Dependable Password Manager
Report

Mateus Cordeiro
Instituto Superior Técnico
Mestrado Engenharia Informática
n°76463
mateus.cordeiro@tecnico.ulisboa.pt

Constantin Zavgorodnii
Instituto Superior Técnico
Mestrado Engenharia Informática
n°78030
constantin.zavgorodnii@tecnico.ulisboa.pt

Henrique Alves
Instituto Superior Técnico
Mestrado Engenharia Informática
n°87891
henriquefalves@tecnico.ulisboa.com

1 Improvements from last checkpoint

Upon receiving feedback from our project in the last checkpoint, we decided to change some parameters.

1.1 Confidentiality

On the last delivery, we failed to ensure confidentiality, i.e the domain, username and password were stored in plain text on the server. Now, before leaving the client, the password is cyphered with RSA/ECB/PKCS1Padding using the client's public key. The domain and username are being concatenated and hashed using SHA-256 and only then sent to the server. This assures confidentiality and impossibility to relate any information with other users information since the pairs username and domain are unique.

1.2 Non-repudiation

We were storing only the password in the server, now we created a data structurer *UserData* which is bundle of the signature and all information present in that signature(domain+username, password, communication information) which enables to prove at any time the authenticity of that password. A history of passwords was also implemented. This data structurer can be save in a persistent way.

1.3 Integrity

Before, the client was retrieving the password and there were no verifications whether it had been changed while on the server. Now, as we store not only the password, but all the contents of its message (signature and components of the signature), it is also verified on the client-side if the data was tampered with.

1.4 Freshness

Replay attacks are an issue. We studied 3 possible solutions on the previous report: sequence numbers, timestamps and challenge-response. We had implemented sequence numbers to ensure freshness, but then decided it was easier to work with N clients if we used challenge-response. Now, before a client performs any operation, it must request a challenge to each server. This challenge is a random byte with a length of 128.

2 New project features

2.1 System architecture

Added *CommunicationLink* class, which implements the communication abstractions, such as Perfect Links and Best-Effort Broadcasts.

2.2 Communication Abstractions

A *perfect link* is implemented as a TCP connection, as we assume its reliable nature guarantees the message delivery. The *best-effort broadcast* is implemented on top of the perfect link.

2.3 Algorithm Implementation

As suggested, we implemented the (N,N) Atomic Register Algorithm. To do so, we added all the extra variables in both the Client and Server Front-Ends.

In our implementation, the servers are independent from each other and do not communicate between themselves, only with the clients.

With the data authentication between end-points we can employ Byzantine (majority) quorums, considering f faults and we can return a response to the user if we have responses *more than* (N+f) / 2 servers.Therefore, algorithms tolerating Byzantine faults usually require that only f <N/3 processes may fail. After testing, we assured that our solution works with both N clients and N servers.

2.4 Byzantine Servers

As expected, we assumed the servers could be byzantine and have allowed them 3 functionalities.

2.4.1 Crashing

The servers crash either during a read or a write operation.

2.4.2 Timeout

The server has a delay during a read of write operation. To the client, this has the same effect as crashing, but the server is still connected afterwards.

2.4.3 Incorrect Answer

The server tries to change its answer to an incorrect one.

2.5 Byzantine Clients

Having arbitrarily faulty clients was not consider on the scope of this project, although we reason that this would lead to a incorrect behavior of the system.