2022

CamDigiKey Integration

RUNNING AND USING CAMDIGIKEY CLIENT

CAMDX TEAM

Version	Description
1.0.0	• Get QR code for scanning • Get user access token • Get organization access token • Renew access token • Verify access token • Log out access token
1.1.0	• Lookup user account • Verify user account token
2.0.0	• Deprecate getting QR code for scanning • Web-to-Web integration (OAuth 2.0) • App-to-App integration (deep-link) • KYC verification APIs

Table of Contents

1. Introduction

CamDigiKey is a E-KYC service provider focusing on convenience and security for users. Basically, users need to register themselves to the application by providing their faces and identity documents such as ID card or passport. Once the user's account is approved, they can use CamDigiKey mobile application to login to any CamDigiKey partner's portals.

For organization or institution, CamDigiKey provides open KYC verification APIs which enable fast user on board on the organization or institution's platform. The open KYC verification APIs consist of checking user info and face against data from MOI and verifying user liveness to ensure user is a real human.

This document will focus on integration with CamDigiKey as authentication/authorization and the usage of open KYC verification APIs via CamDX.

2. CamDigiKey as Authentication Service

2.1. CamDigiKey Client

To use CamDigiKey service, client needs to contact our supporter for account registration with the following information:

- Organization name
- System domain name
- App name
- Logo
- Privacy policy URL
- Web success callback URL
- Web error callback URL
- Android app ID
- iOS bundle ID

Once, the account has been created, partner will receive information:

- Client access credential parameters
    - Domain
    - Client ID
    - HMAC key
    - IV Param key
    - Secret key
- TLS keystore and trust store

