

I. Introduction

L'informatique distribuée considère le scénario dont un certain nombre d'actions distinctes, et pourtant connectés, des appareils informatiques (ou parties) souhaitent effectuer un calcul commun de quelques fonctions. Par exemple, ces dispositifs peuvent être des serveurs qui détiennent un système distribué de bases de données, et la fonction à calculer peut être une mise à jour quelconque de la base de données. L'objectif du calcul multi - partie sécurisé est de permettre aux parties d'effectuer des tâches telles que le calcul distribué d'une manière sécurisée. Alors que l'informatique distribuée classique aborde les questions de l'informatique sous la menace de plantage d'une machine ou d'autres fautes d'inadvertance, le calcul multi - partie sécurisé est concerné par la possibilité d'un comportement délibérément malveillant d'une certaine entité adverse. Autrement dit, il est supposé qu'un protocole d'exécution relève de l'attaque d'une entité externe, ou même d'un sous-ensemble de parties participantes. Le but de cette attaque peut être l'apprentissage d'informations privées ou la provocation d'un résultat incorrect du calcul.

Le modèle du calcul multi - partie sécurisé englobe des tâches aussi simples comme le tirage pile ou face, et aussi complexe comme le vote électronique, les ventes électroniques aux enchères, les systèmes de monnaie électronique, la signature de contrats, les transactions anonymes, et les régimes privés de récupération de l'information. Considérons par exemple les tâches de vote et de ventes aux enchères. La nécessité de privacy pour un protocole d'élection garantit que les parties n'apprennent rien sur les votes individuels des autres parties, et l'exigence de l'exactitude assure qu'aucune coalition entre ces parties ne peut influencer sur le résultat de l'élection au-delà du vote seulement pour leur candidat préféré. De même, dans un protocole d'enchères, l'exigence de la privacy garantit que seul l'enchère gagnante est révélé (ceci peut être désiré), et l'exigence de l'exactitude assure que le plus haut soumissionnaire est en effet la partie gagnante (et donc le commissaire-priseur, ou toute autre partie, ne peut pas biaiser les résultats).

Actuellement, le calcul multi-partie sécurisé trouve ses applications dans le domaine de préservation de privacy en datamining lorsque les données sont distribuées sur plusieurs parties. Le calcul multi-partie sécurisé fournit un modèle formel pour garantir la protection de ces données au moment d'exécution des algorithmes du datamining sur des ensembles de données partitionnées. Dans ce chapitre nous décrivons les principes et fondements du calcul multi-partie sécurisé et nous recueillons les outils de sécurité utilisés pour la préservation de privacy dans l'algorithme de clustering k-means dans.

II. La sécurité dans le calcul multi-partie [64] :

Comme il est mentionné ci-dessus, le modèle que nous considérons est lorsqu'une entité adverse contrôle un certain sous-ensemble de parties et souhaite attaquer l'exécution du protocole. Les parties sous le contrôle de l'adversaire sont appelées corrompues, et suivent les instructions de l'adversaire. Les protocoles sécurisés doivent résister à toute attaque de l'adversaire (où la puissance exacte de l'adversaire sera discutée plus tard). Afin de prouver la sécurité d'un protocole, une définition précise de la sécurité pour le calcul multi-partie est requise. Un certain nombre de définitions différentes ont été proposées qui visent à assurer un nombre de propriétés de sécurité importantes qui sont suffisamment générales pour capturer la plupart (si

pas toutes) les tâches du calcul multi-partie sécurisé. Nous allons maintenant décrire le plus important de ces propriétés.

II. 1 Les propriétés de sécurité dans le calcul multi-partie sécurisé :

a. Privacy: Aucune partie ne doit rien apprendre de plus que sa sortie prescrite. En particulier, la seule information qui devrait être apprise sur les entrées des autres parties est ce qui peut être dérivé de la sortie elle-même. Par exemple, dans une vente aux enchères où la seule offre révélée est celle du plus grand offrant, il est évidemment possible d'en déduire que toutes les autres offres étaient inférieures à l'offre gagnante. Toutefois, cela devrait être la seule information révélée sur les offres de perdantes.

b. exactitude (Correctness): Chacune des parties est garantie que la sortie qu'elle reçoit est correcte. Pour continuer avec l'exemple d'une vente aux enchères, ceci implique que la partie qui a l'offre la plus élevée est garantie de gagner, et aucune partie, y compris le commissaire-priseur ne peut influencer sur le résultat.

c. Indépendance des entrées: les parties corrompues doivent choisir leurs entrées de façon indépendante des parties honnêtes. Cette propriété est cruciale dans une vente aux enchères scellées, où les offres sont tenues secrètes et les parties doivent fixer leurs offres indépendamment des autres. Nous notons que l'indépendance des entrées n'est pas impliquée par la privacy. Par exemple, il est possible de produire une offre plus élevée, sans connaître la valeur de l'offre originale. Une telle attaque peut effectivement être réalisée sur les systèmes de chiffrement (par exemple, étant donné un cryptage de 100 DA, il est possible de générer un chiffrement valide de 101 DA, sans connaître la valeur chiffrée original).

d. livraison garantie du résultat: les parties corrompues ne doivent pas être en mesure d'empêcher les parties honnêtes de recevoir leur sortie. En d'autres termes, l'adversaire ne doit pas être en mesure d'interrompre le calcul en effectuant une attaque de «dénier de service» par exemple.

e. Équité: les parties corrompues devraient recevoir leurs sorties si et seulement si les parties honnêtes reçoivent également leurs sorties. Le scénario où une partie corrompue obtient le résultat et une partie honnête ne le reçoit pas, ne devrait pas être autorisé de se produire. Cette propriété peut être cruciale, par exemple, dans le cas d'un contrat de signature. Plus précisément, il serait très problématique si la partie corrompue reçoit le contrat signé et la partie honnête ne le reçoit pas.

Il est souligné que la liste ci-dessus ne constitue pas une définition de la sécurité, mais plutôt un ensemble de prescriptions, qui devraient tenir pour tout autre protocole sécurisé. En effet, une approche possible pour définir la sécurité est de générer simplement une liste d'exigences séparées (comme ci-dessus) et ensuite dire que le protocole est sécurisé si toutes ces conditions sont remplies. Toutefois, cette approche n'est pas satisfaisante pour les raisons suivantes. Premièrement, il peut être possible qu'une condition importante a été manquée. Cela est particulièrement vrai parce que différentes applications peuvent avoir différents besoins. On tient à une définition qui est assez générale pour la capture de toutes les applications. Deuxièmement, la

définition devrait être suffisamment simple pour qu'il soit trivial de voir que toutes les attaques adversaires possibles sont empêchées par les définitions proposées. Un nouveau paradigme de sécurité est donc apparu pour définir un cadre simple et rigoureux pour le calcul multi-partie sécurisé.

II.2 Le paradigme de la simulation idéal/réel :

La définition standard aujourd'hui ([65] à la suite [66][67][68]) formalise donc la sécurité dans la manière générale suivante. Comme une expérience mentale, envisager un «monde idéal» dans lequel une partie externe de confiance (et incorruptible) est disposé à aider les parties à mener à bien leurs calculs. Dans un tel monde, les parties peuvent simplement envoyer leurs entrées à la tierce partie de confiance, qui calcule alors la fonction désirée et transmet à chaque partie sa sortie prescrite. Étant donné que la seule action menée par une partie est celle de l'envoi de son entrée à la tierce partie de confiance, la seule liberté accordée à l'adversaire est dans le choix des entrées des parties corrompues.

Bien sûr, dans le «monde réel», il n'y a aucune partie externe qui peut être la confiance de toutes les parties. Plutôt, les parties exécutent un certain protocole entre elles sans aucune aide. Malgré cela, un protocole sécurisé devrait émuler le soi-disant "le monde idéal". Autrement dit, un protocole réel qui est exécuté par les parties (en un monde où aucun tiers de confiance existe) est, dit-on sécurisé, si aucun adversaire ne peut faire plus de mal à une exécution réelle que dans une exécution qui a lieu dans le monde idéal. Cela peut être formulé en disant que pour n'importe quel adversaire réalisant une attaque réussie dans le monde réel, il existe un adversaire qui réalise avec succès la même attaque dans le monde idéal. Cependant, il n'existe pas d'attaques réussies qui peuvent être réalisées dans le monde idéal. Nous concluons donc que toutes les attaques d'adversaires sur les exécutions dans le protocole du monde réel doivent également échouer.

Plus formellement, la sécurité d'un protocole est établie en comparant le résultat de l'exécution d'un protocole réel avec le résultat d'un calcul idéal. Autrement dit, pour tout adversaire attaquant une exécution d'un protocole réel, il existe un adversaire attaquant une exécution idéale (avec une tierce partie de confiance), tels que les entrée / sortie des distributions de l'adversaire et les parties participantes à l'exécution réelle et idéale sont essentiellement les mêmes. Cette formulation de la sécurité se nomme *le paradigme de la simulation idéal/réel*.

