TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

LISTE DES ALGORITHMES ET PROTOCOLES

Page

INTRODUCTION

Les dernières années ont vu se démocratiser les téléphones intelligents, dont les possibilités et les applications sont de plus en plus nombreuses et variées. En plus de fournir les services d'un téléphone classique, cette nouvelle génération de téléphones offre beaucoup plus de services, tels que la géolocalisation, la reconnaissance vocale ou encore l'accès internet. Il est désormais également possible d'ajouter des applications additionnelles pour couvrir tous les besoins. De nombreuses applications sont désormais basées sur la localisation de l'usager : la cartographie, la réalité augmentée ou les outils de rencontre. Cependant, ces applications sont pour l'instant limitées aux contextes où les usagers n'ont pas de motivation à falsifier leur position. Plus récemment, il est apparu évident que l'utilisation actuelle des téléphones peut poser de graves problèmes d'atteinte à la vie privée, tels que le traçage des usagers et le recueil de données personnelles.

L'objectif de ce mémoire est de permettre à un individu, que l'on nommera le *prouveur*, d'obtenir des preuves de sa localisation, authentifiées par les utilisateurs environnants, que l'on appellera *témoins*. Ces preuves seront bien sûr datées dans le temps. Il s'agira donc de preuves de localisation à un instant précis. Ces preuves seront ainsi générées par des témoins qui ne devraient pas avoir de motivation à mentir, contrairement au prouveur qui pourrait obtenir certains avantages à falsifier sa preuve de localisation.

Les preuves de localisation permettront à terme de développer une nouvelle génération d'applications. Dans cette nouvelle vague d'applications, les utilisateurs auront une réelle motivation de falsifier leur position. L'article de Saroiu et Wolman (2009) propose quelques applications possibles : (1) le vote électronique où un électeur doit pouvoir prouver qu'il est résident d'une région depuis un certain nombre d'années, (2) le contrôle d'accès basé sur l'emplacement géographique (un utilisateur aurait accès à un service seulement s'il peut prouver qu'il se situe à un emplacement spécifique), (3) le contrôle de présence des étudiants à des activités académiques ou encore (4) lors de la cueillette d'information pour les enquêtes policières. C'est sur ce dernier exemple que nous nous concentrons particulièrement, bien que les autres problèmes soient équivalents.

Supposons un crime ou un délit pour lequel le prouveur souhaite démontrer qu'il n'est pas impliqué. Muni d'un certain nombre de preuves signées par autant de témoins, le prouveur serait capable de convaincre un juge qu'il ne se situait effectivement pas sur les lieux du crime. On peut aussi imaginer mettre en place ce genre de système sur les véhicules afin de prouver qu'un véhicule n'est pas impliqué dans un accident de la route ou un délit routier.

Comme nous le disions plus tôt, les fonctionnalités d'un téléphone intelligent peuvent porter atteinte à la vie privée des usagers et cette application de localisation ne fait pas exception. Il faudra donc se poser plusieurs questions concernant la vie privée et l'anonymat des divers intervenants. Il est évident que le prouveur ne souhaite pas communiquer publiquement sa position précise à chaque fois qu'il demande d'obtenir une preuve de sa localisation (**Propriété I**). Il en va de même pour les témoins, qui préféreront très probablement garder leur identité et leur position précise secrètes (**Propriété II**). Toutes ces données doivent donc être considérées comme des données privées devant être protégées.

Malgré cela, il est nécessaire qu'un juge puisse tout de même récupérer les informations confidentielles des divers participants (**Propriété III**). Ce dernier devant être en mesure de déceler des tentatives de falsification de preuves et des complicités parmi les témoins. En effet, si le prouveur possède suffisamment de témoins complices, il est en mesure de générer de fausses preuves basées sur la localisation de son choix. Le juge doit donc être en mesure de déterminer l'identité des témoins.

Plusieurs approches (par exemple, Saroiu et Wolman (2009) et Luo et Hengartner (2010)) parviennent à répondre à notre problème de preuve de localisation au moyen de points d'accès publics disposés à des endroits stratégiques. Ces points d'accès servent de témoins et de signataires de preuves. Dans un tel cas, la propriété II n'est pas nécessaire puisque les témoins sont remplacés par des points d'accès publics. Une infrastructure complète est donc nécessaire avant que de telles solutions soient envisageables. D'autres approches (par exemple, Gambs *et al.* (2014)) nécessitent que le prouveur diffuse sa position pour obtenir des preuves, violant ainsi la propriété I. L'identité du prouveur est cependant protégée par un système de signature

de groupe. Toutefois, dans ce cas, les témoins ne sont plus des détecteurs impersonnels, mais des participants comme le prouveur désirant que la propriété II soit préservée.

L'objectif de ce mémoire est donc de concevoir des protocoles de calcul multi-parties permettant à un prouveur d'obtenir des preuves de localisation signées par des témoins, tout en respectant la vie privée des participants et en offrant au seul juge la possibilité de connaître ce qui lui est nécessaire.

CHAPITRE 1

PROBLÉMATIQUE

Dans ce court chapitre, nous établissons un modèle de résolution global qui répond au problème posé. Nous pourrons ainsi faire apparaître les différents sous-problèmes qu'il faudra résoudre.

Nous supposerons tout au long de ce mémoire que les téléphones des témoins possèdent une antenne directionnelle qui leur permet de localiser un individu dans le plan. Ainsi, ces antennes directionnelles seront réglées de manière à localiser un individu dans un des quatre quarts de plan possibles, que l'on pourra visualiser comme les quarts de plan nord-est, sud-est, sud-ouest et nord-ouest. Dans l'exemple de la figure 1.1, l'antenne permettra à Alice de localiser Bob dans le quart de plan nord-est.

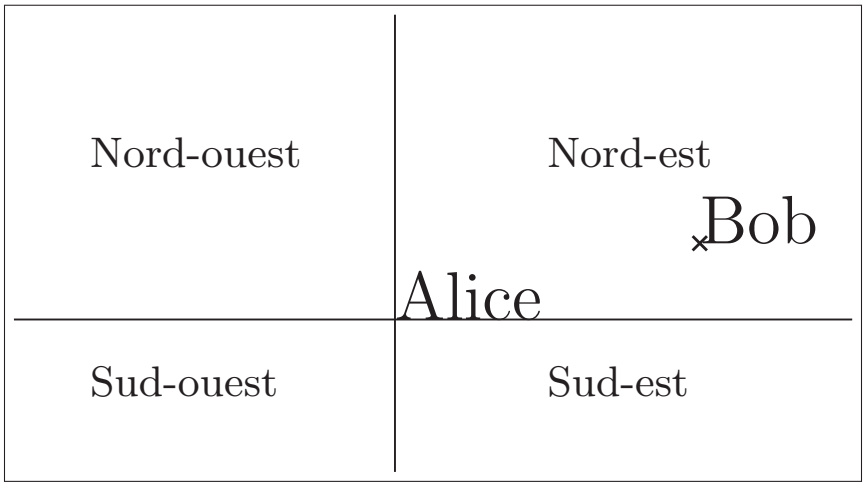

Figure 1.1 Détection avec antenne directionnelle

Cependant, pour permettre ce genre de localisation, Bob doit envoyer un signal ou un message directement à Alice. Cette dernière n'est donc pas en mesure de localiser Bob s'il ne communique pas avec elle.

Avec leur antenne directionnelle, un ensemble de témoins est ainsi capable de déterminer dans quelle portion du plan se situe le prouveur. Cette région correspond à l'intersection des quarts de plan obtenus par chacun des témoins. On obtient la situation représentée par la figure 1.2.

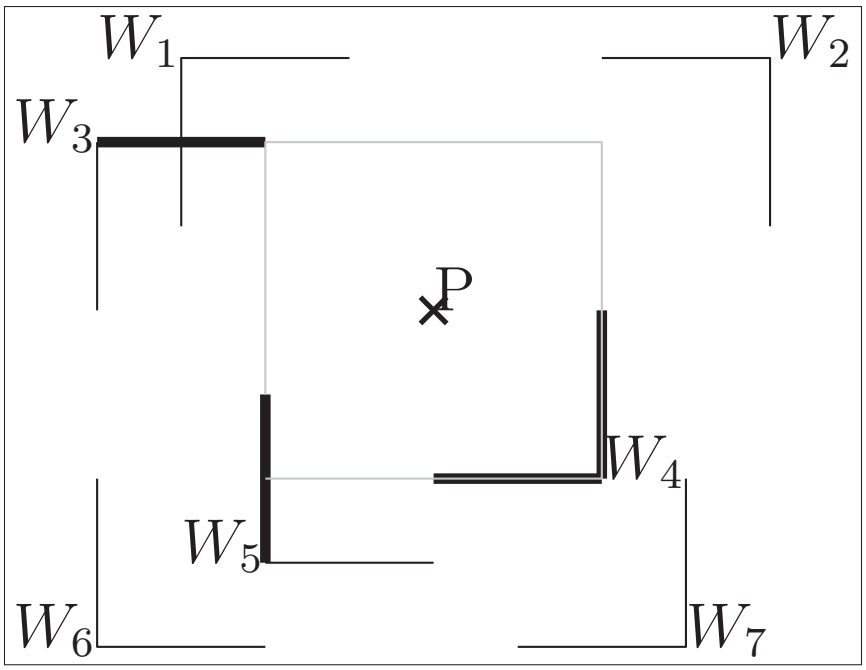

Figure 1.2 Situation initiale

L'objectif de ce mémoire est de permettre aux témoins de calculer une approximation de la position de P correspondant à l'intersection des quarts de plan (symbolisée par le rectangle gris), puis de permettre à P d'obtenir des preuves de cette approximation qui puissent convaincre le juge.

Nous supposons qu'il existe un repère cartésien public à deux dimensions et que chaque témoin W_i connaît sa position (x_i, y_i) dans celui-ci. Cette position peut être précisément déterminée par projection de la latitude et longitude obtenues par un système de radionavigation (GPS ou Galileo).

D'après la situation initiale, pour localiser P, il est nécessaire que les témoins à l'ouest de P (c.-à-d. ceux qui localisent P dans les quarts nord-est ou sud-est) déterminent lequel d'entre eux est

le plus proche de P du côté ouest. Cela revient donc à chercher le témoin W_i à l'ouest de P qui possède la plus grande valeur des x_i. De cette manière, on parvient à déterminer le côté gauche du rectangle gris représenté dans la figure 1.2. Pour cela, nous utiliserons un protocole de calcul multi-partie permettant d'obtenir le maximum de $\{x_i | W_i$ est à l'ouest de $P\}$. L'implémentation de ce protocole sera détaillée dans les chapitres 3 et 4.

On répétera ces opérations quatre fois pour déterminer chaque côté du rectangle en faisant intervenir ceux qui sont à l'ouest de P, puis ceux qui sont à l'est, au nord et enfin au sud. Remarquons que pour ceux qui sont au nord et à l'est, il faudra déterminer la valeur minimale de $\{x_i | W_i$ est à l'est de $P\}$ ou $\{y_i | W_i$ est au nord de $P\}$ parmi les témoins. Chaque témoin participera donc à deux de ces opérations, puisque chaque témoin aide à déterminer deux côtés du rectangle. Par ailleurs, celles-ci peuvent être faites simultanément.

À la fin de ces opérations, P souhaite obtenir les coordonnées du rectangle chiffrées avec la clé du juge (noté T). Il doit alors les faire signer par les témoins, puis conserver les preuves jusqu'à ce que le juge les lui demande.

Pour signer les preuves, les témoins possèdent chacun une clé de groupe qui leur a été délivrée par le juge afin de protéger leur identité. Seul le juge est capable de retrouver l'identité d'un signataire. Le protocole 1.1 présente l'approche générale que nous proposons dans ce mémoire pour générer des preuves de localisation.

Notons que le prouveur est lui aussi équipé d'une antenne directionnelle, il est donc capable de savoir quels sont les témoins à l'est, au nord ... Il n'a pas besoin de demander cette information aux témoins. L'étape 1 du protocole est donc triviale.

Étant donné que chaque témoin est équipé d'une antenne directionnelle, il n'est pas envisageable que ceux-ci communiquent directement entre eux. En effet, cela leur permettrait de se localiser entre eux, ce qui leur donnerait un indice sur la position de leur interlocuteur et sur celle du prouveur. De plus, il n'y aurait alors plus besoin de protocole de calcul de maximum.

Données : Les positions (x_i, y_i) de chaque témoin. Le quart de plan dans lequel se situe P par rapport à chaque témoin.

Résultat : P obtient une preuve par témoin.

Étape 1 : P détermine quels sont les témoins à l'ouest, à l'est, au nord et au sud de sa position.

Étape 2 : L'ensemble des témoins W_i exécutent l'algorithme suivant :

> Les témoins à l'ouest réalisent un protocole de calcul de maximum. P obtient $E_T(x_{max})$.
>
> Les témoins au sud réalisent un protocole de calcul de maximum. P obtient $E_T(y_{max})$.
>
> Les témoins à l'est réalisent un protocole de calcul de minimum. P obtient $E_T(x_{min})$.
>
> Les témoins au nord réalisent un protocole de calcul de minimum. P obtient $E_T(y_{min})$.

Étape 3 : P transfert les résultats obtenus toujours chiffrés à l'ensemble des témoins.

Étape 4 : Chaque témoin fabrique une preuve avec sa clé de groupe et envoie cette preuve à P.

Étape 5 : Sur demande du juge T, P peut fournir les preuves.

Protocole 1.1 : Preuves de localisation

L'implémentation du protocole de calcul de maximum se révèle donc être la principale difficulté de cette approche. C'est indéniablement cette étape 2 qui engendrera le plus de calculs et de communications entre les participants. Plutôt que d'utiliser un protocole multi-partie de calcul de maximum classique qui requiert $n^2 - n$ comparaisons, nous mettrons en place un protocole de comparaisons à n participants nécessitant $O(n \log n)$ comparaisons.

Pour l'ensemble des protocoles proposés, nous nous intéresserons principalement au modèle semi-honnête. Dans ce modèle, les participants suivent exactement le protocole sans y apporter aucune modification. Pour les protocoles de calculs multi-parties, ce modèle suppose aussi que les participants utilisent leurs vrais paramètres privés. Le modèle semi-honnête n'influe donc pas sur le résultat. Toutefois, les participants essaieront de déduire le plus d'informations pos-

sible à partir de ce que les protocoles leur permettent de connaître. Ils chercheront à découvrir les paramètres privés des autres participants. Dans certains cas, ils chercheront aussi à découvrir le résultat privé d'un protocole. Nous verrons en plus le modèle semi-honnête augmenté, dans lequel un adversaire peut falsifier ses paramètres privés.

Nous étudierons aussi toutes les formes de collusion possibles. Ces collusions sont statiques, ce qui signifie qu'elles se mettent en place avant le protocole.

En comparaison, dans le modèle malhonnête, un adversaire peut modifier le protocole ainsi que ces paramètres privés comme il le désire. Il cherchera à découvrir les paramètres privés des autres participants par tous les moyens possibles. Afin d'améliorer l'efficacité de nos protocoles, ce modèle d'adversaire ne sera pas considéré par la suite.

Les motivations des attaquants sont les suivantes :

- Les témoins et le prouveur voudront déterminer la position et l'identité des autres témoins ou du prouveur.
- Le prouveur essaiera de forger de fausses preuves.

Le chapitre deux de ce mémoire sera dédié à la revue de littérature sur ce sujet. Nous y verrons tous les outils cryptographiques qui permettent d'y répondre. Dans les chapitres trois et quatre, nous détaillerons deux approches alternatives au problème du calcul du maximum. Enfin, dans le chapitre cinq, nous verrons comment fabriquer des preuves de localisation qui peuvent convaincre le juge.

CHAPITRE 2

REVUE DE LITTÉRATURE

Ce chapitre est dédié à la revue de littérature. Les différents outils cryptographiques qui sont utilisés dans la suite du mémoire y sont présentés. Les sujets abordés sont la sécurité sémantique, les chiffrements homomorphes, les transferts inconscients et les calculs multi-parties (principalement les protocoles de comparaison et de calcul de maximum).

Dans le reste du mémoire, la notion d'aléatoire implique l'uniformité.

2.1 Notions de sécurité sémantique

Les notions de sécurité sémantiques définies ici sont extraites de Goldreich (2004). Trois niveaux de sécurité sont étudiés : l'indiscernabilité avec des textes clairs choisis (*indistinguishability under chosen-plaintext attack* IND-CPA) et les deux variantes de l'indiscernabilité avec des textes chiffrés choisis (*indistinguishability under chosen-cyphertext attack* IND-CCA1 et IND-CCA2).

Le niveau de sécurité IND-CPA est défini de la manière suivante : il est difficile pour un adversaire d'extraire des informations à partir de messages chiffrés, même en connaissant les chiffrements de messages qu'il a lui-même choisis. En d'autres termes, l'adversaire demande le chiffrement d'autant de messages qu'il désire dans une première phase, puis, dans une seconde phase, choisit arbitrairement deux messages m_0 et m_1. Il reçoit ensuite le chiffrement $E(m_x)$ d'un seul de ces deux messages choisi aléatoirement et doit deviner lequel des deux messages a été chiffré. Le niveau de sécurité IND-CPA est atteint si et seulement si la probabilité de succès de l'adversaire, dont la puissance de calcul est bornée polynomialement, est proche de $1/2$.

Dans le niveau de sécurité IND-CCA1, l'adversaire est autorisé à demander le déchiffrement d'autant de cryptogrammes qu'il le désire pendant la première phase. Dans le niveau de sécurité

IND-CCA2, il peut demander des déchiffrements de cryptogrammes même au cours de la seconde phase, après avoir reçu $E(m_x)$. Les chiffrés doivent toutefois évidemment être différents du défi. Il s'agit du niveau de sécurité le plus élevé.

Pour respecter la sécurité sémantique, un système de chiffrement à clé publique doit être probabiliste (condition nécessaire). Un système de chiffrement est dit probabiliste si pour un message donné, il existe plusieurs chiffrements possibles de ce message. Pour un système de chiffrement non-probabiliste, il suffit en effet à l'adversaire de chiffrer lui-même m_0 et m_1 avec la clé publique s'il s'agit de cryptographie asymétrique, ou qu'il ait déjà demandé le chiffrement de m_0 ou m_1 s'il s'agit de cryptographie symétrique. L'un des deux cryptogrammes obtenus $E(m_0)$ ou $E(m_1)$ sera nécessairement identique à $E(m_x)$. Un système de chiffrement non-probabiliste n'atteint donc pas le niveau IND-CPA.

2.2 Chiffrement homomorphe

Le chiffrement homomorphe est un type particulier de chiffrement qui permet de réaliser certaines opérations (telles que l'addition ou la multiplication) sur des données chiffrées, puis de retrouver le résultat de ces opérations après le déchiffrement. La plupart des systèmes de chiffrement ne permettent de réaliser qu'une seule opération : soit l'addition [Paillier (1999)] ou soit la multiplication [Rivest *et al.* (1978)]. Plus récemment, Gentry *et al.* (2009) a proposé un système de chiffrement homomorphe complet (*fully homomorphic encryption*) permettant ces deux opérations sans aucune contrainte. Toutefois, celui-ci est peu performant et de nombreuses variantes sont apparues depuis (par exemple, Gentry et Halevi (2011) et Coron *et al.* (2011)). Malheureusement, ces variantes ne sont pas encore suffisamment efficaces pour développer des solutions simples. Les systèmes de chiffrement homomorphe complet ne seront pas étudiés dans ce mémoire.

Par définition, un système de chiffrement homomorphe ne peut pas atteindre le niveau de sécurité IND-CCA2. En effet, pour un chiffrement homomorphe additif, l'adversaire peut demander

le déchiffrement de $E(m_x - m_0)$ (respectivement $E(m_x \cdot {m_0}^{-1}$ pour un système multiplicatif). Si le résultat est 0 (respectivement 1), alors $m_x = m_0$, sinon $m_x = m_1$.

Les travaux de Fontaine et Galand (2007) prouvent d'ailleurs que le niveau de sécurité maximal que peut atteindre un système de chiffrement homomorphe est IND-CPA.

2.2.1 Chiffrement RSA

Le chiffrement RSA [Rivest *et al.* (1978)] est l'un des exemples les plus simples et les plus courants de système de chiffrement homomorphe multiplicatif. Ce chiffrement est défini comme suit :

- Soit $p, q \in \mathbb{Z}^*$ deux grands nombres premiers privés
- Soit $N = pq$ le produit rendu public
- Soit l'exposant public $e \in \mathbb{Z}_N^*$ tel que $pgcd(e, \phi(N)) = 1$ où $\phi(N) = (p-1)(q-1)$ est l'indicatrice d'Euler.
- Soit l'exposant privé $d \in \mathbb{Z}_N^*$ tel que $ed \equiv 1 \pmod{\phi(N)}$. Cet exposant est donc l'inverse de e modulo $\phi(N)$ et peut être calculé grâce à l'algorithme d'Euclide étendu.

La fonction de chiffrement $E_{N,e}(\cdot)$ d'un message $m \in \mathbb{Z}_n^*$ est définie de la façon suivante :

$$E_{N,e} : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$$
$$E_{N,e}(m) = m^e \mod N$$

La fonction de déchiffrement $D_{N,d}(\cdot)$ est définie comme suit :

$$D_{N,d} : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$$
$$D_{N,d}(c) = c^d \mod N$$

Ce chiffrement permet de réaliser des multiplications sur des données chiffrées. En effet, il est simple de vérifier que :

$$E(m_1) * E(m_2) \equiv (m_1 * m_2)^e \equiv E(m_1 * m_2) \pmod{N}$$

Cependant, il n'existe aucune façon connue de réaliser d'autres opérations, telles que l'addition, avec ce système.

Le système de chiffrement RSA dans sa version originale n'est pas probabiliste. Le niveau de sécurité sémantique IND-CPA ne peut donc pas être atteint.

2.2.2 Système de Paillier

Le système de Paillier (1999) est un autre exemple de système de chiffrement homomorphe, mais additif. Ce chiffrement est défini comme suit :

- Soit $p, q \in \mathbb{Z}^*$ deux grands nombres premiers privés.
- Soit $N = pq$ le produit rendu public.
- Soit $\phi(N)$ l'indicatrice d'Euler privée.

La fonction de chiffrement $E_N(\cdot)$ d'un message $m \in \mathbb{Z}_N$ est définie comme suit :

$$E_N : \mathbb{Z}_N \times \mathbb{Z}_N^* \to \mathbb{Z}_{N^2}^*$$
$$r \in \mathbb{Z}_N^* \text{ est un entier aléatoire}$$
$$E_N(m, r) = (1+N)^m \cdot r^N \mod N^2$$

Cette version est un cas particulier largement adopté du système de Paillier. La fonction de chiffrement pourrait être généralisée comme suit :

$$E_{n,g}(m, r) = g^m \cdot r^N \mod N^2$$
$$\text{où } g \text{ est un générateur public de } \mathbb{Z}_{N^2}^*$$

En utilisant le théorème d'Euler (pour tout a et N copremiers, $a^{\phi(N)} \equiv 1 \pmod{N}$) et grâce à la formule de Newton : $(\forall m, N \in \mathbb{Z}, (1+N)^m \equiv 1+mN \pmod{N^2})$, on peut définir la fonction de déchiffrement $D_{\phi(N)}$ comme suit :

$$
\begin{aligned}
D_{\phi(N)} &: \mathbb{Z}^*_{N^2} \to \mathbb{Z}_N \\
D_{\phi(N)}(c) &= \frac{(c^{\phi(N)} \mod N^2) - 1}{N} \cdot \phi(N)^{-1} \mod N \\
&= \frac{((1+N)^{m\phi(N)} \cdot r^{N\phi(N)} \mod N^2) - 1}{N} \cdot \phi(N)^{-1} \mod N \\
&= \frac{((1+Nm\phi(N)) \mod N^2) - 1}{N} \cdot \phi(N)^{-1} \mod N \\
&= \frac{Nm\phi(N)}{N} \cdot \phi(N)^{-1} \mod N \\
&= m
\end{aligned}
$$

Le système de Paillier est probabiliste. En effet, l'aléa r permet d'obtenir plusieurs chiffrements différents d'un même message. Il est facile de prouver que ce système atteint le niveau de sécurité sémantique IND-CPA. Soit m_0 et m_1 deux messages connus et c le chiffrement d'un de ces messages. Si c est le chiffrement de m_0, alors $c \cdot (1+N)^{-m_0} \equiv r^N \pmod{N^2}$ qui est un N-résidu modulo N^2. Par conséquent, un attaquant capable de résoudre ce défi en un temps polynomial est aussi capable de résoudre le problème du *decisional composite residuosity assumption* (DCRA) en un temps polynomial. Rappelons l'hypothèse de DCRA : connaissant un entier composé N et un entier z, il est difficile de savoir s'il existe un entier y tel que $z \equiv y^N \pmod{N^2}$.

Notons que lors de la phase de déchiffrement, il est aussi possible de calculer l'aléa r utilisé dans le cryptogramme c grâce à l'équation suivante :

$$r = c^{N^{-1} \mod \phi(N)} \mod N$$

Le système de Paillier permet de réaliser des additions sur des opérandes chiffrés. On parle d'homomorphisme additif. Il est possible de vérifier simplement cette propriété :

$$\begin{aligned} E(m_1, r_1) * E(m_2, r_2) &\equiv (1+N)^{m_1} \cdot {r_1}^N * (1+N)^{m_2} \cdot {r_2}^N \\ &\equiv (1+N)^{m_1+m_2} \cdot r_1 {r_2}^N \pmod{N^2} \\ &= E(m_1 + m_2, r_1 r_2) \end{aligned}$$

Remarquons que dans ce cas, il est possible de se passer du chiffrement du second opérande :

$$\begin{aligned} E(m_1, r_1) * (1+N)^{m_2} &\equiv (1+N)^{m_1} \cdot {r_1}^N * (1+N)^{m_2} \\ &\equiv (1+N)^{m_1+m_2} \cdot {r_1}^N \pmod{N^2} \\ &= E(m_1 + m_2, r_1) \end{aligned}$$

En appliquant les règles ci-dessous, il est possible de réaliser des multiplications à condition qu'un des opérandes ne soit pas chiffré :

$$\begin{aligned} E(m_1, r_1)^{m_2} &\equiv (1+N)^{m_1 m_2} \cdot {r_1}^{m_2 N} \pmod{N^2} \\ &= E(m_1 \cdot m_2, r_1^{m_2}) \end{aligned}$$

Il n'existe en revanche pas de façon connue de réaliser une multiplication sur deux nombres chiffrés.

Le système de Paillier permet aussi de soustraire un nombre connu à une valeur chiffrée de la manière suivante. Sachant que $\forall k \in \mathbb{Z}^*, (1+k)^k \equiv 1 \pmod{k^2}$ grâce à la formule de Newton,

il est possible de vérifier que :

$$\begin{aligned} E(m_1,r) * E(N-m_2,1) &\equiv (1+N)^{m_1+N-m_2} \cdot r^N \\ &\equiv (1+N)^N \cdot (1+N)^{m_1-m_2} \cdot r^N \\ &\equiv (1+N)^{m_1-m_2} \cdot r^N \pmod{N^2} \\ &= E(m_1-m_2,r) \end{aligned}$$

Il en va de même pour multiplier par un nombre négatif. Il est possible de vérifier que :

$$\begin{aligned} E(m_1,r)^{N-m_2} &\equiv (1+N)^{m_1(N-m_2)} \cdot r^{N(N-m_2)} \\ &\equiv (1+N)^{Nm_1} \cdot (1+N)^{-m_1m_2} \cdot r^{N(N-m_2)} \\ &\equiv (1+N)^{-m_1m_2} \cdot r^{N(N-m_2)} \pmod{N^2} \\ &= E(-m_1m_2, r^{N-m_2}) \end{aligned}$$

Ainsi, pour réaliser une soustraction entre deux opérandes chiffrés, il suffit donc de multiplier l'un des deux par -1, puis de les additionner.

2.3 Transfert inconscient

Dans le transfert inconscient 1-parmi-2 (*1-out-of-2 oblivious tranfert*), une des parties, Alice, possède deux secrets M_0 et M_1. L'autre partie, Bob, souhaite récupérer une et une seule de ces données M_i de son choix en s'assurant qu'Alice ignore la valeur de i. De son côté, Alice s'assure que Bob ne puisse pas connaître plus d'un seul secret. Les données d'Alice sont donc protégées tout autant que le choix de Bob.

Une des approches les plus simples pour le transfert inconscient 1-parmi-2 a été donnée par Even *et al.* (1985) et peut être résumée par les quelques étapes décrites dans le protocole 2.1. On supposera qu'Alice possède deux secrets M_0 et M_1 et des fonctions de chiffrement et déchiffrement $E_A(\cdot)$ et $D_A(\cdot)$ respectivement. Les secrets et les clés de chiffrement doivent être de la même longueur. L'opérateur $\oplus$ dénote l'addition bit à bit modulo 2.

Données : Deux secrets M_0 et M_1 de Alice, sa fonction de chiffrement publique $E_A(\cdot)$ et sa fonction de déchiffrement privée $D_A(\cdot)$.