2.2. How It Works

2.2.1. Web-to-Web Integration

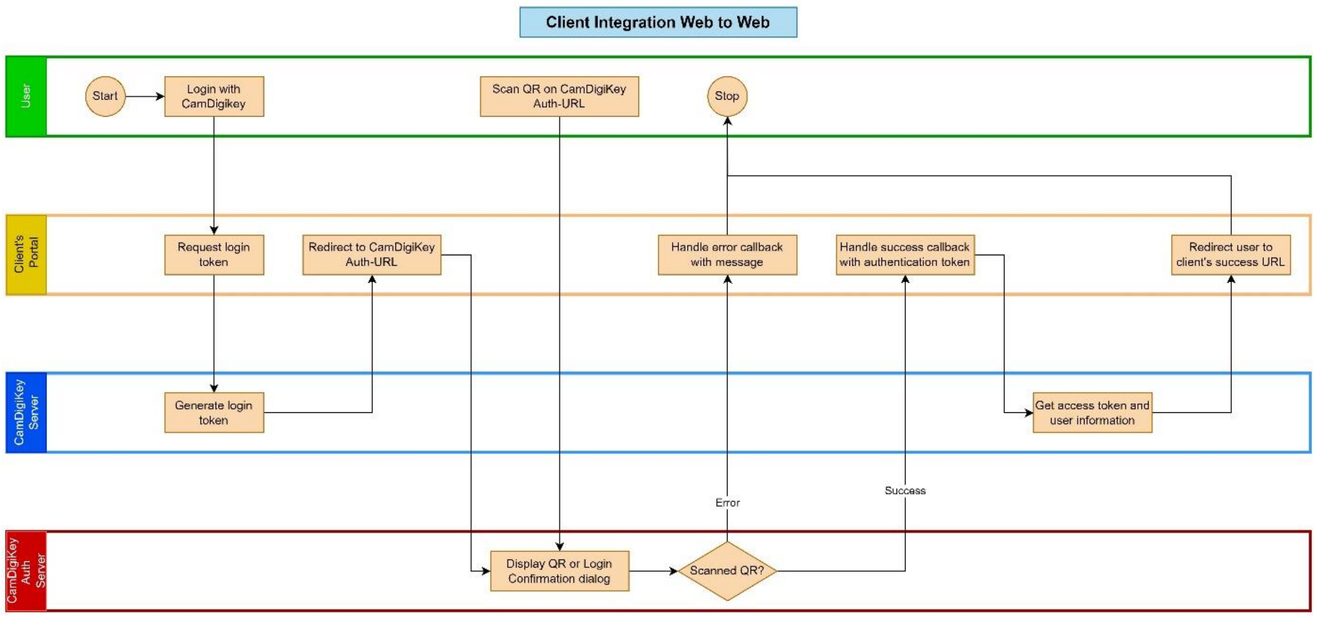

Client Integration Web to Web
User
Client's Portal
CamDigiKey Server
CamDigiKey Auth Server
Start
Login with CamDigikey
Scan QR on CamDigiKey Auth-URL
Stop
Request login token
Redirect to CamDigiKey Auth-URL
Handle error callback with message
Handle success callback with authentication token
Redirect user to client's success URL
Generate login token
Get access token and user information
Error
Success
Display QR or Login Confirmation dialog
Scanned QR?

Figure 1: Web-to-Web Integration

Figure 1 shows interaction between systems when user log in to client's portal via CamDigiKey:

1. When user clicks on login with CamDigiKey, client's application must request **login token** from CamDigiKey server
2. Redirect user to CamDigiKey's authentication server with the login token
3. If user has not logged in, QR code will be displayed for user to scan with their CamDigiKey mobile application. Otherwise, a confirmation dialog will be shown up instead.
4. If the QR code has been successfully scanned or granted by the user via CamDigiKey mobile application, authentication server will redirect user to success callback URL of the client portal with **authentication token**. Otherwise, authentication server will redirect user to error URL of the client portal with error message.
5. After receiving authentication token, client's server can request to CamDigiKey server with the authentication token to get **access token** and **user information**.
6. Finally, client's portal can redirect user to success login URL.

2.2.2. App-to-App Integration

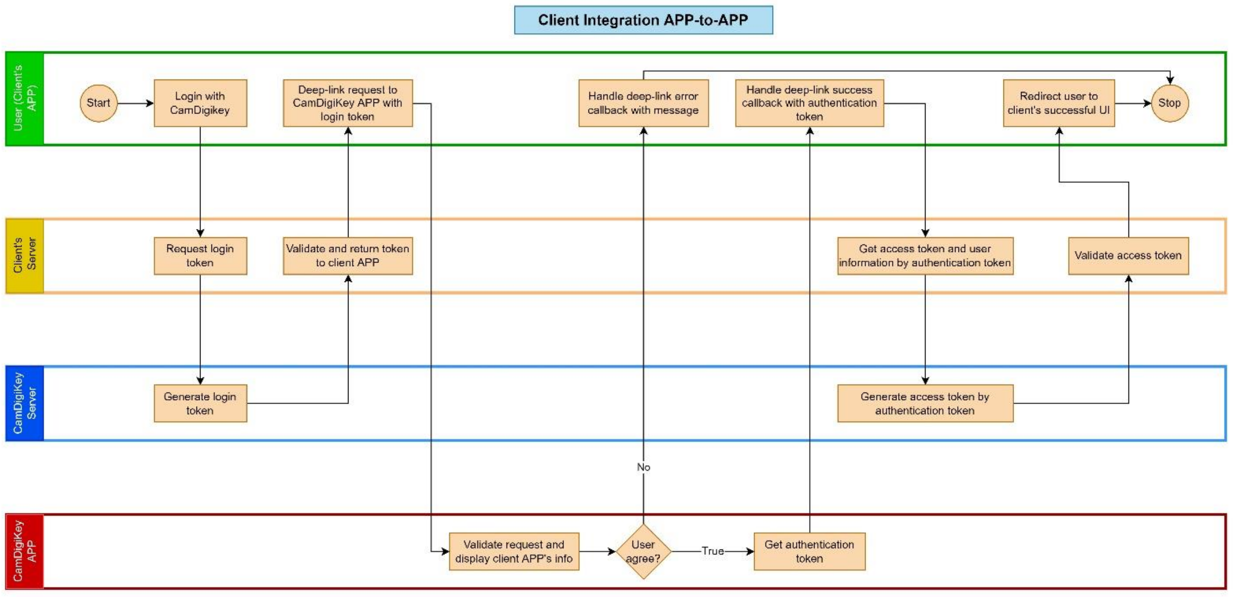

Client Integration APP-to-APP
User (Client's APP)
Client's Server
CamDigiKey Server
CamDigiKey APP
Start
Login with CamDigikey
Deep-link request to CamDigiKey APP with login token
Handle deep-link error callback with message
Handle deep-link success callback with authentication token
Redirect user to client's successful UI
Stop
Request login token
Validate and return token to client APP
Get access token and user information by authentication token
Validate access token
Generate login token
Generate access token by authentication token
No
Validate request and display client APP's info
User agree?
True
Get authentication token

