

Blockchains – Feuille de TME #1

27 septembre 2021

Le but de TME est d'implémenter un calcul de **Proof-of-Work** (aussi appelé minage). 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 Blake2. La plupart des langages de programmation disposent d'une bibliothèque implantant cet algorithme (possiblement externe). Blake2 est configurable pour calculer un hash d'une taille arbitraire. Pour ce TME, nous calculerons des hash de taille 32 (= 32 bytes).

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, "Satoshi Nakamoto" devient "nakamoto:satoshi".

Le hash Blake2 de cet identifiant est :

```
hex_of_bytes( $\mathcal{H}_{\text{Blake2}}$ ("nakamoto:satoshi")) =  
fe33597320decef8d53107fbeb6c33c359ac20cc0a8c2fd3b077ca2069fcd4bf9
```

Rappel : 1 octet = 2 caractères hexadécimaux

Exercice 1. Implantez une fonction `hash_id` qui prend en entrée un nom et un prénom et qui retourne les **octets** du hash Blake2 de l'identifiant comme spécifié plus haut.

Exercice 2. Définissez une fonction `encode_entier` qui encode un entier 32 bits en bytes (en représentation *big-endian*¹). Puis, implantez une fonction `hash_value` qui prend en entrée le hash d'un identifiant et un entier qui retourne le **hash** des deux valeurs concaténées.

```
hash_value( $\mathcal{H}_{\text{Blake2}}$ ("nakamoto:satoshi"), 123) =  
b1817d8dde83dd38ede3386cfc87f1c012310f92b5765637e3939cc7156eca3e
```

Attention :

- Nous devons considérer la représentation **binaire** de l'entier ; la représentation textuelle n'est pas valide.
- La taille de l'entier encodée doit être constante : `encode_entier(123) = 0b00000000 00000000 00000000 01111011` (ou `0x0000007b` en représentation hexadécimale). Si votre encodeur tronque les premiers bits, il vous faudra ajouter des bits de *padding*.
- Vous pouvez définir vous-même l'encodeur ou vous reposer sur une bibliothèque logicielle. Le décodeur ne sera pas nécessaire pour cet exercice.

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.

1. <https://en.wikipedia.org/wiki/Endianness>

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 une identité, un nonce et une difficulté n vérifiant si le hash résultat est bien préfixé d'un nombre de zéro $\geq n$.

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

Par exemple : `mine($\mathcal{H}_{\text{Blake2}}$ ("nakamoto:satoshi"), 15) = 24654`

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

3 Encodage et décodage d'un état de blockchain

Dans une blockchain, lorsque nous recevons un bloc, nous devons être capable de vérifier la cohérence du nouvel état. Pour cela, le créateur du bloc introduit, dans son nouveau bloc, le hash résultant du nouvel état qu'il a calculé. Sur notre nœud, nous devons également faire ce calcul et vérifier l'égalité des résultats. Si les hashes sont équivalents alors l'état annoncé est correct. Si ils diffèrent, cela peut signifier que le producteur de bloc a menti sur son bloc. Dans cet exercice, nous allons encoder un état (simplifié) et le hasher pour être capable d'effectuer cette vérification.

Notre état simplifié sera représenté par une liste (non-bornée) de comptes auxquels on associe un montant.

Voici un exemple d'état sous la forme d'un CSV :

```
1dc653a1447946592fe2871eeb01d8fd6ae353bf04ab789199e38777da3fd0c7, 1003
ad415c298389574a24f009671697dd58a717ec04aaa79bd39a130b1ae7a4b2a9, 8532
b6a46ab620ab41132a7e062bee0bd7ef6af99d5c25b9021edcb949f2cd6c2bbc, 100
d91340a0a4fc7283117fb7871a95e983455275347662345ffaaa75d674def6ec, 943
ff9f179535d17c8f29d7eb8ad3432eb8b16ce684b48527b12a1a71f10d3e63ec, 755
```

Les comptes seront représentés par des identifiants (un hash Blake2) et les montants par un entier 32 bits encodé en *big-endian*, similairement aux exercices précédents. En revanche, pour simplifier la tâche de décodage, nous préfixerons cette liste de compte encodée par le nombre d'éléments présents dans cette liste (sous la forme d'un entier 32 bits).

Exercice 1. Donnez les fonctions `encode_compte` et `decode_compte` qui encode et décode une paire (identité \times montant). Vous testerez que : `decode_compte(encode_compte(<compte>)) = <compte>`

Exercice 2. Implantez les fonctions `encode_etat` et `decode_etat` qui prend une liste de compte et permet l'encodage et le décodage d'un état. Comme spécifié plus haut, nous préfixerons ces données encodées par le nombre d'éléments présents dans la liste. Vous testerez également que : `decode_etat(encode_etat(<etat>)) = <etat>`

Par exemple, l'encodage de l'état précédent sera :

```
000000051dc653a1447946592fe2871eeb01d8fd6ae353bf04ab789199e38777da3fd0c7000003eb
ad415c298389574a24f009671697dd58a717ec04aaa79bd39a130b1ae7a4b2a900002154b6a46ab6
20ab41132a7e062bee0bd7ef6af99d5c25b9021edcb949f2cd6c2bbc00000064d91340a0a4fc7283
117fb7871a95e983455275347662345ffaaa75d674def6ec000003aff9f179535d17c8f29d7eb8a
d3432eb8b16ce684b48527b12a1a71f10d3e63ec000002f3
```

Exercice 3. Récupérez le fichier d'états encodés accompagnés de leur hash sur la page de l'UE (tme1-data) et vérifiez leur cohérence. La première ligne est le hash Blake2, la deuxième, l'état encodé et les données sont espacées d'une ligne blanche. Vous donnerez les indices des éléments incorrects.