Université de Montréal

Sécurité polynomiale en cryptographie

par
Heinz Fiedler Straubhaar

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

31 Août 2012

Université de Montréal
Faculté des arts et des sciences

Ce mémoire est intitulé :

Sécurité polynomiale en cryptographie

présenté par
Heinz Fiedler Straubhaar

a été évalué par un jury composé des personnes suivantes :

Gilles Brassard	président rapporteur
Alain Tapp	directeur de recherche
Gena Hahn	membre du jury

Mémoire accepté le : .....

Résumé

Dans ce mémoire, nous proposons des protocoles cryptographiques d'échange de clef, de mise en gage, et de transfert équivoque. Un premier protocole de transfert équivoque, primitive cryptographique universelle pour le calcul multi-parties, s'inspire du protocole d'échange de clef par puzzle de Merkle, et améliore les résultats existants.
Puis, nous montrons qu'il est possible de construire ces mêmes primitives cryptographiques sans l'hypothèse des fonctions à sens unique, mais avec le problème 3SUM. Ce problème simple —dans une liste de n entiers, en trouver trois dont la somme a une certaine valeur— a une borne inférieure conjecturée de $\Omega(n^2)$.
Mots clefs : cryptographie, cryptographie à clef publique, échange de clef, mise en gage, transfert équivoque.

Abstract

In this work, we propose cryptographic protocols for key exchange, bit commitment and oblivious transfer. Our oblivious transfer protocol, universal cryptographic primitive for multipartie computation, is inspired from Merkle's key exchange protocol with puzzles, and improves on existing results.
Then, we show that it's possible to build those same cryptographic primitives without the hypothesis of one-way functions, but with the 3SUM problem. This simple problem —in a list of n integers, find three that sum is a desired value— has a conjectured lower bound of $\Omega(n^2)$.
Keywords : cryptography, public-key cryptography, key exchange, bit commitment, oblivious transfer.

Table des figures

Liste des protocoles

Table des matières

Notations

$\mathbb{B}$	$\{0,1\}$
lb	Logarithme binaire
$\|r\|$	Taille d'une chaîne de bits
$\mathsf{bin}(x)$	Représentation binaire de l'entier x
$\mathsf{bin}(x)_i$	$i^{\text{ème}}$ bit de $\mathsf{bin}(x)$
$\mathsf{poly}(n)$	Une fonction polynomiale de n

Remerciements

Tout d'abord, j'aimerais exprimer ma gratitude et mes plus grands remerciements à Alain, qui a su dévouer temps et énergie pour m'aider, me soutenir, m'apprendre, et parfois m'endurer pendant ce travail dans le groupe d'informatique théorique. Au delà d'un chercheur talentueux, il a surtout été pour moi un professeur d'exception, de qui j'ai pu apprendre tour à tour comme élève, étudiant, et assistant d'enseignement. La passion n'est qu'une partie de la recette du succès, et sa dévotion totale à toujours améliorer ses qualités d'enseignant et de pédagogue sont probablement la meilleure leçon qu'on puisse recevoir dans une université.

A ma mère Suzanne, qui contre vents et marées à toujours cru en moi et m'a soutenu dans mes projets, des plus simples aux plus compliqués. Peu d'enfants peuvent se targuer d'avoir une mère aussi exceptionnelle, et j'espère être sur la bonne voie pour lui arriver à la cheville un jour.

A ma moitié Catherine, qui m'a soutenu, d'ici ou d'ailleurs, dans les moments les plus durs, et a partagé les moments les meilleurs.

A Rébecca, pour avoir été mon double le plus efficace au département et ailleurs, et pour son aide irremplaçable.

A Karine, sans qui je n'aurais jamais su que franchir les frontières était la meilleure chose qui puisse se faire.

A tous les membres du laboratoire, pour être le groupe de recherche le plus fantastique qui soit sur cette île.

Chapitre 1

Introduction

1.1 Mise en contexte

Traditionnellement, la cryptographie a été utilisée pour protéger une information d'ennemis ou d'espions, avec plusieurs participants à une tâche cryptographique (appelée *protocole*) qui collaborent dans un but commun.

L'utilisation de systèmes permettant de rendre une information illisible par un adversaire peut être retracée dans des formes primitives jusqu'à l'Egypte Ancienne, bien que le système antique le plus connu soit possiblement le chiffre de César [Kah96]. Jusqu'à récemment, le but premier de la cryptographie était d'empêcher la lecture d'informations par un adversaire, soit protéger *le secret* de l'information. L'avènement de la cryptographie à clef publique depuis les années soixante-dix ouvrit les portes à de nouvelle application de la cryptographie, comme la distribution de clefs de chiffrement, ou la signature d'information pour en garantir l'authenticité. Dans tous ces systèmes, les participants collaborent avec confiance pour accomplir une tâche commune et se protéger d'un adversaire commun. Dans le contexte du *calcul multiparti*, en revanche, les participants collaborent pour accomplir une certaine tâche, mais *sans* se faire nécessairement confiance.

A priori contre-intuitif, il existe pourtant des exemples concrets de tâches de ce type. Le premier exemple, proposé par Andrew Yao [Yao82], est celui

du problèmes des millionnaires. Deux millionnaires, Alice et Bob, veulent déterminer lequel est le plus riche, sans pourtant révéler précisément leurs fortunes l'un à l'autre. Bien qu'un peu artificiel, cet exemple cache en fait celui d'une mise aux enchères, ou les participants veulent connaître le gagnant de l'enchère sans révéler leur mise.

Une autre tâche réalisable est le calcul de la majorité. Plusieurs participants votent oui ou non à une question, et veulent le résultat de leur vote comme une majorité de non, ou de oui. Dans ce cas, les participants veulent préserver le secret de leur vote et garantir l'exactitude du résultat.

1.2 Transfert équivoque

La primitive la plus importante en calcul multiparti est le transfert équivoque (Oblivious Transfer). Imaginons une société pharmaceutique internationale détenant les formules pour produire de nombreux médicaments. Un pays désire acheter une formule pour un médicament nécessaire à sa population souffrant d'une carence à vitamine D. La société pharmaceutique veut s'assurer que le protocole de vente ne donnera pas plus qu'une formule à l'acheteur. Le pays acheteur en revanche, veut protéger son achat et ne pas révéler les faiblesses de sa population à ses ennemis. Un protocole de transfert équivoque permet d'effectuer cette tâche, une entité détenant des informations sera capable d'en transmettre une et seule à une entité curieuse, sans pour autant apprendre quelle information a été demandée.

Cependant, l'existence de protocoles de transfert équivoque ne se limite pas à des exemples quelque peu artificiels comme celui-ci. En effet, Joe Kilian a montré [Kil88] que le transfert équivoque est une primitive *complète* pour le calcul biparti. Cette primitive permet ainsi de réaliser n'importe quel protocole cryptographique, et est donc d'une importance remarquable.

1.3 Mise en gage

La mise en gage est une primitive de calcul multiparti probablement plus simple à appréhender que le transfert équivoque. Imaginons deux personnes

désirant parier sur le résultat d'une élection importante. Chaque personne écrit son pari sur une feuille, qu'elle glisse dans un coffre, puis donne ce coffre à l'autre parieur. Au moment où les résultats de l'élection sont connus, il suffit aux parieurs de s'annoncer leurs prédictions, et de se donner les clefs afin de pouvoir vérifier le contenu des paris dans les coffres. La mise en gage est alors simplement une manière cryptographique pour chaque participant de réaliser la même tâche. Au delà de cette utilisation, la mise en gage revêt une importance particulière dans des protocoles de preuve sans divulgation d'information (zero-knowledge proof) [QQQ+90]. Dans un tel protocole, un prouveur veut démontrer à un vérificateur qu'il dispose bel et bien d'une information, comme par exemple le certificat d'une instance d'un problème dans NP.

1.4 Echange de clefs de Merkle

En 1974, Ralph Merkle, alors étudiant à Berkley, proposa un système permettant à deux participants d'échanger un secret sur un canal de transmission authentifié, mais écouté par un espion [Mer78]. Dans ce système, Alice et Bob tirent indépendamment et au hasard des clefs dans un certain domaine de taille N. Ces clefs sont chiffrées en les utilisant comme entrées d'une fonction à sens unique, et les chiffres sont échangés sur le canal authentifié. La probabilité pour qu'Alice et Bob aient tiré la même clef tend vers un rapidement quand le nombre de tirage dépasse $\sqrt{N}$. Cette clef commune peut alors être utilisée comme clef de chiffrement pour établir une communication secrète entre Alice et Bob. Une adversaire, Eve, écoutant la communication entre Alice et Bob devra, en revanche, chiffrer en moyenne la moitié des clefs du domaine avant de tomber sur la clef commune tirée par Alice et Bob. Dans un tel système, Bob et Eve partagent au début exactement la même information, soit le domaine des clefs. Le protocole oblige cependant Eve à travailler plus qu'Alice et Bob pour dévoiler l'information sur laquelle ils se sont entendus. Ce protocole sera le premier protocole d'échange de clef, élément maintenant majeur de la cryptographie à clef publique.

1.5 Transfert équivoque à la Merkle

Gilles Brassard, Louis Salvail et Alain Tapp ont montré un protocole de transfert équivoque utilisant des permutations à sens unique, et se basant sur la méthode proposée par Merkle pour procéder à un échange de clefs [BST09]. Leur protocole est efficace dans le modèle de la sécurité polynomiale du système de Merkle. Dans ce contexte, la différence de travail entre un participant honnête et un participant malhonnête est seulement polynomiale par rapport au paramètre de sécurité du protocole. En règle générale, les cryptographes demandent que la différence soit plus grande que n'importe quel polynôme, soit typiquement une exponentielle. Dans le cas du transfert équivoque, il n'existe cependant pas de méthode connue permettant de l'implémenter sous la seule condition d'existence de fonctions ou permutations à sens unique en conservant une sécurité super-polynomiale. Il s'avère, comme démontré dans [IR89], que ce problème est aussi dur à résoudre que la séparation des classes P et NP.