Figure 2: App-to-App Integration

Figure 2 describes integration process between client's app and CamDigiKey app:

1. When user clicked on login with CamDigiKey in mobile client's app, mobile client app must get **login token** from CamDigiKey server via their client application server.
2. After client's app get login token, client's app can create **deep-link** request to CamDigiKey app.
3. If the login token is valid, CamDigiKey app will prompt users to accept or reject request. Then, **authorization token** will be retrieved and sent back to the client app for retrieving access token. Otherwise, CamDigiKey app will callback to deep-link error URL of the client's app.
4. When client app got authorization token, client app can send it to client' server to get **access token** and **user information**.

2.3. CamDigiKey Client App

CamDigiKey client application is a Java application that build on top of CamDigiKey client library. The client application provides API access (REST) without any authentication. **For security concern, please install the client application for local system access only.** Please follow the steps to config app credential:

- Edit `src/main/resources/application.properties`:

```
# add client credential properties
camdigikey.client-id = <client_id>       # provided by us
camdigikey.hmac-key = <hmac_key>         # provided by us
camdigikey.aes-secret-key = <secret_key> # provided by us
camdigikey.aes-iv-params = <iv_params>   # provided by us
camdigikey.client-domain = <your_application_domain>
camdigikey.server-based-url= <CamDigiKey Url> # Dev : https://mobile-id.demo.camdx.io:8444
                                              # Prod: https://camdigikey.camdx.gov.kh:8444
# add client connection key store file configuration
camdigikey.client-keystore-file = <path to client key store file>
camdigikey.client-keystore-file-password = <password of key store file>
camdigikey.client-keystore-client-key-entry-name = <client_key_entry_name>
camdigikey.client-keystore-client-key-entry-password = <client_key_entry_password>
camdigikey.client-trust-store-file = <path of the trust store>
camdigikey.client-trust-store-file-password = <trust_password>
```

- Rebuild and run app
  - Go to app folder
  - Run: `mvn package`
  - Run: `java -jar target/application-1.0.0.jar`

CamDigiKey client application provides APIs:

2.3.1. Get QR code (deprecated)

~~URL~~	~~/qr-code~~
~~Method~~	~~GET~~
~~Response~~	~~Object:~~ • ~~authCode: String (base 64)~~ • ~~qrImage: String (base 64)~~ ~~Portal system must render QR image in the img tag. For instance, <img src="data:image/png;base64, [qrImage]">~~

2.3.2. Get Login Token

URL	/login-token
Method	GET
Response	Object: • loginUrl: String • loginToken: String (base64)

2.3.3. Get user access token

URL	/access-token
Method	GET
Request	Object: • authToken: String (base 64)
Response	Object: • accessToken: String (base64)

2.3.4. Validate user or organization access token

URL	/
Method	POST
Request	Object:

	• token: String
Response	Object (payload can be **user** or **organization**): • user key payload: Object ○ camdigikey_id: String ○ first_name_en: String ○ last_name_en: String ○ first_name_kh: String ○ last_name_kh: String ○ gender: String (M: Male, F: Female) ○ dob: String (YYYY-MM-DD) ○ iss: String (CamDX-CamDigiKey) ○ **type: String (USER)** ○ personal_code: String (current user's ID card number or passport number) ○ mobile_phone: String ○ email: String ○ nationality: String ○ exp: timestamp (expired date) ○ nbf: timestamp (not before date) ○ iat: timestamp (issued at date) ○ jti: String (JWT ID) ○ is_valid: boolean • organization key payload: Object ○ domain: String ○ iss: String (CamDX-CamDigiKey) ○ **type: String (ORGANIZATION)** ○ exp: timestamp (expired date) ○ iat: timestamp (issued at date) ○ jti: String (JWT ID) ○ is_valid: boolean

2.3.5. Refresh user access token

Note: refresh token API should be called before access token expired.

