



الهيئة العامة للمعلومات المدنية
The Public Authority For Civil Information



Kuwait National Mobile ID

Service Provider Integration Guide

Version 2.1 – September 2020

CONFIDENTIALITY STATEMENT

This document contains proprietary and confidential information. All data and material enclosed are The Public Authority for Civil Information's (PACI) sole property and not to be reproduced in any way without PACI's written consent.

CONFIDENTIAL

REVISION HISTORY

Document Revision History					
Document Revision	Description of Change	Author	Reviewed By	Target SDK Version	Effective Date
1.0	First Version	M. B.	M.T	v1.1.0	12/05/2019
2.0	MID SDK (v1.3.0) Second Version	Y. M.	M.B, M.T	v1.3.0	20/06/2020
2.1	MID SDK (v1.3.1) Second Version	Y. M.	M.B	V1.3.1	18/09/2020

CONFIDENTIAL

TABLE OF CONTENT

1	Introduction.....	5
1.1	Definitions, Acronyms, and Abbreviations	5
2	Mobile ID Service Overview	6
3	Mobile ID Integration Scenarios	7
3.1	Strong Authentication using QR Scanning	7
3.2	Strong Authentication using URL Scheme	8
3.3	Strong Authentication using Push Notification.....	9
3.4	Digital Signature using Push Notification.....	10
3.5	Verify user identity using QR code	11
3.6	Notify user and sending messages using push notification	12
3.7	Check if user has mobile identity and its verification level.....	13
4	Mobile ID Service Integration SDK	14
4.1	SDK Content.....	14
4.2	Setup & Configuration.....	14
4.2.1	Registration Details	14
4.2.1.1	Assurance Levels	14
4.2.2	Pre-Requisites	15
4.3	Integration Flow.....	16
4.4	Included APIs.....	17
4.4.1	Mobile ID Client SDK API	17
4.4.1.1	Constructor	17
4.4.1.2	Initiate Authentication Request QR.....	18
4.4.1.3	Initiate Authentication Request PN	20
4.4.1.4	Initiate Signing Request	22
4.4.1.5	Get User Certificate	23
4.4.1.6	Verify Mobile ID QR Code	24
4.4.1.7	Notify person.....	26
4.4.1.8	Check if end user has mobile ID	27
4.4.2	Request Call-back	28
4.4.3	Tools.....	32
4.4.3.1	Client Certificate Generator Tool.....	32
4.4.3.2	Certificate Generation Steps.....	32
4.5	Mobile ID URL Scheme API	33
4.5.1	Integration Flow.....	33
4.5.2	API Documentation.....	33
	Appendix I: Code Samples.....	34

1 Introduction

The Kuwait Mobile ID is a secure mobile-based Digital ID derived from the Civil ID. It can be used for identity verification, Authentication to online e-services, applying trusted digital signature to documents and transaction, and receiving notification messages from different service providers.

Various Service Providers (SPs) including Government agencies, Oil Sector Companies, Banks, Private Companies, and others can integrate their internal and public e-services with the Mobile ID Service to incorporate the aforementioned authentication and signing capabilities.

This document is intended to describe the integration process and API specifications