Résultat : Bob apprend M_r de son choix.

Étape 1 : Alice choisit aléatoirement deux messages m_0 et m_1 (à ne pas confondre avec les secrets M_0 et M_1), puis les envoie à Bob.

Étape 2 : Bob choisit $r \in \{0,1\}$ et une valeur k aléatoire. Il calcule ensuite $q = E_A(k) \oplus m_r$ et le renvoie à Alice.

Étape 3 : Alice détermine $k'_0 = D_A(q \oplus m_0)$ et $k'_1 = D_A(q \oplus m_1)$. Ces deux alternatives sont indistinguables. Elle transmet $(M_0 \oplus k'_0, M_1 \oplus k'_1)$ à Bob.

Étape 4 : Bob détermine $M_r = (M_r \oplus k'_r) \oplus k$.

Protocole 2.1 : Protocole de transfert inconscient tiré de Even *et al.* (1985)

Remarquons que $k'_r = D_A(q \oplus m_r) = D_A(E_A(k) \oplus m_r \oplus m_r) = k$. De ce fait, puisque Bob a choisi r, il peut sélectionner le bon élément à l'étape 4. Donc $k'_r = k$, ce qui implique que $M_r \oplus k'_r \oplus k = M_r$, le secret convoité par Bob. Dans cet exemple, Bob est donc capable de récupérer un des deux secrets de son choix.

Le problème du transfert inconscient 1-parmi-n (*1-out-of-n oblivious transfer*) généralise celui du transfert inconscient 1-parmi-2 : on considère ici qu'Alice possède n secrets $m_1, m_2, \ldots, m_n$. Le problème du transfert inconscient 1-parmi-n a lui aussi a été largement étudié par un grand nombre de chercheurs depuis sa définition initiale (par exemple, Brassard *et al.* (1987) et Tzeng (2004)) et est l'un des outils les plus importants et les plus utilisés pour les calculs multi-parties. La solution de Tzeng (2004) est présentée dans le protocole 2.2 et s'appuie sur le système de chiffrement de ElGamal (1985).

L'exactitude de ce protocole peut être facilement démontrée de la manière suivante :

$$\frac{b}{a^r} = \frac{m_\alpha (y/h^\alpha)^{k_\alpha}}{g^{k_\alpha r}} = \frac{m_\alpha (g^r h^\alpha / h^\alpha)^{k_\alpha}}{g^{k_\alpha r}} = m_\alpha$$

Données : Les secrets $m_1, ..., m_n$ de Alice. Le choix α de Bob $(1 \leq \alpha \leq n)$.

Résultat : Bob apprend m_α de son choix.

Étape 1 : Alice choisit un grand nombre premier q, g et h deux générateurs du groupe G_q d'ordre q. Alice envoie ces paramètres à Bob.

Étape 2 : Bob vérifie que q est premier. Il choisit aléatoirement $r \in \mathbb{Z}_q$, calcule $y = g^r h^\alpha$ et envoie y à Alice.

Étape 3 : Alice exécute l'algorithme suivant :

> **pour** *i=1* **à** *n* **faire**
>
> Alice choisit aléatoirement $k_i \in \mathbb{Z}_q$.
>
> Alice calcule $c_i = (g^{k_i}, m_i(y/h^i)^{k_i})$.
>
> **fin**
>
> Alice envoie l'ensemble des c_i à Bob.

Étape 4 : On note $c_\alpha = (a, b)$. Bob calcule $m_\alpha = b/a^r$.

Protocole 2.2 : Protocole de transfert inconscient tiré de Tzeng (2004)

Afin de démontrer la sécurité du protocole, il faut montrer que (1) le choix de Bob et (2) les secrets d'Alice ne sont pas divulgués.

Il est simple de remarquer que pour tout choix potentiel $\bar{\alpha}$, il existe un élément $\bar{r}$ tel que $y = g^{\bar{r}} h^{\bar{\alpha}}$. Donc le choix de Bob est inconditionnellement protégé.

D'autre part, Tzeng (2004) a pu démontrer que si le problème de *Decisional Diffie-Hellman* (DDH) est difficile, le protocole protège les secrets d'Alice contre un attaquant semi-honnête. Signalons que l'auteur a aussi proposé un second protocole toujours sécuritaire même contre un attaquant malicieux ne suivant pas forcément le protocole.

Seulement deux communications sont donc nécessaires dans ce protocole. Par ailleurs, Alice doit réaliser $2n$ exponentiations modulaires, contre seulement 2 pour Bob.

2.4 Calculs multi-parties

Les calculs multi-parties sécuritaires sont une branche de la cryptographie permettant à plusieurs parties de calculer coopérativement une fonction publique sans dévoiler les paramètres privés. Le premier protocole à s'inscrire dans cette branche est celui de Yao (1982) qui cherche à résoudre le problème des millionnaires permettant à deux individus de comparer leur fortune respective. Plus tard, les travaux de Goldreich (1998) et de Goldwasser (1997) ont motivé les chercheurs à travailler sur des problèmes multi-parties précis plutôt que sur des problèmes généraux afin d'en améliorer l'efficacité, le but étant de développer des protocoles performants.

Les notions de sécurité d'un protocole de calcul multi-partie ci-dessous sont extraites du livre de Hazay et Lindell (2010). Soit deux participants Alice et Bob dont les valeurs privées respectives sont notées a et b. Alice désire calculer $f_1(a,b)$ et Bob souhaite déterminer $f_2(a,b)$. Un protocole de calcul multi-partie peut être noté sous la forme de la fonction suivante

$$f : (a,b) \mapsto (f_1(a,b), f_2(a,b))$$

Un protocole est dit sécurisé si ce qui peut être calculé par une partie grâce au protocole peut aussi être calculé à partir des valeurs privées et du résultat de cette partie seulement. Ainsi, la valeur b de Bob est protégée s'il existe un simulateur qui puisse générer équiprobablement le même résultat $f_1(a,b)$ quelle que soit la valeur de b (tel que le résultat recherché ne soit bien sûr pas modifié). Le protocole ne permet donc pas à un adversaire d'un apprendre plus qu'il ne sait déjà sur la valeur des participants.

2.4.1 Problème des millionnaires de Yao

Deux millionnaires Alice et Bob possédant respectivement a et b millions cherchent à savoir lequel est le plus riche sans révéler à l'autre sa propre fortune. Ce protocole de comparaison est l'un des premiers protocoles de calcul multi-parties. La solution proposée par Yao (1982) a la particularité de ne pas utiliser de chiffrement homomorphe, mais elle est très limitée dans le

choix des valeurs possibles de a et b. On supposera dans le protocole 2.3 que a et b sont des entiers compris entre 1 et 10.

Données : La donnée d'Alice a et celle de Bob b. La clé publique N_A d'Alice et sa fonction de chiffrement $E_A(\cdot)$.

Résultat : Bob détermine si $a \geq b$ ou $a < b$.

Étape 1 : Bob choisit aléatoirement $x \in_R \mathbb{Z}_{N_A}$ et calcule $k = E_A(x)$.

Étape 2 : Bob envoie à Alice $k - b$.

Étape 3 : Alice exécute l'algorithme suivant :

répéter

Alice génère un nombre premier p de longueur $|N_A|/2$ bits.

pour *i=1* **à** *10* **faire**

Alice calcule $y_i = D_A(k - b + i)$ et $z_i = y_i \pmod{p}$.

fin

jusqu'à $\forall i, \forall j \neq i, |z_i - z_j| \geq 2$;

Étape 4 : Alice envoie à Bob $z_1, z_2, ..., z_a, z_{a+1} + 1, ..., z_{10} + 1$ et p.

Étape 5 : Bob lit le b^{eme} élément envoyé par Alice, noté z'_b. S'il est égal à $x \pmod{p}$, Bob sait que $a \geq b$, sinon $a < b$.

Protocole 2.3 : Problème des millionnaires tiré de Yao (1982)

On peut facilement démontrer que cette solution est exacte. Notons que $y_b = D_A(k)$. Si $a \geq b$, on obtient $z'_b = z_b \equiv x \pmod{p}$. Au contraire, si $a < b$, $z'_b = z_b + 1 \not\equiv x \pmod{p}$.

Cependant, la solution cryptographique proposée par Yao est impraticable si le nombre de valeurs possibles n de a et b est beaucoup plus important (par exemple un million). En effet, ce protocole requiert seulement un chiffrement pour Bob, mais n calculs pour Alice. Cette solution ne nécessite que deux communications entre Alice et Bob, mais le nombre de bits échangés lors de la seconde communication est dans $\Omega(n)$.

D'autres protocoles de comparaison plus efficaces ont été développés depuis. La plupart des approches récentes sont basées sur la représentation binaire de a et b, les valeurs d'Alice et Bob. On notera a_j et b_j les $j^{ème}$ bits de poids fort de a et b respectivement. Dans ce cas, $a > b$ si et seulement s'il existe i^* tel que $a_j = b_j$ pour tout $1 \leq j < i^*$ et $a_{i^*} = 1$, $b_{i^*} = 0$. Si un tel i^* n'existe pas, on peut conclure que $a \leq b$.

Blake et Kolesnikov (2004) proposent un protocole de comparaison qui utilise le système de Paillier. La solution proposée est présentée dans le protocole 2.4.

Données : Les valeurs a et b d'Alice et Bob. La clef publique N_A d'Alice avec la fonction de chiffrement associée $E_A(\cdot)$.

Hypothèse : a et b doivent être différents.

Résultat : Alice détermine si $a < b$ ou $a > b$.

Étape 1 : Alice chiffre chaque bit de a sous la forme $E_A(a_1), ..., E_A(a_l)$ et les envoie à Bob.

Étape 2 : Bob exécute l'algorithme suivant pour calculer les vecteurs d, f, γ, δ :

pour $j = 1$ **à** l **faire**

$E_A(d_j) = E_A(a_j - b_j) = E_A(a_j) \cdot E_A(-b_j)$

$E_A(f_j) = E_A(a_j \oplus b_j) = E_A(a_j - 2a_j b_j + b_j) = E_A(a_j) \cdot E_A(a_j)^{-2b_j} \cdot E_A(b_j)$

$E_A(\gamma_j) = E_A(2\gamma_{j-1} + f_j) = E_A(\gamma_{j-1})^2 \cdot E_A(f_j)$ avec $\gamma_0 = 0$

$E_A(\delta_j) = E_A(d_j + r_j(\gamma_j - 1)) = E_A(d_j) \cdot (E_A(\gamma_j) \cdot E_A(-1))^{r_j}$ où $r_j \in_R \mathbb{Z}_{N_A}$.

fin

Étape 3 : Bob permute les éléments $E_A(\delta_j)$ et les envoie à Alice.

Étape 4 : Alice déchiffre les éléments $E_A(\delta_j)$ permutés. Si l'un d'entre eux vaut 1, $a > b$. Sinon, l'un d'entre eux vaut -1 et cela signifie que $a < b$.

Protocole 2.4 : Problème des millionnaires tiré de Blake et Kolesnikov (2004)

L'exactitude de ce protocole peut facilement être démontrée. Soit i^* le premier bit de poids fort tel que $a_{i^*} \neq b_{i^*}$. Pour tout $1 \leq j < i^*$, on a $a_j = b_j$ et donc $f_j = 0$. Par conséquent, γ_j reste nul et $\delta_j = -r_j$ est aléatoire. Pour i^*, on a $\gamma_{i^*} = f_{i^*} = 1$ et donc $\delta_{i^*} = a_{i^*} - b_{i^*}$. δ_{i^*} est égal à 1 si

$a > b$ sinon, il est égal à -1. Par la suite, pour $i^* < j \leq l$, la suite γ_j est strictement croissante. De ce fait, δ_j sera aléatoire. On obtient donc bien le résultat souhaité avec une très grande probabilité. En effet, la probabilité que $\delta_{j \neq i^*} = 1$ ou -1 est de l'ordre de $1/N_A$.

Pour prouver la sécurité du protocole de Blake et Kolesnikov (2004), il faut montrer que (1) Alice ne peut pas découvrir b et que (2) Bob ne peut pas découvrir a. La protection de a est assurée par la sécurité sémantique du système de Paillier. En effet, toutes les données que Bob obtient sont chiffrées avec la clé d'Alice. La sécurité du point de vue de Bob (protection de b) est assurée par l'ajout de la valeur aléatoire r_j et d'une fonction de permutation. On peut démontrer que pour une fonction de permutation donnée, pour toute valeur possible de b, il existe une unique solution $(r_1, r_2, ..., r_l)$ qui puisse générer le même vecteur δ. Toutes les valeurs de b sont donc possibles et équiprobables. La valeur de Bob est donc inconditionnellement protégée, même si Alice est semi-honnête.

Ce protocole requiert uniquement deux communications, mais celles-ci sont assez volumineuses. On note N_A la clé publique utilisée. Chaque bit d'Alice est chiffré dans $\mathbb{Z}_{{N_A}^2}$ puis transmis à Bob. Bob fait des opérations sur chaque bit puis les retransmet à Alice. Il y a donc $2l|{N_A}^2| = 4l|N_A|$ bits échangés.

Le protocole 2.5 de Lin et Tzeng (2005) est très similaire au protocole de Blake et Kolesnikov (2004). L'article original suggère d'utiliser le système de chiffrement d'ElGamal, mais il propose une alternative pour utiliser un chiffrement additif tel que le système de Paillier. C'est cette alternative qui est ici présentée.

Pour une valeur donnée x, il faut introduire deux ensembles T_0^x et T_1^x définis de la manière suivante :

$$T_0^x = \{x_1 x_2 ... x_{i-1} 1 | x_i = 0\}$$
$$T_1^x = \{x_1 x_2 ... x_i | x_i = 1\}$$

On notera $T_\alpha^x[i]$ l'élément de T_α^x de longueur i bits s'il existe.

D'après l'analyse faite par Lin et Tzeng (2005), il est facile de prouver que pour deux valeurs x et y, $x > y$ si et seulement si T_1^x et T_0^y ont un élément commun. Remarquons que T_0^y peut aussi être écrit sous la forme $T_0^y = \{y_1 y_2 ... y_{i-1}\overline{y_i} | y_i = 0\}$. Par conséquent, si T_1^x et T_0^y ont un élément commun, il existe i^* tel que :

$$
\begin{aligned}
T_1^x[i^*] &= T_0^y[i^*] \\
x_1 x_2 ... x_{i^*} &= y_1 y_2 ... \overline{y_{i^*}} && \text{où } x_{i^*} = 1 \text{ et } y_{i^*} = 0 \\
x_1 x_2 ... x_{i^*} &> y_1 y_2 ... y_{i^*} && \text{impliquant que } x > y
\end{aligned}
$$

La réciproque est évidente : si $x > y$ alors il existe i^* tel que $x_1 x_2 ... x_{i^*} = y_1 y_2 ... \overline{y_{i^*}}$

Pour prouver l'exactitude de ce protocole, il suffit d'analyser les deux cas possibles. Si $a > b$, alors T_1^a et T_0^b ont un élément commun et il existe un (et un seul) i^* tel que $T_1^a[i^*] = T_0^b[i^*]$. Par conséquent, il est évident que $\delta_{i^*} = h(T_1^a[i^*]) - h(T_0^b[i^*]) = 0$. Dans le cas contraire pour lequel $a \leq b$, un tel i^* n'existe pas. En supposant que la fonction de hachage n'admet pas de collision, il n'existe pas de i^* tel que $h(T_1^a[i^*]) - h(T_0^b[i^*]) = 0$. Cependant, si $T_1^a[i]$ n'existe pas, son haché est remplacé par une valeur aléatoire et il existe donc une chance infime de choisir $h(T_0^b[i])$. Si $T_0^b[i]$ n'existe pas, δ_i est choisi non nul. En conclusion, si le vecteur δ contient un zéro, il est très probable que $a > b$. Dans le cas contraire, il est très probable que $a \leq b$.

De la même manière que pour le protocole de Blake et Kolesnikov (2004), l'ajout d'une variable aléatoire k_i et l'utilisation de la fonction de permutation rend impossible toute divulgation d'information.

Signalons que le protocole de Fischlin (2001) parvient au même résultat en utilisant les propriétés des résidus quadratiques pour simuler des opérateurs XOR et AND sur les bits des valeurs. Les charges calculatoire et communicationnelle sont cependant rendues beaucoup plus élevées par ce processus. Ce protocole ne sera pas davantage détaillé ici.

Données : Les valeurs a et b d'Alice et Bob. La clef publique N_A d'Alice avec la fonction de chiffrement associée $E_A(\cdot)$. Une fonction de hachage $h(\cdot)$.

Résultat : Alice détermine $a > b$ ou $a \leq b$.

Étape 1 : A exécute l'algorithme suivant :

A détermine l'ensemble T_1^a.

A génère un vecteur γ de longueur l tel que $\gamma_i = h(T_1^a[i])$ si cet élément existe, sinon γ_i est généré aléatoirement.

Les éléments de γ sont chiffrés grâce à $E_A(\cdot)$ puis transférés à B.

Étape 2 : B exécute l'algorithme suivant :

B détermine l'ensemble T_0^b.

B choisit aléatoirement une fonction de permutation $\pi_B(\cdot)$.

B calcule le chiffrement du vecteur δ de longueur l tel que :

$$E_A(\delta_i) = E_A(k_i(h(T_1^a[i]) - h(T_0^b[i]))) = (E_A(\gamma_i) \cdot E_A(-h(T_0^b[i])))^{k_i}$$

où $k_i \in_R \mathbb{Z}_{N_A}$ est généré aléatoirement pour chaque élément.

Si $T_0^b[i]$ n'existe pas, δ_i est choisi aléatoirement non nul.

B envoie $\pi_B(\delta)$ à A.

Étape 3 : A déchiffre les éléments $E_A(\delta_i)$. Le vecteur δ contient la valeur 0 si et seulement si $a > b$.

Protocole 2.5 : Problème des millionnaires tiré de Lin et Tzeng (2005)

Une autre approche a été imaginée par Ioannidis et Grama (2003) et possède la particularité de ne pas utiliser d'algorithme de chiffrement. Celle-ci est présentée dans le protocole 2.6. À l'inverse des autres protocoles, il est beaucoup plus simple de noter a_i le $i^{ème}$ bit de poids faible de a. On définira la fonction $leftrot(x,r)$ comme étant la rotation de x de r bits vers la gauche, bien que le sens n'ait pas d'importance.

L'exactitude de ce protocole peut être prouvée en analysant la forme des données. Notons tout d'abord que $c = S \oplus (\bigoplus_{i=1}^{d} A'_{i,\overline{b_i}}) = leftrot(\bigoplus_{i=1}^{d} A_{i,\overline{b_i}}, r)$. Par conséquent, le résultat dépend uniquement de $\bigoplus_{i=1}^{d} A_{i,\overline{b_i}}$. Les éléments de A peuvent être de deux types différents selon a_i :

Données : La donnée de Alice a et celle et Bob b de longueur l bits. Un paramètre publique k largement plus grand que l.

Résultat : Bob détermine si $a < b$ ou $a \geq b$.

Étape 1 : Alice génère une matrice A de dimension $l \times 2$ dont les éléments ont k bits et sont nuls. Les colonnes de cette matrice seront numérotées 0 et 1. On notera $A_{i,j,x}$ le $x^{ème}$ bit de poids faible de $A_{i,j}$. Alice choisit s aléatoire tel que $2l+1 \ll s \leq k$.

Étape 2 : Alice exécute l'algorithme suivant :

> Les $k-s$ bits de poids fort de tous les éléments de A sont aléatoires.
>
> Pour tout i, $A_{i,a_i,2i} = 1$, $A_{i,a_i,2i+1} = a_i$ et $A_{i,a_i,0<j<2i}$ aléatoires. Les éléments $A_{i,\bar{a_i},2i}$ restent inchangés.
>
> Alice choisit des nombres S_i $(1 \leq i \leq d)$ de k bits aléatoires.
>
> Alice choisit aléatoirement $0 \leq r < 2k$ et calcule la matrice A' telle que $A'_{i,j} = leftrot(A_{i,j} \oplus S_i, r)$ pour tout i et j.

Étape 3 : Bob utilise un transfert inconscient 1-parmi-2 pour obtenir $A'_{i,\overline{b_i}}$ pour tout $1 \leq i \leq l$.

Étape 4 : Alice envoie $S = leftrot(\bigoplus_{i=1}^{d} S_i, r)$ à Bob.

Étape 5 : Bob calcule $c = S \oplus (\bigoplus_{i=1}^{d} A'_{i,\overline{b_i}})$. Dans cette valeur, Bob recherche une longue série de zéros (la notion de long est définie dans l'article original) suivie d'un 1 et lit le bit à droite de ce 1. Si ce bit est 1, $a \geq b$, sinon $a < b$.

Protocole 2.6 : Problème des millionnaires tiré de Ioannidis et Grama (2003)

Type 1:

$A_{i,a_i} =$ | $k-s$ bits aléatoires | une série de 0 | 1 | a_i | $2i-2$ bits aléatoires |

Type 2:

$A_{i,\overline{a_i}} =$ | $k-s$ bits aléatoires | une série de 0 |

Soit i^* l'indice du premier bit de poids le plus fort qui différencie a et b. Pour tout $i > i^*$, on a donc $a_i = b_i$: $A_{i,\overline{b_i}}$ est du second type et n'influe pas sur le résultat. De ce fait, le résultat dépend uniquement de $\bigoplus_{i=1}^{i^*} A_{i,\overline{b_i}}$. Par définition, $a_{i^*} \neq b_{i^*}$, donc $A_{i^*,\overline{b_{i^*}}}$ est du premier type. Il possède donc une série de 0 suivi de 1 et a_i. Notons que ces deux bits particuliers sont les $2i^{*ème}$ et

$2i^* + 1^{ème}$ bits de poids faibles. Par ailleurs, pour tout $i < i^*$, les $2i^{*ème}$ et $2i^* + 1^{ème}$ bits font partie de la série de 0 et n'influe donc pas le résultat. Le résultat final dépend donc entièrement de $A_{i^*,\overline{b_{i^*}}}$. Le bit lu par Bob sera donc a_{i^*}. Si celui-ci vaut 1, Alice possède donc une valeur plus grande.

La valeur de Bob est protégée d'Alice si le protocole de transfert inconscient est sécurisé. De plus, la valeur d'Alice est elle aussi protégée de Bob grâce aux masques S_i ajoutés aux éléments de la matrice. Par ailleurs, en analysant la position de la longue série de zéros, Bob ne peut pas même déterminer le préfixe de a grâce à la rotation aléatoire qui a été appliquée aux données. En d'autres termes, Bob ne peut pas déterminer l'indice du premier bit de poids fort à différencier a et b.

Quatre solutions différentes au problème des millionnaires viennent d'être présentées. Les protocoles de comparaisons sont parmi les outils essentiels des calculs multi-parties et permettent généralement de résoudre des problèmes bien plus complexes.

2.4.2 Protocoles de calcul de somme

L'objectif de ces protocoles est de permettre à un groupe d'utilisateurs P_i $(0 \leq i \leq k-1)$ possédant chacun une valeur privée x_i de calculer conjointement la somme de leur valeur. La solution la plus évidente a été présentée par Clifton (2001) et est expliquée dans le protocole 2.7.

Seules k communications consécutives sont nécessaires dans ce protocole. En supposant que les données transmises sont chiffrées pour éviter les écoutes clandestines, deux opérations cryptographiques sont nécessaires pour chaque utilisateur. Malheureusement, cette approche possède un désavantage majeur : si P_{i-1} et P_{i+1} forment une collusion, ils peuvent déterminer la valeur de P_i.

On peut améliorer le temps d'exécution de ce protocole en permettant deux communications simultanées. Cette amélioration est présentée dans le protocole 2.8. L'idée principale est de

Données : Entier x_i appartenant à P_i $(0 \leq i \leq k-1)$. Paramètre public N, $\sum_{i=0}^{k-1} x_i \ll N$

Résultat : $\sum_{i=0}^{k-1} x_i$

Étape 1 : P_0 choisit un entier aléatoire $r \in \mathbb{Z}_N$.

Étape 2 : P_0 envoie $X = x_0 + r \mod N$ à P_1.

Étape 3 : L'ensemble des P_i sauf P_0 exécute l'algorithme :

> **pour** $i = 1$ **à** $k-1$ **faire**
>
> > P_i reçoit X et envoie $X + x_i$ à $P_{i+1 \mod k}$.
>
> **fin**

Étape 4 : P_0 reçoit $X = r + \sum_{i=0}^{k-1} x_i$, soustrait r et communique le résultat.

Protocole 2.7 : Protocole de calcul de somme tiré de Clifton (2001)

diviser les participants en deux groupes qui calculeront des sommes partielles. On supposera pour simplifier que le nombre de participants est pair.

Données : Entier x_i appartenant à P_i $(0 \leq i \leq k-1)$. Paramètre public N, $\sum_{i=0}^{k-1} x_i \ll N$.

Résultat : $\sum_{i=0}^{k-1} x_i$.

Étape 1 : P_0 choisit un entier aléatoire $r \in \mathbb{Z}_N$. En même temps, $P_{\frac{k}{2}}$ choisit un entier aléatoire r'.

Étape 2 : P_0 envoie $X = x_0 + r \mod N$ à P_1. $P_{\frac{k}{2}}$ envoie $X' = x_{\frac{k}{2}} + r' \mod N$ à $P_{\frac{k}{2}+1}$.

Étape 3 : L'ensemble des P_i sauf P_0 et $P_{\frac{k}{2}}$ exécute l'algorithme :

> **pour** $i = 1$ **à** $\frac{k}{2} - 1$ **faire**
>
> > P_i reçoit X et envoie $X + x_i$ à P_{i+1}.
> >
> > $P_{\frac{k}{2}+i}$ reçoit X' et envoie $X' + x_i$ à $P_{\frac{k}{2}+i+1 \mod k}$.
>
> **fin**

Étape 4 : $P_{\frac{k}{2}}$ reçoit $X = r + \sum_{i=0}^{\frac{k}{2}-1} x_i \mod N$, et envoie $X^* = X - r' \mod N$ à P_0

Étape 5 : P_0 reçoit $X^* = r - r' + \sum_{i=0}^{\frac{k}{2}-1} x_i \mod N$ et $X' = r' + \sum_{i=\frac{k}{2}}^{k-1} x_i \mod N$. Il communique le résultat $\sum_{i=0}^{k-1} x_i = X^* + X' - r \mod N$.

Protocole 2.8 : Protocole de calcul de somme inspiré de Clifton (2001)

De cette façon, $k+1$ communications sont nécessaires, mais elles peuvent être faites deux par deux. Le temps d'exécution s'en trouve ainsi divisé de moitié. Selon le nombre d'utilisateurs, on peut les diviser en 3, 4 ou davantage encore, améliorant ainsi le temps d'exécution tout en augmentant légèrement le nombre de communications.

Notons qu'aucune information supplémentaire n'est divulguée par rapport au protocole original. En effet, les sommes partielles sont protégées par les variables aléatoires r et r' choisies par deux personnes différentes. Cependant, si P_{i-1} et P_{i+1} forment une collusion, ils sont toujours en mesure de déterminer la valeur de P_i. L'auteur parvient à corriger ce défaut dans une autre approche [Clifton *et al.* (2002)], en fragmentant les valeurs en k segments et en exécutant k protocoles de somme en permutant la position des participants à chaque protocole.

L'approche de Sheikh *et al.* (2010) permet de remédier à ce même problème en fragmentant les valeurs initiales x_i sous la forme $\sum_{j=0}^{k-1} x_{i,j} = x_i$ et en dispersant ces fragments entre les utilisateurs. La collusion de P_{i-1} et P_{i+1} ne permet donc plus de déterminer x_i. Cependant, le protocole est plus complexe : au total, $k(k-1)$ communications doivent être établies pour échanger les fragments.

Le protocole imaginé par Hasan *et al.* (2012) permet d'atteindre un niveau de sécurité similaire en réduisant le nombre de communications. L'innovation principale est de fragmenter x_i en $s+1$ fragments et d'envoyer ces fragments à s participants choisis aléatoirement. Le paramètre s doit donc être choisi de manière à ce qu'il soit très peu probable que les s destinataires forment une collusion. L'ensemble du protocole est présenté dans le protocole 2.9.

La valeur x_i de P_i est donc protégée si au moins un des destinataires choisis par P_i est honnête. Si ce n'est pas le cas, le fragment local conservé par P_i reste protégé (et de ce fait x_i aussi) à condition que P_i ait reçu au moins un fragment de la part d'un participant lui-même honnête. Une grande valeur de s accroît donc la sécurité au détriment de la complexité communicationnelle.

Données : Entier x_i appartenant à P_i $(0 \leq i \leq k-1)$. Paramètre de sécurité $1 \leq s \leq k$.

Résultat : $\sum_{i=0}^{k-1} x_i$.

Étape 1 : Pour tout $i \neq 0$, P_i fragmente x_i sous la forme $x_i = \sum_{j=1}^{s+1} x_{i,j}$. Il choisit ensuite aléatoirement s utilisateurs et envoie à chacun un des fragments. Le dernier fragment $x_{i,s+1}$ reste local.

