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

Introduction à la cryptologie
TD n° 9 : Zero-knowledge [1], suite et fin.

Exercice 1. Bob connaît une suite $x_1, \ldots, x_{1000}$ de 1000 entiers compris entre 1 et 10. Il met cette suite sous la forme d'un polynôme $P \in \mathbb{Z}_p[X]$ de degré au plus 1000, tel que $P(i) = x_i$ pour $1 \leq i \leq 1000$. Ici, p est un nombre premier quelconque suffisamment grand (mettons $p > 10^6$). Alice veut vérifier que tous les entier x_i de Bob sont bien entre 1 et 10. Pour cela, en utilisant les techniques du cours, elle dispose de deux outils (ici légèrement simplifiés) :

— Elle peut choisir $\alpha \in \mathbb{Z}_p$, et interroger Bob sur la valeur de $P(\alpha)$, sans rien révéler à Bob (même pas α ou $P(\alpha)$).
— Elle peut choisir un entier n quelconque et $\alpha \in \mathbb{Z}_p$, et demander à Bob la valeur $H(\alpha)$ pour un polynôme H au choix de Bob de degré au plus n. Toujours sans rien révéler à Bob.

Décrire un protocole qui permet à Alice de s'assurer que les x_i sont tous entre 1 et 10, avec probabilité au moins 99%, en posant seulement deux questions à Bob.
Indication : On peut d'abord définir les polynômes $D = (X-1)(X-2)\cdots(X-10)$ et $T = (X-1)(X-2)\cdots(X-1000)$, puis reformuler la question d'Alice (« est-ce que $P(i) \in \{1, \ldots, 10\}$ pour $i \in \{1, \ldots, 1000\}$ ») sous la forme : « est-ce que un certain polynôme divise un autre polynôme », où les deux polynômes en question sont une expression simple formée à partir de D, T, P.

Exercice 2 (Comment énerver oncle Bob, premier chapitre)**.** Alice et son oncle Bob sont en vacances à la plage, et s'amusent à résoudre des sudokus. Un beau jour, oncle Bob tombe sur un sudoku particulièrement difficile, qu'il n'arrive pas à résoudre. Il le montre à Alice. Alice parvient à le résoudre, et le lendemain, elle l'annonce à Bob. Once Bob est vexé, parce qu'il y a passé plusieurs jours ; il ne la croit pas. Alice est piquée : elle veut alors prouver à oncle Bob qu'elle a réussi à résoudre le sudoku, sans rien lui apprendre sur la solution.
Soit T l'instance du problème sudoku, i.e. une grille 9×9 dont certaines cases contiennent un chiffre dans $[\![1, 9]\!]$, et les autres sont vides. Soit S la solution trouvée par Alice, i.e. une grille 9×9 contenant des chiffres dans $[\![1, 9]\!]$, tels que :

(a) Les chiffres apparaissant dans chacune des 9 lignes de S sont distincts. De même pour les 9 colonnes, et les 9 sous-grilles 3×3 de S.
(b) Les chiffres de S sont égaux à ceux de T partout où les cases de T sont non-vides.

1. Pourquoi utiliser une preuve *de connaissance* zero-knowledge et pas simplement une preuve zero-knowledge ?

Alice tire une permutation aléatoire $\sigma : [\![1, 9]\!] \to [\![1, 9]\!]$. Soit $\sigma(S)$ la grille obtenue en appliquant σ à chaque case de S. Alice commite [2] sur chaque valeur de $\sigma(S)$ (par exemple, pour chaque case contenant un chiffre c, elle tire un élément de $r \in \{0,1\}^{128}$ uniformément aléatoirement et publie $H(r \parallel \sigma(c))$, pour une fonction de hachage fixée H, et où $\parallel$ dénote la concaténation).

2. Finir ce protocole : une fois les *commitments* publiés, quelle(s) question(s) Bob peut-il poser à Alice pour s'assurer que la solution S trouvée par Alice remplit les conditions (a) et (b) (sans apprendre la solution)—de sorte que, si une des conditions n'est pas respectée, il s'en rend compte avec probabilité au moins 1/28 ?
3. Combien de fois faut-il répéter le protocole pour que si Alice a triché, Bob puisse le découvrir avec probabilité au moins 99% ? Note qui pourra (ou pas) être utile pour approximer le résultat : $\log(100) \approx 4.6$.

