Highly Dependable Systems

Sistemas de Elevada Confiabilidade

2021-2022

BFT Banking (BFTB)

Stage 1

Goals

The goal of this project is to develop a highly dependable banking system, with Byzantine Fault Tolerant (BFT) guarantees, named BFT Banking (BFTB). As discussed below, the dependability guarantees of the system will be strengthened throughout stage 1 and stage 2.

The BFTB system maintains a set of bank accounts, each account being uniquely identified by a public key pair. Each account stores:

- the current balance
- the history of incoming (i.e., credit) and outgoing (i.e., withdrawal) operations for that account.

A client of the system can perform transfers between a pair of accounts as long as it has the corresponding private key of the crediting account (the one from which money is withdrawn). The set of all accounts and the associated balance should satisfy the following dependability and security guarantees:

- the balance of each account should always be positive
- the state of the accounts cannot be modified by unauthorized users
- the system should guarantee the non-repudiation of all operations issued on a bank account (both crediting and withdrawals) by both parties (sender and receiver) involved in the transaction
- all operations that modify the balance of the accounts should be logged such that they can be verified by an auditor

Students can assume that there is a Public Key Infrastructure in place, although, for simplicity, clients and servers should use self-generated public/private keys. Furthermore, in this stage students should assume that there is a single, non-malicious server. This restriction will be lifted in stage 2 by requiring multiple servers.

The client API has the following specification:

- open_account(PublicKey key,…)
  Specification: create a new account in the system with the associated public *key,* before first use. In particular, it should make the necessary initializations to enable the first use of the BFTB system. The account should start with a pre-defined positive balance.
- send_amount(PublicKey source, PublicKey destination, int amount, …)

Specification: send a given amount from account *source* to account *destination*, if the balance of the source allows it. If the server responds positively to this invocation, it must be guaranteed that the *source* has the authority to perform the transfer. The transfer will only be finalized when the *destination* approves it via the receive_amount() method defined below.

- check_account(PublicKey key,…)
  Specification: obtain the balance of the account associated with the *key* passed as input. This method also returns the list of pending incoming transfers that require approval by the account's owner, if any.
- receive_amount(PublicKey key, ...)
  Specification: used by the recipient of a transfer to accept in a non-repudiable way a pending incoming transfer that must have been previously authorized by the sender.
- audit(PublicKey key,…)
  Specification: obtain the full transaction history of the account associated with the key passed as input. The returned history should reflect the order of execution of operations.

Design requirements

The design of the BFTB system consists of two main parts: a library that is linked with the application and provides the API specified above, and the server that is responsible for keeping the set of bank accounts. The library is a client of the server, and its main responsibility is to translate application calls into requests to the server. Anomalous inputs shall be detected by the server, which should generate appropriate exceptions and return them to the client.

The following assumptions are done on the system's components:

- The client library is trusted.
- For this stage, the server is assumed to be honest and is only subject to crash failures from which it eventually recovers.
- The attacker can drop, reject, manipulate and duplicate messages.

The system shall operate under the assumption that the communication channels are not secured, in particular solutions relying on secure channel technologies such as TLS are not allowed.

Students must analyze the potential threats to the system, including man-in-the-middle, replay and Sybil attacks, and design application level protection mechanisms to cope with them. There are several approaches to address these issues, so it is up to students to propose a design and justify why it is adequate.

Implementation requirements

The project should be implemented in Java using the Java Crypto API for the cryptographic functions.

We do not prescribe any type of communication technology to interface between the client and the server. In particular, you are free to choose between using sockets, a

remote object interface, remote procedure calls, JSON, or a SOAP-based web service (among others).

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: As a preliminary step, before starting any implementation effort, make sure you have carefully analysed and reasoned about security and dependability issues that may arise and the counter-measures that you plan to integrate in your solution to deal with them. Only then, move to step 2.
- Step 2: Simple server implementation without dependability and security guarantees. Design, implement, and test the server with a trivial test client with the interface above that ignores the crypto parameters (signatures, public keys, etc.)
- Step 3: Develop the client library and complete the server – Implement the client library and finalize the server supporting the specified cryptographic operations.
- Step 4: Dependability and security – Extend the system to support the dependability and security guarantees specified above.

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 4,000 characters addressing:
    - explanation and justification of the design, including an explicit analysis of the possible threats and corresponding protection mechanisms.
    - explanation of the integrity guarantees provided by the system.
    - explanation of other types of dependability guarantees provided.

The deadline is **April 1 at 23:59**. More instructions on the submission will be posted in the course page.