On remarque que la définition informelle ci-dessus est en fait « trop idéale » et doit être assouplie dans les conditions où l'adversaire dispose de la moitié ou plusieurs des parties participantes (dans le cas où il n'y a pas de majorité honnête). Lorsque ce nombre de parties est corrompues, il est connu qu'il est impossible d'obtenir des protocoles généraux pour le calcul multi-partie sécurisée qui garantissent « livraison garantie des sorties » et « équité » (par exemple [69][70]).

Par conséquent, la définition détendue est que l'adversaire est permis d'abandonner le calcul (i.e, amené à arrêter avant la fin), ce qui signifie que la « livraison garantie des résultats » n'est pas respecté. En outre, l'adversaire peut provoquer cet abandon pour prendre place après qu'il a déjà obtenu sa sortie, mais avant tout les parties honnêtes impliquées reçoivent leurs sorties. Ainsi, «l'équité» n'est pas atteinte. Plus vaguement, la définition assouplie est obtenue en modifiant l'exécution idéale et en donnant l'adversaire la possibilité supplémentaire de donner des instructions à la tierce partie de confiance de ne pas envoyer les résultats de certaines des parties honnêtes. Sinon,

la définition reste identique et donc toutes les autres propriétés sont encore préservées.

II.3 La puissance de l'adversaire :

La définition informelle de la sécurité ci-dessus omet une très importante question qui est la puissance de l'adversaire (celui qui attaque l'exécution d'un protocole). Comme il a été mentionné, l'adversaire contrôle un sous-ensemble des parties participantes dans le protocole. Toutefois, la stratégie de corruption n'a pas été décrite (par exemple, quand, ni comment les parties tombent sous le contrôle de l'adversaire), le comportement autorisé de l'adversaire (à savoir, l'adversaire juste recueillis passivement des informations ou peut-il charger les parties corrompues d'agir malicieusement), et comment la complexité de l'adversaire est supposé être (par exemple, Est-il à temps de calcul polynomial ou illimitée). Nous allons maintenant décrire les Principaux types d'adversaires qui ont été envisagées:

1. Stratégie de corruption: cette stratégie vise la question de savoir quand et comment les parties sont corrompues. Il existe deux modèles principaux:

a) le modèle de corruption statique: Dans ce modèle, l'adversaire se voit attribuer un ensemble fixe de parties qu'il contrôle. Les parties honnêtes restent honnêtes et les parties corrompues restent corrompues.

b) le modèle de corruption adaptative: Plutôt que d'avoir un ensemble fixe de parties corrompues, les adversaires d'adaptatifs sont donné la capacité de corrompre les parties pendant le calcul. Le choix de qui corrompre, et quand, peut être arbitrairement décidé par l'adversaire et peut dépendre de sa vision de l'exécution (pour cette raison il est appelé adaptatif). Cette stratégie modélise la menace d'un hacker (intrus) externe pénétrant par effraction dans une machine pendant une exécution. Nous notons que dans ce modèle, une fois une partie est corrompue, elle reste corrompue à partir de ce point.

Un autre modèle, *appelé modèle proactif* [71][72], considère la possibilité que les parties soient corrompues pendant une certaine période de temps seulement. Ainsi, les parties honnêtes peuvent devenir corrompues au long du calcul (comme dans le modèle d'adversaire adaptatif), mais les parties corrompues peuvent devenir aussi honnêtes.

2. Le comportement autorisé à l'adversaire: Un autre paramètre qui doit être défini concernant les initiatives que les parties corrompues sont autorisées à prendre. Une fois de plus, il existe deux principaux types d'adversaires:

a) Adversaires semi-honnêtes: Dans le modèle d'adversaire semi-honnête, même les parties corrompues suivent correctement les spécifications du protocole. Toutefois, l'adversaire obtient l'état interne de toutes les parties corrompues (y compris la transcription de tous les messages reçus), et tente de les utiliser pour apprendre des informations qui devraient rester privés. Il s'agit du modèle d'adversaire le plus faible. Toutefois, il existe certaines situations où il peut réellement modéliser des menaces au système. Les adversaires semi-honnêtes sont aussi appelés «honnêtes-mais-curieux» et «passifs». (L'adversaire semi – honnête sera traité plus en détail dans la section)

b) Adversaires malicieux: Dans ce modèle d'adversaire, les parties corrompues peuvent arbitrairement s'écarter de la spécification du protocole, selon les instructions de l'adversaire. En général, assurer la sécurité dans la présence d'adversaires malveillants est préférable, car ça veille à ce qu'aucune attaque d'adversaire ne peut réussir. Les adversaires malicieux sont également appelés "Active".

3. Complexité: Enfin, on considère la complexité supposée du calcul théorique de l'adversaire. Comme précédemment, il existe deux catégories ici:

a) A temps polynomial: L'adversaire est autorisé à s'exécuter en un temps polynomial (probabiliste) (et parfois, devrait être à temps polynomial). Le modèle de calcul spécifique utilisé diffère, selon que l'adversaire est uniforme (dans ce cas, c'est une Machine de Turing à temps polynomial) ou non-uniforme (dans ce cas, il est modélisé par une famille de circuits de taille polynômiale) (plus de détail dans la section suivante) .

b) A temps illimité: Dans ce modèle, l'adversaire n'a pas de limites de calcul quel qu'il soit.

La distinction ci-dessus concernant la complexité des adversaires donne lieu à deux modèles très différents pour le calcul sécurisé: le modèle « information-theoretic » [73][74] et le modèle calculatoire [6][7]. Dans les paramètres du modèle « information-theoretic », l'adversaire n'est pas lié à aucune classe de complexité (et en particulier, il n'est pas supposé de s'exécuter en un temps polynomial). Par conséquent, les résultats dans ce modèle tiennent inconditionnellement et ne compte sur aucune hypothèse de complexité ou de cryptographie. La seule hypothèse utilisée est que les parties sont reliées par des canaux idéalement privé (par exemple, on suppose que l'adversaire ne peut intercepter ou d'interférer la communication entre les parties honnêtes). En revanche, dans les paramètres du modèle calculatoire, l'adversaire est supposé à être en temps polynomial. Les résultats dans ce modèle supposent habituellement des hypothèses cryptographiques. Nous notons qu'il n'est pas nécessaire ici de supposer que les parties ont accès à des canaux idéaux privés, car ces canaux peuvent être implémentés en utilisant un chiffrement à clé publique. Toutefois, il est supposé que la communication entre les canaux des parties est authentifiée. Cette authentification peut être obtenue en utilisant les signatures numériques [75] et une infrastructure à clés publiques. On remarque que toutes les combinaisons possibles de ces types d'adversaires ont été considérées dans la littérature.

III. La faisabilité du calcul multi-partie sécurisé [64] :

La définition de la sécurité décrite ci-dessus semble être très restrictive dans le sens où aucun succès de l'adversaire n'est toléré. Ainsi, on peut se demander s'il est même possible d'obtenir des protocoles sécurisés dans le cadre de cette définition, et si oui, pour quelles tâches de l'informatique distribuée. Peut-être surprenant, des résultats de faisabilité puissants ont été mis en place, démontrant que, en fait, toute tâche de calcul distribué pouvant être calculée d'une façon sécurisé. Nous allons

maintenant exposer brièvement les plus centraux de ces résultats, Soit n le nombre de parties participantes et soit t la limite du nombre de parties qui peut être corrompues:

1. **Pour $t < n/3$:** (lorsque moins d'un tiers des parties puisse être corrompu), les protocoles du calcul multi-partie sécurisé avec équité et livraison garantie de la sortie peuvent être obtenue pour toute fonction de dans un réseau point-à-point et sans hypothèse sur l'installation. Ceci peut être réalisé aussi bien dans le modèle calculatoire [7] (en supposant l'existence de permutations à trappe), et dans le modèle "information-theoretic" (canal privé) [73][74].
2. **Pour $t < n/2$:** (i.e, dans le cas d'une majorité honnête garantie), les protocoles de calcul multi-partie sécurisé (sans équité et livraison garantie de la sortie) peuvent être réalisés pour n'importe quelle fonction en supposant que les parties aient accès à un canal de diffusion. Ceci peut être réalisé dans le modèle calculatoire [60] (avec les mêmes hypothèses que ci-dessus), et dans le modèle "information-theoretic" [76].
3. **Pour $t \geq n/2$:** (i.e, lorsque le nombre de parties corrompues n'est pas limité), les protocoles du calcul multi-partie sécurisé (sans équité et livraison garantie de la sortie) peuvent être obtenus en supposant que les parties aient accès aux un canal de diffusion et, en outre supposer l'existence de permutations à trappe [6][7][77]. Ces résultats de faisabilité ne tiennent que dans le modèle calculatoire, des résultats analogues pour dans le modèle "information-theoretic" ne peuvent pas être obtenus lorsque $t \geq n/2$ [73].

En résumé, les protocoles du calcul multi-partie sécurisé existent pour n'importe quelle tâche de l'informatique distribuée. Dans le modèle calculatoire, cela vaut pour tous les nombres possibles de parties corrompues, avec la réserve qu'aucune majorité honnête n'existe, et équité et livraison garantie du résultat ne sont pas obtenus. Nous notons que les résultats ci-dessus tiennent tous à l'égard des adversaires malveillants. Cependant, dans les applications de la préservation de privacy en datamining en particulier dans l'algorithme de clustering k-means, les solutions proposées sous le modèle du calcul multi-partie sécurisé sont proposées dans le modèle semi-honnête de l'adversaire en ayant une exécution à temps polynomial probabiliste. Nous donnons dans ce qui suit le coté formel du calcul multi-partie sécurisé dans le modèle semi-honnête.