1.6 3SUM

Le problème $\mathsf{3SUM}$ s'énonce très simplement. Parmi un ensemble de n entiers, peut-on en trouver 3 dont la somme a une certaine valeur ? En règle générale, un tel problème peut se résoudre en $O(n^2)$ étapes grâce à une fouille « intelligente » dans les sommes possibles de ces entiers. Cette borne supérieure sur la complexité du problème est aussi conjecturée comme étant une borne inférieure [GO95]. Si cette conjecture était vérifiée, il existerait une classe de problèmes dont la difficulté est au moins aussi grande que celle de $\mathsf{3SUM}$, appelée $\mathsf{3SUM-Hard}$, qui aurait alors également trouvé une borne inférieure.

1.7 Notre contribution

Si un participant honnête au protocole de [BST09] travaille pendant un temps t, il devra en revanche travailler en un temps $t^{\frac{3}{2}}$ pour tricher à ce même protocole. En utilisant une démarche similaire à la leur et par l'utilisation de la mise en gage, nous proposons d'améliorer la sécurité du protocole en rendant le temps nécessaire à tricher à t^2. Puis, nous utiliserons la primitive $\mathsf{3SUM}$ en lieu et place d'une fonction à sens unique afin d'implémenter

les protocoles d'échange de clef, de mise en gage, et finalement de transfert équivoque. Sans parvenir à égaler les performances des protocoles utilisants des fonctions à sens unique, nous proposons cependant des protocoles à sécurité polynomiale basés sur la conjecture de complexité de 3SUM.

Notre protocole d'échange de clef nécessite un travail de $n^{\frac{3}{2}}$ pour les deux participants, mais un travail d'au moins $\frac{n^2}{log(n)}$ pour un adversaire.

Le protocole de mise en gage ne nécessite que la création de n entiers, après quoi un temps $\frac{n^2}{log(n)}$ est également requis pour briser le protocole.

Quant à notre protocole de transfert équivoque, il nécessite également un travail de $n^{\frac{3}{2}}$ pour être réalisé par les participants honnêtes, et un travail de $\frac{n^2}{log(n)}$ pour être brisé par le participant demandant à obtenir les bits d'information.

Première partie

Préliminaires

Chapitre 2

Préliminaires mathématiques

Dans ce chapitre, nous aborderons les différents objets mathématiques nécessaires à la construction de différents protocoles cryptographiques. Nous commencerons par l'étude générale des fonctions à sens unique, puis des cas plus particuliers des permutations à sens unique et des fonctions de hachage résistantes aux collisions. Puis, nous aborderons le problème 3SUM et ses dérivés. Enfin, nous introduirons les différentes primitives cryptographiques qui seront ensuite réalisées dans nos protocoles.

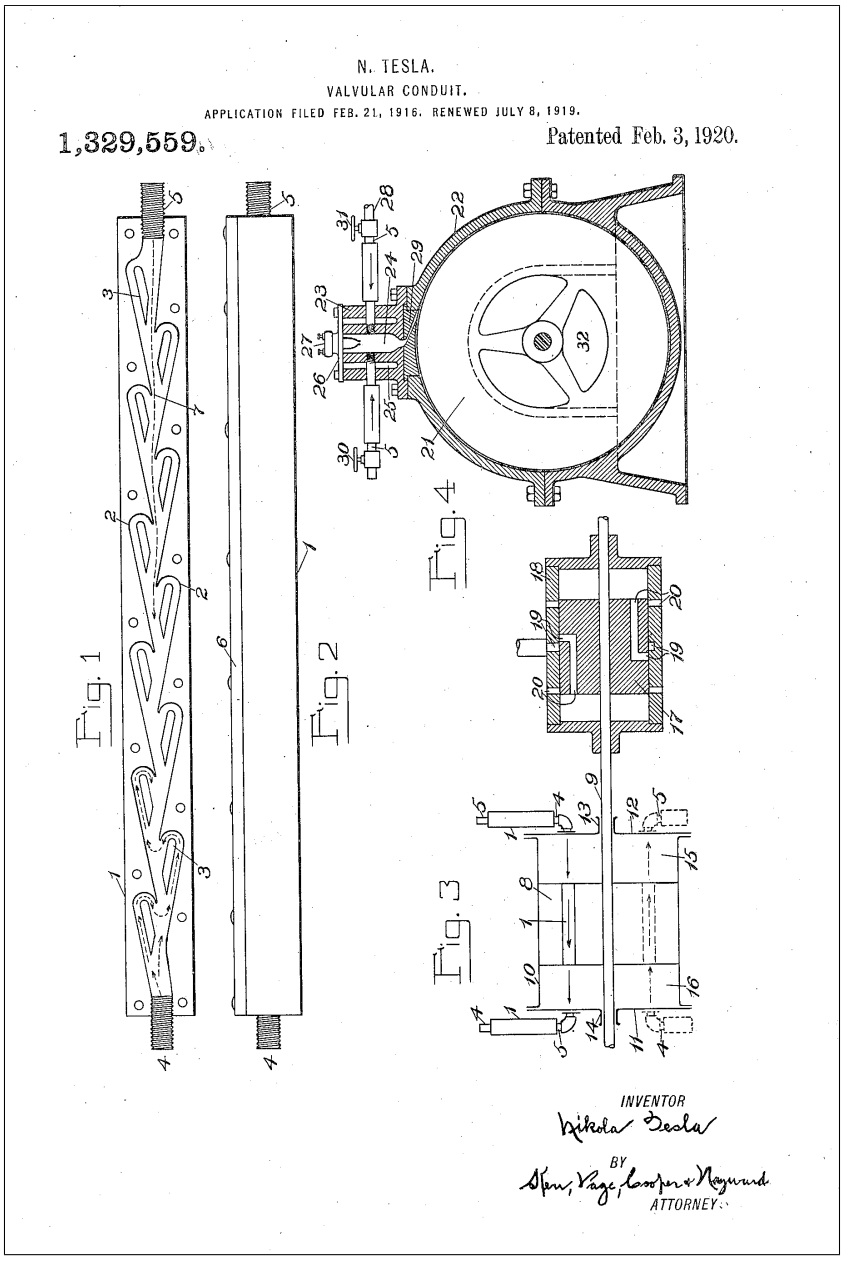

FIGURE 2.1 – Valve hydraulique de Tesla. La forme des obstacles dans le conduit permet au liquide de s'écouler dans le sens de l'orientation des chicanes. Dans l'autre sens, les turbulences bloquent éventuellement l'écoulement du liquide, formant ainsi une conduite à sens unique sans pièce mécanique.

2.1 Fonctions à sens unique

Essentiellement, une fonction a sens unique est une fonction simple à calculer, mais difficile à inverser. Bien que l'existence concrète de telles fonctions n'ait pas encore été démontrée, de nombreux systèmes cryptographiques sont construits autour d'elles. En pratique, il existe un certain nombre de fonctions candidates dont on pense qu'elles sont à sens unique, la multiplication étant possiblement l'exemple le plus connu.

Définitions [KL08]

En règle générale, la simplicité de l'évaluation d'une fonction se définit par l'appartenance de la fonction à la classe de complexité FP, soit que l'évaluation de la fonction se fait en temps polynomial sur la taille de l'entrée.
En revanche, la définition de la difficulté d'inversion requiert plusieurs éléments dans un modèle avec adversaire.

Définition 2.1.1 (Expérience d'inversion $\mathsf{Invert}_{\mathtt{A},f}(n)$)**.**
Soit $f : \{0,1\}^* \to \{0,1\}^*$ une fonction.
L'expérience $\mathsf{Invert}_{\mathtt{A},f}(n)$ réalise la tâche suivante.

1. *Choisir une entrée x de la fonction de taille n.*
2. *Calculer $y \coloneqq f(x)$.*
3. *L'adversaire $\mathtt{A}$ obtient 1^n et y. Il retourne x'.*
4. *La fonction retourne 1 si $f(x') = y$, 0 sinon.*

Informellement, nous demandons à ce qu'aucun adversaire ayant à sa disposition une puissance de calcul polynomiale probabiliste ne soit capable de trouver une pré-image d'une sortie de la fonction avec une probabilité meilleure que négligeable.

Définition 2.1.2. *Une fonction f est* négligeable *si pour tout polynôme p, il existe $N \in \mathbb{N}$ tel que pour tout $n > N$, $f(n) < \frac{1}{p(n)}$.*

Une telle fonction sera généralement appelée $\mathsf{negl}(n)$.

Ainsi, nous pouvons formellement définir le concept de fonction à sens unique.

Définition 2.1.3. *Une fonction est* à sens unique *si elle répond aux deux conditions :*

1. *(Simple à calculer)* $f \in \mathsf{FP}$
2. *(Difficile à inverser) Pour tout algorithme polynomial probabiliste* $\mathtt{A}$*, il existe une fonction négligeable* negl *telle que*

$$\Pr\left[\mathsf{Invert}_{\mathtt{A},f}(n) = 1\right] < \mathsf{negl}(n)$$

En plus de la difficulté d'inverser un élément particulier, il peut s'avérer nécessaire que la découverte de collisions dans la sortie d'une fonction soit difficile. Nous introduisions donc la notion de fonctions résistantes aux collisions *au sens fort.*

Définition 2.1.4 $(\mathsf{Func} - \mathsf{coll}_{\mathtt{A},f}(n))$**.**
Soit $f : \{0,1\}^* \to \{0,1\}^*$ *une fonction.*
Soit $\mathtt{A}$ *un algorithme.*
L'expérience $\mathsf{Func} - \mathsf{coll}_{\mathtt{A},f}(n)$ *réalise la tâche suivante.*

1. *L'algorithme* $\mathtt{A}$ *retourne* x *et* x' *de taille* n*.*
2. *La fonction retourne 1 si* $f(x') = f(x)$ *et* $x \neq x'$*, 0 sinon.*

La résistance aux collisions est alors définie de manière similaire au cas général en utilisant une fonction négligeable.