1.1 Definitions, Acronyms, and Abbreviations

API: Application Programming Interface

Android: Android Mobile Operating System

AR: Arabic

CA: Certificate Authority

DB: Database

EN: English

iOS: Apple Mobile Operating System

MID: Kuwait Mobile ID

OS: Operating System

PACI: Public Authority for Civil Information

PIN: Personal Identification Number

RAM: Random Access Memory

SDK: Software Development Kit

SOAP: Simple Object Access Protocol

SSL: Secure Socket Layer

SP: Service Provider

TLS: Transport Layer Security

TCP: Transmission Control Protocol

2 Mobile ID Service Overview

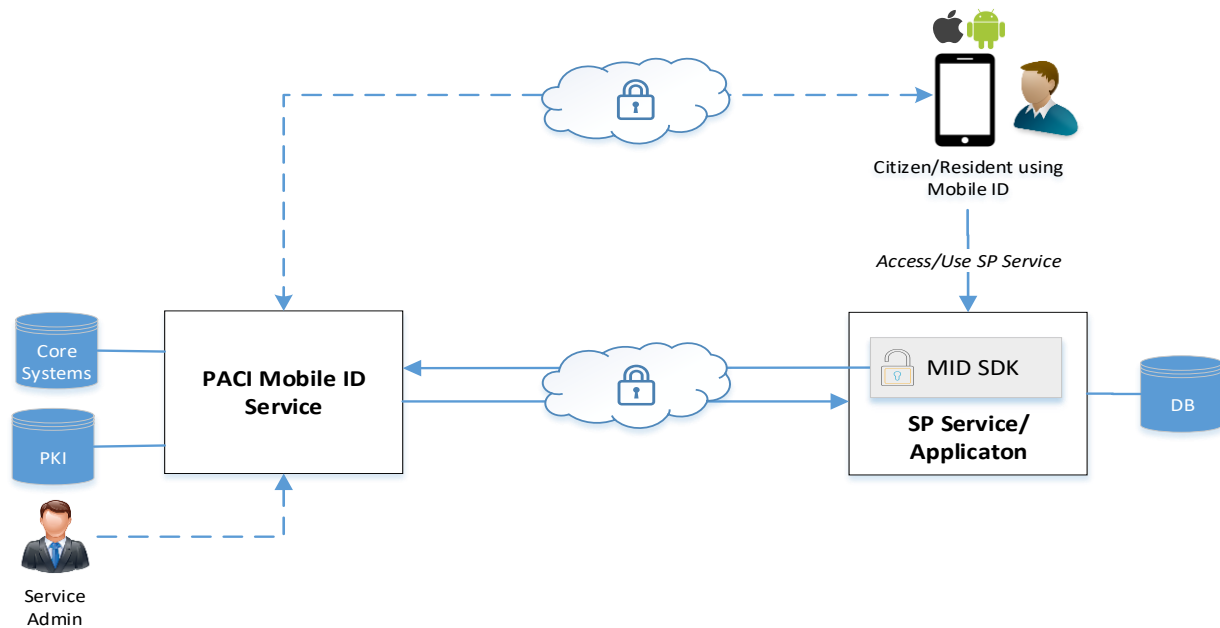


Figure 1 – Mobile ID Service Overview

- PACI MID System is responsible for managing the mobile based digital identities for the citizens and residents and also allows citizens and residents to use their Mobile IDs for Identification, Authentication Digital Signature, and Receiving notifications messages. It also allows different Service Providers (SPs) to integrate their applications and eServices with PACI MID.
- A Citizen/Resident is required to have mobile device with MID App installed from App Store or Play Store. The user's device needs to be enrolled into the Mobile ID Service prior to any usage.
- SP Service represents the SP application or eService which is required to enable user authentication and digital signing of enrolled users using Mobile ID. In order for the service provider to do that, the service provider system should communicate with PACI MID System through PACI MID SDK. The PACI MID SDK provides APIs to allow easy and secure integration with the Mobile ID Service.
- PACI Service Admin is responsible for registering any new service provider to the system.

3 Mobile ID Integration Scenarios

The Mobile ID Service providers (SPs) will have different approaches for integration. Following is a list of the current supported approaches:

- Strong Authentication using QR Scanning
- Strong Authentication using URL Scheme (for Mobile Native Apps & Mobile Browsers)
- Strong Authentication using Push Notification
- Digital Signature using Push Notification
- Verify user identity using QR code
- Notify user by using messages and notifications
- Check if user has mobile identity and its verification level

3.1 Strong Authentication using QR Scanning

In this mode, the user (Mobile ID holder) authenticates to the SP eService/Application by scanning a QR code shown on the login screen.

When the user wants to authenticate to the SP eService/Application, the SP eService/Application obtains a QR code from The Mobile ID Service through the MID SDK and presents it to the user. The user scans the QR code through the MID App and authorizes the operation by entering the PIN code. The MID App utilizes the user's certificate to sign the request and submit it to the MID Service which in turn verifies the transaction and sends a callback to the SP eService/Application including the authorization and user identity details.

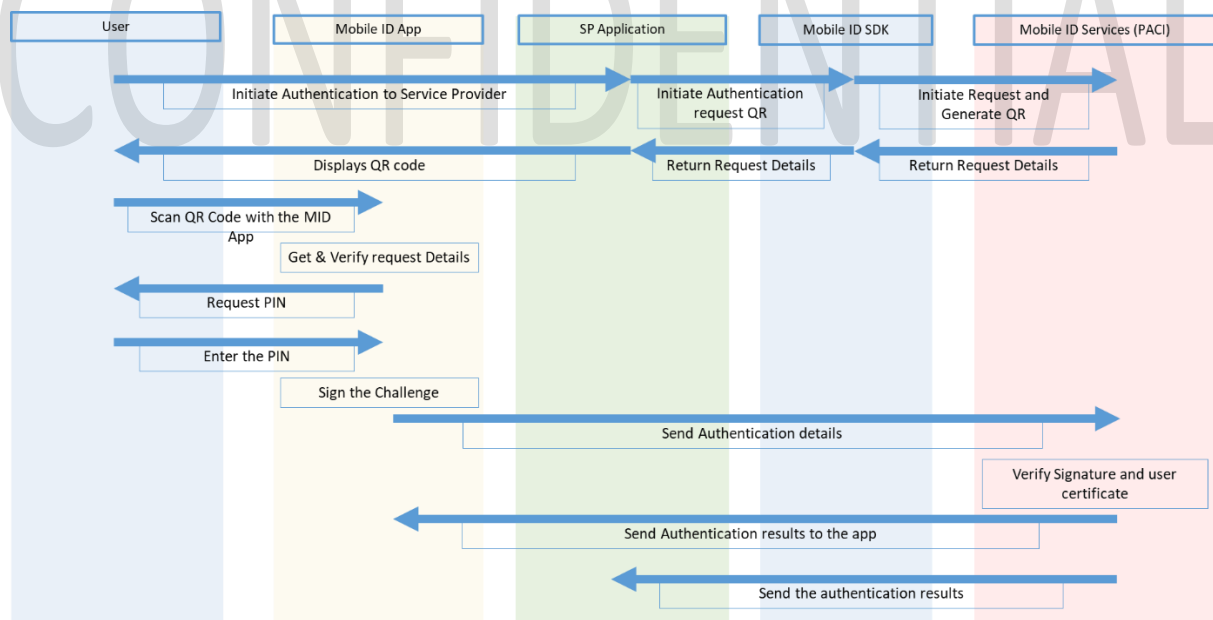


Figure 2 – Process Flow - Authentication through QR Code Scanning

3.2 Strong Authentication using URL Scheme

In this mode, the user (Mobile ID holder) is using his/her mobile device (SP Native App or SP Mobile-friendly Web App) and authenticates to the SP eService/Application by integrating directly with Mobile ID App.