URL	/refresh-user-access-token
Method	GET
Request	Object: • accessToken: String (base 64)
Response	Object: • accessToken: String (base64)

2.3.6. Get organization access token

Note: CamDigiKey also provides organization level access token which is used for API interaction between CamDX participants.

URL	/organization-access-token

Method	GET
Response	Object: • accessToken: String (base64)

2.3.7. Logout access token

URL	/logout-access-token
Method	GET
Request	Object: • accessToken: String (base 64)
Response	Object: • jwtId: string • expiry: timestamp • issued: timestamp • accessToken: String (base64) ○ status: int (0: logout)

2.3.8. Lookup CamDigiKey user account

URL	/lookup-user-profile
Method	GET
Request	Object: • accessToken: String (base 64) • personal_code: String
Response	Object: • personal_code: String • camdigikey_id: String • surname_kh: String • given_name_kh: String • surname_en: String • given_name_en: String • mobile_phone_number: String • email_address: String • expired_date: timestamp • **account_token: String**

2.3.9. Verify user account token

URL	/verify-user-profile
Method	GET
Request	Object: • accountToken: String (base 64)
Response	Object: • personal_code: String • camdigikey_id: String

	• surname_kh: String • given_name_kh: String • surname_en: String • given_name_en: String • mobile_phone_number: String • email_address: String

2.4. CamDigiKey Client Library

Currently, the client library of CamDigiKey is available to be implemented in Java. To use CamDigiKey client library in your project please follow the steps below:

2.4.1. Java Library

- Register Jar file in local Maven
- Add dependencies libraries in `POM.xml`

```
<dependency>
  <groupId>kh.gov.camdx.camdigikey</groupId>
  <artifactId>client</artifactId>
  <version>1.5</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

- Add library properties

```
# add client credential properties
camdigikey.client-id = <client_id>       # provided by us
camdigikey.hmac-key = <hmac_key>         # provided by us
camdigikey.aes-secret-key = <secret_key> # provided by us
camdigikey.aes-iv-params = <iv_params>   # provided by us
camdigikey.client-domain = <your_application_domain>
camdigikey.server-based-url= <CamDigiKey Url> # DEV : https://mobile-id.demo.camdx.io:8444
                                              # Prod: https://camdigikey.camdx.gov.kh:8444

# add client connection key store file configuration
camdigikey.client-keystore-file = <path to client key store file>
camdigikey.client-keystore-file-password = <password of key store file>
camdigikey.client-keystore-client-key-entry-name= <client_key_entry_name>
camdigikey.client-keystore-client-key-entry-password= <client_key_entry_password>
camdigikey.client-trust-store-file = <path of the trust store>
camdigikey.client-trust-store-file-password = <trust_password>
```

- Use `@Autowired` annotation of class `CamDigiKeyClient`

The client Java library provides methods:

```
1. validateJwt(String jwt) : HashMap<String, Object>
2. getOrganizationAccessToken() : HashMap<String, Object>
3. getLoginToken() : HashMap<String, Object>
4. getUserAccessToken(String authCode) : HashMap<String, Object>
5. refreshUserAccessToken(String accessToken) : HashMap<String, Object>
6. logoutAccessToken(String accessToken) : HashMap<String, Object>
7. lookupUserProfile(String accessToken, String personal_code) : HashMap<String, Object>
8. verifyUserProfile(String accessToken) : HashMap<String, Object>
9. verifyAccountToken(String accountToken): HashMap<String, Object>
```

2.4.2. Node Library

- Installation
  - In terminal, go to `node_modules` directory
  - Run: `git clone <url_to_client_lib_repository> camdigikey-client`
  - Run: `cd camdigikey-client && yarn install`
- Usage
  - Import library

```
const { CamDigiKeyClient } = require('camdigikey-client')
// or
import { CamDigiKeyClient } from 'camdigikey-client'
```

  - Create environment variables for CamDigiKey client library

