**Title:** Secure Chat Application with End-to-End Encryption Using Elliptic Curve Cryptography (ECC) and Post-Quantum Cryptography (PQC)

**Concepts:**

1. **Elliptic Curve Cryptography (ECC)**
   * ECDH
   * ECDSA
2. **End-to-End Encryption (E2EE)**
3. **Elliptic Curve Diffie-Hellman (ECDH)**
4. **Elliptic Curve Digital Signature Algorithm (ECDSA)**
5. **Symmetric Encryption (AES)**
6. **Modular Arithmetic & Finite Field Arithmetic**
7. **Post-Quantum Cryptography (PQC)**
   * Kyber (Key Encapsulation Mechanism)
   * Hybrid Key Exchange (Kyber + ECDH)

**Real-World Application or Problem Being Solved:**

1. Prevent Eavesdropping
2. Authenticate Users
3. Secure Key Exchange (Resistant to Quantum Attacks)

**Technology Being Used:**

1. **Programming Languages:**
   * Python, JavaScript for building the application backend and frontend.
2. **Cryptographic Libraries:**
   * PyCryptodome (for Python), ecdsa (for Python), OpenSSL (for C/C++), CryptoJS (for JavaScript)
   * Kyber implementation for Post-Quantum Security
3. **Frontend Development:**
   * React or Vue.js
4. **Networking:**
   * WebSockets or Socket.io
   * SSL/TLS encryption
5. **Server-side:**
   * Node.js or Flask/Django (Python)

**Project Summary:** This Secure Chat Application uses a hybrid cryptographic approach combining Elliptic Curve Cryptography (ECC) and Post-Quantum Cryptography (PQC) to provide End-to-End Encryption (E2EE). This ensures that messages exchanged between users remain secure, private, and authenticated even against potential future quantum computing threats. The key components include:

1. **Key Generation:**
   * Each user generates a public/private key pair using ECC.
   * The public key is shared with other users, while the private key remains secret.
2. **Hybrid Key Exchange (ECDH + Kyber):**
   * Instead of relying solely on ECDH for key exchange, a hybrid approach is used.
   * ECDH establishes an initial shared secret.
   * A Post-Quantum Key Encapsulation Mechanism (Kyber) generates an additional shared secret.
   * Both secrets are combined using HKDF(Hash-based key derivative function) to derive a final AES key, ensuring post-quantum security.
3. **Encryption (AES-256):**
   * The derived symmetric key is used to encrypt and decrypt messages using AES-256.
4. **Message Signing (ECDSA):**
   * Each message is signed with the sender's private key using ECDSA.
   * The recipient verifies the authenticity of the message using the sender’s public key.
5. **User Interface:**
   * The application features a clean, user-friendly interface, supporting secure real-time messaging, contact lists, and group chats.
6. **Real-Time Messaging:**
   * WebSockets or Socket.io ensures low-latency, encrypted message delivery.

**Technical Implementation of Hybrid Key Exchange:**

1. Generate ECDH shared secret.
2. Generate Kyber shared secret using a PQC library.
3. Combine both secrets using HKDF to derive the final AES key.

from cryptography.hazmat.primitives.kdf.hkdf import HKDF

from cryptography.hazmat.primitives import hashes

import os

# Existing ECC-based key exchange (ECDH)

ecdh\_shared\_secret = b"Existing\_ECDH\_Secret" # Replace with actual ECDH output

# New Post-Quantum Kyber-based key exchange

kyber\_shared\_secret = b"Kyber\_Shared\_Secret" # Replace with actual Kyber KEM output

# Combine both secrets using HKDF

salt = os.urandom(16) # Random salt

hkdf = HKDF(

algorithm=hashes.SHA256(),

length=32,

salt=salt,

info=b"Hybrid Key Exchange"

)

final\_aes\_key = hkdf.derive(ecdh\_shared\_secret + kyber\_shared\_secret)

print("Final AES Key:", final\_aes\_key.hex()) # Use this for AES encryption

**Advantages of Choosing Hybrid ECDH + Kyber over Normal RSA or ECC Alone:**

* **Quantum Resistance:** Protects against attacks from quantum computers.
* **Minimal Performance Overhead:** Uses both ECC and Kyber efficiently.
* **Backward Compatibility:** Retains ECC for compatibility while adding quantum security.
* **Low Bandwidth Usage:** ECC and Kyber both have relatively small key sizes compared to RSA.

