Mini-blockchain avec communication TCP et sécurisée par signature RSA

1 Sujet

Sur une petite île, des habitants veulent se passer des notaires ; pour cela, ils décident de mettre en place un système de registre sécurisé sur lequel ils écriront quelle maison est à qui et toutes les transactions immobilières qu'ils réaliseront.

Le problème d'un tel registre est que la personne qui le détient peut en faire ce qu'elle veut : dupliquer des transactions, les supprimer, etc...

Pour pallier ce problème, ils décident de mettre en place une **blockchain**.

En effet, une blockchain a la particularité d'être décentralisée (i.e. tout le monde en possède une copie, il n'y a pas d'autorité suprême comme les banques ou les notaires), elle est de plus sécurisée par divers moyens que vous allez découvrir dans ce projet.

Une blockchain (littéralement une "chaîne de blocs") est formée de blocs qui sont connectés les uns aux autres via le haché du bloc précédent. Par ailleurs, chaque haché doit se terminer par une série de n zéros, il s'agit de la "difficulté" et celle-ci change en fonction du nombre d'utilisateurs de la blockchain.

Sur notre île imaginaire, les blocs contiendront les informations suivantes :
- Numéro du bloc
- Difficulté (en bits)
- Haché du bloc précédent
- Taille des modules RSA (en bits)
- Informations d'une maison
- Nonce

La partie "Informations d'une maison" aura la structure suivante :
- Adresse (de la maison),
- Nom et prénom du propriétaire,
- Adresse publique (cryptographique) du propriétaire.
- Signature du propriétaire pour le message suivant :

(numéro du bloc || adresse || nom et prénom du propriétaire).

Dans la vraie vie, chaque bloc contiendrait bien plus d'informations et surtout des transactions, explicitant d'éventuels changements de propriétaires.

Pour des raisons de simplicité, dans ce projet on se limite à une blockchain contenant le descriptif de qui possède telle ou telle maison ; chaque bloc contenant les informations d'une seule et même maison.

2 Projet

Vous allez tout d'abord créer un programme serveur que vous appellerez `full_node.py` ; il sera capable de générer des blocs sous la forme d'un fichier `blockchain` qui sera ajouté au dossier courant [1].

1. Ce fichier n'est pas nécessairement un fichier texte... à vous de voir.

La saisie des éléments d'un bloc sera faite par l'utilisateur via des questions qui lui seront posées dans le terminal.

Les clés RSA relatives à un propriétaire seront quant à elles générées aléatoirement par votre programme, **elles changeront à chaque nouveau bloc.**

À chaque exécution du programme `full_node.py`, l'utilisateur aura le choix entre deux options :

(1) miner un nouveau bloc,

(2) partager la blockchain avec un simple node.

Si la seconde option est choisie, le full node attend alors la connexion d'un simple node, que vous programmerez dans un fichier `simple_node.py`.

S'il y a une demande de connexion, le full node l'accepte et il reçoit alors le nombre de blocs que le simple node possède dans son répertoire local.

Si ce nombre est inférieur au nombre de blocs que possède le full node [2], le full node lui envoie les blocs manquants.

Il est important de vérifier que le simple node a bien tout reçu, vous mettrez donc en place un protocole que vous détaillerez dans votre rapport pour vous en assurer.

Une fois que le simple node a bien tout reçu, il s'arrête ; le programme `full_node.py`, quant à lui, propose de nouveau le choix entre les deux options (1) et (2) à l'utilisateur, et ainsi de suite.

Enfin, pour consulter la blockchain et vérifier sa **validité**, le simple node peut être exécuté avec le paramètre `-consult` en ligne de commande ; il décrit alors précisemment le contenu du fichier `blockchain` présent dans son répertoire courant (avec une indication de validité pour les items qui le nécessitent).

Encore une fois, ce modèle est incroyablement simplifié par rapport à la vraie vie dans laquelle : il y a bien des full nodes et des simple nodes, néanmoins les full nodes sont en compétitions entre eux pour arriver à miner le prochain bloc, et le plus rapide l'emporte, le nouveau bloc est alors ajouté à la blockchain et tous les simples nodes sont prévenus. De plus, ce n'est

2. Ce qui sera toujours le cas pour ce projet

pas le full node qui choisit ce qu'il veut inscrire dans le bloc (en termes de transactions ou d'informations notariales), il les pioche dans une listes de transactions "à traiter" disponibles pour tous.

