathcal{RE}_i(0) \\
& \qquad\quad\ \ \textit{sinon}, \quad \beta i,j = \mathcal{RE}_i(1) \\
& \qquad \textit{si } i < t, \\
& \qquad\quad \textit{si } j \neq q_{1,i}, \quad \beta i,j = \mathcal{RE}_i(0) \\
& \qquad\quad\ \ \textit{sinon}\ , \quad \beta i,j = \mathcal{RE}_i(1) \\
& \quad b' \leftarrow \{0,1\} \\
& \qquad \textit{si } i = t, \\
& \qquad\quad \textit{si } b' = 0, \\
& \qquad\qquad \textit{si } j \neq q_{0,i}, \quad \beta i,j = \mathcal{RE}_i(0) \\
& \qquad\qquad\quad \textit{sinon}\ , \quad \beta i,j = C \\
& \qquad\quad \textit{si } b' = 1, \\
& \qquad\qquad \textit{si } j \neq q_{1,i}, \quad \beta i,j = \mathcal{RE}_i(0) \\
& \qquad\qquad\quad \textit{sinon}\ , \quad \beta i,j = C \\
& b'' \leftarrow \mathcal{A}_t(\overline{\beta}, \overline{pk})
\end{array}
$$

En utilisant cet adversaire, on peut montrer comme dans le cas linéaire que :

$$|Pr[H^l_{\mathcal{RP},\mathcal{A}}] - |Pr[H^{l-1}_{\mathcal{RP},\mathcal{A}}]| \leqslant 2 \cdot \mathbf{Adv}_{\mathcal{RE}_l}(t) \qquad (2)$$

En insérant l'équation (2) *dans l'équation* (1)*, on peut en conclure que :*

$$\mathbf{Adv}_{\mathcal{RP},\mathcal{A}}(t) \leqslant 2 \cdot \sum_{l=1}^{\alpha} \mathbf{Adv}_{\mathcal{RE}_l}(t).$$

3.4.4 Applications aux protocoles étudiés.

Protocole RQ

Dans le protocole de Kushilevitz et Ostrovsky [13], pour $j \in [\alpha]$, la fonction $\mathcal{E}_j$ est définie par :

$$\mathcal{E}_j : \mathcal{Z}_2 \times \mathcal{R} \mapsto \mathbb{Z}_N^*, \text{ avec } \mathcal{M}_j = \mathbb{Z}_2 \text{ et } \mathcal{C}_j = \mathbb{Z}_N^*.$$

De plus, pour tout $j \in [\alpha]$, on a $\mathcal{C}_j \subseteq {\mathbb{Z}_2}^k$. Pour tout $j \in \{0, \ldots, \alpha - 1\}$, on a donc :

$$f_j : \mathbb{Z}_N^* \mapsto {\mathbb{Z}_2}^k.$$

Dans la première version non récursive, $n = s \cdot t$ i.e., $\lambda_1 = t$ et $\lambda_2 = s$ U ne récupère pas seulement x_i, mais une colonne entière i.e., s éléments de $\mathbb{Z}_N^*$. L'idée du protocole récursive est d'appliquer le même protocole de PIR avec une base de données de taille s. On suppose que i est l'indice de l'élément recherché par l'utilisateur. On a $\alpha = L + 1$ le nombre d'étapes. Pour tout $j \in [\alpha]$, on remarque que $|\mathcal{M}_j| = |\mathcal{C}_1|^{k^{j-1}}$ et $|M_{j+1}| = |C_j|^k = 2^{k^{j+1}}$.

La complexité du côté utilisateur est $\sum_{j=1}^{\alpha} \lambda_j \cdot |\mathcal{C}_j|$ bits. Dans [13], on prend pour tout j, $\lambda_j = n^{\frac{1}{\alpha}}$ et on obtient donc une complexité égale à $\alpha \cdot n^{\frac{1}{\alpha}} \cdot k$ bits pour l'utilisateur. Pour la base de données, la complexité est $k^{\alpha-1} \cdot |C_\alpha|$. Dans [13], ils prennent $|C_\alpha| = n^{\frac{1}{\alpha}}$. On obtient donc une complexité globale égale à $n^{\frac{1}{\alpha}} \cdot (k^{\alpha-1} + k \cdot \alpha)$.

