Project Architecture: Medical Record Wallet 🧬

1. Project Overview & Problem Statement

The **Medical Record Wallet** is a desktop application that allows patients to securely store and retrieve their medical records. Medical records contain highly sensitive personal information, and traditional storage solutions often leave patients vulnerable to data leaks. This project's goal is to build a secure, user-friendly wallet where patients retain full control over their data using a modern **hybrid cryptographic scheme**.

- **MVP Scope:** Patients can encrypt and locally save their medical records. They can then decrypt and view their records after authenticating.
- **Core Technology:** The system will use AES-256 for file encryption and RSA for key encryption.

---

2. Cryptographic Workflow

The system uses a hybrid approach to combine the speed of symmetric encryption with the security of asymmetric encryption.

1. **Key Generation:** Each user has a unique **RSA key pair** (2048-bit minimum). Their private key is encrypted and protected by a password using **PBKDF2**.
2. **Encryption (Storing a Record):**
    - A new, random **AES-256 session key** is generated for the file.
    - The medical record is encrypted using the AES key (preferably in **AES-GCM mode** to ensure both confidentiality and integrity).
    - The AES session key is then encrypted using the user's **public RSA key**.
    - The encrypted file and the encrypted AES key are stored.
3. **Decryption (Viewing a Record):**
    - The user provides their password to decrypt their **private RSA key**.
    - The private RSA key is used to decrypt the AES session key.
    - The recovered AES key is then used to decrypt the medical record.

---

3. Code & System Architecture

This architecture outlines the primary modules and data structure for the application.

- **Main Classes/Modules:**

    - `CryptoManager`: Handles all cryptographic operations (RSA/AES key generation, encryption, decryption).
    - `UserManager`: Manages user creation, authentication, and loading of user keys.
    - `FileManager`: Manages file uploads, downloads, and the orchestration of the encryption/decryption workflow.
    - `GUIController`: Manages user interactions from the graphical interface.

Data Storage Structure: A local directory structure will be used to manage user data.

```
MedicalWallet/
└── users/
    └── [user_id]/
        ├── keypair.pem (encrypted private key)
        ├── public.pem
        └── records/
            ├── record1.enc (encrypted file)
            ├── record1.key (RSA-encrypted AES key)
            └── record1.meta (filename, timestamp, etc.)
```

-

---

4. Implementation & Security

- **Language & Libraries:** The project will be implemented in **C++** using the **Crypto++** library for cryptographic functions and **Qt** for the graphical user interface.
- **Security Best Practices:**
    - Use a cryptographically secure random number generator for all keys.
    - Implement **OAEP padding** for all RSA encryption.
    - Use an authenticated encryption mode like **AES-GCM** to prevent tampering with encrypted records.
    - Securely clear all sensitive data (keys, plaintext files) from memory after use.