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 (*Probabilistically Checkable Proofs* et SNARGs)**.**

1. Let $\mathcal{L} \in \mathrm{P}$. Then the algorithm P checks whether $x \in \mathcal{L}$ using the polynomial decider for $\mathcal{L}$. This algorithm is in $\mathrm{PCP}[0,0] \subseteq \mathrm{PCP}[\log(n),0]$.

   If on the other hand $\mathcal{L} \in \mathrm{PCP}[0,0]$, then there is a polynomial algorithm P that decides $\mathcal{L}$ (without randomness). If P uses $O(\log n)$ randomness, then we can use P to define a deterministic polynomial decider D. The decider D tries all possible randomness r and then runs $P(x, \bot; r)$. Note that there is no proof required to run P, as it never reads any bits from the proof. D rejects iff there is one run that rejects. As $2^{O(\log n)} = \mathsf{poly}(n)$ and there must be at least one r such that $P(x, \bot; r)$ rejects if $x \notin \mathcal{L}$, we have $\mathcal{L} \in \mathrm{P}$.
2. If $\mathcal{L} \in \mathrm{NP}$, then there is some witness y that shows that $x \in \mathcal{L}$. This y suffices as proof, as it can be verified in $\mathsf{poly}(n)$ time whether y is a valid witness (without randomness).

If $\mathcal{L} \in \mathrm{PCP}[O(\log n), \mathsf{poly}n]$, then $w = \{r_i, y_i\}_i$ is a witness, where $r_i \in \{0,1\}^{O(\log n)}$ and y_i are the bits read by $P(x, y; r_i)$ in the proof y when run with randomness r_i. As above, we can run P on all possible inputs w and the result follows.

3. Run P and ask the prover to open the tree at the corresponding positions whenever P tries to read some bit from y. It is a proof of knowledge with soundness error $1/2$ which follows from the properties of P.
4. Each position in the tree can be opened with $O(\lambda \log m)$ bits (as each co-path has length $O(\log m)$ and consists of hash functions), and at most a constant number of positions need to be opened.

Exercice 5 (Échauffement)**.**

1. Si les deux propriétés sont vraies pour tout générateur, trivialement il existe un générateur tel qu'elles sont vraies (on suppose $\mathbb{G}$ non vide). Réciproquement, supposons qu'il existe un générateur g tel que les deux propriétés sont vraies. Soit $h = g^x$ un générateur de $\mathbb{G}$. On a :
   - $e(h,h) = e(g,g)^{x^2} \neq 1$; en effet $e(g,g) \neq 1$, donc comme $\mathbb{G}_T$ est cyclique, $e(g,g)$ est générateur, et d'autre part $x \neq 0$ puisque g^x est générateur, donc $e(g,g)^{x^2} \neq 1$.
   - $e(h^a, h^b) = e(g,g)^{x^2 ab} = e(h,h)^{ab}$.
2. Soit un triplet (g^a, g^b, g^c). On veut vérifier si $c = ab$ (modulo p). Pour cela, il suffit de vérifier :

$$e(g^a, g^b) \stackrel{?}{=} e(g^c, g).$$

   En effet, l'égalité est vraie ssi $e(g,g)^{ab} = e(g,g)^c$, ssi $ab = c \bmod p$ parce que $e(g,g) \neq 1$ et le groupe est cyclique, ce qui implique que $e(g,g)$ est générateur donc l'égalité $e(g,g)^{ab} = e(g,g)^c$ ne peut avoir lieu que si les exposants sont égaux mod p.

Exercice 6 (Échange de clef sans interaction à trois participants)**.**

1. A, B, C tirent respectivement une valeur uniformément aléatoire a, b, c, et publient respectivement g^a, g^b, g^c. Pour trouver le secret commun A, B, C calculent respectivement $e(g^b, g^c)^a$, $e(g^a, g^c)^b$, $e(g^a, g^b)^c$. Les propriétés du couplage e garantissent que ces trois valeurs sont égales à $e(g,g)^{abc}$, les trois parties obtiennent donc bien une valeur commune. D'autre part, un adversaire qui observe les communications obtient g^a, g^b, g^c. Retrouver le secret commun $e(g,g)^{abc}$ est difficile pour lui, puisqu'il s'agit exactement d'une instance du problème de Diffie-Hellman bilinéaire.
2. Supposons qu'on a $n+1$ participants $A_1, \ldots, A_{n+1}$. Le participant A_k tire a_k uniformément aléatoirement et publie g^{a_k}. Après avoir reçu g^{a_i} pour $i \neq k$ des autres participants, il calcule le secret commun :

$$e_n(g_1^a, \ldots, g^{a_{k-1}}, g^{a_{k+1}}, \ldots, g^{a_{n+1}})^{a_k} = e_n(g, \ldots, g)^{a_1 \cdots\cdots a_{n+1}}.$$

   La sécurité de ce protocole repose sur une variante « multilinéaire » du problème de Diffie-Hellman : calculer $e(g, \ldots, g)^{a_1 \cdots\cdots a_{n+1}}$ à partir de g^{a_1}, …, $g^{a_{n+1}}$.

