Simple Blockchain

Introduction

We'll extend the simple blockchain building example used in some of the code notebooks. In this project, we'll cover key generation, digital signing, and transaction and block validations

Housekeeping points

- This is a minimal example and may not follow some standard practices
- The focus is on the main flow, with minimal error handling. Errors in validation logic should be handled appropriately though.
- This has basic blockchain semantics. It doesn't follow bitcoin transaction's multiple sender/receiver complexity. It stores a running balance like Ethereum, instead of a UTXO based structure like bitcoin.
- **Please read the comments in the code, they explain the structure and methods**

Program Organization

All the code is in the **M04-P01-Source-Code.zip** file

- **Block.py:** This class implements a basic Block including some mandatory attributes like previous block hash, index, creation timestamp, transactions, hash target, nonce, and the final block hash.

  The hash_block method generates a hash for the block including all relevant components. The primary methods are already implemented.

- **Account.py:** This class implements a blockchain account including some mandatory attributes like id, balance, nonce, and private/public keys encoded in pem format strings.

  The id is just a text string here, instead of a large hexadecimal address. Uniqueness is assumed but not enforced. It keeps a running balance of value, unlike bitcoin. A few supporting methods are already implemented.

- **Blockchain.py:** This class implements a basic Blockchain including some mandatory attributes like the actual chain of blocks, pending transactions, known accounts, and the current target hash.

Genesis block is automatically created during initialization. A few supporting methods are already implemented.

- **main.py:** This is the main driver program. It creates some accounts, a blockchain instance and blocks. It also triggers a full blockchain validation and prints the blockchain and the current account balances.

Problem Statement

The program structure is already set and there are specific methods that you are expected to implement. Please also read the comments in the code, especially in the methods to be implemented. You can modify main.py to add further demonstration calls if you want or add another caller file to do further tests if preferred.

1. (Medium) Implement the cryptographic functionalities for the blockchain account
    a. Implement private/public key pair generation in *__generate_key_pair* private method in Account class and assign them to *_private_pem* and *_public_pem* attributes. Please refer to code notebooks for hints and use the rsa based protocol from the cryptography lib
    b. Complete the *create_transaction* method in Account class. You'll need to:
        i. Generate the private key object from the pem created in 1.a
        ii. Generate the hash of the transaction message
        iii. Digitally sign the hash with the private key
        iv. Handle formatting appropriately for hashing and signing inputs
        v. You might need to convert the signature from 'bytes' to an appropriate string based format. Base64 is an appropriate choice, base64 library is already imported.
    c. Complete the *__validate_transaction* method in Blockchain class. You'll need to:
        i. Generate the public key object using the public pem from the account
        ii. Hash the transaction message with a similar process as while signing
        iii. Digitally verify the generated hash with the signature
        iv. Handle formatting appropriately for hashing and verification inputs
2. (Medium) Implement validations at various levels
    a. Implement *__process_transactions* method in Blockchain class. For each transaction, you'll need to check that there is enough balance in the sender account and then appropriately transfer correct value from the sender to the receiver
    b. Implement *validate_blockchain* method and the methods called within. This validates the whole blockchain, for three points:
        i. Implement *__validate_chain_hash_integrity* to ensure that the stored previous block hash actually matches the previous block's hashed value, for all blocks. Exclude genesis block for obvious reasons.

ii. Implement *__validate_block_hash_target* to ensure that the generated block hash is actually lesser than the target hash, and is actually the hash of the block with the stored nonce value. Exclude genesis block for obvious reasons.

iii. Implement *__validate_complete_account_balances* to ensure that at no point in the history, any account exceeded its balance in any transaction. You'll need to:

1. Enhance the Account class to store the initial balance
2. Replay through all transactions per block from the start and ensure compliance.

Evaluation Rubric

Total Project Points: **240**

- Basic compilation without errors (**10%**) : **24 Points**
- Correctness:
  - Problem statement - 1.a (**15%**) : **36 Points**
  - Problem statement - 1.b (**15%**) : **36 Points**
  - Problem statement - 1.c (**15%**) : **36 Points**
  - Problem statement - 2.a (**10%**) : **24 Points**
  - Problem statement - 2.b.i (**10%**) : **24 Points**
  - Problem statement - 2.b.ii (**10%**) : **24 Points**
  - Problem statement - 2.b.iii (**15%**) : **36 Points**

Program Instructions

1. Please install cryptography library, '**pip install cryptography**'

2. Download the zipped folder named **M04-P01-Source-Code.zip**, and unzip it on your local machine. Go into the directory named **M04-P01-Source-Code**

3. Make sure that you have Python 3.6, or higher, installed. At your command prompt, run:

   ```
   $ python --version
   Python 3.7.3
   ```

   If not installed, install the latest available version of Python 3.

4. To run the code in the source code folder, run the following command:

   ```
   $ python3 main.py (On many Linux/Mac platforms)
             OR
   $ python main.py       (On Windows/Mac platforms)
   ```

In any case, one of these two commands should work.

5. Alternatively, you could install a popular Python IDE, such as PyCharm or Visual Studio Code, and select a command to build the project from there.

References

1. https://cryptography.io/en/latest/installation/
2. https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/#generation
3. https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/#key-serialization
4. https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/#key-loading
5. https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/#signing
6. https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/#verification
7. https://cryptography.io/en/latest/exceptions/#cryptography.exceptions.InvalidSignature