Université de Bordeaux
Collège Sciences et Technologies
Master 1 - Cryptologie et Sécurité Informatique

PROTOCOLE D'INTERROGATION ANONYME DE BASE DE DONNÉES

Réalisé par :
AKA Leaticia, DIALLO Mamadou Aliou, SADICK Youssouf Charaf

Encadré par :

M. Guilhem Castagnos

guilhem.castagnos@math.u-bordeaux.fr
Institut de Mathématiques de Bordeaux
Université de Bordeaux

Année Scolaire 2024/2025

Table des matières

1 Introduction

Dans un monde où la protection des données personnelles et la confidentialité des utilisateurs sont des enjeux majeurs, la recherche d'informations de manière anonyme devient une problématique essentielle. L'explosion des services en ligne, des transactions électroniques, des réseaux sociaux et des objets connectés, entraine un stockage et un échange d'une énorme quantité de données sensibles. Cette évolution a permis de grandes avancées dans des domaines comme la santé, la finance et le commerce en ligne. Mais en même temps, elle a aussi révélé à quel point les systèmes numériques peuvent être vulnérables aux fuites de données et aux atteintes à la vie privée. En effet, habituellement, pour obtenir un enregistrement contenu dans une base de donnée, le client envoie une requête au serveur en précisant l'élément souhaité, qui lui retourne l'élément demandé. Cependant, cette méthode présente un problème important : l'élément recherché ainsi que l'identité du client sont des informations potentiellement sensibles qui, dans certains cas, ne doivent pas être divulguées, même au serveur qui exécute la requête.

Dans de nombreuses situations, un utilisateur pour protéger son anonymat peut vouloir interroger une base de données sans que le serveur ne sache quel élément il recherche. Cela est particulièrement pertinent dans des contextes où la vie privée de l'utilisateur doit être strictement préservée. Par exemple, un investisseur qui interroge la base de données boursière pour connaître la valeur d'une action donnée peut souhaiter préserver la confidentialité de l'identité de l'action qui l'intéresse. De même, sur internet, un utilisateur peut souhaiter naviguer en mode privé afin d'accéder à des informations sans que les sites web ne puissent suivre son activité ou collecter ses données personnelles.

Pour répondre à ces enjeux, et avec la montée des préoccupations liées à la confidentialité des données, il est devenu essentiel de concevoir des mécanismes permettant aux utilisateurs d'interroger une base de données sans compromettre leur vie privée. C'est dans ce contexte que les protocoles de récupération privée de l'information (Private Information Retrieval - PIR) ont été développés. L'objectif est de permettre à un utilisateur d'extraire une donnée spécifique d'une base sans que le serveur ne puisse déterminer quel élément a été consulté ni révéler l'identité de l'utilisateur. Contrairement aux méthodes traditionnelles d'accès aux bases de données, les protocoles PIR assurent que la requête reste totalement confidentielle, même si le serveur participe au traitement de la requête. Ces protocoles sont donc essentiels dans un contexte où la confidentialité est primordiale, et ils offrent une solution efficace pour protéger les informations sensibles tout en permettant un accès légitime aux données.

Dans ce rapport, le but est de se focaliser sur une approche particulière du protocole PIR, en utilisant des techniques cryptographiques avancées pour préserver la confidentialité des données manipulées. Plus précisément, nous nous intéresserons à l'utilisation de systèmes de chiffrement homomorphes additifs, et en particulier au système chiffrement de Paillier. Le chiffrement homomorphe permet d'effectuer des calculs sur des données chiffrées sans nécessiter leur déchiffrement préalable, ce qui le rend particulièrement utile dans les protocoles de récupération privée d'information(PIR). Grâce à cette propriété, un utilisateur peut interroger une base de données sans exposer le contenu de sa requête. Les schémas reposant sur des groupes abéliens, comme ceux développés par Kushilevitz et Ostrovsky ou par Chang, garantissent que le serveur ne puisse tirer aucune informa-

tion exploitable des requêtes chiffrées. Ainsi, le PIR fondé sur le chiffrement homomorphe assure la confidentialité des recherches tout en optimisant la communication entre l'utilisateur et le serveur. [1] Parmi les schémas existants, le chiffrement de Paillier se distingue par sa capacité à préserver l'addition homomorphe, ce qui le rend adapté à l'implémentation de PIR à base unique. Grâce à cette approche, il est possible de construire des protocoles garantissant que le serveur ne puisse exploiter aucune information sur les requêtes, même en utilisant des analyses statistiques avancées.

Dans ce rapport, nous allons d'abord introduire les concepts généraux du protocole PIR et du chiffrement homomorphe, en expliquant les principes sous-jacents à leur fonctionnement. Nous examinerons ensuite le système de Paillier en détail, en décrivant ses caractéristiques cryptographiques et la manière dont il peut être utilisé pour implémenter un protocole PIR fonctionnel. Enfin, nous proposerons une implémentation pratique d'un protocole PIR utilisant le chiffrement de Paillier sur une base de données à une dimension, avant d'étendre cette approche à des bases de données de dimensions supérieures grâce à une approche récursive. Puis nous analyserons les avantages et les défis associés à cette approche, notamment en termes de complexité algorithmique et de performances.

2 Historique du PIR et concepts généraux

L'interrogation anonyme d'une base de données est possible grâce au protocole PIR introduit en 1995 par Chor, Goldreich, Kushilevitz et Sudan. Cette invention intervient dans un cadre où plusieurs bases de données existent mais ne peuvent pas communiquer entre elles. En effet, leur travaux ont permis de démontrer qu'un PIR efficace sur une seule base de données serait impossible, cependant le contraire a été prouvé en 1997.

D'abord connu sous l'appellation Single-Database PIR, ce protocole est devenu progressivement une primitive importante de cryptographie. Dans ce chapitre il est question de présenter le Single-database PIR ainsi que les notions cryptographiques auxquelles il est rattaché.