Étape 2 : L'ensemble des utilisateurs sauf P_0 suivent l'algorithme suivant :

P_j calcule la somme σ_j de tous les fragments reçus et de son fragment local.

P_j envoie le résultat partiel σ_j à P_0.

Étape 3 : P_0 déchiffre les résultats partiels et les additionne $\sigma = \sum_{i=1}^{k-1} \sigma_i = \sum_{i=1}^{k-1} \sum_{j=1}^{s+1} x_{i,j} = \sum_{i=1}^{k-1} x_i$. Si P_0 possède lui aussi une valeur, il l'ajoute à la somme pour obtenir $\sum_{i=0}^{k-1} x_i$ et diffuse le résultat.

Protocole 2.9 : Protocole de calcul de somme tiré de Hasan *et al.* (2012)

Il faut $(k-1)s$ échanges pour transférer l'ensemble des fragments, puis $k-1$ pour envoyer les sommes partielles à P_0 : $(k-1)(s+1)$ communications sont donc nécessaires pour que le protocole aboutisse. De plus, la plupart d'entre elles peuvent être faites simultanément, ce qui implique un gain de temps non négligeable.

Les protocoles de calcul de somme sont des outils très utilisés. Ils servent en autre à résoudre le problème de recherche du maximum, décrit ci-après.

2.4.3 Problème de recherche du maximum

Ce problème tend à généraliser le problème des millionnaires à n individus. Chacun de ces individus P_i possède une valeur x_i et on cherche à savoir qui possède $\max\{x_i \mid 1 \leq i \leq n-1\}$ tout en protégeant les valeurs. Certaines approches ont un but sensiblement différent : elles visent à diffuser la valeur maximale en gardant secrète l'identité de son possesseur.

Deux approches existent. La première consiste à utiliser un protocole de comparaison classique à deux utilisateurs et à le répéter entre les différents utilisateurs jusqu'à déterminer qui possède

la valeur maximale. L'une des difficultés de cette approche est de ne pas divulguer le résultat des comparaisons entre deux utilisateurs.

La solution proposée par Zhang et Makedon (2005) parvient à surmonter cela grâce à un système de vecteurs qui camoufle les résultats des comparaisons jusqu'au résultat final. Cette solution est présentée dans le protocole 2.10. Elle nécessite l'utilisation d'un protocole de comparaison dit asymétrique, ce qui signifie qu'une seule des deux parties connaît le résultat de la comparaison.

Le vecteur $\pi(T_i')$ généré par P_0 à l'étape 4 représente le vecteur T_i que devrait avoir P_i s'il avait gagné toutes les comparaisons. Par conséquent, si $\pi(T_i') = T_i$, x_i est la valeur maximale.

À la fin de ce protocole, la valeur maximale est connue de tous, mais son propriétaire reste inconnu. Cependant, cela requiert $n^2 - n$ comparaisons avant d'obtenir le résultat, et toutes les valeurs doivent impérativement être différentes deux à deux. La complexité de ce protocole dépend des protocoles de comparaison utilisés aux étapes 3 et 4, ainsi que du protocole de somme des étapes 5 et 6. L'étape la plus contraignante est l'étape 3 pour laquelle il y a $(n^2-n)C$ communications, où C est le nombre de communications du protocole de comparaison utilisé. C étant généralement constant, quelque soit le protocole, la complexité communicationnelle de cette solution est de l'ordre de $O(n^2)$.

La seconde approche cherche à comparer l'ensemble des valeurs bit à bit selon quelques règles simples exposées dans le protocole 2.11. On notera $x_{i,j}$ le $j^{ème}$ bit de poids fort de la valeur x_i et l le nombre de bits des valeurs.

À l'aide du protocole 2.11, on obtient ainsi la valeur maximale. Cependant, les premiers bits significatifs de chacune des valeurs sont révélés. Il s'agit donc de trouver une méthode permettant de dissimuler l'identité du possesseur du maximum, et de protéger entièrement les autres valeurs.

La méthode présentée par Hasan *et al.* (2013) résout ce problème au moyen d'un calcul de somme. On supposera que P_0 est l'utilisateur chargé d'obtenir et de diffuser le maximum. Au

Données : x_i appartenant à chaque utilisateur P_i $(0 \leq i \leq n-1)$
Résultat : $\max\{x_i \mid 1 \leq i \leq n-1\}$
Étape 1 : **pour chaque** $i \neq 0$ **faire**
P_i crée un vecteur V_i de longueur $2n$ alternant les valeurs de x_i et $-x_i$.
P_i chiffre le vecteur (chaque élément) avec sa fonction de chiffrement $E_i(\cdot)$.
Il envoie ce vecteur chiffré $E_i(V_i)$ à P_0.
fin
Étape 2 : P_0 suit l'algorithme suivant :
P_0 choisit un vecteur aléatoire R de longueur $2n$ et une fonction de permutation π.
La fonction π permet d'échanger aléatoirement la $2j^{ème}$ valeur avec la $2j+1^{ème}$ valeur dans un vecteur de taille $2n$.
pour chaque $i \neq 0$ **faire**
P_0 calcule le vecteur chiffré $E_i(V_i') = E_i(\pi(V_i + R)) = \pi(E_i(V_i) \cdot E_i(R))$.
Ce vecteur est transmis à P_i.
fin
Étape 3 : **pour chaque** $i \neq 0$ **faire**
P_i déchiffre $E_i(V_i')$. $V'_{i,j}$ représente la valeur du $j^{ème}$ élément du vecteur V_i'.
pour $j = 0$ **à** $N-1$ **faire**
P_i et P_j comparent $V'_{i,2j}$ (et $V'_{i,2j+1}$) avec $V'_{j,2j}$ (et $V'_{j,2j+1}$).
Un vecteur T_i de taille $2N$ est utilisé pour conserver les résultats.
Si $V'_{i,2j} > V'_{j,2j}$, P_i enregistre $T_{i,2j} = 1$, sinon $T_{i,2j} = 0$. De même pour $T_{i,2j+1}$.
Pour le cas particulier $i = j$, P_i enregistre $T_{i,2i} = T_{i,2i+1} = 0$.
fin
fin
Étape 4 : **pour chaque** $i \neq 0$ **faire**
P_0 génère un vecteur T_i' pour lequel $T'_{i,2j} = 1$, $T'_{i,2j+1} = 0$ et $T'_{i,2i} = T'_{i,2i+1} = 0$.
P_i et P_0 comparent $\pi(T_i')$ et T_i afin de savoir si ces valeurs sont égales. P_i est le seul à connaître le résultat.
Si $\pi(T_i') = T_i$, alors P_i a gagné toutes les comparaisons et possède la valeur maximale.
fin
Étape 5 : Les participants réalisent un protocole de somme de manière à ce que P_0 connaisse le résultat.
Étape 6 : Ils calculent une seconde fois leur somme, mais cette fois le participant possédant la plus grande valeur s'abstient (il ajoute zéro).
Étape 7 : P_0 calcule la valeur maximale par soustraction et la diffuse aux utilisateurs. Si P_0 reçoit deux sommes égales, c'est lui qui possède la plus grande valeur et la diffuse.

Protocole 2.10 : Protocole de calcul de maximum tiré de Zhang et Makedon (2005)

Données : Les entiers $x_0, x_1, ..., x_i, ..., x_{n-1}$

Résultat : $\max\{x_i \mid 1 \leq i \leq n-1\}$

pour $j = 1$ **à** l **faire**

  si $\exists i$ *tel que* $x_{i,j} = 1$ **alors**

    Le j^{eme} bit du maximum est 1. Les participants qui ont envoyé $x_{i,j} = 0$ sont éliminés.

  sinon

    Le j^{eme} bit du maximum est 0.

fin

Protocole 2.11 : Calcul de maximun

lieu de lui transférer la valeur de chaque $x_{i,j}$, les autres utilisateurs lui envoient la somme $\sum_i x_{i,j}$ grâce à un des protocoles déjà présentés. Cependant, P_0 peut encore déterminer combien de personnes possèdent $x_{i,j} = 1$. L'auteur propose alors de remplacer les 1 par des valeurs aléatoires non nulles. P_0 obtiendra donc un nombre aléatoire s'il existe i tel que $x_{i,j} = 1$ et une somme nulle sinon. Avec l'ensemble de ces recommandations, on obtient le protocole 2.12.

Données : Les entiers x_i appartenant à chaque utilisateur $P_i (0 \leq i \leq n-1)$.

Résultat : $m = \max\{x_i \mid 1 \leq i \leq n-1\}$

pour $j = 1$ **à** l **faire**

  Étape 1 : $\forall i$, si $x_{i,j} = 1$, P_i choisit aléatoirement $v_{i,j} \neq 0$, sinon il choisit $v_{i,j} = 0$.

  Étape 2 : L'ensemble des P_i calculent $\sum_i v_{i,j}$ et P_0 obtient le résultat.

  Étape 3 : Si $\sum_i v_{i,j} \neq 0$, le $j^{ème}$ bit du maximum (noté m_j) est 1, sinon 0. P_0 diffuse m_j.

  Étape 4 : Chaque P_i détermine s'il est éliminé de la façon suivante :

  si $m_j = 1$ *et* $x_{i,j} = 0$ **alors**

    P_i est éliminé, il continue le protocole avec la valeur $x_i = 0$.

  fin

fin

Protocole 2.12 : Protocole de calcul de maximum tiré de Hasan *et al.* (2013)

Si $\sum_i v_{i,j} \neq 0$ lors de l'étape 3, cela signifie qu'au moins un participant possède la valeur $x_{i,j} = 1$. Par conséquent, $m_j = 1$. De plus, tous les utilisateurs ayant $x_{i,j} = 0$ ne peuvent pas posséder la valeur maximale, ils doivent donc être éliminés lors de l'étape 4.

Si on utilise le protocole de somme présenté dans le même article (voir protocole 2.9), $l(2n^2 - 3n + 1)$ communications sont nécessaires pour calculer toutes les sommes et lk pour diffuser le résultat au fur et à mesure. On obtient donc une complexité communicationnelle de $l(2n^2 - 2n + 1)$.

2.5 Conclusion sur les calculs multi-parties

Les protocoles de comparaison et de calcul de maximum sont les plus intéressants concernant notre contexte et notre approche du problème. Il est donc important d'analyser leur sécurité et leur complexité. Aucun des protocoles vus dans cette revue de littérature ne divulgue d'information face à un attaquant semi-honnête. Cependant, la complexité des protocoles de calcul de maximum aussi bien calculatoire que communicationnelle ($\Omega(n^2)$) peut s'avérer trop élevée pour des téléphones intelligents, dont les capacités de calcul et de transmission sont limitées. Il sera donc intéressant de trouver des méthodes alternatives spécifiques à notre contexte : c'est l'objectif des deux prochains chapitres.

CHAPITRE 3

PROTOCOLES DE COMPARAISONS APPLIQUÉS À UN ARBRE BINAIRE

L'objectif de ce chapitre est de proposer un protocole permettant au prouveur P d'obtenir la valeur maximale $E_T(x_{max})$ détenue par les témoins W_i $(1 \leq i \leq n)$ chiffrée avec la clé du juge T. Pour cela, nous chercherons à savoir lequel des témoins W_i possède la plus grande valeur x_i, puis à permettre à P de récupérer cette valeur chiffrée avec la clé du juge. Il s'agit donc d'un protocole de comparaison à n individus avec une partie externe P qui doit recevoir le résultat.

Contrairement aux divers protocoles de calcul de maximum étudiés en revue de littérature, P construira un arbre binaire de comparaison avec les n témoins dans lequel ceux-ci seront éliminés jusqu'à ce qu'il n'en reste plus qu'un. L'avantage d'une telle méthode est évident : il n'est plus nécessaire de comparer l'ensemble des témoins deux à deux. Il y a donc à la fois un gain de temps et de communication. Cependant, pour réaliser cela, il est indispensable que P connaisse le résultat de certaines comparaisons.

On posera toutefois certaines contraintes : P ne doit pas être en mesure de connaître d'autres comparaisons que celles requises par l'arbre et les témoins ne doivent absolument rien apprendre tout au long du protocole. Le possesseur du maximum ne doit pas même savoir que sa valeur est effectivement la plus grande. De plus, P ne doit connaître aucune des valeurs non-chiffrées des témoins, ni même partiellement. Il ne doit pas non plus être en mesure de récupérer une valeur chiffrée autre que $E_T(x_{max})$.

Ce chapitre présente trois protocoles de comparaison qui peuvent répondre à de tels objectifs. Les deux premiers sont des adaptations des protocoles de Blake et Kolesnikov (2004) et de Lin et Tzeng (2005). Le troisième protocole a été conçu spécifiquement pour ce contexte, ce qui nous permet de réduire considérablement la charge communicationnelle et calculatoire au détriment d'une certaine divulgation d'information qui sera quantifiée et analysée.

3.1 Adaptation de Blake et Kolesnikov (2004)

Dans un premier temps, nous nous intéresserons au cas où il n'y a que deux témoins et un prouveur. Nous généraliserons ensuite cette approche à n témoins pour répondre au problème initial.

Deux individus Alice et Bob (notés A et B dans les protocoles) possèdent chacun un entier privé et une troisième personne P souhaite savoir lequel des deux possède la plus grande valeur. Les protocoles énoncés dans ce chapitre ont pour objectif de permettre à P de savoir qui à la plus grande valeur sans qu'Alice ou Bob n'apprennent aucune information et n'aient à divulguer leur valeur.

Les solutions proposées dans cette section sont des adaptations du protocole de Blake et Kolesnikov (2004).

3.1.1 Protocole

Le système de chiffrement utilisé est le système de Paillier. On notera N_A la clé publique d'Alice, $E_A(\cdot)$ sa fonction de chiffrement, N_P la clé publique de P et $E_P(\cdot)$ sa fonction de chiffrement.

Considérons le cas général où on peut se permettre d'établir une communication directe entre Alice et Bob. On considérera aussi que tout le monde est capable de capturer les messages émis par les autres participants. Notons a et b les valeurs respectives d'Alice et Bob, et a_i le i^{eme} bit de poids fort de a. On suppose que A et B se sont entendus sur la longueur l en bits de leur valeur.

Le protocole 3.1 est une solution possible au problème, mais nous montrerons qu'elle n'est pas sécurisée face à certaines formes de collusion. Par ailleurs, elle ne tolère pas que $a = b$ comme nous l'expliquerons plus tard.

Données : Les valeurs a et b de A et B. Les clés publiques N_A et N_P de A et P avec les fonctions de chiffrement associées $E_A(\cdot)$ et $E_P(\cdot)$ respectivement.

Résultat : P détermine $a > b$ ou $a < b$.

Étape 1 : A chiffre chaque bit a_i grâce à $E_A(\cdot)$ et envoie l'ensemble des bits chiffrés à B.

Étape 2 : B exécute l'algorithme suivant :

B choisit aléatoirement $r_B \in_R \mathbb{Z}_{N_A}$ et une permutation $\pi_B(\cdot)$.

B calcule :

pour $i = 1$ **à** l **faire**

$E_A(d_i) = E_A(a_i - b_i) = E_A(a_i) \cdot E_A(-b_i)$.

$E_A(f_i) = E_A(a_i \oplus b_i) = E_A(a_i - 2a_i b_i + b_i) = E_A(a_i)^{1-2b_i} \cdot E_A(b_i)$.

$E_A(\gamma_i) = E_A(2\gamma_{i-1} + f_i) = E_A(\gamma_{i-1})^2 \cdot E_A(f_i)$ avec $\gamma_0 = 0$.

$E_A(\delta_i) = E_A(d_i + r_i(\gamma_i - 1) + r_B) = E_A(d_i) \cdot (E_A(\gamma_i) \cdot E_A(-1))^{r_i} \cdot E_A(r_B)$

où $r_i \in_R \mathbb{Z}_{N_A}$.

fin

B envoie $(E_A(\delta_1^*), ..., E_A(\delta_l^*)) = \pi_B(E_A(\delta_1), ..., E_A(\delta_l))$ à A.

B envoie $E_P(r_B)$ à P.

Étape 3 : A déchiffre l'ensemble des $E_A(\delta_i^*)$ et les chiffre à nouveau avec $E_P(\cdot)$. A choisit une permutation $\pi_A(\cdot)$ et envoie ensuite $(E_P(\delta_1^{**}), ..., E_P(\delta_l^{**})) = \pi_A(E_P(\delta_1^*), ..., E_P(\delta_l^*))$ à P.

Étape 4 : P déchiffre l'ensemble des $E_P(\delta_i^{**})$. Si le vecteur δ^{**} contient la valeur $r_B + 1$, on a $a > b$. Si le vecteur contient la valeur $r_B - 1$, on a $a < b$.

Protocole 3.1 : Protocole de comparaison

3.1.1.1 Exactitude

L'exactitude de ce protocole est la même que celle du protocole original vue en revue de littérature.

Notons que le vecteur δ^{**} est une permutation du vecteur δ. Si le vecteur δ ne contient ni la valeur $r_B + 1$, ni la valeur $r_B - 1$, cela signifie que $a = b$. P est donc capable de déterminer si

les valeurs d'Alice et Bob sont égales, ce que nous souhaitons éviter. Ce protocole ne tolère donc pas que $a = b$.

3.1.1.2 Preuves de sécurité

On s'intéresse à la sécurité face à un attaquant semi-honnête, c'est-à-dire un attaquant qui essayera d'apprendre le plus d'information possible sans empêcher le protocole d'aboutir. Il doit donc suivre le protocole initial. Toutefois, il peut aussi effectuer d'autres calculs afin d'obtenir des informations supplémentaires.

Il faut s'assurer que (1) Alice ne peut pas connaître b, (2) Bob ne peut pas connaître a et que (3) P ne peut connaître ni a ni b. Par ailleurs, il faut aussi vérifier que (4) P est le seul à connaître le résultat de la comparaison et (5) que personne ne sait la position du premier bit à différencier a et b.

La liste suivante indique les variables connues par chaque partie.

- Alice : $a, \delta^*, \delta^{**}, \pi_A(\cdot)$.
- Bob : $b, r_i(\forall i), r_B, \pi_B(\cdot)$.
- P : δ^{**}, r_B.

Supposons qu'Alice soit semi-honnête. Rappelons que les valeurs du vecteur δ^* obtenu par Alice sont aléatoires sauf pour un des éléments qui vaut $1 + r_B$ ou $-1 + r_B$. Mais puisque la valeur de r_B est inconnue d'Alice, toutes ces valeurs lui sont donc indistinguables. En effet, toutes les valeurs de b peuvent générer le même vecteur δ^* de façon équiprobable. Pour n'importe quel b donné, on peut déterminer i^* le plus petit indice qui différencie a et b. Sans perte de généralité, on choisit $\pi_B(\cdot)$ comme étant la fonction identité. On a donc $\delta^* = \delta$.

Par conséquent :

$$
\begin{aligned}
r_B &= \delta_{i^*} - 1 && \text{si } a > b \\
r_B &= \delta_{i^*} + 1 && \text{sinon} \\
r_i &= (\delta_i - r_B) \cdot (-1)^{-1} \pmod{N_A} && 1 \leq i < i^* \\
r_i &= (\delta_i - (a_i - b_i) - r_B) \cdot (\gamma_i - 1)^{-1} \pmod{N_A} && i^* < i \leq l
\end{aligned}
$$

Notons que γ_i est une suite croissante et que $\gamma_l - 1 < 2^l - 1 \ll \sqrt{N_A}$. Par conséquent, comme les facteurs premiers de N_A sont de taille équivalente, on en déduit que $\gamma_i - 1$ et N_A sont copremiers et que $(\gamma_i - 1)^{-1}$ existe quelque soit i. Par ailleurs, remarquons que r_{i^*} n'a pas besoin d'être défini, cette valeur n'influe pas sur le vecteur δ. Par conséquent, il existe une solution unique $(r_1, ..., r_{i^*-1}, r_{i^*+1}, ..., r_l)$ qui puisse générer le vecteur δ. La démonstration est équivalente pour n'importe quelle permutation $\pi_B(\cdot)$ donnée. Pour toute valeur de b, il existe donc le même nombre de solutions équiprobables. La valeur de Bob est ainsi inconditionnellement protégée. Les objectifs (1) et (5) sont ainsi assurés même si Alice est semi-honnête. Cependant, si Alice parvient à déchiffrer $E_P(r_B)$ (qu'elle peut obtenir grâce à une écoute passive), elle est en mesure de connaître le résultat de la comparaison, et donc d'avoir une information partielle sur b. Mais l'objectif (4) est assuré par la sécurité sémantique du système de Paillier, si la puissance de calcul d'Alice est bornée polynomialement.

Supposons que Bob soit à son tour semi-honnête. Bob ne possède aucune donnée non chiffrée qui dépende de a. Les exigences (2),(4) et (5) sont donc assurées grâce à la sécurité sémantique du système de Paillier, si la puissance de calcul de Bob est bornée polynomialement.

Supposons que P soit semi-honnête. P possède uniquement le vecteur δ^{**} dont l'une des valeurs est $1 + r_B$ ou $-1 + r_B$. La position de cette valeur est rendue aléatoire par les permutations $\pi_A(\cdot)$ et $\pi_B(\cdot)$. Les autres valeurs du vecteur sont aléatoires. On peut démontrer comme précédemment que toutes les valeurs de a et b sont équiprobables (tant que le résultat de la comparaison est respecté). Les valeurs d'Alice et Bob sont donc inconditionnellement protégées. Toutefois, si P réalise une écoute passive, il connaît le chiffrement de chaque bit de a.

Les objectifs (3) et (5) sont donc vérifiés grâce à la sécurité sémantique du système de Paillier, si la puissance de calcul de P est bornée polynomialement.

Si Alice ou Bob est semi-honnête augmenté, il peut mentir sur sa valeur a ou b. Toutefois, cela n'apporte aucune information supplémentaire et le résultat obtenu sera faussé. Le protocole est donc sécurisé face à un attaquant semi-honnête augmenté.

On peut montrer facilement que ce protocole est sécurisé face à une collusion d'Alice et P. La collusion d'Alice et Bob peut quant à elle être ignorée. En effet, ensemble, ils peuvent connaître le résultat de la comparaison, ainsi que le premier bit à différencier a et b. Toutefois, connaissant a et b, on ne peut pas les empêcher de réaliser eux-mêmes un protocole de comparaison classique.

Comme nous le disions plus tôt, ce protocole n'est pas sécurisé face à certaines formes de collusion, en l'occurrence celle de Bob et P. En effet, si on analyse le vecteur δ en ignorant les permutations $\pi_B(\cdot)$ et $\pi_A(\cdot)$, celui-ci ressemble à ceci :

$$\delta = [r_B - r_1, r_B - r_2, ..., r_B - r_{i^*-1}, r_B \pm 1, x, ...]$$

i^* représente l'indice du premier bit qui différencie a et b. Avec les permutations, toutes ces valeurs sont mélangées et indistinguables pour P qui n'est capable de reconnaître que $r_B \pm 1$. Cependant, elles ne sont pas indistinguables pour Bob qui pourra reconnaître $r_B - r_1$, $r_B - r_2$, jusqu'à $r_B - r_{i^*-1}$. Bob et P sont donc capable ensemble de connaître l'indice du premier bit à différencier a et b. Connaissant aussi b, ils peuvent déterminer le préfixe de a. De plus, ils peuvent aussi déterminer tous les autres bits de a de la manière suivante :

$$\begin{aligned} a_{i^*+1} = b_{i^*+1} &\iff d_{i^*+1} = 0 \iff f_{i^*+1} = 0 \iff \gamma_{i^*+1} = 2 \qquad \text{sachant que } \gamma_{i^*} = 1 \\ &\iff \delta_{i^*+1} = r_{i^*+1} + r_b \end{aligned}$$

Par conséquent, si la valeur $r_{i^*+1} + r_b$ est présente dans δ, alors $a_{i^*+1} = b_{i^*+1}$, sinon on a $a_{i^*+1} = \overline{b_{i^*+1}}$. Sachant cela, on peut déterminer la valeur de a_{i^*+2} de la même manière et ainsi de suite. La permutation $\pi_A(\cdot)$ est donc totalement inutile.

Ce protocole est donc sécurisé uniquement si une telle collusion n'est pas possible.

3.1.2 Protocole corrigé

Le protocole 3.2 présente une deuxième solution possible en tenant compte des problèmes de sécurité rencontrés précédemment. Contrairement au protocole initial, celui-ci tolère que les valeurs a et b soient égales. Seules les étapes 3 et 4 ont été modifiées par rapport à la section précédente.

3.1.2.1 Exactitude

On peut démontrer l'exactitude de ce protocole en analysant les deux cas possibles. Les permutations n'ont pas d'impact sur l'exactitude, on considérera que $\pi_A(\cdot)$ et $\pi_B(\cdot)$ sont les fonctions identités. On a donc $\delta^* = \delta$ et $\mu^* = \mu$.

On suppose que $a \neq b$. Notons i^* l'indice du premier bit tel que $a_i \neq b_i$. On peut distinguer trois parties dans le vecteur δ et dans le vecteur μ :

- Pour tout $1 \leq i < i^*$, on a $a_i = b_i$, donc $f_i = 0$ et ainsi $\gamma_i = 2\gamma_{i-1} = 0$. Par conséquent, δ_i est égal à $r_B - r_i$ et est donc aléatoire, de même que μ_i.
- Pour $i = i^*$, on a $a_i \neq b_i$, donc $f_i = 1$ et $\gamma_i = 2\gamma_{i-1} + f_i = f_i = 1$. On obtient $\delta_i = d_i + r_B$, puis :

  $$\mu_i = (\delta_i - r_B + r_A) * k_A = (d_i + r_A) * (1 + r_A)^{-1} \mod N_A$$

  Si $a > b$, on a $a_i > b_i$ et $\mu_i = (1 + r_A) * (1 + r_A)^{-1} = 1$. Si $a < b$, μ_i est aléatoire.
- Pour tout $i^* < i \leq l$, on a $\gamma_{i-1} \geq 1$ et donc $\gamma_i \geq 2$. Par conséquent, δ_i est aléatoire quelques soient les valeurs de a_i et b_i. Il en va de même pour μ_i.

Données : Les valeurs a et b de A et B. Les clefs publiques N_A et N_P de A et P avec les fonctions de chiffrement associées $E_A(\cdot)$ et $E_P(\cdot)$, respectivement.

Résultat : P détermine $a > b$ ou $a \leq b$.

Étape 1 : A chiffre chaque bit a_i grâce à $E_A(\cdot)$ et envoie l'ensemble des bits chiffrés à B.

Étape 2 : B exécute l'algorithme suivant :

B choisit aléatoirement $r_B \in_R \mathbb{Z}_{N_A}$ et une permutation $\pi_B(\cdot)$.

B calcule :

pour $i = 1$ **à** l **faire**

$E_A(d_i) = E_A(a_i - b_i) = E_A(a_i) \cdot E_A(-b_i)$.

$E_A(f_i) = E_A(a_i \oplus b_i) = E_A(a_i - 2a_ib_i + b_i) = E_A(a_i)^{1-2b_i} \cdot E_A(b_i)$.

$E_A(\gamma_i) = E_A(2\gamma_{i-1} + f_i) = E_A(\gamma_{i-1})^2 \cdot E_A(f_i)$ avec $\gamma_0 = 0$.

$E_A(\delta_i) = E_A(d_i + r_i(\gamma_i - 1) + r_B) = E_A(d_i) \cdot (E_A(\gamma_i) \cdot E_A(-1))^{r_i} \cdot E_A(r_B)$

où $r_i \in_R \mathbb{Z}_{N_A}$.

fin

B envoie $(E_A(\delta_1^*), ..., E_A(\delta_l^*)) = \pi_B(E_A(\delta_1), ..., E_A(\delta_l))$ et $E_P(-r_B)$ à A.

Étape 3 : A déchiffre les éléments $E_A(\delta_i^*)$ et exécute l'algorithme suivant :

A choisit $r_A \in_R \mathbb{Z}_{N_P}$ et une permutation $\pi_A(\cdot)$.

A calcule $k_A = (1 + r_A)^{-1} \pmod{N_P}$ et $\forall i$,

$E_P(\mu_i) = E_P((\delta_i^* + r_A - r_B) * k_A) = (E_P(\delta_i^* + r_A) \cdot E_P(-r_B))^{k_A}$

A envoie $(E_P(\mu_1^*), ..., E_P(\mu_l^*)) = \pi_A((E_P(\mu_1), ..., E_P(\mu_l))$ à P.

Étape 4 : P déchiffre l'ensemble des $E_P(\mu_i^*)$. Si le vecteur μ^* contient la valeur 1, on a $a > b$. Sinon, on a $a \leq b$.

Protocole 3.2 : Protocole de comparaison

Si $a = b$, le vecteur δ ne contient qu'une seule partie dont tous les éléments sont aléatoires et indistinguables. Contrairement au protocole précédent, celui-ci fonctionne avec $a = b$, sans qu'aucun des participants ne puisse déduire des informations supplémentaires.

3.1.2.2 Preuves de sécurité

Il faut s'assurer que (1) Alice ne peut pas connaître b, (2) Bob ne peut pas connaître a et que (3) P ne peut connaître ni a ni b. Par ailleurs, il faut aussi vérifier que (4) P est le seul à connaître le résultat de la comparaison, (5) que personne ne sait si $a = b$ et (6) que personne ne sait la position du premier bit à différencier a et b.