IV. Le calcul multi-partie sécurisé :

Le calcul multi – partie sécurisé (Secure multi-party computation) fait référence au problème général du calcul sécurisé d'une fonction a données distribuées. A. Yao [6], a initialement postulé le problème de la comparaison bipartie sécurisé (le protocole du millionnaire de Yao) et il a développé une solution prouvablement sécurisée. Ceci a été étendu au calcul multi partie par O. Goldreich et al. [8][7], qui ont développé le cadre (framework) formel du calcul multi partie sécurisé et ils démontrent que calculer confidentiellement (privately) une fonction est équivalent à la calculer d'une manière sécurisée.

IV.1 Notations d'écriture:

Dans ce qui suit nous faisons référence aux distributions de probabilités discrètes. Traditionnellement, une variable aléatoire est définie comme une fonction sur un espace d'échantillons entiers ou réels. Le terme variable aléatoire est utilisé aussi pour faire référence aux fonctions sur un espace d'échantillons de strings binaires. Par exemple, nous pouvons dire que X est une variable aléatoire qui a des valeurs dans l'ensemble des strings de telle sorte que $Prob(X=00)=1/4$ et $Prob(X=111)=3/4$.

Dans la plupart des cas l'espace de probabilités est celui de tous les strings d'une longueur donnée.

Toutes nos expressions de probabilités font référence à des fonctions de variables aléatoires qui sont définies à l'avance. Typiquement, nous pouvons écrire $Prob(f(X)=1)$, où X est une variable aléatoire définie à l'avance, et f est une fonction. Une importante convention est que toutes les occurrences du même symbole dans une expression de probabilité font référence à la même et l'unique variable aléatoire. Donc, si $E(.,.)$ est une expression dépendant de deux variables et X est une variable aléatoire, alors $Prob(E(X,X))$ dénote la probabilité que $E(x,x)$ se prenne quand x est choisi avec une probabilité $Prob(X=x)$. A savoir,

$$Prob(E(X, X)) = \sum_x Prob(X = x).val(E(x, x))$$

Où $val(E(X,X))$ est égal à 1 si $E(x,x)$ se prend et est égal à 0 sinon.

Donc, si X et Y sont deux variables aléatoires indépendantes, alors $Prob(E(X,Y))$ dénote la probabilité que $E(x,y)$ se prend quand la pair (x,y) est choisie avec la probabilité $Prob(X=x).Prob(Y=y)$. A savoir,

$$Prob(E(X, Y)) = \sum_x Prob(X = x).Prob(Y = y).val(E(x, y))$$

IV.2 Algorithme probabiliste en temps polynomial:

Le plus important dans cette discussion est l'association du calcul 'efficace' avec le calcul en un temps polynomial probabiliste. A savoir, nous considérons comme efficace seulement les algorithmes probabilistes (ie. la machine de Turing probabiliste) où le temps d'exécution est lié à un polynôme de la longueur de l'entrée. Une façon de voir les algorithmes probabilistes est de permettre à l'algorithme de faire des mouvements aléatoires ("coin tosses"). Formellement, ceci peut être modélisé par une machine de Turing dans laquelle la fonction de transition qui trace les pairs de la forme $(\langle état \rangle, \langle symbole \rangle)$ en deux triplets possibles de la forme $(\langle état \rangle, \langle symbole \rangle, \langle direction \rangle)$. Le prochain pas d'une telle machine est déterminé par un choix aléatoire de l'un des deux triplets. Ces choix aléatoires sont appelés "jets internes de pièce de monnaie" ("internal coin tosses") de la machine. La sortie de la machine probabiliste, M , pour une entrée x n'est pas un string, mais plutôt une variable aléatoire de strings comme des valeurs possibles.

Cette variable aléatoire notée $M(x)$, est induite par les jets internes de pièces (coin tosses) de M . $Prob(M(x) = y)$ signifie la probabilité que la machine M produit y sur l'entrée x . L'espace de probabilité est celui de tous les résultats possibles des jets internes de pièce de monnaie pris avec une distribution uniforme de probabilité.

Le cas simple est lorsque, sur l'entrée x , la machine M fait toujours le même nombre de jets internes de pièces indépendamment de leurs résultats. Puisqu'on ne considère que les machines à temps polynomiale, on suppose que le nombre de jets internes de pièces fait par M sur l'entrée x est indépendant de leurs résultats et il est noté par $t_M(x)$.

IV.3 Indistinguabilité calculable (computational indistinguishability):

Le concept du calcul efficace mène naturellement à un nouveau type d'équivalence entre les objets. "Des objets sont considérés à être calculablement équivalents s'ils ne peuvent pas être indiqués à part par aucune procédure efficace". La considération que des objets indistincts comme des objets équivalents est l'un des paradigmes de la science et les situations de la vie réelle. Donc nous croyons que la notion d'indistinguabilité calculable (computational indistinguishability) est fondamentale.

La formulation de la notion de l'indistinguabilité calculable est donnée comme standard dans la théorie de complexité, en considérant les objets comme des séquences infinies de strings. Donc, les séquences: $\{x_n\}_{n \in \mathbb{N}}$ et $\{y_n\}_{n \in \mathbb{N}}$ seraient calculablement indistinctes (computationally indistinguishable), si aucune procédure efficace ne peut les indiquer à part. En d'autres mots, aucun algorithme efficace, D , ne peut accepter infiniment plusieurs x_n , lorsqu'il rejette leur y_n contre - parties (Pour chaque algorithme efficace D et tout n suffisamment large, il se tient que D accepte x_n si et seulement si D accepte y_n).

Des objets qui sont calculablement indistincts dans le sens perçu au dessus, peuvent être considérés comme équivalents pour n'importe quel but pratique.

La discussion ci – dessus est naturellement étendue à l'arrangement probabiliste. En outre, comme nous allons le voir cette extension rapporte des conséquences très utiles. Lâchement parlant, deux distributions sont dites calculablement indistinctes si aucun algorithme efficace ne peut les indiquer à part. Donnons un algorithme efficace D , nous considérons la probabilité que D accepte un string pris de la première distribution. De la même manière, nous considérons la probabilité que D accepte un string de la deuxième distribution. Si ces deux probabilités sont proches, nous disons que D ne distingue pas les deux distributions. Encore, la formulation de cette discussion est avec le respect de deux séquences infinies de distributions de telles séquences sont appelée 'les ensembles probabilistes' (probability ensembles).

IV.3.1 Définition d'un ensemble probabiliste:

Soit I un ensemble dénombrable d'index.

Un ensemble indexé par I est une séquence de variables aléatoires indexées par I .

A savoir, $X = \{X_i\}_{i \in I}$, où les X_i sont des variables aléatoires, est un ensemble indexé par I .

Nous utilisons, à la fois N ou un sous ensemble de $\{0,1\}^*$ comme l'ensemble des index.

En général, dans nos applications, un ensemble de la forme $X = \{X_n\}_{n \in \mathbb{N}}$ a chaque variable aléatoire X_n couvrant des strings de longueur n .

Alors qu'un ensemble de la forme $X = \{X_w\}_{w \in \{0,1\}^*}$ aura chaque X_w couvrant des string de longueur $|w|$.

IV.3.2 Définition de l'indistinguabilité calculable :

Soit $S \subseteq \{0,1\}^*$ un ensemble de strings (chaines de caractères), S est dit l'ensemble des index.

Deux ensembles, $X = \{X_w\}_{w \in S}$ et $Y = \{Y_w\}_{w \in S}$ sont dit calculablement indistincts en un temps polynomial, si pour chaque algorithme probabiliste polynomial D , chaque polynôme $p(\cdot)$, et tout w suffisamment long:

$$|\Pr ob(D(X_w, w) = 1) - \Pr ob(D(Y_w, w) = 1)| < \frac{1}{p(|w|)}$$

Les probabilités dans cette définition sont prises sur les variables aléatoires correspondantes X_i (ou Y_i) et les jets internes de pièces (internal coin tosses) de l'algorithme D . (il est permis que l'algorithme D soit probabiliste).

IV.4 Le calcul biparti:

Le problème d'un protocole biparti est de spécifier un processus aléatoire qui transforme des paires d'entrées (une pour chaque partie) en des paires de sorties (résultats) une pour chaque partie. Ce processus est une fonctionnalité notée:

$f: \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, où $f = (f_1, f_2)$. Ainsi, pour chaque d'entrée (x, y) , la paire de sortie est une variable aléatoire $(f_1(x, y), f_2(x, y))$ prenant ses valeurs dans l'ensemble des strings..

La première partie (avec l'entrée x) veut obtenir $f_1(x, y)$ et la deuxième partie (avec l'entrée y) veut obtenir $f_2(x, y)$.

