École normale supérieure **2019-2020**
Département d'informatique

Introduction à la cryptologie
TD n° 9 : Correction.

Exercice 1. On définit D et T suivant l'indication. La réponse à la question d'Alice est oui ssi T divise $D \circ P$. Cela est équivalent à : il existe un polynôme H de degré au plus 9000 tel que $TH = D \circ P$. L'idée est que pour vérifier cette égalité polynomiale, il suffit de vérifier l'égalité en un point uniforme. En effet, comme les polynômes sont de degré au plus 10000, soit ils sont égaux partout, soit ils sont égaux sur au plus 10000 points. Or on vit dans $\mathbb{Z}_p$ avec $p > 10^6$, donc dans le second cas, les polynômes sont distincts sur plus de 99% de leurs entrées.
En fin de compte, il suffit à Alice de poser deux question à Bob. D'abord, elle tire α uniformément dans $\mathbb{Z}_p$. Ensuite, elle demande : (1) $P(\alpha)$; et (2) $H(\alpha)$ pour H au choix de Bob de degré au plus 9000. Elle accepte la preuve de Bob ssi $T(\alpha)H(\alpha) = D(P(\alpha))$. Suivant l'argumentaire plus haut, si la réponse à la question est oui, il est possible à Bob de calculer un H de degré au plus 9000 et donc une valeur $H(\alpha)$ appropriée. Réciproquement si la réponse est non, pour tout choix de H de degré au plus 9000 que Bob peut faire, Alice verra que l'égalité $T(\alpha)H(\alpha) = D(P(\alpha))$ n'est pas vérifiée, sauf avec probabilité moins de 1% (sur son choix de α).

Exercice 2 (Comment énerver oncle Bob, premier chapitre)**.**

1. Le but d'Alice est de prouver qu'elle a trouvé une solution, pas juste qu'il en existe une. D'ailleurs, si l'on fait confiance à la source de sudoku utilisée, on sait déjà qu'il existe une solution, donc il n'y a rien à prouver de ce côté.
2. Bob demande à Alice de dévoiler son commit sur un des ensembles de cases suivants : soit une des colonnes, soit une des lignes, soit un des blocs 3×3, soit l'ensemble des cases non-vides de T (le problème de sudoku qu'Alice prétend résoudre). Il y a donc $3 \cdot 9 + 1 = 28$ questions possibles que Bob peut poser. Si Alice ne connaît pas réellement de solution au problème de sudoku T, lorsqu'elle répond à la question de Bob en dévoilant son commit, la réponse est incorrecte pour au moins une des 28 questions possibles (en effet, par contraposée, si les valeurs sur lesquelles elle s'est engagée satisfont toutes les 28 conditions, elles forment une réponse correcte au sudoku, modulo la permutation σ). Si Bob tire uniformément parmi les 28 questions qu'il peut poser, si Alice triche Bob a donc au moins une chance sur 28 de s'en rendre compte.
3. Alice peut tricher avec probabilité (au plus) 27/28 à chaque étape du protocole. Il suffit donc de répéter le protocole q fois pour q tel que $(27/28)^q \leq 1/100$, donc $q \geq \log(100)/\log(28/27) \approx 126,6$. On peut d'ailleurs approximer cette quantité par $\log(100)/\log(1 + 1/27) \approx 27 \log(100)$.
4. Avant la réponse formelle, donnons d'abord une réponse intuitive. Lorsque Bob demande à voir les valeurs d'une ligne, une colonne ou un bloc, il reçoit une permutation uniformément aléatoire des chiffres de 1 à 9. En particulier, cette permutation est indépendante de la solution S (ici, on utilise implicitement une propriété de groupe : composer une permutation uniformément aléatoire avec un élément quelconque donne une permutation uniformément aléatoire). Bob n'apprend donc rien. De même s'il demande à voir les cases non-vides de T. Plus formellement, la définition de zero-knowledge nous demande de construire un simulateur qui produit un transcript indistinguable du transcript d'une interaction réelle entre Alice et Bob. Pour ce faire, le simulateur tire d'abord uniformément la question que Bob va poser, parmi les 28 possibles. Supposons que Bob demande la première ligne (les autres cas sont similaires). Alors le simulateur crée un commit sur une permutation uniforme des chiffres $[\![1, 9]\!]$ pour la première ligne, et des commits sur des valeurs quelconques pour les autres cases. Pour terminer le transcript, le simulateur ajoute la question de Bob (qui porte sur la première ligne), puis ajoute le dévoilement des

