Applied Cryptography – Project assignment
October 25, 2019

The goal of the semester project is to use the cryptographic building blocks that we learned during the lectures in practice. To make it fun and somewhat less of an effort, the semester project can be carried out in groups of max 4 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 4-5 pages document that contains the description of their design in sufficient details (see some hints for content below), and submit that document **by November 13, 2019 (midnight)**.

In the implementation phase, teams should write their programs and submit their work (source files in a zip file) **by December 9, 2019 (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 present the project and run a demo in the classroom on **December 13, 2019**.

The application

The application that teams have to develop is a **secure file transfer** application. It should consist of a client program and a server program that can communicate with each other securely. The client should interact with the user via some user interface (console or GUI). The user can type in commands, which are sent to the server by the client. The server should execute the received command and respond with some result or acknowledgement. The client should then provide some feedback to the user about the result of the operation. The commands are related to file operations such as uploading files to and downloading files from the server, creating folders, listing the content of a folder, etc. If you have ever used FTP, you can pretty well imagine how this application should work.

Your file transfer application should support at least the following commands:

- MKD – creating a folder on the server
- RMD – removing a folder from the server
- GWD – asking for the name of the current folder (working directory) on the server
- CWD – changing the current folder on the server
- LST – listing the content of a folder on the server
- UPL – uploading a file to the server
- DNL – downloading a file from the server
- RMF – removing a file from a folder on the server

The server should be able to handle folders and files of multiple users. Obviously, users should have access only to their own folders and files. Ideally, the server should be able to handle multiple

parallel connections with different clients, but to make things simple, it is sufficient if your application can handle a single connection at a time. This would mean that while a user is logged in and interacts with the server, other users would not be able to log in. However, when the user logs out, the server becomes available again to other users.

The user should login to the server with a password, and to make sure that the password is sent to the server (and not to an attacker), the server should be authenticated first by cryptographic means. You can assume, for instance, that the server has a public key – private key pair, and the client possesses already the public key of the server (i.e., the public key is obtained out-of-band). The server should not store passwords in cleartext in order to avoid leaking them if the server is hacked. Similarly, the server should not store its private key in cleartext. This is all we want to prescribe here, identifying other security requirements is part of the project.

The task

In the first (design) phase of the project, teams should design a security architecture for the file transfer application. This includes identifying attacker models, deriving security requirements from the attacker model and the functional specification of the application given above, and designing the cryptographic protocols that would satisfy the security requirements. All this needs to be written up in a design document that contains the following:

Design document outline:

- Overview of the application and its functional requirements
- Attacker models and trust assumptions
- Security requirements derived from the attacker models
- Protocol specifications
    - General approach followed to satisfy the security requirements
    - For each protocol:
        - Assumptions about the availability of long-term keys
        - Message sequence diagram and message processing rules
        - Cryptographic algorithms used and their parameters
        - Exact format of each protocol message
        - Representation of protocol state
        - Protocol state machine (if the protocol is complex)
- Arguments on why the security requirements are satisfied by the proposed protocols

The general requirement on the depth of details of the design document is that someone should be able to start implementation based on it without making decisions him- or herself that affect security.

In the second (implementation) phase, teams will implement their file transfer 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!

Client implementations can be command line programs or they may have a graphical user interface. But 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 design as simple as you can, because complex protocols are more likely to have flaws and they are more difficult to implement.
- In the design, you can can borrow ideas from protocols presented in lectures and exercise sessions (e.g., key exchange, secure channels, *etc.*). Also try to use the techniques we used in lectures and exercises to represent your design (e.g., message sequence diagrams, message format definitions, *etc.*)
- Be explicit and precise! Don't leave room for implementers to make their own decisions that can affect security!
- Try to use cryptographic primitives that are supported by the crypto library that you will use in your implementation. If something is not available but you really want to use it, then consult with me on how it could be securely implemented with reasonable effort.
- In the implementation, you can re-use stuff that you have already implemented in exercises or homework assignments.
- Discuss extensively within your team, as the goal of the project is to foster collaborative work. In addition, you can 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 they can help you!
- Start your implementation work in time, because implementation takes usually more time than you envision at the beginning. Also, you may want to change your design if you encounter some issues during the implementation (this is fine, design and implementation is never a linear process in practice), and re-design and re-implementation can take extra time.