Cette fonctionnalité est souvent notée par $(x, y) \rightarrow (f_1(x, y), f_2(x, y))$

IV.5 Le modèle semi-honnête:

L'objectif d'un protocole biparti sécurisé est de protéger une partie honnête contre un comportement malhonnête des autres parties. Il existe deux types d'adversaires, pour lesquels les modèles de sécurité sont définis, le modèle semi honnête et le modèle malicieux. Dans le modèle malicieux, une partie peut refuser de participer au protocole, peut substituer ses entrées locales avec des entrées différentes ou peut quitter le protocole prématurément. Cependant, Le modèle semi honnête est plus simple et il est plus répandu dans les applications du datamining en particulier dans l'algorithme de clustering k-means. Une partie semi – honnête (aussi dite honnête mais curieuse) suit correctement les règles du protocole en utilisant ses entrées correctes (ses vraies entrées), à l'exception qu'elle garde un enregistrement de tous ses calculs intermédiaires durant l'exécution du protocole, et elle est libre après d'utiliser ce qu'elle voit durant l'exécution du protocole pour corrompre la sécurité.

IV.6 Le protocole biparti sécurisé :

Une définition formelle du calcul sécurisé dans le modèle semi honnête est donnée comme suit:

Soit Π un protocole biparti pour calculer la fonction f . La vue de la première partie, (resp. la deuxième) durant l'exécution de Π sur (x, y) dénoté $VIEW^{\Pi}_1$ (resp. $VIEW^{\Pi}_2$) est:

$$\begin{aligned} VIEW^{\Pi}_1 &= (x, r, m_1, \dots, m_i) \\ VIEW^{\Pi}_2 &= (y, r, m_1, \dots, m_i), \end{aligned}$$

où r représente le résultat du premier (resp. deuxième) jet interne de pièces, et m_i représente le $i^{ème}$ message qui a été reçu.

Le résultat de la première (resp. la deuxième) partie durant une exécution de Π sur (x, y) , dénoté $OUTPUT^{\Pi}_1$ (resp. $OUTPUT^{\Pi}_2$) est implicite dans la vue propre à une partie:

$$OUTPUT^{\Pi}(x, y) = (OUTPUT^{\Pi}_1(x, y), OUTPUT^{\Pi}_2(x, y)).$$

Nous disons que le protocole Π est sécurisé contre un adversaire semi honnête (calcule confidentiellement f), s'il existe des algorithmes (simulateurs) probabilistes en un temps polynomial, dénotés S_1 et S_2 , tels que:

$$\begin{aligned} \{(S_1(x, f_1(x, y)), f(x, y))\}_{x, y} &\stackrel{c}{=} \{(VIEW_1^{\pi}(x, y), OUTPUT^{\pi}(x, y))\}_{x, y} \\ \{(S_2(x, f_2(x, y)), f(x, y))\}_{x, y} &\stackrel{c}{=} \{(VIEW_2^{\pi}(x, y), OUTPUT^{\pi}(x, y))\}_{x, y} \end{aligned}$$

Où $\stackrel{c}{=}$ dénote l'indistinguabilité calculable pour les familles (non uniformes) des circuits de taille polynomiale.

Informellement, La preuve de privacy est donnée en comparant la sécurité dans le monde réel où les parties utilisent un protocole biparti sécurisé, avec celle du monde idéal où les parties envoient les entrées à une partie de confiance qui calcule la fonction et retourne uniquement le résultat aux parties. Pour garantir la privacy dans le modèle semi honnête, des outils de sécurité sont utilisés dans l'algorithme k-means durant le calcul des différentes fonctionnalités de l'algorithme, nous décrivant les plus utilisés.

V. L'évaluation sécurisée d'un circuit:

Le plus important résultat du calcul multi-parti sécurisé est la preuve constructive que n'importe quelle fonction polynomiale calculable peut être calculée d'une façon sécurisée. Ceci remonte à la définition de A.Yao et la solution du « problème des millionnaires » en 1982 [78] où deux millionnaires veulent découvrir qui est le plus riche, mais sans révéler leur fortune l'un à l'autre. Yao, à plus tard démontré que n'importe quelle fonction calculable représentée par un circuit booléen de taille polynomiale avec des entrées réparties entre deux parties, peut être évaluée de telle sorte qu'une partie ne peut apprendre que le résultat [6].

Dans le protocole de Yao une des parties calcule une version brouillée d'un circuit booléen pour évaluer la fonction désirée. Le circuit brouillé consiste au chiffrement de toutes les valeurs possibles des bits sur tous les fils possibles du circuit. Le nombre des chiffrements est approximativement $4m$, où m est le nombre de portes dans le circuit. Le chiffrement peut être à clé symétrique, qui a une longueur de texte chiffré de 64 bits. Le circuit brouillé est envoyé à l'autre partie, qui peut être évalué le circuit final pour avoir le résultat final. Pour bien expliquer le protocole de Yao, il est utile de rappeler d'abord comment un tel circuit est calculé.

Soit f une fonction à temps polynomial (assumons maintenant que c'est une fonction déterministe), et soit x et y les entrées de la partie A et B respectivement.

La première étape est de voir la fonction f comme un circuit booléen C .

Le circuit $C(x, y)$ est calculé porte par porte (gate-by-gate), à partir des fils d'entrée jusqu'aux fils de sortie. Une fois les fils entrants vers une porte g ont obtenu les valeurs $\alpha, \beta \in \{0, 1\}$, il est possible de donner aux fils sortant de la porte, la valeur $g(\alpha, \beta)$. La sortie du circuit est donnée par les valeurs obtenues dans les fils de sortie du circuit. Ainsi, pour l'essentiel, le calcul d'un circuit consiste à affecter les valeurs '0' et '1' appropriées aux fils du circuit.

Dans la description ci – dessous, nous faisons référence à quatre types différents des fils dans un circuit:

Les fils "circuit - input": reçoivent les valeurs d'entrée x et y .

Les fils "circuit-output": portent la valeur $C(x, y)$.

Les fils "gate-input": qui entrent dans une certaine porte g .

Les fils "gate-output": qui quittent une certaine porte g .

Nous présentons actuellement une description de haut niveau du protocole de Yao. La construction est actuellement un compilateur qui prend n'importe quelle fonction à temps polynomial f , ou actuellement un circuit C qui calcule f , et construit un protocole pour calculer f d'une manière sécurisée dans la présence d'adversaires semi honnêtes. Dans un protocole sécurisé, la seule valeur apprise par une partie doit être son résultat. Donc, toutes les valeurs portées à tout les fils qui ne sont pas "circuit-output", ne doivent pas être apprise par aucune partie.

Considérons deux parties : Alice (expéditeur) à l'entrée x , Bob (récepteur) à l'entrée y . Alors les étapes du protocole sont :

- Alice construit un circuit brouillé et l'envoie à Bob (Déformation du circuit).
- Alice et Bob interagissent de telle sorte que le récepteur obtient les clés des fils d'entrée associés à x et y . (interaction pour donner les clés d'Alice à Bob).
- En donnant ces clés, le récepteur calcule le circuit. (calcul du circuit)
- Le récepteur prend sa sortie, et envoie la sortie chiffrée de l'expéditeur à l'expéditeur.
- Conclure le protocole.

V.1 Construction du circuit brouillé :

Pour chaque fil dans le circuit, deux valeurs aléatoires sont spécifiées de telle sorte qu'une valeur représente 0 et l'autre représente 1. Par exemple, soit w l'étiquette d'un certain fil, alors deux valeurs k_w^0 et k_w^1 sont choisies, où k_w^σ représente le bit σ . Une importante observation ici est que même si l'une des parties sait la valeur de k_w^σ obtenu par le fil w , ceci n'aide pas à déterminer si $\sigma = 0$ ou $\sigma = 1$ (car k_w^0 et k_w^1 sont équitablement distribués). Bien sur la difficulté avec une telle idée est qu'elle fait apparaître le calcul du circuit impossible.

Soit g une porte avec des fils entrants w_1 et w_2 et le fil de sortie w_3 .

En donnant deux valeurs aléatoires k_1^σ et k_2^τ , il n'apparaît pas possible de calculer la porte car σ et τ sont inconnus.

Nous avons besoin donc d'une méthode pour calculer la valeur du fil de sortie de la porte (aussi des valeurs aléatoires k_3^0 ou k_3^1), en donnant les valeurs des deux fils d'entrée à la porte. En bref, cette méthode implique de fournir « une table de calcul brouillée » qui transforme les valeurs d'entrée aléatoires en des valeurs de sorties

aléatoires. Cependant, cette transformation peut avoir la propriété, qui, en donnant deux valeurs d'entrée, il est possible d'apprendre la valeur de sortie qui correspond à la sortie de la porte (cette valeur doit être gardée secrète).