2.1 Historique et évolution

- 1995-1997

  Chor, Goldreich, Kushilevitz et Sudan ont proposés une méthode qui consiste à faire plusieurs copies indépendantes de la base de données afin que l'utilisateur repartisse sa requête entre plusieurs bases sans qu'aucune ne puisse reconstruire la requête complète. Cette approche rend la réalisation d'un PIR impossible en présence d'une seule base données. Deux années plus tard, Kushilevitz et Ostrovsky introduisent le PIR réalisable avec une seule base données ou Single-Database PIR en supposant l'existence d'un chiffrement à clef publique. La méthode repose sur l'utilisation du chiffrement homomorphe de Goldwasser-Micali permettant d'effectuer des calculs sur des données chiffrées sans les déchiffrer. Cette approche a une complexité sous-linéaire.

- 1999-2005

  Christian Cachin, Silvio Micali et Markus Stadler introduisent en 1999 la première solution PIR avec une communication polylogarithmique en utilisant l'hypothèse phi-hiding qui porte sur la difficulté de trouver des facteurs de $\varphi(m)$ où m est un nombre dont la factorisation est inconnu et φ est la fonction indicatrice d'Euler. Elle vise à utiliser la difficulté de trouver des facteurs cachés dans certains nombres pour masquer l'indice de la requête. En 2000, Kushilevitz et Ostrovsky proposent un PIR sans hypothèses algébriques en s'appuyant uniquement sur des permutations à trappe unidirectionnelles permettant une applicabilité à des contextes divers. Cependant, l'absence de proprietés algébribres rend les calculs plus coûteux. Dès 2004, Chang, Lipmaa, Gentry, Ramzan proposent des techniques basées sur le chiffrement de Pailler permettant d'interroger des blocs de données plutôt que des bits individuels. Cette approche améliore l'efficacité du PIR avec un rapport communication/taille des blocs qui tends vers 1 dans certains cas.

2.2 Le PIR

Un protocole d'interrogation anonyme de base de données (ou Private Information Retrieval) est une technique cryptographique permettant à un utilisateur de récupérer une donnée dans une base de données sans la divulguer au serveur dans l'objectif de garantir l'anonymat des requêtes de l'utilisateur. Le PIR est une version faible du `1-out-of-n oblivious transfer` dans lequel l'utilisateur ne doit pas obtenir d'information sur d'autres données de la base que celle qu'il demande. En effet, l'interrogation

anonyme de base de données est impérative dans un contexte de protection de la vie privée afin d'éviter le traçage des utilisateurs par des services en ligne. De plus, le PIR sert à préserver la confidentialité d'informations sensibles notamment des informations médicales, juridiques, financières etc. Une solution naïve consiste à télécharger la base de données dans son entièreté pour garantir l'anonymat. Toutefois, cette approche engendre des coûts de communication très élevés. Dans la suite, nous étudierons l'évolution des méthodes qui, au fil des années, ont permis de minimiser la quantité de données échangées tout en maintenant un haut niveau de confidentialité.

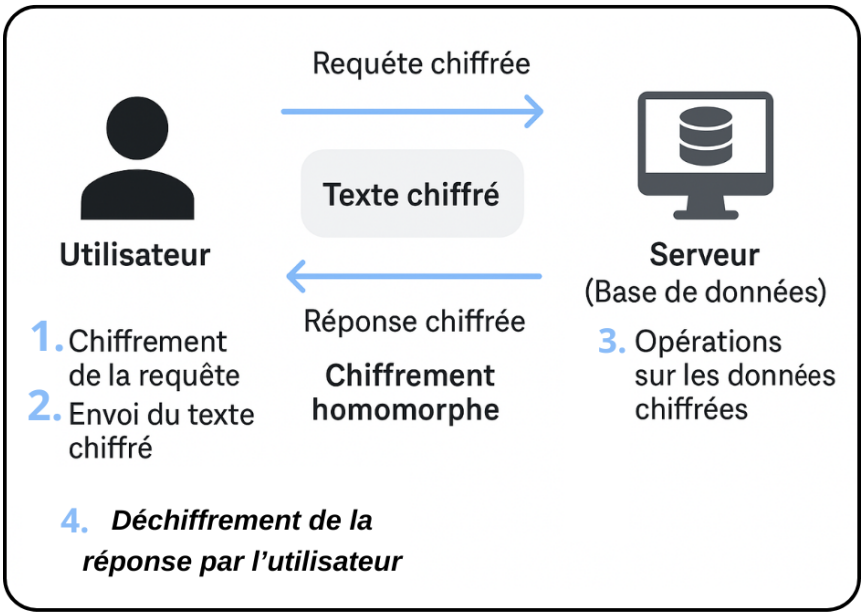

FIGURE 1 – Schéma illustrant le fonctionnement du PIR (Private Information Retrieval).

Variantes de PIR

- **Le PIR symétrique :** Le Private Information Retrieval Symétrique (SPIR) est un protocole cryptographique permettant à un utilisateur de récupérer un message parmi un ensemble de K messages stockés sur N bases de données répliquées et non collusives tout en garantissant simultanément la confidentialité de l'utilisateur et celle des bases de données. Il assure que les bases de données ne peuvent pas déterminer quel message a été consulté par l'utilisateur et que ce dernier n'obtient aucune information supplémentaire au-delà du message sélectionné, garantissant ainsi la protection des autres messages. Contrairement au PIR classique, qui ne préserve que la confidentialité de l'utilisateur, le SPIR impose une contrainte supplémentaire de confidentialité aux bases de données, ce qui réduit sa capacité et nécessite généralement une source de hasard commune entre les bases de données. Il n'est pas réalisable avec une seule base de données sans l'introduction d'une source de hasard du côté de l'utilisateur. Ce protocole est utilisé dans des applications où il est essentiel de préserver simultanément la vie privée des utilisateurs et la confidentialité des données stockées, notamment dans les systèmes de requêtes sécurisées et les bases de données confidentielles. Le terme symétrique est utilisé car le serveur et le client requièrent la confidentialité.