```
# CamDigiKey Client Credential Configuration
CAMDIGIKEY_CLIENT_ID="client_id"
CAMDIGIKEY_HMAC_KEY="hmac_key"
CAMDIGIKEY_AES_SECRET_KEY="secret_key"
CAMDIGIKEY_AES_IV_PARAMS="iv_param"
CAMDIGIKEY_CLIENT_DOMAIN="<application_domain>"
CAMDIGIKEY_SERVER_BASED_URL="<camdigikey_url>"
# Dev : https://mobile-id.demo.camdx.io:8444
# Prod: https://camdigikey.camdx.gov.kh:8444

# CamDigiKey Client Connection KeyStore File Configuration
CAMDIGIKEY_CLIENT_KEYSTORE_FILE="<path to client keystore file>" # e.g. "./client1.p12"
CAMDIGIKEY_CLIENT_KEYSTORE_FILE_PASSWORD="<password of keystore file>"

CAMDIGIKEY_CLIENT_TRUST_STORE_FILE="<path to trusted keystore file>"
CAMDIGIKEY_CLIENT_TRUST_STORE_FILE_PASSWORD="<password of keystore file>"
```

  - Usage: e.g. get login token

```
const res = await CamDigiKeyClient.getLoginToken();
console.log(res);
```

The client node library provides functions:

```
1. validateJwt(jwt: string): Promise<Object>
```

2. `getOrganizationAccessToken(): Promise<Object>`
3. `getLoginToken(): Promise<Object>`
4. `getUserAccessToken(authToken: string): Promise<Object>`
5. `refreshUserAccessToken(accessToken: string): Promise<Object>`
6. `logoutAccessToken(accessToken: string): Promise<Object>`
7. `lookupUserProfile(accessToken: string, personalCode: string): Promise<Object>`
8. `verifyAccountToken(accountToken: string): Promise<Object>`

2.5. CamDigiKey Mobile Integration

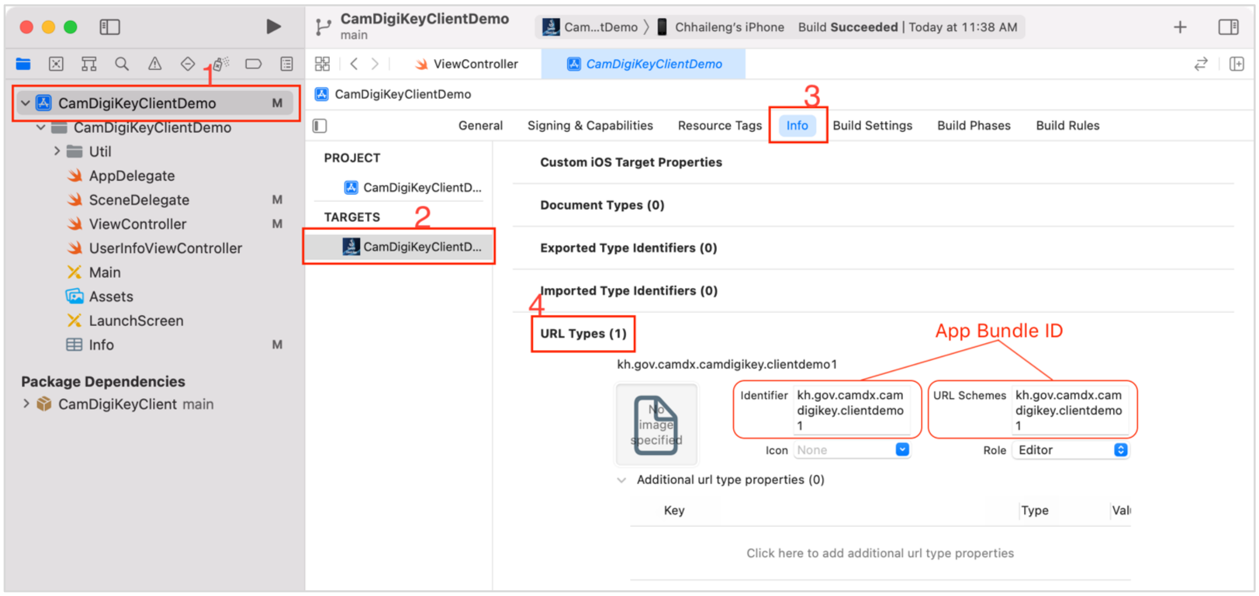
To integrate app-to-app authentication with CamDigiKey app, a client must have an API server which its endpoints communicate with CamDigiKey server using CamDigiKey client library. Then, client mobile application requests to client API server to get `loginToken` (as shown in figure 2). These are the standard format of DeepLink URLs communication:

- Client App opens CamDigiKey App (Login with CamDigiKey):
  `camdigikey://login?token=<loginToken>`
- CamDigiKey App opens Client App (Login callback):
  - Success: `<client_app_id>://camdigikey_login_callback?authToken=<authToken>`
  - Error: `<client_app_id>://camdigikey_login_callback?message=<message>`

The following are the mobile integration steps:

2.5.1. iOS

- Create a DeepLink URL Scheme:
  - In the Xcode project, click on the project name and select your project target
  - Open **Info** tab and expand **URL Types**
  - Click + and enter App bundle ID as **Identifier** and **URL Schemes**



kh.gov.camdx.camdigikey.clientdemo1

Identifier kh.gov.camdx.camdigikey.clientdemo1

URL Schemes kh.gov.camdx.camdigikey.clientdemo1

Icon None

Role Editor

- Alternately, you can create a DeepLink URL Scheme via `info.plist` file as follow:

