RSA report:

-Sreekar Reddy(2021318)
-Satwik(2021276)

The RSA-based Public-key Certification Authority (CA) algorithm involves several steps to establish a secure communication infrastructure using RSA encryption for key generation, certificate generation, and certificate verification. Here's a detailed description of the algorithm:

1. Key Generation:

The CA generates its public and private RSA keys using the generate_keys(bits)` function. These keys are crucial for certificate signing and verification. The public key (`PU_CA`) will be distributed to clients for certificate verification, while the private key (`PR_CA`) will be used by the CA to sign certificates.

2. Client Key Generation:

Clients generate their own RSA key pairs using the `generate_keys(bits)` function. These key pairs consist of a public key (`PU`) and a private key, which will be used for secure communication.

3. Certificate Generation:

When a client requests a certificate from the CA, the client provides its identity (`IDA`), public key (`PU`), timestamp (`TA`), duration (`DURA`), and the CA's identity (`IDCA`).
The client generates a certificate request and sends it to the CA.
The CA verifies the client's identity and other details.
The CA generates a certificate containing the client's information along with a digital signature. The signature is created by encrypting the certificate contents using the CA's private key (`PR_CA`), ensuring the integrity and authenticity of the certificate.
The certificate is then sent back to the client.

4. Certificate Verification:

When a client receives a certificate, it verifies the certificate's authenticity before establishing secure communication with other clients.

The client extracts the information from the certificate and the digital signature.

The client decrypts the digital signature using the CA's public key (`PU_CA`) to obtain the original certificate information.

If the decrypted information matches the certificate contents, the certificate is considered valid and trustworthy. Otherwise, it's rejected.

5. Secure Communication:

Once the certificates are verified, clients can securely communicate with each other using RSA encryption.

Clients use each other's public keys obtained from the certificates to encrypt messages intended for each other.

The recipients use their private keys to decrypt the messages.

6. Certificate Revocation:

If a client's private key is compromised or if the client's information changes, the CA may revoke the client's certificate.

Revocation information can be maintained by the CA, and clients can periodically check for certificate revocations.

7. Renewal and Expiration:

Certificates have a limited validity period (`DURA`) after which they expire.

Clients may renew their certificates by requesting new ones from the CA before expiration.

This RSA-based Public-key Certification Authority algorithm provides a robust mechanism for secure communication over public networks by ensuring the authenticity and integrity of communication parties through digital certificates signed by a trusted CA.

Code explanation:

1.The code uses the libraries random, ast and datetime and timedelta :

Random- Used for generating random numbers

Ast-provides abstract syntax trees for python code, used here for safely evaluating strings as Python expressions.

Datetime and timedelta- used for handling dates and times

2.There are helper functions for RSA which are :

gcd(a,b) - computes the greatest common divisor (GCD) of two numbers using the euclidean algorithm.

modinv(a,m)- computes the modular multiplicative inverse of a modulo m using the extended euclidean algorithm

is_prime(n) - checks if a number n is prime by testing divisibility from 2 to the square root of n

3.RSA key Generation Functions :

generate_rsa_keys(p,q):

Generates RSA public and private keys given two prime numbers p and q.

Calculates n as the modulus phi as euler's totient function and generates a random exponent e such that 1< e < phi and gdc(phi,e) = 1

Computes the corresponding private exponent d as the modular multiplicative inverse of e modulo phi

Returns the public key (e,n) and private (d,n)

4.RSA Encryption and Decryption Functions:

rsa_encrypt(private_or_public_key, plaintext): Encrypts plaintext using RSA with a given private or public key.

rsa_decrypt(private_or_public_key, ciphertext): Decrypts ciphertext using RSA with a given private or public key.

Certificate Authority (CA) Class:

__init__(self, p, q): Initializes the Certificate Authority with RSA keys and an empty dictionary for certificates.

issue_certificate(self, client_id, public_key): Issues a certificate for a client with a specified ID and public key.

give_key(self): Returns the CA's public key.

get_certificate(self, client_id): Retrieves a certificate for a client.

Client Class:

__init__(self, ca, client_id, p, q): Initializes a client with a specified ID, Certificate Authority, and RSA keys.

request_certificate(self, other_id): Requests a certificate for another client from the CA.

request_public_key(self): Requests the public key of another client from the CA.

send_message(self, to_client_id, message): Encrypts and sends a message to another client.

receive_message(self, to_client_id, message, ack_msg): Decrypts a received message and sends an acknowledgment.

recv_ack(self, msg): Receives and decrypts an acknowledgment message.

Main Section:

Checks if the script is being run directly (`if __name__ == '__main__':`).

Prompts the user to input prime numbers for the CA and the clients.

Creates a Certificate Authority and two clients (Alice and Bob).

Requests certificates and public keys for Alice and Bob from the CA.
Simulates communication between Alice and Bob by sending messages back and forth.

This code simulates a basic Public Key Infrastructure (PKI) system using RSA encryption for secure communication between clients (Alice and Bob) with the help of a Certificate Authority.