- **L'oblivious transfer (OT) :** L'OT est un protocole cryptographique permettant à un expéditeur (serveur) de transmettre un ou plusieurs messages à un receveur (client), sans savoir lequel a été reçu. Introduit par Michael O. Rabin en 1981, ce concept garantit que l'expéditeur demeure ignorant du choix du receveur, tout en empêchant ce dernier d'accéder aux autres messages non sélectionnés.

  Une variante importante, appelée **1-out-of-2 OT**, permet à un receveur de choisir entre deux messages sans révéler son choix à l'expéditeur. Ce principe a été généralisé en **1-out-of-n OT**, où le receveur sélectionne un élément parmi n, sans obtenir d'informations sur les autres éléments. Bien que l'oblivious transfer (OT) et le Private Information Retrieval (PIR) visent tous deux à préserver la confidentialité des requêtes d'un utilisateur, ils diffèrent dans leur finalité. En effet, l'OT impose une contrainte supplémentaire : le receveur ne peut obtenir qu'un seul élément parmi plusieurs, sans possibilité d'accéder aux autres. Ainsi, alors que le PIR protège uniquement la confidentialité de la requête, OT garantit à la fois la confidentialité du choix du receveur et l'accès restreint aux données.

Aujourd'hui, il existe des techniques visant à réduire davantage la complexité des requêtes PIR. Ce sont soit des techniques basées sur le chiffrement homomorphe de Paillier ou de Damgard-Jurik ou d'autres variantes plus avancées ou des approches basées sur des schémas de hachage exploitant des fonctions de hachage résistantes aux collisions pour optimiser l'accès aux données. Dans la suite, nous étudierons le PIR basé sur le chiffrement homomorphe de Paillier.

3 Fondements cryptographiques et conception du PIR

3.1 Chiffrement homomorphe et conception d'un protocole

Le chiffrement homomorphe est un concept fondamental de cryptographie permettant d'effectuer des calculs sur des données chiffrées sans nécessiter leur déchiffrement préalable. Contrairement aux schémas classiques de chiffrement, qui demandent de déchiffrer un message avant d'exécuter une opération dessus, le chiffrement homomorphe permet d'appliquer des opérations directement sur le texte chiffré, générant un chiffré qui, une fois déchiffré, est identique à celui qui aurait été obtenu en effectuant les mêmes opérations sur les données en clair.

Cet algorithme est particulièrement pertinent dans le cadre des protocoles PIR, où il est primordial que le serveur puisse exécuter des requêtes sur la base de données sans jamais accéder aux données en clair. L'utilisation de ce chiffrement permettra à l'utilisateur de chiffrer sa requête puis d'envoyer la requête chiffrée au serveur, qui effectuera les opérations nécessaires sans jamais voir les données initiales en clair. Une fois la réponse reçue, l'utilisateur pourra alors la déchiffrer pour obtenir l'information souhaitée, garantissant ainsi la confidentialité de la requête.

De noumbreux systèmes de chiffrement homomorphe existent, chacun présente des caractéristiques différentes. Parmi eux, on a le chiffrement totalement homomorphe(*Fully Homorphic Encryption - FHE*), qui permet d'effectuer aussi bien des additions que des multiplications sur des données chiffrées, et le chiffrement partiellement homomorphe (*Partial Homomorphic Encryption - PHE*), qui est limité à certaines opérations. Le chiffrement de Paillier, utilisé dans notre cas, appartient à cette seconde catégorie et permet d'effectuer des additions homomorphes. Cette propriété est particulièrement utile pour la conception d'un protocole PIR efficace, car elle permet de « masquer » les données de la bases de données côté serveur.[4]

3.2 Le chiffrement de Paillier

Le chiffrement de Paillier est un schéma de chiffrement homomorphe probabiliste basé sur le problème de la résidualité quadratique en degré composite. Proposé par Pascal Paillier en 1999, il repose sur l'hypothèse de l'intractabilité du *Composite Residuosity Class Problem* (CRCP) et offre des propriétés homomorphes intéressantes pour diverses applications cryptographiques, notamment le calcul sécurisé et la recherche privée d'information (PIR).

3.2.1 Propriétés homomorphes

Le chiffrement de Paillier présente une **propriété homomorphe additive** qui permet d'effectuer des calculs sur des données chiffrées. Plus précisément, si $c_1 = \mathcal{E}(m_1)$ et $c_2 = \mathcal{E}(m_2)$ sont les chiffrés de m_1 et m_2, alors le produit $c_1 \cdot c_2 \mod n^2$ est un chiffrement de $m_1 + m_2 \mod n$. Autrement dit :

$$\mathcal{E}(m_1) \cdot \mathcal{E}(m_2) \equiv \mathcal{E}(m_1 + m_2) \mod n^2,$$

où $\mathcal{E}$ est la fonction de chiffrement.

Cela permet de réaliser des calculs sur des données sans avoir besoin de les déchiffrer, ce qui est essentiel pour le *Private Information Retrieval* (PIR), où un utilisateur peut interroger une base de données sans révéler la requête au serveur.

Le chiffrement de Paillier repose sur la génération de deux grands nombres premiers p et q différents, ainsi que sur l'utilisation d'un élément $g \in \mathbb{Z}_{n^2}^*$ choisi de manière appropriée (premier avec n^2).

3.2.2 Génération des clés

La génération de clés pour le chiffrement de Pailler se fait selon le cavenas qui suit :

— Choisir deux grands nombres premiers p et q, et définir $n = pq$.

— Calculer n^2 qui sera utilisé comme module pour les opérations de chiffrement et de déchiffrement.