```
<key>CFBundleURLTypes</key>
```

```
<array>
  <dict>
    <key>CFBundleTypeRole</key>
    <string>Editor</string>
    <key>CFBundleURLName</key>
    <string>kh.gov.camdx.camdigikey.clientdemo1</string>
    <key>CFBundleURLSchemes</key>
    <array>
      <string>kh.gov.camdx.camdigikey.clientdemo1</string>
    </array>
  </dict>
</array>
```

- Create a Protocol for CamDigiKey delegate functions

```
protocol CamDigiKeyClientDelegate {
  func didAuthorizedLoginRequest(authToken: String)
  func didFailAuthorizedLoginRequest(message: String)
}
```

- Create a View Controller class for awaiting screen (use when CamDigiKey App is opened)

```
class AwaitingAuthorizationVC: UIViewController {
  var delegate: CamDigiKeyClientDelegate?

  override func viewDidLoad() {
    super.viewDidLoad()

    // MARK : Add your own design for awaiting screen
  }
}
```

- Login with CamDigiKey
  - Inside the function which handles login button, request to client API server to get `loginToken`
  - Once the `loginToken` is responded, call `loginWithCamDigiKey` function

```
@IBAction func loginWithCamDigiKey(_ sender: UIButton) {
  // MARK : Step 1. Add your logic for requesting `loginToken` from Client API Server
  let loginToken = "LOGIN_TOKEN_RESPONDED_FROM_CLIENT_API_SERVER"
  // MARK : Step 2. Generate login request with `loginToken` to CamDigiKey App
  let url = URL(string: "camdigikey://login?token=\(loginToken)")
  UIApplication.shared.open(url!) { (result) in
    if result {
      let awaitingVC = AwaitingAuthorizationVC()
      awaitingVC.modalPresentationStyle = .fullScreen
      awaitingVC.delegate = self
      self.present(awaitingVC, animated: true, completion: nil)
    } else {
      // MARK : Add your logic to handle when CamDigiKey App can't be opened
    }
  }
}
```

- Once user authorized the login request in CamDigiKey app, an `authToken` will be returned and passed to client app via DeepLink. `LoginViewController` can get `authToken` by conform to `CamDigiKeyClientDelegate`.