La liste suivante indique les variables connues par chaque partie.

- Alice : $a, \delta^*, r_A, k_A, \pi_A(\cdot)$.
- Bob : $b, r_i(\forall i), r_B, \pi_B(\cdot)$.
- P : μ^* et éventuellement r_B grâce à une écoute passive (ou en agissant comme intermédiaire).

Supposons qu'Alice soit semi-honnête. On peut démontrer exactement comme dans la section 3.1.1.2 que toute valeur possible de b peut générer le même vecteur δ^* équiprobablement. Les objectifs (1) et (5) sont ainsi assurés même si Alice est semi-honnête. Cependant, si Alice parvient à déchiffrer $E_P(-r_B)$, elle est en mesure de connaître le résultat de la comparaison. Mais l'objectif (4) est assuré par la sécurité sémantique du système de Paillier, si la puissance de calcul d'Alice est bornée polynomialement.

Supposons que Bob soit à son tour semi-honnête. Bob ne possède aucune donnée non chiffrée qui dépende de a. Les exigences (2),(4) et (5) sont donc assurées grâce à la sécurité sémantique du système de Paillier, si la puissance de calcul de Bob est bornée polynomialement.

Supposons que P soit semi-honnête. P possède uniquement le vecteur δ dont l'une des valeurs est 1 ou -1. La position de cette valeur est rendue aléatoire par les permutations $\pi_A(\cdot)$ et $\pi_B(\cdot)$. Toutes les valeurs de a et b peuvent générer le même vecteur μ^* équiprobablement. La démonstration est similaire à celle vue dans la section 3.1.1.2. Les valeurs d'Alice et Bob sont donc inconditionnellement protégées. Toutefois, si P réalise une écoute passive lors de l'étape 1, il connaît le chiffrement de chaque bit de a. Les objectifs (3) et (5) sont donc assurés dans

ce cas grâce à la sécurité sémantique du système de Paillier, si la puissance de calcul de P est bornée polynomialement.

Si Alice ou Bob est semi-honnête augmenté, il peut mentir sur sa valeur a ou b. Toutefois, cela n'apporte aucune information supplémentaire et le résultat obtenu sera faussé. Le protocole est donc sécurisé face à un attaquant semi-honnête augmenté.

La liste ci-dessous indique le gain de connaissances que pourrait provoquer une collusion de deux utilisateurs parmi les trois.

- AB : ensemble, ils peuvent connaître le résultat de la comparaison. Cependant, rien ne les empêche d'appliquer un protocole de comparaison classique entre eux.
- BP : $b, r_i(\forall i), r_B, \pi_B(\cdot), \mu^*$.
- AP : $a, \delta^*, r_A, r_B, \pi_A(\cdot), \mu$.

L'attaque que Bob et P pouvait réaliser dans la première version du protocole est désormais rendue impossible grâce à l'ajout de la variable aléatoire r_a. Cependant, grâce à une autre attaque, nous montrons ici qu'ils peuvent encore trouver le premier bit à différencier a et b, et même trouver a sous certaines conditions. L'attaque détaillée ci-après nécessite que les deux premiers bits de a soient égaux aux deux premiers bits de b. On met de côté pour l'instant les deux permutations. On a donc :

$$
\begin{aligned}
a_1 &= b_1 & a_2 &= b_2 \\
\delta_1 &\equiv r_1 + r_B \pmod{N_A} & \delta_2 &\equiv r_2 + r_B \pmod{N_A} \\
\mu_1 &\equiv (\delta_1 - r_B + r_A) \cdot (1 + r_A)^{-1} & \mu_2 &\equiv (r_2 + r_A) \cdot (1 + r_A)^{-1} \pmod{N_P} \\
&\equiv (r_1 + r_A) \cdot (1 + r_A)^{-1} \\
&\equiv ((r_1 - 1) + (1 + r_A)) \cdot (1 + r_A)^{-1} \\
&\equiv (r_1 - 1) \cdot (1 + r_A)^{-1} + 1 \pmod{N_P} \\
(1 + r_A)^{-1} &\equiv (\mu_1 - 1) \cdot (r_1 - 1)^{-1} \pmod{N_P}
\end{aligned}
$$

Connaissant r_1 et μ_1, Bob et P sont donc capable de retrouver r_A s'il n'y a pas de permutation. On peut ignorer la fonction $\pi_B(\cdot)$ puisqu'elle est connue de Bob. Voyons maintenant comment Bob et P peuvent découvrir r_A si Alice a appliqué une permutation $\pi_A(\cdot)$. Bob et P ne connaissent pas la position de μ_1 dans le vecteur μ^*.

Supposons que $\mu_1^* = \mu_1$, on obtient une valeur r_A comme nous venons de l'expliquer. Si la supposition est exacte, alors l'élément $\mu_2 = (r_2 + r_A) \cdot (1 + r_A)^{-1}$ existe dans le vecteur μ^*. Réciproquement, si μ_2 existe dans le vecteur, il est extrêmement probable que $\mu_1^* = \mu_1$. Si μ_2 n'existe pas, alors la supposition est erronée et on peut réessayer en supposant que $\mu_2^* = \mu_1$ et ainsi de suite jusqu'à ce qu'il existe μ_2 dans le vecteur. Nous venons ainsi de découvrir r_A malgré la permutation $\pi_A(\cdot)$. Connaissant r_A, Bob et P peuvent maintenant réaliser la même attaque que dans la première version du protocole. Ils peuvent donc connaître la valeur a à condition que les deux premiers bits de a et b soient égaux. La solution à ce problème est évidente, Alice doit générer une valeur r_A différente pour chaque élément du vecteur μ.

Alice et P ensemble sont en mesure de calculer tous les éléments $\delta_i^* - r_B$. Ce qui revient au résultat du protocole original, dans lequel la valeur -1 peut apparaître. Alice et P peuvent donc déterminer s'il existe i tel que $\delta_i^* - r_B = -1$, et donc déterminer si $a > b$, $a < b$ ou $a = b$. En d'autres termes, ils sont en mesure de savoir si a et b sont égaux. Selon le contexte, ce n'est pas toujours une divulgation importante, mais nous allons voir comment s'en protéger sans calcul ni communication supplémentaire.

Cette solution n'est donc pas suffisamment protégée face aux collusions.

3.1.3 Protocole final

Comme nous l'avons vu dans la section précédente, il est nécessaire qu'Alice choisisse des r_A différents pour chaque élément du vecteur μ. Sans cela, sa valeur a ne sera pas protégée contre la collusion de Bob et P.

Dans l'autre forme de collusion qui nous pose problème, Alice et P sont capables de détecter que $a = b$ et c'est donc la valeur b de Bob qui n'est plus protégée. Notons que le cas problématique existe car il est possible que $a = b$, ce qui est exclu dans le protocole original. Nous allons donc faire en sorte que Bob, seul, soit en mesure de transformer a et b pour qu'ils ne soient pas égaux.

Les modifications par rapport au protocole précédent sont très légères : r_A est généré aléatoirement pour chaque élément du vecteur μ, et Alice envoie à Bob tous ses bits chiffrés sauf le bit de poids faible a_l. Bob détermine lui-même la valeur a'_l qui remplacera a_l dans le reste du protocole. On notera a' la valeur de a avec le dernier bit remplacé par a'_l.

- Si $b_l = 0$, alors $a'_l = 1$.
- Si $b_l = 1$, alors $a'_l = 0$.

On est ainsi certain que les valeurs de a' et b sont différentes.

Aucune autre modification n'est nécessaire sur le reste du protocole. On obtient le protocole 3.3. Remarquons toutefois qu'on obtient donc un résultat sensiblement différent. En effet, le résultat est désormais de la forme $a \geq b$ ou $a \leq b$.

3.1.3.1 Exactitude

On peut facilement prouver l'exactitude de ce nouveau protocole en analysant les quatre cas possibles.

Si a et b diffèrent avant le dernier bit, alors le résultat n'est pas influencé par a'_l. On a donc bien $a' > b \implies a > b$ et $a' < b \implies a < b$. Si a et b diffèrent uniquement sur le dernier bit, alors il n'y a aucune transformation sur a, donc $a' = a$. Si $a = b$ et $b_l = 1$, on obtient $a'_l = 0$ et donc $a' < b$. À l'inverse, si $a = b$ et $b_l = 0$, on obtient $a'_l = 1$ et donc $a' > b$.

Données : Les valeurs a et b de A et B. Les clefs publiques N_A et N_P de A et P avec les fonctions de chiffrement associées $E_A(\cdot)$ et $E_P(\cdot)$ respectivement.

Résultat : P détermine $a \geq b$ ou $a \leq b$.

Étape 1 : A chiffre chaque bit a_i $(1 \leq i < l)$ grâce à $E_A(\cdot)$ et envoie l'ensemble des bits chiffrés à B.

Étape 2 : B reçoit les données de A et exécute l'algorithme suivant :

B redéfinit $a_l = \overline{b_l}$.

B choisit aléatoirement $r_B \in_R \mathbb{Z}_{N_A}$ et une permutation $\pi_B(\cdot)$.

B calcule :

pour $i = 1$ **à** l **faire**

$E_A(d_i) = E_A(a_i - b_i) = E_A(a_i) \cdot E_A(-b_i)$.

$E_A(f_i) = E_A(a_i \oplus b_i) = E_A(a_i - 2a_i b_i + b_i) = E_A(a_i)^{1-2b_i} \cdot E_A(b_i)$.

$E_A(\gamma_i) = E_A(2\gamma_{i-1} + f_i) = E_A(\gamma_{i-1})^2 \cdot E_A(f_i)$ avec $\gamma_0 = 0$.

$E_A(\delta_i) = E_A(d_i + r_i(\gamma_i - 1) + r_B) = E_A(d_i) \cdot (E_A(\gamma_i) \cdot E_A(-1))^{r_i} \cdot E_A(r_B)$

où $r_i \in_R \mathbb{Z}_{N_A}$.

fin

B envoie $(E_A(\delta_1^*), ..., E_A(\delta_l^*)) = \pi_B(E_A(\delta_1), ..., E_A(\delta_l))$ et $E_P(-r_B)$ à A.

Étape 3 : A déchiffre les éléments δ_i^* et exécute l'algorithme suivant :

A choisit $r_{A,i} \in_R \mathbb{Z}_{N_P}$ $(\forall 1 \leq i \leq l)$, une permutation $\pi_A(\cdot)$ et calcule :

$E_P(\mu_i) = E_P((\delta_i^* + r_{A,i} - r_B) * (1 + r_{A,i})^{-1}) = (E_P(\delta_i^* + r_{A,i}) \cdot E_P(-r_B))^{(1+r_{A,i})^{-1}}$

A envoie $(E_P(\mu_1^*), ..., E_P(\mu_l^*)) = \pi_A((E_P(\mu_1), ..., E_P(\mu_l))$ à P.

Étape 4 : P déchiffre l'ensemble des μ_i^*. Si le vecteur μ^* contient la valeur 1, on a $a \geq b$. Sinon, on a $a \leq b$

Protocole 3.3 : Protocole de comparaison

En réunissant tous ces cas, on ne peut pas avoir $a' = b$ et les résultats seront :

$$a' > b \implies a > b \text{ ou } (a = b \text{ et } b_l = 0)$$

$$a' < b \implies a < b \text{ ou } (a = b \text{ et } b_l = 1)$$

3.1.3.2 Preuves de sécurité

Le protocole n'ayant presque pas changé, les objectifs de sécurité qui étaient atteints précédemment le sont toujours ici.

La collusion d'Alice et P ne peut plus déterminer si $a = b$. En effet, il a été démontré dans l'exactitude qu'il est impossible d'obtenir $a' = b$ sans la complicité de Bob. Par conséquent, l'une des valeurs $1 + r_B$ ou $-1 + r_B$ est nécessairement présente dans le vecteur δ^* et il n'est plus possible de déterminer avec certitude que $a = b$.

La collusion de Bob et P ne peut plus réaliser l'attaque que nous avions présentée dans la version précédente. En effet, même si les deux premiers bits de a sont égaux à ceux de b, μ_1 et μ_2 sont générés à partir de deux aléas $r_{A,1}$ et $r_{A,2}$ différents. Grâce à la permutation, ils ne peuvent donc pas calculer $r_{A,1}$ et $r_{A,2}$. On peut facilement montrer que toutes les valeurs possibles de a peuvent générer le même résultat de manière équivalente (à condition que le résultat de la comparaison soit respecté). Cette forme de collusion n'engendre donc aucune divulgation.

Cependant, la collusion de Bob et P est capable d'affiner le résultat. En effet, Bob connaît b_l et donc a'_l. Grâce à cela, ils peuvent déterminer les deux règles suivantes :

- Si $b_l = 1$, $a' > b \implies a > b$ et $a' < b \implies a \leq b$.
- Si $b_l = 0$, $a' > b \implies a \geq b$ et $a' < b \implies a < b$.

Par conséquent, si $b_l = 1$ et $a' > b$, alors Bob et P sont certains que $a \neq b$. Il en va de même si $b_l = 0$ et $a' < b$. Cependant, dans aucun cas ils ne sont capables de déterminer avec certitude que $a = b$. La divulgation d'information engendrée par cette forme de collusion est un risque qu'on peut accepter puisque, dans notre contexte, nous n'exigeons pas de cacher que $a \neq b$.

On considère maintenant le cas particulier où nos deux témoins, Alice et Bob, ne peuvent pas ou ne veulent pas établir de communication directe entre eux. Pour résoudre ce problème, on peut réutiliser le même protocole en utilisant P comme intermédiaire entre eux.

P servant d'intermédiaire, il connaît toutes les informations circulant entre les témoins Alice et Bob. Étant donné qu'on supposait précédemment qu'il était déjà capable d'écouter tous les messages échangés, cela ne représente pas un gain d'information supplémentaire. Les objectifs sont toujours atteints même si *P* est semi-honnête. Cependant, *P* est désormais capable de modifier les messages lors de leurs transmissions (modèle d'adversaire malicieux). Il est capable de modifier r_B. Cependant, s'il modifie les données qui transitent, il n'aura pas un résultat exploitable à la fin du protocole.

3.1.3.3 Analyse de la complexité

Trois communications sont nécessaires pour que le protocole aboutisse. De plus, en notant $|N|$ la taille en bits des clés publiques (et donc $2|N|$ la taille d'un message chiffré avec le système de Paillier) et l la taille des données a et b, un total de $6l|N|$ bits sont échangés.

D'un point de vue calculatoire, Alice réalise $4l-1$ opérations cryptographiques (chiffrements, déchiffrements et exponentiations modulaires), contre $5l+3$ opérations cryptographiques pour Bob. *P* réalise quant à lui l déchiffrements.

Notons qu'une partie des opérations de chiffrements effectuées par Bob peuvent être faites en amont, avant de recevoir les données d'Alice.

Si *P* sert d'intermédiaire entre Alice et Bob, les communications des étapes 1 et 2 doivent être doublées. Par conséquent, cinq communications sont désormais nécessaires et $10l|N|$ bits sont échangés. De plus, tous ces bits ont *P* comme expéditeur ou destinataire. La charge communicationnelle de *P* doit donc être prise en compte.

3.1.4 Généralisation à *n* individus

Pour simplifier la représentation, on supposera pour l'instant que n est une puissance de 2.

On pourrait appliquer le protocole précédent $\log_2(n)$ fois en ne conservant à chaque fois que les "gagnants" de l'itération précédente, mais cela représente un gain d'information pour les can-

didats. En effet, s'ils ne sont plus interrogés, cela signifie qu'ils ont perdu l'itération précédente (c.-à-d. qu'ils n'avaient pas la plus grande valeur) et à l'inverse, s'ils sont encore interrogés, qu'ils ont gagné l'itération précédente. Il s'agit donc d'un gain de connaissance dont on préférerait se passer.

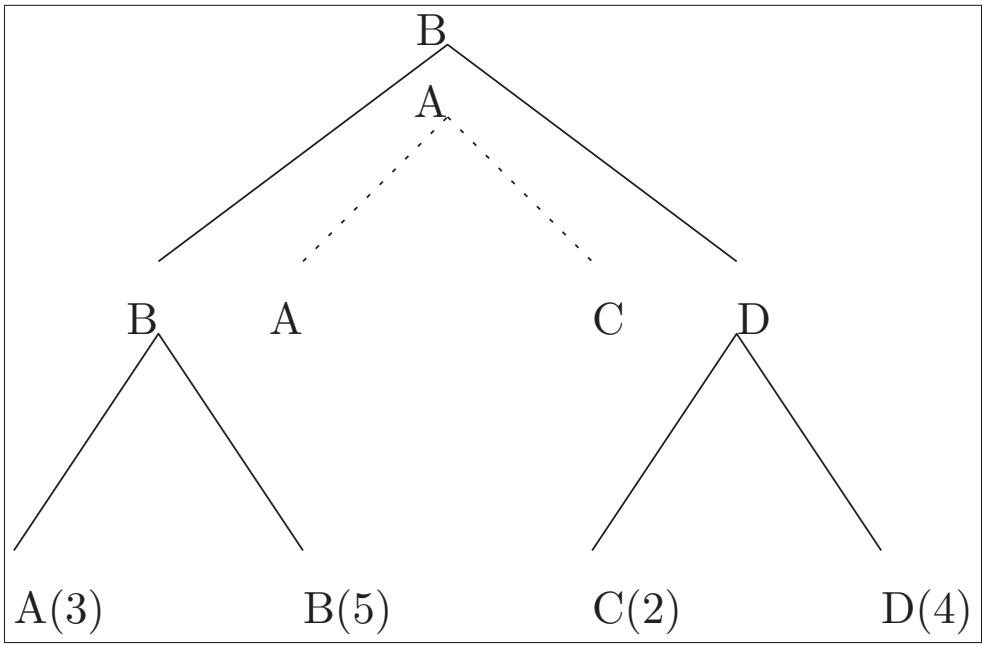

Figure 3.1 Protocole de comparaison appliqué à 4 candidats

Pour pallier ce problème, le bon sens voudrait donc qu'on continue à interroger les perdants entre eux pour empêcher que la présence ou l'absence d'interrogation révèle des résultats. La figure 3.1 permet de visualiser cette approche. L'arbre plein représente l'arbre des gagnants, c'est celui qui nous intéresse véritablement et qui nous permettra à la fin de savoir qui possède la valeur maximale. L'arbre en pointillé est celui des perdants. Bien que leurs comparaisons ne permettent pas de calculer le maximum, il représente un gain d'information supplémentaire pour P. Dans cet exemple, on souhaiterait éviter que P connaisse le résultat de la comparaison entre A et C. Le protocole 3.4 présente l'ensemble de ces exigences de manière plus générale.

Données : La valeur x_i de chaque témoin.

Résultat : P détermine le possesseur W_{max} de x_{max}.

Étape 1 : P définit $\mathscr{G} = \{W_i | \forall i\}$ l'ensemble des gagnants et $\mathscr{P} = \emptyset$ l'ensemble des perdants. Ces deux ensembles ne sont connus que de P.

répéter

Étape 2 : P choisit aléatoirement des couples de gagnants (W_i, W_j) parmi $\mathscr{G}$ et des couples de perdants notés $(W_{i'}, W_{j'})$ parmi $\mathscr{P}$. Un individu doit appartenir à un et un seul couple.

Étape 3 : L'ensemble des participants exécutent le protocole suivant :

pour chaque *couple* (W_i, W_j) **faire**

W_i et W_j réalisent un protocole de comparaison tel que P connaisse le résultat.

Le candidat qui possède la moins grande valeur est éliminé. Il est donc supprimé de $\mathscr{G}$ et ajouté à $\mathscr{P}$.

fin

pour chaque *couple* $(W_{i'}, W_{j'})$ **faire**

$W_{i'}$ et $W_{j'}$ réalisent un protocole de comparaison tel que P obtienne un résultat inexploitable.

fin

jusqu'à $\mathscr{G}$ *ne contient qu'un seul candidat*;

Étape 4 : P détermine W_{max} comme étant le seul élément de $\mathscr{G}$.

Protocole 3.4 : Généralisation de 2 à n participants.

Dans cette section, nous cherchons à concevoir un protocole de comparaison à deux participants qui correspond à l'étape 3 du protocole 3.4.

3.1.4.1 Protocole modifié

Le protocole 3.5 permet de réaliser la première itération de comparaison, c.-à-d. lorsqu'aucun participant n'a été éliminé ($\mathscr{P} = \emptyset$ dans le protocole 3.4). Les trois premières étapes sont identiques au cas à deux individus.

Données : Les valeurs a et b de A et B. Les clefs publiques N_A et N_P de A et P avec les fonctions de chiffrement associées $E_A(\cdot)$ et $E_P(\cdot)$ respectivement.

Résultat : P détermine $a \geq b$ ou $a \leq b$.

Étape 1 : A chiffre chaque bit a_i $(1 \leq i < l)$ grâce à $E_A(\cdot)$ et envoie l'ensemble des bits chiffrés à B.

Étape 2 : B reçoit les données de A et exécute l'algorithme suivant :

B redéfinit $a_l = \overline{b_l}$.

B choisit aléatoirement $r_B \in_R \mathbb{Z}_{N_A}$ et une permutation $\pi_B(\cdot)$.

B calcule :

pour $i = 1$ **à** l **faire**

$E_A(d_i) = E_A(a_i - b_i) = E_A(a_i) \cdot E_A(-b_i)$.

$E_A(f_i) = E_A(a_i \oplus b_i) = E_A(a_i - 2a_i b_i + b_i) = E_A(a_i)^{1-2b_i} \cdot E_A(b_i)$.

$E_A(\gamma_i) = E_A(2\gamma_{i-1} + f_i) = E_A(\gamma_{i-1})^2 \cdot E_A(f_i)$ avec $\gamma_0 = 0$.

$E_A(\delta_i) = E_A(d_i + r_i(\gamma_i - 1) + r_B) = E_A(d_i) \cdot (E_A(\gamma_i) \cdot E_A(-1))^{r_i} \cdot E_A(r_B)$

où $r_i \in_R \mathbb{Z}_{N_A}$.

fin

B envoie $(E_A(\delta_1^*), ..., E_A(\delta_l^*)) = \pi_B(E_A(\delta_1), ..., E_A(\delta_l))$ et $E_P(-r_B)$ à A.

Étape 3 : A déchiffre les éléments $E_A(\delta_i^*)$ et exécute l'algorithme suivant :

A choisit $r_{A,i} \in_R \mathbb{Z}_{N_P}$ $(\forall 1 \leq i \leq l)$, une permutation $\pi_A(\cdot)$ et calcule :

$E_P(\mu_i) = E_P((\delta_i^* + r_{A,i} - r_B) * (1 + r_{A,i})^{-1}) = (E_P(\delta_i^* + r_{A,i}) \cdot E_P(-r_B))^{(1+r_{A,i})^{-1}}$

A envoie $(E_P(\mu_1^*), ..., E_P(\mu_l^*)) = \pi_A((E_P(\mu_1), ..., E_P(\mu_l))$ à P.

Étape 4 : P déchiffre les éléments $E_P(\mu_i^*)$. Si le vecteur μ^* contient la valeur 1, alors $a \geq b$ et $s_{A,1} = 0$. Sinon, $a \leq b$ et $s_{A,1} = 1$. P envoie $E_P(s_{A,1})$ à A.

Étape 5 : B choisit aléatoirement α et β dans $\mathbb{Z}_{N_P}$, les chiffre avec la clé de A et les envoie à A.

Étape 6 : A reçoit les données de B. On définit $s_{B,1} = s_{A,1} - 1$. Elle détermine $E_P(\alpha s_{B,1} + \beta) = (E_P(s_{A,1}) \cdot E_P(-1))^{\alpha} \cdot E_P(\beta)$ et l'envoie à B.

Étape 7 : B calcule $E_P(s_{B,1}) = (E_P(\alpha s_{A,1} + \beta) \cdot E_P(-\beta))^{\alpha^{-1}} = E_P(s_{A,1} - 1)$

Protocole 3.5 : Protocole de comparaison - itération 1

À la fin de cette première itération, P connaît le résultat de la comparaison. Alice possède $E_P(s_{A,1})$ et Bob possède $E_P(s_{B,1})$. Ces deux valeurs sont liées par l'équation $s_{B,1} = s_{A,1} - 1$. Si Alice est gagnante, on a $s_{A,1} = 0$ sinon $s_{A,1} = 1$. Dans les deux cas, le gagnant possède $E_P(0)$. Lors des itérations suivantes, on se sert de cette valeur pour rendre aléatoires les résultats des comparaisons entre les perdants.

Le protocole 3.6 présente le déroulement de l'itération $j > 1$, lorsque des participants ont été éliminés ($\mathscr{P} \neq \emptyset$ dans le protocole 3.4). Il vaut aussi pour la première itération, mais est inutilement plus complexe. Seules les étapes 2 et 3 ont été modifiées.

3.1.4.2 Exactitude

Les éléments du vecteur μ sont de la forme $\mu_i = (\delta_i^* - r_B + s_B + r_{A,i} + s_A) * (1 + r_{A,i})^{-1}$. Par conséquent, si $s_A = s_B = 0$, cela correspond au cas simple à deux individus déjà étudiés. Le protocole est donc correct dans ce cas.

On sait que $r_{A,j} = 0$ si et seulement si Alice a gagné l'itération j. Par conséquent, si Alice a gagné toutes les itérations jusqu'à $j-1$, alors $s_A = \sum_{x=1}^{j-1} s_{A,x} k_{A,x} = 0$. Nous examinerons la réciproque dans la preuve de sécurité. Il en va de même pour s_B. Par conséquent, si Alice et Bob ont gagné toutes les itérations précédentes, en d'autres termes s'ils n'ont jamais été éliminés, alors $s_A = s_B = 0$ et le résultat obtenu est bien celui désiré.

3.1.4.3 Preuves de sécurité

Les objectifs de sécurité qui étaient atteints dans le cas simple à deux individus le sont toujours maintenant. En effet, les éléments de sécurité et les variables aléatoires qui permettaient d'atteindre ces objectifs n'ont pas été modifiés. Cependant, il faut désormais considérer deux autres objectifs :

1. P ne doit pas être en mesure de générer $s_{A,j} = s_{B,j} = 0$, ce qui correspond à n'éliminer personne durant l'itération j.

Données : Les valeurs a et b de A et B. Les clefs publiques N_A et N_P de A et P avec les fonctions de chiffrement associées $E_A(\cdot)$ et $E_P(\cdot)$ respectivement.

Résultat : P détermine $a \geq b$ ou $a \leq b$.

Étape 1 : A chiffre chaque bit a_i $(1 \leq i < l)$ grâce à $E_A(\cdot)$ et envoie l'ensemble des bits chiffrés à B.

Étape 2 : B reçoit les données de A et exécute l'algorithme suivant :

B redéfinit $a_l = \overline{b_l}$.

B choisit aléatoirement $r_B \in_R \mathbb{Z}_{N_A}$ et une permutation $\pi_B(\cdot)$.

B calcule :

pour $i = 1$ **à** l **faire**

$E_A(d_i) = E_A(a_i - b_i) = E_A(a_i) \cdot E_A(-b_i)$.

$E_A(f_i) = E_A(a_i \oplus b_i) = E_A(a_i - 2a_ib_i + b_i) = E_A(a_i)^{1-2b_i} \cdot E_A(b_i)$.

$E_A(\gamma_i) = E_A(2\gamma_{i-1} + f_i) = E_A(\gamma_{i-1})^2 \cdot E_A(f_i)$ avec $\gamma_0 = 0$.

$E_A(\delta_i) = E_A(d_i + r_i(\gamma_i - 1) + r_B) = E_A(d_i) \cdot (E_A(\gamma_i) \cdot E_A(-1))^{r_i} \cdot E_A(r_B)$ avec $r_i \in_R \mathbb{Z}_{N_A}$.

fin

B choisit aléatoirement $k_{B,j-1} \in_R \mathbb{Z}_{N_P}$ et calcule

$E_P(s_B) = E_P(\sum_{x=1}^{j-1} s_{B,x}k_{B,x}) = \prod_{x=1}^{j-1} E_P(s_{B,x})^{k_{B,x}}$.

B envoie $(E_A(\delta_1^*), ..., E_A(\delta_l^*)) = \pi_B(E_A(\delta_1), ..., E_A(\delta_l))$ et $E_P(-r_B + s_B)$ à A.

Étape 3 : A déchiffre l'ensemble des $E_A(\delta_i^*)$ et exécute l'algorithme suivant :

A choisit $r_{A,i} \in_R \mathbb{Z}_{N_P}$ $(\forall 1 \leq i \leq l)$, une permutation $\pi_A(\cdot)$ et calcule :

$E_P(s_A) = E_P(\sum_{x=1}^{j-1} s_{A,x}k_{A,x}) = \prod_{x=1}^{j-1} E_P(s_{A,x})^{k_{A,x}}$

$E_P(\mu_i) = E_P((\delta_i^* + r_{A,i} - r_B + s_B + s_A) * (1 + r_{A,i})^{-1})$

$= (E_P(\delta_i^* + r_{A,i}) \cdot E_P(-r_B + s_B) \cdot E_P(s_A))^{(1+r_{A,i})^{-1}}$

A envoie $(E_P(\mu_1^*), ..., E_P(\mu_l^*)) = \pi_A((E_P(\mu_1), ..., E_P(\mu_l))$ à P.

