Lab5
Secure communications

This lab will be part of your final grade.

Introduction

In this lab, you will design and implement a minimal secure communication protocol. The primary focus will be on creating a protocol that provides authenticated key exchange using asymmetric cryptography, followed by securing the communication channel with symmetric encryption and message authentication codes (MAC). The lab consists of two main parts: protocol design and implementation.

Instructions

Each question guides you through the process of designing and implementing your protocol. Complete each question step by step.

- ☐ 2 handouts are expected:
  - A specification of your protocol, **by the end of the last session**.
  - Your implementation, along with a report describing your final protocol and its known limitations.
- ☐ You can work in pairs.
- ☐ Language: Python
- ☐ Resource:
  - SOG-IS Crypto Working Group. *SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms*. 2023. URL: https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.3.pdf

1 Protocol Design

In this first part, you will not implement anything. You should be focusing on thinking what you need to do, how you will do it, and why you want to do it this way.

▶ Question 1. Design the Authenticated Key Exchange Protocol
Your first task is to design a protocol that allows two parties (Alice and Bob) to securely establish a shared secret. The protocol should be based on asymmetric cryptography, and allow for **mutual authentication** include the following:

1. Use asymmetric cryptography (e.g., RSA) to authenticate both parties (you must implement it yourself).
2. Allow the parties to generate or use self-signed certificates if needed (you can use OpenSSL or other external source to do so).
3. Establish a shared secret that can be used for symmetric encryption (you must implement it yourself).
4. Make sure to provide mutual authentication (both Alice and Bob authenticate each other).

Write a detailed specification of your protocol, including:

- The message sequence (who sends what and in which order).
- The cryptographic algorithms used for each step.
- How keys are generated, exchanged, and authenticated.
- Any fallback mechanisms or error handling.

▶ Question 2. Design the symmetric communication channel
After establishing a shared secret, you need to design a protocol to secure the communication channel. This involves:

1. Using the shared secret to encrypt messages (e.g., AES).
2. Ensuring message integrity and authenticity.
3. Considering replay attack prevention measures.
4. Discussing how the communication session will be terminated securely.

Document the design of your secure communication channel, including:

- Any algorithms used, and the reasoning behind your choice.
- How the session is secured and maintained.
- Measures taken to protect against known attacks (e.g., replay attacks, man-in-the-middle attacks).

Submission for Part 1: Submit your detailed protocol design after the last lab session. Your design should cover the requirements mentioned in Part 1 and clearly describe the limitations you are aware of (2-3 pages, no more than 5).

Your specification may change slightly by the end of the project, but should not be completely different.

■ 2 Protocol Implementation

◆ 2.1 Prepare your environment

This project is slightly more complex than previous labs, and you may need to install additional packages. To avoid breaking system dependencies, you will work in a *virtual environment*. Luckily, Python provides an easy way to use them:

Creation of a python virtual environment

- ☐ Open a terminal.
- ☐ Install Python and some needed dependencies:
  ```
  $ sudo apt install python3 python3-pip python3-dev python3-venv
  ```
- ☐ Go in your git repository
- ☐ Create a python virtual environment:
  ```
  $ python3 -m venv python_venv
  ```
- ☐ Activate the python environment (you will need to activate it in your terminal every time, or activate it within VS code)
- ☐ Generate the `requirement.txt` file
  ```
  $ pip freeze > requirements.txt
  ```
- ☐ Commit/push the created `requirement.txt`, but `not` the `python_venv` folder.

◆ 2.2 Description of the provided application

A template application is available on Moodle. This minimalist application provides a working example of local communication application. For simplicity, only the client sends message, while the server automatically responds "ACK" on each message.

```
usage: app.py [-h] [--server] [--client] [--debug-level {0,1,2}] [--host HOST] [--port PORT]
```

You can use it to establish a communication channel between two terminal:

1. Open a first terminal, and run
   `$ python app.py --server`
2. In a second terminal, run
   `$ python app.py --client`
3. Send messages from the client to the server, you should see messages received in both terminals.

A logging system is already implemented, that you can enable with the `--debug-level {0,1,2}` argument. Try it to see what information are printed, and feel free to add additional messages for debugging.

As you may notice, this application is currently insecure. It's your job to implement a secure key exchange, and a way to secure the following communications!

◆ 2.3 Implementation

▶ Question 1. Implement the Authenticated Key Exchange
Implement the asymmetric part of your protocol in Python. You may provide basic functionalities like certificate generation (self-signed certificates) using existing libraries.

▶ Question 2. Implement the secure communication channel
Using the shared secret obtained from the authenticated key exchange, implement a secure communication channel. You may use existing Python packages (such as `pycryptodome`) for these operations. The focus should be on correctly applying cryptographic primitives to secure the communication.

▶ Question 3. Integrate with the provided application code
The application code provided contains a skeleton for a secure communication application. Your task is to:

1. Identify where to integrate your protocol implementation (key exchange and secure channel functions).
2. Use the provided API definitions to ensure consistent communication.
3. Test your integration using the sample client and server provided.

Submission for Part 2: Submit the complete Python code for your implementation, along with an updated protocol specification. The final specification should reflect any changes made during implementation and highlight known limitations or potential improvements.

Before submitting, make sure that your project includes an up-to-date `requirements.txt` that lists all package requirements for running the application. You can generate it using the following command in a terminal, with your environment activated.

■ Deliverables

Submit the following:

- **Part 1:** Protocol design document (2-3 pages).
- **Part 2:** Python code for your implementation, final protocol specification (2-3 pages), and testing results.