Définition 2.1.5. *Une fonction est* résistante aux collisions *si Pour tout algorithme polynomial probabiliste* $\mathtt{A}$*, il existe une fonction négligeable* negl *telle que*

$$\Pr\left[\mathsf{Func} - \mathsf{coll}_{\mathtt{A},f}(n) = 1\right] < \mathsf{negl}(n)$$

Au *sens faible*, nous demandons à ce que l'adversaire ne puisse trouver une collision étant donné une entrée de la fonction.

Définition 2.1.6 $(\mathsf{Func} - \mathsf{second}_{\mathtt{A},f}(n))$**.**
Soit $f : \{0,1\}^* \to \{0,1\}^*$ *une fonction.*
Soit $\mathtt{A}$ *un algorithme.*
L'expérience $\mathsf{Func} - \mathsf{coll}_{\mathtt{A},f}(n)$ *réalise la tâche suivante.*

1. *Choisir une entrée x de taille n.*
2. *L'adversaire* $\mathtt{A}$ *obtient x.*
3. *L'adversaire* $\mathtt{A}$ *retourne x' de taille* $\mathsf{poly}(n)$.
4. *La fonction retourne 1 si $f(x') = f(x)$ et $x \neq x'$, 0 sinon.*

Et à nouveau, nous définissons la résistance aux deuxièmes préimages.

Définition 2.1.7. *Une fonction est* résistante aux deuxièmes pré-images *si pour tout algorithme polynomial probabiliste* $\mathtt{A}$*, il existe une fonction négligeable* negl *telle que*

$$\Pr\left[\mathsf{Func} - \mathsf{second}_{\mathtt{A},f}(n) = 1\right] < \mathsf{negl}(n)$$

Permutations à sens unique

Une permutation est une fonction particulière, une bijection d'un ensemble dans lui-même. Ainsi, les permutations à sens uniques se définissent de manière similaire aux fonctions à sens uniques. Evidemment, une permutation étant une bijection, il est par définition impossible de trouver des collisions, et la seule définition de difficulté d'inversion est nécessaire.

Définition 2.1.8. *Une* permutation à sens unique *est une fonction à sens unique qui est aussi une bijection du domaine de la fonction.*

Prédicats difficiles [KL08]

Connaissant la sortie d'une fonction à sens unique, il est difficile d'en connaître l'entrée. Cependant, il pourrait être possible de connaître de l'information sur l'entrée. Afin de pallier cette situation, Oded Goldreich et Leonid Levin ont montré que toute fonction à sens unique admet aussi un prédicat difficile : une fonction retournant un bit facile à calculer sur l'entrée de la fonction à sens unique, mais difficile à calculer sur la seule sortie de cette fonction [GL89].

Définition 2.1.9 (Expérience d'inversion $\mathsf{InvertHC}_{\mathtt{A},f,b}(n)$)**.**
Soit $f : \{0,1\}^* \to \{0,1\}^*$ une fonction à sens unique.
Soit $b : \{0,1\}^* \to \{0,1\}$ un prédicat pour f.
Soit $\mathtt{A}$ *un algorithme.*
L'expérience $\mathsf{InvertHC}_{\mathtt{A},f,b}(n)$ *réalise la tâche suivante.*

1. *Choisir une entrée x de la fonction de taille n.*
2. *Calculer $y = f(x)$.*
3. *Calculer $w = b(x)$.*
4. *L'adversaire* $\mathtt{A}$ *obtient* 1^n *et y. Il retourne w'.*
5. *La fonction retourne 1 si $w = w'$, 0 sinon.*

Définition 2.1.10. *Soit f une fonction à sens unique. Alors b est un* prédicat difficile *pour f si quelque soit* $\mathtt{A}$ *un algorithme probabiliste polynomial, $Pr[\mathsf{InvertHC}_{\mathtt{A},f,b}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n)$.*

Théorème 2.1.1 (Goldreich-Levin)**.** *Soit f une fonction à sens unique. De f, on construit g une fonction telle que $g(x, r) = (f(x), r)$, où $\|r\| = \|x\|$ une chaîne de bits uniformément distribués.*
Alors, $b(x, r) = \bigoplus_{i=1}^{\|x\|} x_i r_i$ est un prédicat difficile pour g.

2.2 Modèle de la boîte noire et oracle aléatoire

Par la suite, il s'avérera nécessaire de renforcer les notions de difficulté d'inversion des fonctions. A cet effet, nous utilisons le modèle de la boîte noire. Dans ce modèle, informellement, au lieu d'utiliser la *description* d'une fonction, nous obtenons une *boîte noire* évaluant la fonction à notre place.

Définition 2.2.1. *Soit f une fonction. Soit* $\mathtt{Q}$ *un participant à un protocole. Une* boîte noire *ou un* oracle *pour f est un participant* $\mathtt{X}$ *qui pour toute requête x de* $\mathtt{Q}$*, retourne $f(x)$.*

Nous pouvons ensuite imaginer une boîte noire ou la fonction n'est pas définie a priori, mais est une fonction *aléatoire*. A chaque requête d'un participant, la boîte noire fait un choix aléatoire uniforme sur le codomaine de f, retourne ce choix au participant, et mémorise le couple requête-choix. De manière équivalente, nous pouvons aussi dire que la fonction a été choisie sur une distribution aléatoire uniforme parmi les fonctions de domaine et codomaine voulus. Ainsi, dans ce modèle, il n'est possible d'obtenir une évaluation de la fonction qu'en la demandant explicitement à la boîte noire $\mathtt{X}$, et ce à coût unitaire. Ce modèle est celui de *l'oracle aléatoire*,

fréquemment utilisé en cryptographie.

Plus spécifiquement, si la fonction f est une permutation aléatoire sur un ensemble de taille n, et connaissant $f(x)$, il faut alors en moyenne $\frac{n}{2}$ requêtes à la boîte noire pour retrouver x. Les permutations aléatoires en boite noire répondent ainsi à toutes les définitions sur les permutations à sens unique, étant donné que la seule action possible pour obtenir une inversion d'un élément est l'évaluation de la permutation.

2.3 Paradoxe des anniversaires

Le paradoxe des anniversaires est un problème bien connu en théorie des probabilités, et également en cryptographie. Dans son exemple le plus connu, on demande de déterminer combien de personnes il faut rassembler pour avoir une bonne probabilité qu'au moins deux d'entre elles partagent le même anniversaire. La réponse est paradoxale, au sens étymologique premier. En effet, alors que l'on pourrait s'attendre à devoir réunir environ la moitié de 365 personnes, il suffit de 23 personnes réunies pour que la probabilité d'un anniversaire commun dans ce groupe dépasse 50%.

En toute généralité, il suffit de faire $\sqrt{n}$ tirages aléatoires uniformes sur un ensemble de taille n pour avoir une probabilité élevée de tirer deux fois le même élément de l'ensemble.

Plus généralement, nous allons considérer le cas du tirage aléatoire d'éléments d'un ensemble de taille n. Soit t le nombre de tirage. Nous considérons que tous les tirages sont indépendants, et que la distribution de ceux-ci est uniforme. Nous allons alors déterminer non pas la probabilité d'obtenir plus d'une fois le même élément, mais au contraire la probabilité d'obtenir un élément différent par tirage. Il y a n possibilités pour le premier tirage. La probabilité d'obtenir alors le même élément au second tirage est de $\frac{1}{n}$. Alors, la probabilité complémentaire d'obtenir un élément différent est de $1 - \frac{1}{n}$. La probabilité d'obtenir à nouveau un élément déjà tiré au troisième tirage est de $\frac{1}{n} + \frac{1}{n} = \frac{2}{n}$, et la probabilité complémentaire d'obtenir un élément nouveau est alors de $1 - \frac{2}{n}$. En continuant ce raisonnement, on obtient que la probabilité d'obtenir un élément nouveau au tirage i est de $1 - \frac{i}{n}$. Ainsi, pour t tirages, nous pouvons multiplier ces probabilité individuelles, et obtenir la probabilité $q(t)$ de t tirages distincts.

Théorème 2.3.1. *Soit U un ensemble de taille n. Soit t le nombre de tirages effectués dans cet ensemble. On définit $q(t)$ comme la probabilité d'obtenir t tirages distincts. Alors,*

$$\boxed{q(t) \leq e^{-\frac{t^2}{2n}}} \tag{2.1}$$

Démonstration.

$$q(t) = \prod_{i=0}^{t-1}(1 - \frac{i}{n})$$

Le passage au logarithme nous ensuite

$$\ln\, q(t) = \sum_{i=0}^{t-1} \ln(1 - \frac{i}{n}) \tag{2.2}$$

Le développement de Taylor de $\ln(1 - \frac{i}{n})$ nous donne pour $\frac{i}{n} \leq 1$

$$\ln(1 - \frac{i}{n}) = -\sum_{k=1}^{\infty} \frac{(-1)^k}{k}(-\frac{i}{n})^k$$

$$\ln(1 - \frac{i}{n}) \leq -\frac{i}{n}$$

En remplaçant dans (2.2), nous obtenons

$$\ln\, q(t) \leq \sum_{i=0}^{t-1} -\frac{i}{n} = -\frac{t(t-1)}{2n}$$

Alors, en prenant l'exponentielle, nous retrouvons $q(t)$

$$q(t) \leq e^{-\frac{t(t-1)}{2n}} \leq e^{-\frac{t^2}{2n}}$$

□

Puis, nous pouvons déterminer une borne supérieure pour $p(t) = 1 - q(t)$, soit la probabilité d'obtenir au moins une fois le même tirage.