— Choisir une base $g \in \mathbb{Z}_{n^2}^*$. Pour la suite, admettons que $g = n + 1$. La clé publique est composée de (n, g) et la clé privée est associée à la factorisation de n c'est-à-dire les facteurs p et q, ou équivalemment $\lambda = \text{ppcm}(p-1, q-1)$.

3.2.3 Chiffrement

Le chiffrement d'un message m, se fait en choisissant un nombre aléatoire $r \in \mathbb{Z}_n^*$ nécessaire au calcul du texte chiffré c tel que :

$$c = g^m \cdot r^n \mod n^2.$$

Le chiffrement est probabiliste, c'est-à-dire que chaque exécution avec un même message m donnera un chiffré différent à cause de l'aléa introduit par r.

3.2.4 Déchiffrement

Le déchiffrement dans le cryptosystème de Paillier utilise la clé privée λ, et se fait de la façon suivante :

$$m = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n$$

où la fonction L est définie par :

$$L(x) = \frac{x-1}{n}$$

On sait que le texte chiffré est de la forme :

$$c = g^m \cdot r^n \bmod n^2$$

avec $r \in \mathbb{Z}_n^*$ un aléa choisi aléatoirement lors du chiffrement, et $m \in \mathbb{Z}_n$ le message à chiffrer. En élevant c à la puissance λ, on obtient :

$$c^\lambda = (g^m \cdot r^n)^\lambda \bmod n^2 = g^{m\lambda} \cdot r^{n\lambda} \bmod n^2$$

Comme $r \in \mathbb{Z}_n^*$ et que λ est un multiple de $\varphi(n)$, on a :

$$r^{n\lambda} \equiv 1 \mod n^2$$

Ce qui permet de simplifier :

$$c^\lambda \equiv g^{m\lambda} \mod n^2$$

On applique ensuite la fonction L à cette expression :

$$L(c^\lambda \bmod n^2) = \frac{g^{\lambda m} - 1}{n}$$

Par ailleurs, la valeur $L(g^\lambda \bmod n^2)$ est une constante qui peut être pré-calculée lors de la génération des clés :

$$L(g^\lambda \bmod n^2) = \frac{g^\lambda - 1}{n}$$

Ainsi, on peut réécrire le message déchiffré comme :

$$m = \frac{\frac{g^{\lambda m}-1}{n}}{\frac{g^\lambda - 1}{n}} \bmod n = \frac{g^{\lambda m} - 1}{g^\lambda - 1} \bmod n$$

Un choix courant pour le générateur g est $g = n + 1$. Ce choix simplifie considérablement les calculs. En effet :

$$g^{\lambda m} = (n+1)^{\lambda m} \quad \text{et} \quad g^\lambda = (n+1)^\lambda$$

En utilisant le développement binomial dans l'anneau $\mathbb{Z}_{n^2}$, on obtient :

$$(n+1)^{\lambda m} = 1 + \lambda mn \mod n^2 \quad \text{et} \quad (n+1)^\lambda = 1 + \lambda n \mod n^2$$

On en déduit :

$$\frac{(n+1)^{\lambda m} - 1}{(n+1)^\lambda - 1} = \frac{1 + \lambda mn - 1}{1 + \lambda n - 1} = \frac{\lambda mn}{\lambda n} = m$$

Ce calcul montre que la formule de déchiffrement permet bien de retrouver le message original m à partir du texte chiffré c. Ainsi, grâce à l'utilisation de la fonction L et de l'arithmétique modulaire dans $\mathbb{Z}_{n^2}$, le chiffrement et le déchiffrement dans le cryptosystème de Paillier fonctionnent correctement et garantissent la récupération fidèle du message clair.

Le chiffrement de Paillier repose sur l'hypothèse de résidualité composite. Le problème de résidualité modulo n^2 (l'identification de n-résidus) est considéré comme difficile, ce qui permet de garantir la sécurité du système. Ainsi, Le schéma est sémantiquement sûr sous l'hypothèse de la *Decisional Composite Residuosity Assumption* (DCRA).

De ce fait, le *Private Information Retrieval* (PIR) repose souvent sur le chiffrement de Paillier. Grâce à l'homomorphisme additif, il est possible d'effectuer des opérations sur des valeurs chiffrées, garantissant ainsi la confidentialité des requêtes.

3.3 PIR sur une base de donnée à une dimension : 1dPIR

Le 1dPIR est un modèle de PIR entre deux parties : un client et un serveur. Le serveur possède un vecteur x de taille N et le client a un indice $i \in [N]$ tel que le client puisse obtenir l'élément x_i et que le serveur envoie une réponse spécifique à l'élément i. Ce processus se fait en trois étapes :

On considère une base de données de taille $l = N$ avec $l \in [N]$, représentée sous la forme :

$$x = (x_1, x_2, \ldots, x_l)$$

où chaque x_i est un élément de la base.

Initialisation : Elle consiste à la génération de la requête du client. Grâce à la fonction indicatrice δ, tous les indices sont mis à zero excepté l'indice recherché.

$$\delta_{t,i_k^*} = \begin{cases} 1 & \text{si } t = i_k^* \\ 0 & \text{sinon} \end{cases}$$

Puis, ces zeros et un sont chiffrés à l'aide du chiffrement homomorphe de paillier noté $\mathcal{E}_g$:

$$\alpha_t = \mathcal{E}_g(\delta_{t,i_k^*}, r_t)$$

où chaque r_t est choisi aléatoirement dans $\mathbb{Z}_n$.

L'utilisateur envoie le vecteur $(\alpha_1, \alpha_2, \ldots, \alpha_l)$ au serveur.

Filtrage : Le serveur effectue un produit scalaire entre le vecteur chiffré et la base de données :

$$\sigma = \prod_{t=0}^{l-1} (\alpha_t)^{x_t} \mod n^2$$

et renvoie σ au client.