Exercice 7 (Sécurité des signatures de Boneh-Boyen)**.**

1. Pour une signature légitime on a :

$$e(\sigma, ug^m) = e(g^{(x+m)^{-1}}, g^{x+m}) = e(g,g) = g_T.$$

   La signature est donc acceptée.

2. Soit $(h_0, \ldots, h_q) \in \mathbb{G}^{q+1}$. Pour vérifier qu'il s'agit d'une instance de q-SDH, on procède par récurrence. Si $q = 0$, il suffit de vérifier que h_0 est générateur, donc simplement vérifier $h_0 \neq 1$ puisqu'on est dans un groupe cyclique. Pour $q > 0$, supposons qu'on a vérifié par récurrence que $(h_0, \ldots, h_{q-1})$ est une instance de $(q-1)$-SDH. Alors il suffit de vérifier $e(h_0, h_q) = e(h_1, h_{q-1})$. En effet, posons $h_0 = g$, $h_1 = g^x$, $h_{q-1} = g^y$, $h_q = g^z$. Alors l'égalité précédente est vraie ssi $z = xy \bmod p$. Or par récurrence on sait que $y = x^{q-1}$. On déduit $z = g^x$, CQFD.

   Supposons maintenant qu'on a une solution prétendue à une instance q-SDH $(g, g^x, g^{x^2}, \ldots, g^{x^q})$. On a donc une paire (m, h) et on souhaite vérifier $h = g^{(x+m)^{-1}}$. Pour cela, il suffit de vérifier $e(h, g^x g^m) = e(g, g)$. Noter que h, g^x, g, m sont connus.

3. (a) En tirant plein parti de notre connaissance magique de x, lorsque l'adversaire demande la signature de m_k, nous pouvons lui renvoyer :

$$\sigma_i = h^{(x+m_k)^{-1}} = g^{\prod_{i \neq k}(x+m_i)}.$$

   En fin de compte, l'adversaire forge une signature (m^*, σ^*) pour $m^* \notin \{m_i\}$. On a $\sigma^* = h^{(x+m^*)^{-1}}$. Pour répondre à l'instance q-SDH, on voudrait connaître $g^{(x+m^*)^{-1}}$ plutôt que $h^{(x+m^*)^{-1}}$. Pour cela, il suffit d'observer :

$$g^{(x+m^*)^{-1}} = \left(h^{(x+m^*)^{-1}}\right)^{\prod (x+m_i)^{-1}}.$$

   La réponse à l'instance q-SDH est $(m^*, g^{(x+m^*)^{-1}})$.
   Remarque. Bien sûr avec notre connaissance magique de x on aurait pu calculer cela directement. Mais on souhaite quand même utiliser la sortie de l'adversaire, pour mieux se préparer au raisonnement qui suit, où on ne connaît plus x.

   (b) Une observation critique est la suivante : l'instance q-SDH $(g, g^x, g^{x^2}, \ldots, g^{x^q})$ nous permet de calculer $g^{P(x)}$ pour n'importe quel polynôme P de degré au plus q à coefficients connus. En effet, si $P(X) = \sum_{i \leq q} a_i X^i$, alors

$$g^{P(x)} = \prod_{i \leq q} (g^{x^i})^{a_i}.$$

   En particulier, même sans connaître x, l'instance q-SDH nous permet de calculer $h = g^{\prod(x+m_i)}$: en effet, en développant $\prod(X + m_i)$ on obtient un polynôme de degré q, dont on peut calculer les coefficients explicitement (puisque les m_i sont connus).

   (c) Avec la question précédente, on sait fournir à l'adversaire une signature valide des messages m_i. L'adversaire nous renvoie alors une signature $\sigma^* = h^{(x+m^*)^{-1}}$ d'un message $m^* \notin \{m_i\}$. Comme plus haut, il nous reste à déduire $g^{(x+m^*)^{-1}}$ pour avoir une solution $(m^*, g^{(x+m^*)^{-1}})$ à l'instance q-SDH. Pour calculer cette valeur, l'idée générale est la même que dans la question précédente, mais avec quelques étapes supplémentaires. On a

$$\begin{aligned}
h^{(x+m^*)^{-1}} &= g^{\frac{\prod(x+m_i)}{x+m^*}} \\
&= g^{\frac{P_1(x+m^*)}{x+m^*}} && \text{pour un certain polynôme } P_1 \text{ connu de degré } q \\
&= g^{P_2(x+m^*)+\frac{c}{x+m^*}} && \text{pour } P_2 \text{ connu de degré } q-1 \text{ et une constante } c.
\end{aligned}$$

   Par la question précédente, nous savons calculer $g^{P_2(x+m^*)}$ (noter que m^* est connu), donc on trouve :

$$g^{(x+m^*)^{-1}} = \left(h^{(x+m^*)^{-1}} g^{-P_2(x+m^*)}\right)^{c^{-1}}.$$