Théorème 2.3.2. *Soit U un ensemble de taille n. Soit t le nombre de tirages effectués dans cet ensemble, et soit $p(t)$ la probabilité d'obtenir au moins une fois le même tirage. Alors,*

$$\boxed{p(t) \leq \frac{t(t-1)}{2n}} \tag{2.3}$$

Démonstration. Soit C_i l'évènement que le i$^{\text{ème}}$ tirage est le même qu'un tirage précédent. Au i$^{\text{ème}}$ tirage, il y a eu au maximum $i-1$ tirage distincts prédécemment, et donc

$$\mathsf{Pr}(C_i) \leq \frac{i-1}{n} \tag{2.4}$$

Alors,

$$p(t) = \mathsf{Pr}(\bigvee_{i=1}^{t} C_i) \tag{2.5}$$

En utilisant (2.4), on obtient

$$\begin{aligned} p(t) &\leq \mathsf{Pr}(\bigvee_{i=1}^{t} C_i) \\ &\leq \sum_{i=1}^{t} \mathsf{Pr}(C_i) \\ &\leq \sum_{i=1}^{t} \frac{i-1}{n} \\ &= \frac{t(t-1)}{2n} \end{aligned}$$

Et donc la bonne supérieure

$$p(t) \leq \frac{t(t-1)}{2n} \tag{2.6}$$

□

Ainsi, avec 2.1, si $t = \sqrt{2n}$, nous trouvons $q(t) \leq e^{-1} \simeq 0.37$. C'est ce résultat qui est à la base du paradoxe des anniversaires, où l'on constate qu'avec 23 élèves, la probabilité que tous soient nés des jours différents tombe sous les 50%.

2.4 3SUM

FIGURE 2.2 – Les Trois Grâces. Copie romaine d'une statue grecque du II^{ème} siècle av. J.-C., Metropolitan Museum of Art, New York.

Le problème 3SUM se définit simplement de la manière suivante [GO95].

Définition 2.4.1 (Problème 3SUM)**.** *Soit A un ensemble d'entiers tel que $|A| = n$ (la* taille*). Soit d un entier. Existe-t-il trois éléments de A distincts, a, b et c, tels que $a + b + c = d$?*

Le plus souvent le problème rencontré dans la littérature utilise $d = 0$. Afin de simplifier la notation dans la suite du travail, nous abuserons de la terminologie pour inclure tous les choix de d.

Théorème 2.4.1. *Il existe un algorithme pouvant résoudre une instance du problème* 3SUM *de taille n et de paramètre d en temps $\Theta(n^2)$ [Hof09].*

Démonstration. L'algorithme consiste en une fouille non exhaustive des triplets de A.

Algorithme 1: Algorithme pour résoudre 3SUM

```
Entrées : A
Sorties : (a, b, c)
Trier(A);
pour i ← 0 à n − 3 faire
    a ← A[i];
    k ← i + 1;
    l ← n − 1;
    tant que k < l faire
        b ← A[k];
        c ← A[l];
        si a + b + c = d alors
            sortie ← (a, b, c);
            fin;
        si a + b + c > d alors
            l ← l − 1;
        sinon
            k ← k + 1;
```

□

Cette borne supérieure est conjecturée comme étant également une bornée *inférieure* sur le problème 3SUM, et également sur une classe de problèmes associés, appelée $3\mathsf{SUM} - \mathsf{Hard}$.

Conjecture 2.4.1. *L'algorithme 1 est optimal pour le problème* 3SUM.

Définition 2.4.2 ($3\mathsf{SUM} - \mathsf{Hard}$)**.** *Un problème* P *est* $3\mathsf{SUM} - \mathsf{Hard}$ *si chaque instance de* 3SUM *de taille* n *peut se résoudre en utilisant un nombre constant d'instances de* P *de taille* $O(n)$ *et* $o(n^2)$ *temps de calcul supplémentaire.*
On écrit alors $3\mathsf{SUM} \lll_{n^2} \mathsf{P}$.

Lemme 2.4.1 ([GO95])**.** *Soit* $3\mathsf{SUM} \lll_{n^2} \mathsf{P}$. *Si* $\Omega(n^2)$ *est une borne inférieure pour* 3SUM, *alors* $\Omega(n^2)$ *est aussi une borne inférieure pour* P.

Par exemple, le problème suivant fait partie de la classe $3\mathsf{SUM} - \mathsf{Hard}$.

Définition 2.4.3 (Problème 3SUM′)**.** *Soit* A, B *et* C *des ensembles d'entiers, tels que* $|A| = |B| = |C| = n$.
Existe-il trois éléments $a \in A, b \in B, c \in C$ *tels que* $a + b = c$ *?*

Il existe une multitude de problèmes $3\mathsf{SUM} - \mathsf{Hard}$, et beaucoup sont liés au domaine de la géométrie [Kin04]. Par exemple, le problème d'alignement de trois points. Dans ce problème, nous recevons un ensemble de n points dans le plan, et cherchons si trois d'entre eux sont alignés. Ce problème se résout simplement. En effet, les n points peuvent former $n(n-1) < n^2$ paires différentes, chacune représentant de manière univoque une droite. Il suffit donc de générer la droite associée à chaque couple et de vérifier qu'elle n'a pas déjà été générée. De manière intéressante, ce problème d'alignement peut être utilisé pour résoudre directement une instance de 3SUM avec $d = 0$, en associant à chaque entier i de l'instance de 3SUM le point (i, i^3) dans le plan. En effet, soient trois points $(x_1, {x_1}^3), (x_2, {x_2}^3)$ et $(x_3, {x_3}^3)$. On peut montrer que ces trois points sont colinéaires si et seulement si $x_1 + x_2 + x_3 = 0$ [GO95][Kin04].

Ces points définissent en effet trois droites. La droite entre les points $(x_i, {x_i}^3)$ et $(x_j, {x_j}^3)$ est de pente $m = \frac{x_i^3 - x_j^3}{x_i - x_j} = x_i^2 + x_i x_j + x_j^2$, et de hauteur $h = -x_i x_j (x_i + x_j)$.

Si les trois points sont colinéaires, alors les trois droites sont de mêmes pentes et hauteurs, et nous devons alors résoudre le système d'équations :

$$\begin{cases} x_1^2 + x_1x_2 + x_2^2 = m \\ x_1^2 + x_1x_3 + x_3^2 = m \\ x_2^2 + x_2x_3 + x_3^2 = m \\ -x_1x_2(x_1 + x_2) = h \\ -x_1x_3(x_1 + x_3) = h \\ -x_2x_3(x_2 + x_3) = h \end{cases}$$

La résolution de ce système amène alors bien à $x_1 + x_2 + x_3 = 0$.

Chapitre 3

Fonctionnalités cryptographiques

3.1 Aléa partagé

Certaines constructions cryptographiques ne sont possibles que si tous les intervenants partagent préalablement une certaine information. Dans le cas de protocole symétriques classiques par exemple, on pense évidemment à la clef privée. Plus généralement, nous donnerons parfois aux intervenants dans nos protocoles la possibilité d'accéder à de l'aléa partagé.

Définition 3.1.1. *Une* chaîne aléatoire partagée $\mathcal{R}$ *de taille* n *est une chaîne de* n *bits uniformément distribuée sur* $\mathbb{B}^n$ *diffusée à tous les participants à un protocole cryptographique une et une seule fois.*

Alors que le modèle de l'oracle aléatoire offre un accès « illimité » à une fonction aléatoire, la chaîne aléatoire partagée est une mise à disposition limitée de bits aléatoires.

3.2 Echange de clefs

Alice et Bob veulent échanger un secret sur un canal authentique, mais surveillé par Eve. Toutes les ressources mises à disposition d'Alice et Bob sont aussi à disposition de Eve.
A la fin du protocole, Alice et Bob partagent un secret, qu'il sera difficile

pour Eve de déterminer.

3.3 Mise en gage

Un protocole de mise en gage est une manière de réaliser la tâche informelle suivante. Deux participants, Alice et Bob, ne se font pas confiance. Alice veut parier sur le résultat d'une épreuve avec Bob, mais ne veut pas dévoiler son choix avant que le résultat de l'épreuve soit connu de tous. Elle écrit alors son choix sur une feuille de papier, qu'elle enferme dans un coffre-fort verrouillé par combinaison. Ce coffre est alors transmis à Bob. Si elle désire dévoiler son choix, Alice transmet la combinaison pour ouvrir le coffre à Bob ainsi que son pari, que Bob peut alors vérifier en ouvrant le coffre.

Les objectifs de sécurité varient selon le participant. Alice veut empêcher Bob d'apprendre son pari avant qu'elle ne le décide elle-même. Le coffre-fort doit donc résister à une attaque malicieuse de Bob. Bob de son côté veut éviter qu'Alice ne change son pari après que le coffre n'ait été transmis. Il faut alors éviter que le coffre-fort contienne plusieurs chambres cachées, s'ouvrant chacune selon une combinaison différente, ou la présence d'un mécanisme quelconque permettant de modifier le choix sur la feuille.

Nous pouvons alors définir formellement ces différents concepts pour un protocole de mise en gage, réalisé en deux phases.

Schéma 1 Mise en gage COMMIT

Une émettrice `Alice` et un receveur `Bob` participent à la mise en gage. `Alice` possède un bit b auquel elle veut s'engager auprès de `Bob`.

Paramètre de sécurité

`Alice` et `Bob` s'entendent sur un paramètre de sécurité $n \in \mathbb{N}$.

Mise en gage

`Alice` transmet $s = \mathsf{COMMIT}(b)$ à `Bob`.

Schéma 2 Ouverture de la mise en gage $\mathsf{OPEN} - \mathsf{COM}$

Ouverture

`Alice` envoie le message (c, b) à `Bob`.

Vérification

`Bob` vérifie que $\mathsf{OPEN} - \mathsf{COM}(c, s) = b$.

Nous pouvons alors définir la sécurité d'un protocole de mise en gage dans un modèle avec adversaire.

Définition 3.3.1. *Un protocole de mise en gage est* (t, ϵ)*-camouflant si pour tout* `Bob` *malhonnête travaillant en temps maximum* t*, la probabilité* P *pour* `Bob` *d'ouvrir la mise en gage est*

$$P \leq \frac{1}{2} + \epsilon$$

Si $\epsilon = 0$*, alors on dira du protocole qu'il est* parfaitement camouflant*.*

De manière similaire, nous pouvons définir la résistance au changement de la mise en gage.

Définition 3.3.2. *Un protocole de mise en gage est* (t, ϵ)*-liant si pour toute* `Alice` *malhonnête travaillant en temps maximum* t*, la probabilité* P *pour* `Alice` *de modifier sa mise en gage est*

$$P \leq \epsilon$$

Si $\epsilon = 0$*, alors on dira du protocole qu'il est* parfaitement liant*.*

3.4 Transfert équivoque

Une émettrice `Alice` possède deux bits d'information, x_0 et x_1. Un demandeur `Q` veut apprendre $x_y, y \in \mathbb{B}$. Pour modéliser la fonctionnalité, nous introduisons un participant ancillaire pour jouer le rôle d'intermédiaire, soit `X`.