Ceci est accompli en voyant les quatre possibles entrées à la porte: k_1^0 , k_1^1 , k_2^0 , k_2^1 comme des clés de chiffrement. Aussi, les valeurs de sortie k_3^1 et k_3^0 , qui sont aussi des clés, sont chiffrés sous les clés appropriées des fils entrants. Par exemple, soit g une porte « **OR** ». Alors, la clé k_3^1 est chiffrée sous la paire de clés associée aux valeurs $(1,1)$, $(1,0)$ et $(0,1)$. En revanche, la clé k_3^0 est chiffrée sous la paire de clé $(0,0)$. Voir table,

Fil d'entrée w_1	Fil d'entrée w_2	Fil de sortie w_3	Table de calcul
k_1^0	k_2^0	k_3^0	$E_{k_1^0}(E_{k_2^0}(k_3^0))$
k_1^0	k_2^1	k_3^1	$E_{k_1^0}(E_{k_2^1}(k_3^1))$
k_1^1	k_2^0	k_3^1	$E_{k_1^1}(E_{k_2^0}(k_3^1))$
k_1^1	k_2^1	k_3^1	$E_{k_1^1}(E_{k_2^1}(k_3^1))$

La table de calcul "OR".

A noter qu'en donnant les clés du fil d'entrée k_1^α et k_2^β correspondant à α et β , et les quatre valeurs de la table (trouvé dans la quatrième colonne de la table), il est possible à déchiffrer et obtenir la clé du fil de sortie $k_3^{g(\alpha,\beta)}$. En outre, comme exigé ci-dessus, ceci est la seule valeur qui peut être obtenu (les autres clés sur les fils d'entrée ne sont pas connus et donc seule une valeur de la table qui peut être déchiffrée). En un autre mot, il est possible de calculer la clé de sortie $k_3^{g(\alpha,\beta)}$ de la porte, et seulement cette clé, sans rien apprendre sur les valeurs réelles de α , β ou $g(\alpha, \beta)$. (Nous notons que les valeurs de la table sont aléatoirement ordonnées de telle sorte que la position des clés ne révèle rien sur la valeur qui est associée avec). En dépit de cet ordre aléatoire, la construction spécifique est telle que: en prenant une paire de clés d'un fil d'entrée, il est possible de localiser l'entrée de la table qui est chiffrée par ces clés.

La construction su citée peut être décrite métaphoriquement en utilisant des "boîtes verrouillées" (locked boxes). L'idée basique, est que chaque fil est alloué deux clés du cadenas; une clé est associée au bit 0, l'autre au bit 1. Alors, pour chaque porte, quatre boîtes doublement verrouillées sont fournies, où chaque boîte est associé à une ligne dans la table de vérité calculant la porte (i.e, une boîte est associée aux entrées $(0,0)$, une autre $(0,1)$ et ainsi de suite). Les quatre boîtes sont verrouillées de telle sorte que chaque paire de clés (un de chaque fil d'entrée) ouvre exactement une boîte. En outre, dans chaque boîte, une clé unique relative au fil de sortie est stockée. Cette clé est choisie de telle sorte qu'elle associe correctement les bits d'entrée au bit de sortie de la porte. (Par exemple, si les clés qui ouvrent la porte sont associées à 0 et 1 et la porte calcule la fonction **AND**, alors la clé dans la boîte est la clé associée à 0 dans le fil de sortie.). La première importante observation est que, donnant l'ensemble des clés qui sont associées aux entrées des parties, il est possible de calculer le circuit en ouvrant les boîtes verrouillées un à la fois (pour chaque porte, une seule boîte s'ouvrira). Le processus se termine aux boîtes de sortie de la porte, qui peut contenir la sortie

actuelle plutôt qu'une clé. La deuxième importante observation est que le calcul du circuit ne révèle absolument aucune information au-delà de la sortie elle-même. Ceci est dû au fait que les clés ne sont pas étiquetées et ainsi il est impossible de savoir si une clef donnée est associée à zéro ou à un. Ce tout se tient sous l'hypothèse que les clés associées avec les fils du circuit d'entrée sont obtenues d'une « manière aveuglée » (oblivious manner) qui ne révèle pas l'association avec les entrées des parties. En outre nous devons supposer qu'un seul ensemble de clés est fourni (ainsi dans chaque porte seulement une unique boîte est ouverte). Naturellement, dans la construction actuelle du circuit brouillé, le double chiffrement remplace les boîtes doublement verrouillées et les clés de déchiffrement remplacent les clés physiques du cadenas.

V.2 Interaction entre les deux parties :

Il reste pour nous de décrire comment le récepteur obtient les clés pour les fils circuit-input. Si son $i^{\text{ème}}$ bit d'entrée est 0 et le fil w_i reçoit cette entrée, alors l'expéditeur remet juste au récepteur la clé (the string) k_i^0 .

A noter que puisque toutes les clés sont équitablement distribuées, le récepteur ne peut apprendre rien sur les entrées de l'expéditeur à partir de ces clés. Concernant le récepteur, c'est plus problématique. L'expéditeur ne peut pas le remettre toutes les clés se rapportant à son entrée (i.e les deux clés 1 et 0 sur le fil d'entrée du récepteur), car ça permet au récepteur de calculer plus que juste sa sortie. (Pour une entrée donnée x de l'expéditeur, ceci permet au récepteur de calculer $C(x, y)$ pour chaque y). Ceci est une extra information par rapport à la valeur unique $C(x, y)$. D'autre part, le récepteur ne peut pas dire ouvertement à l'expéditeur quelles clés à m'envoyer. Car, alors l'expéditeur serait en savoir l'entrée du récepteur.

V.3 Calcul de la sortie :

En donnant la clé de l'expéditeur au récepteur, le récepteur calcule le circuit comme déjà décrit, obtient la sortie et conclue le protocole. Cette description fait apparaître seulement comment on obtient ces sorties, en ignorant la sortie de l'expéditeur. Cependant, la sortie du récepteur peut inclure la sortie de l'expéditeur dans une forme chiffré (où seulement l'expéditeur qui connaît la clé de déchiffrement). Puis le récepteur peut juste expédier à l'expéditeur sa sortie à la fin du calcul. Puisque la sortie de l'expéditeur est chiffrée, le récepteur n'apprend rien davantage que sa propre sortie, comme exigé.

V.4 Discussion :

Cette approche, si séduisante de sa simplicité et de sa généralité, sa complexité dépend de la taille du circuit, qui dépend à son tour de la taille des entrées. Ceci est impraticable pour les grandes entrées et de nombreuses parties, comme si le cas du datamining. Cependant, pour un nombre d'opérations simples à deux parties la complexité est raisonnable. Pour deux parties, le coût du calcul est $O(m)$ où m est la taille du circuit.

L'évaluation sécurisée d'un circuit de Yao permet le calcul efficace des fonctions à entrées limitées (comme la comparaison de deux nombres) à cause de son coût élevé, et elle est utilisée fréquemment dans l'algorithme k-means pour la comparaison la sécurisée des distances.

VI. Le chiffrement homomorphe :

Un outil fort de sécurité pour le calcul d'une grande gamme de fonctions est le chiffrement homomorphe. Avec le chiffrement homomorphe nous pouvons éviter le chiffrement au niveau bit à partir de circuit brouillé décrit dans la section IV.1. Le chiffrement homomorphe est une classe spéciale des schémas à clé publique, sémantiquement sécurisé et qui, en plus des garanties standards, satisfait les propriétés suivantes:

$$\begin{aligned} E(t_1) \cdot E(t_2) &= E(t_1 + t_2) \\ E(t_1)^{t_2} &= t_2 \cdot E(t_1) \end{aligned}$$

Où E est la fonction de chiffrement du cryptosystème à clé publique et t_1, t_2 sont des éléments du domaine de données.

La propriété d'homomorphisme de ces schémas permet à une troisième partie d'opérer sur des valeurs cachées, pour avoir une valeur cachée du résultat.

Nous décrivons dans ce qui suit les principales propriétés des ces schémas de chiffrement et nous détaillons ensuite l'un des plus répandu de ces schémas.

VI.1 La sécurité sémantique :

Le premier critère de sécurité propre à un schéma de chiffrement à clé publique est la difficulté du problème de l'inversion de sa fonction de chiffrement. Formalisant un autre critère de sécurité qu'un schéma de chiffrement devrait satisfaire au delà à être à sens unique, Shafi Goldwasser et Silvio Micali [79] introduisirent la notion de sécurité sémantique aussi connu sous le vocable d'indistinguabilité des textes chiffrés. Est un critère de sécurité important dans le cadre des algorithmes de chiffrement asymétrique.

Pour qu'un cryptosystème basé sur la cryptographie asymétrique soit sémantiquement sûr, il faut que l'adversaire (avec une puissance calculatoire limitée) soit incapable d'obtenir des informations significatives sur le message (le texte clair) en ayant seulement le texte chiffré et la clé publique correspondante (attaque à texte clair choisi) (IND-CPA).

La définition initialement proposée n'offre pas un moyen franc pour prouver la sécurité des cryptosystèmes. Goldwasser et Micali [80] démontrent que la sécurité sémantique est équivalente à la propriété d'indistinguabilité des textes chiffrés 'ciphertext indistinguishability'. Cette équivalence permet des preuves de sécurité des cryptosystèmes, et en conséquence, la définition d'indistinguabilité est utilisée plus généralement que la définition originale de la sécurité sémantique.

