Blockchains – Feuille de TME #1

14 septembre 2019

Le but de TME est d'implémenter un calcul de `Proof-of-Work` ou aussi appelé mineur. Plutôt que de hacher des blocs, comme dans le protocole Bitcoin, nous allons hacher des identités accompagnés de *nonces* jusqu'à en trouver une qui puisse satisfaire la preuve de travail.

Ce TME peut être réalisé dans le langage de programmation de votre choix.

1 Structure des identifiants

L'algorithme de hachage que nous allons utiliser pour ce TME est SHA256. La plupart des langages de progammation disposent d'une bibliothèque implantant cet algorithme (externe ou non).

Nous devons dans un premier temps établir la structure des données que nous voulons hacher. Un identifiant est composé d'un `nom` et d'un `prénom`. On concatène ces deux chaînes de caractères par le caractère ':'. Par exemple, "Jean-pierre Koff" devient `"koff:jean-pierre"`. Pour établir un format constant de cette valeur, plutôt que de considérer une chaîne de caractère de taille variable, nous allons considérer un préfixe de son hash. Ainsi, un identifiant ne sera pas une chaîne de caractère mais les 8 premiers octets du résultat de la fonction de hachage SHA256.

Par exemple :

$$\mathcal{H}_{\texttt{SHA256}}(\texttt{"koff:jean-pierre"}) = \texttt{9932c2a8764d8dc8203f9...}$$
$$\texttt{Prefix}_8(\mathcal{H}_{\texttt{SHA256}}(\texttt{"koff:jean-pierre"})) = \texttt{9932c2a8764d8dc8}$$

Rappels :
- 1 octet = 2 caractères hexadécimaux
- On peut utiliser la commande `xxd -b` pour passer de la représentation hexadécimale à la représentation binaire

Exercice 1. Implantez une fonction `hash_id` qui prend en entrée un nom et un prénom et qui retourne les 8 premiers **octets** du hash SHA256 de l'identifiant comme décrit plus haut.

Exercice 2. Implantez une function `hash_value` qui prend en entrée 8 octets et un entier (le nonce) **32 bits** et qui retourne le hash des deux valeurs concaténées.

2 Minage

Nous allons maintenant implémenter le mineur de hash. Dans Bitcoin, pour qu'un bloc soit valide, il est nécessaire que le résultat de son hash soit préfixé d'un certain nombre de zéros. Ce nombre, appelé difficulté, est dynamique en fonction du temps.

Exercice 1. Implantez une fonction `count_zero_prefix` qui à partir d'une chaîne d'octets compte le nombre de **bit** à 0 qui sont en préfixe du paramètre.

Exercice 2. Donnez une fonction `is_valid` qui prend en entrée un nom, prénom, un nonce et une difficulté n vérifiant si le hash résultat est bien préfixé d'un nombre de zéro $\geq n$.

Exercice 3. Donnez une fonction `mine` qui prend en entrée un nom, prénom, une difficulté et qui retourne un nonce pour lequel le hash calculé sera valide selon la difficulté donnée.

Par exemple : `mine("koff", "jean-pierre", 15) = 13015`

Exercice 4. Testez votre mineur en traçant la courbe de temps en fonction de la difficulté. Que constatez-vous ?

Bonus. Optimisez et parallélisez votre algorithme.

3 Authentification par Proof-of-Work

Écrivez un client-serveur simple utilisant ce mécanisme pour l'authentification :

- Lorsqu'un client se connecte à un serveur, le serveur envoie deux entiers (32bits) : une difficulté et une valeur aléatoire.
- Le client calcule la nonce satisfaisant la difficulté énoncée
- Le client envoie finalement sa nonce au serveur
- Si la nonce est invalide, le serveur coupe la connection, si elle est valide, le serveur envoie la chaîne " :)" puis coupe la connection.