Reconstruction : L'utilisateur récupère x_i en déchiffrant σ à l'aide de la fonction de déchiffrement $\mathcal{D}_g$:

$$x_i = \mathcal{D}_g(\sigma)$$

3.4 PIR sur une base de données à deux dimensions : 2dPIR

Le 2dPIR est une extension du PIR à une dimension, où la base de données est organisée sous la forme d'un vecteur de taille $N = l^2$ avec $l \in [N]$. Le serveur possède ce vecteur x, et le client veut obtenir l'élément $x(i^*, j^*)$ sans révéler ses indices. Le serveur doit répondre en envoyant une réponse de taille inférieure à N bits.

On considère une base de données de taille $l \times l$, représentée sous la forme :

$$x = \begin{bmatrix} x_{1,1} & \cdots & x_{1,l} \\ \vdots & \ddots & \vdots \\ x_{l,1} & \cdots & x_{l,l} \end{bmatrix}$$

où chaque $x_{i,j}$ est un bit, et $i, j \in [l]$. L'élément $x(i^*, j^*)$ est en réalité $x = i + jl$.

Initialisation : Le client génère deux vecteurs de requêtes, un pour la ligne et un pour la colonne, en utilisant la fonction indicatrice δ, définie par :

$$\delta_{t,i_k^*} = \begin{cases} 1 & \text{si } t = i_k^* \\ 0 & \text{sinon} \end{cases}$$

Ces valeurs sont ensuite chiffrées avec le chiffrement homomorphe de Paillier :

$$\alpha_t = \mathcal{E}_g(\delta_{t,i^*}, r_t), \quad \beta_t = \mathcal{E}_g(\delta_{t,j^*}, s_t)$$

où r_t, s_t sont choisis aléatoirement dans $\mathbb{Z}_n$. Le client envoie les vecteurs chiffrés $(\alpha_1, \alpha_2, \ldots, \alpha_l)$ et $(\beta_1, \beta_2, \ldots, \beta_l)$ au serveur.

Filtrage : Le serveur calcule pour chaque ligne t :

$$\sigma_i = \prod_{t=0}^{l-1} (\beta_t)^{x_{i,t}} \mod n^2$$

Séparation et filtrage final : Le serveur sépare σ_i en deux parties où u_i et v_i sont respectivement le quotient et le reste de la division euclidienne de σ_i par n.

$$\sigma_i = u_i \cdot n + v_i$$

Ensuite, le serveur calcule u et v de la façon suivante :

$$u = \prod_{t=0}^{l-1} (\alpha_t)^{u_t} \mod n^2 \qquad v = \prod_{t=0}^{l-1} (\alpha_t)^{v_t} \mod n^2$$

Il envoie u et v au client.

Reconstruction : L'utilisateur récupère $x(i^*, j^*)$ en appliquant le déchiffrement de Paillier :

$$x(i^*, j^*) = \mathcal{D}_g(\mathcal{D}_g(u) \cdot n + \mathcal{D}_g(v)).$$

Sous l'hypothèse CRA [1], le PIR à deux dimensions(ou 2-hypercube) peut-être vu comme une implémentation en un tour du PIR à une dimension.

En effet, chaque indice i est chiffré sous la forme suivante :

$$\sigma_i = \mathcal{E}_g(x(i, j^*), r_i)$$

pour un certain $r_i \in \mathbb{Z}_n$, en exploitant la propriété d'homomorphisme additif du cryptosystème $\mathcal{E}_g$. De manière similaire, on peut exprimer u et v comme suit :

$$u = \mathcal{E}_g(u_i^*, r_u), \quad v = \mathcal{E}_g(v_i^*, r_v)$$

pour certains $r_u, r_v \in \mathbb{Z}_n$.

Ensuite, nous avons l'équation :

$$\mathcal{D}_g(u)n + \mathcal{D}_g(v) = u_i^* \cdot n + v_i^* = \sigma_i^* = \mathcal{E}_g(x(i^*, j^*), r_i^*)$$

pour $r_i^* \in \mathbb{Z}_n^*$. En appliquant l'opération de déchiffrement, on obtient finalement :

$$\mathcal{D}_g(\mathcal{D}_g(u)n + \mathcal{D}_g(v)) = x(i^*, j^*).$$

Cela confirme que le processus permet bien de récupérer la valeur correcte $x(i^*, j^*)$, assurant ainsi la validité du schéma.[3]

1. CRA (Composite Residuosity Assumption) : Si la factorisation de n est inconnue, il n'existe aucun moyen de distinguer en temps polynomial (PPT - Probabilistic Polynomial Time) les résidus n-ièmes modulo n^2 des autres valeurs. [5]

3.5 PIR sur une base de données à d-dimensions : dDPIR

Le dDPIR est un protocole de récupération d'information privée (PIR) appliqué à une base de données structurée comme un hypercube à d-dimensions, de taille totale $N = \ell^d$, où ℓ est un entier et $d \geq 1$. Le client souhaite accéder à un élément spécifique $x(i_1^*, i_2^*, \ldots, i_d^*)$, sans révéler les indices $i_1^*, \ldots, i_d^* \in [1, \ell]$.

Initialisation : Le client commence par générer d vecteurs de requêtes, un pour chaque dimension. Pour chaque dimension k, il encode son indice cible i_k^* dans un vecteur de ℓ éléments chiffrés : la position i_k^* est chiffrée comme 1 (avec le chiffrement de Paillier), les autres comme 0. Ces vecteurs chiffrés sont ensuite transmis au serveur.

Filtrage : Le serveur traite les dimensions de la base une à une, en partant de la dernière ($k = d$). Pour chaque combinaison possible des indices des $d - 1$ premières dimensions, il applique le vecteur de requêtes associé à la dimension d, ce qui lui permet de filtrer une tranche de données correspondant à i_d^* (sans connaître sa valeur réelle). Il obtient ainsi un ensemble intermédiaire de chiffrés.

