Applied Cryptography – Project assignment
November 5, 2018

The goal of the semester project is to use the cryptographic building blocks that we learned during the lectures 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 can be carried out in groups of 2-3 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 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 2-3 pages document that contains the description of their design in sufficient details (see some hints for content below), and submit that document **by November 16, 2018 (midnight)**.

In the implementation phase, teams should write their programs and submit their work (source files in a zip file) **by December 7, 2018 (midnight)**. You can use Python and PyCryptodome for your implementation, but this is not mandatory. If you prefer another programming language, then you can use that with an appropriate crypto library.

Finally, all teams will run a demo in the classroom on **December 14, 2018**.

As for the problem to solve, you have 2 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.

In the second (implementation) phase, teams will implement their chat application. As sort of some help, we prepared and made available some Python code that provides a simplified abstraction of a network interface. In the folder 'netsim', we provided a package that simulates message sending via a network, but does not use real networking. Teams can use this package during development as it makes debugging easier (messages sent and received are saved in files and you can inspect them). The package is documented and examples are provided for sending and receiving messages.

While useful, teams are not mandated to use the code provided in 'netsim' in their implementation: if you want to start from scratch (e.g., because you want to implement your project in another programming language, or for any other reason), then we will not stop you doing that!

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 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 homework assignments (e.g., secure channel implementations).
- Use Piazza for discussion with other groups, and in particular to share technical difficulties that you encounter; others may have already solved the very same problem, and can help you!

Design document outline

- Overview of system architecture and operation (main components and envisioned operation of the application as a whole)
- Attacker model (assumptions made about attackers relevant for your application)
- Security requirements identified based on the attacker model
- Detailed description of your system satisfying the security requirements
    - e.g., in case of Option A: description of the secure channel protocol used by chat participants (message sequence figures, message format specification, description of how messages are processed and protocol states are handled)
    - e.g., in case of Option B: description of how passwords are generated and how they are encrypted.

Project presentation outline

- Brief overview of your design
- Brief overview of your implementation, and some key features that you are proud of
- Demo (including normal operation and some potential attacks)
- Main difficulties that you encountered during the project
- Main lessons that you learned, and some advice you would give to students of the next semester