When the user wants to authenticate to the SP eService, the SP eService/Application obtains a unique URL Scheme from The Mobile ID Service through the MID SDK and presents it to the user as a button. When the user presses the button, the Mobile ID App opens and shows the request details to the user who authorizes the operation by entering the PIN code. The Mobile ID App utilizes the user's certificate to sign the request and submit it to the MID Service which in turn verifies the transaction and sends a callback to the SP eService/Application including the authorization and user identity details.

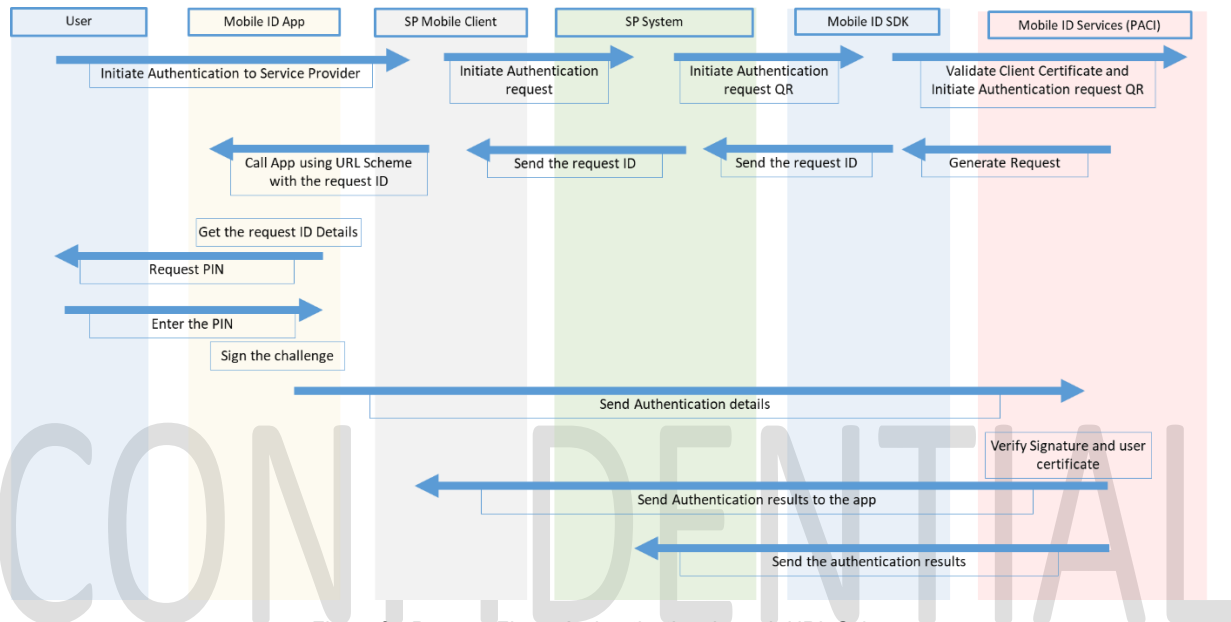


Figure 3 - Process Flow - Authentication through URL Scheme

3.3 Strong Authentication using Push Notification

In this mode, the user (Mobile ID holder) gets a push notification from the Service Provider service/application asking him to authorize a certain transaction. Once the Mobile ID App receives the notification sent by Mobile ID service, The Mobile ID App presents the details of the request. The user then accepts the request by tapping "Accept" button and enters the PIN code. The Mobile ID App utilizes the user's certificate to sign the request and submit it to the MID Service which in turn verifies the transaction and sends a callback to the SP eService /Application including the authorization and user identity details.

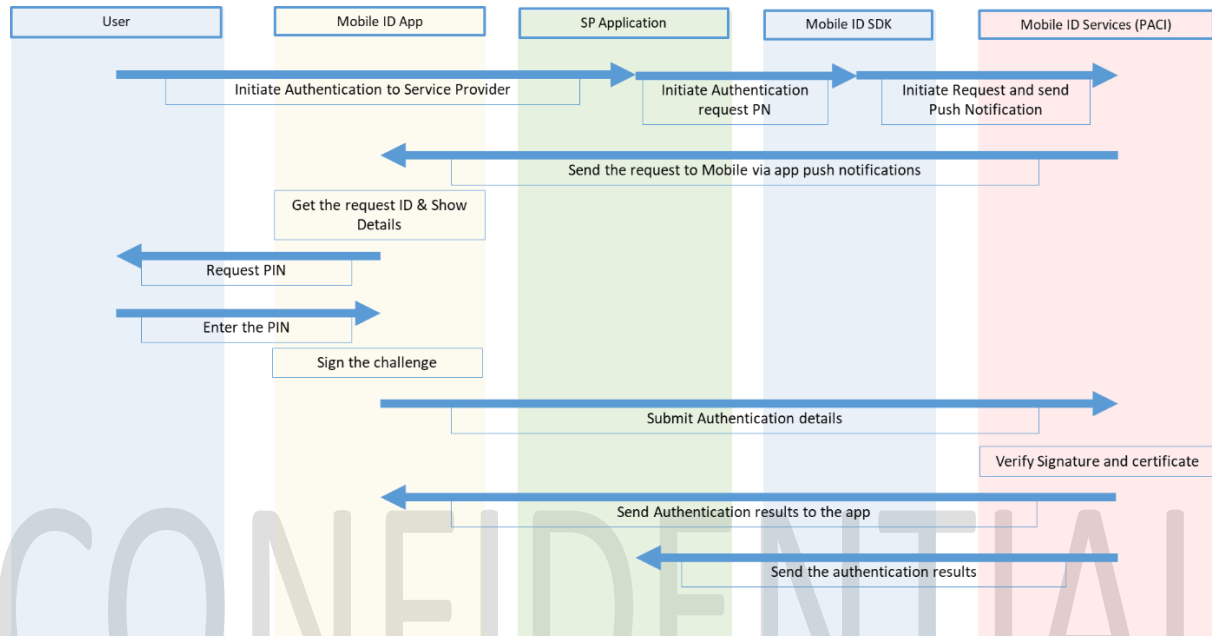


Figure 4 - Process Flow - Authentication through Push Notification

3.4 Digital Signature using Push Notification

For applying digital signing of data using the Mobile ID, the Service Provider needs to have a Digital Signature System which is capable of computing the hash of the document or data to be signed, sending the hash to PACI Mobile ID Service through the Mobile ID SDK, and completing the signing process once the signed hash is returned.