Schéma 3 Transfert équivoque OT_1^2

1. `Alice` envoie (x_0, x_1) à `X`.
2. `Bob` envoie y à `X`.
3. `X` envoie x_y à `Bob`, et rien à `Alice`.

Transfert équivoque aléatoire

Une primitive équivalente au transfert équivoque est le transfert équivoque *aléatoire* [Cré88]. Dans ce mode de transfert, l'émettrice `Alice` obtient deux bits uniformément distribués, β_0 et β_1. Le demandeur obtient $(\beta_0, 0)$ avec probabilité $\frac{1}{2}$, $(\beta_1, 1)$ sinon.
Il est possible de construire un transfert équivoque normal en partant de cette primitive aléatoire.

Protocole 1 Transfert équivoque $\mathsf{DeRandOT}_1^2$

1. Soit $p = (\beta_0, \beta_1)$.
2. `Bob` obtient β_w et w. Il désire apprendre x_y.
3. `Bob` annonce $\alpha = w \oplus y$.
4. `Alice` envoie $\Gamma = (x_0 \oplus \beta_{\alpha\oplus 0}, x_1 \oplus \beta_{\alpha\oplus 1})$.
5. `Bob` calcule $\Gamma_y \oplus \beta_w$.

Théorème 3.4.1. *Le protocole est correct.*

Démonstration. A la fin du protocole, `Bob` a appris $\Gamma_y \oplus \beta_w = x_y \oplus \beta_{\alpha\oplus y} \oplus \beta_w = x_y \oplus \beta_{w\oplus y\oplus y} \oplus \beta_w = x_y \oplus \beta_w \oplus \beta_w = x_y$. □

Deuxième partie

Protocoles

Chapitre 4

Protocoles basés sur les fonctions à sens unique

Le modèle le plus général que nous utilisons est celui des fonctions à sens unique. En particulier, nous proposerons nos protocoles en utilisant des permutations en « boîte noire ». Ce modèle permet de simplifier les preuves de sécurité. En effet, la permutation n'étant accessible que par le biais d'un oracle aléatoire, l'inversion d'une sortie de la permutation nécessite de faire un nombre de requêtes à l'oracle proportionnel à la taille du domaine de la permutation.

4.1 Echange de clefs

Deux participants au protocole, `Alice` et `Bob`, veulent échanger une information sur un canal de communication non sécurisé —mais authentifié— espionné par une adversaire, Eve.
Nous utilisons l'hypothèse de l'existence d'une permutation aléatoire f, utilisable par tous comme une boîte noire.

Le protocole est une application directe du paradoxe des anniversaires. Chaque participant évalue la permutation à sens unique sur des entrées choisies aléatoirement dans le domaine, et transmet les résultats des évaluations à l'autre, jusqu'à ce qu'ils aient chacun trouvé le même élément au cours

de leur évaluation. Si le domaine de la permutation est de taille n, on s'attend à ce que les participants évaluent la permutation environ $\sqrt{n}$ fois avant d'avoir obtenu au moins une fois la même évaluation. L'élément évalué commun devient alors le secret partagé.

Du côté de l'adversaire en revanche, il est nécessaire d'inverser l'évaluation commune des deux participants, ce qui nécessite environ $\frac{n}{2}$ évaluations de la permutation.

Protocole 2 Echange de clef avec une permutation à sens unique $\mathsf{KEX}_{\mathsf{func}}$ [Mer78]

---

1. `Alice` et `Bob` s'entendent sur le paramètre de sécurité k.
2. Soit $A = [1, \ldots, k^2]$.
3. `Alice` génère aléatoirement un sous-ensemble B de A, de taille k.
4. `Alice` calcule $Y = (f(B_1), \ldots, f(B_k))$.
5. `Alice` envoie Y à `Bob`.
6. `Bob` évalue f sur des entrées aléatoirement choisies dans A jusqu'à ce qu'il trouve b tel que $f(b) \in Y$.
7. `Bob` annonce l'élément commun $f(b)$ à `Alice`.
8. `Alice` cherche $f(b)$ dans Y.
9. `Alice` et `Bob` partagent à présent $A_i = b$.

Travail pour les participants Nous considérons pour la suite le temps nécessaire à l'évaluation de la permutation f sur une entrée comme une constante, utilisée comme comme unité de temps. `Alice` doit évaluer f sur k entrées, et donc travaille en $O(k)$.

D'après le paradoxe des anniversaires, `Bob` doit évaluer f sur $O(k)$ entrées dans A en moyenne avant de trouver un élément évalué par `Alice`. Ainsi, `Bob` doit travailler en $O(k)$ également.

Théorème 4.1.1. *Le protocole est brisé par une attaquante* `Eve` *pouvant faire* k^2 *évaluations de la permutation à sens unique.*

Démonstration. Soit `Eve` une telle attaquante. En obtenant $f(b)$, `Eve` peut simplement évaluer f sur $[1, \ldots, k^2]$ jusqu'à retrouver b. □

Théorème 4.1.2. *Le protocole est résistant à une attaquante* `Eve` *pouvant faire* $o(k^2)$ *évaluations de la permutation à sens unique.*

Démonstration. La permutation f étant une boite noire, `Eve` n'a d'autre choix que d'évaluer la permutation jusqu'à trouver l'élément commun de `Alice` et `Bob`, soit $\Omega(k^2)$ évaluations.

□

4.2 Mise en gage

Nous montrons ici une méthode simple de mise en gage utilisant des permutations à sens unique en boîte noire [Gol01]. Cette méthode est inconditionnellement liante, et calculatoirement camouflante.

`Alice` et `Bob` disposent d'une permutation en boîte noire f. `Alice` veut mettre en gage un bit b auprès de `Bob`.

Protocole 3 Mise en gage avec une permutation $\mathsf{COMMIT} - \mathsf{FUNC}$

1. `Alice` et `Bob` s'entendent sur un paramètre de sécurité n.
2. `Alice` choisit aléatoirement deux chaines de bits a et r de taille n.
3. `Alice` calcule $y = f(a)$, $\bigoplus_i a_i r_i = h$ et $c = b \oplus h$.
4. `Alice` envoie (y, r, c) à `Bob`.

Une fois la mise en gage effectuée, il est alors possible de procéder à son ouverture.

Protocole 4 Ouverture de la mise en gage $\mathsf{OPEN} - \mathsf{FUNC}$

1. Pour révéler sa mise en gage, `Alice` dévoile a à `Bob`.
2. `Bob` calcule $y = f(a)$, $\bigoplus_i a_i r_i = h$ et $b = c \oplus h$.

Théorème 4.2.1. *Le protocole de mise en gage est inconditionnellement liant pour* `Alice`.

Démonstration. Par construction du protocole, f est une permutation, et donc il n'existe pas $x \neq x'$ tels que $f(x) = f(x')$. □

Théorème 4.2.2. *Le protocole est sûr pour* `Alice` *au sens calculatoire.*

Démonstration. Par le théorème de Goldreich-Levin, h est un bit difficile pour f, et donc il est aussi dur pour `Bob` de déterminer h que d'inverser f. □

4.3 Transfert équivoque

Le protocole prend place entre un demandeur `Bob` et une émettrice `Alice`. A leur disposition, une permutation aléatoire en boîte noire f et un protocole de mise en gage en boîte noire $(\mathsf{INIT}-\mathsf{COM}, \mathsf{COMMIT}, \mathsf{OPEN}-\mathsf{COM})$. Ce protocole est fortement inspiré du protocole d'échange de clef. Cependant, pendant son exécution, nous évitons que les participants apprennent exactement l'évaluation de la permutation qu'ils ont en commun. Ainsi, les deux participants possèdent un « secret » partagé, sans savoir lequel exactement. Si les deux participants arrivent alors à s'entendre sur une paire d'éléments évalués par l'envoyeur, contenant l'élément commun, il leur est possible d'effectuer un transfert équivoque.

Nous nous basons sur la méthode introduite dans [BST09], que nous modifions pour intégrer un protocole de mise en gage dans le déroulement du transfert équivoque. A l'aide de la mise en gage, nous pouvons permettre aux participants au protocole de se transmettre des informations sans toutefois les révéler, permettant de reposer la sécurité du protocole de transfert équivoque sur la sécurité du protocole de mise en gage fourni et d'améliorer les résultats précédents. Si un participant honnête doit travailler en temps t, il lui faudra un temps $t^{\frac{3}{2}}$ pour briser le protocole de [BST09], contre t^2 dans notre cas. Par la suite, nous écrirons souvent le paramètre de sécurité n comme $n = 2^r$, afin de simplifier la notation, comme $\mathsf{lb}(n) = r$.

Protocole 5 Transfert équivoque $\mathsf{OT} - \mathsf{FUNC}$

1. `Bob` et `Alice` s'entendent sur le paramètre de sécurité $n = 2^r$.
2. `Alice` génère aléatoirement un ensemble X de $2k$ entiers dans $[1, \ldots, n]$ avec $k = \sqrt{2n}$.
3. `Alice` calcule $S = f(X)$.
4. `Alice` transmet S à `Bob`.
5. `Bob` évalue aléatoirement f sur des entiers dans $[1, \ldots, n]$ jusqu'à ce qu'il trouve y tel que $f(y)$ dans S.
6. `Bob` annonce à `Alice` avoir trouvé un élément, et s'engage à $f(y)$ bit par bit en utilisant le protocole de mise en gage fourni.
   Donc, `Bob` transmet
   $$K = (\mathsf{COMMIT}(\mathsf{bin}(y)_1), \ldots, \mathsf{COMMIT}(\mathsf{bin}(y)_{\|y\|}))$$
   à `Alice`.
7. `Alice` effectue une permutation aléatoire sur S pour former S'.
8. Avec S', `Alice` génère $P = ((S'_1, S'_2), \ldots, (S'_{2k-1}, S'_{2k}))$, soit les éléments de S disposés aléatoirement en couples.
9. `Alice` transmet P à `Bob`.
10. `Bob` annonce P_γ tel que $f(y) \in P_\gamma$ à `Alice`, soit le couple de P contenant l'élément qu'il a réussi à inverser.
    Soit $P_\gamma = (f(x_0), f(x_1))$, et $y = x_w$.
11. `Bob` dévoile à `Alice` les mises en gage des bits communs des éléments de P_γ, soit le message
    $$W = \bigcup_i \mathsf{OPEN}(K_i) \text{ tels que } \mathsf{bin}(f(x_0))_i = \mathsf{bin}(f(x_1))_i$$
12. `Alice` transmet $\pi \in [1, \ldots, r]$ à `Bob`.
13. `Alice` dispose de $\beta_0 = \mathsf{bin}(x_0)_\pi$ et $\beta_1 = \mathsf{bin}(x_1)_\pi$.
14. `Bob` dispose de $\beta_w = \mathsf{bin}(x_w)_\pi$.

