Highly Dependable Systems

Sistemas de Elevada Confiabilidade

2017-2018

HDS Coin
Stage 1

Goals

Cryptocurrencies have become extremely popular with many new cryptocurrencies appearing every day. Following this trend, the goal of the project is to create a new cryptocurrency – HDS Coin – with dependability guarantees which will be strengthened throughout stage 1 and stage 2.

The HDS Coin system maintains a set of ledgers, each ledger being associated with a distinct account and uniquely identified by a public key. Each ledger stores:

- the history of incoming and outgoing operations for each individual account,
- the current balance of the account.

A client of the system can perform transfers between a pair of accounts as long it has the corresponding private key of the crediting account (the one from which money is withdrawn).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 forms a ledger that must enforce the following dependability and security guarantees and invariants:

- the balance of each account should be always positive, i.e., transfers that would lead to negative balances should be rejected by the system.
- the accounts cannot be tampered with by a malicious server or clients not owning the respective private key.
- the system should guarantee the non-repudiation of all operations issued on a bank account (both sending and receiving HDS coins) by both the parties involved in the transaction.
- all operations that modify the balance of the accounts should be logged such that they can be verified by any 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.

The client API has the following specification:

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

Specification: submit the request for transferring a given amount from account source to account destination, if the balance of the source allows it. If the server responds positively to this call, it must be guaranteed that the source has the authority to perform the transfer. The transfer will only be finalized when the receiver approves it via the receive_amount() method (see below).

- check_account(PublicKey key,…)
  Specification: obtain the balance of the account associated with key. 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 recipient of a transfer to accept in a non-repudiable way a pending incoming transfer that must have been previously authorized by the source.
- audit(PublicKey key,…)
  Specification: obtain the full transaction history of the account associated with key.

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.

Design requirements

The design of the HDS Coin 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 ledgers associated with the 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-side.

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

- For now, the server is assumed to be honest and is only subject to crash failures from which it eventually recovers.
- An attacker can drop, manipulate and duplicate messages.

Students will have to 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 you to propose a design and justify why it is adequate.

Implementation requirements

Your project must 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, or a SOAP-based web service.

Implementation Steps

To help in your design and implementation task, we suggest that you break up this task 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 you progress faster. Here is a suggested sequence of steps that you can follow:

- Step 1: As a preliminary step, before starting any implementation effort, make sure to have carefully analyzed 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 crypto 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 6 at 17:00**. More instructions on the submission will be posted in the course page.