Ensuite, pour chaque dimension $k = d-1$ jusqu'à 1, le serveur applique un traitement identique : il prend les résultats précédents (fixant les dimensions $k+1$ à d) et applique le vecteur de requêtes pour la dimension k. Cela réduit progressivement l'hypercube jusqu'à isoler l'élément souhaité.

Séparation et filtrage final : À chaque niveau de filtrage, le serveur effectue une décomposition des résultats intermédiaires en deux parties u et v par division euclidienne. Ces deux parties sont ensuite combinées avec le vecteur de requêtes suivant, permettant de poursuivre la descente dans l'hypercube. Ce processus aboutit à une liste finale de 2^{d-1} chiffrés, envoyés au client.

Reconstruction : Le client reçoit les 2^{d-1} chiffrés et les traite en $d - 1$ étapes. À chaque étape, il regroupe les chiffrés par paires (u, v), déchiffre chaque valeur, puis calcule :

$$D_g(u) \cdot n + D_g(v)$$

afin de reconstituer progressivement l'élément ciblé. Au terme des $d - 1$ étapes, le client obtient enfin $x(i_1^*, i_2^*, \ldots, i_d^*)$.

3.6 Complexité du PIR avec le chiffrement de Paillier

L'extraction efficace d'un ou plusieurs éléments d'une base de données sans compromettre la confidentialité est un défi central du PIR. La solution naïve consiste à exécuter k fois un protocole PIR classique pour récupérer k bits sur n, ce qui entraîne une complexité totale de *O(kn)*, rendant cette approche trop coûteuse pour de grandes bases de données. Une optimisation a été introduite par Ishi, Kushilevitz, Ostrovsky et Sahai qui consiste à utiliser une fonction de hachage pour répartir la base en k sous-bases de taille *O(n/k)*. L'utilisateur sélectionne une fonction de hachage aléatoirement et l'envoie au serveur, qui partitionne la base en k sous-bases. Il applique ensuite un protocole PIR séparément sur chacune des sous-bases, réduisant le coût global à $O(\sigma \log k.n)$ au lieu de *O(n/k)* obtenue précédemment. Cette approche améliore considérablement la complexité bien qu'elle soit applicable uniquement lorsque toutes les requêtes proviennent du même utilisateur.[1]

L'utilisation du chiffrement de Paillier dans le protocole PIR apporte une garantie considérable en termes de confidentialité, mais entraîne un coût computationnel et communicationnel significatif. Il est donc indispensable d'examiner la complexité du PIR afin de saisir les coûts liés à sa mise en œuvre et de suggérer des améliorations appropriées.

Complexité computationnelle

La complexité computationnelle du PIR est principalement influencée par trois opérations cryptographiques coûteuses :

- Le chiffrement des requêtes par l'utilisateur, qui repose sur l'exponentiation modulaire dans un anneau de taille n^2.

- L'application de multiplications et d'exponentiations modulaires sur des valeurs chiffrées effectuée par le serveur, c'est-à-dire les différents calculs homomorphes.

- Le déchiffrement de la réponse par l'utilisateur, qui nécessite également une exponentiation modulaire et l'application de la fonction de déchiffrement de Paillier.

La génération des clés dans le chiffrement de Paillier repose sur l'exponentiation modulaire qui a une complexité en $O(\log n)$. Cependant, le chiffrement d'un élément nécessite le calcul de $g^m.r^n \mod n^2$, soit deux exponentiations modulaires, ce qui donne une complexité en $O((\log n)^3)$ dans le cas d'une exponentiation naïve.

Ainsi, la charge computationnelle la plus élevée repose sur le serveur, qui doit effectuer une multiplication exponentielle pour chaque élément de la base de données. Pour une base de données à N éléments, chaque opération nécessitera une exponentiation modulaire avec une puissance dans $\mathbb{Z}_{n^2}$, soit une complexité totale en $O(N(\log n)^3)$.

Complexité communicationnelle

Un défi majeur du PIR avec le chiffrement de Paillier est son coût en communication, qui est directement lié à la taille des valeurs chiffrées échangées entre le client et le serveur. soit Eg la fonction de chiffrement de Paillier.

- **Requête envoyée par l'utilisateur**
  - L'utilisateur envoie un vecteur de sélection chiffré de taille N, où chaque élément est chiffré individuellement avec $\mathcal{E}_g$.
  - Puisque chaque valeur chiffrée occupe $O((\log n)^3)$ bits, la taille totale du message envoyé est $O(N(\log n)^3)$.
- **Réponse du serveur**
  - Le serveur retourne une valeur chiffrée unique contenant la somme pondérée homomorphe des éléments de la base de données.
  - Cette réponse est de taille $O((\log n)^3)$, car elle consiste en un seul chiffré issu de $\mathcal{E}_g$.

Ainsi, bien que le coût de communication soit linéaire en N pour l'envoi de la requête, où chaque élément est chiffré individuellement avec Paillier et la taille totale du message envoyé est $O(N(\log n)^3)$, le coût de la réponse du serveur est considérablement optimisé. En effet, au lieu d'envoyer toute la base de données, le serveur retourne une seule valeur chiffrée, ce qui permet une réduction significative de la taille des données échangées, la réponse étant de taille $O((\log n)^3)$.

Toutefois, dans le cas du PIR multidimensionnel, où l'utilisateur effectue plusieurs requêtes successives pour explorer différentes dimensions de la base de données (par exemple, d'abord sur les lignes, puis sur les colonnes, etc.), le coût de communication devient $O(dL_d^{d-1}(\log n)^3)$, avec d représentant le nombre de dimensions et L_d la taille de chaque dimension. Cela est dû au fait que chaque dimension nécessite une requête distincte, et chaque requête implique des opérations qui augmentent la charge en fonction du nombre de dimensions, rendant la communication plus coûteuse au fur et à mesure

que les dimensions augmentent. Ce coût est donc multiplicatif par rapport au nombre de requêtes et à la taille des dimensions de la base de données, mais il reste bien optimisé par rapport à une transmission naïve de toute la base de données.