```
extension LoginViewController: CamDigiKeyClientDelegate {
  func didAuthorizedLoginRequest(authToken: String) {
    print("Login success, authToken: \(authToken)")
    // MARK : Step 3. Add your logic to request for `accessToken` with `authToken` from Client
        API Server
    // MARK : Step 4. Add your logic to request for user information with `accessToken` from
        Client API Server
  }
  func didFailAuthorizedLoginRequest(message: String) {
    // MARK : Add your logic to handle authorization failed
    print("Login failed, message: \(message)")
  }
}
```

- Handling DeepLink for login callback
  - In `SceneDelegate`, override a function as following

```
extension SceneDelegate {
  func scene(_ scene: UIScene, openURLContexts URLContexts: Set<UIOpenURLContext>) {
    let rootVC = window?.rootViewController
    if let awaitingVC = rootVC?.presentedViewController as? AwaitingAuthorizationVC {
      if let urlContext = URLContexts.first, let host = urlContext.url.host, host ==
          "camdigikey_login_callback" {
        let data = URLComponents(url: urlContext.url, resolvingAgainstBaseURL:
            false)?.queryItems?.first
        if data?.name == "authToken" {
          awaitingVC.dismiss(animated: true) {
            if let authToken = data?.value {
              awaitingVC.delegate?.didAuthorizedLoginRequest(authToken: authToken)
            } else {
              awaitingVC.delegate?.didFailAuthorizedLoginRequest(message: "No Auth Token")
            }
          }
        } else {
          awaitingVC.dismiss(animated: true) {
            awaitingVC.delegate?.didFailAuthorizedLoginRequest(message: data?.value ?? "no
                errorMessage")
          }
        }
      }
    } else {
      // MARK : Add your logic to handle when login request has been canceled
    }
  }
}
```

2.5.2. Android

a. Get `loginToken` from your own server:

b. Handle callback from Login with CamDigiKey:

```
val resultLauncher =
    registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { result ->
        if (result.resultCode == Activity.RESULT_OK) {
            findViewById<TextView>(R.id.tvMessage).text = "Approved: ${result.data.toString()}"
            val authToken = result.data?.getStringExtra("authToken")
            Log.d("authToken =", "$authToken")
            val i = Intent(this, ClientInfoActivity::class.java)
            val flags = Intent.FLAG_ACTIVITY_CLEAR_TASK or Intent.FLAG_ACTIVITY_NEW_TASK
            flags.let { i.flags = it }

            i.putExtra("authToken", authToken)
            startActivity(i)
        } else {
            findViewById<TextView>(R.id.tvMessage).text = "Rejected: ${result.data.toString()}"
        }
    }
```

c. Setup Login with CamDigiKey on your activity:

```
findViewById<Button>(R.id.btSendRequest).setOnClickListener {
    getLoginToken()
    Log.d("LoginToken", "${loginToken}")
    val intent = Intent(Intent.ACTION_ASSIST).apply {
        data = Uri.parse("camdigikey://login?token=${loginToken}")
        setPackage("kh.gov.camdx.camdigikey.debug")
        putExtra(Intent.EXTRA_ASSIST_PACKAGE, BuildConfig.APPLICATION_ID )
    }

