Analyse technique et critique de la blockchain

TD1 : mini-blockchain pour un système de chat décentralisé

Objectifs

Vous êtes une petite équipe d'ingénieurs chargée de concevoir un système de messagerie instantanée (chat) sans serveur central, reposant sur une blockchain distribuée entre les machines des utilisateurs.

Chaque message envoyé dans le chat est enregistré dans la blockchain, répliquée sur l'ensemble des nœuds. L'interface utilisateur peut être minimale (ligne de commande ou petit GUI rudimentaire) : l'objectif principal est la compréhension technique et la mise en perspective critique de la technologie blockchain appliquée à un cas d'usage simple.

Instructions

Lisez bien l'énoncé, il contient beaucoup d'informations et chaque mot est utile!

Aidez-vous de la documentation que je vous indiquerai. Lorsqu'il utilise une nouvelle librairie, un développeur passe beaucoup de temps dans la documentation pour comprendre son fonctionnement.

Utilisez Python à 100%, Python est fourni avec un grand nombre d'outils qui facilitent votre travail, utilisez -les!

Coder proprement avec des noms de variables et de fonctions/méthodes bien choisis. Ajoutez des commentaires utiles à votre code afin de pouvoir encore le comprendre dans quelques jours.

Respecter les coding style. Je vous conseille de suivre les consignes du PEP8[1] ou le Google Python Styleguide[2].

Simple is beautiful. N'essayez pas de coder compliqué, restez simple, cela sera plus efficace et moins sujet aux erreurs.

Réfléchissez avant de coder, prenez un peu de temps pour dessiner/écrire vos idées sur une feuille. Le temps de la réflexion que vous prendrez, vous économisera énormément de temps de debug.

Le TD sera réalisé en groupe de 4 ou 5 élèves

1 Spécifications fonctionnelles

Vous devez concevoir une application répartie qui permet les fonctionnalités suivantes.

1.1 Nœuds, utilisateurs et clés

- Chaque instance de l'application représente un nœud de la blockchain.
- Chaque utilisateur est identifié par une paire de clés cryptographiques :
  - une clé privée utilisée pour signer les messages,
  - une clé publique qui permet aux autres nœuds de vérifier les signatures.
- La clé publique de l'utilisateur doit être rendue visible sur la blockchain via une transaction d'inscription d'utilisateur.

1. PEP8 : `https://www.python.org/dev/peps/pep-0008/`
2. Google Python Styleguide : `https://google.github.io/styleguide/pyguide.html`

1.2 Types de transactions, signatures et mempool

Vous devez gérer au minimum deux types de transactions :

1. Transaction d'inscription d'utilisateur (*UserRegister*) :
   — contient un identifiant d'utilisateur (pseudo) et une clé publique,
   — permet d'ajouter un nouvel utilisateur au système,
   — doit être signée (par exemple par l'utilisateur lui-même ou par une clé spéciale, à définir).
2. Transaction de message de chat (*ChatMessage*) :
   — contient un identifiant (ou hash),
   — la référence à l'utilisateur (pseudo ou identifiant lié à sa clé publique),
   — le contenu du message,
   — un timestamp,
   — une **signature numérique** calculée sur la transaction (ou au minimum sur le message + timestamp) avec la clé privée de l'utilisateur.

Mempool et diffusion

— Chaque nœud maintient une mempool contenant les transactions valides reçues et non encore intégrées dans un bloc.
— Lorsqu'une nouvelle transaction (UserRegister ou ChatMessage) est créée localement :
   1. elle est signée si nécessaire (ChatMessage, UserRegister si vous le décidez),
   2. elle est vérifiée localement (signature, format, unicité basique),
   3. elle est ajoutée à la mempool locale,
   4. elle est diffusée aux nœuds voisins.
— Lorsqu'un nœud reçoit une transaction :
   — il vérifie qu'elle n'est pas déjà connue (mempool ou blockchain),
   — il vérifie la **signature** à l'aide de la clé publique appropriée,
   — il vérifie sa cohérence (par exemple, un ChatMessage ne peut être accepté que si l'utilisateur existe déjà dans la blockchain),
   — si elle est valide, il l'ajoute à sa mempool,
   — il la rediffuse vers ses voisins si nécessaire.