3 Informations techniques

— Par convention, le haché du bloc 0 est la chaîne dont tous les octets sont nuls.
— La difficulté de minage sera fixée en dur dans votre programme pour correspondre à l'ajout d'un bloc environ toutes les minutes. Pensez à bien expliquer dans votre rapport comment vous l'avez calculée, pour cela vous pouvez programmer un petit programme `hash_rate.py`.
— Pour la génération des nombres premiers, je vous laisse faire comme vous le sentez, mais essayez de justifier un peu comment marche votre méthode : si elle vous semble sûre ou pas ? efficace ou pas [3] ?
— Pour la signature RSA, vous utiliserez un algorithme de type square and multiply et l'exposant public e sera 65537.
— Lors de l'envoi d'informations dans le canal TCP, il peut arriver que des données se perdent, expliquez comment votre protocole prend en charge cela pour être sûr de tout bien envoyer / recevoir.
— Pour la fonction de hachage, vous utiliserez la fonction SHA-256 [4].
— Votre blockchain doit pouvoir contenir tous les types de caractères affichables, cela inclut donc les caractères avec accents et les smileys.
— Si vous avez bien suivi le sujet, vous remarquez que la clé privée du propriétaire est "perdue" lorsque le bloc est généré ; cela n'est pas grave, car on peut supposer que le propriétaire qui a exécuté le full node (dans la vraie vie) en possède évidemment une copie.
— Si le programme full node est exécuté alors qu'il y a déjà un fichier blockchain dans le repertoire courant, les nouveaux blocs seront ajoutés à la suite.
— **Bonus :** essayer de rendre votre fichier `blockchain` le plus petit possible, si vous y arrivez, donner des tailles dans votre rapport (avant et après), et expliquez comment vous avez procédé.
— Pas de panique, après avoir eu le sujet, vous aurez droit à une démonstration de ma solution en salle de TP, ce qui vous permettra de tout bien comprendre, et surtout : vous pourrez poser toutes vos questions [5] !

3. Ne vous en faites pas, j'attends ici quelque chose de très simple, la génération de nombres premiers **cryptographiquement sûrs** sera au programme du Master 2 !
4. Comme c'est actuellement le cas dans le réseau Bitcoin.
5. Sauf celles de la section suivante ! ☺

4 Questions de cours [5 points]

Pour la longueur des réponses aux questions suivantes, soyez raisonnable : une phrase par question c'est trop peu, deux pages par question c'est trop !

- Dans notre modèle, le propriétaire de la maison signe les informations importantes du bloc, quelle faille de sécurité évidente voyez-vous avec ce modèle ? Expliquez brièvement comment cela se passerait dans la vraie vie avec des "transactions".
- Comment s'appelle le fait que les ordinateurs des utilisateurs "tournent en boucle" pour trouver le nonce ? Existe-t-il d'autres moyens de prouver que l'on "travaille" dans un contexte de blockchain ?
- Si un peu plus de la moitié des habitants de l'île se réunissent de manière malicieuse, comment pourraient-ils nuire à la blockchain ? Comment s'appelle cette attaque ? Décrivez aussi précisément que possible comment marche l'attaque.
- Pour la signature, je n'ai pas précisé ce qu'il fallait signer... décrivez **précisement** ce que vous signez et expliquez pourquoi.

5 Instructions pour le Projet

- Ce projet est à réaliser **en binôme**.
- Toutes similitudes suspectes entre les rapports ou codes de groupes différents ou de groupes des années précédentes seront **fortement sanctionnées**.
- Le langage à utiliser est Python 3 (comme en TP). Les commandes vues en cours, TD, TP et celles indiqués dans le sujet sont largement suffisantes pour ce projet. Tout appel à des modules ou fonctions autres **est interdit** et également **fortement sanctionné**.
- Votre code doit s'exécuter sans erreur, c'est la base de tout travail en informatique. Un code qui ne s'exécute pas est noté sur la moitié des points.
- Le rendu de votre projet contiendra **uniquement** : vos fichiers de codes et un rapport au format pdf. Le tout sera à mettre dans une archive (par exemple .zip).
- Trouvez le juste milieu pour le rapport : pas besoin de faire 20 pages, mais 2 pages ne sont pas suffisantes ! Décrivez bien comment marchent vos protocoles **(avec des schémas !)**, vos principaux algorithmes, comment vous avez surmonté ou pas les difficultés, etc.
- N'oubliez pas d'ajouter une "notice d'utilisation de votre code", par exemple avec des captures d'écrans et quelques indications.
- Le projet est à envoyer par email à

  maxime.bros@unilim.fr

  au plus tard le **Dimanche 16 Janvier 2022 à 23h55 au plus tard**, tout retard sera sanctionné [6].

Bonne chance à tous !

6. Malus de 2 points par tranche de 12h.

6 Annexe : code pour vous aider

```python
# ##### hachage solution 1
import subprocess
bytes_string=bytes([0,1,2,3])
to_be_hashed = open("to_be_hashed","wb")
to_be_hashed.write(bytes_string)
to_be_hashed.close()
texte_commande="openssl dgst -sha256 to_be_hashed"
commande = subprocess.Popen(texte_commande, shell=True,stdout=subprocess.PIPE)
(resultat, ignorer) = commande.communicate()
print(resultat)
# la commande ci-dessous permet de supprimer le fichier temporaire créé
suppression = subprocess.run("rm to_be_hashed",shell=True,stdout=subprocess.PIPE)

# ##### hachage solution 2
import hashlib
h = hashlib.sha256()
bytes_string=bytes([0,1,2,3])
h.update(bytes_string)
hashvalue = h.hexdigest()
print(hashvalue)

# ##### test de primalité
import subprocess
commande = "openssl prime "
n = 13
r = subprocess.run(commande+str(n),shell=True,stdout=subprocess.PIPE)
resultat_openssl = r.stdout
print(resultat_openssl)
```

Pour calculer des temps, vous pouvez utiliser le module `time` et la commande suivante :

```
time.time()
```