'Indistinguabilité' sous l'attaque à texte plein choisi est généralement définie par le jeu suivant :

- Un adversaire polynomial probabiliste à temps limité a une clé publique, qui peut être utilisé pour générer n'importe quel nombre de textes chiffrés (en temps polynomial)
- L'adversaire génère deux messages m_0, m_1 de même longueur et les transmet à un oracle de défi avec la clé publique.
- L'oracle de défi sélectionne l'un des messages en renversant justement une pièce de monnaie, chiffre le message avec la clé publique, et retourne le texte chiffré résultant ' c ' à l'adversaire.

Les algorithmes de chiffrement sémantiquement sécurisé incluent les schémas Golwasser-Micali, El Gamal et Paillier sont considérés comme prouvablement sécurisé, que leur sécurité sémantique peut être réduite à la résolution de certains problèmes mathématiques durs (Le problème décisionnel de Diffie-Hellman, la résiduosit  quadratique ou la résiduosit  composite). Autres algorithmes non s mantiquement s curis  comme RSA, peuvent l' tre (sous des hypoth ses plus fortes) en utilisant les techniques g n riques de Padding comme OAEP (Optimal Asymmetric Encryption Padding) propos e en 1994 M.Bellare et P.Rogaway [81].

Le chiffrement d'un message m est toujours un groupe d'éléments de la forme : $E(m,r) = g^m r^e \in G$, où e est un entier publique, g un élément fixe et publique dans G ,

et r est choisi en aléatoire dans un sous groupe multiplicatif R de G . Puisque R est un sous groupe, ces schémas ont la propriété d'homomorphisme additif: le chiffrement de m_1+m_2 peut être obtenu comme $E(m_1, r_1).E(m_2, r_2) = E(m_1+m_2, r_1.r_2)$.

VI.3 Quelques schémas de chiffrement homomorphe :

Le premier cryptosystème homomorphe, appelé le cryptosystème de Goldwasser-Micali, a été proposé en 1984 [80] lorsqu'ils ont introduit la notion de chiffrement probabiliste. La sécurité sémantique est prouvée sous l'hypothèse de la résiduosit  quadratique sur le groupe Z_n^* o  n est le produit de deux grands nombres premiers p et q . En raison de son taux d'expansion prohibitif au cours du chiffrement (chaque bit du texte clair est crypt  comme un texte chiffr  d'au moins 1024 bits), le sch ma s'av re inefficace. Une extension naturelle de ce cryptos  t me est le cryptos  t me de Benaloh [82] qui permet le cryptage des blocs de grande taille   la fois, il fait appelle au probl me de la r  siduosit  de degr  r dans le groupe Z_n^* o  n est le produit de deux grands nombres premiers p et q , il peut  tre vu comme une g n ralisation directe du Goldwasser-Micali. Cependant, le d chiffrement est inefficace puisqu'il est bas  sur une recherche exhaustive sur tous les textes clairs possibles. En 1997, D. Naccache et J. Stern [83] propos rent un sch ma de chiffrement  galement bas  sur la classe de r  siduosit  de haut degr  dans Z_n^* . L'objectif  tait d'am liorer le taux d'expansion trop  lev  des sch mas pr c dents et le chiffrement Naccache-Stern fut le premier sch ma probabiliste s mantiquement sur de bande passante significative. L'id e essentielle est de choisir r de plus grande taille mais lisse et d'utiliser l'algorithme Polig-Hellman lors du d chiffrement. La s curit  s mantique est toujours prouv e sous l'hypoth se de r  siduosit  primaire.

Au m me moment, Okamoto et Uchiyama [84] ont propos  une tr s diff rente am lioration du sch ma de Benaloh- Fischer en changeant la structure du groupe. Ils s lectionnent $G = R = Z_N^*$ avec $N = p^2q$ au lieu de $N = pq$ comme en Goldwasser-Micali, Benaloh-Fischer and Naccache-Stern. Le sch ma n'est pas   sens unique car on peut factoriser N . La s curit  s mantique est prouv e sous l'hypoth se de p -sous groupe. Mais ils existent de tr s simples attaques   texte chiffr  choisi qui peuvent recouvrir la factorisation de N . Le sch ma atteint des bandes passantes similaires   Naccache-Stern, cependant le d chiffrement est plus efficace.

Paillier a r cemment propos  dans [85] une extension du sch ma d'Okamoto-Uchiyama o  $N = pq$, $M = Z_N$, $G = Z_{N^2}^*$, $R = Z_N^*$. La s curit  s mantique est prouv e sous l'hypoth se de r  siduosit  composite d cisionnelle. Le cryptos  t me r sultant est plus efficace que les sch mas pr c demment mentionn s. En plus il n'existe pas d'attaques adaptatives   texte chiffr  choisi recouvrant la cl  secr te. Aussi le cryptos  t me de Paillier fournit des algorithmes de chiffrement et de d chiffrement rapides et il chiffre des messages de 1024-bits en des textes d'au moins de 2048-bits, ce qui est raisonnable si on travaille sur un large texte clair. Pour ces raisons, les cryptos  t mes de Paillier sont actuellement les meilleurs candidats des cryptos  t mes avec homomorphisme additif.

VI.4 Le cryptos  t me de Paillier :

Le cryptos  t me de Paillier [85], invent  par Pascal PAILLIER en 1999, est un algorithme probabiliste asym trique pour la cryptographie   cl  publique. Le probl me de calculer les classes du ni me r  sidu est connu qu'il est calculablement difficile. Ceci est connu comme l'hypoth se de r  siduosit  composite, sur laquelle le

cryptosystème est basé. Le cryptosystème comme vu fournit une sécurité sémantique contre l'attaque à texte clair choisi (IND-CPA). La capacité à distinguer avec succès le texte chiffré à défi remonte essentiellement à décider la résiduodité composite.

VI.4.1 Notations :

1. Z_n est l'anneau des entiers modulus n , noté aussi Z/nZ .
2. Z_n^* est le groupe des inversibles modulus n ou groupe des unités de l'anneau Z/nZ .
3. L'indicatrice d'Euler ϕ est la fonction de l'ensemble des entiers strictement positifs dans lui-même, qui à n associe le nombre d'entiers positifs inférieurs ou égaux à n et premiers avec n .
4. L'ordre du groupe Z_n^* : $|Z_n^*| = \phi(n)$.
5. La fonction de Carmichael d'un nombre entier positif n , dénoté $\lambda(n)$, est définie comme le plus petit entier positif m tel que : $a^m = 1 \mod n$ pour n'importe quel entier a premier avec n . En un autre mot, le nombre $\lambda(n)$ définit l'exposant du groupe Z_n^* : $\exp(Z_n^*) = \lambda(n)$

Dans ce qui suit, on considère que :

1. n est le produit de deux grands nombres premiers p et q et il sera noté : $n = pq$.
2. Il est supposé que p est premier relativement à $q-1$ et de même, q est premier avec $p-1$.
3. Le groupe choisi est $G = Z_{n^2}^*$ d'unité $e = 1$, d'ordre $\phi(n^2) = pq(p-1)(q-1) = n\phi(n)$ et d'exposant $\exp(Z_{n^2}^*) = pq \text{ppcm}(p-1, q-1) = n\lambda(n)$. Dans ce qui suit, il est adopté λ au lieu de $\lambda(n)$.
4. On a donc: $\forall w \in Z_{n^2}^* : w^\lambda = 1 \mod n$
 $w^{n\lambda} = 1 \mod n^2$

VI.4.2 Résiduodité composite de degré n :

Définition :

Un nombre z est dit un n ième résidu modulo n^2 s'il existe un nombre entier $y \in Z_{n^2}^*$, tel que :

$$z = y^n \mod n^2.$$

L'ensemble des n èmes résidus est un sous groupe multiplicatif de $Z_{n^2}^*$ d'ordre $\phi(n)$.

Chaque n ième résidu z a exactement n racines de degré n .

Le problème de décider la n ième résiduodité, i.e distinguer les n èmes résidus des non n èmes résidus va être noté CR[n]. Décider la n ième résiduodité est connu être calculablement difficile (décider si un élément du groupe $Z_{n^2}^*$ est un n ième résidu).

Conjecture : Il n'existe pas un distingueur à temps polynomial pour les n èmes résidus modulo n^2 , c.à.d. CR[n] est intractable.

L'hypothèse d'intractabilité est nommée l'assomption décisionnelle de résiduodité composite. A cause de l'auto-réduction aléatoire, la validité cette assomption dépend seulement du choix de n : Si n est dur à factoriser, décider la résiduodité de degré n est calculatoirement difficile.

VI.4.3 Classe de résiduosité composite:

Soit g un élément de $Z_{n^2}^*$, on dénote par ε_g la fonction sur les entiers définit par :

$$\begin{aligned} Z_n \times Z_n^* &\rightarrow Z_{n^2}^* \\ (x, y) &\rightarrow g^x \cdot y^n \bmod n^2 \end{aligned}$$

En dépendant de g , la fonction ε_g a quelques propriétés intéressantes:

Lemme1: si l'ordre de g est un multiple non nul de n , alors ε_g est bijectif.