1.3 Blocs et arbres de Merkle

— Les transactions de la mempool sont regroupées en blocs.
— Chaque bloc contient au minimum :
   — un index (hauteur),
   — le hash du bloc précédent,
   — un timestamp,
   — une liste de transactions,
   — la racine de Merkle calculée sur les transactions du bloc,
   — un hash du bloc calculé à partir de son contenu (incluant la racine de Merkle).
— Vous devez implémenter un arbre de Merkle simple :
   — calculer un hash pour chaque transaction (feuille),
   — combiner les hash deux à deux jusqu'à obtenir la racine,
   — gérer le cas d'un nombre impair de feuilles (par duplication ou autre stratégie simple).
— La création de bloc peut suivre une règle simple, par exemple :
   — création d'un bloc toutes les X secondes si la mempool n'est pas vide, ou
   — dès qu'un certain nombre de transactions est atteint
   — choisissez une preuve à utiliser
— Un mécanisme de sélection des transactions depuis la mempool doit être défini (par exemple : prendre toutes les transactions disponibles, avec éventuel ordre de priorité).

1.4 Propagation des blocs et synchronisation

- Lorsqu'un nœud crée un nouveau bloc :
  - il vérifie la cohérence du bloc (hash, racine de Merkle),
  - il l'ajoute à sa blockchain,
  - il retire de la mempool les transactions contenues dans ce bloc,
  - il diffuse ce bloc à ses voisins.
- Lorsqu'un nœud reçoit un bloc :
  - il vérifie :
    - l'intégrité du hash du bloc,
    - la cohérence avec le bloc précédent (index et hash),
    - la racine de Merkle (recalculée à partir des transactions),
    - la validité des transactions (signatures, utilisateurs existants, non-duplication).
  - s'il est valide, il l'ajoute à sa blockchain,
  - il supprime les transactions correspondantes de sa mempool,
  - il le rediffuse éventuellement à ses voisins.
- En cas de conflit (forks), vous adopterez une règle simple, par exemple :
  - choisir la chaîne la plus longue (plus grand nombre de blocs) ou la plus ancienne, etc.
  - et réaligner la mempool en conséquence (réinjection des transactions orphelines si nécessaire).

1.5 Interface de chat

- L'interface peut être en mode texte (terminal).
- Le chat affiche les messages valides présents dans la blockchain dans l'ordre :

```
[timestamp][pseudo] message
```

- L'arrivée de nouveaux blocs doit déclencher une mise à jour de la vue locale du chat.
- Seuls les messages correctement signés par un utilisateur préalablement inscrit sont affichés.
- L'inscription d'un nouvel utilisateur est aussi un message à afficher

2 Contraintes techniques minimales

- Langage : Python
- Hash : utilisation d'une fonction de hachage cryptographique standard (SHA-256, SHA-3, ...).
- Signatures :
  - utilisation d'une bibliothèque existante (par exemple RSA),
- Communication réseau :
  - sockets, WebSockets, HTTP minimaliste, ou tout mécanisme permettant la communication P2P
- Interface : une simple boucle input/print en console est suffisante mais une GUI est possible

3 Livrables

1. À rendre dans un dépot github
2. Code source de l'application (commenté, structuré).
3. Rapport incluant au minimum (dans le readme du github par exemple) :
   - nom des membres du groupe
   - architecture retenue,
   - détaillez et expliquez vos choix techniques (protocole réseau, structure des blocs, arbre de Merkle, gestion des signatures et des clés, gestion de la mempool),

— retour critique (positif, négatif) sur l'emploi de la blockchain pour faire un mini-chat,
— retour critique de la blockchain