1. « divulgation nulle de connaissance »
2. du verbe franglais *commiter*.

4. Prouver que le protocole est *zero-knowledge*.

Exercice 3 (Sudoku zero-knowledge *sans interaction*)**.** Pour transformer un protocole zero-knowledge tel que celui de l'exercice précédent en un protocole non-interactif, on peut utiliser l'heuristique de Fiat-Shamir : au lieu que Bob envoie son choix de bits aléatoires à chaque itération du protocole, ces bits sont choisis par une fonction déterministe pseudo-aléatoire, typiquement une fonction de hachage. Fixons une telle fonction de hachage H.

1. Alice procède de la manière suivante : elle exécute le protocole interactif de l'exercice précédent, mais à chaque nouvelle itération, elle remplace les inputs de Bob par la sortie de H, avec comme entrée le transcript de toute la communication qui précède. Montrer que ce protocole permet à Alice de donner une preuve fausse. Quelle modification simple faut-il faire pour corriger le problème ?
2. Supposons que le protocole est itéré $\log(100)/\log(1+1/27)$ fois. Que pensez-vous de la sécurité du protocole ?

Exercice 4 (Oncle Bob, version plus efficace : preuves d'(in)égalités)**.** Pour améliorer la performance du protocole précédent, on va s'intéresser à des preuves zero-knowledge d'égalité et inégalité. Pour cela, on utilise un schéma de *commitment* classique, où $\mathsf{com}(x)$ désigne un commitment sur une valeur x. On définit l'encodage d'un bit x comme $c_x = (\mathsf{com}(x_0), \mathsf{com}(x_1))$, où x_0 est un bit uniforme et $x_1 = x \oplus x_0$. Noter que l'encodage est probabiliste. On écrira $c_x \sim c_y$ pour signifier que le bit encodé par c_x et celui encodé par c_y sont égaux, i.e. $x_0 \oplus x_1 = y_0 \oplus y_1$.
Alice souhaite prouver à Bob que deux bits encodés sont égaux, sans révéler leur valeur, i.e. prouver $c_x \sim c_y$ en zero-knowledge. Le protocole est le suivant. Alice envoie la valeur $x_0 \oplus y_0$. Bob répond par un bit b uniforme. Alice révèle son commitment (au sens de com) sur x_b et y_b.

1. Comment Bob peut-il se convaincre que $c_x \sim c_y$? (avec probabilité de se tromper au plus 1/2.)
2. Comment modifier le protocole pour prouver l'inégalité entre les deux bits ?

On remarque qu'après une exécution du protocole, les encodages doivent tous être « jetés » : ils ne sont pas réutilisables dans une autre preuve. Pour pallier à ce problème, on modifie le protocole comme suit.

— Alice génère deux nouveaux encodages c'_x, c''_x de x, et deux nouveaux encodages c'_y, c''_y de y.
— Bob envoie une valeur $a \in \{1,2,3\}$ et un bit b, uniformément aléatoires.
— Si $a = 1$, Alice prouve $c_x \sim c_y$. Si $a = 2$, Alice prouve $c_x \sim c'_x$ et $c_x \sim c''_x$. Si $a = 3$, Alice prouve $c_y \sim c'_y$ et $c_y \sim c''_y$.
— Par la suite, ce seront les encodages c''_x et c''_y qui seront utilisés pour représenter x et y, sauf si $a = 3$, auquel cas on utilisera c'_x et c'_y. Les autres encodages sont « jetés » et ne joueront plus aucun rôle.

En résumé, Alice prouve la conjonction $c_x \sim c_y \ \wedge (c_x \sim c'_x \wedge c_x \sim c''_x) \wedge (c_y \sim c'_y \wedge c_y \sim c''_y)$.

3. Que se passe-t-il si on tente de simplifier le protocole en éliminant l'utilisation de c''_x et c''_y ?
4. Si la preuve est fausse (au sens où la conjonction ci-dessus est fausse), donner une borne supérieure sur la probabilité qu'elle soit acceptée par Bob.
5. 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.