18-631: INTRODUCTION TO INFORMATION SECURITY

HOMEWORK 2 – FALL 2024

1. Implement a Simplified TLS Handshake Simulation

Objective: This assignment is designed to help you understand the fundamental steps involved in establishing a secure communication channel using the TLS (Transport Layer Security) protocol. By manually implementing the TLS handshake, you will gain deeper insight into how protocol version negotiation, cipher suite selection, and key exchange mechanisms contribute to securing communications. This exercise will also enhance your comprehension of the cryptographic principles that underpin the TLS protocol, preparing you for more advanced topics in network security and encryption.

1.0. Task: Simplified TLS Handshake Simulation

1.0.1. Task Details:

You are required to implement a simplified version of the TLS handshake process using a programming language of your choice: C++, Python, or Java. Your program should be interactive, guiding the user through the following steps:

A. Protocol Version Negotiation:

- Allow the user to input the supported TLS versions for both a client and a server.
- Your program should identify and select the highest common TLS version supported by both parties.

B. Cipher Suite Selection:

- Allow the user to input the supported cipher suites for both the client and the server.
- Your program should select the most secure cipher suite that is mutually supported by both parties.

C. Diffie-Hellman Key Exchange:

- Simulate the key exchange process using a basic Diffie-Hellman method.
- The program should compute a shared secret key between the client and server without using external cryptographic libraries.

1.0.2. User Interaction:

- ***Menu System:*** Implement a menu-based system that allows users to navigate through the steps of the TLS handshake. The menu should include pre-defined options for both supported TLS versions and cipher suites. The options should include:
  - ***At least one scenario where the handshake succeeds*** (e.g., the client and server share a common protocol version and cipher suite).

- ***At least one scenario where the handshake fails*** (e.g., no common protocol version or cipher suite is found, leading to an error).

This approach will allow users to see both successful and failed handshake processes, helping them understand the importance of protocol compatibility in secure communications.

- ***Guidance:*** Provide clear prompts for user input at each step, ensuring that users can easily understand and follow the process.
- ***Output****:* After completing the handshake, your program should display:
  - The negotiated TLS version.
  - The selected cipher suite.
  - The established shared secret key.
  - Any error messages that occur during the handshake process, explaining why the handshake failed.

1.0.3. Example Menu Options:

- ***Supported TLS Versions:***
  1. TLS 1.0, TLS 1.1, TLS 1.2, TLS 1.3 (Client)
  2. TLS 1.2, TLS 1.3 (Server)
  3. TLS 1.0, TLS 1.1 (Server - expected to cause a failure)
- ***Supported Cipher Suites:***
  1. AES-128-GCM, AES-256-GCM, CHACHA20-POLY1305 (Client)
  2. AES-256-GCM, CHACHA20-POLY1305 (Server)
  3. DES-CBC3-SHA (Server - expected to cause a failure if not supported by the client)

1.1. Implementation Guidelines:

- ***Language Choice*****:** You may implement this task using C++, Python, or Java. Ensure that your code is well-documented, with comments explaining the purpose of each major section and function.
- ***Modular Code****:* Structure your code to separate the different components of the handshake (e.g., version negotiation, cipher suite selection, key exchange) into distinct functions or classes.
- ***Error Handling:*** Include basic error handling to manage invalid user inputs (e.g., unsupported versions or cipher suites).

1.2. Submission Requirements:

- ***Source Code:*** Submit your source code files (.cpp, .py, .java) along with any necessary build instructions.
- ***Report:*** Include a PDF file explaining how to compile and run your program, along with a description of what your approach achieves and snapshot showing your menus and both results (i.e., Successful and failed handshake) long with the interpretation of what results tells.
- ***Demo:*** Your program should be easy to demonstrate, with clear instructions provided in the REPORT for setting up and running the simulation.

1.3. Grading Criteria:

- ***Correctness:*** Does your implementation correctly simulate the TLS handshake process? Are the correct protocol version and cipher suite selected?
- *User Interaction:* Is the program user-friendly? Does it guide the user effectively through the handshake process?
- ***Code Quality:*** Is your code well-organized, modular, and well-documented? Have you included appropriate error handling?
- ***Originality:*** Does your implementation demonstrate a thorough understanding of the TLS handshake and cryptographic principles?

1.4. Academic Integrity:

This assignment will be submitted through **Gradescope**, where your code will be checked for plagiarism and code similarity. You are expected to adhere to the highest standards of academic integrity. Be mindful when using generative AI tools, as submitting AI-generated code without proper understanding or attribution will be considered a violation of academic integrity policies. Ensure that your work is original, and that you fully understand and can explain every part of the code you submit.

Happy Coding