SLH 2024-2025

Exercices 10 - Commitments

1. Casino En ligne

Dans cet exercice, nous allons implémenter un système de roulette en ligne dans lequel ni les joueurs, ni la banque ne peuvent tricher.

Le joueur choisit un nombre x entre 1 et `MAX`; la banque tire un nombre b au hasard. Le joueur gagne si ce nombre correspond à celui choisi par le joueur ($b = x$).

2. Mauvaises solutions

1. Dans un premier protocole, la banque tire le nombre au hasard et l'annonce au joueur, puis le joueur annonce le nombre qu'il avait choisi. Dans ce protocole, qui peut tricher et pourquoi ?
2. Dans un second protocole, le joueur envoie son choix à la banque, puis la banque annonce le nombre tiré au hasard. Dans ce protocole, qui peut tricher et pourquoi ?

3. Solution avec engagement (Commitment)

Dans les protocoles précédents, la partie (banque ou joueur) qui communique sa valeur en 2eme peut tricher en changeant d'avis après avoir reçu l'information. Nous allons empêcher ceci en utilisant un protocole de commitment ad-hoc:

1. Le joueur commence par choisir sa valeur x, puis choisit une valeur aléatoire r, calcule $c = \mathrm{HMAC}(x, r)$ (x tient lieu de clé, et r de message), et envoie c à la banque
2. La banque choisit son nombre b, le signe, et l'envoie au joueur
3. le joueur vérifie la signature, qui lui prouve qu'il a gagné ou perdu.
4. Le joueur envoie x et r à la banque
5. La banque recalcule $c' = \mathrm{HMAC}(x, r)$ et vérifie que $c = c'$, ce qui lui prouve que le joueur a gagné ou perdu.

Questions:

1. Imaginons que le joueur décide de tricher, et de mentir sur son choix initial: au lieu de son x initial, il prétend avoir choisi $x = b$. Que va-t-il alors se passer ?
2. Pouvez-vous montrer que dans ce cas, tricher est au moins aussi difficile que de trouver des collisions dans la fonction de hachage (c'est-à-dire que celui qui est capable de tricher à ce protocole est automatiquement capable de trouver des collisions également) ?

4. Implémentation

Complétez le template donné en exercice.

Comme fonction de HMAC, vous pouvez utiliser celle fournie par `dryoc` dans la structure `dryoc::genericHash::GenericHash`.

Comme source aléatoire, vous pouvez utiliser le trait `rand::Rng` et le générateur `rand::rngs::OsRng`.