CSE 4057 Programming Assignment

Secure Device-to-Server Communication

Due: 03.06.2025, 23:59

In this programming assignment, you will implement a **simplified secure communication protocol** between a **device** and a **server**. The device will send two types of data: **text messages** and **images** (both simulated). The server will send back relevant **acknowledgement messages**.

Your implementation must provide secure message exchange using various cryptographic primitives.

Prerequisite: It is assumed that you have basic knowledge on socket programming.

Required Security Features:

1) Public Key Generation and Certification

- The device and the server each generate a **public/private key pair** (using either **RSA** or **ECC**).
- Implement a simple **Certificate Authority (CA)** that:
  - Verifies identities.
  - Issues and signs certificates using its own private key.
  - Certificates must be exchanged and validated during communication.

2) Handshaking and Key Generation

- Both parties initiate a handshake by exchanging **"Hello"** messages.
- Each "Hello" message must include:
  - The sender’s **certificate**.
  - A randomly generated **nonce** (to ensure freshness).
- After exchanging "Hello" messages:
  - Establish a **shared master secret** using either:
    - RSA-encrypted random secrets, or
    - ECDH (Elliptic Curve Diffie-Hellman) key exchange.
- From the master secret, derive:
  - **Two symmetric encryption keys** (one for each direction).
  - **Two MAC keys**.
  - **One IV (Initialization Vector)** for symmetric encryption.

3) End-to-End Encrypted Text Communication

- For every text message:
    - Generate a **Message Authentication Code (MAC)** using **HMAC** with the **SHA-256** hash algorithm and the appropriate MAC key.
    - **Encrypt** the text message **along with its MAC** using the derived symmetric encryption key (AES recommended).
- Upon receiving a message:
    - **Decrypt** the content.
    - **Verify** the MAC.
- All received plaintext messages should:
    - Be **printed** on the screen.
    - Be **written** to a **log file** (together with timestamps).

4) End-to-End Encrypted Image Delivery

- For each image:
    - The device generates a **digital signature** using its private key.
    - The image and the signature are **encrypted together** and sent to the server.
- Upon receipt:
    - The server **decrypts** the image and the signature.
    - **Verifies** the digital signature.
    - **Saves** the verified image to local storage.
    - Sends back an **acknowledgement message**, also **encrypted**.

5) (Bonus) Key Update Mechanism

- Implement a **key update mechanism** to enhance security (forward secrecy).
- After a certain number of messages or after a timer expires:
    - Derive a **new set of encryption and MAC keys** (e.g., using HKDF).
    - Ensure that **compromised old keys** cannot decrypt new or past messages.
- Clearly describe your **key update method** in your README file.

6) (Bonus) End-to-End Encrypted Video Transfer

- Extend the system to support **secure, encrypted video transfer** from the device to the server.
- Bonus points will be awarded based on:
    - Correct encryption and transmission.
    - Signature verification.
    - Performance and reliability.

7) (Bonus) Real Device-to-Server Communication

- In this project, you are required to **simulate** text messages and images. For text messages, you may use **randomly generated values**. For images, you may use any **stored image files** for transmission. However, **if you implement a real-world scenario** where a **device** (such as a **Raspberry Pi** or any device equipped with a **camera and sensors**) sends **actual sensor readings** and **real-time captured images** to a server, you will receive **bonus points**.

Additional Requirements:

- **Logging**:
    - Print **all sent and received messages** (plaintext or ciphertext) to a **log file**.
    - Include timestamps and sender/receiver information.
- **Communication Protocol**:
    - You may design your own simple protocol for sending/receiving **text**, **images**, and **acknowledgements**.
    - The protocol should be **clearly documented** in the README.
- **Socket Programming**:
    - You must use **network sockets** (TCP recommended) to connect the device and server.
    - You can:
        - Use **two different machines** (preferred), or
        - Use **localhost** on a single machine (acceptable for testing).

- **Language & Libraries**:
    - You may use **any high-level programming language** (Python, Java, C++).
    - You may use cryptographic libraries such as:
        - Python: cryptography, PyCryptodome
        - Java: javax.crypto
    - **You may not use SSL/TLS libraries** like OpenSSL.
You must implement the handshake and encryption mechanisms yourself.
- **Group Work**:
    - Allowed for up to **three students per group**.
    - You must clearly declare:
        - (i) **How you communicated and coordinated** as a team.
        - (ii) **Detailed division of labor** (who implemented what parts).
        - (iii) **How you integrated and merged your codes**.

    - This information must be **included in your README** file.

Security Holes:

While you will implement important security features, **security holes may still exist**.

- Try to **identify potential vulnerabilities** in your system (e.g., MITM attacks, weak randomness, replay attacks).
- Offer **possible solutions** and **countermeasures**.
- **(Bonus)** You may implement additional security enhancements for extra points.

Submission Details:

- Submit a **zip file** containing:
    - **Source Code** (well-structured and well-commented).
    - **README** file, including:
        - A clear description of your design and implementation.
        - Division of labor (for group work).
        - A description of identified security holes and proposed countermeasures.
    - The zip file name must **include the full names of all group members**.
- Submit your project **via Google Classroom**. Please upload a single file per group.
- **Plagiarism Warning**:
    - Your codes will be checked for plagiarism.
    - **Any detected plagiarism will be severely penalized.**