Nous dénotons par $B_\alpha \subset Z_{n^2}^*$, l'ensemble des éléments d'ordre $n\alpha$ et par B leur union disjointe pour $\alpha = 1, \dots, \lambda$. Dans ce cas g est appelé base de résiduosité de degré n .

VI.4.3.1 Définition de la classe de résiduosité composite :

Supposons que $g \in B$. Pour un élément $w \in Z_{n^2}^*$, on appelle la n ième classe de résiduosité de w selon g , l'unique entier $x \in Z_n$, pour lequel il existe $y \in Z_n^*$ tel que :

$$\varepsilon_g(x, y) = w$$

La classe de w est notée : $[w]_g$

Lemme2: $[w]_g = 0$ si et seulement si w est le n ième résidu modulo n^2 .

$$\forall w_1, w_2 \in Z_{n^2}^* \quad [w_1 w_2]_g = [w_1]_g + [w_2]_g \bmod n$$

Donc, la fonction de classe de résiduosité est un homomorphisme de $(Z_{n^2}^*, \times)$ à $(Z_n, +)$.

VI.4.3.2 Calcul de la classe de résiduosité composite :**Définition :**

Le problème de la classe de résiduosité composite est le problème calculatoire $\text{Class}[n]$ définit comme suit : soit $w \in Z_{n^2}^*$ et $g \in B$, calculer $[w]_g$.

Conjecture :

Si n est dur à factoriser alors il n'existent pas d'algorithme probabiliste polynomial qui résolve $\text{Class}[n]$ i.e. $\text{Class}[n]$ est difficile. Cette conjecture est appelée « hypothèse de la classe de résiduosité composite » (Computational Composite Residuosity Assumption CCRA).

Calcul :

Soit l'ensemble S_n définit comme suit : $S_n = \{u < n^2 / u = 1 \bmod n\}$ est un sous groupe multiplicatif des entiers modulus n^2 , pour lequel, on peut définir la fonction L , telle que : $\forall u \in S_n \quad L(u) = (u-1)/n$.

Lemme : $\forall w \in Z_{n^2}^* \quad L(w^\lambda \bmod n^2) = \lambda [w]_{1+n} \bmod n$

Pour n'importe quel $g \in B$ et $w \in Z_{n^2}^*$, nous pouvons alors calculer :

$$\frac{L(w^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} = [w]_g \bmod n \quad (1)$$

Puisque $L(g^\lambda \bmod n^2) = \lambda [g]_{1+n} \bmod n$, on a $L(g^\lambda \bmod n^2) \in Z_n$, donc pour que $L(g^\lambda \bmod n^2)$ soit inversible modulo n , il faut que le $\text{pgcd}(L(g^\lambda \bmod n^2), n) = 1$.

VI.4.4 Le chiffrement probabiliste de Paillier :

Le schéma de chiffrement est basé sur le problème de la classe de résiduosit  composite. La d marche consiste   utiliser \mathcal{E}_g pour le chiffrement et l' galit  (1) pour le d chiffrement.

La g n ration des param tres :

1. Choisir deux grands nombres premiers p et q al atoirement et ind pendamment l'un de l'autre.
 2. Calculer $n = pq$ et $\lambda = \text{ppcm}(p-1, q-1)$
 3. S lectionner al atoirement un entier g tel que $g \in Z_{n^2}^*$
 4. S'assurer que n divise l'ordre de g en v rifiant l'existence de l'inverse multiplicatif modulaire suivant : $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$
Ceci peut  tre effectivement v rifi  si : $\text{pgcd}(L(g^\lambda \bmod n^2), n) = 1$.
- Les cl s publiques de chiffrement sont (n, g)
 - Les cl s priv es de chiffrement sont (p, q) ou λ

Le chiffrement :

1. Soit m le message   chiffrer o  $m \in Z_n$
2. S lectionner un al atoire r o  $r \in Z_n$
3. Calculer le texte chiffr  comme : $c = g^m \cdot r^n \bmod n^2$

Le d chiffrement :

1. Le texte chiffr  $c \in Z_{n^2}^*$
2. Calculer le message $m = L(c^\lambda \bmod n^2) \cdot \mu \bmod n$

VI.4.4.1 Preuve de s curit :

Le caract re   sens unique de ce sch ma de chiffrement est  quivalent   l'hypoth se CCRA. Sa s curit  s mantique est, elle,  quivalente   l'hypoth se CR[n].

1. Non inversion de la fonction de chiffrement:

L'inversion de ce sch ma de chiffrement est  quivalente   r soudre le probl me de la classe de r siduosit  composite Class[n].

2. S curit  s mantique :

La s curit  s mantique de ce sch ma de chiffrement est bas e sur la difficult  du probl me CR[n].

Preuve : Supposons que m_0 et m_1 sont deux messages connus et c le texte chiffr  de soit m_0 ou m_1 . Suite au lemme 2, c est le texte chiffr  de m_0 si et seulement si $c g^{-m_0} \bmod n^2$ est un ni me r sidu. Donc, un attaquant   texte clair choisi doit d cider la r siduosit  composite, ce qui est le probl me CR[n].

VI.4.4.2 Propriétés homomorphiques :

Une caractéristique notable du cryptosystème de Paillier est ses propriétés homomorphiques. Comme la fonction de chiffrement est additivement homomorphe, les identités suivantes peuvent être décrites:

a. L'addition homomorphe des textes clairs:

- Le produit de deux textes chiffrés décrypte la somme de leurs textes clairs correspondants : $D(E(m_1, r_1) \times E(m_2, r_2) \bmod n^2) = m_1 + m_2 \bmod n$.

- Le produit d'un texte chiffré avec un texte clair soulevant g déchiffre les textes clairs correspondants : $D(E(m_1, r_1) \times g^{m_2} \bmod n^2) = m_1 + m_2 \bmod n$.

b) La multiplication homomorphe des textes clairs:

Un texte clair chiffré, élevé à la constante k décrypte le produit du texte plein et la constante: $D(E(m_1, r_1)^k \bmod n^2) = k \times m_1 \bmod n$.

Cependant, en donnant les chiffrements de deux messages, il n'existe pas un moyen connu pour calculer le chiffrement du produit de ces messages sans connaître la clé privée.

Les propriétés d'homomorphisme de ces cryptosystèmes permettent de calculer d'une façon sécurisée le produit scalaire des vecteurs de données dans les applications du datamining, en particulier dans l'algorithme k-means pour le calcul des distances. Nous décrivons dans ce qui suit l'un des protocoles les plus répandus du produit scalaire privé.

VII. Le produit scalaire sécurisé :

Dans le contexte de la préservation de privacy en datamining, plusieurs protocoles privés (partagés) du produit scalaire ont été proposés. L'objectif est que l'un des participants obtient le produit scalaire privé des vecteurs de toutes les parties. En outre, il est souvent exigé qu'aucune information sur les vecteurs privés, sauf ce qui peut être en déduit du produit scalaire, ne soit révélée durant le protocole. Ainsi, puisque les demandes d'exploration de données manipulent des quantités énormes de données, il est souhaitable que le protocole du produit scalaire soit aussi très efficace. L'un des premiers protocoles du produit scalaire sécurisé est le protocole donné par Vaidya et Clifton [86] qui tente de calculer le produit scalaire d'une manière sécurisée en calculant le produit scalaire sur la version brouillée des vecteurs binaires de données, de telle sorte qu'à la fin du protocole les deux parties obtiennent le produit scalaire commun sans qu'aucune ne voit le vecteur de l'autre. Un autre protocole, plus général a été aussi proposé par Du et Atallah [87] pour le calcul du produit scalaire partagé et sécurisé, en utilisant le transfert inconscient « oblivious transfer ». Cependant, dans [88], les auteurs montrent que le protocole de Vaidya et Clifton n'est pas privé et que celui de Du et Attallah peut être cassé. A savoir, ils permettent à une des deux parties de retrouver l'entrée privée d'une autre partie avec une probabilité très proche de un, après l'exécution du protocole correspondant pour la première fois. Comme un résultat positif, les auteurs proposent un autre protocole de produit scalaire sécurisé basé sur le chiffrement homomorphe, ils prouvent que le nouveau produit scalaire est privé sous les hypothèses cryptographiques standards. Plus précisément, il n'existe pas un algorithme probabiliste à temps polynomial qui

peut substituer Alice (resp. Bob) peut obtenir une quantité non négligeable d'information sur Bob (resp. Alice). Ceci veut dire, en particulier, que ce protocole peut être utilisé un nombre polynomial de fois avec n'importe quel vecteurs privés d'Alice ou de Bob dans n'importe quel contexte. Ainsi, la version « communication-optimale » est plus efficace en communication que les protocoles de Du-Attallah et Vaidya-Clifton. Nous décrivons dans ce qui suit, ce protocole de produit scalaire sécurisé.

VII. 1 Définition d'un protocole de produit scalaire:

Nous définissons un protocole entre Alice et Bob comme un protocole de produit scalaire (SP), lorsque Bob obtient sur les entrées privées d'Alice:

$\mathbf{x} = (x_1, \dots, x_N) \in \mathbb{Z}_m^N$ et les entrées privées de Bob $\mathbf{y} = (y_1, \dots, y_N) \in \mathbb{Z}_m^N$, le produit scalaire : $x \cdot y = \sum_{i=1}^N x_i \cdot y_i$.

