Introduction to Cryptocurrencies - Ex1

Due date: Tuesday, April 21

This exercise can be solved in pairs. Larger groups are not allowed. Be sure to write both names in a comment at the top of each file. Only a single partner should submit the exercise and you must specify the other partner when submitting (in the moodle upload link)

1 High Level Goal

We wish to build a very naive simple prototype of a secure payment system. We will start with one that relies on a single trusted authority - the Bank.

In a later exercise, we will replace the Bank with a distributed system built according to Nakamoto's consensus. Individual users will be represented by "Wallets" that control private keys and create transactions.

2 Overview

There are two types of entities in our system: the Bank and wallets. Wallets represent end users that want to send or receive money ("execute transactions"), and the Bank represents a database that holds records of past transactions and confirms new ones.

The system will work in discrete time steps. We call each step a "day". At the end of each day, the bank publishes a commitment to a new block. The block represents the latest part of the current state of the system, with all blocks forming a blockchain. The commitment is to the following data:

1. All the approved transactions of that day.
2. The previous commitment, which represents the previous state.

Wallets can perform transactions as they wish, and can ask the bank for blocks that were published since they were last online. For simplicity, all transactions transfer the same amount: a single coin, from a single address to a single address. All addresses are simply public keys. It is only possible to pay with a coin that was transferred in an approved transaction.

To create money, the bank can create a transaction that has no source address. It does not need to be signed, but, in order to differentiate between money creation transactions, random bytes are placed in their signature fields; see the template file for more details. To generate random bytes use the secrets module. The function "secrets.token_bytes(n)" can create n random bytes in a cryptographically secure manner. This package is part of python's standard distribution and does not need to be installed.

For this exercise, assume the bank does not change past blocks in the blockchain: every new block is appended to the end of the current chain.

We provide you with the APIs of the wallet and the bank. Your task is to fully implement these APIs in Python 3.6 or above (make sure you are compatible with these versions). Moreover, we provide **some** useful pytests that you can use to check your work. These tests are incomplete, and additional ones that we do not share will be used to evaluate your code. The tests are written as attacks on the security of your system, so make sure to cover all the cases that might be used by an attacker. The tests assume that your implementation follows the *exact* API. Don't break it.

Digital Signatures You should use the *ecdsa*[1] package to handle signing and verifying messages. This is a pure-python package without any dependencies, therefore should be easily installed via *pip*. Your code must use it and have no other dependencies.

Here is a simple code example that uses this package:

```
import ecdsa
private_key = ecdsa.SigningKey.generate()
h = private_key.sign(text)
pub_key = private_key.get_verifying_key().to_der()
try:
    ecdsa.VerifyingKey.from_der(pub_key).verify(h, text)
    return True
except:
     # Verification failed
     return False
```

In addition to signatures, you will need to generate hashes of blocks and transactions (transaction ids will be a hash of their contents). To do so, use "hashlib.sha256()".

Modularity and clean code Make sure your code is modular and well written (but do not over-generalize). This will save you time in the next exercise (some refactoring of the code of this exercise will be needed).

3 APIs

The Python skeleton files we provide contain methods that define the APIs of Block, Transaction, Wallet, and Bank. They are annotated with types to help you make sense of the code, and to prevent bugs. We highly recommend

[1]`https://github.com/warner/python-ecdsa`

running mypy to typecheck your code as you write it. If you are not familiar with Python's type system, see here: `https://mypy.readthedocs.io/en/stable/getting_started.html`

3.1 Tests

We've included some pytests. You are encouraged to write your own as you code. A secret set of tests will be used on your code to help determine your final grade.

To make sure these tests pass, be sure to abide the API - do not change any of the member names provided in the provided template file.

4 Food for thought

Think about the following points (no need to write answers to these questions):

- Assuming that all wallets fully validate the blockchain, can the bank move money out of someone's account against their wishes?
- Can the bank revert a transaction that previously entered the blockchain and was published?
- Can wallets disagree about the contents of blocks?

5 Submission

Submit your files in a single zip file called ex1.zip. The zip should contain a README with a brief description of the project. The README should additionally contain the names, IDs and logins of both partners submitting the exercise. To run your solution we will import ex1.py. Make sure this will only define the relevant classes (as in the template file) and will not run additional code.

Good Luck!