**Comparison with Popular Chat Applications (WhatsApp):**

|  |  |  |
| --- | --- | --- |
| **Feature** | **WhatsApp (Signal Protocol)** | **Secure Chat App (ECC + PQC)** |
| **Key Exchange** | ECDH (Curve25519) | ECDH + Kyber Hybrid |
| **Encryption Algorithm** | AES-256 in CBC mode | AES-256 in GCM mode |
| **Message Authentication** | HMAC-SHA256 | HMAC-SHA256 |
| **Digital Signatures** | ECDSA (Curve25519) | ECDSA |
| **Forward Secrecy** | Double Ratchet Algorithm | Can be implemented |
| **Quantum Security** | No | Yes (Kyber-based KEM) |

**Step-by-Step Cryptographic Workflow:**

1. **Key Generation:**
   * Each user generates an ECC key pair (private + public key).
   * Public keys are shared with other users.
2. **Key Exchange (Hybrid ECDH + Kyber for Secure Agreement):**
   * Two users exchange their public ECC keys and Kyber encapsulated keys.
   * Both compute the same shared secret from ECDH.
   * A Kyber shared secret is derived and combined using HKDF.
   * The final shared secret is used to derive a symmetric AES key.
3. **Message Encryption (AES-256 for Confidentiality):**
   * Sender encrypts the message using AES-256 in GCM mode.
   * The encrypted message is then signed with ECDSA.
4. **Message Integrity Check (HMAC-SHA256):**
   * A HMAC is generated for the message to ensure it hasn’t been tampered with.
5. **Message Authentication & Non-Repudiation (ECDSA for Signing):**
   * The sender signs the encrypted message using ECDSA.
   * The recipient verifies the signature using the sender’s public key.
6. **Decryption & Verification:**
   * The recipient decrypts the message using AES-256.
   * Verifies the HMAC to ensure the message wasn’t altered.
   * Uses ECDSA verification to authenticate the sender.

**Final Architecture:**

1. **ECC (Curve25519 or secp256k1) for Key Exchange (ECDH)**
2. **Kyber KEM for Post-Quantum Secure Key Exchange**
3. **AES-256 in GCM Mode for Message Encryption**
4. **HMAC-SHA256 for Message Integrity**
5. **ECDSA for Digital Signatures (Authentication & Non-Repudiation)**

This approach ensures strong security, efficient performance, and minimal computational overhead, making it an ideal upgrade for post-quantum secure messaging.

**Reference Links:**

1. **Kyber, Key-Encapsulation Mechanism(KEM):**

[**https://medium.com/@hwupathum/crystals-kyber-the-key-to-post-quantum-encryption-3154b305e7bd#:~:text=In%20today's%20digital%20world%2C%20our,that%20take%20supercomputers%20millions%20of%E2%80%A6&text=Kyber%2C%20one%20of%20these%20selected,face%20of%20potential%20quantum%20threats**](https://medium.com/@hwupathum/crystals-kyber-the-key-to-post-quantum-encryption-3154b305e7bd#:~:text=In%20today's%20digital%20world%2C%20our,that%20take%20supercomputers%20millions%20of%E2%80%A6&text=Kyber%2C%20one%20of%20these%20selected,face%20of%20potential%20quantum%20threats)**.**

**Modules**

**Server Side Modules**The server is responsible for managing connections, handling key exchange, relaying encrypted messages, and ensuring secure communication.

**Client Side Modules**

The client application is responsible for secure key exchange, message encryption, authentication, and real-time communication.

