# Highly Dependable Systems

# Sistemas de Elevada Confiabilidade

**2021-2022**

BFT Banking (BFTB)

Stage 2

## Goals

The goal of the second stage of the project is to extend the implementation of the first stage to tolerate the possibility that the system may be subject to Byzantine faults, affecting both server and client processes.

More precisely, we will keep the same specification for the BFTB system from the previous stage, but enhance the client and server implementations to make them resilient to Byzantine faults.

On the server side, students will need to replace the single server used in the first stage with a set of N replicas, which collectively implement a Byzantine Fault Tolerant service. The size N of this set is chosen in such a way that depends on a value *f*, which is a system parameter representing the maximum number of tolerated Byzantine faults, i.e., attacks that can succeed while preserving the correctness of the BFTB system. Additionally, the servers shall protect themselves from Denial of Service attacks, by employing anti-spam mechanisms aimed at reducing the rate at which they may receive expensive operations.

On the client side, in this stage we will assume that a malicious user may alter arbitrarily the client library to attack the system, e.g., in order to break the system's consistency. Students must design and implement solutions to protect the BFTB system from such malicious users. It is however outside of the scope of the project to integrate on the client side techniques that aim at protecting a legitimate user from a malicious client library (that may, e.g., leak the user's private key or produce fake replies for the user without even contacting the servers).

## Design requirements

The basic change to the existing design in this stage of the project is to replace the client-server communication between the library and the server with a replication protocol between the library (acting as a client of that protocol) and a set of server replicas.

Each bank account shall be logically mapped to a (1-N) atomic register, where *check_account()* and *audit()* operations shall be treated as read operations, whereas *send_amount()* and *receive_amount()* shall be treated as write operations. Regarding the specifications of atomic registers, *open_account()* should be regarded as a special initialization operation, to be executed before any subsequent read/write operation.

## Implementation Steps

To help in the design and implementation task, we suggest that students break up the project into a series of steps, and thoroughly test each step before moving to the next one. Having an automated build and testing process (e.g.: JUnit) will help students progress faster. Here is a suggested sequence of steps:

- Step 1: Replicate the server, without implementing any scheme to synchronize the state of the various replicas and assuming non-byzantine clients. This will involve starting N server replicas instead of one, and replacing the client to server communication with a loop that repeats each communication step N times.
- Step 2: As for the next step, students are advised to implement first the (1,N) Byzantine Regular register algorithm in Section 4.7 of the course book (Introduction to Reliable and Secure Distributed Programming, 2nd Edition)
- Step 3: Next, consider a transformation from (1,N) Byzantine Regular register to a (1,N) Byzantine Atomic register.
- Step 4: The (1,N) Byzantine Atomic register assumes that only servers can be Byzantine, but that clients are subject only to crash faults. Reason on what could happen in case of malicious clients that attempt to corrupt the state of the register and propose, if needed, solutions to cope with this issue.
- Step 5: Implement the new set of dependability tests.

## Submission

Submission will be done through Fénix. The submission shall include:

- a self-contained zip archive containing the source code of the project and any additional libraries required for its compilation and execution. The archive shall also include a set of demo applications/tests that demonstrate the mechanisms integrated in the project to tackle security and dependability threats (e.g., detection of attempts to tamper with the data). A README file explaining how to run the demos/tests is mandatory.
- a concise report of up to 6,000 characters addressing:
    - explanation and justification of design changes introduced in the second stage,
    - explanation of the dependability guarantees provided by the system

The deadline is **April 22 at 23:59**. More instructions on the submission will be posted in the course page.