CSCM29 - Blockchain, Cryptocurrencies and Smart Contracts

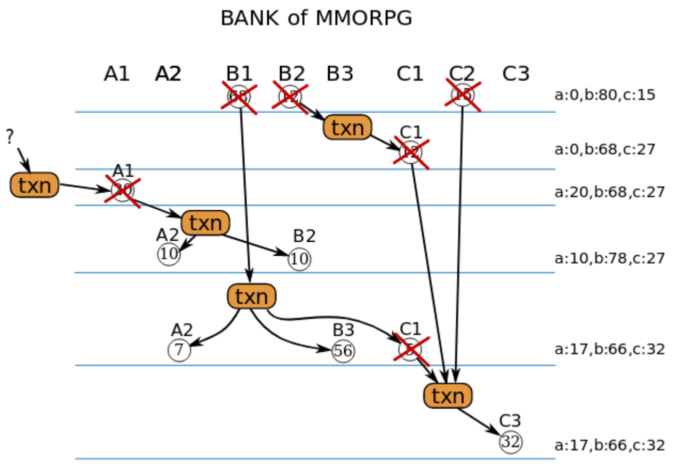
Arnold Beckmann, Anton Setzer

Lab 4

29 October 2019

Statement about Academic Integrity By submitting this coursework, electronically and/or hardcopy, you state that you fully understand and are complying with the university's policy on Academic Integrity and Academic Misconduct. The policy can be found at `https://myuni.swansea.ac.uk/academic-life/academic-misconduct`

Introduction In this Lab we are going to develop a transaction dag for a cryptocurrencies. Remember the transaction dag as discussed during the lecture[1]:



So we have transactions having a list of inputs and a list of outputs. Each output consists of an amount to be spent and an address of a recipient, where the address is essentially a hashed version of the public key of the recipient. The inputs are the unspent outputs of previous transactions (called utxo = unspent transaction outputs). They are given by the id of a transaction (which is the hash of the data forming a transaction), an output number, the public key which hashes to the address of that output, and a signature which signs the part of the transaction to be signed (consisting of the the input to be signed off and the list of transaction outputs) using the private key corresponding to the public key of the recipient.

[1]Source: Bitcoin Brownbag Talk by Warner, `http://www.lothar.com/presentations/bitcoin-brownbag/`

Location of the Code. The code is in the git repository available at `https://github.com/csetzer/CSCM29.git` in the subdirectory

`CSCM29/2019-20/Lab4/`

Overview over the code provided. Addresses, public and private keys are encoded as byte arrays. We maintain maps

- PublicMap.java which map user names as given by strings to public keys and public keys to addresses and names; this is supposed to be the names given by a name to other people known to him or her;
- PublicPrivateMap.java which contain an element of PublicMap.java and map in addition public keys to private keys. (This map should of course be kept private to a particular user, and maintain the private keys for that user);
- TxIdNameMap.java, which maps transaction ids to clear text names and clear text names to transaction ids; this map is used in order to reference to transactions in a more readable format.

We will provide methods toString and toStringBase58 in most classes:

- toStringBase58 will give only the information given by the java class in question, and represents addresses, public keys and private keys as strings of Base58, which is the standard of representing these items in cryptocurrencies. There will be sometimes versions which have as parameters strings indicating how to format the result, and versions which provide default output.
- toString methods will use the before mentioned maps in order to print out names instead of addresses or public keys; again there are versions with formatting information as parameters. Using the maps the method will look up as well additional information, for instance for input given by a transaction id and an output number it will determine the senders address and name, and the amount which was sent.
- print and printBase58 do the same, but print out the resulting string.

Most classes have as well a test() method which runs some test cases, and a main method which executes test(). The reason of separating the two is that one might write different test() methods and let the main method execute selected test cases.

Structure of the Code given. The code given has the following classes

- Output.java is the data structure for outputs of a transaction, given by the recipient's address and the amount which is given to the recipient. There are operations for getting the amount, the recipient's address and using a PublicKeyMap getting the recipient's name.
- OutputList.java represents a list of outputs. It has constructors forming the empty list, the one element list and the two element list, and methods for adding an output.

- TxOutput.java represents a transaction output, given by a transaction id and an output number.
- Input.java maintains the inputs of a transaction. They are given by a transaction output, a public key (which is needed since outputs contain only addresses which are hashed versions of public keys, and therefore addresses cannot be used directly for checking signatures), and a signature for the transaction. What is signed for an input is the input data for this input and the list of outputs. There is constructor having as arguments the name of a previous transaction, the output number, a list of outputs of the new transaction for which this input will be used, and the above mentioned maps. Using the maps it creates the message to be signed, looks up the public keys and private keys needed, and creates the signature needed.
- InputList.java organises a list or inputs given by Input.java. It has constructors for forming a list with 0, 1 and 2 inputs and methods for adding more inputs to a list of inputs. It has an operation for computing the sum of amounts.
- Transaction.java represents a transaction given by a name (used only for getting more readable print outs), a list of inputs and a list of outputs. It has operations for creating the data to be signed, and for checking whether the signatures are valid.

  A method checkTransaction (to be implemented by the students) in Transaction.java checks the correctness of the transaction by checking that the signatures are valid, the public keys in the inputs correspond to the addresses in the unspent transaction output, that all transaction inputs are unspent transaction outputs and the same transaction output doesn't occur twice, and that the sum of the inputs is greater than the sum of outputs. Coin base transactions have no inputs, and here we don't require the condition of the sum of outputs being greater than the sum of inputs.
- UTXOlist.java maintains a list of unspent transaction outputs, by having a map from addresses to a set of transaction outputs, a map from unspent transactions to the amount being spent, and to the address it was sent to.

  It has a method for adding transaction outputs to it and removing them. UTXOlist.java has as well a method processTransaction which processes a transaction by removing all inputs from the list utxo of unspent transaction outputs and adding all new outputs to it.
- PublicKeyMap, PublicPrivateKeyMap and TxIdNameMap organise the maps in question.
- The class TXledger.java will maintain the transaction based ledger given by a list of utxo and a TxIdNameMap. The main function in TXledger.java (mostly to be created by the students) runs test cases for the above.
- SampelKeyMap.java creates an example list of public private keys for each name provided.
- KeyUtils.java and Crypto.java maintain various operations related to public/private keys and addresses.

Tasks

- **Task 1:** Implement methods in Transaction.java for checking that a transaction is correct w.r.t. the TXledger. The overall check should run all 4 checks in sequence, and stops as soon as a mistake is found returning false.
- **Task 2** Define in TxLedger.java examples implementing a sequence of transactions. The precise scenario is described in the comments of the test() method of TxLedger.java.
- **Task 3** Create for each check defined in Task 1 one test case, which will fail that particular check and passes all checks coming before it.