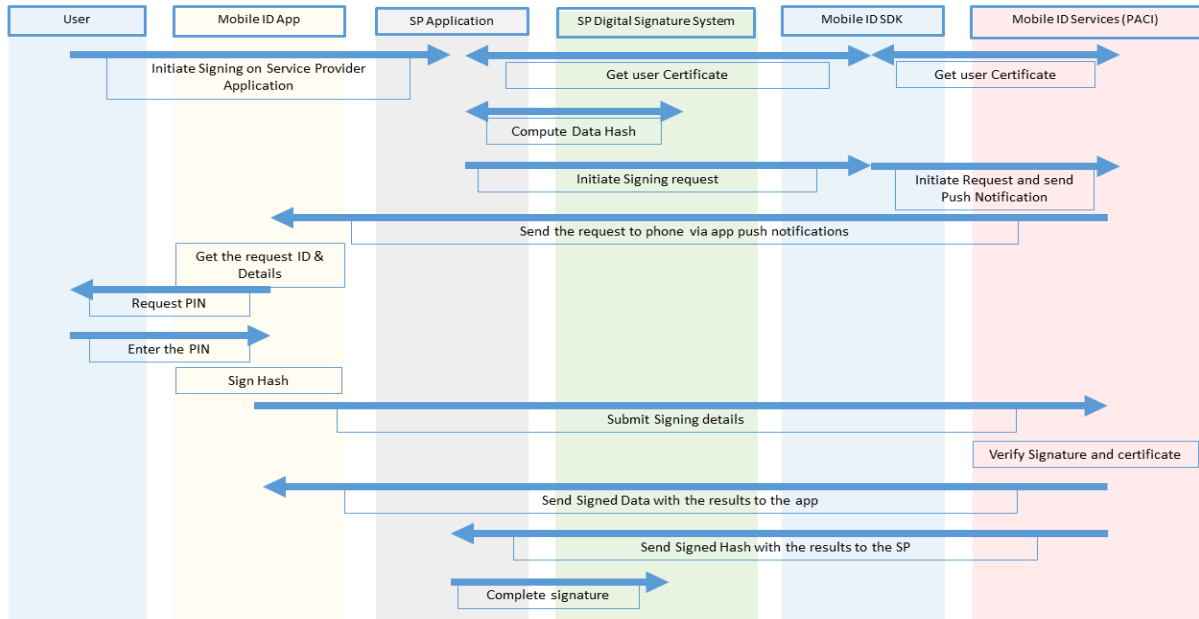


Figure 5 - Process Flow - Digital Signature using Push Notification

3.5 Verify user identity using QR code

In this mode, the SP can verify the end user's identity and retrieve his/her personal data by scanning the QR code shown in his/her Kuwait Mobile ID APP.

When the SP wants to verify the user identity and retrieve his/her personal data, the user will open the QR Code screen from Kuwait Mobile ID on his/her phone and the SP will scan the generated QR Code. The SP eService/Application will send the QR Code value to the Mobile ID Service through the MID SDK which in turn verifies the user's identity and returns his/her personal data to the SP.

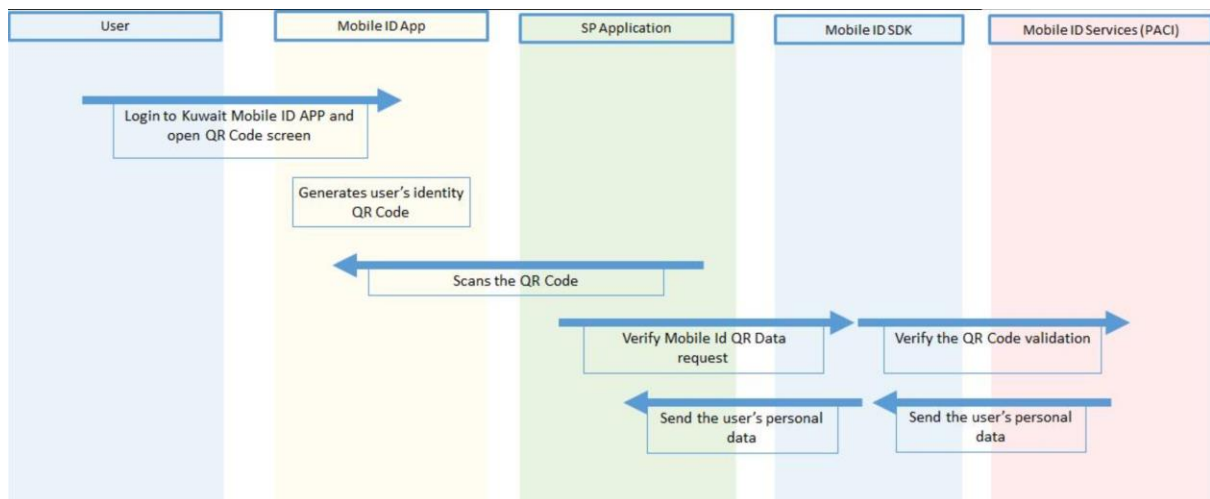


Figure 6 - Process Flow - Verify user identity using QR code

3.6 Notify user and sending messages using push notification

In this mode, the SP can notify an end user and send him/her a message on the Kuwait Mobile ID APP, the user (Mobile ID holder) gets a push notification from the Service Provider service/application that displays the message subject and the user can view the message from the Kuwait Mobile ID APP.

