Instituto Superior Técnico

Highly Dependable Systems

Project Report – First Delivery

Duarte Miguel nº 87527

Pedro Teixeira nº 87555

Pedro Agostinho nº 87557

Index

Introduction

Nowadays, communication via client-server architecture is common and most of them can be critical like a bank transfer or a call to the hospital. So the security of these communications is something that we must consider.

There are many security problems that we can consider but for this section of our work we will only consider that the communication channel is insecure.

Since the client and the server communicate through an insecure channel, we need to protect the communication from man-in-the-middle attacks that can drop, reject, manipulate and duplicate messages.

Development

We developed a system called DPAS, a board-based forum where each user has a personal board, where only he can write and a general board where everyone can write.

System and General Architecture:

The system is based on a client-server architecture communicating via remote objects using the JavaRMI library. The server has a remote object implementing the following interface:

1. A client can register himself in the system:

register (client's public key,
user's number,
signature)

2. A client can post an announcement in his board:

post (client's public key,
message,
sequence number,
signature)

3. A client can post an announcement in the general board:

postGeneral (client's public key,
message,
sequence number,
signature)

4. A client can read from another user's board:

read (target-user's public key,
number of announcements,
sequence number
signature
client's public key)

5. A client can read from the general board:

readGeneral (number of announcements,
sequence number
signature
client's public key)

Public and Private Key Management:

For testing purposes, before a server is run, a unique number identifier must be indicated (at the moment only server 1 works). The client's number must also be indicated, as well as the number of the server this client wants to connect to. This is because it's assumed that all clients and all servers know all present public keys in the system and each of them needs access to its keystore and certificate which are uniquely named (client1's keystore is client1_keystore.jks and client1's certificate is client1_certificate.crt). All keys in the system are RSA keys with 2048 bits.

Each member of the system has access to its own private keystore, protected with a password and each private key in this keystore is also protected with a password (for testing purposes all passwords follow this format: client1's keystore password = client1's private key password = client1password). Each member also emits a self-signed certificate carrying its public key. Each member also knows all certificates and can extract the public keys from them.

In short, each member has access to:

- its private keystore (if it knows the keystore's password)

- its private key, extracted from the keystore (if it knows the private key's password)

- his and everyone else's certificates from which it can extract the public keys.

Message Exchange:

To guarantee message integrity and avoid message duplication, every message is concatenated with a sequence number (and with any other information sensitive to change), hashed, and encrypted with the sender's private key, therefore, the signature for each method is the following:

client to server:

register:

$\{(user's\ number)_{digested}\}_{encrypt:\ client's\ private\ key}$

Post:

$\{(\text{message} + \text{sequence number})_{digested}\}_{encrypt:\ client's\ private\ key}$

postGeneral:

$\{(\text{message} + \text{sequence number})_{digested}\}_{encrypt:\ client's\ private\ key}$

read:

$\{(\text{target's public key} + \text{number of announcements} + \text{sequence number})_{digested}\}_{encrypt:\ client's\ private\ key}$

readGeneral:

$\{(\text{number of announcements} + \text{sequence number})_{digested}\}_{encrypt:\ client's\ private\ key}$

server to client:

(for now, only read and readGeneral return acknowledgements, we're still considering if the rest of the methods also need similar structures)

read:

$\{(announcements)_{digested}\}_{encrypt:\ server's\ private\ key}$

readGeneral:

$\{(announcements)_{digested}\}_{encrypt:\ server's\ private\ key}$

Concurrency:

Every time an announcement is added to the General Board, this method is locked to avoid having two announcements posted with the same id.

State Persistence:

For now, the relevant server state is saved in a .txt file every time a method returns and if already existing, loaded when a server loads.

To Do:

In this intermediate phase, there are still DEBUG messages to be omitted and some methods in our AsymmetricCrypto library, are deprecated and are not used. Furthermore, most of the application exceptions are not properly treated, but do not cause the application to crash either (only print their stack trace).