**Cryptocurrencies (WS2022/23)** Dr. Zeta Avarikioti, Prof. Matteo Maffei

# Kerma - Task 2

## Object exchange

In this exercise, you will extend your `Kerma` node to implement content addressable object exchange and gossiping.

1. Maintain a local database of known objects. The database should survive reboots. You can use the package `level-ts` if you are coding in Typescript.

2. Implement a function to map objects to objectids. The objectid is obtained by taking a SHA-256 hash of the canonical JSON representation of the object. Canonical representation is important so that each object has a unique objectid. You can test your function using the Genesis block and its blockid given in [1].

   For Typescript, you may use the `fast-sha256` library to compute SHA-256 hashes. Note that the package takes inputs and returns outputs as Uint8 arrays while objectids are represented as hex strings in the protocol. Implement the required type conversions.

3. Implement object exchange using the `getobject, ihaveobject, object` messages.

   a) On receiving an `ihaveobject` message, request the sender for the object using a `getobject` message if the object is not already in your database.

   b) On receiving an object, ignore objects that are already in your database. Accept objects that are new and store them in your database if they are valid.

   c) Implement gossiping: Broadcast the knowledge of newly received valid objects to your peers using the `ihaveobject` message.

   d) On receiving a `getobject` message, send the requested object if you have it in your database.

## Transaction Validation

In this exercise, you will implement transaction validation for your `Kerma` node.

1. Create the logic to represent a transaction. See [1] for the structure of a transaction.

2. Create the logic for transaction validation. Transaction validation has the following steps:

   a) For each input, validate the outpoint. For this, ensure that a valid transaction with the given txid exists in your object database and that the given index is less than the number of outputs in the outpoint transaction.

b) For each input, verify the signature. Our protocol uses ed25519 signatures. A Typescript package for ed25519 is available. Note that signatures and public keys are given as hex strings in our protocol but the package uses Uint8 arrays, so you would have to convert between the two.

c) Outputs contain a public key and a value. The public keys must be in the correct format and the value must be a non-negative integer.

d) Transactions must respect the law of conservation, i.e. the sum of all input values is at least the sum of output values.

For now, assume that a coinbase transaction is always valid. We will validate these starting in the next homework.

*Tip*: To avoid errors due to processing invalid transactions, first verify that the received transaction is of the required format. For example, the transaction must contain the keys `"inputs"` and `"outputs"`, each input must contain the keys `"outpoint"` and `"sig"`, public keys and signatures must be hexadecimal strings of the required length, `index` and `value` must be non-negative integers in the correct range, etc.

3. When you receive a transaction object, validate it. If the transaction is valid, store it in your object database and gossip it using an `ihaveobject` message. If it is invalid, send an `error` message to the node(s) who sent the transaction and do not gossip it.

Don't worry about maintaining a perfect UTXO set just yet. We'll focus on this in later problem sets. For this homework, you may consider blocks and coinbase transactions to always be valid.

You should test your transaction validation by generating different valid and invalid transactions, signed using a private key of your choice. Here is a simple example that you can use for testing (the first is a coinbase, the second is a valid transaction that spends from the first):

```
{"object":{"height":0,"outputs":[{
"pubkey":"8dbcd2401c89c04d6e53c81c90aa0b551cc8fc47c0469217c8f5cfbae1e911f9",
"value":50000000000}],"type":"transaction"},"type":"object"}
```

```
{"object":{"inputs":[{"outpoint":{"index":0,
"txid":"1bb37b637d07100cd26fc063dfd4c39a7931cc88dae3417871219715a5e374af"},
"sig":"1d0d7d774042607c69a87ac5f1cdf92bf474c25fafcc089fe667602bfefb0494
726c519e92266957429ced875256e6915eb8cea2ea66366e739415efc47a6805"}],
"outputs":[{
"pubkey":"8dbcd2401c89c04d6e53c81c90aa0b551cc8fc47c0469217c8f5cfbae1e911f9",
"value":10}],"type":"transaction"},"type":"object"}
```

## Sample Test Cases

*Important: make sure your node is running all the time. Therefore, make sure that there are no bugs that crash your node. If our automatic grading script can not connect to your node,*

*you will not receive any credit. Taking enough time to test your node will help you ensure this. Below is a (non-exhaustive) list of test cases that your node will be required to pass. We will also use these test cases to grade your submission. Consider two nodes: Grader 1 and Grader 2.*

1. Object Exchange:

- If Grader 1 sends a new valid transaction object and then requests the same object, Grader 1 should receive the object.
- If Grader 1 sends a new valid transaction object and then Grader 2 requests the same object, Grader 2 should receive the object.
- If Grader 1 sends a new valid transaction object, Grader 2 must receive an `ihaveobject` message with the object id.
- If Grader 1 sends an `ihaveobject` message with the id of a new object, Grader 1 must receive a `getobject` message with the same object id.

2. Transaction Validation:

- On receiving an object message from Grader 1 containing any invalid transactions, Grader 1 must receive an error message and the transaction must not be gossiped to Grader 2. Beware: invalid transactions may come in many different forms!
- On receiving an object message from Grader 1 containing a valid transaction, the transaction must be gossiped to Grader 2.

**Due date: 14th November, 11.59pm**

# References

[1] Kerma - task 1 (192.065 cryptocurrencies (vu 4,0) 2022w). `https://tuwel.tuwien.ac.at/course/view.php?id=51148#section-4`, 11 October, 2022.