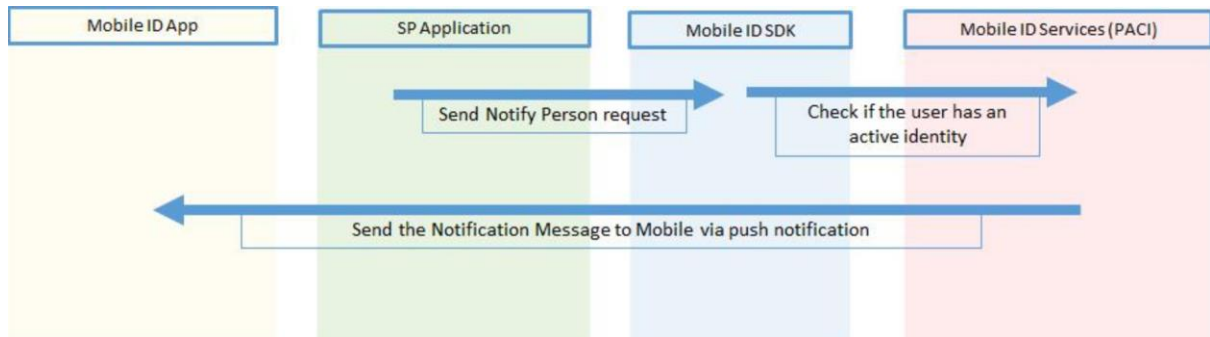


Figure 7 - Process Flow - Notify user by using messages and notifications

CONFIDENTIAL

3.7 Check if user has mobile identity and its verification level

In this mode, the SP can check if a specific end user has a valid identity on Kuwait Mobile ID APP.

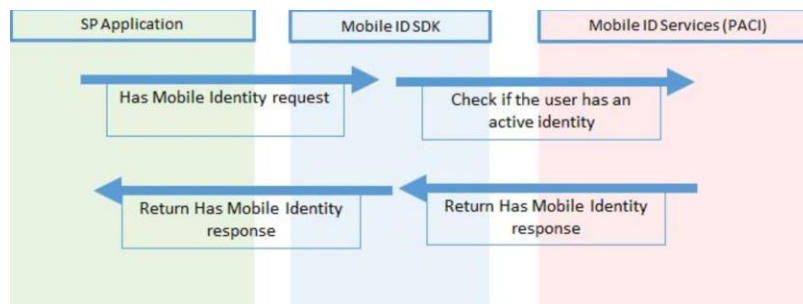


Figure 8 - Process Flow - Check if user has mobile identity and its verification level

CONFIDENTIAL

4 Mobile ID Service Integration SDK

4.1 SDK Content

The Mobile ID SDK contains the following folders:

- **Bin** - Contains the following files:
 - *MIDAuthServiceClient.dll* – A .NET Library including the API to be referenced by the Service Provider Application
 - *Other Binaries & Dependencies*
- **Documents** - Contains Mobile ID SP SDK Documentation
- **Tools** - Contains the following tools
 - MID Client Certificate Manager
- **Samples** - Sample SP client code

4.2 Setup & Configuration

4.2.1 Registration Details

A service provider needs to be registered with PACI for Mobile ID Services before starting to use the SDK. The following needs to be provided to PACI:

- Name of Service Providing Entity (English & Arabic)
- List of eServices to be provided (Name & Description in English & Arabic)
- Minimal list of server IP Addresses accessing the service
- SP Logo (PNG Format, 512x512)
- Callback BaseURL
- Contact person info (name, phone number, and email)
- Default assurance level (low, medium, or high)

4.2.1.1 Assurance Levels

The Assurance Level is a confidence level that the SP demands when using Mobile ID authentication or signing services. It is based on the channel that the end-user used to enroll for a Mobile ID. Therefore, the Assurance Level is limited by the identity verification level used to verify the identity of the Mobile ID holder.

Assurance Level	Enrollment Channel	Recommended Transactions/Use-Cases
Low	Online Enrollment	Face-to-Face identity verification (i.e. Verify ID with Mobile ID generated QR Code)
Medium	Upgraded Online Enrollment	Online transactions with medium or low risk and impact (e.g. online login, enquiries, service access, ...)
High	Self-Service Kiosk or Attending at PACI Service Centers	Online transactions with high risk and greater impact (e.g. transactions which have financial impact, contract signing, ...)

SPs will have two options either setting the required user's assurance level in the method's parameter (which require assurance level) or depending on their default required assurance level that they have provided to PACI. In case the SP didn't set the assurance level while initiating the request and there was no default value for this SP the value will be set to "High" by default.

The user's assurance level must be equal or higher than the provided assurance level by the SP in order to be able to authorize the auth/sign request.