Étape 4 : P déchiffre les éléments $E_P(\mu_i^*)$. Si le vecteur μ^* contient la valeur 1, alors $a \geq b$ et $s_{A,j} = 0$. Sinon, $a \leq b$ et $s_{A,j} = 1$. P envoie $E_P(s_{A,j})$ à A.

Étape 5 : B choisit aléatoirement α et β dans $\mathbb{Z}_{N_P}$, les chiffre avec la clé de A et les envoie à A.

Étape 6 : A reçoit les données de B. On définit $s_{B,j} = s_{A,j} - 1$. Elle détermine $E_P(\alpha s_{B,j} + \beta) = (E_P(s_{A,j}) \cdot E_P(-1))^{\alpha} \cdot E_P(\beta)$ et l'envoie à B.

Étape 7 : B calcule $E_P(s_{B,j}) = (E_P(\alpha s_{B,j} + \beta) \cdot E_P(-\beta))^{\alpha^{-1}}$

Protocole 3.6 : Protocole de comparaison - itération j

2. P ne doit pas connaître le résultat d'une comparaison faisant intervenir au moins un participant éliminé.

Le premier point est rendu impossible par les étapes 5, 6 et 7. Alice et Bob sont les seuls à pouvoir générer conjointement $E_P(s_{B,j})$ en fonction de $E_P(s_{A,j})$. Autrement, P doit parvenir à déchiffrer $E_A(\alpha)$ et $E_A(\beta)$. L'objectif (1) est donc atteint grâce à la sécurité sémantique du système de Paillier, si la puissance de calcul de P est bornée polynomialement.

Le second point n'est pas impossible, mais très peu probable. En effet, $s_A = \sum_{x=1}^{j-1} s_{A,x}k_{A,x} = 0$ possèdent d'autres solutions que $\forall x, s_{A,x} = 0$. Il y a $1/N_P$ chances qu'Alice choisisse $k_{A,j-1}$ tel que $s_{A,j-1}k_{A,j-1} = -\sum_{x=1}^{j-2} s_{A,x}k_{A,x}$ si $s_{A,j-1} \neq 0$ et $\sum_{x=1}^{j-2} s_{A,x}k_{A,x} \neq 0$. Il en va de même pour s_B. La probabilité que cela apparaisse à la fois sur s_A et sur s_B, ou que $s_A = -s_B$ est donc négligeable. On peut donc considérer que le résultat obtenu est purement aléatoire : le résultat de la comparaison est inconditionnellement protégé, même si P est semi-honnête.

Il faut aussi prouver à nouveau qu'Alice et Bob ne peuvent pas connaître le résultat de la comparaison. Ils connaissent désormais les variables $E_P(s_{A,j})$ et $E_P(s_{B,j})$ respectivement, qui indiquent le résultat. Le résultat est sécurisé grâce à la sécurité sémantique du système de Paillier, si leur puissance de calcul est bornée polynomialement.

3.1.4.4 Analyse de la complexité

Afin de réduire la complexité, les étapes 2 et 5 peuvent être faites simultanément. De plus, le calcul de $\prod_{x=1}^{j-1} E_P(s_{A,x})^{k_{A,x}}$ ne requiert pas $j-1$ exponentiations, mais seulement une puisque le calcul de $\prod_{x=1}^{j-2} E_P(s_{A,x})^{k_{A,x}}$ a déjà été effectué lors de l'itération précédente.

La complexité communicationnelle ci-dessous est déterminée en considérant que P sert d'intermédiaire entre Alice et Bob. Si ce n'est pas le cas, le nombre de bits échangés et de communications est largement réduit.

Pour une seule itération de ce protocole avec seulement deux individus, il y a huit communications qui permettent de transmettre $5l+7$ messages chiffrés. Il y a donc $(10l+14)|N|$ bits

échangés. On rappelle que $|N|$ est la longueur des clés publiques et que les messages chiffrés font $2|N|$ bits avec le système de Paillier.

Cependant, l'étape 1 du protocole est redondante d'une itération à l'autre. En effet, les bits de a, bien que chiffrés, ne changent pas d'une itération à l'autre, ni la clé publique d'Alice. Par conséquent, P peut les stocker lors de la première itération, puis les retransmettre directement à B lors de l'itération suivante. Il n'y a alors plus que $(8l+18)|N|$ bits échangés et sept communications dès la deuxième itération. Par simplicité, nous ne considérerons pas cette amélioration dans le calcul de complexité. Nous obtiendrons ainsi un plafond de la complexité communicationnelle.

La complexité calculatoire est légèrement augmentée : il y a $4l+5$ calculs cryptographiques pour Alice, $5l+8$ pour Bob et $l+1$ pour P. Encore une fois, une grande partie des chiffrements réalisés par Bob peut être faite en amont.

Si on considère maintenant l'ensemble du système, incluant les n participants, il y a $n/2$ protocoles de comparaisons par itération, donc $4n$ communications et $(5l+7)|N|n$ bits échangés par itération. Il y a par ailleurs $\log n$ itérations avant d'aboutir au résultat. Si n n'est pas une puissance de 2 comme nous l'avions supposé, il y a $\lceil \log n \rceil$ itérations. L'ensemble du protocole nécessite donc $8\lfloor n/2 \rfloor \lceil \log n \rceil$ communications et $(10+14)|N|\lfloor n/2 \rfloor \lceil \log n \rceil$ bits échangés dans le cas où les participants doivent communiquer par P.

3.1.4.5 Divulgation d'information

Comme nous l'avons déjà vu, P obtient plus d'information que seulement le propriétaire de la valeur maximale. En effet, il connaît aussi le résultat de certains protocoles de comparaison. Rappelons que les valeurs des témoins correspondent à leur position. On cherche ici à quantifier ces connaissances.

Supposons que le nombre de participants n soit égal à 2^k. Il y a en tout k itérations et 2^{k-1} comparaisons à chaque itération, dont la majorité est illisible par P. En reprenant l'arbre en

figure 3.1, on en déduit que P connaît le résultat de 2^{k-1} comparaisons à la première itération, 2^{k-2} à la deuxième itération, puis 2^{k-3} et ainsi de suite. En tout, P en connaît $\sum_{i=0}^{k-1} 2^i = 2^k - 1 = n - 1$. Il n'y a que $n-1$ comparaisons dont le résultat soit utile pour P.

En comparaison, il a été prouvé [Cormen *et al.* (2009)] qu'un minimum de $n \log n$ comparaisons sont nécessaires pour ordonnancer n valeurs. Par conséquent, à l'issue du protocole, P ne possède pas assez d'information pour trier l'ensemble des valeurs.

On remarquera cependant que si deux participants sont amenés à être comparés une deuxième fois entre eux, ils peuvent alors en déduire qu'ils ne possèdent pas la plus grande valeur. Il faudra donc veiller à ce que deux témoins ne soient pas comparés deux fois entre eux. Dans la mesure du possible, on essayera même de choisir les comparaisons entre les témoins de manière à ce que, connaissant l'ensemble des comparaisons (mais pas le résultat de ces comparaisons), pour chaque témoin, il existe un arbre de comparaison qui fait gagner ce témoin. Ainsi, un adversaire, qui parviendrait à savoir précisément quel témoin est comparé à quel témoin, ne serait pas davantage renseigné sur l'issue de l'ensemble du protocole.

3.2 Adaptation de Lin et Tzeng (2005)

Nous cherchons maintenant à obtenir le même résultat en utilisant le protocole de Lin et Tzeng (2005). Ce protocole requiert moins de calculs que celui de Blake et Kolesnikov (2004) : cela nous permettra donc de réduire la charge calculatoire pour n participants.

Comme précédemment, nous procéderons en deux temps. D'abord nous nous intéresserons au cas où il n'y a que deux témoins et un prouveur. Nous généraliserons ensuite cette approche à n témoins pour répondre au problème initial.

3.2.1 Protocole

Le système de chiffrement utilisé est le système de Paillier. On notera N_A la clé publique d'Alice, $E_A(\cdot)$ sa fonction de chiffrement, N_P la clé publique de P et $E_P(\cdot)$ sa fonction de chiffrement.

Considérons le cas général où on peut se permettre d'établir une communication directe entre Alice et Bob. On considérera aussi que tout le monde est capable de capturer les messages émis par les autres participants. Notons a et b les valeurs respectives d'Alice et Bob, et a_i le i^{eme} bit de poids fort de a. On suppose que ces derniers se sont entendus sur la longueur l en bits de leurs valeurs respectives.

Pour une valeur donnée x, il nous faut introduire deux ensembles T_0^x et T_1^x définis de la manière suivante :

$$T_0^x = \{x_1 x_2 ... x_{i-1} 1 | x_i = 0\}$$
$$T_1^x = \{x_1 x_2 ... x_i | x_i = 1\}$$

D'après l'analyse faite en revue de littérature, pour deux valeurs x et y, $x > y$ si et seulement si T_1^x et T_0^y ont un élément commun. On notera $T_\alpha^x[i]$ l'élément de T_α^x de longueur i bits s'il existe.

Le protocole 3.7 présente une solution possible au problème. Cette solution est une adaptation du protocole de Lin et Tzeng (2005). Les modifications qui y ont été apportées sont similaires à celles apportées au protocole de Blake et Kolesnikov (2004) dans la section précédente. Le raisonnement suivi est le même et seul le résultat final est présenté.

3.2.1.1 Exactitude

Les permutations sont ignorées dans cette section afin de simplifier les notations.

Données : Les valeurs a et b de A et B. Les clefs publiques N_A et N_P de A et P et les fonctions de chiffrement associées $E_A(\cdot)$ et $E_P(\cdot)$ respectivement. Une fonction de hachage $h(\cdot)$.

Résultat : P détermine $a > b$ ou $a \leq b$.

Étape 1 : A exécute l'algorithme suivant :

A détermine l'ensemble T_1^a.

A génère un vecteur γ de longueur l tel que $\gamma_i = h(T_1^a[i])$ si cet élément existe, sinon γ_i est généré aléatoirement.

Les éléments de γ sont chiffrés avec $E_A(\cdot)$ puis transférés à B.

Étape 2 : B exécute l'algorithme suivant :

B détermine l'ensemble T_0^b, choisit aléatoirement $r_B \in_R \mathbb{Z}_{N_A}$ et une permutation $\pi_B(\cdot)$.

B calcule le vecteur δ de longueur l tel que :

$E_A(\delta_i) = E_A(k_i(h(T_1^a[i]) - h(T_0^b[i])) + r_B) = (E_A(\gamma_i)) \cdot E_A(-h(T_0^b[i])))^{k_i} \cdot E_A(r_B)$

où k_i est généré aléatoirement pour chaque élément.

Si $T_0^b[i]$ n'existe pas, δ_i est choisi aléatoirement.

B envoie $(E_A(\delta_1^*), ..., E_A(\delta_l^*)) = \pi_B(E_A(\delta_1), ..., E_A(\delta_l))$ et $E_P(-r_B)$ à A.

Étape 3 : A déchiffre les éléments $E_A(\delta^*[i])$ et exécute l'algorithme suivant :

A choisit $r_{A,i} \in_R \mathbb{Z}_{N_P}$ $(1 \leq i \leq l)$, une permutation $\pi_A(\cdot)$ et calcule :

$E_P(\mu_i) = E_P((\delta_i^* + r_{A,i} - r_B) \cdot {r_{A,i}}^{-1}) = (E_P(\delta_i^* + r_{A,i}) \cdot E_P(-r_B))^{{r_{A,i}}^{-1}}$

A envoie $(E_P(\mu_1^*), ..., E_P(\mu_l^*)) = \pi_A(E_P(\mu_1), ..., E_P(\mu_l))$ à P.

Étape 4 : P déchiffre les éléments $E_P(\mu_i^*)$. Si μ^* contient la valeur 1, alors $a > b$. Sinon, $a \leq b$.

Protocole 3.7 : Protocole de comparaison

L'ensemble $T_1^a \cap T_0^b$ possède un (et un seul) élément si et seulement si $a > b$. Cet élément commun est $T_1^a[i^*] = T_0^b[i^*]$ où i^* est l'indice du premier bit tel que $a_{i^*} = 1$ et $b_{i^*} = 0$. Par conséquent, $\delta_{i^*} = (h(T_1^a[i^*]) - h(T_0^b[i^*]))k_i + r_B = r_B$. On en déduit que $\mu[i^*] = (r_B + r_A - r_B)r_A^{-1} = 1$. Le vecteur final μ contient donc bien la valeur 1.

Maintenant, supposons que $a \leq b$. Dans ce cas, $T_1^a \cap T_0^b = \emptyset$. Par conséquent, $h(T_1^a[i]) = h(T_0^b[i])$ est impossible si on considère que la fonction $h(\cdot)$ est parfaite (sans collision). Le vecteur δ contient des valeurs $xk_i + r_B$ avec x non nulle. Les éléments de μ sont donc de la forme $(xk_i + r_B + r_A - r_B)r_A^{-1} = r_A^{-1}(xk_i + r_A) \neq 1$.

Le résultat obtenu est donc bien celui attendu : $a > b$ si et seulement si μ contient la valeur 1.

3.2.1.2 Preuves de sécurité

Il faut s'assurer que (1) Alice ne peut pas connaître b, (2) Bob ne peut pas connaître a et que (3) P ne peut connaître ni a ni b. Par ailleurs, il faut aussi vérifier que (4) P est le seul à connaître le résultat de la comparaison, (5) que personne ne sait si $a = b$ et (6) que personne ne sait la position du premier bit à différencier a et b.

La liste suivante indique les variables connues par chaque partie à la fin du protocole.

- Alice : $a, \gamma, \delta^*, \pi_A(\cdot)$.
- Bob : $b, k_i, r_B, \pi_B(\cdot)$.
- P : μ^*, r_B.

Rappelons que les valeurs du vecteur δ^* sont aléatoires sauf pour un des éléments qui vaut peut-être r_B. La valeur r_B étant inconnue d'Alice, toutes ces valeurs lui sont indistinguables. On peut prouver que toutes les valeurs possibles de b sont équiprobables. Soit b une valeur possible telle que $a > b$. On peut déterminer i^* telle que $T_1^a[i^*] = T_0^b[i^*]$. On peut choisir sans perte de généralité $\pi_B(\cdot)$ comme étant la fonction identité. On peut calculer la solution suivante :

$$
\begin{aligned}
&r_B = \delta_{i^*}^* \\
&k_i = (\delta_i^* - r_B)(\gamma_i - h(T_0^b[i]))^{-1} && \text{si } T_0^b[i] \text{ existe et } i \neq i^* \\
&\delta_i^* \text{ ne dépend pas de } T_0^b[i] \text{ ni } k_i && \text{si } T_0^b[i] \text{ n'existe pas ou } i = i^*
\end{aligned}
$$

Toute valeur de b peut donc générer équiprobablement le vecteur δ. Si $a \leq b$, la démonstration est similaire, à l'exception de r_B qui ne peut pas être déterminé avec certitude. La démonstration est identique pour n'importe quelle permutation $\pi_B(\cdot)$ choisie. La valeur b de Bob est donc inconditionnellement protégée. Les objectifs (1), (5) et (6) sont ainsi assurés même si Alice est semi-honnête. Cependant, si Alice parvient à déchiffrer $E_P(r_B)$, elle est en mesure de connaître le résultat de la comparaison, et donc d'avoir une information partielle sur b. Mais l'objectif (4) est assuré par la sécurité sémantique du système de Paillier, si la puissance de calcul d'Alice est bornée polynomialement.

Supposons que Bob soit à son tour semi-honnête. Bob ne possède aucune donnée non chiffrée qui dépende de a. Les exigences (2),(4) et (5) sont donc assurées grâce à la sécurité sémantique du système de Paillier, si la puissance de calcul de Bob est bornée polynomialement.

Maintenant, supposons que P soit semi-honnête. Il possède uniquement le vecteur μ^* dont l'une des valeurs est potentiellement 1. De la même manière que précédemment, on peut démontrer que toutes les valeurs possibles de a et b auraient pu donner ce même vecteur μ^* de manière équiprobable (à condition bien sûr que le résultat $a > b$ soit respecté). Les valeurs a et b sont donc inconditionnellement protégées. De plus, la position du 1 est rendue aléatoire par les permutations $\pi_A(\cdot)$ et $\pi_B(\cdot)$. Les objectifs (3),(5) et (6) sont vérifiés. Si P réalise une écoute passive, il est capable de récupérer tous les bits chiffrés de a. Dans ce cas, le protocole est sécurisé grâce à la sécurité sémantique du système de Paillier, si la puissance de calcul de P est bornée polynomialement.

Si P sert d'intermédiaire entre Alice et Bob, il connaît toutes les informations circulant entre les témoins. Étant donné qu'on supposait avant qu'il était déjà capable d'écouter tous les messages, cela ne représente pas un gain d'information supplémentaire. Cependant, il est désormais capable de modifier les messages lors de leurs transmissions. Cependant, s'il modifie les données qui transitent, il ne pourra pas obtenir un résultat exploitable à la fin du protocole.

Les preuves de sécurité face aux collusions sont très similaires à celles que nous avons vues dans notre première solution.

La collusion d'Alice et Bob peut être ignorée. Ensemble, ils peuvent connaître le résultat de la comparaison, ainsi que le premier bit à différencier a et b. Toutefois, connaissant a et b, on ne peut pas les empêcher de réaliser eux-mêmes un protocole de comparaison classique.

La collusion d'Alice et P ne peut pas exploiter le vecteur δ^* grâce à la permutation $\pi_B(\cdot)$. De plus, grâce à la multiplication de chaque élément du vecteur par un k_i différent, Alice et P ne sont pas en mesure de déterminer $h(T_1^a[i]) - h(T_0^b[i])$ sauf bien sûr dans le cas où les hachés sont égaux. Ils ne peuvent donc rien savoir de plus que ce que le résultat leur permet de découvrir.

La collusion de Bob et P ne peut rien exploiter non plus. L'indice du premier bit de l'élément commun (s'il existe) est rendu aléatoire par la permutation $\pi_A(\cdot)$. De plus, ils ne peuvent pas établir de correspondance entre les éléments de δ et ceux de μ. En effet, on a l'équation $\mu_i = {r_{A,i}}^{-1}(\delta_i^* + r_{A,i} - r_B)$, qui ne peut être résolue sans connaître l'ensemble des $r_{A,i}$.

3.2.1.3 Analyse de la complexité

Trois communications sont nécessaires pour que le protocole aboutisse. En notant $|N|$ la taille des clés publiques (et donc $2|N|$ la taille d'un message chiffré avec le système de Paillier) et l la taille des données a et b, un total de $(6l+2)|N|$ bits sont échangés. Si P sert de relais entre Alice et Bob, les communications des étapes 1 et 2 doivent être doublées. Par conséquent, cinq communications sont désormais nécessaires et $(10l+4)|N|$ bits sont échangés.

Le nombre d'opérations cryptographiques dépend de la taille des ensembles T_1^a et T_0^b. Par conséquent, plus a contient de 1 et b de 0, plus nombreuses seront les opérations cryptographiques nécessaires. Les calculs de complexité ci-dessous correspondent donc au pire cas possible où a ne contient que des 1 et b que des 0. Alice réalise $4l$ opérations cryptographiques (chiffrements, déchiffrements et exponentiations modulaires), contre $2l+2$ opérations cryptographiques pour Bob. P réalise quant à lui l déchiffrements.

Cependant, une partie des opérations de chiffrement effectuées par Bob peuvent être faites en amont, avant de recevoir les données d'Alice.

3.2.2 Généralisation à *n* individus

Pour simplifier la représentation, on supposera pour l'instant que n est une puissance de 2.

3.2.2.1 Protocole modifié

Le protocole 3.8 permet de réaliser la première itération de comparaison, c.-à-d. lorsqu' aucun participant n'a été éliminé. Les trois premières étapes sont identiques au cas à deux individus.

Tout comme le protocole modifié de Blake et Kolesnikov (2004), à la fin de cette première itération, P connaît le résultat de la comparaison. Alice possède $E_P(s_{A,1})$ et Bob possède $E_P(s_{B,1})$. Ces deux valeurs sont liées par l'équation $s_{B,1} = s_{A,1} - 1$. Si Alice est gagnante, on a $s_{A,1} = 0$ sinon $s_{A,1} = 1$. Dans les deux cas, le gagnant possède $E_P(0)$. Lors des itérations suivantes, on se sert de cette valeur pour rendre aléatoires les résultats des comparaisons entre les perdants.

Le protocole 3.9 présente le déroulement de l'itération j. Il vaut aussi pour la première itération, mais est inutilement plus complexe. Seules les étapes 2 et 3 ont été modifiées.

L'exactitude et les preuves de sécurité de ce protocole sont identiques aux sections 3.1.4.2 et 3.1.4.3 sur le protocole adapté de Blake et Kolesnikov (2004). La divulgation d'information associée à l'ensemble du protocole à n participants a déjà été étudiée dans la section 3.1.4.5.

3.2.2.2 Analyse de la complexité

L'analyse de la complexité est similaire à celle de la section 3.1.4.4 sur le protocole de Blake et Kolesnikov (2004). Certains éléments de justification ne sont donc pas détaillés ici.

Afin de réduire la complexité, les étapes 2 et 5 peuvent être faites simultanément. De plus, le calcul de $\prod_{x=1}^{j-1} E_P(s_{A,x})^{k_{A,x}}$ ne requiert qu'une seule exponentiation modulaire.

L'étape 1 du protocole est redondante d'une itération à l'autre. En effet, le vecteur γ ne change pas d'une itération à l'autre, ni la clé publique de A. Par conséquent, P peut stocker ce vecteur

Données : Les valeurs a et b de A et B. Les clés publiques N_A et N_P de A et P avec les fonctions de chiffrement associées $E_A(\cdot)$ et $E_P(\cdot)$ respectivement. Une fonction de hachage $h(\cdot)$.

Résultat : P détermine $a > b$ ou $a \leq b$.

Étape 1 : A exécute l'algorithme suivant :

A génère l'ensemble T_1^a et un vecteur γ de longueur l tel que $\gamma_i = h(T_1^a[i])$ si cet élément existe, sinon γ_i est généré aléatoirement.

Les éléments de γ sont chiffrés avec $E_A(\cdot)$ puis transférés à B.

Étape 2 : B exécute l'algorithme suivant :

B détermine l'ensemble T_0^b, choisit aléatoirement $r_B \in_R \mathbb{Z}_{N_A}$ et une permutation $\pi_B(\cdot)$.

B calcule le vecteur δ de longueur l tel que

$E_A(\delta_i) = E_A(k_i(h(T_1^a[i]) - h(T_0^b[i])) + r_B) = (E_A(\gamma_i)) \cdot E_A(-h(T_0^b[i])))^{k_i} \cdot E_A(r_B)$

où k_i est généré aléatoirement pour chaque élément.

Si $T_0^b[i]$ n'existe pas, δ_i est choisi aléatoirement.

B envoie $(E_A(\delta_1^*), ..., E_A(\delta_l^*)) = \pi_B(E_A(\delta_1), ..., E_A(\delta_l))$ et $E_P(-r_B)$ à A.

Étape 3 : A déchiffre les éléments $E_A(\delta_i^*)$ et exécute l'algorithme suivant :

A choisit $r_{A,i} \in_R \mathbb{Z}_{N_P}$ $(1 \leq i \leq l)$, une permutation $\pi_A(\cdot)$ et calcule :

$E_P(\mu_i) = E_P((\delta_i^* + r_{A,i} - r_B) \cdot {r_{A,i}}^{-1}) = (E_P(\delta_i^* + r_{A,i}) \cdot E_P(-r_B))^{{r_{A,i}}^{-1}}$

A envoie $(E_P(\mu_1^*), ..., E_P(\mu_l^*)) = \pi_A(E_P(\mu_1), ..., E_P(\mu_l))$ à P.

Étape 4 : P déchiffre les éléments $E_P(\mu_i^*)$. Si μ contient la valeur 1, alors $a > b$,sinon $a \leq b$. Si $a > b$, P prends $s_{A,1} = 0$, sinon $s_{A,1} = 1$. P envoie $E_P(s_{A,1})$ à A.

Étape 5 : B choisit aléatoirement α et β dans $\mathbb{Z}_{N_P}$, les chiffre avec la clé de A et les envoie à A.

Étape 6 : A déchiffre les données de B, puis, sachant que $s_{B,1} = s_{A,1} - 1$, elle détermine $E_P(\alpha s_{B,1} + \beta) = (E_P(s_{A,1}) \cdot E_P(-1))^{\alpha} \cdot E_P(\beta)$ et l'envoie à B.

Étape 7 : B calcule $E_P(s_{B,1}) = (E_P(\alpha s_{B,1} + \beta) \cdot E_P(-\beta))^{\alpha^{-1}}$.

Protocole 3.8 : Protocole de comparaison - itération 1

chiffré lors de la première itération, puis le retransmettre pour les itérations suivantes. Par simplicité, nous ne considérerons pas cette amélioration dans le calcul de complexité.

Données : Les valeurs a et b de A et B. Les clés publiques N_A et N_P de A et P avec les fonctions de chiffrement associées $E_A(\cdot)$ et $E_P(\cdot)$ respectivement. Une fonction de hachage $h(\cdot)$.

Résultat : P détermine $a > b$ ou $a \leq b$.

Étape 1 : A exécute l'algorithme suivant :

A génère l'ensemble T_1^a et un vecteur γ de longueur l tel que $\gamma_i = h(T_1^a[i])$ si cet élément existe, sinon γ_i est généré aléatoirement.

Les éléments de γ sont chiffrés avec $E_A(\cdot)$ puis transférés à B.

Étape 2 : B exécute l'algorithme suivant :

B détermine l'ensemble T_0^b, choisit aléatoirement $r_B \in_R \mathbb{Z}_{N_A}$ et une permutation $\pi_B(\cdot)$.

B calcule le vecteur δ de longueur l tel que

$E_A(\delta_i) = E_A(k_i(h(T_1^a[i]) - h(T_0^b[i])) + r_B) = (E_A(\gamma_i)) \cdot E_A(-h(T_0^b[i])))^{k_i} \cdot E_A(r_B)$

où k_i est généré aléatoirement pour chaque élément.

Si $T_0^b[i]$ n'existe pas, δ_i est choisi aléatoirement.

B choisit $k_{B,j-1} \in_R \mathbb{Z}_{N_P}$ et calcule $E_P(s_B) = E_P(\sum_{x=1}^{j-1} s_{B,x}k_{B,x}) = \prod_{x=1}^{j-1} E_P(s_{B,x})^{k_{B,x}}$.

B envoie $(E_A(\delta_1^*), ..., E_A(\delta_l^*)) = \pi_B(E_A(\delta_1), ..., E_A(\delta_l))$ et $E_P(-r_B + s_B)$ à A.

Étape 3 : A déchiffre les éléments $E_A(\delta_i^*)$ et exécute l'algorithme suivant :

A choisit aléatoirement $r_{A,i} \in_R \mathbb{Z}_{N_P}$ $(1 \leq i \leq l)$, $k_{A,j-1}$, $\pi_A(\cdot)$ et calcule :

$E_P(s_A) = E_P(\sum_{x=1}^{j-1} s_{A,x}k_{A,x}) = \prod_{x=1}^{j-1} E_P(s_{A,x})^{k_{A,x}}$.

$E_P(\mu_i) = E_P((\delta_i^* - r_B + s_B + s_A + r_{A,i}) \cdot {r_{A,i}}^{-1})$

$= (E_P(\delta_i^* + r_{A,i}) \cdot E_P(s_A) \cdot E_P(-r_B + s_B))^{{r_{A,i}}^{-1}}$

A envoie $(E_P(\mu_1^*), ..., E_P(\mu_l^*)) = \pi_A(E_P(\mu_1), ..., E_P(\mu_l))$ à P.

Étape 4 : P déchiffre les éléments $E_P(\mu_i^*)$. Si μ contient la valeur 1, alors $a > b$,sinon $a \leq b$. Si $a > b$, P prends $s_{A,j} = 0$, sinon $s_{A,j} = 1$. P envoie $E_P(s_{A,j})$ à A.

Étape 5 : B choisit aléatoirement α et β dans $\mathbb{Z}_{N_P}$, les chiffre avec la clé de A et les envoie à A.

Étape 6 : A déchiffre les données de B, puis, sachant que $s_{B,j} = s_{A,j} - 1$, elle détermine $E_P(\alpha s_{B,j} + \beta) = (E_P(s_{A,j}) \cdot E_P(-1))^{\alpha} \cdot E_P(\beta)$ et l'envoie à B.

Étape 7 : B calcule $E_P(s_{B,j}) = (E_P(\alpha s_{B,j} + \beta) \cdot E_P(-\beta))^{\alpha^{-1}}$.

