UCBL - Master Info
Cryptographie

Correction TD 5

Exercice 1.

1 - Alice reçoit $f_A(x_A, x_B) = x_A + x_B$ de la part de la partie de confiance $\mathfrak{T}$. Elle peut donc en déduire $x_B = f_A(x_A, x_B) - x_A$.

2 - L'indépendance des entrées n'est pas garantie (premier critère de la définition 9). Ainsi, Alice peut, par exemple, systématiquement choisir $x_A = -x_B$ afin que Bob retourne toujours 0 ce qui est impossible dans le cas idéal.

3 - Il suffit de montrer que les 3 critères de la définition 9 sont vérifiés :

1. *Indépendance des entrées. Alice commence par entrer (dans le protocole) la valeur x_A encryptée dans X_A. Bob choisit x_B ne connaissant que X_A c'est à dire sans aucune information sur x_A (sous réserve que Paillier soit sémantiquement sûr).*

2. *Confidentialité des données. D'après 1, les données sont dévoilées dans le cas idéal. On a donc pas à se soucier de ce point ici.*

3. *Correction du protocole. Bob ne peut pas dévier du protocole car il ne fait qu'envoyer x_B à Alice. Supposons donc qu'Alice soit malveillante, i.e. elle envoie les mauvaises valeurs à l'étape 3. Dans ce cas, Bob s'en rendra compte à l'étape 4 et le protocole s'arrête. On est donc ramené au cas où Bob et Alice respectent le protocole et dans ce cas ils retournent bien $x_A + x_B$.*

Exercice 2. *Dans le cas idéal, si Alice entre $x_A = 0$ alors elle reçoit (de la partie de confiance $\mathfrak{T}$) $f_A(0, x_B) = 0x_B = 0$ et, en particulier, n'apprend rien sur x_B. Or dans le protocole réel, si Alice entre $x_A = 0$ elle obtient quand même x_B (à l'étape 2). Ceci suffit à prouver que le protocole n'est pas sûr.*

Exercice 3.
1 - Pour corriger ceci, il faut procéder comme dans le protocole Somme ci-dessus. Lors de la décryption de c effectuée à l'étape 3, Alice obtient la valeur encryptée p et le nombre aléatoire r (voir décryption de Paillier). Au lieu d'envoyer seulement p, elle envoie (p, r) ce qui permettra à Bob de vérifier (à l'étape 4) que c est bien une encryption de p.

2 - Dans le cas idéal, Bob peut entrer des valeurs arbitraires dans le protocole. Cependant, il ne peut pas forcer Alice à retourner une valeur non-nulle α qu'il aurait lui-même choisie. Alors que dans le protocole proposé, à l'étape 2, il peut envoyer une encryption c d'un valeur p qu'il peut arbitrairement choisir.

Exercice 4.

1 - On remarque que M_j est une encryption de $(j-i)\alpha_j$. Aussi M_i encrypte 0 et M_{1-i} encrypte un nombre choisi aléatoirement. On peut donc utiliser les M_j pour masquer la réponse r_j sans connaître i. Pour faire ceci, il suffit de définir R_j de la manière suivante, i.e. $R_j = Paillier.Encrypt(pk, r_j) \oplus M_j$.

2 - Il est facile de voir que si Bob et Alice respectent le protocole alors le protocole est sûr (Bob obtient une seule réponse et Alice ne connait pas la question). Demandons nous maintenant quels avantages Bob ou Alice auraient à dévier du protocole. La seule possibilité pour Bob (de dévier du protocole) est de choisir $i \notin \{0,1\}$. Dans ce cas, il récoit, à l'étape 3, R_0 et R_1 qui encryptent des nombres aléatoires et ne divulgent donc aucune information. Bob n'en tire donc aucun avantage. Que peut faire Alice ? Elle peut, à l'étape 2, envoyer des encryptions R_0 et R_1 de valeurs arbitraires. Cependant, elle peut faire cela dans le cas idéal (elle répond ce qu'elle veut). Même si on ne l'a pas formellement prouvé, on peut raisonnablement penser que ce protocole est sûr.

3 - Immédiat. Il suffit de définir $A_j \leftarrow Paillier.Encrypt(pk, -j)$, $M_j = \alpha_j \circ (I \oplus A_j)$ et $R_j = M_j \oplus Paillier.Encrypt(pk, r_j)$.

Exercice 5.