4.2.2 Pre-Requisites

The SP system that integrates with the Mobile ID SP SDK requires the following prerequisites

- *Operating System*
 - Windows Server 2012 or later (Recommended)
- *.Net Framework Version*
 - .NET Framework 4.0 or higher
- *Admin Rights*
 - the application user running the SP application integrating with Mobile ID SDK must have access to server certificate store (*Local machine – Personal*)
- *Trust & Accessibility*
 - Service provider must install a TLS client certificates issued by PACI in order to authenticate SDK API calls
 - Kuwait National Root certificate must be installed in the trusted store in the SP Server
 - Service Provider must provide the server IPs to be allowed by PACI Mobile ID Service
 - Service Provider Callback Handler must be SSL secure and accessible from within PACI Mobile ID servers
- *Additional Notes*
 - Any needed SDK operation must be approved and allowed by PACI before the SP is able to use it.

4.3 Integration Flow

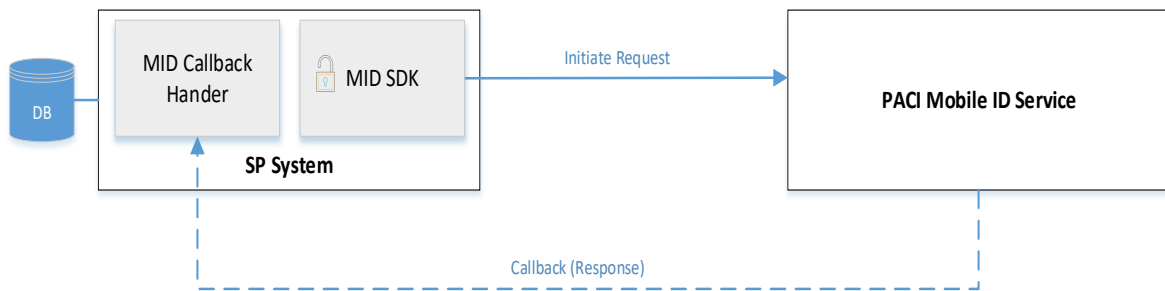


Figure 9 – Integration Flow

- The service provider will need to create and deploy a RESTful web service (*MID Service Callback Handler*) that accepts and processes callbacks from the Mobile ID Service.
- The SP's MID Service Callback Handler web service shall expect a schema as defined in section 4.4.2.
- If digital signing is required, the service provider needs to have a signing solution which is capable of computing the hash of the document or data to be signed, sending the hash to PACI Mobile ID Service through the Mobile ID SDK, and completing the signing process once the signed hash is returned.
- Note: Some operations are Synchronous such as *Verify Identity*, *Send Notification*, and *Has Mobile ID*

CONFIDENTIAL

4.4 Included APIs

This section describes the different integration API provided by the Mobile ID SDK to the service providers.

4.4.1 Mobile ID Client SDK API

This is an API available to server-side applications and allows integrating strong authentication, digital signing, validating verifying user identity, notifying user, and checking if user has mobile identity capabilities into the service provider's application

4.4.1.1 Constructor

```
public MIDAuthServiceClient (
    X509Certificate2 serviceProviderClientCertificate,
    string midServiceHostName,
    string midServicePort
)
```

- **Summary:**

Initializes a new instance of the MIDAuthServiceClient Class

- **Parameters:**

- Service provider client certificate,
- MID authentication service host name
- MID authentication service communication port

- **Description:**

Instantiates a new instance of the service client class

4.4.1.2 Initiate Authentication Request QR

```
public MIDAuthSignClientResponse<AuthenticationQRData> InitiateAuthRequestQR(
    AuthenticateRequest authRequestData
)
```

Overloaded

```
public MIDAuthSignClientResponse<AuthenticationQRData> InitiateAuthRequestQR(
    AssuranceLevel assuranceLevel, AuthenticateRequest authRequestData
)
```

- Summary**

- Initiates a new authentication request and return base64 encoded QR Image in PNG format.
- The minimum user's assurance level required for authentication is medium.
- In case of using option 1 the assurance level value will be set to the SP default assurance level that the SP requests from PACI.
- By using option 2 the service provider can set the assurance level to medium or high.
- Meeting the assurance level depends on the verification level carried out during the end user enrollment process. Please referee to section 4.2.1.1

- Input Parameters**

Authenticate Request			
Property Name	Description	Mandatory	Length
Service Provider ID	GUID representing the identity of the service provider	Yes	150
Service Description EN	English Description of the provided service	Yes	150
Service Description AR	Arabic Description of the provided service	Yes	150
Authentication Reason En	English Description of the authentication reason	Yes	MAX
Authentication Reason Ar	Arabic Description of the authentication reason	Yes	MAX
Additional Data	Additional Data provided by the service provided and it will be returned in the callback, UDF	No	1000
Challenge	Optional and can be passed by the service provider	No	500
SP Callback URL	Call back URL passed by the service provider and it must be restful service	No	250
Request User Details	Boolean values indicate if the service providers need to retrieve the end user data	No	Bool (default value is false)

Assurance Level (Overloaded)			
Property Name	Description	Mandatory	Length
Assurance Level	Enum value that declare the required assurance level by the end user. Note: "Low" assurance level cannot be used in the current version of the SDK.	NA	Enum (Low, Medium, High)

- **Output Parameters**

Authentication QR Data	
Property Name	Description
QR Code Image	Base64 Encoded image of the QR Data
QR Code	The ID of the created authentication request

- **Description**

- This call initiates a request on the Mobile ID Service after applying all validations.
- QR data will be returned to SP service and should be presented to the end user to be scanned through the Mobile ID App.
- The authentication request is valid for 5 minutes only before it gets expired.
- The callback will be sent asynchronously once the end-user provides the needed authorization.