Protocole de Chang

Dans le protocole de Chang, pour tout $j \in [\alpha]$ est définie par :

$$\mathcal{E}_j : \mathcal{Z}_N \times \mathcal{R} \mapsto \mathbb{Z}_{N^2}^*,$$

avec $\mathcal{M}_j = \mathbb{Z}_N$, $\mathcal{C}_j = \mathbb{Z}_{N^2}^*$, $\mathcal{R} = \mathbb{Z}_N^*$ et $\alpha = 2$. On pose $m = [\sqrt{n}]$ et $\lambda_1 = \lambda_2 = m$. On remarque que $\mathbb{Z}_{N^2}^* \cup \{0\} \subset \mathbb{Z}_{N^2}$. En effet, pour tout chiffré $c \in \mathbb{Z}_{N^2}^*$, on a $c = m_0 \cdot N + m_1$, avec $m_0, m_1 \in \mathbb{Z}_N$. On définit donc $f_0 = f_\alpha = Id$ et la fonction f_j pour $j = 1$ par :

$$f_j : \mathbb{Z}_{N^2}^* \mapsto \mathbb{Z}_N \times \mathbb{Z}_N.$$

On a donc pour $j = 2$, $\mathcal{M}_j \subseteq \mathcal{C}_{j-1} \times \mathcal{C}_{j-1}$ i.e., $l_{j-1} = 2$. Dans ce protocole , $\alpha = 2$. Supposons de manière générale que l'on choisisse de représenter la base de données comme un hyper-cube de dimension α plus grand, avec tout les λ_j égaux à $n^{\frac{1}{\alpha}}$. On aura pour ce protocole $|\mathcal{C}_j| \geqslant |\mathcal{M}_{j+1}|^2$. Pour faciliter les calculs on admettra qu'il y a égalité. Et alors $|\mathcal{C}_j| = |\mathcal{C}_1|^{2^{j-1}}$. Si $\mathbb{Z}_N \simeq \{0,1\}^l$, on a $|\mathcal{C}_{j+1}| = (2l)^{2^j}$

Pour ce protocole, l'utilisateur envoie $2\sqrt{n} \cdot 2l$ bits et la base de données envoie les derniers chiffrés, soit $4 \cdot l$ bits. Ce qui donne une complexité globale égale à $4l \cdot (\sqrt{n} + 1)$ bits. De manière plus générale pour un α-hypercube , la complexité est $\alpha \cdot n^{\frac{1}{\alpha}} \cdot 2l$ bits envoyés et pour la base de données, elle est égale à $2^{\alpha-1} \cdot 2l$ bits.

Protocole de Lipmaa

Le protocole de Lipmaa s'adapte parfaitement à cette généralisation compte tenu de la construction. On suppose que l'entier s est fixé et est défini par le protocole. Le cas $s = 1$ correspond au cryptosystème de Pailler. Ce protocole permet d'améliorer de manière significative le facteur d'expansion.

On définit la fonction de chiffrement dont on se servira qui est définie pour tout $j \in [\alpha]$ par :

$$\mathcal{E}_j : \mathbb{Z}_{N^{j+s-1}} \times \mathbb{Z}_N^* \mapsto \mathbb{Z}_{N^{j+s}}.$$

On remarque que toutes les fonctions f_j sont égales à l'identité. On suppose que $\mathbb{Z}_N \simeq \{0,1\}^l$. À la $\alpha_{\text{ième}}$ itération, , le serveur calcule un seul élément de taille $(s + \alpha\xi)l$ bits. Lorsque le chiffrement repose sur le cryptosystème de Dämgard-Jurik, le facteur d'expansion du protocole de Lipmaa est $\xi \approx 1$ et la taille des messages à chiffrer est donc $(s + j) \cdot l$. Pour tout j, on a $\mathcal{M}_j = \mathcal{C}_{j-1}$. Et $|\mathcal{M}_j| = |\mathcal{C}_j| = |\mathcal{M}_{j-1}|$. Si $\mathbb{Z}_N \simeq \{0,1\}^l$, on a alors $|\mathcal{C}_1| = |\mathcal{M}_1| * l$ et $|\mathcal{M}_j| = |\mathcal{C}_1| * l^{j-1}$. Si la taille des messages de départ est l^s, à la fin on a $|\mathcal{M}_\alpha| = l^{s+\alpha-1}$. Et si $s = 1$, $|\mathcal{M}_j| = l^j$. On obtient comme pour le protocole RC, $|\mathcal{M}_2| = |\mathcal{M}_1|^2$ et $\mathcal{M} = \mathcal{Z}_N$ comme pour le protocole de Chang.

Pour ce protocole , l'utilisateur envoie $\sum_{j=1}^{\alpha} \lambda_j \cdot |\mathcal{C}_j|$ bits. Si $s = 1$ et que pour tout $j \in [\alpha], \lambda_j = n^{\frac{1}{\alpha}}$, l'utilisateur envoie $\sum_{j=1}^{\alpha}(n^{\frac{1}{\alpha}} - 1) \cdot (1 + jl) = \alpha \cdot (n^{\frac{1}{\alpha}} - 1) \cdot (1 + \frac{\alpha+1}{2}) \cdot l$ bits. Et la base de données envoie $(1 + \alpha) \cdot l$ bits. Ce qui donne une complexité globale égale à $\alpha \cdot (n^{\frac{1}{\alpha}} - 1) \cdot (1 + \frac{\alpha+1}{2}) \cdot l + (1 + \alpha) \cdot l$ bits.

