## 1. System Architecture Design

Client-Server Model:

Server: Acts as the CA, handling key registration, certificate issuance, and possibly certificate revocation. This server will maintain a database or a secure store of public keys and corresponding certificates.

Clients: Users (like Alice and Bob) who will register with the CA to obtain their certificates and then use these certificates to authenticate themselves and encrypt communications with other clients.

## 2. Server as a Certificate Authority

Key and Certificate Management:

Registration and Key Submission: Clients generate their key pairs locally and submit their public keys to the server for certification.

Certificate Issuance: The server signs the public keys with its own private key, creating certificates which are then sent back to the clients.

Certificate Storage: The server maintains a list or database of all issued certificates and their corresponding public keys, which can be referenced for authentication and verification during communications.

## 3. Client Implementation

Initial Setup:

Key Pair Generation: Each client generates an RSA key pair upon their initial setup.

Register with the CA: Submit the public key to the server and receive a certificate.

Communication with Other Clients:

Certificate Exchange: Before secure communication can start, clients exchange certificates.

Verification: Clients use the CA's public key (which should be pre-installed or securely obtained) to verify the certificates received from other clients.

Secure Messaging: Use the public key from the verified certificate to encrypt messages and establish secure channels (e.g., using AES for symmetric encryption of session communications).

## 4. Secure Communication Protocols

Using TLS/SSL Concepts:

Employ concepts similar to TLS/SSL for secure communications:

Handshake Protocol: Implement a handshake mechanism where certificates are exchanged and verified before any sensitive data is transmitted.

Session Keys: Use techniques like Diffie-Hellman within this secure environment to generate session keys for symmetric encryption.

## 5. Scalability and Security Measures

Scalability: The server should be designed to handle multiple connections simultaneously and manage a growing number of client registrations and certificate issuances.

Security Features: Implement logging, monitoring, and possibly rate limiting to prevent abuse (e.g., denial-of-service attacks) and to enhance security.

## 6. Testing and Documentation

Testing: Rigorously test the registration process, certificate issuance, and secure communication. Include tests for error handling and security breach attempts.

Documentation: Provide comprehensive documentation covering the architecture, flow of data, security measures, and user instructions.

## 7. Technology and Tools

Python Libraries: Utilize libraries like Flask for setting up the server, SQLAlchemy for database interactions, and PyCryptodome for cryptographic functions.

Networking: Use secure protocols and ensure all data transmitted between the client and server is encrypted.

By structuring your system in this way, you ensure a scalable, secure, and efficient CA model that supports the dynamic registration of new users while providing robust security features for communications. This setup not only fulfills the assignment's requirements but also offers a realistic model for secure network communication systems.

Flow:

Receiving message and Yusuf is Registered:

**CA and server are used interchangeably**

1. Yusuf connects to server
2. Y signs in (already registered with CA)
3. Server prompts directory for messages for Y
4. Y downloads messages meant for him
   1. Automatically send certificate for sender, gets sent certificate encrypted with CA private key from CA
      1. Good for man in the middle attacks
   2. (For each message, session key included)
   3. Decode from Base64 to utf-8
   4. Decrypt session key component with Y Private Key
   5. Decrypt message body with session key (gotten from prev step)
   6. Decompress message body (to get actual message and signature)
   7. Decrypt message digest with Greg’s (senders) Public Key
   8. Verify Message is unchanged (check integrity) compare hashes

Sending Message and Greg is Registered:

1. Greg connects to server
2. G signs in
3. G wants to send message to Y
   1. Query server to see if Y is registered (the following assumes Y is registered)
4. CA sends G a certificate that is encrypted with CA’s private key
   1. Decrypt with CA public key
   2. Certificate includes Yusuf’s Public key
5. Encrypt session key with Y public key
6. Encrypt message digest with G private key
7. Compress message body (everything except session key)
8. Encrypt zipped file with session key
9. Prepend to encrypted message
10. Encode and send message to Server

Non-registered User:

1. Prompted to register or exit

Server:

* Acts as a
  + Certificate Authority (CA)
  + Authentication Server (AS)
  + Message store
  + Communication bridge

Todo:

* Priority:
* Integrate CA into server
  + Directory/ data store
    - Possibly JSON
* What the hell DH?? Ask bossman about symmetric encryption
* Hashing
* Compression
* Encoding
* Message format
* Implement image + caption sending

NOTES:  
- Currently, email acts as the keyID

A diagram of a process

Description automatically generated