VII.2 Définition d'un protocole partagé de produit scalaire:

Un protocole est un protocole de produit scalaire partagé (SSP), lorsqu'Alice reçoit une valeur aléatoire $s_A \in \mathbb{Z}_m$ uniformément distribuée, et Bob reçoit une valeur dépendante $s_B \in \mathbb{Z}_m$ de telle sorte que : $s_A + s_B \equiv (x \cdot y) \mod m$.

VII.3 Définition du protocole de produit scalaire privé:

Un protocole de produit scalaire est dit privé lorsqu'après l'exécution du protocole, Bob n'obtient aucune autre connaissance que $x \cdot y$ et Alice n'obtient aucune nouvelle connaissance. En particulier, Alice finit par ne rien savoir de neuf au sujet du vecteur de Bob, et Bob finit par ne rien savoir au sujet du vecteur d'Alice ce qui n'est pas impliqué par x et $x \cdot y$.

L'objectif des protocoles partagés et privés du produit scalaire est que l'un des participants obtient le produit scalaire des vecteurs privés de toutes les parties.

VII. 4 Le protocole privé cryptographique SSP :

Dans ce qui suit, nous décrivons un protocole SSP basé sur le chiffrement homomorphe. A noter qu'un protocole SP privé peut être obtenu à partir de SSP en définissant $s_B \leftarrow 0$.

Théorème 1 :

Assumons que $\Pi = (Gen, Enc, Dec)$ est un cryptosystème homomorphe à clé publique sémantiquement sécurisé avec $P(sk) = \mathbb{Z}_m$ pour un grand m . Posons,

$$\mu = \lfloor \sqrt{m/N} \rfloor.$$

Le protocole 1 est un protocole SSP sécurisé dans le modèle semi honnête, en supposant que $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_\mu^N$. La privacy d'Alice est garantie lorsque Bob est une machine probabiliste à temps polynomial. La privacy de Bob est considéré comme « information-theoretical ».

Protocole1 : Le protocole homomorphe privé SSPEntrée privées : Les vecteurs privés $x, y \in \mathbb{Z}_\mu^N$.Sorties privées : Les parts $s_A + s_B \equiv x \cdot y \pmod{m}$

Alice :

1. Générer une paire de clé privée et publique (sk, pk) .
2. Envoyer pk à Bob.
3. Pour $i \in \{1, \dots, N\}$:
 - 3.1. Générer une nouvelle chaîne de caractère aléatoire r_i .
 - 3.2 Envoyer $c_i = Enc_{pk}(x_i; r_i)$ à Bob.

Bob :

4. $w \leftarrow \prod_{i=1}^N c_i^{y_i}$.
5. Générer un texte aléatoire s_B et une chaîne de caractère aléatoire r' .
6. Envoyer $w' = w \cdot Enc_{pk}(-s_B; r')$ à Alice.

Alice :

7. Calculer $s_A = Dec(w') = x \cdot y - s_B$

Preuve :

Clairement, le protocole est correct si tous les participants sont honnêtes. Puisque le cryptosystème est sémantiquement sécurisé, Bob voit seulement N textes chiffrés, pour lesquelles, il ne peut pas découvrir les textes clairs. D'autre part, Alice ne voit qu'un chiffrement aléatoire de $s_A = x \cdot y - s_B$, où s_B est aléatoire. Mais Alice a la clé, donc elle peut décrypter le message w . Ainsi, Alice n'obtient aucune information du tout.

VII.5 Considérations pratiques :

A noter qu'Alice et Bob ont besoin d'exécuter ce protocole plusieurs fois, ils peuvent réutiliser les mêmes clés publiques et privées, donc la première étape peut être exécutée une seule fois. La cryptographie à clé publique est gourmande en calcul. Pour estimer le coût de calcul du nouveau protocole du produit scalaire, le nombre des chiffrements, des déchiffrements et des multiplications des textes chiffrés. Bob doit manipuler N exponentiations et un chiffrement. Alice doit manipuler N chiffrements et 1 déchiffrement.

VII.6 Le coût de communication estimé :

Le seul sérieux inconvénient des nouveaux protocoles est le coût de communication. Puisqu'Alice envoie N textes chiffrés c_i . Le coût est k'/μ , où k' est seulement la taille de chaque texte chiffré en bits. Lorsqu'on utilise le plus actuellement efficace des cryptosystèmes de chiffrement homomorphes sémantiquement sécurisés (exemple : celui de Paillier), $k' \approx 2048$.

VIII. Autres primitives de sécurité :

Il existe encore d'autres primitives de sécurité qui ont été utilisé dans l'algorithme k-means.

VIII.1 Les schémas de secret partagé additif (additive secret sharing schemes) :

Un schéma de secret partagé [89] permet à t participants parmi n de recouvrir un secret, alors qu'un ensemble de moins de t participants n'apprennent rien du secret. Un schéma de secret partagé (t, n) est un ensemble de deux fonctions, la fonction de partage P qui prend un secret s comme entrée et crée n parts du secret : $P(s) = (s_1, \dots, s_n)$ et la fonction R de reconstruction du secret qui retourne le secret s en rassemblant les t parts ou rien. Le schéma du secret partagé (t, s) est dit additivement homomorphe si $R(s_1 + s'_1, \dots, s_t + s'_t) = s + s'$.

T. Pedersen et al. [90] montre que les schémas de secret partagé additif sont plus efficaces en coût de communication et de calcul par rapport aux crypto systèmes homomorphes lorsqu'ils sont appliqués à la préservation de privacy dans le datamining.

VIII.2 Les parts aléatoires :

Les parts aléatoires [46] sont définies comme suit : Alice et Bob ont des parts aléatoires de la valeur x , dans le corps fini F de taille N . Alice sait une valeur $a \in F$ et Bob connaît la valeur de $b \in F$ de telle sorte : $(a + b) \bmod N = x$ avec a et b sont aléatoires uniformément dans l'ensemble F . À noter que le besoin que $x \in F$ implique que $(a + b) \bmod N$ est actuellement égal à x et non pas seulement $(a + b)$ et x sont congruents modulo N .

VIII.3 La somme sécurisée :

Le problème de la somme sécurisée [91] est simple mais extrêmement utile dans plusieurs applications. Les algorithmes du datamining distribué calculent fréquemment la somme des valeurs des données des sites et l'emploient ainsi en tant que primitif fondamental. Le problème est défini comme suit :

On suppose k parties, P_1, \dots, P_k . La partie P_i a une valeur x_i . Ensemble ils veulent calculer la somme $S = \sum_{i=1}^k x_i$ d'une façon sécurisée. Il est considéré aussi la somme S est un nombre dans le corps F . Si on assume trois parties, le protocole suivant calcule une telle somme :

- P_1 génère un nombre aléatoire r à partir d'une distribution aléatoire uniforme sur le corps F .
- P_1 calcule $S_1 = x_1 + r \bmod |F|$ et l'envoie à P_2
- Pour les parties P_2, \dots, P_{k-1} :
 - P_i reçoit $S_{i-1} = r + \sum_{j=1}^{i-1} x_j \bmod |F|$
 - P_i calcule $S_i = S_{i-1} + x_i \bmod |F| = r + \sum_{j=1}^i x_j \bmod |F|$ et l'envoie au site P_{i+1} .
- P_k reçoit $S_{k-1} = r + \sum_{j=1}^{k-1} x_j \bmod |F|$

- P_k calcule $S_k = S_{k-1} + x_i \bmod |F| = r + \sum_{j=1}^k x_j \bmod |F|$ et l'envoie au site P_l .
- P_l calcule $S = S_k - r \bmod |F| = \sum_{j=1}^k x_j \bmod |F|$ et l'envoie à toutes les parties.

En suivant ces étapes les parties participantes au calcul peuvent avoir le résultat sans révéler leurs données. Cependant le coût de cette primitive de sécurité est considérable [91].

L'application de ces primitives de sécurité dans l'algorithme k-means, permet d'avoir un protocole de préservation de privacy qui est démontrée sécurisé dans le modèle semi-honnête. Néanmoins, les opérations de calcul dans l'algorithme k-means ainsi que ses itérations rendent la tâche de protection des items lors de son exécution loin d'être totalement accomplie. Ceci va être étudié et analysé dans le chapitre suivant.

IX. Conclusion :

Nous avons présenté dans ce chapitre le modèle de sécurité adopté pour la préservation de privacy dans l'algorithme de clustering k-means, qui est le modèle semi-honnête, c'est le modèle de sécurité le plus simple et le plus faisable, nous avons présenté aussi les outils de sécurité les plus utilisés pour réaliser la sécurité du calcul dans ce modèle, tout en montrant la motivation derrière l'utilisation d'une telle ou telle technique de sécurité. Nous nous intéressons dans le chapitre suivant à la façon dont été appliquée ces outils de sécurité dans l'algorithme k-means suivant les différentes approches et études algorithmiques réalisées pour la préservation de privacy dans l'algorithme k-means. Et à quel degré fournissent-ils la protection des items lors de l'exécution de l'algorithme k-means.