Théorème 4.3.1. *Le protocole est équivoque si* COMMIT *est camouflante.*

Démonstration. Les seules informations transmises par `Bob` dans le protocole sont les mises en gage dans K. Si le protocole de mise en gage est camouflant, `Alice` n'apprend donc aucune information supplémentaire.

Puis, seules les mises en gage des bits communs de x_0 et x_1 étant ouvertes, `Alice` n'apprend rien sur l'élément inversé par `Bob`.

□

Ainsi, le protocole est sûr pour le demandeur pour autant que la mise en gage le soit. Mais évidemment, si cette mise en gage n'était pas présente, l'émettrice `Alice` n'aurait pas obtenu plus d'informations sur l'élément inversé par `Bob`. La mise en gage ne joue donc ici pas un rôle important pour la preuve de sécurité. Pour l'émettrice en revanche, la mise en gage prend toute son importance. Supposons que `Bob` soit malhonnête, et soit capable d'inverser les deux éléments de la paire finale P_γ. Afin d'utiliser cet adversaire malhonnête pour effectuer une réduction, nous devons être capable de l'utiliser pour inverser une sortie quelconque de la permutation, $s = f(y)$. Nous pouvons alors utiliser la mise en gage. Nous remplaçons la boîte noire de mise en gage, par une boîte biaisée dont les mises en gage peuvent facilement s'ouvrir. Alors, au moment de former les paires dans P, nous pouvons ouvrir la mise en gage effectuée par `Bob` sur son élément, pour pouvoir insérer dans la paire P_γ la sortie de la permutation que nous voulons inverser.

Théorème 4.3.2. *Un* `Bob` *malhonnête capable de briser le protocole en temps* $o(\frac{n^2}{log(n)})$ *est capable d'inverser* f *en temps* $o(n^2)$ *sur un domaine de taille* n^2*.*

Démonstration. Le protocole de mise en gage est remplacé par un protocole biaisé, où $\mathsf{OPEN} - \mathsf{COM}$ ne nécessite pas la participation de `Bob` pour être appelée, nous permettant d'ouvrir toutes ses mises en gage.

`Alice` génère S de taille $n-1$, et transmet $S' = S \cup s$. Ainsi, s est inclue dans l'ensemble S, et il suffira de forcer `Bob` à utiliser cet élément dans la suite du protocole.

Puis, Bob répond normalement par ses mises en gage K, qui sont ouvertes grâce à $\mathsf{OPEN} - \mathsf{COM}$, permettant alors de dévoiler l'élément s' inversé par Bob.

Nous pouvons donc construire P avec un élément $P_\gamma = (s, s')$ et ainsi forcer Bob à utiliser l'élément s que l'on cherche à inverser. Nous pouvons ensuite transmettre $\pi = 1$, et Bob va ainsi posséder le premier bit de s. Nous répétons alors cette procédure pour tous les autres bits de s, soit $\lceil \mathsf{lb}(s) \rceil$ fois, et obtenons s au complet. Ainsi, si Bob brise le protocole en temps $o(\frac{n^2}{log(n)})$, il pourra inverser s en temps $o(n^2)$.

□

Nous constatons donc que si le protocole est exécuté en temps t par un receveur, il faudra un temps t^2 afin de le briser pour autant que l'hypothèse de complexité sur la permutation à sens unique soit valide, et en négligeant les facteurs logarithmiques.

Chapitre 5

Protocoles basés sur 3SUM

Nous présentons dans ce chapitre des protocoles d'échange de clef, de mise en gage et de transfert équivoque basés sur le problème 3SUM, inspirés des protocoles précédents basés sur des fonctions à sens unique.
Il est facile de créer de toute pièce une instance de 3SUM, il suffit en effet de générer un ensemble de n entiers, avec trois d'entre-eux dont la somme est nulle. Le meilleur algorithme connu nécessite, en revanche, de l'ordre de n^2 étapes, et est conjecturé comme étant optimal [GO95]. Cette différence entre création et résolution de l'instance fait écho aux différences qui existent dans nos protocoles cryptographiques utilisant des fonctions à sens unique.

Nous proposons donc d'utiliser le problème 3SUM pour implémenter ces protocoles, et tenter d'approcher les performances de certains protocoles avec des fonctions à sens uniques.

5.1 Initialisation

Chaque protocole basé sur le problème 3SUM nécessitera la génération d'une liste d'entiers, utilisée pour créer une ou plusieurs instances de 3SUM. Idéalement, nous aimerions permettre aux participants pouvoir générer eux-mêmes la liste de nombre utilisés.
Cependant, une telle génération des nombres nous confronte à des problèmes de sécurité dans les protocoles de mise en gage et de transfert équivoque.

Définition 5.1.1. *Soient A, B et C trois ensembles d'entiers. Une* collision *est un couple de triplets $(a, b, c), (a', b', c') \in A \times B \times C$, tels que $a+b+c = a'+b'+c'$ et $(a, b, c) \neq (a', b', c')$.*

Or, nous le verrons par la suite, l'existence de collisions compromet la sécurité des protocoles.

Afin d'éviter d'existence de collisions, nous utilisons une chaîne aléatoire partagée $\mathcal{R}$. Une chaîne de bits nous permet en effet de définir une liste d'entiers. Pour cela, il faut fixer non seulement la taille de la liste, mais aussi le nombre de bits dévoués à la représentation de chaque entier. Nous appellerons ces paramètres n et m, respectivement.

Le paramètre de sécurité n représente la taille de l'instance de 3SUM, alors que m sera fixé pour limiter le nombre de collisions possibles lors de la génération d'une instance de 3SUM. Comme précédemment, nous écrirons n comme $n = 2^r$, afin de simplifier la notation de $\mathsf{lb}(n) = r$.

En effet, supposons pour le bien de l'exemple que les entiers sont choisis parmi 0 et 1. Alors nécessairement chaque triplet aura une somme comprise entre 0 et 3. Le domaine pour la somme des triplets étant si limité, on comprend aisément qu'une valeur de n —même modeste— aura pour conséquence des collisions dans les sommes de triplets. Il est donc nécessaire que la somme des entiers ait une image plus importante pour minimiser la probabilité de collisions.

Protocole 6 Mise en place d'instance de 3SUM $\mathsf{INIT} - 3\mathsf{SUM}$

Le protocole prend place entre deux participants `Alice` et `Bob`.

1. `Alice` et `Bob` s'entendent sur les paramètres de sécurité r, $n = 2^r$, $k \in \mathbb{N}$, $m = 6 + k$.
2. Une chaîne aléatoire $\mathcal{R}$ de taille $3nmr$ est diffusée aux participants.
3. Les $3nmr$ premiers bits de $\mathcal{R}$ forment $3n$ entiers représentés sur mr bits.
4. `Alice` et `Bob` s'entendent sur des listes de taille n, A, B et C.

Théorème 5.1.1. *Soient $n = 2^r$ et $m = 6+k$, $k, n \in \mathbb{N}$. Soit p la probabilité qu'il existe une collision dans A, B et C retournés par* $\mathsf{INIT} - \mathsf{3SUM}$ *sur entrée r, n, k. Alors,*

$$p \leq \frac{1}{n^k}$$

Démonstration. Soit $A + B + C = S$ et $m = 6 + k$.
Nous considérons alors la somme au modulo n^m. Ainsi, $S \bmod n^m$ est uniformément distribué dans $[0, \ldots, n^m - 1]$.

Nous appliquons alors le théorème (2.6) pour k et m, *i.e.* la probabilité d'obtenir une collision dans $S \bmod n^m$ soit au plus de $\frac{1}{n^k}$, et donc

$$\frac{n^3(n^3-1)}{2n^m} \leq \frac{1}{n^k}$$

Nous recherchons alors une valeur pour m qui satisfasse cette inégalité. Soit

$$n^m \geq n^{6+k},\ m \geq 6 + k$$

Alors

$$n^m \geq \frac{n^{6+k}}{2} \rightarrow n^6 \leq \frac{2n^m}{n^k}$$

Pour approcher la forme recherchée, nous introduisions le terme n^3.

$$n^6 - n^3 \leq \frac{2n^m}{n^k}$$

Et finalement, nous vérifions bien

$$\frac{n^3(n^3-1)}{2n^m} \leq \frac{1}{n^k}$$

Nous constatons qu'une collision dans S a évidemment pour conséquence une collision dans $S \bmod n^k$. Cependant, l'inverse n'est pas vrai. On peut trouver $a + b + c = a' + b' + c' \bmod n^k$ dans $S \bmod n^k$ tels que $a + b + c \neq a' + b' + c'$. □

5.2 Échange de clef

L'échange de clefs nécessite pour les deux participants `Alice` et `Bob` de définir les paramètres de sécurité, puis de générer une instance de 3SUM.

Nous le verrons plus loin, il est possible pour l'un ou l'autre des participants de générer ladite instance sans modifier la sécurité ou la complexité du protocole. En effet, le « participant » malhonnête dans ce protocole, l'attaquant `Eve`, est le seul à ne pas pouvoir influencer la génération de l'instance. Nous pouvons donc demander à un des participants `Alice` ou `Bob` de générer la chaîne aléatoire du protocole $\mathsf{INIT}-\mathsf{3SUM}$.