CONFIDENTIAL

4.4.1.3 Initiate Authentication Request PN

```
public MIDAuthSignClientResponse<string> InitiateAuthRequestPN(
    AuthenticateRequest authRequestData
)
```

Overloaded

```
public MIDAuthSignClientResponse<string> InitiateAuthRequestPN(
    AssuranceLevel assuranceLevel, AuthenticateRequest authRequestData
)
```

- **Summary**

- Initiate new authentication request and push notification to the user mobile
- The minimum user's assurance level required for authentication is medium.
- In case of using option 1 the assurance level value will be set to the SP default assurance level that was provided to PACI.
- By using option 2 the service provider can set the assurance level to medium or high.
- Meeting the assurance level depends on the verification level carried out during the end user enrollment process. Please referee to section 4.2.1.1

- **Input Parameters**

Authenticate Request			
Property Name	Description	Mandatory	Length
Service Provider ID	GUID representing the identity of the service provider	Yes	150
Service Description EN	English Description of the provided service	Yes	150
Service Description AR	Arabic Description of the provided service	Yes	150
Authentication Reason En	English Description of the authentication reason	Yes	MAX
Authentication Reason Ar	Arabic Description of the authentication reason	Yes	MAX
Additional Data	Additional Data provided by the service provided and it will be returned in the callback, UDF	No	1000
Challenge	Optional and can be passed by the service provider	No	500
SP Callback URL	Call back URL passed by the service provider and it must be restful service	No	250
Person Civil No	User Civil No to get the mobile device and push the notification to it	Yes	50
Request User Details	Boolean values indicate if the service providers need to retrieve the end user data	No	Bool (default value is false)

Assurance Level Overloaded			
Property Name	Description	Mandatory	Length
Assurance Level	Enum value that declare the required assurance level by the end user. Note: "Low" assurance level cannot be used in the current version of the SDK.	NA	Enum (Low, Medium, High)

- **Output Parameters**

Request ID	
String	The ID of the created authentication request

- **Description**

- This call initiates a request on the Mobile ID Service after applying all validations.
- The end user gets a push notification from the SP Service asking him to authenticate
- The authentication request is valid for 5 minutes only before it gets expired.
- The callback will be sent asynchronously once the end-user provides the needed authorization.

CONFIDENTIAL

4.4.1.4 Initiate Signing Request

```

public MIDAuthSignClientResponse<string> InitiateSigningRequest (
    SigningRequest signingRequest
)
    
```

- **Summary**

- Initiate new signing request and push notification to the user mobile
- The minimum user's assurance level required for authentication is high.
- Meeting the assurance level depends on the verification level carried out during the end user enrollment process. Please referee to section 4.2.1.1

- **Input Parameters**

Signing Request			
Property Name	Description	Mandatory	Length
Service Provider ID	GUID representing the identity of the service provider	Yes	150
Prompt EN	English Description of the provided service	Yes	150
Prompt AR	Arabic Description of the provided service	Yes	150
Signing Reason En	English Description of the Signing reason	Yes	MAX
Signing Reason Ar	Arabic Description of the Signing reason	Yes	MAX
Additional Data	Additional Data provided by the service provided and it will be returned in the callback, UDF	No	1000
Challenge	Hash of the document to be signed, must be passed by the service provider	Yes	1000
SP Callback URL	Call back URL passed by the service provider and it must be restful service	No	250
Person Civil No	User Civil No to get the mobile device and push the notification to it	Yes	50
Preview File	binary of document to signed to preview on the mobile	No	MAX
File Type	Files Types can be "application/pdf" or "application/XML"	No	150
File Name	Name of the file	No	150
Request User Details	Boolean values indicate if the service providers need to retrieve the end user data	No	Bool (default value is false)

- **Output Parameters**

Request ID	
String	The ID of the created Sign request

- **Description**

- This call initiates a request on the Mobile ID Service after applying all validations.
- The end user gets a push notification from the SP Service asking him to sign
- The Signing request is valid for 5 minutes only before it gets expired.
- The callback will be sent asynchronously once the end-user provides the needed authorization.

4.4.1.5 Get User Certificate

```
public MIDAuthSignClientResponse<string> GetUserCertificate(
    string serviceProviderId,
    string civilNo
)
```

- **Summary**
Get User Mobile ID Certificate for the use of digital signature

- **Input Parameters**

Property Name	Description	Mandatory	Length
Service Provider ID	GUID representing the identity of the service provider	Yes	150
Civil No	User Civil No to get the mobile ID Certificate	Yes	50

- **Output Parameters**

User's Certificate	
String	The user's certificate

- **Description**
 - This call Gets the Mobile ID Certificate of specified user and return it back to the SP Service

4.4.1.6 Verify Mobile ID QR Code

```

public MIDAuthSignClientResponse<IdentityDetails> VerifyMobileIdQRData (
    string serviceProviderId,
    string qrData
)
    
```

- Summary**
 Synchronously gets personal data for identity verification QR.

