Certificate Authority and RSA-Based Secure Messaging

This project implements a secure communication system utilizing a Certificate Authority (CA) and RSA encryption. It allows clients to exchange messages securely through RSA public-private key pairs, with certificates managed and verified by a central CA. The system leverages Python and ZeroMQ for inter-process communications.

Table of Contents

Overview

This secure messaging project implements a centralized Certification Authority to issue and manage RSA-based digital certificates for clients. Each client can:

- Generate and register its RSA key pair with the CA.
- Request certificates of other clients securely from the CA.
- Verify the authenticity of certificates provided by the CA.
- Encrypt messages with RSA to securely communicate with other clients.

Project Structure

```
- certification_authority.py   # CA server for client registration and certificate management
- client.py                    # Client-side application for messaging and certificate handling
- rsa_utils.py                 # RSA cryptographic functions and utilities
- certificate_logs.txt         # Log file storing issued client certificates
```

Features

- Certification Authority (CA): Handles registration, certificate issuance, and validation.
- RSA Key Generation: Clients and CA generate secure RSA key pairs.
- Encrypted Communication: Secure messaging between authenticated clients.
- Certificate Verification: Clients verify certificates cryptographically.
- Message Integrity: Verification through public-key encryption methods.

Technologies Used

- Python 3: Core programming language for implementation.
- ZeroMQ: Lightweight messaging library for socket communication.
- JSON: Data serialization and storage.
- RSA: Cryptographic algorithm for encryption and digital signatures.

Setup and Installation

1. Clone the Repository

```
git clone https://github.com/arpan21020/Network-Security.git
cd Network-Security
```

2. Install Dependencies

Ensure Python 3 and pip are installed. Install the necessary Python packages:

```
pip install pyzmq
```

Running the Application

Step 1: Start the Certification Authority (CA)

Launch the CA server first:

```
python certification_authority.py
```

The CA runs on `tcp://*:5555` by default.

Step 2: Start Clients

Each client requires a unique identifier (`client_id`):

```
python client.py <client_id>
```

Example to start two clients:

```
python client.py 101
python client.py 102
```

Usage

Clients have a simple interactive menu:

```
*********Menu*********
1) Request certificate
2) Send message to some client
3) Check for incoming messages
4) Exit
```

- Option `1` fetches and verifies another client's certificate.
- Option `2` sends an encrypted message to a client.
- Option `3` checks incoming encrypted messages.

Detailed Workflow

Client Registration:

- Clients generate RSA key pairs.
- Clients register public keys with the CA.
- CA creates certificates, encrypting client information with its private key.

Certificate Exchange:

- Clients request other clients' certificates from the CA.
- Received certificates are decrypted and verified using CA's public key.

Secure Communication:

- Sender encrypts messages using the recipient's public key from their certificate.
- Recipient decrypts messages using their own private key.

Certificate Format

Certificates issued by the CA have two main sections:

```
{
    "plain_data": {
        "client_id": "101",
        "public_key": [1998397, 3910199],
        "port": 6101,
        "timestamp": 1743351235,
        "duration": 3600,
        "ca_id": "MAIN_CA_2025"
    },
    "encrypted_data": [2144012, 1057702, 1010800, ...]
}
```

- `plain_data`: Client details and RSA public key.
- `encrypted_data`: RSA-encrypted string of `plain_data` using CA's private key.

File Descriptions

File	Description
`certification_authority.py`	Manages client registration, certificate issuance, and verification.
`client.py`	Allows users to register, fetch certificates, send and receive encrypted messages.
`rsa_utils.py`	RSA encryption utilities including prime generation, key generation, and encryption/decryption functions.
`certificate_logs.txt`	Logs all client certificates issued by the CA.