4 Implémentation

Dans cette section, nous présentons l'implémentation du chiffrement de Paillier et du protocole PIR. Nous détaillons chaque composante, en commençant par la mise en place de chiffrement de Paillier qui commence par la génération des clés(publiques et privées), puis le chiffrement et le déchiffrement des messages, avant de passer à l'implémentation du protocole PIR allant d'une dimension à n-dimensions.

4.1 Chiffrement Paillier

Comme décrit ci-haut, le chiffrement de Paillier repose sur des opérations d'exponentiation modulaire dans un anneau $\mathbb{Z}_{n^2}$, où $n = pq$ avec p et q deux grands nombres premiers. La génération des clés est effectuée en calculant n et g, ainsi que les clés privées λ et μ.

La génération de clés du chiffrement Paillier est implémentée à travers une fonction `keys` :

Dans cette fonction, p et q sont générés comme des nombres premiers aléatoires de taille m et différents. Les clés publiques $public\_key$ et privées $private\_key$ sont calculées et retournées.

Comme décrit dans la section 3.2.3, le chiffrement d'un message m se fait en choisissant un nombre r aléatoire et en calculant $c = g^m \cdot r^n \mod n^2$.

```python
# Fonction de chiffrement
def encryption(m, public_key):
    n, g = public_key
    n2 = pow(n, 2)
    g = n + 1
    r = randint(2, n-1)
    while gcd(r, n) != 1:
        r = randint(2, n-1)
    c = (pow(g, m, n2)* pow(r, n, n2)) % n2
    return c
```

Le déchiffrement repose sur une exponentiation modulaire et l'application de la fonction de déchiffrement de Paillier.

```python
# Fonction de déchiffrement
def decryption(c, public_key, private_key):
    lambda_, mu = private_key
    n, g = public_key
    n2 = pow(n, 2)
    L_c = ZZ((pow(c, lambda_, n2) - 1))// n
    m = (L_c * mu) % n
    return m
```

La fonction `encryption` chiffre un message en utilisant l'exponentiation modulaire et un nombre aléatoire r. La fonction `decryption` déchiffre un message chiffré en effectuant une exponentiation modulaire et en appliquant le calcul de Paillier.

4.2 Protocole PIR

PIR 1D

Le chiffrement de Paillier ainsi opérationnel permet de mettre en place le socle du protocole c'est à dire le `1dPIR`. Il comprend principalement deux fonctions :

— `init_pir1D(data, i_star, public_key)` : une fonction qui chiffre la requête de l'utilisateur en générant un vecteur unitaire chiffré représentant l'indice $i^\star$.

— `server_op1D(data, alpha, public_key)` : une fonction où sont faites les opérations homomorphes côté serveur.

```python
# Fonction indicatrice :
def indicator_fn(t, i_star):
        return 1 if t == i_star else 0

#Chiffrement par l'utilisateur
def init_pir1D(data, i_star, public_key):
    L = len(data)
    alpha = [encryption(indicator_fn(t, i_star), public_key) for t in
    ↪  range(L)]
    return alpha

#Calculs côté serveur
def server_op1D(data, alpha, public_key):
    L = len(data)
    n, g = public_key
    n2 = pow(n, 2)
    data_bis = [[item] for item in data]

    sigma = prod(pow(alpha[t], data_bis[t][0], n2) for t in range(L))

    return [sigma]
```

Après les calculs homomorphes du serveur, la fonction retourne une liste d'un élément qu'on peut simplement déchiffrer avec notre fonction de déchiffrement implémentée précédemment en utilisant la clef privée.

```python
plaintext = decryption(sigma[0], public_key, private_key)
```

PIR nD

Maintenant que nous disposons d'une implémentation fonctionnelle en une dimension, il est naturel de vouloir l'étendre à des structures plus complexes, telles que des tableaux à deux dimensions (matrices $l \times l$), voire à trois, quatre ou n dimensions.

Cependant, implémenter un PIR spécifique pour chaque dimension peut vite devenir lourd et redondant. Une première idée consiste à reproduire le schéma 1D en utilisant des algorithmes itératifs pour chaque dimension. Mais cette méthode entraîne une complexité croissante et une duplication de code peu optimale.

Pour répondre à ce problème, nous adoptons une approche récursive plus élégante et plus efficace. Celle-ci repose sur l'utilisation du cas de base (le PIR en une dimension),

puis applique récursivement les mêmes opérations sur les $d-1$ dimensions restantes.

L'implémentation de cette approche repose principalement sur trois fonctions clés :

— `init_pirnD(data, i_star, public_key)` : une fonction qui chiffre les différentes coordonnées de la requête utilisateur selon la dimension de la base de données.

— `server_opnD(d, data, request, public_key)` : une fonction où sont effectuées les opérations homomorphes côté serveur sur la base de données selon la dimension d courante.

— `reconstruct_nD(ciphertexts, public_key, private_key)` : une fonction qui permet au client de déchiffrer la réponse chiffrée pour retrouver l'élément demandé.

```python
def server_op_rec_cD(dim, data, request, public_key):
    n, g = public_key
    n2 = n**2
    L = len(data)
    result = [1] * (2**(dim-1))

    if dim == 1:
        return server_op1D(data, request[-1], public_key)

    sigma = []
    for d in range(L):
        res = server_op_rec_cD(dim - 1, data[d], request[1:], public_key)
        for i in range(2**(dim-2)):
            u = ZZ(res[i]) // n
            v = res[i] % n
            result[2*i] = (result[2*i] * pow(request[0][d], u, n2)) % n2
            result[2*i+1] = (result[2*i+1] * pow(request[0][d], v, n2)) % n2

    return result
```

Ce bloc de code marque la fin de l'implémentation du protocole PIR appliqué à une base de données multidimensionnelle. Par ailleurs, un début d'implémentation en langage `C` est également proposé, illustrant la portabilité du protocole vers des environnements à plus bas niveau. Le code correspondant est présenté en annexe.