Comparaison

Nous avons fait les calculs avec une base de données contenant $n = 2^{20}$ bits et un module de taille $N = 2^{10}$. On obtient pour les trois protocoles précédents, les résultats suivants :

résiduosité quadratique	$(2^{10})^{\frac{1}{\alpha}} \cdot ((2^{10})^{\alpha-1} + 2^{10} \cdot \alpha)$ bits	$\approx 1,2 \cdot 10^{27}$
haute résiduosité, Chang	$2 \cdot 2^{10} \cdot (\alpha \cdot 2^{10\frac{1}{\alpha}} + 2^{\alpha-1})$ bits	$\approx 1,1 \cdot 10^{6}$
Lipmaa, $s = 1$	$2^{10} \cdot (\alpha \cdot ((2^{10})^{\frac{1}{\alpha}} - 1)(1 + \frac{\alpha+1}{2}) + (1 + \alpha))$ bits	$\approx 7,7 \cdot 10^{4}$

FIG. 3.3 – Ordre de grandeur de la complexité en terme de bits pour $n = 2^{20}$ et $N = 2^{10}$. La dernière colonne donne les valeurs approximatives de la complexité lorsque $\alpha = 10$.

remarque 3.6 *Dans la figure ci-dessus, pour simplifier, nous avons choisi de donner la complexité pour $\alpha = 10$ dans les trois cas. Il est tout de même important de préciser qu'en prenant un α particulier pour chacun des protocoles, (le minimum de la fonction exprimant la complexité du protocole concerné, colonne 2) nous aurions eu une meilleure comparaison des trois cas.*

Bibliographie

[1] M. Abdalla, O. Chevassut, P.-A. Fouque, and D. Pointcheval. A simple threshold authenticated key exchange from short secrets. In B. K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 566–584, Chennai, India, Dec. 4–8, 2005. Springer-Verlag, Berlin, Germany.

[2] A. Ambainis. Upper bound on communication complexity of private information retrieval. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *ICALP*, volume 1256 of *Lecture Notes in Computer Science*, pages 401–407. Springer, 1997.

[3] A. Baimel and Y. Ishai. Information retrieval private information retrieval : A unified construction. available at www.cs.bgu.ac.il/beimel/pub.html.

[4] A. Beimel and Y. Ishai. Information-theoretic private information retrieval : A unified construction. In F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors, *ICALP*, volume 2076 of *Lecture Notes in Computer Science*, pages 912–926. Springer, 2001.

[5] A. Beimel, Y. Ishai, E. Kushilevitz, and J.-F. Raymond. Breaking the o(n1/(2k-1)) barrier for information-theoretic private information retrieval. In *FOCS*, pages 261–270. IEEE Computer Society, 2002.

[6] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414, Prague, Czech Republic, May 2–6, 1999. Springer-Verlag, Berlin, Germany.

[7] Y.-C. Chang. Single database private information retrieval with logarithmic communication. In H. Wang, J. Pieprzyk, and V. Varadharajan, editors, *ACISP*, volume 3108 of *Lecture Notes in Computer Science*, pages 50–61. Springer, 2004.

[8] B. Chor and N. Gilboa. Computationally private information retrieval (extended abstract). In *STOC*, pages 304–313, 1997.

[9] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6) :965–981, 1998.

[10] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In K. Kim, editor, *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer, 2001.

[11] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2) :270–299, 1984.

[12] Y. Ishai and E. Kushilevitz. Improved upper bounds on information-theoretic private information retrieval (extended abstract). In *STOC*, pages 79–88, 1999.

[13] E. Kushilevitz and R. Ostrovsky. Replication is not needed : Single database, computationally-private information retrieval. In *FOCS*, pages 364–373, 1997.

[14] E. Kushilevitz and R. Ostrovsky. One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. In B. Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 104–121, Bruges, Belgium, May 14–18, 2000. Springer-Verlag, Berlin, Germany.

[15] H. Lipmaa. An obvilious transfer protocol with log-squared communication. Technical report, International Association for Cryptologic Research, Feb. 2004.

[16] R. Ostrovsky and W. E. S. III. Private searching on streaming data. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 223–240. Springer, 2005.

[17] R. Ostrovsky and V. Shoup. Private information storage (extended abstract). In *STOC*, pages 294–303, 1997.

[18] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.