|  |  |
| --- | --- |
| **Server** | **Client** |
| 1. **Connection Management Module**  Purpose: Manages multiple client connections and facilitates secure communication.  Maintains active client sessions.  Handles WebSocket connections for real-time messaging.  Securely stores client public keys (without storing private keys). | 1. Connection & Communication Module  Purpose: Establishes a secure connection to the server.  Uses Web Sockets for real-time message exchange.  Handles asynchronous communication with the server. |
| 2. Key Exchange & Hybrid Key Derivation Module  Purpose: Implements a hybrid post-quantum key exchange mechanism.  Generates an ECC key pair (ECDH).  Uses Kyber512 for post-quantum security.  Combines ECDH and Kyber keys using HKDF to derive a shared AES key.  Ensures perfect forward secrecy (PFS). | 2. Key Exchange & Hybrid Key Derivation Module  Purpose: Implements the hybrid post-quantum key exchange.  Generates ECC key pair and derives shared secret using ECDH.  Uses Kyber512 for quantum-resistant key exchange.  Derives AES-256 encryption key using HKDF. |
| 3. Message Encryption & Decryption Module  Purpose: Ensures confidentiality and integrity of messages using encryption.  Encrypts messages using AES-256 in GCM mode.  Decrypts incoming messages before forwarding to the recipient.  Uses HMAC-SHA256 to ensure message integrity. | 3. Message Encryption & Decryption Module  Purpose: Encrypts and decrypts messages before transmission.  Encrypts messages using AES-256 in GCM mode for confidentiality.  Decrypts incoming messages before displaying them.  Uses HMAC-SHA256 to verify message integrity. |
| 4. Digital Signature & Authentication Module  Purpose: Prevents message forgery and verifies sender authenticity.  Uses ECDSA (Elliptic Curve Digital Signature Algorithm) for signing messages.  Ensures non-repudiation, meaning a sender cannot deny sending a message.  Allows message verification using the sender’s public key. | 4. Digital Signature & Verification Module  Purpose: Ensures message authenticity and prevents forgery.  Uses ECDSA to sign messages before sending.  Verifies received messages using the sender’s public key.  Provides non-repudiation, ensuring the sender cannot deny a message. |
| 5. Secure Message Relay Module  Purpose: Handles the transmission of encrypted messages between clients.  Relays encrypted messages without decrypting them (zero knowledge).  Ensures low-latency message delivery using WebSockets.  Prevents MITM (Man-in-the-Middle) attacks by enforcing encryption. | 5. User Interface & Messaging Module  Purpose: Provides an intuitive chat interface for secure communication.  Displays encrypted messages in real-time.  Supports group chats and one-on-one messaging.  Notifies users of new messages, key exchange status, and security alerts. |
| 6. Logging & Security Audit Module  Purpose: Logs security-related events for auditing and debugging.  Logs authentication attempts, key exchanges, and connection requests.  Detects suspicious activity (e.g., multiple failed authentication attempts). | 6. Key Management & Secure Storage Module  Purpose: Manages cryptographic keys securely.  Stores private keys securely (not in plaintext).  Allows key regeneration and expiration for added security. |

**Additional Improvement:**

**Cause: Possible Attacks In Traditional**

**Why Use Kyber in Addition to ECDH?**

|  |  |  |
| --- | --- | --- |
| **Attack Type** | **RSA + AES (Traditional)** | **ECDH + Kyber Hybrid (Quantum-Safe)** |
| Quantum Attacks (Shor's Algorithm) | Broken | Secure |
| Harvest Now, Decrypt Later | Vulnerable | Resistant |
| MITM via Key Compromise | Single Key Breaks Security | Hybrid Security |
| Forward Secrecy Attacks | Not Always Ensured | Strong Forward Secrecy |
| Side-Channel Attacks | RSA/AES Timing Attacks Possible | Kyber is More Resistant |

Theory:

**Key-Encapsulation Mechanism (KEM)**

A Key-Encapsulation Mechanism (KEM) is used to send a symmetric key between two parties using asymmetric algorithms. Unlike Diffie-Hellman key exchange method where the shared secret is directly generated through mutual computations, a KEM employs asymmetric algorithms. In this method, the sender encapsulates the symmetric key within a cipher-text using the recipient’s public key. Upon receiving the cipher-text, the recipient then decapsulates and retrieves the symmetric key using their private key, ensuring a secure and authenticated exchange without directly sharing the symmetric key during transmission.

**Summary of the Flow**

1. Client connects to the server securely using WebSockets.
2. Key exchange happens using a hybrid ECDH + Kyber approach.
3. A shared AES-256 encryption key is derived using HKDF.
4. Messages are encrypted & signed before sending.
5. Server relays messages securely without decryption.
6. Recipient verifies, decrypts, and reads the message.
7. Keys are managed securely to ensure forward secrecy.
8. Logs & security checks monitor system integrity.