Emzano Signing SDK

A signing SDK that generates non-exportable P-256/RSA keys, builds PKCS#10 CSRs with user authentication, calls a backend to issue certificates, and caches certificate chains securely. Android is a pure Android SDK; iOS is a native Swift package designed for wrapper-friendly APIs (React Native / Capacitor later).

Modules

- `sdk-android` (Android SDK): core API, error model, network abstractions, crypto + CSR + secure storage, and Android wrapper (`AndroidSigningSdk`).
- `sdk-ios` (Swift package): native iOS SDK (`SigningSdk`), Keychain crypto + CSR, URLSession networking, secure storage.
- `sample-android` (App): Demonstrates initialization, key generation, CSR creation, mock certificate issuance, and cache reading with a minimal UI.
- `sample-ios` (SwiftUI app): Demonstrates init/key/CSR/issue/cached cert using the Swift package with a mock network client and token provider.

Android overview

- Keys: EC P-256 generated in Android Keystore with `SHA256withECDSA`. `AuthPolicy.PerUse` or `AuthPolicy.TimeWindow(seconds)` configure user authentication requirements.
- Biometrics: `BiometricPrompt` UI strings **must** be supplied by the caller (`BiometricUiConfig`). The SDK never hardcodes prompt strings.
- CSR: Built with BouncyCastle; signature authorized via `BiometricPrompt` on a `FragmentActivity`.
- Storage: Certificate chain cached via `EncryptedSharedPreferences` per alias.
- Networking: Ktor client with optional OkHttp certificate pinning (set `PinningConfig` in `SdkConfig`).

Android wrapper (`AndroidSigningSdk`)

```
val sdk = AndroidSigningSdk(context)
val config = SdkConfig(
    baseUrl = "https://api.example.com",
    pinningConfig = PinningConfig(host = "api.example.com", sha256Pins
= listOf("pin1", "pin2"))
)
sdk.initialize(config)
```

Biometric UI config (from app strings):

```
val ui = BiometricUiConfig(
    title = getString(R.string.biometric_title),
    subtitle = getString(R.string.biometric_subtitle),
    description = getString(R.string.biometric_description),
    negativeButtonText = getString(R.string.biometric_negative)
)
```

Key & CSR operations (require `FragmentActivity` for `BiometricPrompt`):

```
sdk.generateKeyPair(this, alias, AuthPolicy.PerUse, ui,
KeyAlgorithm.RSA_2048)
val csr = sdk.createCsr(this, alias, subject, CsrExtensions(), ui)
val signature = sdk.signDigest(this, alias, digestBase64,
DigestAlgorithm.SHA256, ui)
```

Issuing and caching certificates (async issue + polling):

```
val tokenProvider = object : TokenProvider {
    override suspend fun getAccessToken() = "access-token"
    override suspend fun refreshAccessToken() = null // or refreshed
token
}
val request = IssueCertificateRequest(
    enName = "Ali",
    enLastName = "Ahmadi",
    postalCode = "1234567890",
    city = "Tehran",
    provinceName = "Tehran",
    base64Csr = csrPem.toPemBase64()
)
val certResult = sdk.issueCertificate(alias, tokenProvider, request)
val cached = sdk.getCachedCertificate(alias)
```

Helper to convert PEM CSR into base64 payload:

```
private fun String.toPemBase64(): String =
    replace("-----BEGIN CERTIFICATE REQUEST-----", "")
        .replace("-----END CERTIFICATE REQUEST-----", "")
        .replace("\\s".toRegex(), "")
```

Polling starts after 10 seconds and repeats every 2 seconds by default. Override via `SdkConfig.pollingConfig` or pass a

custom `CertificatePollingConfig` to `issueCertificate`. Configure endpoints with `SdkConfig.issueEndpoint` and `SdkConfig.checkEndpoint`.

`SdkResult.Ok` contains the value; `SdkResult.Err` carries `SdkError` (JSON friendly code/message/metadata). Error codes include `KEY_NOT_FOUND`, `KEY_INVALIDATED`, `KEYSTORE_UNAVAILABLE`, `BIOMETRIC_FAILED`, `BIOMETRIC_CANCELED`, `NETWORK_ERROR`, `AUTH_REQUIRED`, `SERVER_ERROR`, `CSR_BUILD_ERROR`, `CERT_CACHE_ERROR`, `INVALID_ARGUMENT`, `INTERNAL_ERROR`.

iOS overview (native Swift)

- Keys: EC P-256 or RSA 2048 stored in Keychain; EC uses Secure Enclave on device with `kSecAccessControlPrivateKeyUsage` and user presence.
- Biometrics: `BiometricPromptConfig` requires a non-empty `localizedReason` provided by the app.
- CSR: Built with manual ASN.1 DER encoding and `SecKeyCreateSignature` (`SHA256withECDSA` or `SHA256withRSA`), wrapped as PEM.
- Storage: Certificate cache stored in Keychain (JSON payload per alias).
- Networking: URLSession with optional SHA256 certificate pinning (set `PinningConfig` in `SdkConfig`).

iOS usage

```
import EmzanoSigning

let sdk = SigningSdk()
let config = SdkConfig(baseUrl: "https://api.example.com")
let _ = await sdk.initialize(config: config)

let prompt = BiometricPromptConfig(localizedReason: "Confirm to sign
request")
let _ = await sdk.generateKeyPair(alias: "ios-key", authPolicy:
.perUse, algorithm: .rsa2048)
let csr = await sdk.createCsr(alias: "ios-key", subject:
SubjectDn(commonName: "User"), extensions: CsrExtensions(), prompt:
prompt)
let signature = await sdk.signDigest(alias: "ios-key", digestBase64:
digestBase64, digestAlgorithm: .sha256, prompt: prompt)

let request = IssueCertificateRequest(
    enName: "Ali",
    enLastName: "Ahmadi",
    postalCode: "1234567890",
```

```
    city: "Tehran",
    provinceName: "Tehran",
    base64Csr: csrPem.toPemBase64()
)
let cert = await sdk.issueCertificate(alias: "ios-key", tokenProvider:
tokenProvider, request: request)
private extension String {
    func toPemBase64() -> String {
        return replacingOccurrences(of: "-----BEGIN CERTIFICATE
REQUEST-----", with: "")
            .replacingOccurrences(of: "-----END CERTIFICATE REQUEST----
-", with: "")
            .replacingOccurrences(of: "\\s", with: "", options:
.regularExpression)
    }
}
```

Sample apps

- Android: launch `sample-android`. Buttons: initialize, generate key, create CSR, issue certificate (mock client), show cached certificate. Uses `BiometricUiConfig` strings from resources; mock backend returns dummy cert chain.
- iOS: open `sample-ios/SampleIosApp.xcodeproj` and run the SwiftUI app. The project references the local Swift package at `sdk-ios`. Buttons mirror Android flow; uses `SimpleTokenProvider` and `MockNetworkClient` for demo cert issuance/caching.

Tests

- `sdk-android test`: verifies auth-refresh state machine and caching behavior.
- `sdk-android androidTest`: basic keystore generation and CSR failure path without an activity (ensures graceful error mapping).

Building

Use the bundled Gradle wrapper for Android:

```
./gradlew :sample-android:installDebug
```

Requires JDK 17 and Android SDK with compileSdk 34.

For iOS sample:

- Open `sample-ios/SampleIosApp.xcodeproj`
- Build/run on iOS 14+ simulator or device