

Rapport Sur Mini Blockchain en Python

Nom : Boudaoud

Prénom : Marwa

Groupe : 01.

1. Introduction

Ce projet consiste à créer une mini-blockchain en Python afin de comprendre concrètement le fonctionnement d'un bloc, le calcul du hash, le minage (Proof-of-Work), ainsi que la validation de la chaîne.

L'objectif n'est pas de reproduire Bitcoin, mais d'apprendre les bases : hash immuable, lien entre les blocs, difficulté, nonce et sensibilité aux modifications.

2. Présentation de l'implémentation

2.1. Structure d'un bloc :

Chaque bloc est représenté par une classe Python contenant les attributs suivants :

- **index** : position du bloc dans la chaîne
- **timestamp** : horodatage de la création du bloc
- **data** : contenu du bloc (transaction ou information)
- **previous_hash** : empreinte du bloc précédent
- **nonce** : variable utilisée durant le processus de minage
- **hash** : empreinte SHA-256 résultant des champs précédents

La structure garantit que toute modification interne entraîne un changement complet du hash grâce aux propriétés cryptographiques de SHA-256.

2.2. Fonction de hachage :

La fonction `calculate_hash()` concatène les données essentielles du bloc :

- index
- timestamp
- contenu (data)
- previous_hash
- nonce

Le tout est encodé et passé dans `hashlib.sha256()`.

Ce mécanisme fournit une empreinte unique pour chaque bloc, assurant intégrité et traçabilité.

3. Conception de la Blockchain

La classe `Blockchain` maintient une liste ordonnée de blocs. Elle assure plusieurs responsabilités :

3.1. Création du bloc Genesis

Un premier bloc, appelé **Genesis Block**, est généré automatiquement lors de l'initialisation.

Il constitue l'ancre de la chaîne et possède un `previous_hash` fixé à "0".

3.2. Ajout de nouveaux blocs

La méthode `add_block(data)` crée un nouveau bloc à partir des données fournies.

Avant d'être ajouté à la chaîne, celui-ci doit obligatoirement être **miné**, garantissant ainsi son authenticité.

4. Proof-of-Work (PoW)

Le Proof-of-Work constitue le cœur du mécanisme de sécurité.

Le principe est simple : le hash d'un bloc n'est considéré valide que s'il commence par un certain nombre de zéros, déterminé par la **difficulté**.

Pour atteindre cette condition, l'algorithme :

1. initialise le nonce à 0
2. calcule le hash
3. vérifie si le hash respecte la difficulté
4. sinon, incrémente le nonce et recommence

Ce procédé peut nécessiter des milliers d'itérations selon la difficulté choisie. Il démontre le coût computationnel nécessaire pour produire un bloc valide.

5. Validation de la chaîne

La méthode `is_chain_valid()` vérifie la conformité de l'ensemble de la blockchain selon trois critères :

5.1. Vérification des liens entre blocs

Le champ `previous_hash` de chaque bloc doit correspondre exactement au hash du bloc précédent.

C'est ce qui assure la continuité et l'intégrité structurelle de la chaîne.

5.2. Vérification du hash recalculé

Pour chaque bloc, la fonction recalcule le hash à partir des données réelles. Si le hash recalculé diffère du hash stocké, la chaîne est immédiatement invalidée.

5.3. Vérification de la difficulté

Chaque hash doit respecter la règle imposée par la difficulté (ex. commencer par "00").

Un hash qui ne satisfait pas cette règle est considéré invalide.

6. Sensibilité au Tampering (altération volontaire)

Afin d'illustrer l'importance du Proof-of-Work, une modification manuelle a été effectuée sur les données d'un bloc intermédiaire.

Suite à cette altération :

- le hash du bloc modifié change
- son `previous_hash` ne correspond plus à celui attendu
- la chaîne devient **immédiatement invalide**
- les blocs suivants héritent également d'un hash incorrect

Cette démonstration met en évidence la forte résistance d'une blockchain aux modifications non légitimes : toute tentative de falsification exige un re-minage complet des blocs suivants.