    try {
        resultLauncher.launch(intent)
    } catch (e: Exception) {
        Toast.makeText(this, "Please install CamDigiKey app", Toast.LENGTH_SHORT).show()
    }
}
```

3. Open KYC Verification APIs

Open KYC verification APIs are available over CamDX (Cambodia Data eXchange) which is an interoperability platform of Cambodia government.

The KYC verification APIs consist of 5 APIs:

3.1. Verify User Info

Verify user info with data from MOI	
End Point	/api/2.0/kyc/info
Method	POST

Format	JSON

Request	{
	    "idNumber" : "id_number",
	    "firstNameKh" : "first_name_kh",
	    "lastNameKh" : "last_name_kh",
	    "firstNameEn" : "first_name_en",
	    "lastNameEn" : "last_name_en",
	    "gender" : "M_or_F",
	    "dob" : "yyyy-MM-dd",
	    "issuedDate" : "yyyy-MM-dd",
	    "expiredDate" : "yyyy-MM-dd"
	}

Response	{
	    "error": 0,
	    "message": "Successfully",
	    "data": {
	        "idNumber": "id_number",
	        "score": 1, #Range [0,1]
	        "incorrectFields": [requested_fields_or_empty]
	    }
	}

3.2. Verify User Info and User Face

Verify user info and user's face with data from MOI	
End Point	/api/2.0/kyc/info-face
Method	POST
Format	JSON

Request	{
	    "userInfo" : {
	        "idNumber" : "id_number",
	        "firstNameKh" : "first_name_kh",
	        "lastNameKh" : "last_name_kh",
	        "firstNameEn" : "first_name_en",
	        "lastNameEn" : "last_name_en",
	        "gender" : "M_or_F",
	        "dob" : "yyyy-MM-dd",
	        "issuedDate" : "yyyy-MM-dd",
	        "expiredDate" : "yyyy-MM-dd"
	    },
	    "faceImg" : "base_64_content"
	}

Response	{
	"error": 0,
	"message": "Successfully",
	"data": {
	"userInfo": {
	"idNumber": "id-card-number",
	"score": 1, #Range [0,1]
	"incorrectFields": []
	},
	"faceMoiScore": 0.9792682 # Range [0,1]
	}
	}

3.3. Verify User Info, ID Card and User Face

Verify user info, face on ID card, and MOI data	
End Point	/api/2.0/kyc/info-face-idcard
Method	POST
Format	JSON

Request	{
	"userInfo" : {
	"idNumber" : "id_number",
	"firstNameKh" : "first_name_kh",
	"lastNameKh" : "last_name_kh",
	"firstNameEn" : "first_name_en",
	"lastNameEn" : "last_name_en",
	"gender" : "M_or_F",
	"dob" : "yyyy-MM-dd",
	"issuedDate" : "yyyy-MM-dd",
	"expiredDate" : "yyyy-MM-dd"
	},
	"faceImg" : "base_64_content",
	"idImage" : "base_64_content"
	}

Response	{
	    "error": 0,
	    "message": "Successfully",
	    "data": {
	        "userInfo": {
	            "idNumber": "id_number",
	            "score": 1, #Range[0,1]
	            "incorrectFields": []
	        },
	        "faceDocumentScore": 0.9457877, #Range[0,1]
	        "faceMoiScore": 0.9792682 #Range[0,1]
	    }
	}

3.4. Verify User Info, ID Card Image, and Liveness

Verify user info, ID card image, liveness video with MOI data	
End Point	/api/2.0/kyc/info-idcard-liveness
Method	POST
Format	JSON

Request	{
	#1: turn face to the left, 2: turn face to the right, 3: nod the head
	    "actions":["2","1","3"],
	    "userInfo" : {
	        "idNumber" : "id_number",
	        "firstNameKh" : "first_name_kh",
	        "lastNameKh" : "last_name_kh",
	        "firstNameEn" : "first_name_en",
	        "lastNameEn" : "last_name_en",
	        "gender" : "M_or_F",
	        "dob" : "yyyy-MM-dd",
	        "issuedDate" : "yyyy-MM-dd",
	        "expiredDate" : "yyyy-MM-dd"
	    },
	    "idImage" : "base_64_content",
	    "liveness" : "base_64_content"
	}

Response	{
	    "error": 0,
	    "message": "Successfully",
	    "data": {
	        "userInfo": {
	            "idNumber": "010682723",
	            "score": 1,

	"incorrectFields": [] }, "livenessScore": 0.9248705, "faceDocumentScore": 0.94440883, "faceMoiScore": 0.9769943 } }

3.5. Khmer ID Card OCR

Extract user information from ID card image	
End Point	/api/2.0/kyc/ocr-idcard
Method	POST
Format	JSON

Request	{
	"idImage" : "base_64_content"
	}

Response	{
	"error": 0,
	"message": "Successfully",
	"data": {
	"idNumber" : "id_number",
	"firstNameKh" : "first_name_kh",
	"lastNameKh" : "last_name_kh",
	"firstNameEn" : "first_name_en",
	"lastNameEn" : "last_name_en",
	"gender" : "M_or_F",
	"dob" : "yyyy-MM-dd",
	"issuedDate" : "yyyy-MM-dd",
	"expiredDate" : "yyyy-MM-dd"
	"MRZ1" : "String",
	"MRZ2" : "String",
	"MRZ3" : "String",
	}
	}

4. Contact us

Email: camdx-info@techostartup.center

Address: RUPP's Compound Russian Federation Blvd., Toul Kork, Phnom Penh, Cambodia