# HDS Coin - Stage 2
## SEC 2017-2018

Gabriel Maia, 81878
Instituto Superior Técnico, Lisboa
gabriel.maia@tecnico.ulisboa.pt

Artur Esteves, 91063
Instituto Superior Técnico, Lisboa
arturesteves@tecnico.ulisboa.pt

Ricardo Morais, 91064
Instituto Superior Técnico, Lisboa
ricardojhmorais@tecnico.ulisboa.pt

In the second stage of our HDS Coin implementation, we extended the code from the first stage in order to maintain the previous security and dependability guarantees, while adding Byzantine fault tolerance, through the use of multiple server replicas.

We followed the implementation steps presented in the project proposal. We modified the client and the server to allow for multiple server instances to be launched and for the client to communicate with them. Servers can be initialized and started through a bootstrapping script that takes the number of server failures to tolerate as the only argument. For simplicity, throughout the system the addresses of the servers are hard-coded, with the first server listening on port 4570, the second server on 4571, and so on.

## 1 Byzantine servers

To deal with Byzantine servers and tolerate the failure of at most (N - 1) / 3 server replicas on a system with a total of N server replicas, we implemented the (1, N) Byzantine regular register algorithm detailed in the course book. In our implementation, each ledger is mapped to a register, making the authenticated-data variant of the algorithm a good fit, as each ledger is naturally signed and timestamped through the chain of transactions and their signatures. Since we are using HTTP for communication between the clients and servers, plus the security guarantees from Stage 1, we can also consider that perfect authenticated point-to-point links are already in place.

The Byzantine regular register was transformed in a (1, N) Byzantine atomic register by applying the transformation presented in the course lectures, which consisted on adding a write-back phase to the read algorithm. When clients perform an account audit, the client library sends (writes) the result of the register read operation to each server, therefore "helping" correct servers complete any pending (delayed) writes before the client issues a new write operation (i.e. submitting a new transaction).

## 2 Byzantine clients

There are two possible scenarios to consider with Byzantine clients: a client that sends corrupt messages or attempts to violate constraints (for example, spending more than their ledger balance allows) in a coherent way by sending the same messages to all servers, or a malicious client that sends individually correct messages, but sends a different one to each replica in an attempt to cause the state of the system to diverge between replicas. The first scenario is mitigated using the mechanisms implemented in stage 1; the second scenario is specific to this phase.

A client that sends different, individually correct messages to each replica can perform a denial of service attack over other ledgers, for example by signing different transactions to the same destination with varying amounts of HDS coins. If a byzantine quorum accepts the transaction, the destination ledger will read inconsistent results when checking their account, and will not be able to proceed.

We solved this problem by implementing a signed echo broadcast mechanism, where servers only accept requests that have verifiably also been sent to a byzantine quorum of other replicas. All write operations (sending transactions, receiving transactions, and write-backs) are protected by this mechanism. This mechanism has two phases for each write; in the first one, clients send their requests to each replica, and wait for a byzantine majority of them to reply with their signature of the request. (The replicas do not actually change the register in this phase.) In the second phase, the client sends these signatures to all replicas, which only change the register if a byzantine majority of the signatures can be verified.