commits de la première ligne. Pour tout ce qui concerne la première ligne, ce comportement est indistinguable d'une exécution légitime du protocole (les distributions sont même identiques). La seule différence porte sur les commits des autres cases. Mais le propriété d'un schéma de commitment indique que ces commits ne révèlent rien sur les valeurs correspondantes (tant que le commit n'est pas dévoilé) ; ces commits sont donc indistinguables (cette fois, en un sens calculatoire et non l'identité des distributions) des commits d'une exécution réelle du protocole. De cette manière, la propriété de zero-knowledge se ramène à la sécurité du schéma de commitment.

Exercice 3 (Sudoku zero-knowledge *sans interaction*)**.**

1. Alice exécute des itérations du protocole précédant, en remplaçant les bits aléatoires envoyés par Bob (qui expriment, à chaque itération, quelle question parmi les 28 Bob souhaite poser) par la sortie d'une fonction de hachage, dont l'entrée est l'intégralité de la communication jusqu'à ce moment (i.e. tous les bits échangés entre Alice et Bob jusqu'à ce point). Pour tricher, Alice procède de la manière suivante. À chaque itération, elle tire uniformément la question que Bob va poser. Puis, elle commite sur une permutation aléatoire des chiffres qui va satisfaire cette question particulière, et commite sur des valeurs quelconques pour le reste—exactement comme le simulateur de l'exercice précédent. Il se peut bien sûr qu'une fois ces commitments effectués, la fonction de hachage lui donne une question qui n'est pas celle qu'Alice voulait. Dans ce cas, elle recommence simplement jusqu'à ce que ce soit le cas : cela coûtera environ 28 essais par itération, donc un surcoût en temps non significatif par rapport à une exécution honnête.

   Pour empêcher ce comportement, il suffit que H prenne en entrée d'emblée tous les commitments de toutes les itérations du protocole. Plus précisément, si C est l'ensemble de ces commitments, on peut fixer l'aléa de Bob pour la k-ième itération du protocole à $H(C \parallel k)$. De cette manière, pour qu'Alice puisse tricher, il faut que la sortie de H lui donne la question qui l'arrange pour toutes les itérations du protocole simultanément. Si ce n'est pas le cas, son seul recours est de modifier C, mais cela modifie les questions de Bob pour toutes les étapes du protocole—tandis que dans le cas plus haut, Alice pouvait modifier la réponse de Bob à une question isolée sans modifier sa réponse aux questions précédentes.
2. Ce choix de nombre d'itérations permettait à Bob de s'assurer que si Alice ment, il s'en rende compte avec probabilité au moins 1%. Dans le cas d'un protocole non-interactif, ce n'est pas acceptable : en effet, il suffirait à Alice de réessayer 100 fois jusqu'à trouver une preuve qui marche. Dans le cas interactif, ce n'est pas possible parce que si elle réessayait le protocole 100 fois, Bob s'en rendrait compte (au moins dans certains scénarios, où cette borne de 1% serait acceptable). Dans le cas non-interactif, il est donc impératif que la probabilité de triche pour Alice soit négligeable au sens cryptographique, pas seulement faible : par exemple 2^{-128}. Noter que cela n'implique pas un nombre exponentiel d'itérations du protocole, puisque la probabilité de triche décroît exponentiellement avec le nombre d'itérations.

Exercice 4 (Oncle Bob, version plus efficace : preuves d'(in)égalités)**.**

1. Bob vérifie $x_b \oplus y_b = x_0 \oplus y_0$. Si $b = 0$ cette égalité est triviale. Si $b = 1$ elle découle de $x_0 \oplus x_1 = y_0 \oplus y_1$. Si cette dernière égalité était fausse, un des deux choix de b ne passerait pas la vérification, donc Bob s'en rend compte avec probabilité 1/2.
2. Bob vérifie $x_b \oplus y_b = x_0 \oplus y_0 \oplus b$.
3. Si on répète le protocole pour deux choix de b différents, tous les commitments sont révélés, donc Bob apprend la valeur de x (et de y). On perd donc la propriété de zero-knowledge. Dans d'autres cas (par exemple l'isomorphisme de graphe, discuté dans un TD précédent), on avait pu répéter le protocole pour réduire la probabilité de triche, mais une différence cruciale est que la première étape de ces protocoles étaient probabilistes, et consistaient en la production d'une instance « aléatoire »de problème. Ici Alice envoie simplement $x_0 \oplus y_0$, donc répéter le

protocole, c'est l'exécuter pour des b différents. Pour résoudre ce problème, une solution est en gros de re-randomiser l'instance à la première étape, ce qui sera fait dans la suite.

