RSA-based Public-key Certification Authority (CA) System Documentation

Deepak Thappa, 2021319
Madhav Krishan Garg, 2021333

1. Introduction

This document describes the implementation of an RSA-based Public-key Certification Authority (CA) system as part of Project 0 for the Network Security course. The system consists of a central Certification Authority and multiple clients that can request and verify digital certificates to securely exchange messages.

2. System Architecture

2.1 Components

1. **Certification Authority (CA)**:
   - Generates its own RSA key pair
   - Maintains a registry of client public keys
   - Issues digital certificates for registered clients
   - Signs certificates with its private key
2. **Clients**:
   - Generate their own RSA key pairs
   - Can request certificates from the CA
   - Can verify certificates using the CA's public key
   - Can exchange encrypted messages once certificates are verified

2.2 Certificate Format

The certificate format follows the specification from the project description:

Copy
```
CERT_A = [(ID_A, PU_A, T_A, DUR_A, ID_CA) || ENC_PR-CA(ID_A, PU_A, T_A, DUR_A, ID_CA)]
```

In our implementation, this is represented as:

Copy
```
"client_id|public_key_n|public_key_e|timestamp|duration|ca_id||signature"
```

3. Implementation Details

3.1 Key Components

`custom_rsa.py`

This module provides all the cryptographic operations needed for the system:

- Prime number generation using the Miller-Rabin primality test
- RSA key pair generation
- Encryption and decryption functions
- Digital signature creation and verification

`ca_system.py`

This module contains the main system implementation with:

- `CertificationAuthority` class
- `Client` class
- Main demonstration logic

3.2 Key Algorithms

1. **Key Generation**:
    - Generates two large prime numbers (p and q)
    - Computes $n = p * q$ and $\varphi(n) = (p-1)(q-1)$
    - Chooses public exponent $e = 65537$
    - Computes private exponent $d = e^{-1} \bmod \varphi(n)$
2. **Certificate Creation**:
    - Collects client information (ID, public key)
    - Adds timestamp and validity duration
    - Creates a signature of this data using CA's private key
    - Combines data and signature into certificate format
3. **Message Exchange**:
    - Sender encrypts message with recipient's public key
    - Recipient decrypts with their private key
    - Includes acknowledgment mechanism

4. Sample Execution Flow

1. CA initializes and generates its key pair
2. Two clients (ClientA and ClientB) initialize with their own key pairs

3. Clients register their public keys with the CA
4. Each client requests its certificate from the CA
5. Clients exchange and verify each other's certificates
6. ClientA sends three test messages to ClientB
7. ClientB responds with acknowledgements for each message

Sample output:

```
Certificate verification successful!

Client A sends: Hello1
Client B receives: Hello1
Client B sends: ACK for Hello1
Client A receives: ACK for Hello1

Client A sends: Hello2
Client B receives: Hello2
Client B sends: ACK for Hello2
Client A receives: ACK for Hello2

Client A sends: Hello3
Client B receives: Hello3
Client B sends: ACK for Hello3
Client A receives: ACK for Hello3
```

5. Security Considerations

1. **Key Management**:
    - Each entity generates its key pair
    - Private keys never leave their owners
    - CA's public key is pre-shared with clients
2. **Certificate Validation**:
    - Signature verification ensures certificate authenticity
    - Timestamp and duration prevent replay attacks
3. **Message Confidentiality**:
    - All messages are encrypted with the recipient's public key
    - Only the intended recipient can decrypt messages