Protocole 7 Protocole d'échange de clef $\mathsf{KEX}-\mathsf{3SUM}$

1. `Alice` et `Bob` s'entendent sur les paramètres de sécurité r et $n = 2^r$. Soit $k = 2$.
2. `Alice` et `Bob` appellent $\mathsf{INIT}-\mathsf{3SUM}$ avec r, n et k.
3. `Alice` choisit aléatoirement $\lceil n^{\frac{3}{2}} \rceil$ triplets parmis $A \times B \times C$. La somme de chaque triplet est calculée, et transmise à `Bob` dans la liste S.
4. `Bob` choisit aléatoirement des triplets parmi $A \times B \times C$, jusqu'à trouver un triplet (a, b, c) tel que $a + b + c = s \in S$.
5. `Bob` transmet s à `Alice`.
6. `Alice` transmet $\pi \in_r [1, \ldots, 24r]$ à `Bob`.
7. `Alice` et `Bob` possèdent le secret partagé $\mathsf{bin}(a, b, c)_\pi$.

Théorème 5.2.1. `Alice` *et* `Bob` *s'entendent sur une collision en temps espéré* $O(\sqrt{n^3})$.

Démonstration. Comme précédemment, $|A \times B \times C| = n^3$. Alors, par le paradoxe des anniversaires, `Alice` et `Bob` s'entendront sur un élément commun (une collision) avec probabilité constante après $O(\sqrt{n^3}) = O(n^{\frac{3}{2}})$ étapes. □

De plus, $k = 2$, et donc la probabilité d'existence d'une collision dans (A, B, C) est plus petite que $\frac{1}{n^2}$. Alors, la probabilité pour les participants d'obtenir chacun un élément *différent* d'une collision est plus petite que $\frac{1}{n^2}$,

et donc asymptotiquement nulle. Cette probabilité peut encore être réduite en augmentant le paramètre k.

Théorème 5.2.2. *Le protocole est brisé par une adversaire* `Eve` *en* $O(n^2)$ *opérations.*

Démonstration. Connaissant les ensembles A, B, C et l'entier s, `Eve` résoud l'instance 3SUM $(A, B, C, s, 1^n)$ en $O(n^2)$ opérations, et aprend donc n'importe quel bit de $\mathsf{bin}(a, b, c)$. □

Puis, nous montrons la sécurité du protocole face à une adversaire.

Théorème 5.2.3. *Une adversaire* `Eve` *capable de briser le protocole (soit d'apprendre* $\mathsf{bin}(a, b, c)_\pi$*) en temps* $o(\frac{n^2}{log(n)})$ *est capable de résoudre le problème* 3SUM *en temps* $o(n^2)$*.*

Démonstration. Supposons qu'il existe `Eve` comme énoncé. Nous l'utilisons alors pour résoudre une instance de notre choix de 3SUM, $I = (A, B, C, s, 1^n)$.

`Alice` et `Bob` utilisent A, B et C, et donc `Alice` émet (A, B, C).

`Alice` génère ensuite $\lceil n^{\frac{3}{2}} \rceil - 1$ triplets, et en calcule les sommes respectives dans la liste S. Elle transmet $S' = S \cup \{s\}$ à `Bob`.

`Bob` répond $s, \pi \in_r [1, \ldots, 24\mathsf{lb}(n)]$. Ainsi, nous avons forcé l'utilisation de l'instance I dans le protocole, sur laquelle `Eve` doit à présent travailler. Il suffit de répéter l'opération pour chaque bit de s jusqu'à ce qu'`Eve` ait découvert tous les bits. Ainsi, en répétant la réduction pour $\pi = 1$ jusqu'à $\pi = 24\mathsf{lb}(n)$, on résoudra une instance de 3SUM en temps $o(log(n)\frac{n^2}{log(n)}) = o(n^2)$.

□

5.3 Mise en gage

Une fois qu'il est possible de générer instance de 3SUM, nous pouvons l'utiliser pour implémenter un protocole de mise en gage, où `Alice` s'engage auprès de `Bob` sur la valeur d'un bit z.

Protocole 8 Protocole de mise en gage $\mathsf{COMMIT} - \mathsf{3SUM}$

1. `Alice` et `Bob` s'entendent sur les paramètres de sécurité r, $n = 2^r$ et $k \in \mathbb{N}$, $m = 6 + k$.
2. `Alice` et `Bob` appellent $\mathsf{INIT} - \mathsf{3SUM}$.
3. `Alice` choisit aléatoirement $a \in A, b \in B$ et $c \in C$. Elle calcule la somme $a + b + c = s$.
4. `Alice` détermine une position aléatoire π de la chaîne $\mathsf{bin}(a|b|c)$, dont la valeur est z.
5. `Alice` transmet le message (s, π) à `Bob`.

Le protocole de mise en gage nécessite tout d'abord pour le receveur et l'émettrice de disposer d'assez d'entiers, pour ensuite pouvoir instancier une instance 3SUM sur ces entiers. Or, la manière de générer l'instance de 3SUM a son importance. Evidemment, on ne peut pas demander à l'émettrice `Alice` de générer l'instance, il serait en effet trivial pour elle de créer une instance avec des collisions, lui permettant de modifier son engagement.

Le cas de la génération par `Bob` de l'instance est malheureusement plus difficile. Nous n'avons pas réussi à créer un protocole de génération d'instance qui soit à la fois efficace en temps, et sûr face à `Bob` malicieux, et mettons ainsi à leur disposition une chaîne aléatoire partagée, de laquelle ils vont pouvoir générer des entiers naturels de la taille de leur choix via $\mathsf{INIT} - \mathsf{3SUM}$.

Une fois la mise en gage effectuée par `Alice`, il est alors possible de procéder à son ouverture.

Protocole 9 Ouverture de la mise en gage $\mathsf{OPEN}-\mathsf{3SUM}$

1. `Alice` dévoile sa mise en gage en envoyant le triplet (a, b, c) et le bit z à `Bob`.
2. `Bob` vérifie alors que $\mathsf{bin}(\mathsf{a}|\mathsf{b}|\mathsf{c})_\pi = z$.

Dans le protocoles, il s'agit pour `Bob` de s'assurer que `Alice` ne pourra pas facilement trouver $a, a' \in A,\ b, b' \in B,\ c, c' \in C$ tels que $(a, b, c) \neq (a', b', c')$ et $a + b + c = a' + b' + c'$ (une collision).

En effet, pour que `Alice` modifie son engagement, elle doit trouver une collision dans S afin d'obtenir $(a, b, c) \neq (a', b', c')$ tels que $a + b + c = a' + b' + c' = s$ et $\mathsf{bin}(a'|b'|c')_\pi = z \oplus 1$.

Théorème 5.3.1. *La probabilité pour* `Alice` *de réussir à modifier son engagement dans le protocole* $\mathsf{COMMIT}-\mathsf{3SUM}$ *est inférieure à* $\frac{1}{n^k}$.

Démonstration. Par construction de S, la probabilité p qu'il existe $(a, b, c) \neq (a', b', c')$ tels que $a + b + c = a' + b' + c'$ est limitée par k. On a $p \leq \frac{1}{n^k}$. □

Ainsi, nous pouvons choisir de rendre plus difficile pour `Alice` de tricher en augmentant la taille de la représentation binaires des entiers de l'instance $\mathsf{3SUM}$ via le paramètre k.

La sécurité de l'ouverture dépend de la difficulté qu'a `Bob` pour obtenir $\mathsf{bin}(\mathsf{a}|\mathsf{b}|\mathsf{c})_\pi$ avant une action de `Alice`.

Pour démontrer la sécurité de la mise en gage, nous construisons une réduction nous permettant d'inverser une instance de $\mathsf{3SUM}$ de taille n en temps espéré $o(n^2)$ en utilisant un receveur `Bob` malicieux, capable de retourner z avant l'ouverture par $\mathsf{OPEN}-\mathsf{3SUM}$.

Théorème 5.3.2. `Bob` *malicieux capable d'apprendre* z *avec certitude en un temps* $o(\frac{n^2}{log(n)})$ *peut résoudre le problème* $\mathsf{3SUM}$ *en temps* $o(n^2)$.

Démonstration. Nous utilisons `Bob` pour résoudre une instance de $\mathsf{3SUM}$ de notre choix $I = (A, B, C, s, 1^n)$. Nous générons la chaîne aléatoire $\mathcal{R}$ à l'aide de (A, B, C), comme $\mathcal{R} = \mathsf{bin}(A)|\mathsf{bin}(B)|\mathsf{bin}(C)$, et demandons l'utilisation

du paramètre de sécurité n. Ainsi, `Bob` et `Alice` vont utiliser le triplet d'ensembles de I dans la mise en gage.

Puis, `Alice` se commet en envoyant le message $(s, \pi = 1)$. `Bob` obtient alors $z_i = \mathsf{bin}(a|b|c)_i$.

On recommence alors la procédure avec $i = i + 1$ jusqu'à $i = 3m\mathsf{lb}(n)$ pour apprendre $\mathsf{bin}(a'|b'|c')$ dans son intégralité.
Alors, on a résolu une instance $I = (A, B, C, s, 1^n)$ de 3SUM en temps $o(log(n)\frac{n^2}{log(n)}) = o(n^2)$, en contradiction avec l'hypothèse de complexité sur 3SUM.

□

5.4 Transfert équivoque basé sur 3SUM

Le protocole prend place entre un demandeur `Bob` et une émettrice `Alice`. A leur disposition, une chaîne aléatoire partagée $\mathcal{R}$ pour $\mathsf{INIT} - 3\mathsf{SUM}$ et un protocole de mise en gage en boîte noire.

