# 🔐 Giải Thích Cơ Chế Khóa & Token

## 1️⃣ Các Thành Phần Chính

**Các khóa và token** đóng vai trò bảo mật quá trình xác thực. Cụ thể gồm:

- **Private Key** (Khóa bí mật)
- **Public Key** (Khóa công khai)
- **Access Token**
- **Refresh Token**

Mỗi thành phần có nhiệm vụ riêng để bảo vệ thông tin đăng nhập và duy trì phiên làm việc của người dùng.

---

## 2️⃣ Cơ Chế Tổng Quan

Quá trình **đăng nhập & tạo token** hoạt động như sau:

1️⃣ **Người dùng nhập email và password.**

2️⃣ **Hệ thống kiểm tra thông tin**, nếu đúng sẽ tạo **cặp khóa Public/Private.**

3️⃣ **Dùng Public/Private Key để tạo Access Token & Refresh Token.**

4️⃣ **Lưu Refresh Token và Private Key vào database để xác thực sau này.**

5️⃣ **Trả Access Token & Refresh Token cho người dùng** để sử dụng API.

Mục tiêu của việc này là bảo vệ tài khoản người dùng, chống lại các cuộc tấn công như **token hijacking (đánh cắp token)** và **replay attack (tấn công phát lại)**.

---

## 3️⃣ Công Dụng Của Từng Thành Phần

### 🔑 1. Private Key & Public Key (Cặp Khóa Bất Đối Xứng)

📌 **Tạo trong AccessService:**

```
const privateKey = crypto.randomBytes(64).toString('hex');
const publicKey = crypto.randomBytes(64).toString('hex');
```

🔸 **Công dụng:**

- **Private Key**: Chỉ server biết, dùng để **ký (sign)** token.
- **Public Key**: Dùng để **xác minh token** (verify) khi client gửi request.

🔸 **Lý do sử dụng:**

- **Đảm bảo tính bảo mật**: Client không bao giờ thấy Private Key.
- **Ngăn chặn giả mạo token**: Chỉ có server mới có thể tạo Access Token hợp lệ.
- **Dễ dàng xác thực mà không cần lưu trữ token** trên server.

---

## 🔐 2. Access Token

📌 **Tạo bằng Public/Private Key:**

```
const tokens = await createKeyPair({ userId, email }, publicKey,
privateKey);
```

🔸 **Công dụng:**

- Dùng để **xác thực người dùng** mỗi lần họ gọi API.
- Thời gian sống **ngắn** (ví dụ: 15 phút) để tăng bảo mật.

🔸 **Lý do sử dụng:**

- **Nhanh chóng và tiện lợi:** Không cần truy vấn database mỗi lần người dùng gọi API.
- **Giảm tải cho server**: Chỉ cần kiểm tra chữ ký (verify) bằng Public Key.
- **Chống tấn công "Man-in-the-Middle"** (MITM): Vì không truyền password.

📌 **Cách sử dụng Access Token:**

Mỗi lần người dùng gọi API, họ gửi token qua header:
`Authorization: Bearer <access_token>`

- Server kiểm tra token bằng Public Key để xác nhận người dùng hợp lệ.

🔹 **Nếu token hết hạn?** → Người dùng cần **yêu cầu token mới bằng Refresh Token**.

---

## ♻️ 3. Refresh Token

📌 **Lưu vào database:**

```
await keyTokenService.createKeyToken({
    refreshToken: tokens.refreshToken,
    privateKey,
    publicKey,
    userId
});
```

🔸 **Công dụng:**

- Dùng để **tạo Access Token mới khi token cũ hết hạn**.
- Thời gian sống **dài hơn Access Token** (ví dụ: 7 ngày hoặc 30 ngày).

🔸 **Lý do sử dụng:**

- **Không bắt người dùng đăng nhập lại quá thường xuyên**.
- **Tăng bảo mật**: Nếu hacker lấy được Access Token, nó cũng sẽ hết hạn nhanh chóng.
- **Kiểm soát được việc sử dụng lại Refresh Token** trong database.