Protocole 3.9 : Protocole de comparaison - itération j

La complexité communicationnelle est déterminée en considérant que P sert d'intermédiaire entre Alice et Bob. Pour une seule itération de ce protocole avec seulement deux individus, il

y a huit communications et $(10l+18)|N|$ bits échangés. Quant à la complexité calculatoire, il y a $4l+6$ calculs cryptographiques pour Alice, $2l+7$ pour Bob et $l+1$ pour P. Encore une fois, une grande partie des chiffrements réalisés par Bob peut être faite en amont. De plus, ces calculs sont réalisés dans les pires cas possibles, c'est à dire dans le cas peu probable ou a ne contient que des 1 et b que des 0.

Si on considère maintenant l'ensemble du système incluant les n participants et la totalité des itérations, le nombre de communications et de bits échangés doit être multiplié par $\lfloor n/2 \rfloor \lceil \log n \rceil$.

3.3 Protocole de comparaison optimisé

Dans cette section, nous allons détailler un troisième protocole de comparaison spécifiquement conçu pour notre contexte. Comme pour les protocoles précédents, il est destiné à comparer les valeurs de deux individus seulement et nous montrerons comment l'appliquer à n individus.

Contrairement aux protocoles vus en revue de littérature, il tient compte du fait qu'il existe une troisième partie P semi-honnête qui sert d'intermédiaire et qui possède le résultat à la fin. De plus, nous souhaitons que les participants ne puissent pas connaître le résultat de la comparaison. Ces distinctions, comparées aux protocoles de comparaison classiques, nous permettent d'alléger considérablement la charge calculatoire et communicationnelle.

3.3.1 Protocole

Le système de chiffrement utilisé est le système de Paillier. On note a et b les valeurs d'Alice et Bob. On suppose que la différence $(a-b)$ tient sur l bits. On note t la taille en bits de certains paramètres de sécurité. Cette variable sera explicitée plus loin.

Dans le protocole 3.10 que nous allons présenter, Alice modifiera au préalable a pour que le bit le moins significatif a_l soit 0. $E_A(a)$ sera ensuite envoyé à Bob, qui calculera lui-même $E_A(a') = E_A(a+\overline{b_l}) = E_A(a) \cdot E_A(\overline{b_l})$. On notera que seul le bit le moins significatif peut différencier a et a'.

Données : Les valeurs a et b de A et B de longueur l bits. Les fonctions de chiffrement $E_A(\cdot)$ de A et $E_P(\cdot)$ de P et N_A la clé publique de A.

Résultat : P détermine $a \geq b$ ou $a \leq b$.

Étape 1 : A met à zéro le bit le moins significatif a_l de a et envoie $E_A(a)$ à B.

Étape 2 : B exécute l'algorithme suivant :

B calcule $E_A(a') = E_A(a + \overline{b_l}) = E_A(a) \cdot E_A(\overline{b_l})$.

B choisit aléatoirement $k_1 \in_R [2^l; 2^t[, \beta \in_R]-2^l; 2^l[$ et $r_B \in_R \mathbb{Z}_{N_A}$.

B calcule $E_A(k_1(a'-b)+\beta+r_B) = (E_A(a') \cdot E_A(-b))^{k_1} \cdot E_A(\beta + r_B)$.

B envoie $E_A(k_1(a'-b)+\beta+r_B)$ et $E_P(-r_B)$ à A.

Étape 3 : A exécute l'algorithme suivant :

A déchiffre $E_A(k_1(a'-b)+\beta+r_B)$ et chiffre avec $E_P(\cdot)$.

A choisit aléatoirement $k_2 \in_R [2^l; 2^t[$ et $\alpha \in_R]-2^l; 2^l[$.

A calcule

$E_P(k_2(k_1(a'-b)+\beta)+\alpha) = (E_P(k_1(a'-b)+\beta+r_B) \cdot E_P(-r_B))^{k_2} \cdot E_P(\alpha)$ et

l'envoie à P.

Étape 4 : P déchiffre $E_P(k_2(k_1(a'-b)+\beta)+\alpha)$. Si $k_2(k_1(a'-b)+\beta)+\alpha > 0$ alors $a \geq b$. Sinon $a \leq b$.

Protocole 3.10 : Protocole de comparaison

Si nous suivons les recommandations actuelles, la taille des clés publiques, et donc celle des messages, est de 2048 bits. La première moitié de ce groupe fini sera réservée aux nombres positifs et la seconde moitié aux nombres négatifs. Ce qui nous laisse 2047 bits pour les nombres positifs et autant pour les négatifs. Si nous ne voulons pas perdre le signe des données, et par la même occasion le résultat recherché, l'inéquation (3.1) doit être respectée :

$$
\begin{aligned}
-2^{2047} < k_2(k_1(a-b)+\beta)+\alpha < 2^{2047} \qquad & (3.1)\\
2^t(2^t 2^l + 2^l) + 2^l \leq 2^{2047} \qquad & \\
2^t(2^t 2^l) \leq 2^{2047} \qquad & \text{en supposant } t \gg l\\
2t + l \leq 2047 \qquad &
\end{aligned}
$$

Dans notre contexte, les témoins sont à moins d'un kilomètre les uns des autres et la position est précise au mètre près. La différence $(a-b)$ peut donc tenir sur dix bits. On peut donc prendre $l = 10$ et $t = 1018$. On a alors :

$$
\begin{aligned}
k_1 &\in_R [2^{10}; 2^{1018}[\qquad & \beta &\in_R]-2^{10}; 2^{10}[\\
k_2 &\in_R [2^{10}; 2^{1018}[\qquad & \alpha &\in_R]-2^{10}; 2^{10}[
\end{aligned}
$$

3.3.1.1 Exactitude

On sait que $a' \neq b$, alors $|k_1(a'-b)| \geq 2^l > \beta$. Étant donné que k_1 est positif, le signe de $k_1(a'-b)+\beta$ est donc le même que celui de $(a'-b)$ et $k_1(a'-b)+\beta \neq 0$. Par la suite, $|k_2(k_1(a'-b)+\beta)| \geq 2^l > \alpha$. Le choix de α n'influe donc pas sur le signe de $k_2(k_1(a'-b)+\beta)+\alpha$. De plus, k_2 étant choisi nécessairement positif, le signe de $k_2(k_1(a'-b)+\beta)+\alpha$ correspond aussi à celui de $(a'-b)$. En définitive, le signe de $(a'-b)$ est bien le même que celui de $k_2(k_1(a'-b)+\beta)+\alpha$ qui ne peut pas être nul. On obtient donc le résultat suivant : si $k_2(k_1(a'-b)+\beta)+\alpha > 0$ alors $a' > b$, mais si $k_2(k_1(a'-b)+\beta)+\alpha < 0$ alors $a' < b$.

Si a et b se différencient sur des bits autres que le bit le moins significatif, il est évident que le changement effectué sur a n'influe pas sur le résultat. On en déduit $a' > b \implies a > b$ et $a' < b \implies a < b$.

Si a et b sont égaux sauf sur le bit le moins significatif, alors $a' = a$ et le résultat est le même que précédemment.

Si $a = b$ et $b_l = 0$, alors $a'_l = 1$ et on obtient donc $a' > b$. À l'inverse, si $b_l = 1$, en suivant le même raisonnement, on obtient $a' < b$.

On peut déduire de ces différents cas que $k_2(k_1(a'-b)+\beta)+\alpha > 0 \implies a' > b \implies a \geq b$ et $k_2(k_1(a'-b)+\beta)+\alpha < 0 \implies a' < b \implies a \leq b$.

3.3.1.2 Preuves de sécurité

Les objectifs de sécurité sont les mêmes que précédemment. Il faut s'assurer que (1) Alice ne peut pas connaître b, (2) Bob ne peut pas connaître a et que (3) P ne peut connaître ni a ni b. Par ailleurs, il faut aussi vérifier que (4) P est le seul à connaître le résultat de la comparaison et (5) que personne ne sait si $a = b$.

La liste suivante indique les variables connues par chaque partie à la fin du protocole.

- $A : a, k_2, \alpha, k_1(a'-b)+\beta+r_B$.
- $B : b, k_1, \beta, r_B$.
- $P : k_2(k_1(a'-b)+\beta)+\alpha$ et r_B grâce à une écoute passive.

Supposons qu'Alice soit semi-honnête. Elle ne peut pas exploiter $k_1(a'-b)+\beta+r_B$ grâce à la variable r_B choisie aléatoirement par Bob. En effet, la variable r_B est beaucoup plus grande que $k_1(a'-b)+\beta$. Les objectifs de sécurité (1), (4) et (5) sont donc atteints même si Alice est semi-honnête. Toutefois, elle connaît $E_P(-r_B)$ et pourrait connaître le résultat de la comparaison si elle parvenait à le déchiffrer. Les objectifs de sécurité sont atteints grâce à la sécurité sémantique du système de Paillier, si la puissance de calcul d'Alice est bornée polynomialement.

Bob ne possède aucune donnée non chiffrée. Les objectifs (2), (4) et (5) sont donc atteints grâce à la sécurité sémantique du système de Paillier, si la puissance de calcul de Bob est bornée polynomialement.

Le résultat connu par P est $k_2(k_1(a'-b)+\beta)+\alpha$, ce qui ne lui permet pas de calculer avec certitude $(a'-b)$. Nous analyserons plus loin la divulgation d'information liée à ce résultat.

On remarquera que le bit le moins significatif de a n'est jamais divulgué. Il est donc impossible pour Bob et P d'établir la relation $a = b$.

Si Alice et P forment une collusion, ils sont en mesure de calculer $k_1(a'-b)+\beta$. Nous analyserons la divulgation d'information liée à ce résultat dans la section suivante.

Si Bob et P forment une collusion, ils sont en mesure de connaître $k_2(k_1(a'-b)+\beta)+\alpha$, k_1 et β. Nous analyserons la divulgation d'information associée à ces données. Bien que nous analysions un modèle semi-honnête, il est tout à fait possible pour Bob de choisir $a'_l = b_l$ plutôt que $\overline{b_l}$, sans empêcher le protocole d'aboutir. Il lui est donc possible de générer $a' = b$ (selon la valeur de a). Ce défaut pourrait être atténué en laissant Alice et Bob concaténer une série de bits aléatoires à a et b. Cependant, nous n'analysons pas le modèle malicieux ici.

En considérant maintenant que Alice et Bob ne peuvent établir de communication directe entre eux et que tous les messages doivent transiter par P, les preuves de sécurité sont les mêmes. En effet, r_B est le seul message que P peut déchiffrer et modifier, mais le modifier empêcherait le protocole d'aboutir.

3.3.1.3 Entropie

On cherche dans cette section à quantifier la divulgation d'information liée aux résultats obtenus par P, par la collusion d'Alice et P ou par la collusion de Bob et P. Les résultats qui nous intéressent sont :

1. $k_2(k_1(a'-b)+\beta)+\alpha$ connu par P.
2. $k_1(a'-b)+\beta$ connu par la collusion d'Alice et P.
3. $k_2(k_1(a'-b)+\beta)+\alpha$ sachant β et k_1 connus par la collusion de Bob et P.

Remarquons que le troisième résultat est équivalent au deuxième. De plus, le deuxième est plus précis que le premier et peut donc révéler plus d'information. Par conséquent, nous étudierons uniquement ce deuxième cas puisque les autres sont soit équivalents, soit plus sécurisés.

Ne connaissant ni k_1, ni β, le résultat $k_1(a'-b)+\beta$ peut être assimilé à une droite $y = k_1x+\beta$ où, sachant y, on cherche à calculer x selon les différentes valeurs possibles de k_1 et β.

On a supposé que la différence $a-b$ faisait au plus l bits. De plus, $a'-b$ ne peut pas être nul. x est donc strictement compris entre -2^l et 2^l et $x \neq 0$. Cependant, rappelons que le k_1 est positif et que β n'influe pas sur le signe de y : le signe de y est donc le même que celui de x. Donc, connaissant y, on peut déterminer le signe de x. Nous nous intéresserons donc uniquement au cas $1 \leq x < 2^l$.

Considérons l'équation suivante :

$$y = k_1x+\beta \tag{3.2}$$

avec les paramètres :

$$\begin{aligned} t &\gg l \\ 2^l \leq k_1 &< 2^t \\ 1 \leq x &< 2^l \\ -2^l < \beta &< 2^l \end{aligned}$$

et l'intervalle de la figure 3.2. L'intervalle $[y-(2^l-1); y+(2^l-1)]$ représente toutes les valeurs possibles de $y-\beta = k_1x$. Par conséquent, pour un x donné, il y a autant de valeurs possibles de k_1 que de points dans l'intervalle.

Plaçons nous pour l'instant dans le cas le plus simple $2^{l+1} \leq y < 2^t$ pour lequel toutes les valeurs de x sont possibles et pour lequel le nombre de solutions (k_1, β) est maximal. Pour un

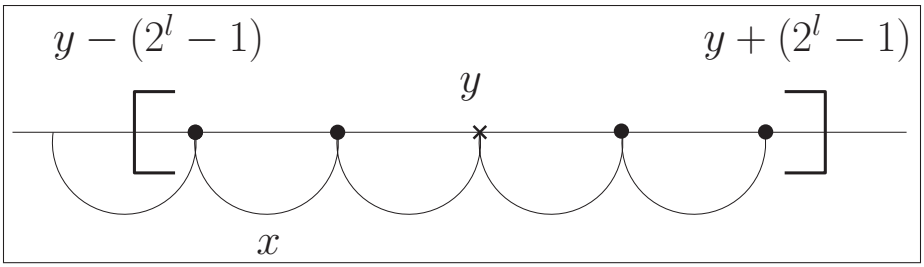

Figure 3.2 $2^{l+1}-1$ - Intervalle

x donné, le nombre de solutions (k_1,β) de l'équation (3.2) est donné par :

$$\left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor$$

Par conséquent, le nombre total de solutions (k_1,β) est donné par :

$$S=\sum_{x=1}^{2^l-1}\left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor$$

On peut donc facilement établir la probabilité $\Pr(X=x|Y=y)$ et l'entropie $H(X|Y=y)$.

$$\Pr(X=x|Y=y)=\frac{\left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor}{S}$$

$$\begin{aligned}
H(X|Y=y) &= \sum_{x=1}^{2^l-1}\Pr(X=x|Y=y)\log\frac{1}{\Pr(X=x|Y=y)} \\
&= \sum_{x=1}^{2^l-1}\frac{\left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor}{S}\log\frac{S}{\left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor} = \frac{1}{S}\sum_{x=1}^{2^l-1}\left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor\left(\log S-\log\left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor\right) \\
&= \frac{\log S}{S}\sum_{x=1}^{2^l-1}\left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor-\frac{1}{S}\sum_{x=1}^{2^l-1}\left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor\log\left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor \\
&= \log S-\frac{1}{S}\sum_{x=1}^{2^l-1}\left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor\log\left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor
\end{aligned}$$

Cependant, k_1 et β étant bornés, le nombre de solutions de l'équation (3.2) connaissant y n'est pas toujours maximal. Par exemple, si $y > 2^t + 2^l$, il n'est pas possible d'avoir $x = 1$. Nous pouvons malgré tout définir un certain nombre d'intervalles optimaux :

Tableau 3.1 Intervalles optimaux

Intervalle	y	x
1	$2^{l+1} \leq y < 2^t$	$1 \leq x \leq 2^l - 1$
2	$2^n + 2^{l+1} \leq y < 2 \cdot 2^t$	$2 \leq x \leq 2^l - 1$
$\cdots$	$\cdots$	$\cdots$
k	$(k-1) \cdot 2^t + 2^{l+1} \leq y < k \cdot 2^t$	$k \leq x \leq 2^l - 1$
$\cdots$	$\cdots$	$\cdots$
$2^l - 1$	$(2^l - 2) \cdot 2^t + 2^{l+1} \leq y < (2^l - 1) \cdot 2^t$	$x = 2^l - 1$

Chacun de ces intervalles contient $(2^t - 2^{l+1})$ valeurs possibles de y. On supposera que tous les $Y = y$ sont équiprobables à l'intérieur d'un intervalle optimal. On peut donc établir :

$$(k-1) \cdot 2^t + 2^{l+1} \leq y < k \cdot 2^t \implies \Pr(Y = y) = \frac{S_k}{(2^t - 2^l)(2^l - 1)(2^{l+1} - 1)}$$

$$\text{avec } S_k = \sum_{x=k}^{2^l - 1} \left\lfloor \frac{2^{l+1} - 1}{x} \right\rfloor$$

Cette probabilité correspond au nombre de solutions de l'équation 3.2 pour un y donné divisé par le nombre total de combinaisons. Il est donc possible de calculer l'entropie conditionnelle :

$$
\begin{aligned}
H(X|Y) &= \sum_{y} \Pr(Y=y)\cdot H(X|Y=y) \\
&\geq \sum_{k=1}^{2^l-1} \frac{(2^t-2^{l+1})S_k}{(2^t-2^l)(2^l-1)(2^{l+1}-1)} \cdot \left[\log S_k - \frac{1}{S_k} \sum_{x=k}^{2^l-1} \left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor \log \left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor \right] \\
&\geq \frac{2^t-2^{l+1}}{(2^t-2^l)(2^l-1)(2^{l+1}-1)} \left[\sum_{k=1}^{2^l-1} S_k \log S_k - \sum_{k=1}^{2^l-1} \sum_{x=k}^{2^l-1} \left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor \log \left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor \right] \\
&\geq \frac{2^t-2^{l+1}}{(2^t-2^l)(2^l-1)(2^{l+1}-1)} \left[\sum_{k=1}^{2^l-1} S_k \log S_k - \sum_{k=1}^{2^l-1} k \left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor \log \left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor \right]
\end{aligned}
$$

Le passage de la première à la deuxième ligne est possible si on considère que tous les y sont équiprobables dans un même intervalle optimal. Remarquons aussi que si t est très largement supérieur à l, alors $2^t - 2^l \approx 2^t - 2^{l+1}$ et le résultat est indépendant de t.

$$
H(X|Y) = \frac{1}{(2^l-1)(2^{l+1}-1)} \left[\sum_{k=1}^{2^l-1} S_k \log S_k - \sum_{k=1}^{2^l-1} k \left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor \log \left\lfloor \frac{2^{l+1}-1}{x} \right\rfloor \right]
$$

Cette entropie peut être calculée pour de petites valeurs de l (voir le tableau 3.2).

Tableau 3.2 $H(X|Y)$ pour de petites valeurs de l

l	$H(X\|Y)$	$H(X\|Y)/l$
10	8.170	0.817
15	12.642	0.843
20	17.116	0.856
25	21.591	0.864
30	26.065	0.869
32	27.855	0.870

On conserve donc plus de 80% des bits d'entropie, ce qui est suffisant dans notre contexte. Le protocole est donc suffisamment sécurisé face aux différentes formes de collusion étudiées.

3.3.1.4 Analyse de la complexité

La complexité à la fois communicationnelle et calculatoire est bien meilleure que celle des autres protocoles. Il faut toujours trois communications, mais seulement quatre données chiffrées sont échangées. En notant $|N|$ la taille de la clé publique, et donc $2|N|$ celle des messages chiffrés, $8|N|$ bits sont échangés au cours du protocole. Si Alice et Bob ne peuvent établir de communication directe, il faut cinq communications et $14|N|$ bits échangés.

Cinq opérations cryptographiques (incluant chiffrements, déchiffrements et exponentiations modulaires) sont nécessaires pour Alice et autant pour Bob, contre seulement un déchiffrement pour P.

L'avantage de ce protocole est que ni la complexité communicationnelle, ni calculatoire ne dépendent de la taille des valeurs a et b.

3.3.2 Généralisation à n individus

Nous pouvons facilement généraliser le protocole précédent à n participants de la même manière que nous l'avons fait pour les protocoles de Blake et Kolesnikov (2004) et Lin et Tzeng (2005).

3.3.2.1 Protocole modifié

Le protocole 3.11 permet de réaliser la première itération de comparaison, c.-à-d. lorsqu'aucun participant n'a été éliminé. Les trois premières étapes sont identiques au cas à deux individus.

À la fin de cette première itération, P connaît le résultat de la comparaison. Alice possède $E_P(s_{A,1})$ et Bob possède $E_P(s_{B,1})$. Ces deux valeurs sont liées par l'équation $s_{B,1} = s_{A,1} - 1$. Si Alice est gagnante, on a $s_{A,1} = 0$ sinon $s_{A,1} = 1$. Dans les deux cas, le gagnant possède $E_P(0)$. Lors des itérations suivantes, on se sert de cette valeur pour rendre aléatoires les résultats des comparaisons entre les perdants.

Données : Les valeurs a et b de A et B de longueur l bits. Les fonctions de chiffrement $E_A(\cdot)$ de A et $E_P(\cdot)$ de P et N_A la clé publique de A.

Résultat : P détermine $a \geq b$ ou $a \leq b$.

Étape 1 : A met à zéro le bit le moins significatif a_l de a et envoie $E_A(a)$ à B.

Étape 2 : B exécute l'algorithme suivant :

B calcule $E_A(a') = E_A(a + \overline{b_l}) = E_A(a) \cdot E_A(\overline{b_l})$.

B choisit aléatoirement $k_1 \in_R [2^{l_x}; 2^{l_k}[, \beta \in_R]-2^{l_x}; 2^{l_x}[$ et $r_B \in_R \mathbb{Z}_{N_A}$.

Il calcule $E_A(k_1(a'-b)+\beta+r_B) = (E_A(a') \cdot E_A(-b))^{k_1} \cdot E_A(\beta + r_B)$.

B envoie $E_A(k_1(a'-b)+\beta+r_B)$ et $E_P(-r_B)$ à A.

Étape 3 : A exécute l'algorithme suivant :

A déchiffre $E_A(k_1(a'-b)+\beta+r_B)$ et chiffre avec $E_P(\cdot)$.

A choisit aléatoirement $k_2 \in_R [2^{l_x}; 2^{l_k}[$ et $\alpha \in_R]-2^{l_x}; 2^{l_x}[$.

A calcule

$E_P(k_2(k_1(a'-b)+\beta)+\alpha) = (E_P(k_1(a'-b)+\beta+r_B) \cdot E_P(-r_B))^{k_2} \cdot E_P(\alpha)$ et l'envoie à P.

Étape 4 : P déchiffre $E_P(k_2(k_1(a'-b)+\beta)+\alpha)$. Si $k_2(k_1(a'-b)+\beta)+\alpha > 0$ alors $a \geq b$. Sinon $a \leq b$. Si $a \geq b$, P prends $s_{A,1} = 0$, sinon $s_{A,1} = 1$. P envoie $E_P(s_{A,1})$ à A.

Étape 5 : B choisit aléatoirement γ et δ dans $\mathbb{Z}_{N_P}$, les chiffre avec la clef de A et les envoie à A.

Étape 6 : A déchiffre les données de B, puis, sachant que $s_{B,1} = s_{A,1} - 1$, il détermine $E_P(\gamma s_{B,1} + \delta) = (E_P(s_{A,1}) \cdot E_P(-1))^{\gamma} \cdot E_P(\delta)$ et l'envoie à B.

Étape 7 : B calcule $E_P(s_{B,1}) = (E_P(\gamma s_{B,1} + \delta) \cdot E_P(-\delta))^{\gamma^{-1}}$.

Protocole 3.11 : Protocole de comparaison - itération 1

Le protocole 3.12 présente le déroulement de l'itération j. Il vaut aussi pour la première itération, mais est inutilement plus complexe. Seules les étapes 2 et 3 ont été modifiées.

Si Alice et Bob ont gagné toutes les itérations, on a $s_A = s_B = 0$. Le résultat calculé est donc exactement le même que pour le cas à deux individus.

Données : Les valeurs a et b de A et B de longueur l bits. Les fonctions de chiffrement $E_A(\cdot)$ de A et $E_P(\cdot)$ de P et N_A la clé publique de A.

Résultat : P détermine $a \geq b$ ou $a \leq b$.

Étape 1 : A met à zéro le bit le moins significatif a_l de a et envoie $E_A(a)$ à B.

Étape 2 : B exécute l'algorithme suivant :

B calcule $E_A(a') = E_A(a + \overline{b_l}) = E_A(a) \cdot E_A(\overline{b_l})$.

B choisit aléatoirement $k_1 \in_R [2^{l_x}; 2^{l_k}[$, $\beta \in_R]-2^{l_x}; 2^{l_x}[$ et $r_B \in_R \mathbb{Z}_{N_A}$.

Il calcule $E_A(k_1(a'-b)+\beta+r_B) = (E_A(a') \cdot E_A(-b))^{k_1} \cdot E_A(\beta + r_B)$.

B choisit aléatoirement $k_{B,j-1} \in_R \mathbb{Z}_{N_P}$ et calcule

$E_P(s_B) = E_P(\sum_{x=1}^{j-1} s_{B,x} k_{B,x}) = \prod_{x=1}^{j-1} E_P(s_{B,x})^{k_{B,x}}$.

B envoie $E_A(k_1(a'-b)+\beta+r_B)$ et $E_P(-r_B + s_B)$ à A.

Étape 3 : A exécute l'algorithme suivant :

A calcule $E_P(s_A) = E_P(\sum_{x=1}^{j-1} s_{A,x} k_{A,x}) = \prod_{x=1}^{j-1} E_P(s_{A,x})^{k_{A,x}}$.

A déchiffre $E_A(k_1(a'-b)+\beta+r_B)$ et chiffre avec $E_P(\cdot)$.

A choisit aléatoirement $k_2 \in_R [2^{l_x}; 2^{l_k}[$ et $\alpha \in_R]-2^{l_x}; 2^{l_x}[$.

A calcule $E_P(k_2(k_1(a'-b)+\beta+s_B)+\alpha+s_A)$ et l'envoie à P.

Étape 4 : P déchiffre $E_P(k_2(k_1(a'-b)+\beta+s_B)+\alpha+s_A)$. Si cette valeur est positive alors $a \geq b$. Sinon $a \leq b$. Si $a \geq b$, P prends $s_{A,j} = 0$, sinon $s_{A,j} = 1$. P envoie $E_P(s_{A,j})$ à A.

Étape 5 : B choisit aléatoirement γ et δ dans $\mathbb{Z}_{N_P}$, les chiffre avec la clef de A et les envoie à A.

Étape 6 : A déchiffre les données de B, puis, sachant que $s_{B,j} = s_{A,j} - 1$, il détermine $E_P(\gamma s_{B,j} + \delta) = (E_P(s_{A,j}) \cdot E_P(-1))^{\gamma} \cdot E_P(\delta)$ et l'envoie à B.

Étape 7 : B calcule $E_P(s_{B,j}) = (E_P(\gamma s_{B,j} + \delta) \cdot E_P(-\delta))^{\gamma^{-1}}$.

Protocole 3.12 : Protocole de comparaison - itération j

L'exactitude et les preuves de sécurité de ce protocole sont identiques aux sections 3.1.4.2 et 3.1.4.3 sur le protocole adapté de Blake et Kolesnikov (2004). La divulgation d'information associée à l'ensemble du protocole à n participants a déjà été étudiée dans la section 3.1.4.5.

3.3.2.2 Analyse de la complexité

L'analyse de la complexité est similaire à celle de la section 3.1.4.4 sur le protocole de Blake et Kolesnikov (2004). Certains éléments de justification ne sont donc pas détaillés ici.

Afin de réduire la complexité, les étapes 2 et 5 peuvent être faites simultanément. De plus, le calcul de $\prod_{x=1}^{j-1} E_P(s_{A,x})^{k_{A,x}}$ ne requiert qu'une seule exponentiation modulaire.

L'étape 1 du protocole est redondante d'une itération à l'autre. En effet, la valeur a ne change pas d'une itération à l'autre, ni la clé publique de A. Par conséquent, P peut stocker cette valeur lors de la première itération, puis la retransmettre pour les itérations suivantes. Par simplicité, nous ne considérerons pas cette amélioration dans le calcul de complexité.

La complexité communicationnelle est déterminée en considérant que P sert d'intermédiaire entre Alice et Bob. Pour une seule itération de ce protocole avec seulement deux individus, il y a huit communications et $28|N|$ bits échangés. Quant à la complexité calculatoire, il y a onze calculs cryptographiques pour Alice, dix pour Bob et deux pour P.

Si on considère maintenant l'ensemble du système incluant les n participants et la totalité des itérations, le nombre de communications et de bits échangés doit être multiplié par $\lfloor n/2 \rfloor \lceil \log n \rceil$.

3.4 Transfert du maximum

Grâce aux trois protocoles présentés dans ce chapitre, P est en mesure de déterminer quel témoin possède la plus grande valeur x_{max}. Ce témoin sera nommé W_{max}. La prochaine étape consiste donc à permettre à P de récupérer cette valeur $E_T(x_{max})$ chiffrée avec la clé du juge. C'est le but de cette section. Les objectifs de sécurité sont les suivants :

- Les participants W_i au nombre de n ne doivent pas savoir s'ils possèdent ou non x_{max}.
- P doit être en mesure de récupérer uniquement $E_T(x_{max})$ et aucune des autres valeurs. Dans notre contexte, cela ne lui serait de toute manière d'aucune utilité sauf s'il collaborait avec T pour découvrir la position de plusieurs participants.

