RSA-Based Public-Key Certification Authority (CA) System Design Documentation

Authors: Arav Amawate(2022091) and Angadjeet Singh(2022071)
Date: 30-03-2025

---

1. Introduction

This document describes the design and implementation of an RSA-based public-key Certification Authority (CA) system. The system is developed to support a secure communication environment where clients can obtain their own and peers' public-key certificates from a central CA. These certificates enable secure exchange of messages using RSA encryption and decryption techniques. This document provides an overview of the system architecture, components, protocols, and implementation details.

---

2. System Overview

The system comprises three main components:

- **Certification Authority (CA):** A server that registers client public keys, issues digitally signed certificates, and validates certificates upon request.
- **Clients (e.g., Client A and Client B):** End-user applications that register with the CA, request and retrieve certificates, and exchange secure messages.
- **RSA Cryptography Module:** A module that handles the generation of RSA key pairs, encryption, and decryption of messages using standard RSA algorithms.

The design follows a public-key infrastructure (PKI) model where the CA acts as the trusted third party. The CA digitally signs certificates using its private key, ensuring integrity and authenticity. Clients use these certificates to exchange messages that are encrypted with RSA, ensuring confidentiality.

---

3. System Components

3.1 Certification Authority (CA)

The CA is responsible for:

- **Client Registration:** Accepting and storing client public keys.
- **Certificate Issuance:** Creating certificates that include client ID, public key, issuance time, validity duration, and a CA digital signature.
- **Certificate Verification:** Validating a certificate's authenticity by checking the signature and ensuring the certificate has not expired.

3.2 Client Applications

Each client (e.g., A and B) is equipped with:

- **Key Pair Generation:** Each client already possesses an RSA public-private key pair.
- **Certificate Request:** The client requests the CA to sign a certificate for its public key.
- **Peer Communication:** Once in possession of a valid certificate for a peer, clients exchange encrypted messages. For instance, Client A encrypts messages (e.g., "Hello1", "Hello2", "Hello3") with Client B's public key, and B replies with acknowledgments ("ACK1", "ACK2", "ACK3").

3.3 RSA Cryptography Module

The RSA module is central to the security of the system. Its responsibilities include:

- **Key Generation:** Random generation of two large prime numbers (p and q), computation of modulus.

Sure, here is the expression written in proper mathematical notation:

Let

$$n = p \times q$$

Then Euler's totient function is:

$$\phi(n)$$

If p and q are distinct prime numbers, then:

$$\phi(n) = \phi(p \times q) = (p-1)(q-1)$$

- **Public/Private Key Pair Computation:**

1. **Choose an encryption exponent**

$$e \quad \text{such that} \quad 1 < e < \phi(n) \quad \text{and} \quad \gcd(e, \phi(n)) = 1$$

(i.e., e is coprime to $\phi(n)$)

2. **Compute the decryption exponent**

$$d \equiv e^{-1} \mod \phi(n)$$

(i.e., d is the modular inverse of e modulo $\phi(n)$, satisfying $ed \equiv 1 \mod \phi(n)$)

The public key is (e, n), and the private key is (d, n).

- **Encryption:** The module encrypts messages block by block, using RSA with appropriate padding (PKCS#1-like).
- **Decryption:** The module decrypts each block of the encrypted message using the corresponding private key and then un-pads the message to recover the original plaintext.

---

4. Communication Protocol and Data Flow

4.1 Registration Phase

1. **Client Registration:**
    - A client connects to the CA server and sends its client ID and public key.
    - The CA registers the client and sends back a confirmation along with the CA's public key.

4.2 Certificate Issuance and Verification

1. **Certificate Request:**
    - The client sends a certificate signing request.
    - The CA creates a certificate containing client ID, public key, issuance time, validity duration, and a digital signature.
2. **Certificate Verification:**
    - Clients can request the CA to validate a certificate.
    - The CA decrypts the signature using its private key and compares it with the expected hash of the certificate data. If they match and the certificate is still valid (not expired), the certificate is considered authentic.

4.3 Secure Message Exchange

1. **Obtaining Peer Certificates:**
    - Clients request the CA for a peer's certificate.
    - Once a certificate is validated, the client extracts the peer's public key.
2. **Encryption and Decryption:**
    - Client A encrypts messages using Client B's public key.
    - Client B decrypts the messages with its private key and vice versa.

This protocol ensures that all messages are confidential and only decipherable by the intended recipient.

---

5. Implementation Details

5.1 CA Server

- **Socket Programming:**
  The CA server uses Python's socket module to handle TCP connections. It listens for incoming client requests, processes them (registration, certificate signing, certificate retrieval, verification), and sends back the appropriate responses using Python's pickle module for serialization.

- **Certificate Structure:**
  The certificate is represented as a Python dictionary containing:

  - `client_id`
  - `public_key`
  - `issue_time`
  - `duration`
  - `signature` (digital signature created by encrypting a SHA-256 hash of the certificate data with the CA's private key)

5.2 Client Application

- **Certificate Operations:**
  Clients perform the registration, certificate request, and certificate validation by sending specific action messages to the CA. The client code encapsulates these functions into a class, enabling clean and modular interaction.

- **Message Handling:**
  Secure message exchange is facilitated by methods that encrypt messages with the recipient's public key and decrypt incoming messages with the client's private key.

5.3 RSA Module

- **Key Generation:**
  The RSA module implements key generation using Python's `Crypto.Util.number` for generating large prime numbers and computing the modulus and totient.

- **Padding and Encoding:**
  For secure encryption, padding is applied to each block (using PKCS#1-like padding), and encrypted blocks are converted to a base64 string for transmission.

---

6. Testing and Results

The system was tested using two client applications:

- **Client A** registered and requested its certificate from the CA.

- **Client B** similarly registered with the CA.

- Client A retrieved Client B's certificate and used the public key contained within to encrypt and send three test messages ("Hello1", "Hello2", "Hello3").

- Client B decrypted these messages and responded with acknowledgment messages ("ACK1", "ACK2", "ACK3").

---

7. Conclusion and Future Work

This document presented an RSA-based public-key Certification Authority system that allows clients to securely obtain certificates and exchange encrypted messages. The design leverages robust cryptographic techniques to ensure data integrity and confidentiality.

Future Enhancements:

- **Robust Error Handling:** Implement more granular error checks and logging mechanisms.

- **Scalability:** Optimize the CA for handling multiple simultaneous requests.

- **Enhanced Security:** Incorporate additional security features (e.g., certificate revocation, more advanced padding schemes) to strengthen the system against potential attacks.

- **User Interface:** Develop a graphical interface for ease of use and better user interaction.