📌 **Cách hoạt động:**

1. Khi Access Token hết hạn, client gửi Refresh Token lên server.
2. Server kiểm tra Refresh Token trong database.
3. Nếu hợp lệ, server cấp một **Access Token mới** và cập nhật Refresh Token.
4. Nếu Refresh Token đã bị sử dụng trước đó, từ chối yêu cầu (ngăn chặn replay attack).

📌 **Lưu Refresh Token vào DB giúp:**

- Hủy bỏ Refresh Token nếu người dùng **đăng xuất hoặc bị tấn công**.
- Ngăn chặn việc dùng lại Refresh Token cũ.

---

## 4️⃣ Mối Quan Hệ Giữa Các Thành Phần

| Thành Phần | Lưu Ở Đâu? | Công Dụng |
|---|---|---|
| **Private Key** | Chỉ trên Server | Ký (sign) token |
| **Public Key** | Server & Client | Xác minh (verify) token |
| **Access Token** | Client (localStorage/cookie) | Xác thực API |
| **Refresh Token** | Database (MongoDB) & Client | Cấp lại Access Token |

---

## 5️⃣ Tại Sao Không Dùng Chỉ Access Token Hoặc Refresh Token?

🚫 **Chỉ dùng Access Token:**

- Hacker có thể **đánh cắp Access Token** và dùng nó cho đến khi hết hạn.
- Nếu thời gian sống của Access Token quá dài, hacker có thể **giả mạo người dùng trong thời gian dài**.
- Nếu quá ngắn, người dùng phải đăng nhập lại liên tục → gây khó chịu.

🚫 **Chỉ dùng Refresh Token:**

- Refresh Token có thời gian sống dài, nếu hacker lấy được → **có thể tạo vô hạn Access Token**.
- Nếu không lưu Refresh Token trong database, **không thể hủy token khi cần**.

✅ **Kết hợp Access Token + Refresh Token:**

- Access Token **ngắn hạn** → Hạn chế rủi ro nếu bị đánh cắp.
- Refresh Token **dài hạn nhưng kiểm soát được** → Có thể thu hồi nếu bị đánh cắp.

---

## 6️⃣ Bảo Mật Thêm Cho Token

💡 **Cách tăng cường bảo mật:**
✅ **Lưu Refresh Token trong HTTP-only Cookie** để tránh bị đánh cắp qua JavaScript (XSS Attack).
✅ **Dùng HTTPS để mã hóa token khi gửi qua mạng**.
✅ **Thêm IP hoặc User-Agent vào Refresh Token** để ngăn chặn dùng trên thiết bị khác.
✅ **Cơ chế "Single Logout"**: Khi người dùng đăng xuất, hủy Refresh Token trong database.
✅ **Rotate Refresh Token**: Mỗi lần cấp Access Token mới, tạo Refresh Token mới để ngăn chặn replay attack.

---

## 7️⃣ Kết Luận

1️⃣ **Private Key & Public Key** giúp tạo và xác minh token mà không cần lưu trữ trên server.
2️⃣ **Access Token** giúp xác thực nhanh nhưng có thời gian ngắn để giảm rủi ro.
3️⃣ **Refresh Token** giúp cấp lại Access Token nhưng được lưu trữ và kiểm soát chặt chẽ trong database.
4️⃣ **Dùng kết hợp cả hai giúp tối ưu giữa bảo mật và trải nghiệm người dùng**.

🚀 **Bạn có thể thử kiểm tra hệ thống bằng cách**:
✅ **Đăng nhập, kiểm tra Access Token & Refresh Token**.
✅ **Chờ Access Token hết hạn, thử gọi API xem có lỗi không**.
✅ **Dùng Refresh Token để lấy Access Token mới**.
✅ **Xóa Refresh Token khỏi DB và kiểm tra lại xem có đăng nhập lại được không**.