# Introduction to Cryptocurrencies - Ex1

Due date: Wednesday, March 30

## 1 High Level Goal

We wish to build a simple prototype of a secure payment system. We will start with one that relies on a single trusted authority ("The Bank"). Individual users will be represented by "Wallets" that control private keys and create transactions. The bank will confirm transactions in blocks and will abide by the UTxO model.

In a later exercise, we will replace the Bank with a distributed system built according to Nakamoto's consensus.

## 2 Overview

There are two types of entities in our system: Bank and wallets. The wallets represent end-users that want to send or receive money, and the Bank represents a centralized 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 in the form of a hash of that 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.

The 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. A wallet can only transfer a coin that it has received in a previous approved transaction.

To create money, the bank (and only 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 the signature field of such money-creation transactions; 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 the 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.

# 3 APIs

The Python skeleton files we provide contain methods that define the APIs of Block, Transaction, Wallet, and Bank. **The behavior of these elements is defined in the documentation. Read it carefully!**

The code is also annotated with types to help you understand it. We highly recommend 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`

We provide the APIs of four classes: `Wallet`, `Bank`, `Transaction`, and `Block`. Your task is to fully implement these APIs in Python 3.6 or above (make sure you are compatible with these versions. The school currently runs Python 3.8 which is the recommended version).

**Digital Signatures** You should use the `pyca/cryptography` package (`https://cryptography.io/en/latest/`) to handle signing and verifying messages. This package can be easily installed via *pip*. To help you use this package easily, you'll find three functions that provide the basic utilities need for digital signatures (signing, verifying, and generating keys) in the file utils.py:

- `sign(message:bytes, private_key:PrivateKey) -> Signature`
- `verify(message:bytes, sig:Signature, pub_key:PublicKey) -> bool`
- `gen_keys() -> Tuple[PrivateKey, PublicKey]`

The types PrivateKey, Signature, and PublicKey above are subtypes of bytes that are defined in utils.py as well.

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 the function `hashlib.sha256().digest()`. See more details here: `https://docs.python.org/3/library/hashlib.html`

**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 (you'll be able to use the code from this exercise as a starting point if you wish). The code is currently split to several files and a "`__init__.py`" is included to join them together into a single package.

# 4 Tests

We provide *some* useful tests (in pytest) that you can use to check your work and to get a feel for how the objects work together. These tests are incomplete, and additional ones that we do not share will be used to evaluate your code. We encourage you to write your own as you code.

If you've installed pytest (using pip), then running "`python -m pytest`" in the root directory of the project should locate the tests in the tests subdirectory and run them together with the code. (Calling "`pytest`" directly may not correctly discover the package ex1, as it doesn't set the python path to the root directory.)

The tests are written either as consistency checks, or 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.

To be clear: under regular circumstances the Bank and each of the Wallets would be running as a separate process on a different computer and connecting over the internet. In our small prototype implementation, these are represented by objects on the same machine. We assume the attacker does not have access to internal fields or non-public methods of Banks, and Wallets, but that it may change the information they communicate between them. For example, a transaction that is just a sequence of bytes can be created and sent to the bank by any user including malicious ones, Signatures from one transaction can be copied onto another message, etc.

The attacker is thus able to create new transaction messages or manipulate blocks that the bank sends to the wallets. Note that the hash of a block or a transaction is computed from its contents and so cannot be manipulated independently of the contents of the message.

# 5 Food for thought

Think about the following points (no need to write answers to these questions, or to implement anything in code):

- 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 was previously entered into the blockchain and published?
- Can wallets disagree about the contents of the blocks?

# 6 Submission

Submit your files in a single zip file called ex1.zip. The zip file should be flat (without a directory structure) and should contain only the python files in the

subdirectory ex1 (do not include the tests directory). It should additionally contain a README file (not README.txt). The README file should include your name, ID and login as well as a brief description of the exercise. To run your solution we will import from ex1 (as the tests in the test directory do). Make sure this will only define the relevant classes (as in the template file) and will not run additional code.

## Good Luck!