Project 2: Topic 1

Murmly

Luca Campa

Department of Computer Science
Universität Innsbruck

17 April, 2025

Topics

- Introduction
- Task Description
- Learning Objective
- Dates
- Evaluation

Introduction

In the last 20 years, several chat applications were proposed (e.g. Whatsapp, Signal, Telegram, . . .). However, one characteristic that is always under debate is their security, and in particular the security of the exchanged messages. The term *E2EE* (End-to-End-Encryption) stands for a paradigm that almost all the communication platforms are incorporating into their solutions.
For example, **Signal**'s *Double Ratcheting* has become the major standard for such applications.
The idea is to make the messages exchanged between two parties unreadable by the central server through which the message is passing. Exactly, you heard correctly. There is always a central server that acts as a router for your messages and that permits the subscription to the messaging APP.

Task description

Using `Python` and the encryption systems we have discussed, develop a simple chat platform (even on command line, you don't need a GUI). Here we list the requirements for the user, the server and the cryptographic solution.

User

- Can communicate with the central server and subscribe to the platform (username, password, whatever you prefer)
- Can login to the messaging app managed by the central server
- Can see if there are other users online
- Can start a E2EE (End-to-End-Encrypted) communication with one of the users online
- Group chats are forbidden. Only one-to-one.

Task description

Central server

- Receives subscription requests from users
- Acts as a router for the messages exchanged between two users. That is: if A sends a message *m* to B, that message is going to S (Server) and then to B ($A \rightarrow S \rightarrow B$)
- The server MUST not be able to read the messages passing through it.
- It contains a database of users, their login information, and it knows even if they are online or not.

Task description

Cryptographic requirements

- Users must be authenticated to the server.
- When A starts a E2EE communication with B, they exchange a secret key (securely).
- The rest of the communication between A and B must be encrypted with one of the symmetric primitives we have seen during the lectures.
- (OPTIONAL +3 points) A good practice is to update (EFFICIENTLY, without a lot of communication overhead) the shared secret key after a certain amount of messages. How to update it to prevent that an attacker who knows one of the past keys will not be able to recover future (updated) keys? (This is usually called **Backward secrecy**)
- (OPTIONAL +3 points) How to update it to prevent that an attacker who knows one of the keys will not be able to recover past keys? (This is usually called **Forward secrecy**)

(Please, email me if you have doubts about those points)

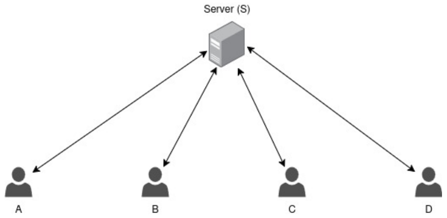
Task description

Your project must be structures in the following way:

- it must contain a `src` folder where the tool will be located
- it must contain a `README.md` file that contains the detailed usage instructions with a detailed justification of the implementation choices.
- I must be able to use *Wireshark* to intercept the communications between the involved parties (and I must not be able to understand anything, the intercepted messages should be encrypted), therefore make them communicate through common network mechanisms, e.g. TCP sockets, HTTP, . . . (please, email me if you have doubts about this point)

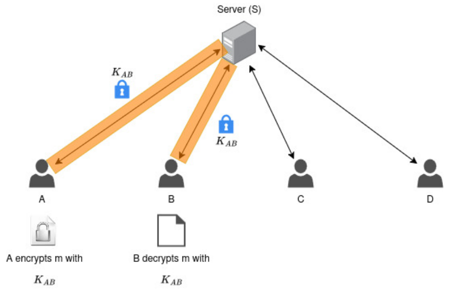
Task description

The figure shows the ideal network you will work with.



Task description

The figure shows the E2EE communication between A and B.



But. . . how did they exchange K_{AB}? You must use one of the public key cryptography mechanisms we have defined so far.

Learning objective

- Understanding how to protect the communication between two parties.
- Understanding how E2EE works.
- Understanding the usage of the cryptographic schemes we have tackled during the lecture: e.g. Diffie-Hellman and AES-GCM

Dates

- **Deadline for submission**: 11 May 2025 at 23:59
- **Oral presentation**: 15 May 2025

Evaluation

The project will count as the 20% of the final grade, in particular:

- 10% Implementation:
    - 2% if the code is running
    - 8% if the code returns correct results on all the test inputs (we will provide you with some of them, but not all)
- 10% Oral presentation: every member of the group must explain one of the parts of the project.

Bonus points (OPTIONAL items) will be taken into account for the final mark of the project. If you obtain more than 20 points, the additional points will be taken into account for the final mark of the course.
It is suggested, but not mandatory, to do the project in groups of **maximum 4** students.