Akash Kushwaha 2021514
Suyash Kumar 2021293

Network Security - Assignment 3
RSA-based PKDA
Report

1. Introduction

This project implements a Public Key Distribution Authority (PKDA) using RSA encryption to securely distribute public keys to clients. Two clients communicate confidentially after securely obtaining each other's public key via the PKDA.

2. System Overview

The system consists of three main components:

- **PKDA Server (pkda.py)**: This server acts as the central authority for managing and distributing public keys. It listens for client requests, decrypts incoming messages, retrieves the requested public key, and sends it securely back to the requesting client.
- **Client 1 (client1.py)**: This client initiates a request to the PKDA to obtain the public key of another client. After retrieving the key, it engages in encrypted communication with the other client.
- **Client 0 (client0.py)**: Similar to Client 1, this client requests the PKDA for a public key and then uses it to establish secure communication with Client 1.

3. Working Flow

1. pkda.py (Public Key Distribution Authority - PKDA)

This file implements the PKDA server, which is responsible for securely distributing public keys to clients.

- Loads PKDA Credentials: Reads the private (d) and public (e, n) keys from creds/pkda.json.
- Establishes a Server Socket: Listens for incoming client connections on a predefined IP (HOST) and port (PORT).
- Handles Client Requests:
    - Accepts incoming client connections.
    - Receives an encrypted request(encrypted by the public key of PKDA) for a public key of another client.
    - Decrypts the request using the PKDA's private key.
    - Retrieves the requested client's public key from the respective credential file (clientX.json).
    - Constructs a response containing the requested public key and a nonce.
    - Sends the response back to the client (currently unencrypted, though it should be encrypted with the initiator's public key).
    - Closes the client connection.

---

2. client1.py (Client 1 Implementation)

This file represents Client 1, which interacts with the PKDA and another client (Client 0) for secure communication.

- Loads Credentials: Reads the private (d, n) and public (e, n) keys from client1.json, and also loads PKDA's public key from pkda.json.
- Requests Client 0's Public Key from PKDA:
    - Creates a request dictionary containing:
        - Request type
        - Initiator (Client 1)
        - Requested client (Client 0)
        - Timestamp (ignored for now)
        - Nonce
    - Encrypts the request with PKDA’s public key.
    - Sends the encrypted request to PKDA.
    - Receives and decrypts the response from PKDA, extracting Client 0's public key.
- Establishes Secure Communication with Client 0:
    - Listens on PORT_CLIENT1 for incoming connections.
    - Waits for an encrypted message from Client 0.
    - Decrypts the received message using Client 1's private key.
    - Displays the message and allows the user to input a response.
    - Encrypts the response using Client 0’s public key and sends it.
    - Continues this communication loop until "exit" is entered.

---

3. client0.py (Client 0 Implementation)

This file implements Client 0, which interacts with PKDA to retrieve Client 1’s public key and then communicates securely.

- Loads Credentials: Reads Client 0’s private (d, n) and public (e, n) keys from client0.json, and also loads PKDA’s public key.
- Requests Client 1’s Public Key from PKDA:
  - Creates and encrypts a request to PKDA for Client 1’s public key.
  - Sends the request to PKDA and receives the response.
  - Decrypts the response to retrieve Client 1’s public key.
- Initiates Secure Communication with Client 1:
  - Connects to Client 1’s listening port (PORT_CLIENT1).
  - Reads user input, encrypts it using Client 1’s public key, and sends it.
  - Waits for an encrypted response, decrypts it using Client 0’s private key, and displays it.
  - Repeats this process until "exit" is entered.

---

4. rsa.py (RSA Encryption & Decryption Library)

This file implements basic RSA encryption and decryption functions.

- encrypt(message, e, n):
  - Converts the message into bytes.
  - Encrypts the message using the RSA formula: ciphertext = (message^e) % n.
- decrypt(encrypted, d, n):
  - Decrypts the message using: plaintext = (ciphertext^d) % n.
  - Converts the decrypted integer back into a string.

---

5. server_creds.py (Server Configuration)

Defines constants for network communication.

- HOST = "127.0.0.1" → Localhost IP
- PORT = 6666 → PKDA listens on this port.
- PORT_CLIENT0 = 1234 → Dedicated port for Client 0.
- PORT_CLIENT1 = 1233 → Dedicated port for Client 1.

---

6. gen_creds.py (Key Generation Script)

This script generates RSA key pairs for PKDA, Client 0, and Client 1.

- generate_primes() → Uses OpenSSL to generate six 2048-bit prime numbers.
- generatekeys(p, q) → Uses two prime numbers to compute RSA key components
- Stores Generated Keys:
    - Saves each entity's credentials (n, e, d) in client0.json, client1.json, and pkda.json.

4. Implementation Details

- **Encryption**:
    - RSA encryption is used for securing communication between clients and the PKDA.
    - Each message is encrypted with the recipient's public key, ensuring confidentiality.
- **Authentication**:
    - The PKDA authenticates requests and ensures that only legitimate clients receive public keys.
    - The response includes the original nonce to confirm integrity and prevent replay attacks.
- **Communication**:
    - The system uses socket programming for establishing and maintaining TCP connections.
    - The PKDA runs on a dedicated port, listening for incoming requests from clients.
    - Clients use separate ports to receive encrypted messages from each other.
- **Security Measures**:
    - Nonces and timestamps are used to ensure freshness and prevent replay attacks.
    - Messages are structured to include request type, initiator, responder, and security parameters.

5. Running the System

To execute the system, follow these steps:

1. **Start the PKDA Server**:
    - Open a terminal and run python pkda.py.
    - The server will start listening for client requests.
2. **Run Client 1**:
    - Open a second terminal and run python client1.py.
    - Client 1 will request the public key of Client 0 from the PKDA.
3. **Run Client 0**:
    - Open a third terminal and run python client0.py.

    - Client 0 will request the public key of Client 1 and then start secure communication.

4. **Exchange Messages:**
    - Once both clients have obtained each other's public key, they can exchange encrypted messages.
    - Client 0 sends "Hi1", "Hi2", "Hi3", and Client 1 responds with "Got-it1", "Got-it2", "Got-it3".
    - Messages remain confidential as they are encrypted with RSA.