OneID and Electronic Signature: Detailed Documentation

Table of Contents

---

1. Introduction

OneID and Electronic Signature is a secure authentication and identity verification system designed to enhance user verification and data security. The system integrates multiple authentication methods, including password-based authentication and private key-based authentication. It also employs biometric verification, such as facial recognition and Optical Character Recognition (OCR), to validate identity documents.

This system ensures secure user identification and document verification while providing a seamless login experience for end users. It is particularly useful in environments where security and identity verification are critical, such as online banking, government portals, and enterprise applications.

---

2. System Overview

The OneID system provides a multi-faceted approach to authentication and identity verification. Users can log in through traditional credential-based authentication or utilize RSA-based private key authentication. The system also verifies user identity by comparing live facial images with those on ID cards and extracting data from identity documents through OCR.

Key components of the system:

- **User Authentication**: Credential-based login and private key authentication.
- **Identity Verification**: Validation using ID card images, OCR, and facial recognition.
- **Electronic Signatures**: Users can securely sign documents with cryptographic keys.
- **Secure Data Handling**: User information, passwords, and private keys are securely encrypted and stored.

---

3. Technology Stack

- **Backend**: Spring Boot (Java), Hibernate (JPA), Spring Security
- **Frontend**: HTML, JavaScript, Bootstrap
- **Database**: H2 Database (for development), MySQL (for production)
- **Security**: RSA Encryption, SSL/TLS
- **OCR and Face Recognition**: Tesseract OCR, Python Face Recognition

---

4. Key Functionalities

The OneID system incorporates various features to ensure secure authentication and identity verification. These functionalities include:

a) User Registration

- Users sign up with their **username, email, and password**.
- Strong password validation is enforced.
- A verification email with a security code is sent to verify the email address.

b) Login Mechanisms

- **Credential-based authentication**: Users log in with their **username and password**.
- **Private Key authentication**: Users authenticate by signing a random string with their **private key**.

c) Identity Verification

- **OCR-based ID card scanning**: Extracts name, surname, and SSN from the uploaded ID card.
- **Facial Recognition**: Matches the user's live photo with the ID card image.

d) Electronic Signature

- Users can digitally sign documents using their private keys.
- The system verifies the signature using the corresponding public key stored in the database.

e) Data Security

- All sensitive data is encrypted.
- Public and private key pairs are generated for each user.
- SSL/TLS is enforced for secure communication.

---

5. Authentication Mechanisms

Credential-Based Authentication

1. User enters their **username and password**.
2. Password is validated against the stored hash.
3. If correct, a verification code is sent to the user's email.
4. User enters the code to complete the login process.

Private Key Authentication

1. User uploads their **private key file**.
2. A **random challenge string** is generated.
3. The user signs the string with their private key.

4. The signature is sent to the server.
5. The server verifies the signature using the **stored public key**.
6. If valid, the user is authenticated.

---

6. Security Features

SSL/TLS Implementation

- Enforces encrypted communication using SSL.
- Uses a self-signed certificate for development and production SSL certificates for secure environments.

Public/Private Key Generation

- Uses RSA (2048-bit) encryption to generate key pairs.
- Public keys are stored securely in the database.
- Private keys are stored on the user's local machine.

Encryption of Sensitive Data

- Passwords are stored as **hashed values** using bcrypt.
- Sensitive user data (e.g., ID card images) is stored in **binary format** with access restrictions.

---

7. User Management and Data Handling

- Users can update their **profile information**.
- Admins can validate users based on ID verification.
- If **name, surname, and SSN match**, the account is verified.
- If **facial recognition succeeds**, the account is fully validated.

---

8. Biometric Verification and OCR

OCR (Optical Character Recognition)

1. Extracts text from the **uploaded ID card image**.

2. Compares the extracted text with user-provided data.
3. If the data matches, the system marks the ID as verified.

Face Recognition

1. Captures a **live photo** from the user.
2. Compares it with the **photo on the ID card**.
3. If the match is successful, the system marks the face as verified.

---

9. Services and Business Logic

UserService

- Handles user creation, updating, and validation.

KeyService

- Generates and manages **public/private keys** for each user.

OCRService

- Extracts user data from ID cards and matches it with stored records.

ImageService

- Compares ID card photos with live photos for identity verification.

AuthService

- Manages authentication, including **password-based and private key authentication**.

---

10. Controllers and API Endpoints

AuthController

- Handles **user login and registration**.
- Manages verification codes and validation.

DataController

- Handles **uploading ID cards and live photos**.

VerificationController

- Validates **private key authentication**.
- Checks whether a signed challenge is correct.

---

11. Frontend Implementation

Credential-Based Login

```
<form action="/login" method="post">
  <input type="text" name="username" placeholder="Username" required>
  <input type="password" name="password" placeholder="Password" required>
  <button type="submit">Login</button>
</form>
```

Private Key Authentication

```
async function encryptAndLogin(event) {
  let privateKeyFile = document.getElementById("private-key").files[0];
  let privateKey = await readFile(privateKeyFile);
  let signature = await signWithPrivateKey(privateKey, "challenge_string");
  fetch("/oneid/verifyLogin", {method: "POST", body: JSON.stringify({username, signature})});
}
```

---

12. Deployment and Configuration

- **Application Configuration**:
- spring.datasource.url=jdbc:h2:file:~/data/users
- spring.jpa.hibernate.ddl-auto=update
- server.ssl.enabled=true
- **Running the Application**:

- mvn spring-boot:run

---

13. Conclusion

The OneID system is designed for secure user authentication using **electronic signatures, OCR-based ID verification, and facial recognition**. By leveraging **RSA encryption, SSL/TLS, and biometric validation**, it ensures **high security** and **reliable identity verification**. This system is particularly beneficial for applications requiring strong authentication, such as **financial services, healthcare, and government systems**.