7. Captures d'écran à intégrer

1. Sortie du minage (Proof-of-Work)

```
AttributeError: 'Blockchain' object has no attribute 'difficulty'
PS C:\Users\imene\OneDrive\Bureau\blockchain> python blockchain.py
>>
Initialisation de la blockchain (difficulty=3)...
Miné : index=0 nonce=6210 hash=0002b72aca528c4545dfd0ed6d319f98db83c731acd5f614fc552d37687191f9 (temps=0.026s, essais=6210)
Miné : index=1 nonce=431 hash=00086648852bd380a20110d36387932b53ec8a17c9a70cc07dcf5f1a0cc25577 (temps=0.002s, essais=431)
Miné : index=2 nonce=1987 hash=0002b6230d9bc62d28ae08907c95a7c3787346a003fe54fe0e70591891228965 (temps=0.009s, essais=1987)
Miné : index=3 nonce=2225 hash=00007bff19029604571c27b78b95826dc259e8e77cb5c59ec56645a68cc6d98d (temps=0.008s, essais=2225)
```

2. Affichage complet de la chaîne après minage :

```

--- Chaîne complète après minage ---
--- Bloc 0 ---
Horodatage : 1763645366.6982913
Données : Genesis Block
Nonce : 6210
Hash : 0002b72aca528c4545dfd0ed6d319f98db83c731acd5f614fc552d37687191f9
Hash précédent : 0

--- Bloc 1 ---
Horodatage : 1763645366.8035762
Données : Transaction : Alice -> Bob
Nonce : 431
Hash : 00086648852bd380a20110d36387932b53ec8a17c9a70cc07dcf5f1a0cc25577
Hash précédent : 0002b72aca528c4545dfd0ed6d319f98db83c731acd5f614fc552d37687191f9

--- Bloc 2 ---
Horodatage : 1763645366.8058488
Données : Transaction : Bob -> Charlie
Nonce : 1987
Hash : 0002b6230d9bc62d28ae08907c95a7c3787346a003efe54f0e70591891228965
Hash précédent : 00086648852bd380a20110d36387932b53ec8a17c9a70cc07dcf5f1a0cc25577

--- Bloc 3 ---
Horodatage : 1763645366.8149407
Données : Transaction : Charlie -> David
Nonce : 2225
Hash : 00007bff19029604571c27b78b95826dc259e8e77cb5c59ec56645a68cc6d98d
Hash précédent : 0002b6230d9bc62d28ae08907c95a7c3787346a003efe54f0e70591891228965

Vérification de la blockchain : True

--- Tampering : modification du bloc 2 (sans re-miner) ---
--- Bloc 0 ---
Horodatage : 1763645366.6982913
Données : Genesis Block
Nonce : 6210
Hash : 0002b72aca528c4545dfd0ed6d319f98db83c731acd5f614fc552d37687191f9
Hash précédent : 0

```

3. Résultat de validation (chaîne valide)

4. Résultat de validation après modification (chaîne invalide)

```

Output (Ctrl+Shift+U)
--- Bloc 3 ---
Horodatage : 1763645366.8149407
Données : Transaction : charlie -> David
Nonce : 2225
Hash : 00007bff19029604571c27b78b95826dc259e8e77cb5c59ec56645a68cc6d98d
Hash précédent : 0002b6230d9bc62d28ae08907c95a7c3787346a003efe54f0e70591891228965

Vérification de la blockchain : True

--- Tampering : modification du bloc 2 (sans re-miner) ---
--- Bloc 0 ---
Horodatage : 1763645366.6982913
Données : Genesis Block
Nonce : 6210
Hash : 0002b72aca528c4545dfd0ed6d319f98db83c731acd5f614fc552d37687191f9
Hash précédent : 0

--- Bloc 1 ---
Horodatage : 1763645366.8035762
Données : Transaction : Alice -> Bob
Nonce : 431
Hash : 00086648852bd380a20110d36387932b53ec8a17c9a70cc07dcf5f1a0cc25577
Hash précédent : 0002b72aca528c4545dfd0ed6d319f98db83c731acd5f614fc552d37687191f9

--- Bloc 2 ---
Horodatage : 1763645366.8058488
Données : Transaction : Bob -> Eve (MODIFIE)
Nonce : 1987
Hash : 810a42748400164c2a91b5da52eb59132dca91d3ed60198fc83f17819c7a2ce2
Hash précédent : 00086648852bd380a20110d36387932b53ec8a17c9a70cc07dcf5f1a0cc25577

--- Bloc 3 ---
Horodatage : 1763645366.8149407
Données : Transaction : Charlie -> David
Nonce : 2225
Hash : 00007bff19029604571c27b78b95826dc259e8e77cb5c59ec56645a68cc6d98d
Hash précédent : 0002b6230d9bc62d28ae08907c95a7c3787346a003efe54f0e70591891228965

Invalid: block 2 does not satisfy difficulty (prefix 000)
Vérification après modification (doit être False) : False

```

8. Conclusion

Ce projet a permis de comprendre de manière approfondie les mécanismes qui sous-tendent les blockchains modernes.

L'implémentation réalisée démontre clairement :

- la structure imbriquée des blocs
- l'importance du hash comme mécanisme d'intégrité
- le rôle central du Proof-of-Work
- l'immuabilité de la chaîne
- la vulnérabilité immédiate en cas d'altération d'un bloc