Dependable Password Manager
Report Stage 1

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

Abstract

In the project, we implement a reliable password manager. This report details our architectural and security choices. The threats were identified using the STRIDE model. It also explains how each threat is faced.

The communication is performed using both symmetrical (AES) and asymmetrical (RSA) keys. The messages carry a signature, are padded and have sequence numbers.

1. Architecture

Figure 1 represents the project architecture. The Client Application represents the user interface. The classes Client, Client Crypto and Client Front End represent the library the user has access to. The class Crypto Library has the methods regarding data encryption, decryption, signatures and verification. They are parallel for both the Client and the Server.

To implement freshness, we used sequence numbers and modified the Server API to always receive an extra argument.

The classes Client Crypto, Server Crypto and Server implement the Server API. The class Client implements the Client API as specified originally. The classes Client Front End and Server Front End implement the Communication API. This one is similar to the Server API, but all the arguments are reduced to a single argument from the class Message. Both Front Ends decouple this class into the original arguments upon reception. A Message also contains the signature, randomIv and passwordIv.

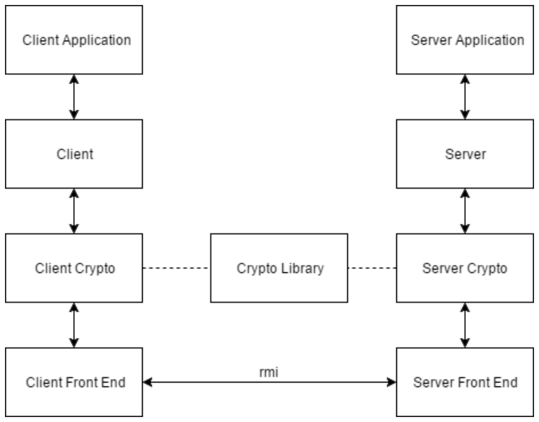


Figure 1. Project architecture

1.1 Assumptions

For the project, we assumed certain circumstances were met:

- The implementation requires the program to run on top of the Java VM with the version 8.
- The user must provide a KeyStore of the type JCEKS(Java Cryptography Extension KeyStore) containing a RSA key-pairs with a size of 2048-bit referring to the public and private key of the user, public key of the server and an AES(Advanced Encryption

Standard) 128-bit key. The password is also needed to unlock the KeyStore.

- Anyone can have access to the Public Key of the server and each Client.
- The cryptographic protocol Transport Layer Security (TLS) was not allowed to use.

2. Threat Modeling

To identify the threats the system might encounter when exposed to external identities we chose to follow the STRIDE model:

Spoofing Illegally access and use another user's credentials, such as password.

Tampering with data Changes in data when in transit, jeopardizing its integrity. We assume both the client and the server have no malicious intent towards the local data.

Repudiation Changes on data with no traceability to the responsible identity.

Information disclosure Confidentiality of the information can be compromised on the network and server. We assume the server will not corrupt its local data, but may access it.

Denial of service Rendering the service unavailable by either flooding the server with requests, dropping packets or introducing network delays. These threats were considered out of the project's scope.

Elevation of privilege Exploiting a bug in the system in which the user elevates its status to access privileged information. This threat does not apply to the project.

3. Security

This section details our group's choices to counter the aforementioned threats.

3.1. Data at rest

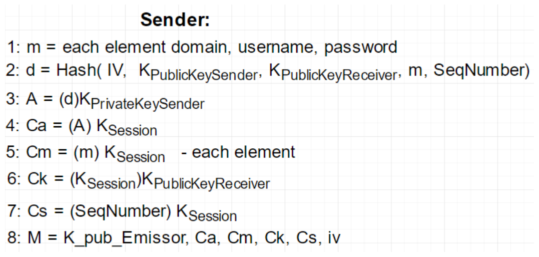
Before leaving the client local machine all data(domain, user,password) is separately encrypted using an AES with the CBC (Cipher Block Chaining) mode and the padding described in PKCS#5, to prevent any correlation between the passwords(i.e. same username for different domains, different username with different domains) the initialization vector(IV) is different for each unique tuple (username, domain) and it is stored by the server. This encryption provides the **Confidentiality** needed by taking into account the server being "honest but curious". Because of implementation problems this feature had to be postponed to the next delivery.

Integrity is not ensured but discussed by the group if the encryption was done using the Galois Counter Mode (GCM), The key has a length of 128-bit, known only by the client.

