Network Security Assignment - 03
Project - Zero

Ketan Mohan Garg (2022248)
Keshav Bindlish (2022246)

Introduction

This project demonstrates a **custom RSA-based Certificate Authority (CA) system** for secure communication between two clients. The CA issues signed certificates to authenticate public keys, ensuring trust between **Client A and Client B**. Both clients verify these certificates before exchanging encrypted messages. Client A sends **three test messages ("Hello1", "Hello2", "Hello3")**, and Client B responds with **acknowledgments ("ACK1", "ACK2", "ACK3")**. This ensures a secure and authenticated communication channel using RSA encryption and digital signatures.

Assumptions:

a. that clients already (somehow) know their own [private-key, public-key], but do not have their own or certificates or that of others.
b. that clients already (somehow) know the public key of the certification authority.
c. that CA has the public keys of all the clients.

CODE EXPLANATION

This Python program makes a Certificate Authority (CA) issuing signed certificates for two clients, A and B. It uses modular arithmetic for RSA-like key generation, signing, and encryption. The CA generates and signs certificates with its private key, and clients verify them using the CA's public key. Clients then securely exchange encrypted messages using each other's public keys. The program demonstrates certificate issuance, signature verification, encryption, decryption, and acknowledgment-based secure communication.

Working of Each Function

1. egcd(a, b): Computes the greatest common divisor (GCD) of two numbers using the extended Euclidean algorithm, returning the GCD along with coefficients for solving modular inverses.
2. modinv(a, m): Computes the modular inverse of a modulo m using the extended Euclidean algorithm, ensuring a and m are coprime; raises an exception otherwise.
3. sign_certificate(client_id, public_key): Creates a certificate by signing client data using the CA's private key, generating a time stamped message and its cryptographic signature via modular exponentiation.

4. request_certificate(client_id): Retrieves a signed certificate for a client by verifying its identity and generating a signed public key from the CA's stored values.
5. Main verification logic: Clients validate signed certificates by checking if the decrypted signature matches the expected message hash using the CA's public key.
6. Communication logic: Client A encrypts messages using B's public key, B decrypts using its private key, then responds with an acknowledgment encrypted with A's public key.

Outputs: