Security Protocols – Homework Project Assignment
March 5, 2019

The goal of the semester homework project is to use the cryptographic building blocks that we learned during the lectures and classroom exercises in practice. We believe that a learning-by-doing approach leads to deeper understanding of the subject.

To make it fun and somewhat less of an effort, the semester project should be carried out in groups of 3 students. This also allows students to gain experience in team work and collaborative development of software. So, please, team up with your buddy, and be prepared for the challenge!

The homework project has two phases: (1) in the first phase, teams have to come up with a design and submit a sufficiently detailed description of their envisioned system, and (2) in the second phase, teams have to implement their designs using a real crypto library.

In the design phase, teams should produce a 3-4 pages document that contains the description of their design in sufficient details, and submit that document **by April 7, 2019 (midnight)**. The design document should contain at least the following:

- functional requirements of the application
- attacker model (assumptions about the goals and capabilities of an attacker)
- security requirements of the application (induced by the attacker model)
- system architecture (modules and their relationships)
- cryptographic protocols, including key exchange (if any)
- brief security analysis (how the design satisfies the security requirements).

In the implementation phase, teams should implement their design. The preferred language for implementation is Python 3 with the PyCryptodome library, but teams could also use other languages and crypto libraries. Teams should complete their implementation and submit their work in a zip file **by May 8, 2019 (midnight)**. The submissions should contain at least the following:

- commented source code
- some minimal documentation of the code
- a 3-4 min video showing the working application and its basic usage.

As for the problem to solve, you have 3 options to choose from:

Option A: secure, multi-party chat application

If you choose this option, then you have to develop a secure, multi-party chat application. Some requirements for the envisioned chat application are the following:

- it must support chat groups larger than 2 participants (i.e., multi-party chat)
- chat sessions can be handled by a chat server, which would be trusted for forwarding chat messages to the participants of a chat session, but we don't want that the server can access the content of those messages
- hence we want to use end-to-end encryption for chat messages
- we also want to protect integrity of chat messages, want to avoid accepting replayed chat messages, and we want to be sure that the source of each chat message is authenticated

- you may assume that participants have public key certificates that are issued off-line (no need to implement this as part of the chat application or any registration process), and you can use a simplified certificate format for the purposes of this project
- the private key of each chat participant can be stored locally by the given participant in a password protected file
- relying on a chat server is not mandatory; your application can also be peer-to-peer
- optional: the application may not assume that all chat participants are on-line at the same time; participants of a chat session may join the session later, go off-line, come back on-line later on, etc.

Implementations can be command line programs or they may have a graphical user interface. Please note that the focus should be on getting the crypto part correct, and not on fancy features and nice look.

Option B: secure file transfer application

If you choose this option, then you have to develop a secure file transfer application. Some requirements for the envisioned application are the following:

- it must be a client-server application supporting file upload to and download from the server
- it must use a protocol that ensures confidentiality and integrity of control messages (e.g., commands sent to the server) and transferred files
- the protocol must also ensure message origin authentication and replay protection
- In addition, the application should support the concept of sessions, at the beginning of which the client and the server should mutual authenticate each other
- client authentication can be based on passwords, but then the passwords should be stored on the server in a way that protect against on-line password guessing attacks and off-line dictionary attacks (if the password database is stolen by the attacker)
- the application should support commands similar those of FTP applications, but you can focus on the most essential commands, you don't need to implement all commands of FTP.

Implementations can be command line programs or they may have a graphical user interface. Please note that the focus should be on getting the crypto part correct, and not on fancy features and nice look.

Option C: secure password management application

If you choose this option, then you have to develop a secure password management application. Some requirements for the envisioned password management application are the following:

- it must support encrypted storage of passwords for multiple accounts, where encryption can use a key derived from a master password
- your scheme should withstand off-line dictionary attacks (so even if the master password is somewhat weak, it should be difficult for attackers to figure it out off-line using the encrypted password file/database)
- the application should support generation of a new password for a new account added (so the user can fill in the account name, URL, …, but the application should generate the password)
- the application should support searching by account name, URL, …, decrypting the password of a selected account, and copying it to the clipboard

- the time the master password and any decrypted account password spend in memory must be minimized (so you should not read in, decrypt, and keep in memory the password file/database during the entire time your application is running).

Implementations can be command line programs or they may have a graphical user interface. Please note that the focus should be on getting the crypto part correct, and not on fancy features and nice look.

Useful hints

- Do not over-complicate things; try to make your protocols as simple as you can, because complex protocols are more likely to have flaws and they are more difficult to implement.
- Try to use cryptographic primitives that are supported by the crypto library that you will use in your implementation.
- You can re-use stuff that you have already implemented in classroom exercise sessions (e.g., secure channel implementations).