Protocole 10 Transfert équivoque $\mathsf{OT} - \mathsf{3SUM}$

1. `Bob` et `Alice` s'entendent sur les paramètre de sécurité r, $n = 2^r$ et $k \in \mathbb{N}$, $m = 6 + k$.
2. `Bob` et `Alice` appellent $\mathsf{INIT} - \mathsf{SUM}$ avec n, k et r.
3. `Alice` choisit aléatoirement $t = 2\lceil n^{\frac{3}{2}} \rceil$ triplets dans $A \times B \times C$ et en calcule leurs sommes respectives.
   `Alice` envoie ces sommes à `Bob` dans la liste S.
4. `Bob` choisit aléatoirement des triplets dans $A \times B \times C$, et en calcule leur somme respective jusqu'à trouver $(a, b, c) \in A \times B \times C$ tels que $a + b + c = s \in S$.
5. `Bob` se commet à chaque bit de la représentation binaire de s auprès de `Alice`, soit r mise en gage.
   `Bob` transmet donc à `Alice`

$$K = (\mathsf{COMMIT}(\mathsf{bin}(s)_1), \ldots, \mathsf{COMMIT}(\mathsf{bin}(s)_{\|s\|}))$$

6. `Alice` effectue une permutation aléatoire sur S pour former S'.
7. `Alice` transmet

$$P = ((S'_1, S'_2), \ldots, (S'_{t-1}, S'_t))$$

   à `Bob`.
8. `Bob` cherche le couple $P_\gamma = (s_0, s_1)$ dans P tel que $s \in P_\gamma$. Soit $s = s_w$.
9. `Bob` dévoile à `Alice` les mises en gage des bits communs des éléments de P_γ, soit le message

$$W = \bigcup_i (\mathsf{OPEN}(K_i)) \text{ tels que } \mathsf{bin}(s_0)_i = \mathsf{bin}(s_1)_i$$

10. `Alice` vérifie les mises en gage.
11. `Alice` transmet $\pi \in_r [1, \ldots, 3mr]$ à `Bob`.
    Soient $s_0 = a_0 + b_0 + c_0$ tels que $a_0 < b_0 < c_0$.
    Soient $s_1 = a_1 + b_1 + c_1$ tels que $a_1 < b_1 < c_1$.
12. `Alice` dispose de $\beta_0 = \mathsf{bin}(a_0|b_0|c_0)_\pi$ et $\beta_1 = \mathsf{bin}(a_1|b_1|c_1)_\pi$.
13. `Bob` dipose de β_w.

Ainsi, à la fin du protocole, `Alice` possède les deux bits β_0 et β_1. Il s'agit alors de prouver que `Bob` ne possède qu'un seul de ces deux bits, soit β_w, et qu'`Alice` ne sait quel bit `Bob` possède.

La preuve de sécurité du protocole face à `Alice` est identique au cas des fonctions à sens unique.

Puis, nous pouvons utiliser un `Bob` malicieux, capable de briser le protocole en sa faveur et d'apprendre les deux bits de `Alice` pour résoudre une instance de 3SUM.

Théorème 5.4.1. `Bob` *malicieux capable de briser le protocole en temps* $o(\frac{n^2}{log(n)})$ *peut résoudre le problème* 3SUM *en temps* $o(n^2)$.

Démonstration. Nous utilisons `Bob` pour résoudre une instance de 3SUM de notre choix $I = (A, B, C, s, 1^n)$, et cherchons $(a, b, c) \in A \times B \times C$ tel que $a + b + c = s$. Le protocole de mise en gage est remplacé par un protocole biaisé, où $\mathsf{OPEN} - \mathsf{COM}$ ne nécessite pas la participation de `Bob` pour être appelée, nous permettant d'ouvrir toutes ses mises en gage. Nous utilisons $A \times B \times C$ ainsi que les paramètres n, k et $m = 6 + k$ pour générer la chaîne aléatoire de départ, afin que les ensembles A, B et C soient ceux utilisés dans le protocole.

`Alice` génère S de taille $n - 1$, et transmet $S' = S \cup s$. Ainsi, s est inclus dans l'ensemble S, et il suffira de forcer `Bob` à utiliser cet élément dans la suite du protocole. Ainsi, `Bob` répond normalement par K, qui sont ouvertes grâce à $\mathsf{OPEN} - \mathsf{COM}$, permettant alors de dévoiler l'élément s' inversé par `Bob`. Nous pouvons donc construire P avec un élément $P_i = (s', s)$ et ainsi forcer `Bob` à utiliser l'élément s que l'on cherche à inverser. `Alice` annonce $\pi = 1$, et ainsi $\beta_0 = \mathsf{bin}(a'|b'|c')_\pi$ et $\beta_1 = \mathsf{bin}(a|b|c)_\pi$ sont connus par `Bob`. Nous recommençons alors la procédure avec $\pi = \pi + 1$ jusqu'à $\pi = 3m\mathsf{lb}(n)$ pour apprendre $\mathsf{bin}(a|b|c)$, et donc a, b, c.

Ainsi, si le temps pour `Bob` pour briser le protocole est de $o(\frac{n^2}{log(n)})$, en répétant la procédure pour chaque bit de $\mathsf{bin}(a|b|c)$, il aura résolu une instance de 3SUM en temps $o(n^2)$.

□

Ainsi, autant dans le protocole d'échange de clef que dans celui de transfert

équivoque, nous constatons donc que si protocole s'effectue en temps t par les participants honnêtes, un adversaire ou participant malhonnête ne peut faire mieux que de travailler en $t^{\frac{4}{3}}$ (en omettant les facteurs logarithmiques).

Conclusion

Dans ce travail, nous avons présenté plusieurs protocoles cryptographiques. Tout d'abord, nous avons montré comment utiliser des permutations à sens unique afin d'effectuer un transfert équivoque. Sous cette seule hypothèse, notre protocole demande un travail quadratique t^2 pour être brisé par un demandeur malhonnête capable d'effectuer le protocole de manière honnête en temps t.

Puis, nous avons montré qu'il était possible d'effectuer des échanges de clefs, des mises en gage et des transferts équivoques en utilisant comme primitive, non pas une fonction à sens unique, mais le problème polynomial 3SUM. Dans toutes ces situations, nous nous sommes cependant limités à une définition de sécurité plus faible que celle couramment utilisée. En effet, dans le cas des échanges de clef et des transferts équivoques avec 3SUM, si le temps honnête pour réaliser le protocole est de t, on peut s'attendre à ce qu'un adversaire doive travailler en temps $t^{\frac{4}{3}}$ pour briser les protocoles.

Plusieurs questions restent ouvertes. Dans le cas du transfert équivoque avec permutations à sens unique, nous pouvons nous demander si ce protocole est optimal, ou si des améliorations sont encore possibles. Dans le cas des protocoles utilisants la primitive 3SUM, nous nous sommes confrontés à plusieurs problèmes. Tout d'abord, la création de l'instance de 3SUM dans les protocoles de mise en gage et de transfert équivoque requière l'utilisation d'aléa partagé. Nous n'avons en effet pas réussi à construire une méthode permettant aux participants aux protocoles de générer eux-mêmes les entiers nécessaires à 3SUM. Nous ne savons pas si cette contrainte est essen-

tielle à un protocole basé sur 3SUM, ou si il s'agit d'une lacune de nos protocoles, qui pourraient alors être améliorés.

Finalement, nous nous sommes concentrés sur le problème 3SUM, possiblement le plus simple à appréhender de sa classe. Il en existe cependant plusieurs autres qui pourraient être explorés pour simplifier, ou améliorer certains protocoles. Récemment, Emanuele Viola [Vio11] a proposé le problème 3XOR, où les entiers sont remplacés par des chaînes de bits, et les sommes par des ou-exclusifs sur les chaînes. L'utilisation de ce problème, conjecturé également comme étant $\mathsf{3SUM} - \mathsf{Hard}$, pourrait offrir des facilités dans le traitement des protocoles, et dans l'extraction des bits difficiles, résolvant ainsi un problème important de nos protocoles.

Malgré ces questions, les protocoles qui ont été présentés ici sont —à notre connaissance—, les premiers protocoles cryptographiques pour le calcul biparti utilisant comme hypothèse de complexité la difficulté d'un problème polynomial. Nous pouvons donc espérer par la suite trouver d'autres problèmes de ce types utiles à la création de nouveaux protocoles cryptographiques.

Bibliographie

[BST09] Gilles Brassard, Louis Salvail et Alain Tapp : Oblivious transfer à la Merkle. *In Proceedings of the 2009 Third International Conference on Quantum, Nano and Micro Technologies*, pages 102–108, 2009.

[Cré88] Claude Crépeau : Equivalence between two flavours of oblivious transfers. *In Proceedings of CRYPTO '87*, pages 350–354, 1988.

[GL89] Oded Goldreich et Leonid A. Levint : A hard-core predicate for all one-way functions. *In Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, 1989.

[GO95] Anka Gajentaan et Mark H Overmars : On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry*, pages 165–185, 1995.

[Gol01] Oded Goldreich : *Foundations of Cryptography : Volume 1, Basic Tools*. Cambridge University Press, 2001.

[Hof09] Michael Hoffman : Computational geometry. Cours à l'Ecole Polytechnique Fédérale de Zürich, 2009.

[IR89] Russell Impagliazzo et Steven Rudich : Limits on the provable consequences of one-way permutations. *In Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 44–61, 1989.

[Kah96] David Kahn : *The Codebreakers : The Story of Secret Writing.* Scribner, 1996.

[Kil88] Joe Kilian : Founding cryptography on oblivious transfer. *In Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31, 1988.

[Kin04] James King : A Survey of 3SUM-Hard Problems, 2004.

[KL08] Jonathan Katz et Yehuda Lindell : *Introduction to modern cryptography.* Chapman & Hall/CRC, 2008.

[Mer78] Ralph C. Merkle : Secure communications over insecure channels. *Commun. ACM*, 1978.

[QQQ+90] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michael Quisquater, Louis C. Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, Soazig Guillou et Thomas A. Berson : How to explain zero-knowledge protocols to your children. *In Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, pages 628–631, 1990.

[Vio11] Emanuele Viola : Reducing 3XOR to listing triangles, an exposition. *Electronic Colloquium on Computational Complexity (ECCC)*, 2011.

[Yao82] Andrew C. Yao : Protocols for secure computations. *Foundations of Computer Science, IEEE Annual Symposium on*, 1982.