4. Si la conjonction $c_x \sim c_y \wedge c_x \sim c'_x \wedge c_y \sim c'_y$ est fausse, au moins une des trois formules atomiques et fausse. Il y a une chance sur trois que Bob demande la preuve de l'égalité correspondante. S'il le fait, il y a ensuite une change sur deux qu'il se rende compte que l'égalité est fausse (par la question 1). En fin de compte, il y a donc au moins une chance sur six que Bob se rende compte de la triche.
5. Suivant l'indication, Alice crée un grand nombre de commitments $c^i_x \sim c_x$ et $c^i_y \sim c_y$, pour $i \in [\![1, n]\!]$. Bob choisit uniformément $k \in [\![1, n]\!]$. Bob envoie ensuite n valeurs $a_i \in \{1, 2, 3\}$ uniformément aléatoires, pour $i \neq k$. Si $a = 1$ (resp. $a = 2$, $a = 3$), Bob vérifie $c^i_x \sim c_x$ (resp. $c^i_y \sim c_y$, $c^i_x \sim c^i_y$) suivant le protocole d'origine. Pour chaque i Bob vérifie au choix $c^i_x \sim c_x$ ou $c^i_y \sim c_y$ ou $c^i_x = c^i_y$. Par la suite, c^k_x et c^k_y seront utilisés comme nouveaux encodages de x et y.
   Il reste à montrer que ce protocole fonctionne. Il est clair qu'il est correct : si $c_x \sim c_y$ Bob accepte. Le point crucial est de montrer que si ce n'est pas le cas, la probabilité que Bob accepte décroît exponentiellement avec n (une décroissance exponentielle est souhaitable parce que par exemple le coût en communication du protocole croît linéairement avec n). Si $c_x \sim c_y$ est faux, pour chaque $i \neq k$ il y a une probabilité au moins $1/6$ que Bob s'en rende compte. La probabilité que Bob accepte en fin de compte est donc au plus $(5/6)^{n-1}$.
6. Supposons maintenant qu'on sait de manière similaire construire une preuve zero-knowledge d'inégalité dans un ensemble quelconque. Comment modifier le protocole de l'exercice 1 pour réduire notoirement la quantité (nombre de bits) d'informations échangées entre Alice et Bob ?

Exercice 5 (*Probabilistically Checkable Proofs* et SNARGs)**.** Un langage $\mathcal{L}$ admet une *probabilistically checkable proof* ssi il existe un algorithme P en temps polynomial probabiliste à sortie dans $\{0, 1\}$ tel que : pour tout $x \in \mathcal{L}$, il existe une « preuve » y telle que $P(x, y)$ renvoie toujours 1 (accepte), et pour tout $x \notin \mathcal{L}$, et pour tout y, $P(x, y)$ renvoie 0 avec probabilité au moins $1/2$. Plus exactement, on dit que $\mathcal{L}$ est dans $\mathrm{PCP}[A, B]$ ssi il existe un P comme précédemment qui de plus : (1) ne consomme que A bits d'aléa ; et (2) lit au plus B bits dans y.

1. Soit n la taille de l'entrée x. Montrer $\mathrm{PCP}[0, 0] = \mathrm{PCP}[O(\log n), 0] = \mathrm{P}$.
2. Montrer $\mathrm{PCP}[O(\log n), \mathrm{poly}(n)] = \mathrm{NP}$.

Le théorème PCP (hautement non-trivial) donne $\mathrm{PCP}[O(\log n), O(1)] = \mathrm{NP}$. On souhaite utiliser ce résultat pour déduire une technique de preuve interactive succinte. Dans ce but, pour prouver $x \in \mathcal{L}$ en connaissant une preuve y, Alice construit un arbre de Merkle sur y.

3. Comment construire une preuve de connaissance succinte à partir de cet arbre de Merkle ?
4. Montrer que le protocole est succint, au sens où la quantité de bits envoyés par Alice pour convaincre Bob est $O(\lambda n)$, où n est la taille de l'entrée $x \in \mathcal{L}$, et λ est la taille de sortie de la fonction de hachage utilisée dans l'arbre de Merkle.