The way the symmetric key is provided was highly taken in account, we thought the user could provide a master password and then using the PBKDF2(Password-Based Key Derivation Function 2) we could produce a 256-bit key to encrypt the data. The harder implementation, the partial limitation of the AES 128-bit sized key regarding cryptography law in different countries [1] and the need for another user input that comprise user experience were the arguments used for implementing the solution stated before.

3.2. Transport Security

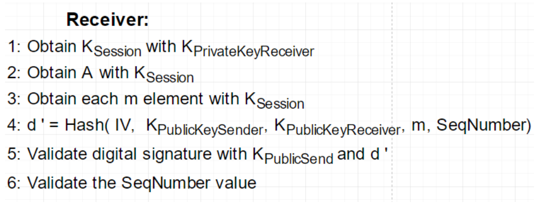
Both Figure 2 and 3 describe the implementation on the our protocol that ensures the security over the network.



Sender:

1: m = each element domain, username, password
2: d = Hash(IV, $K_{PublicKeySender}$, $K_{PublicKeyReceiver}$, m, SeqNumber)
3: A = (d)$K_{PrivateKeySender}$
4: Ca = (A) $K_{Session}$
5: Cm = (m) $K_{Session}$ - each element
6: Ck = ($K_{Session}$)$K_{PublicKeyReceiver}$
7: Cs = (SeqNumber) $K_{Session}$
8: M = K_pub_Emissor, Ca, Cm, Ck, Cs, iv

Figure 2. Sender Side of the Transport Security Protocol



Receiver:

1: Obtain $K_{Session}$ with $K_{PrivateKeyReceiver}$
2: Obtain A with $K_{Session}$
3: Obtain each m element with $K_{Session}$
4: d ' = Hash(IV, $K_{PublicKeySender}$, $K_{PublicKeyReceiver}$, m, SeqNumber)
5: Validate digital signature with $K_{PublicSend}$ and d '
6: Validate the SeqNumber value

Figure 3. Receiver Side of the Transport Security Protocol

3.2.1 Confidentiality

On top of the encryption done by the user locally we use a Session Key generated every type the application is turn

[1]This limitation can be overcome by installing in the JDK/JRE the JCE Unlimited Strength Jurisdiction Policy Files provided by Oracle

on by the client. The Key size is 128-bit and is an instance of AES with the CBC (Cipher Block Chaining) mode and the padding described in PKCS#5, the initialization vector is generated by the user.

3.2.2 Integrity, Authentication and Non-Repudiation

It is important to guarantee the authenticity of the user who sent a given message, to provide this security feature we use the key-pair present in the KeyStore generated for RSA and the length of 2048-bits. The algorithm use is the ”SHA256withRSA”, by providing the data and the Private Key of the sender we are able to prove on the receiver side by using only the Public Key of the Sender. There are other algorithms that supply authenticity, message authentication code (e.g. HMAC), they use secret key shared by the two parties in the communication. Although more efficient they do not provide Non-Repudiation of the message. Given this characteristic the digital signature scheme is well suited for a low-bandwidth system so no drawbacks in efficiency.

3.2.3 Freshness

Replay attacks are an issue. When a user requests to save a password *x*, a man-in-the-middle may keep this message from reaching the server. If, later on, the user changes his password again to *y* and the server accepts it, the attacker can send the original message and the password will be changed to an older client request.

To counter this attack, we studied 3 possible solutions: sequence numbers, timestamps and challenge-response.

Sequence Numbers Each client has a sequence number attached to his requests and, whenever a message is sent, that number is incremented by 1. It is also included in the signature. A message is only accepted if its sequence number is +1 than the one in the previous message sent. All other sequence numbers are rejected.

TimeStamps Whenever the sender was about to send a message it would attach the time to the message. The timeStamps was the problem of clock synchronization and the definition of a time window to accepted a given message.

Challenge-Response Challenge-response ensures freshness by issuing a challenge to the sender that only he can solve and, upon answer validation, the message is accepted. The main drawback to this technique is the extra round trip spent on confirmation.

After discussing these 3 solutions, we opted to use sequence numbers. For that, we added a method to the Server API called *getSequenceNumber* (with the client’s public key as sole argument), invoked upon client startup.The server will generate a random sequence number return this value to the client. It is important to mention that the this values is included in the digital signature.

This solution ensures freshness in all the messages sent.