Highly Dependable Systems

Sistemas de Elevada Confiabilidade

2016-2017

Dependable Password Manager

Stage 1

Goals

Recently, there have been news of compromised password managers, in particular those associated with web browsers. Following these lines, the goal of the project is to implement a distributed password manager with dependability guarantees, which will be strengthened throughout the course.

The password manager has the notion of domains (e.g.: a website), username and passwords. We start with a simplified specification: each client can store an arbitrary number of passwords and each password can be updated several times. Naturally, clients should not be able to access the domains, usernames or passwords of other clients. For now we assume the server can be "honest but curious" so client information cannot be stored in clear text. At later stages, we will further assume that the server can be maliciously compromised. The value that is read by a client should reflect previous updates that completed before the read started (and it may or may not reflect updates that happen concurrently).

To simplify the design and evaluation, at this stage, it is assumed that the public/private key pairs of all parties are self-generated beforehand and can be trusted without the need for additional verifications (e.g., to check whether a give public/private has been revoked).

Design requirements

The design of the dependable password manager consists of two main parts: a library that is linked with the application and provides the API specified below, and the password manager server that is responsible for storing the (domain, username, password) triplets.

The library is a client of the server, and its main responsibility is to translate application calls into requests to the server. The client API has the following specification:

- init(KeyStore ks, ...)
  Specification: initializes the library before its first use. This method should receive a reference to a key store that must contain the private and public key

of the user, as well as any other parameters needed to access this key store (e.g., its password) and to correctly initialize the cryptographic primitives used at the client side. These keys maintained by the key store will be the ones used in the following session of commands issued at the client side, until a close() function is called.

- register_user()
  Specification: registers the user on the server, initializing the required data structures to securely store the passwords.
- save_password(byte[] domain, byte[] username, byte[] password)
  Specification: stores the triple (domain, username, password) on the server. This corresponds to an insertion if the (domain, username) pair is not already known by the server, or to an update otherwise.
- retrieve_password(byte[] domain, byte[] username) -> byte[] password
  Specification: retrieves the password associated with the given (domain, username) pair. The behavior of what should happen if the (domain, username) pair does not exist is unspecified.
- close(): concludes the current session of commands with the client library.

The API of the server is the following:

- register(Key publicKey)
  Specification: registers the user in the server. Anomalous or unauthorized requests should return an appropriate exception or error code
- put(Key publicKey, byte[] domain, byte[] username, byte[] password)
  Specification: stores the triple (domain, username, password) on the server. This corresponds to an insertion if the (domain, username) pair is not already known by the server, or to an update otherwise. Anomalous or unauthorized requests should return an appropriate exception or error code
- get(Key publicKey, byte[] domain, byte[] username) -> byte[] password
  Specification: retrieves the password associated with the given (domain, username) pair. Anomalous or unauthorized requests should return an appropriate exception or error code.

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.

Nonetheless, the system shall be able to guarantee the integrity and confidentiality of the information stored/returned to the users. These integrity and confidentiality guarantees shall encompass not only the passwords it stores, but also of the corresponding domains and usernames. The system shall also ensure that only legitimate users are allowed to store/retrieve information into/from the server.

Finally, the system shall support non-repudiation of any action that led to altering the state of the users' passwords.

Students are required to dissect and analyze the potential threats to the system, including man-in-the-middle and replay attacks, and design application level protection mechanisms.

There are several approaches to accomplish the above goals, so it is up to the student to propose a design and justify why it is adequate.

Implementation requirements

Your project has to 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. As mentioned above, though, communication SHALL not be secured (e.g., solutions like HTTPS or TLS are not allowed).

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: Password server with no security guarantees – Design, implement, and test the password and a trivial test client with the interface above that ignores the crypto parameters (signatures, public keys, etc.)

Step 2: Develop the client library and complete the server – Implement the client library and finalize the server supporting the specified crypto operations.

Step 3: Integrity validation– Extend the system to ensure integrity of the passwords stored by the server and returned by the server to the clients.

Step 4: Extend the system to be robust against additional threats such as replay attacks and prevent unauthorized access to the service.

Submission

Submission will be done through Fenix. 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 reports of up to 4,000 characters addressing:
    - explanation and justification of the design

- explanation of the threats considered and the security mechanism that deal with them

The deadline is March 17, at 17:00. More instructions on the submission will be posted in the course page.