3.4.1 Protocole

Rappelons qu'à la fin du protocole de comparaison, tous les participants W_i possèdent un élément aléatoire de la forme $\sum_{j=1}^{\lceil \log n \rceil}(s_{W_i,j}k_{W_i,j})$ chiffré avec la clé de P. Cette somme est nulle uniquement pour le propriétaire du maximum W_{max}, elle est aléatoire pour les autres témoins. Les objectifs précédents peuvent être atteints à l'aide du protocole 3.13.

Données : Les valeurs x_i et $\sum_{j=1}^{\lceil \log n \rceil}(s_{W_i,j}k_{W_i,j})$ des témoins W_i. La clé publique N_P de P. P connaît lequel des W_i est W_{max}.

Résultat : P obtient $E_T(x_{max})$.

Étape 1 : Tous les témoins W_i génère un nombre aléatoire $\alpha_i \in \mathbb{Z}^*_{N_P}$. Ils calculent $E_P(\alpha_i + \sum_{j=1}^{\lceil \log n \rceil}(s_{W_i,j}k_{W_i,j}))$ et l'envoient à P.

/* Notons que $E_P(\alpha_{max} + \sum_{j=1}^{\lceil \log n \rceil}(s_{W_{max},j}k_{W_{max},j})) = E_P(\alpha_{max})$ puisque $s_{W_{max},j} = 0$ pour tout j.*/

Étape 2 : P déchiffre la donnée provenant de W_{max}, il obtient α_{max}. Il transmet ensuite à l'ensemble des témoins $E_T(\alpha_{max})$.

Étape 3 : Tous les W_i reçoivent $E_T(\alpha_{max})$ et calculent $E_T(\alpha_{max} - \alpha_i + x_i)$. Pour W_{max}, on obtient donc $E_T(x_{max})$.

Étape 4 : P reçoit $E_T(x_{max})$ et change l'aléa du chiffrement.

Protocole 3.13 : Protocole de transfert de maximum

3.4.2 Exactitude

On considère uniquement le cas de W_{max}. À la première étape, étant donné que la somme $\sum_{j=1}^{\lceil \log n \rceil}(s_{W_{max},j}k_{W_{max},j}) = 0$, on a $E_P(\alpha_{max} + \sum_{j=1}^{\lceil \log n \rceil}(s_{W_{max},j}k_{W_{max},j})) = E_P(\alpha_{max})$. Le reste du protocole est évident. P obtient donc bien $E_T(x_{max})$

3.4.3 Preuves de sécurité

Il faut s'assurer que (1) les témoins ignorent s'ils possèdent ou non la valeur maximale, que (2) le prouveur ne puisse pas récupérer de valeur déchiffrée et que (3) le prouveur ne puisse pas récupérer $E_T(x_i) \neq E_T(x_{max})$.

Supposons que les témoins sont semi-honnêtes. Toutes les données reçues par les témoins sont chiffrées soit avec la clé de P, soit avec celle du juge. De plus, l'aléa de l'algorithme de chiffrement est modifié dans la dernière étape du protocole. Les témoins n'ont donc aucun moyen de savoir s'ils possèdent ou non la valeur maximale, même si on leur transmet $E_T(x_{max})$ après le protocole. Le premier objectif est donc atteint grâce à la sécurité sémantique du système de Paillier, si la puissance de calcul des témoins est bornée polynomialement.

Les valeurs $E_T(x_i)$ (autres que $E_T(x_{max})$) sont protégées par l'ajout de la variable aléatoire $\sum_{j=1}^{\lceil \log n \rceil}(s_{W_i,j}k_{W_i,j})$, dont personne ne connaît le résultat déchiffré. Il est donc impossible pour P de déterminer α_i et par conséquent de récupérer la valeur correspondante $E_T(x_i)$. Les valeurs $E_T(x_i)$ sont donc inconditionnellement protégées.

La seule valeur que peut obtenir P est $E_T(x_{max})$ qui est protégée par la sécurité sémantique du système de Paillier. L'objectif (2) est atteint si la puissance de P est bornée polynomialement.

3.4.4 Analyse de la complexité

Pour réduire le nombre de communications, l'étape 1 de ce protocole peut être faite lors de la dernière itération du protocole de comparaison.

On suppose que la diffusion du même message à n individus requiert une seule communication et non pas n. Le protocole de transfert de maximum nécessite ainsi $n+1$ communications. En notant $|N|$ la taille des clés publiques, il y a $(4n+2)|N|$ bits échangés.

Chaque témoin a besoin de réaliser deux chiffrements, contre un chiffrement et un déchiffrement pour P.

3.5 Comparaison des protocoles et conclusion

Le tableau 3.3 récapitule les différents protocoles présentés dans le cas général à n individus. Les différentes complexités ne valent que pour une seule instance des protocoles. Autrement, il faut multiplier ces résultats par $\lfloor n/2 \rfloor \lceil \log n \rceil$. L'algorithme de chiffrement utilisé et le système de Paillier. La longueur de la clé publique est notée $|N|$.

Pour la complexité communicationnelle, on suppose que toutes les communications doivent passer par P.

Tableau 3.3 Récapitulatif des protocoles de comparaison

Protocole	**Calculs cryptographiques**			**Bits échangés**
	A	B	P	
Blake et Kolesnikov (2004) adapté	$4l+5$	$5l+8$	$l+1$	$(10l+14)\|N\|$
Lin et Tzeng (2005) adapté	$4l+6$	$2l+7$	$l+1$	$(10l+18)\|N\|$
Protocole optimisé	11	10	2	$28\|N\|$

Quelque soit le protocole utilisé, huit communications sont nécessaires avant d'obtenir un résultat. De plus, il faut aussi ajouter la complexité liée au protocole de transfert de maximum si P souhaite connaître $E_T(x_{max})$.

Pour l'ensemble des protocoles, incluant les n participants, la complexité de nos protocoles est en $O(l \cdot n \log n)$ ou en $O(n \log n)$ contrairement à la plupart des autres protocoles qui sont en $\Omega(l \cdot n^2)$. Le gain de temps de calcul et de communication est donc évident.

Du point de vue de la sécurité, le protocole de Lin et Tzeng (2005) est équivalent à celui de Blake et Kolesnikov (2004). Il n'y a donc pas de réel avantage à choisir celui de Blake et Kolesnikov (2004). Pour ces deux protocoles, il ne peut y avoir aucune divulgation d'information, pas même face à une collusion.

Comme nous l'avons vu par le calcul et l'analyse de l'entropie, ce n'est pas le cas de notre protocole optimisé qui divulgue une certaine quantité d'information. L'analyse de l'entropie

nous a permis de quantifier cette divulgation. Dans notre contexte, on peut espérer que plus de 80% des bits d'entropie seront conservés à l'issue du protocole.

Le choix du protocole à exécuter dépend donc des contraintes de sécurité et des capacités de calculs et de communications. Dans notre contexte, la principale difficulté est la communication. De plus, on peut se permettre de divulguer un peu d'information si cela nous permet de réduire la charge communicationnelle et ainsi le temps de calcul. Dans notre contexte de preuves de localisation, le protocole optimisé sera donc recommandé.

CHAPITRE 4

PROTOCOLE DE CALCUL DE MAXIMUM BIT À BIT

Dans ce chapitre, nous avons pour but de permettre à P de découvrir la valeur maximale x_{max} détenue par les témoins W_i $(1 \leq i \leq n)$. La méthode présentée est une solution alternative aux solutions du chapitre précédent, les objectifs étant sensiblement différents. En effet, nous considérons que P ne cherche plus à savoir quel est le témoin qui possède la plus grande valeur, mais qu'il cherche à connaître directement cette valeur. Par ailleurs, nous considérons que les témoins peuvent aussi découvrir aussi la valeur maximale, mais pas son possesseur.

Cette méthode se distingue des protocoles de calcul de maximum évoqués dans la revue de littérature par une complexité moins élevée au détriment d'une moins bonne sécurité face aux collusions.

4.1 Signatures de groupe

Tout d'abord, nous aurons besoin d'un système de signature de groupe. Ce système est constitué de trois algorithmes, notés $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Vrf})$ et définis de la manière suivante :

- $\mathrm{Gen}(\lambda) = (v, s_k)$ génère une clé publique de groupe v et une clé de signature s_k par membre du groupe à partir d'un paramètre de sécurité λ. Cette fonction nécessite l'intervention des autorités responsables du groupe.
- $\mathrm{Sig}(s_k, m) = \sigma$ génère la signature σ du message m avec la clé de signature s_k.
- $\mathrm{Vrf}(v, m, \sigma) \mapsto \{0, 1\}$ retourne 1 si et seulement si la signature σ correspond bien au message m avec la clé publique de groupe v.

On supposera dans le reste du mémoire que les témoins W_i possèdent déjà une clé de signature s_i et une clé de vérification v_i.

Nous définissons dans cette section les propriétés que doit posséder ce système :

- **Correcte** : La vérification d'une signature est correcte si et seulement si la signature est issue d'un membre du groupe.
- **Non-forgeable** : Les membres du groupe sont les seuls à pouvoir générer des signatures de groupe valides.
- **Anonyme** : Les membres du groupe ne peuvent pas déterminer l'identité d'un signataire sans l'aide des autorités (en l'occurrence le juge).
- **Traçabilité** : Les autorités sont capables de déterminer l'identité d'un signataire.
- **Non-reliable** (*unlinkable*) : Étant donnés deux messages et leur signature respective, il n'est pas possible de déterminer si ces deux signatures ont été générées par la même personne.
- **Unique** : pour un membre du groupe donné, il ne lui est pas possible de générer deux signatures différentes d'un même message.

La solution proposée par Franklin et Zhang (2012) répond à l'ensemble de ces spécifications.

4.2 Initialisation

Le protocole de calcul de maximum que nous proposons dans ce chapitre suppose qu'un certain nombre de tâches soient préalablement appliquées. Les témoins doivent connaître le nombre total de témoins n qui participent au protocole. De plus, ceux-ci doivent être ordonnés de façon claire et indéniable. En effet, l'ordre des témoins sera important. On suppose aussi que les horloges des témoins et du prouveur sont synchronisées. On pourra accepter des erreurs de quelques millisecondes quant à la synchronisation.

Pour atteindre ces objectifs, on peut imaginer les étapes suivantes présentées dans le protocole 4.1.

C'est une méthode simple qui permet d'initialiser correctement le système. Les témoins sont ordonnés de manière claire et le nombre total de témoins correspond au nombre de signatures qu'ils reçoivent.

Données : La clé de signature s_i et de vérification v_i de chaque témoin W_i.

Résultat : Le nombre n et l'ordre des témoins.

Étape 1 : P diffuse une requête à l'instant t :

" Je voudrais des preuves de ma localisation à l'instant t "

Étape 2 : Chaque témoin W_i qui le désire exécute l'algorithme suivant :

Il vérifie que t est cohérent avec leur horloge.

Il génère une clé publique N_i éphémère.

Il renvoie $\sigma_i = \mathrm{Sig}(t, s_i)$ et sa clé publique N_i à P.

Étape 3 : P collecte les signatures de groupe σ_i et les clés publiques N_i et diffuse le tout à l'ensemble des témoins.

Étape 4 : Chaque participant (y compris P) exécute l'algorithme suivant :

Il vérifie que les signatures de groupes sur t sont différentes deux à deux.

Il vérifie que toutes les signatures sont valides.

Le nombre de témoins n correspond au nombre de signatures reçues.

Les témoins seront désormais ordonnés par ordre croissant de σ_i.

Protocole 4.1 : Protocole d'initialisation

De plus, le fait de vérifier que les signatures de groupe σ_i sont différentes deux à deux empêche un des témoins ou le prouveur de forger plusieurs identités. En effet, on rappelle que le système de signature de groupe doit posséder la propriété d'unicité. Toutefois, il est possible pour P de voler l'identité d'un témoin W_i. En effet, il suffit qu'il génère une clé de Paillier N_P de son choix et qu'il la transmette à la place de N_i lors de l'étape 3. P cumule donc à la fois les rôles du prouveur et d'un témoin. Toutefois, W_i peut détecter que sa clé a été modifiée et peut donc choisir de ne plus participer au protocole et de ne pas délivrer de preuve. On verra lors du chapitre suivant que s'il manque des preuves par rapport au nombre de témoins, le juge considère que l'ensemble des preuves est irrecevable.

Ce protocole d'initialisation requiert $n+2$ communications. Si on note $|N|$ la taille des clés de Paillier et s la taille d'une signature, il y a $(s+|N|)n$ bits échangés sans compter la requête émise par P.

4.3 Protocole de calcul de maximum

Maintenant que les témoins sont ordonnés et possèdent les clés de chiffrement de l'ensemble des témoins, ils sont en mesure de procéder au protocole de calcul de maximum. Le principe de celui-ci est assez simple : pour chaque bit j, les témoins choisissent un témoin de chiffrement noté W_J, dont la clé sera utilisée pour calculer $\bigvee_{i=1}^{n} x_{i,j}$. W_J déchiffrera ensuite le résultat, qui sera transmit via P à l'ensemble des témoins qui sauront d'eux-mêmes s'ils sont éliminés ou non.

On note x_{max} la valeur maximale qu'on cherche à calculer et $x_{i,j}$ le j^{eme} bit significatif de x_i. On suppose que les témoins se sont entendus sur la longueur l des données. Le protocole 4.2 présente cette solution.

L'étape 1 du protocole sera expliquée plus tard, en tenant compte des contraintes de sécurité.

4.3.1 Exactitude

L'exactitude de ce protocole peut être démontrée assez simplement. Si $k \sum_{i=1}^{n} x_{i,j} = 0$, alors il est très probable que $x_{i,j} = 0$ pour tout i. Par conséquent, si $k \sum_{i=1}^{n} x_{i,j} = 0$, alors $x_{max,j} = 0$.

Dans le cas inverse, si $k \sum_{i=1}^{n} x_{i,j} \neq 0$, alors il est évident qu'il existe au moins un $x_{i,j} \neq 0$. Par conséquent, on a $x_{max,j} = 1$. Dans ce second cas, il faut éliminer du protocole les témoins qui possédaient $x_{i,j} = 0$. C'est le but de l'étape 7, les témoins ne sont pas éliminés, mais ils continuent le protocole en envoyant que des zéros, ce qui n'influe pas sur le résultat final.

On obtient donc bien le résultat recherché.

4.3.2 Preuves de sécurité

Dans nos objectifs de sécurité, il faut s'assurer que (1) les valeurs x_i de chaque individu ne sont pas divulguées, ni même partiellement, et que (2) le possesseur du maximum reste inconnu.

Données : Pour chaque témoin W_i, sa valeur x_i, sa clé de chiffrement N_i et sa fonction de chiffrement associée $E_i(\cdot)$.

Résultat : Les témoins et P connaissent x_{max}.

pour $j = 1$ **à** l **faire**

Les témoins exécutent l'algorithme suivant :

Étape 1 : Les W_i choisissent la témoin de chiffrement W_J.

Étape 2 : Chaque W_i exécute l'opération suivante :

Si $x_{i,j} = 1$, W_i remplace $x_{i,j}$ par une valeur aléatoire dans $\mathbb{Z}^*_{N_J}$.

Étape 3 : W_i envoie $E_J(x_{i,j})$ à P.

Étape 4 : P choisit aléatoirement $k \neq 0$, additionne toutes les données reçues

$$E_J(k\sum_{i=1}^{n} x_{i,j}) = (\prod_{i=1}^{n} E_J(x_{i,j}))^k$$

et envoie cette somme à W_J.

Étape 5 : W_J déchiffre cette somme et obtient le message m et l'aléa r qui sont envoyés à P.

Étape 6 : P vérifie que $E_J(m,r) = E_J(k\sum_{i=1}^{n} x_{i,j})$. Si $m = 0$, alors $x_{max,j} = 0$, sinon $x_{max,j} = 1$. P envoie $x_{max,j}$ à l'ensemble des témoins.

Étape 7 : Chaque témoin W_i exécute l'algorithme suivant :

si $x_{max,j} = 1$ *et* $x_{i,j} = 0$ **alors**

w_i est éliminé. Il continue le protocole avec $x_i = 0$.

fin

Protocole 4.2 : Calcul de maximum

Toutes les données que possède P sont chiffrées, sauf chaque bit de x_{max}. Les valeurs x_i et l'identité de W_{max} sont donc protégées grâce à la sécurité sémantique du système de Pailler, si la puissance de calcul de P est bornée polynomialement. Si P souhaite déchiffrer un bit spécifique $E_J(x_{i,j})$, il doit l'envoyer pour déchiffrement à W_J, mais il risque alors de ne plus pouvoir connaître le résultat, puisque W_J refusera de déchiffrer une autre donnée. P connaît aussi le résultat $\sum_{i=1}^{n} x_{i,j}$, mais cette valeur est entièrement aléatoire s'il existe au moins un témoin

dont le bit vaut 1 (auquel cas $x_{i,j}$ est remplacé par une valeur aléatoire). Par conséquent, P n'est ni en mesure de déterminer les valeurs $x_{i,j}$, ni de déterminer combien de témoins possèdent le bit 1 (sauf dans le cas particulier où il n'y en a pas).

Les témoins autres que W_J ne reçoivent aucune donnée, mis à part le maximum qui est révélé bit après bit. Ils ne peuvent donc rien apprendre. S'ils sont capables de réaliser une écoute passive et d'obtenir les bits chiffrés des autres individus, les objectifs sont atteints grâce à la sécurité sémantique du système de Paillier, si la puissance de calcul des témoins est bornée polynomialement.

W_J est en mesure de connaître le résultat avant les autres, mais les étapes 5 et 6 l'empêche de mentir sur le résultat. Par ailleurs, il connaît la valeur de $k \sum_{i=1}^{n} x_{i,j}$, mais grâce à k, il ne peut pas détecter si $x_{J,j} = \sum_{i=1}^{n} x_{i,j}$. En d'autres termes, il ne peut pas savoir s'il y a une autre personne que lui-même qui possède $x_{J,j} = 1$.

Dans le cas d'un témoin W_a semi-honnête augmenté, il peut envoyer des bits $x'_{a,j} = 0$ quelque soit la véritable valeur de x_a. Cela lui permet de fausser le résultat de $\bigvee_{i=1}^{n} x_{i,j}$ et d'obtenir $\bigvee_{i=1;i\neq a}^{n} x_{i,j}$. Le résultat du protocole risque cependant d'être faussé.

En cas de collusion entre P et W_J, ils sont en mesure de déchiffrer le j^{eme} de la valeur de chacun des témoins. Avec un tel protocole, cela ne peut pas être empêché, mais, dans notre contexte, nous sommes prêts à l'accepter s'ils ne peuvent pas découvrir les autres bits. La difficulté du protocole porte donc sur le choix du témoin de chiffrement W_J parmi l'ensemble des témoins. Ce choix sera analysé puis expliqué dans la section suivante.

Les collusions entre deux témoins, ou davantage, ne leur permettent pas d'apprendre plus d'informations. En effet, ils ne reçoivent aucune donnée autre que le maximum révélé bit à bit.

4.3.3 Choix du témoin de chiffrement

Comme nous l'avons vu, nous voulons empêcher P et un des témoins (noté W_C) de collaborer pour connaître tous les bits de chacune des valeurs. Pour cela, il faut donc que P et W_C ne puissent pas influencer (ou pas plus que les autres) sur le choix du témoin de chiffrement W_J. On veut éviter que W_C soit toujours choisi comme témoin de chiffrement. De plus, on apprécierait que ce choix ne génère pas de communication supplémentaire.

La solution proposée ici s'appuie sur les éléments obtenus durant la phase d'initialisation. Soit $h(\cdot)$ une fonction de hachage. Pour le j^{eme} bit, W_J sera choisi de la manière suivante :

$$J = h(t + j) \pmod{n}$$

Pour rappel, t était un élément de la requête envoyée par P pour amorcer le protocole.

Cette méthode ne génère pas de communication et une quantité de calculs négligeable devant les algorithmes de chiffrements.

Du point de vue de la sécurité, P a très peu de choix sur t puisque les horloges de tous les participants sont synchronisées et qu'ils vérifient cette variable. Si malgré tout il décide de choisir t pour favoriser W_C, il doit le faire avant de connaître n. En effet, n dépend du nombre de participants qui répondent favorablement à la requête de P. Ni P ni aucun témoin ne peut donc véritablement influencer le choix en sa faveur. P peut toutefois ignorer certaines réponses positives pour réduire la valeur de n.

La sécurité du protocole dépendra donc principalement de n. Il est évident que plus n est grand, moins W_C aura de chance d'être choisi comme témoin de déchiffrement.

Pour résumer, voici le protocole final en tenant compte de ces améliorations.

Seul le choix de W_J a changé par rapport au protocole précédent et nous avons déjà prouvé que le protocole était exact quelque soit le choix de W_J.

Données : L'instant publique t une fonction de hachage $h(\cdot)$, la valeur x_i, la clé de chiffrement n_i de W_i $(1 \leq i \leq n)$ et sa fonction de chiffrement $E_i(\cdot)$.

Résultat : Les témoins et P connaissent x_{max}.

pour $j = 1$ **à** l **faire**

Les témoins exécutent l'algorithme suivant :

Étape 1 : W_i calculent $J = h(t + j) \pmod{n}$.

Étape 2 : Chaque W_i exécute l'opération suivante :

Si $x_{i,j} = 1$, W_i remplace $x_{i,j}$ par une valeur aléatoire dans $\mathbb{Z}^*_{N_J}$.

Étape 3 : W_i envoie $E_J(x_{i,j})$ à P.

Étape 4 : P choisit aléatoirement $k \neq 0$, additionne toutes les données reçues

$$E_J(k\sum_{i=1}^{n} x_{i,j}) = (\prod_{i=1}^{n} E_J(x_{i,j}))^k$$

et envoie cette somme à W_I.

Étape 5 : W_J déchiffre cette somme, on note m et r le message et l'aléa déchiffrés. m et r sont envoyés à P.

Étape 6 : P vérifie que $E_J(m,r) = E_J(k\sum_{i=1}^{n} x_{i,j})$. Si $m = 0$, alors $x_{max,j} = 0$, sinon $x_{max,j} = 1$. P envoie $x_{max,j}$ à l'ensemble des témoins.

Étape 7 : Chaque témoin W_i exécute l'algorithme suivant :

si $x_{max,j} = 1$ *et* $x_{i,j} = 0$ **alors**

w_i est éliminé. Il continue le protocole avec $x_i = 0$.

fin

Protocole 4.3 : Calcul de maximum corrigé

4.3.4 Analyse de la complexité

Dans notre contexte, lorsque P diffuse $x_{max,j}$ à l'ensemble des témoins, nous supposerons que cela nécessite une seule communication sans-fil. Si ce n'est pas le cas, il faudra évidemment en compter n.

Avec cette supposition, il faut $(n+3)l$ communications et $(2n+8)l|N|$ bits échangés, en notant $|N|$ la taille de la clé utilisée avec le système de Paillier. À cette complexité, il faut ajouter celle du protocole d'initialisation, soit $(s+|N|)n$ bits échangés et $n+2$ communications, en notant n la taille des clés de chiffrement et s la taille d'une signature de groupe.

Pour chaque bit, les témoins doivent réaliser un chiffrement, le témoin de chiffrement W_J doit réaliser un déchiffrement de plus. De son côté, P doit réaliser un chiffrement, une exponentiation et $n-1$ multiplications modulaires. Il faut donc multiplier ces résultats par l pour obtenir la complexité de la totalité du protocole. Cependant, le témoin de chiffrement n'est pas toujours le même.

4.3.5 Conclusion

Dans ce chapitre, nous sommes parvenus à implémenter un protocole de calcul de maximum, dont la complexité n'est pas polynomiale. En effet, qu'il s'agisse de la complexité communicationnelle ou calculatoire, notre protocole est en $O(l \cdot n)$ contrairement à la plupart des autres protocoles qui sont en $O(l \cdot n^2)$.

Cependant, nous avons vu qu'en cas de collusion entre P et un des témoins, il y a un risque de divulguer quelques bits des valeurs x_i de chaque témoin. Dans notre contexte, c'est un risque acceptable si cela permet de réduire le temps de calcul.

CHAPITRE 5

GÉNÉRATION DES PREUVES DE LOCALISATION

Dans ce chapitre, nous supposons que le prouveur P possède la valeur $E_T(x_{max})$ (obtenu grâce à un protocole de comparaison appliqué à un arbre binaire) ou x_{max} (obtenu grâce à un protocole de calcul de maximum bit à bit). Cette valeur représente une approximation de la position de P. Cependant, dans sa forme actuelle, elle n'est pas suffisante pour convaincre le juge T de la position de P. En effet, rien n'empêche P jusqu'ici de choisir lui-même une valeur x'_{max} falsifiée.

Par conséquent, il est nécessaire que les témoins donnent une preuve que x_{max} représente bien la valeur maximale des témoins.

Deux protocoles sont présentés dans ce chapitre : le premier doit être utilisé après un protocole de comparaison appliqué à un arbre binaire, tandis que le second doit être utilisé après un protocole de calcul de maximum bit à bit. Il faut noter cependant que le premier protocole peut traiter sans distinction les deux cas, mais est inutilement trop complexe s'il est utilisé après un protocole de calcul de maximum bit à bit.

Dans ce chapitre, nous avons besoin d'un système de signature de groupe. Les contraintes sont les mêmes que pour le chapitre précédent (voir la section 4.1). Le système de chiffrement sera noté $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Vrf})$. On suppose que tous les participants possèdent déjà une clé de signature et une clé unique de groupe. Nous supposons enfin que la signature $\mathrm{Sig}(s_p, t)$, le moment présent signé avec la clé du prouveur, est connue de tous. Cette signature sera par exemple transmise lors de la requête initiale faite par le prouveur.

5.1 Protocole de signature dans le cas d'un protocole de comparaison avec arbre binaire

Dans cette section, nous supposons qu'un protocole de comparaison appliqué à un arbre binaire a été utilisé pour obtenir $E_T(x_{max})$ connu de P. L'objectif est de permettre aux témoins de

générer une preuve que x_{max} est bien la valeur maximale parmi les témoins, tout en gardant la valeur déchiffrée x_{max} inconnue de tous (sauf du juge).

Pour un témoin W_i, on note x_i sa valeur et s_i sa clé de signature de groupe. On note nb_it_i le nombre d'itérations (c.à-d. le nombre de protocoles de comparaison) effectuées par W_i. Remarquons que ce nombre d'itérations est connu à la fois de W_i et de P. Tous les témoins W_i connaissent également l'instant t_i auquel ils ont reçu la requête de preuves de localisation de P
.

On suppose que la différence $(x_{max} - x_i)$ est un entier de moins de $l_x = 10$ bits. Si on donne un numéro à chaque mètre, il y a donc au plus un kilomètre entre le témoin le plus proche de P et le témoin le plus éloigné. P et l'ensemble des témoins suit le protocole 5.1.

Données : Le maximum $E_T(x_{max})$ connu par P, les valeurs x_i, $nb\_it_i, t_i, \mathrm{Sig}(s_p, t)$ et la clé de signature s_i des témoins W_i. La clé publique N_T et la fonction de chiffrement $E_T(\cdot)$ du juge.

Résultat : P obtient une preuve par témoin.

Étape 1 : P redéfinit $E_T(x_{max}) \leftarrow E_T(x_{max}) \cdot r^{N_T}$ où r est choisi aléatoirement. Il envoie $E_T(x_{max})$ à l'ensemble des témoins.

Étape 2 : L'ensemble des témoins W_i exécute l'algorithme suivant :

- W_i choisit aléatoirement $k_i \in_R [2^{l_x}; 2^{l_k}[$ et $r_i \in_R]-2^{l_x}; 2^{l_x}[$.
- W_i calcule $E_T(k_i(x_{max} - x_i) + r_i) = (E_T(x_{max}) \cdot E_T(-x_i))^{k_i} \cdot E_T(r_i)$.
- W_i signe $S_i = \mathrm{Sig}(s_i, [E_T(x_{max}), E_T(k_i(x_{max} - x_i) + r_i), t_i, nb\_it_i, \mathrm{Sig}(s_p, t)])$
- W_i envoie $E_T(k_i(x_{max} - x_i) + r_i), t_i, S_i$ à P.

Étape 3 : P reçoit l'ensemble des preuves et les conserve.

Protocole 5.1 : Protocole de signature des preuves

En suivant les recommandations concernant la génération des clés pour le système de Paillier, nous prenons des clés de 2048 bits. Avec $l_x = 10$, on peut donc prendre $l_k = 2037$.

À la fin de ce protocole, P dispose d'autant de preuves que de témoins. On remarquera que pour W_{max}, $k_i(x_{max} - x_i) + r_i = r_i \in]-2^{l_x}; 2^{l_x}[$, cela sera utile pour convaincre le juge dans la prochaine section. Pour tous les autres témoins, $k_i(x_{max} - x_i) + r_i \notin]-2^{l_x}; 2^{l_x}[$. Par ailleurs, le signe de $k_i(x_{max} - x_i) + r_i$ est le même que celui de $x_{max} - x_i$ si $x_{max} \neq x_i$. Par conséquent, $x_{max} \neq x_i \implies k_i(x_{max} - x_i) + r_i > 0$, sinon, le juge doit refuser les preuves soumises.