1 - Le protocole est correct car la clause j est satisfaite si et seulement si C_j encrypte 0. Cependant, si elle n'est pas satisfait, C_j encrypte une valeur qui pourrait divulger de l'information. Pour pallier ceci, Il faut multiplier C_j par un nombre choisi aléatoirement. Comme f est un arbre de désision, il y a au plus une clause qui est satisfaite. Cependant, en l'état, on sait laquelle est satisfaite (la première, la deuxième ...) ce qui peut déliver de l'information sur f. Pour empécher ceci, on peut envoyer une permutation aléatoirement choisie des C_j. Pour finir, le nombre de clauses est divulguée. On pourrait imaginer ajouter des clauses factices (jamais vérifiées) si ce nombre est trop petit.

Retrouver f en $r+1$ requêtes. Pour chaque clause j, à chaque requête $(x_1, \ldots, x_r)$, Bob reçoit une évaluation de $v_j(x_1, \ldots, x_r) = a_{j1}x_1 + \ldots + a_{jr}x_r + a_{j,r+1}x_{r+1} + a_{j,2r}x_{2r}$ avec $a_{ji} = 1$ si x_i "est dans la clause" et $a_{ji} = 0$. On remarque aussi que $a_{ji} + a_{j,i+r} \in \{0,1\}$ car x_i et $\overline{x_i}(= x_{i+r})$ ne peuvent être simultanément dans la clause.

Montrons comment Bob peut retrouver les coefficients a_{ji} et donc la clause j à partir des valeurs $u_0, u_1, \ldots, u_r$ définies par :

- $u_0 = v_j(0, \ldots, 0)$

- $u_i = v_j(x_1, \ldots, x_r)$ avec $x_i = 1$ et $x_{i' \neq i} = 0$

Il suffit de remarquer que les coefficients $a_{ji}, a_{j,i+r}$ peuvent être retrouvés connaissant $u_i - u_0$ par :

- $a_{ji} = a_{j,i+r} = 0$ si $u_i - u_0 = 0$

- $a_{ji} = 1$ (et donc $a_{j,i+r} = 0$) si $u_i - u_0 = 1$

- $a_{ji} = 0$ (et donc $a_{j,i+r} = 1$) si $u_i - u_0 = -1$

2 - On propose l'évolution suivante. Bien que cette évolution dévoile le nombre de clauses de f, elle peut s'avérer intéressante en pratique.

EvalDNF

---

1. *Bob génère* $(pk, sk) \leftarrow Paillier.KeyGen(\lambda)$*, publie* pk *et envoie* $X_1 \leftarrow Paillier.Encrypt(pk, x_1), \ldots, X_r \leftarrow Paillier.Encrypt(pk, x_r)$ *à Alice.*

2. ***Alice génère aléatoirement*** $\alpha_1, \ldots, \alpha_s \in \mathbb{Z}_n^*$ ***et une permutation*** σ ***de*** $\{1, \ldots, s\}$**.** *Alice génère* $X_{i+r} = Paillier.Encrypt(pk, 1) \oplus (-1 \circ X_i)$ *pour tout* $i = 1, \ldots, r$*. Ensuite, elle calcule* $C_j = \alpha_j \circ (X_{i_{j1}} \oplus \ldots \oplus X_{i_{js}} \oplus Paillier.Encrypt(pk, -t))$ *pour tout* $j = 1, \ldots, s$*.* ***Elle envoie à Bob*** $(D_1, \ldots, D_s) = (C_{\sigma(1)}, \ldots, C_{\sigma(s)})$**.**

3. *Pour tout* $j = 1, \ldots, s$*, Bob decrypte* $d_j \leftarrow Paillier.Decrypt(sk, D_j)$*. S'il existe* $j \in \{1, \ldots, s\}$ *tel que* $d_j = 0$ *alors Bob retourne* true *sinon il retourne* false*.*

---

3 - Le protocole précédent dévoile le nombre de clauses de f*. Utiliser un FHE permettrait de générer (à l'étape 2) une encryption* D *qui encryptrait le produit des valeurs encryptées par* $D_1, \ldots, D_s$*, i.e.* $D = D_1 \otimes \cdots \otimes D_s$*. Ainsi Bob recevrait une valeur* $d \leftarrow Paillier.Decrypt(sk, D)$ *qui serait nulle si* $f(x_1, \ldots, x_r) = 1$ *et aléatoire sinon. Autrement dit, le protocole ne dévoilerait rien sur* f *mis à part* $f(x_1, \ldots, x_r)$ *comme dans le cas idéal.*