5 Conclusion

Dans ce rapport de TER, nous avons étudié le concept d'interrogation anonyme de bases de données, dans un contexte où la confidentialité des données personnelles s'impose comme une exigence incontournable. Après avoir présenté un état de l'art des techniques existantes, nous avons détaillé les fondements cryptographiques sous-jacents, en particulier le chiffrement de Paillier. Grâce à sa propriété homomorphe additive et à une sécurité basée sur l'hypothèse de la résiduosité composite, ce dernier constitue un outil de choix pour la conception de protocoles PIR efficaces.

Nous avons retracé l'évolution des schémas de PIR depuis les travaux fondateurs de Chor et al. (1995) et de Kushilevitz-Ostrovsky (1997), jusqu'aux approches plus modernes exploitant des structures multidimensionnelles. En particulier, les schémas proposés par Yan-Cheng Chang en 2004, basés sur une organisation récursive des données en hypercubes, offrent un bon compromis entre la complexité côté client et côté serveur. Cette évolution progressive témoigne d'un effort constant de la communauté scientifique pour améliorer à la fois l'efficacité et la sécurité de ces protocoles.

Nous avons ensuite implémenté plusieurs preuves de concept permettant d'effectuer des requêtes anonymes sur des bases de données unidimensionnelles (1dPIR), puis multidimensionnelles (ndPIR). Ces implémentations, bien que minimalistes, illustrent la faisabilité concrète de l'interrogation anonyme dans divers contextes. Elles sont conçues pour être compréhensibles, reproductibles, et constituent une base pratique pour des expérimentations ou extensions futures.

Toutefois, ce projet met aussi en lumière certaines limitations pratiques, notamment dues à la complexité computationnelle des opérations cryptographiques (principalement les exponentiations modulaires), et au coût communicationnel, en particulier pour les grandes bases de données. Ces contraintes techniques rappellent que, si les fondements théoriques du PIR sont aujourd'hui bien établis, leur application à grande échelle reste un défi, nécessitant des optimisations algorithmiques et, potentiellement, des approches hybrides.

Enfin, ce travail s'inscrit dans une perspective plus large : celle du développement de systèmes confidentiels respectueux de la vie privée. L'interrogation anonyme constitue une brique essentielle de ces futurs environnements, notamment dans des domaines sensibles comme la santé ou la finance. À terme, le croisement de techniques issues du PIR, du cloud computing, et de la cryptographie avancée pourrait permettre de concilier performance, scalabilité et protection des utilisateurs dans un monde numérique de plus en plus interconnecté.

6 Annexes

6.1 PIR à une dimension

Cette implémentattion utilise la bibliothèque *Gmp* pour sa rapidité et sa précision dans les opérations arithmétiques.

```

#include "paillier.h"
#include "pir.h"
#include <stdio.h>
#include <stdlib.h>
...


int indicator_fn(const mpz_t t, const mpz_t t0)
{
  return t == t0;
}

/* Chiffrement des requêtes de l'utilisateur */
void init_pir1D(mpz_t *alpha, size_t L, const mpz_t i_star, const mpz_t n,
↪  const mpz_t g, gmp_randstate_t state)
{
  for (size_t t = 0; t < L; ++t)
  {
    mpz_t t_mpz, msg;
    mpz_init_set_ui(t_mpz, t);
    mpz_init(msg);

    // Calcul de l'indicateur
    if (mpz_cmp(t_mpz, i_star) == 0)
      mpz_set_ui(msg, 1);
    else
      mpz_set_ui(msg, 0);

    mpz_init(alpha[t]);

    // Chiffrement
    encryption(alpha[t], msg, n, g, state);

    mpz_clear(t_mpz);
    mpz_clear(msg);
  }
}
```

```c
/* Calculs homomorphes côté serveur */
void server_op1D(mpz_t result, const mpz_t *data, const mpz_t *alpha, size_t
↪  L, const mpz_t n)

{
  mpz_t n2, tmp;
  mpz_init(n2);
  mpz_init(tmp);

  mpz_pow_ui(n2, n, 2); // n²
  mpz_set_ui(result, 1);

  for (size_t t = 0; t < L; ++t)
  {
    mpz_powm(tmp, alpha[t], data[t], n2);
    mpz_mul(result, result, tmp);
    mpz_mod(result, result, n2);
  }

  mpz_clear(n2);
  mpz_clear(tmp);
}

```

7 Bibliographie

Références

[1] Rafail Ostrovsky, William E. Skeith III, *A Survey of Single-Database PIR : Techniques and Applications*, https://eprint.iacr.org/2007/059.pdf

[2] Pascal Paillier, *Public Key Cryptosystems based on Composite Degree Residue Classes*, https://link.springer.com/content/pdf/10.1007/3-540-48910-X_16.pdf

[3] Yan-Cheng Chang, *Single Database Private Information Retrieval with Logarithmic Communication*, https://eprint.iacr.org/2004/036.pdf

[4] Craig Gentry, *Fully Homomorphic Encryption Using Ideal Lattices*, https://www.cs.cmu.edu/~odonnell/hits09/gentry-homomorphic-encryption.pdf

[5] Pascal Paillier, David Pointcheval, *Efficient public-key cryptosystems provably secure against active adversaries* https://link.springer.com/chapter/10.1007/978-3-540-48000-6_14

[6] https://gmplib.org/manual/