5.1.1 Preuves de sécurité

Les objectifs de sécurité sont les suivants : il faut que (1) les valeurs x_i des témoins restent privées, que (2) les témoins ignorent s'ils possèdent ou non la valeur maximale $x_i = x_{max}$ et que (3) P ne soit pas en mesure de forger des preuves. Par ailleurs, un témoin qui participe à deux protocoles souhaite que (4) ses deux preuves ne puissent pas être liées.

La protection de x_{max} est assurée par l'utilisation du système de Paillier. Rappelons que dans le cas du protocole de comparaison avec arbre binaire, la valeur $E_T(x_{max})$ est chiffrée par le témoin W_{max} puis transmise à P. Cependant, ce témoin ignore que sa valeur a été retenue. Lorsque P diffuse $E_T(x_{max})$ dans l'étape 1, il ne faut donc pas que W_{max} puisse identifier ce chiffrement comme étant celui qu'il a lui-même émis. Le système de Paillier étant un algorithme de chiffrement probabiliste, il suffit à P de changer l'aléa du message chiffré, opération réalisée lors de l'étape 1 par le calcul de $E_T(x_{max}) \leftarrow E_T(x_{max}) \cdot r^{N_P}$. Il est donc très difficile pour les témoins de savoir s'ils possèdent ou non x_{max}. L'objectif (2) est atteint même contre des témoins semi-honnêtes grâce à la sécurité sémantique du système de Paillier, si la puissance de calcul des témoins est bornée polynomialement.

Supposons maintenant que P est semi-honnête. La clé de chiffrement utilisée est celle du juge. P ne peut donc pas déchiffrer les données. Les seules valeurs claires sont t_i et nb_it_i dont il connait déjà le contenu. Le juge est quant à lui capable de lire toutes les données. L'objectif (1) est donc lui aussi atteint contre un prouveur semi-honnête si la puissance de calcul de P est bornée polynomialement.

Les témoins, tenant à leurs vies privées, ont camouflé leurs positions dans l'équation $k_i(x_{max} - x_i) + r_i$. Le juge ne peut donc pas déterminer une autre valeur que x_{max}. La divulgation d'information liée à cette équation a déjà été analysée de manière équivalente dans le troisième chapitre (voir section 3.3.1.3). Nous avions montré que 80% des bits d'entropie étaient conservé avec un tel résultat. L'objectif (1) est donc atteint même face au juge.

Si les témoins sont des adversaires semi-honnêtes augmentés et qu'ils modifient leurs valeurs x_i, t_i ou nb_it_i, les preuves signées par ces témoins ne seront pas recevables. Cependant, nous considérons qu'ils n'ont pas de réelle motivation à faire cela. En outre, les preuves signées par les autres témoins honnêtes resteront toutefois recevables.

Le système de signature de groupe étant non-forgeable, P est capable de calculer des preuves uniquement avec sa clé de groupe. L'algorithme de signature étant traçable pour le juge, il est capable de déterminer quelles sont les preuves signées par P. Le troisième objectif est ainsi atteint si l'algorithme de signature de groupe est non-forgeable et traçable.

Si P est semi-honnête augmenté et s'il décide de falsifier la valeur de x_{max}, cela aboutira à des preuves non valables comme nous le verrons dans la prochaine section.

Par ailleurs, si un témoin participe à plusieurs protocoles simultanément ou à des moments différents et s'il génère plusieurs preuves, les propriétés d'anonymat et de non-liaison empêchent les témoins et le ou les prouveurs de déterminer si ces preuves ont été faites par le même témoin. En effet, la signature dépend de $E_T(x_{max})$, $E_T(k_i(x_{max} - x_i) + r_i)$, t_i et nb_it_i : il est très peu probable que l'ensemble de ces paramètres soit égal dans les deux preuves. Nous avons déjà vu que pour deux messages signés différents, il n'est pas possible de relier entre elles les signatures grâce à la propriété de non-liaison du système de signature. Le quatrième objectif est ainsi atteint.

5.1.2 Analyse de la complexité

On suppose que la diffusion du même message à n individus requiert une seule communication et non pas n. Le protocole de signature de preuves énoncé nécessite ainsi $n+1$ communications. En notant $|N|$ la taille des clés publiques et $|S|$ celle des signatures, on obtient $(2|N|+|S|)n+2|N|$ bits échangés. On ignore le transfert de t_i et nb_it_i dont la longueur en bit est négligeable devant les données chiffrées et devant les signatures.

D'un point de vue calculatoire, pour chaque témoin, il y a deux chiffrements, une exponentiation modulaire et une signature. P n'a besoin que de changer l'aléa de $E_T(x_{max})$.

5.1.3 Analyse des preuves

On suppose ici que P dispose de n preuves signées par autant de témoins W_i. De plus, il possède aussi le chiffrement $E_T(x_{max})$ de la valeur maximale. Rappelons que cette valeur maximale représentera, à terme, une approximation de la position x de P, tel que $x > x_{max}$.

P souhaite convaincre le juge de sa position x à un instant t. Pour cela, il donne au juge :

- Sa position x, l'instant t et sa signature $\mathrm{Sig}(s_p,t)$.
- Le maximum $E_T(x_{max})$ que le protocole de comparaison lui a permis de déterminer.
- Les n preuves $\big(t_i, nb\_it_i, E_T(k_i(x_{max}-x_i)+r_i), S_i = \mathrm{Sig}(s_i, [E_T(x_{max}), E_T(k_i(x_{max}-x_i)+r_i), t_i, nb\_it_i, \mathrm{Sig}(s_p,t)])\big)$ générées par les témoins.

Le juge procède en plusieurs étapes pour vérifier la bonne foi de P. Il commence par déchiffrer x_{max}. Si $x_{max} > x$, alors P essaie de mentir sur sa position. Sinon, le juge continue.

Il vérifie grâce aux signatures de groupe que P n'a pas forgé lui-même les preuves et que celles-ci proviennent bien de n personnes distinctes et différentes de P. Cette vérification est rendue possible par la propriété de traçabilité du système de signature.

Il vérifie ensuite que les t_i présents dans les preuves sont cohérents avec l'instant t déclaré par P. On admettra une certaine incertitude due à la synchronisation des témoins.

Il vérifie que $E_T(x_{max})$ est bien la valeur qui a été signée dans chaque preuve. Si ce n'est pas le cas, P a lui-même choisi la valeur de x_{max} pour falsifier sa position.

Il déchiffre ensuite les éléments $(k_i(x_{max} - x_i) + r_i)$. Ils devraient tous être supérieurs à -2^{l_x}. Si l'un d'entre eux est inférieur à cette valeur, cela signifie qu'il existe au moins un témoin W_i tel que $x_i > x_{max}$. x_{max} n'est donc pas la valeur maximale et P a probablement essayé de faire signer aux témoins des fausses preuves. Au moins un des $(k_i(x_{max} - x_i) + r_i)$ doit être compris dans l'intervalle $[-2^{l_x}; 2^{l_x}]$. Si ce n'est pas le cas, cela signifie que personne ne possède x_{max}. Encore une fois, P a choisi lui-même une valeur de x_{max}.

Le juge doit vérifier les valeurs de nb_it_i. Elles représentent le nombre d'itérations auxquelles les témoins W_i ont participé. Le juge peut donc en déduire le nombre de témoins qui ont participé au protocole de comparaison et le comparer au nombre de preuves effectivement reçues. Si le nombre de preuves reçues est inférieur, cela signifie que P a probablement éliminé des témoins au fur et à mesure du protocole de comparaison afin de faire baisser la valeur de x_{max}. P essaie donc de tricher sur sa position.

Enfin, le juge doit s'assurer que ces preuves sont bien associées au prouveur. Pour cela, il vérifie que $\mathrm{Sig}(s_p, t)$ est bien la valeur qui a été signée dans chaque preuve. Si ce n'est pas le cas, cela signifie que le prouveur utilise les preuves d'un autre utilisateur et qu'il essaie de tricher sur sa position.

5.2 Protocole de signature dans le cas d'un protocole de calcul de maximum bit à bit

Dans le cas où le protocole de calcul de maximum bit à bit a été utilisé, la valeur x_{max} est connue de tous. Il n'est donc pas nécessaire de la garder secrète ni de la transférer aux témoins. On peut donc alléger les exigences de sécurité et le protocole précédent pour traiter ce second cas.

De plus, le nombre d'itérations nb_it_i n'a plus de sens et peut être remplacé directement par le nombre de participants n, ce qui sera d'ailleurs plus précis pour le juge. On obtient le protocole 5.2.

Données : Le maximum x_{max} et la signature $\mathrm{Sig}(s_p,t)$ connus de tous, les valeurs x_i, n, t_i et la clé de signature s_i de chaque témoin W_i.

Résultat : P obtient une preuve par témoin.

Étape 1 : L'ensemble des témoins W_i exécute l'algorithme suivant :

si $x_{max} > x_i$ **alors**
| $\alpha_i \leftarrow 1$

sinon si $x_{max} = x_i$ **alors**
| $\alpha_i \leftarrow 0$

sinon
| $\alpha_i \leftarrow -1$

W_i signe $S_i = \mathrm{Sig}(s_i, [x_{max}, E_T(\alpha_i), t_i, n, \mathrm{Sig}(s_p,t)])$.

W_i envoie $E_T(\alpha_i), t_i, S_i$ à P.

Étape 2 : P reçoit l'ensemble des preuves et les conserve.

Protocole 5.2 : Protocole de signature des preuves

La valeur de α_i indique la cohérence de x_{max} par rapport à x_i. Si le maximum x_{max} a été obtenu honnêtement, alors α_i vaut 0 si $x_i = x_{max}$ et 1 sinon. La valeur -1 indique que x_{max} a été forgé par P et que $x_i > x_{max}$. Notons par ailleurs que P ne peut pas déchiffrer α_i.

5.2.1 Preuves de sécurité

Les objectifs de sécurité sont désormais moins nombreux : il faut que (1) les valeurs x_i des témoins restent privées et que (2) P ne soit ni en mesure de forger des preuves, ni (3) de vérifier si une preuve est valide (c.à-d. de déchiffrer $E_T(\alpha_i)$). Par ailleurs, un témoin qui participe à deux protocoles souhaite que (4) ses deux preuves ne puissent pas être liées.

Les valeurs x_i des témoins ne sont jamais communiquées dans le protocole. Le premier objectif est donc atteint sans condition. Seul α_i est transmis et indique uniquement si $x_i = x_{max}$, si $x_i > x_{max}$ ou si $x_i < x_{max}$. La variable α_i indique aussi la validité de la preuve. Toutefois, α_i est chiffré avec la clé du juge et seul ce dernier peut donc connaître cette valeur. Le troisième objectif est donc atteint grâce à la sécurité sémantique associée au système de Paillier, si la puissance de calcul de P est bornée polynomialement.

P est capable de calculer des preuves uniquement avec sa clé de groupe. L'algorithme de signature étant traçable pour le juge, ce dernier est capable de déterminer quelles sont les preuves signées par P. Le deuxième objectif est ainsi atteint.

Par ailleurs, si un témoin participe à plusieurs protocoles simultanément ou à des moments différents et s'il génère plusieurs preuves, les propriétés d'anonymat et de non-liaison empêchent les témoins et le ou les prouveurs de déterminer si ces preuves ont été faites par le même témoin. En effet, la signature dépend de x_{max}, $E_T(\alpha_i)$, t_i et n : il est très peu probable que l'ensemble de ces paramètres soit égal dans les deux preuves. Nous avons déjà vu que pour deux messages signés différents, il n'est pas possible de relier entre elles les signatures grâce à la propriété de non-liaison du système de signature. Le quatrième objectif est ainsi atteint.

5.2.2 Analyse de la complexité

n communications sont nécessaires pour mettre en place ce protocole. En notant $|N|$ la taille des clés publiques et $|S|$ celle des signatures, on obtient $(2|N| + |S|)n$ bits échangés.

Chaque témoin a besoin de réaliser uniquement un chiffrement et une signature. Ce protocole optimisé au cas du protocole de calcul de maximum bit à bit est donc plus léger que le précédent.

5.2.3 Analyse des preuves

On suppose ici que P dispose de n preuves signées par autant de témoins W_i. De plus, il possède aussi la valeur maximale x_{max}. Rappelons que cette valeur maximale représentera, à terme, une approximation de la position x de P, tel que $x > x_{max}$.

P souhaite convaincre le juge de sa position x à un instant t. Pour cela, il donne au juge :

- L'instant t, le nombre de participants n et la signature $\mathrm{Sig}(s_p, t)$.
- Le maximum x_{max} que le protocole de calcul de maximum lui a permis de déterminer.
- Les n preuves $\left(E_T(\alpha_i), t_i, S_i = \mathrm{Sig}(s_i, [x_{max}, E_T(\alpha_i), t_i, n, \mathrm{Sig}(s_p, t)])\right)$ générées par les témoins.

Le juge procède en plusieurs étapes pour vérifier la bonne foi de P.

Il vérifie grâce à la signature de groupe que P n'a pas forgé lui-même les preuves et que celles-ci proviennent bien de n personnes distinctes et différentes de P. Cette étape de vérification est rendue possible par la propriété de traçabilité du système de signature.

Il vérifie ensuite que les t_i présents dans les preuves sont cohérents avec l'instant t déclaré par P. On admettra une certaine incertitude due à la synchronisation des témoins.

Il vérifie ensuite que le x_{max} correspond bien aux N preuves envoyées par P. Si ce n'est pas le cas, P a lui-même choisi la valeur de x_{max} pour falsifier sa position.

Il déchiffre ensuite les différents α_i. Ils devraient tous être égaux à 1 ou 0. Si l'un d'entre eux vaut -1, cela signifie qu'il existe au moins un témoin W_i tel que $x_i > x_{max}$: x_{max} n'est donc pas la valeur maximale et P a probablement essayé de faire signer aux témoins des fausses preuves. De plus, au moins un des α_i doit être nul. Si ce n'est pas le cas, cela signifie que personne ne possède x_{max}. Cela signifie que P a choisi lui-même une valeur surestimée de x_{max}.

Le juge doit enfin vérifier que le nombre de preuves n correspond bien au nombre de témoins déclarés dans les signatures. Si ce n'est pas le cas, cela signifie que P a probablement volontairement ignoré certains témoins afin de faire baisser la valeur de x_{max}. P essaie donc de tricher sur sa position.

Enfin, le juge doit s'assurer que ces preuves sont bien associées au prouveur. Pour cela, il vérifie que $\mathrm{Sig}(s_p, t)$ est bien la valeur qui a été signée dans chaque preuve. Si ce n'est pas le cas, cela signifie que le prouveur utilise les preuves d'un autre utilisateur et qu'il essaie de tricher sur sa position.

5.3 Construction du rectangle

Rappelons que l'objectif initial est d'approximer la position de P à l'aide d'un rectangle et que chaque côté du rectangle est obtenu au moyen d'un protocole de comparaison appliqué à un arbre binaire ou au moyen d'un protocole de calcul de maximum bit à bit (voir le modèle de résolution général 1.1). Le côté ouest du rectangle est défini par la valeur maximale x_{max} parmi l'ensemble $\{x_i | W_i$ est à l'Ouest de $P\}$, le côté nord est défini par la valeur minimale y_{min} parmi l'ensemble $\{y_i | W_i$ est au Nord de $P\}$ et ainsi de suite. Pour transformer un protocole de calcul de maximum en un protocole de calcul de minimum, la méthode la plus simple consiste à calculer préalablement le complément à un des valeurs, puis à déterminer le maximum comme précédemment. Quatre instances de ces protocoles seront donc nécessaires pour obtenir les quatre côtés. De même, quatre protocoles de signature doivent être utilisés pour prouver la validité et la recevabilité de chacun de ces côtés, et donc de l'ensemble du rectangle.

Cela suppose toutefois une difficulté supplémentaire pour T, il doit être en mesure de déterminer quelles sont les preuves pour le côté est du rectangle , quelles sont celles pour le côté sud et ainsi de suite. Tels que les protocoles 5.1 et 5.2 sont établis, rien n'empêche pour l'instant P d'inverser les valeurs de x_{max} et x_{min} avec celles de y_{max} et y_{min} respectivement. P, qui se situe à la position (x, y), peut ainsi avoir des preuves de la position falsifiée (y, x).

Une méthode simple et évidente pour corriger ce problème est de modifier les protocoles 5.1 et 5.2 en demandant aux témoins de signer :

$$S_i = \mathrm{Sig}(s_i, [\text{“}x_{max}\text{”}, E_T(x_{max}), E_T(k_i(x_{max} - x_i) + r_i), t_i, nb\_it_i]) \qquad \text{protocole 5.1}$$

$$S_i = \mathrm{Sig}(s_i, [\text{“}x_{max}\text{”}, x_{max}, E_T(\alpha_i), t_i, n]) \qquad \text{protocole 5.2}$$

Le premier élément de la signature “x_{max}” est simplement un marqueur indiquant au juge que cette preuve est valable uniquement pour le côté ouest du rectangle, défini par x_{max}. On définira donc quatre valeurs possibles de ce marqueur : “x_{max}”, “x_{min}”, “y_{max}” et “y_{min}”. Le juge peut ainsi facilement déterminer quelles preuves appartiennent à quel côté du rectangle.

5.4 Conclusion

Dans ce chapitre, nous venons de définir la création des preuves par les témoins, leurs transferts à P et enfin leurs analyses par le juge. Nous sommes parvenus à l'ensemble de nos objectifs de sécurité, P n'est pas en mesure de forger de fausses preuves et la vie privée des témoins est respectée, même vis-à-vis du juge.

Si P doit prouver à T sa position à l'instant t, il est désormais en mesure de le faire. Par ailleurs, le juge est capable de détecter toute tentative de falsification de la part de P.

CONCLUSION

Depuis quelques années, les fonctionnalités et les applications des téléphones s'accroissent très rapidement. Dans ce mémoire, nous avons présenté une nouvelle fonctionnalité permettant aux utilisateurs de téléphones intelligents d'obtenir des preuves de leur localisation à un instant donné. Ces preuves permettent à une nouvelle génération d'applications basées sur la localisation de voir le jour. La protection de la vie privée et l'anonymat étant devenus des contraintes incontournables, la méthode présentée a pour objectif primordial de protéger la vie privée des usagers. Ni les identités ni les positions précises des individus ne sont divulguées au cours des différents protocoles proposés.

La solution proposée nécessite que les téléphones des utilisateurs soient dotés d'antennes directionnelles, ce qui n'est pas le cas dans les technologies actuelles. De plus, nous supposions que les antennes directionnelles sont calibrées de manière à n'avoir que quatre quarts de plan possibles : nord-est, sud-est, sud-ouest et nord-ouest, ce qui est assez difficile à obtenir avec une bonne précision. En effet, les gyroscopes et boussoles dont sont équipés les téléphones ne sont, aujourd'hui, pas assez précis pour obtenir nos quatre quarts de plan de manière fiable. Cependant, cette problématique nous a permis d'étudier plusieurs sous-problèmes, dont les applications possibles sont beaucoup plus vastes que les preuves de localisation seulement.

L'une des principales innovations de ce mémoire a été de concevoir un protocole de comparaison à deux individus dont la complexité est bien inférieure à celle des protocoles déjà existants. Tandis que la complexité de la plupart des protocoles de l'état de l'art s'exprime en $\Omega(l)$ (l étant la longueur en bit des données), celle de notre protocole est constante et ne dépend pas de la longueur des données. Notre solution réduit donc considérablement la charge communicationnelle et calculatoire. Cette importante amélioration s'explique principalement par trois points spécifiques à notre contexte :

- Il existe un tiers qui obtient le résultat à la fin de la comparaison.
- Les deux autres participants (Alice et Bob) ne découvrent jamais le résultat de la comparaison.

- Alice et Bob sont témoins de la position de P : ils n'ont rien à gagner ni à perdre à l'issue de ce protocole et n'ont donc aucune motivation à modifier le protocole.

Cependant, il existe un désavantage non négligeable à notre solution : la divulgation d'information n'est pas nulle. Nous avons démontré que plus de 80% des bits d'entropie sont conservés au cours du protocole, ce qui est tout à fait satisfaisant dans notre contexte. Ce protocole convient donc dans certains contextes précis pour lesquels les capacités de calcul et de transfert sont limitées et pour lesquels une faible divulgation d'informations privées est acceptable.

Ce mémoire nous a aussi permis de concevoir une méthode permettant de généraliser un protocole de comparaison classique à deux individus (tel que celui du paragraphe précédent) en un protocole de comparaison à n individus. L'objectif d'un tel protocole est de comparer l'ensemble des participants pour déterminer lequel d'entre eux possède la plus grande valeur. Les approches classiques vues en revue de littérature nécessitent $\Omega(n^2)$ comparaisons, car tous les participants sont comparés deux à deux. Notre approche parvient au résultat en seulement $O(n \log n)$ comparaisons. Pour obtenir cette complexité, nous avons imaginé réaliser des protocoles de comparaisons classiques autour d'un arbre binaire de recherche. On réduit ainsi considérablement le nombre de comparaisons à effectuer. Cependant, il est nécessaire qu'un tiers construise au fur et à mesure du protocole cet arbre binaire, ce qui nécessite qu'il connaisse le résultat de certaines comparaisons intermédiaires. Notre protocole divulgue donc certaines informations en plus du résultat recherché, contrairement aux protocoles de calcul de maximum classiques. Cette divulgation a été quantifiée : le tiers découvre $n-1$ comparaisons, ce qui est bien inférieur aux $n \log n$ comparaisons nécessaires pour trier les valeurs des participants. Par le même raisonnement que précédemment, ce protocole est ainsi une solution intéressante dans des contextes où la charge calculatoire et communicationnelle est primordiale.

Enfin, un protocole de calcul de maximum bit à bit a aussi été imaginé. Le but de cette famille de protocoles est de déterminer la valeur maximale parmi un ensemble de participants, sans révéler l'identité de son possesseur. La complexité de la majorité des protocoles de calcul de maximum s'exprime en $\Omega(l \cdot n^2)$, tandis que notre protocole est de l'ordre de $O(l \cdot n)$. Cela s'explique principalement par des objectifs de sécurité moins stricts compte tenu de notre

contexte. En effet, nous avons démontré que certaines formes de collusion peuvent permettre (aux membres de cette collusion) de découvrir quelques bits de la valeur de chaque participant. Les mécanismes de sécurité mis en place dans ce protocole empêchent cependant cette collusion de choisir le nombre et l'index des bits ainsi révélés. De plus, un grand nombre de participants permet de réduire le risque de divulgation, qui ne peut malgré tout jamais être nul en cas de collusion.

Ces trois innovations se distinguent donc de l'état de l'art à la fois par leur complexité réduite en terme de calculs et de transferts, et par leur divulgation d'information non nulle. Toutefois, nous avons démontré dans chaque cas que la divulgation d'information restait suffisamment faible compte tenu de notre contexte. Ces protocoles sont donc intéressants lorsque le contexte nécessite de faire un compromis entre la complexité et la protection de la vie privée.

La prolongation de ce mémoire serait de répondre au problème de preuves de localisation uniquement au moyen de calculs multi-parties et sans utiliser d'équipement spécifique, tel que les antennes directionnelles. Pour cela, les protocoles d'estimation de distance (*distance-bounding protocols*) semblent être une approche intéressante. Il serait aussi intéressant d'essayer de parvenir aux mêmes résultats avec une divulgation d'information inexistante.

BIBLIOGRAPHIE

Blake, Ian F et Vladimir Kolesnikov. 2004. « Strong conditional oblivious transfer and computing on intervals ». In *Proc. of Advances in Cryptology – ASIACRYPT 2004*. p. 515–529. Springer Berlin Heidelberg.

Brassard, Gilles, Claude Crépeau, et Jean-Marc Robert. 1987. « All-or-nothing disclosure of secrets ». In *Proc. of Advances in Cryptology—CRYPTO'86*. p. 234–238. Springer Berlin Heidelberg.

Clifton, Chris. 2001. « Privacy preserving distributed data mining ». In *Proc. of the 13th European Conference on Machine Learning*. p. 19–23.

Clifton, Chris, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, et Michael Y Zhu. 2002. « Tools for privacy preserving distributed data mining ». *ACM SiGKDD Explorations Newsletter*, vol. 4, n° 2, p. 28–34.

Cormen, Thomas, Clifford Stein, Ronald Rivest, et Charles Leiserson, 2009. *Introduction to algorithms*.

Coron, Jean-Sébastien, Avradip Mandal, David Naccache, et Mehdi Tibouchi. 2011. « Fully homomorphic encryption over the integers with shorter public keys ». In *Proc. of Advances in Cryptology–CRYPTO 2011*. p. 487–504. Springer.

ElGamal, Taher. 1985. « A public key cryptosystem and a signature scheme based on discrete logarithms ». In *Advances in cryptology*. p. 10–18. Springer.

Even, Shimon, Oded Goldreich, et Abraham Lempel. 1985. « A randomized protocol for signing contracts ». *Communications of the ACM*, vol. 28, n° 6, p. 637–647.

Fischlin, Marc. 2001. « A cost-effective pay-per-multiplication comparison method for millionaires ». In *Proc. of Topics in Cryptology – CT-RSA 2001*. p. 457–471. Springer.

Fontaine, Caroline et Fabien Galand. 2007. « A survey of homomorphic encryption for nonspecialists ». *EURASIP Journal on Information Security*, vol. 2007, p. 1–15.

Franklin, Matthew et Haibin Zhang. 2012. « Unique group signatures ». In *Proc. of Computer Security – ESORICS 2012*. p. 643–660. Springer.

Gambs, Sébastien, Marc-Olivier Killijian, Matthieu Roy, et Moussa Traoré. 2014. « PROPS : A PRivacy-Preserving Location Proof System ». In *Proc. of the IEEE 33rd International Symposium on Reliable Distributed Systems (SRDS)*. p. 1–10. IEEE.

Gentry, Craig et Shai Halevi. 2011. « Implementing Gentry's fully-homomorphic encryption scheme ». In *Proc. of Advances in Cryptology–EUROCRYPT 2011*. p. 129–148. Springer.

Gentry, Craig et al. 2009. « Fully homomorphic encryption using ideal lattices. ». In *Proc. of the 41st ACM Symposium on Theory of Computing (STOC)*. p. 169–178.

Goldreich, Oded. 1998. « Secure multi-party computation ». *Manuscript. Preliminary version*.

Goldreich, Oded, 2004. *Foundations of cryptography : volume 2, basic applications*.

Goldwasser, Shafi. 1997. « Multi party computations : past and present ». In *Proc. of the 16th Annual ACM Symposium on Principles of Distributed Computing*. p. 1–6. ACM.

Hasan, Omar, Lionel Brunie, et Elisa Bertino. 2012. « Preserving privacy of feedback providers in decentralized reputation systems ». *Computers & Security*, vol. 31, n° 7, p. 816–826.

Hasan, Omar, Jingwei Miao, Sonia Ben Mokhtar, et Lionel Brunie. 2013. « A privacy preserving prediction-based routing protocol for mobile delay tolerant networks ». In *Proc. of the 27th IEEE International Conference on Advanced Information Networking and Applications (AINA)*. p. 546–553. IEEE.

Hazay, Carmit et Yehuda Lindell, 2010. *Efficient secure two-party protocols : Techniques and constructions*.

Ioannidis, Ioannis et Ananth Grama. 2003. « An efficient protocol for Yao's millionaires' problem ». In *Proc. of the 36th IEEE Annual Hawaii International Conference on System Sciences*. p. 6–pp. IEEE.

Lin, Hsiao-Ying et Wen-Guey Tzeng. 2005. « An efficient solution to the millionaires' problem based on homomorphic encryption ». In *Applied Cryptography and Network Security*. p. 456–466. Springer Berlin Heidelberg.

Luo, Wanying et Urs Hengartner. 2010. « Veriplace : a privacy-aware location proof architecture ». In *Proc. of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*. p. 23–32. ACM.

Paillier, Pascal. 1999. « Public-key cryptosystems based on composite degree residuosity classes ». In *Proc. of Advances in cryptology – EUROCRYPT'99*. p. 223–238. Springer Berlin Heidelberg.

Rivest, Ronald L, Adi Shamir, et Len Adleman. 1978. « A method for obtaining digital signatures and public-key cryptosystems ». *Communications of the ACM*, vol. 21, n° 2, p. 120–126.

Saroiu, Stefan et Alec Wolman. 2009. « Enabling new mobile applications with location proofs ». In *Proc. of the 10th Workshop on Mobile Computing Systems and Applications*. p. 3. ACM.

Sheikh, Rashid, Beerendra Kumar, et Durgesh Kumar Mishra. 2010. « A distributed k-secure sum protocol for secure multi-party computations ». *arXiv preprint arXiv :1003.4071*.

Tzeng, Wen-Guey. 2004. « Efficient 1-out-of-n oblivious transfer schemes with universally usable parameters ». *IEEE Transactions on Computers*, vol. 53, n° 2, p. 232–240.

Yao, Andrew Chi-Chih. 1982. « Protocols for secure computations ». In *Proc. of the 23rd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*. p. 160–164.

Zhang, Sheng et Fillia Makedon. 2005. « Privacy preserving learning in negotiation ». In *Proc. of the 2005 ACM Symposium on Applied Computing*. p. 821–825. ACM.