- Input Parameters**

Property Name	Description	Mandatory	Length
Service Provider ID	GUID representing the identity of the service provider	Yes	150
QR Data	QR Decoded Data from the Kuwait Mobile ID verification screen	Yes	50

- Output Parameters**

Person Identity Data	
Property Name	Data Type
Civil ID	String
Full Name AR	String
Full Name EN	String
Nationality AR	String
Nationality EN	String
Birth Date	String
Gender	String
Mobile Number	String
Email Address	String
Blood Group	String
Photo	String / Base64
Card Expiry Date	DateTime
DocumentNumber	String
PassportNumber	String
Gov Id Data	Object
Gov Id Data- Civil Id Expiry Date	Date Time
Gov Id Data- MOI Reference	String
Gov Id Data- Sponsor Name	String
Address	Object
Address - Governorate	String
Address - Area	String
Address - PACI Building Number	String
Address - Building Type	String
Address - Floor Number	String
Address - Building Number	String
Address - Block Number	String
Address - Unit Number	String
Address - Street Name	String
Address - Unit Type	String

Additional Output	
Property Name	Data Type
Data Signing Certificate	Byte[]
Data Signature	String

- **Description**
 - This call verifying the Mobile ID verification QR and return the personal data
 - The ID verification request is valid for 1 minute only before it gets expired.
- **Object Validation**

The received object contains DataSignature field that holds the signed value of the PersonalData object and it can be used to validate the data integrity and nonrepudiation.

Object validation can be achieved by following the below steps:

1. Create an object that contains the personalData object (fields ordering should follow the same ordering of the Person Identity Data table)
2. Get the DataSigningCertificate from the received object and compare it to the signing certificate that will be provided by PACI with the package.
3. Get the public key of the DataSigningCertificate.
4. Compute hash for the generated object in step 1 using sha256.
5. Use the public key to verify the computed hash against the DataSignature value from the received object.

CONFIDENTIAL

4.4.1.7 Notify person

```
public MIDAuthSignClientResponse<string> NotifyPerson(
    NotificationRequest notificationRequest
)
```

- **Summary**

Sends a message to the end user and push notification.

- **Input Parameters**

The input parameter of type NotificationRequest consists of the following attributes:

Notification Request			
Property Name	Description	Mandatory	Length
Service Provider ID	GUID representing the identity of the service provider	Yes	150
Person Civil No	User Civil No to get the mobile device and push the message to it	Yes	50
Notification Subject En	English displayed notification subject	Yes	150
Notification Subject Ar	Arabic displayed notification subject	Yes	150
Notification Message En	English message text	Yes	MAX
Notification Message Ar	Arabic message text	Yes	MAX
Notification Options	Passes options that control the properties of the notification being sent	No	Dictionary<String, String>

- **Description**

- The end user gets a push notification from the SP Service With the message subject.
- The end user can view the message from Kuwait Mobile ID application.

4.4.1.8 Check if end user has mobile ID

```

public MIDAuthSignClientResponse<HasMobileIdResponse> HasMobileIdentity(
    string serviceProviderId,
    string civilNo
)
    
```

- **Summary**
Check if the specified user has active identity.

- **Input Parameters**

Property Name	Description	Mandatory	Length
Service Provider ID	GUID representing the identity of the service provider	Yes	150
Civil No	User Civil No to get the mobile ID Certificate	Yes	50

- **Output Parameters**

Property Name	Data Type
HasMobileId	bool
Civil No	String
Status	String
Verification Level	Enum

- **Description**
 - This call checks the user identity and incase the user has active identity it returns the user's Assurance level.

4.4.2 Request Call-back

- **Description:**

Callback schema are JSON messages posted to the RESTful web service which the service provider will create and provide to handle the result feedback to authentication and signing requests.

- **Callback JSON Schema:**

```

{
  "MIDAuthSignResponse": {
    "RequestDetails": {
      "RequestID": "7a80c36f-95d1-4406-889e-13adc9fecf61",
      "RequestType": "Authentication",
      "ServiceProviderId": "1",
      "Challenge": "{ \"ServiceProviderID\": \"1\", \"Challenge\": \"8924fe62-4069-4609-acc7-564aced0e957\", \"Datetime\": \"7/6/2020 11:58:45 PM\", \"Prompt\": \"PN APP\" }\", \"CivilNo\": \"123456789012\" }",
      "ResultDetails": {
        "ResultCode": "1",
        "UserAction": "1",
        "UserCivilNo": "123456789012",
        "UserCertificate": "MIIFDzCCA/egAwIBAgIQCbWInLKPlxZDokKZmKXdxTANBgqhkiG9w0BAQsFADBIMQswCQYDVQQGEwJLVzEzMDEGA1UEChMqVGHlIFB1YmXpYyBBdXRob3JpdHkgZm9yIENpdmlsIEluZm9ybWw0aW9uMR4wHAYDVQQDEsVQQUNJIERlbW9uc3RyYXRpb24gQ0EwHhcNMjAwMDAwMDAwWWhcNMjAwMTAxMjM1OTU5WjB8MQswCQYDVQQGEwJLVzEzMDEGA1UECg",
        "SignatureData": "MIIGGyJKoZIhvcNAQcCoIIICzCCCAcCAQExDzANBgglghkgBZQM EA gEFADAzBgqhkiG9w0BBwGgJgQkNTdhNDQ1ZWUtYTlhNS00ZTAxLTlhODctYjllYjhlM2NiZThioIIFLjCCBSowggQSoAMCAQICEFWva9YyvKMcjDSI7H3m8towDQYJKoZIhvcNAQELBQAwYjELMAkGA1UEBhMCSlxcMzAxXBgggZyYxZCzAJBgNVBAYTAktXMTMwMQYDVQQKD CpUaGUgUHVibGljIEF1dG",
        "HashAlgorithm": "SHA-2",
        "SigningDatatype": "PKCS7",
        "TransactionDate": "2020-07-06T23:58:45.18"
      },
      "PersonalData": {
        "CivilID": "123456789012",
        "FullNameAr": "اسم حامل البطاقة",
        "FullNameEn": "CARD HOLDER NAME",
        "NationalityEn": "KWT",
        "NationalityAr": "كويتي",
        "BirthDate": "1992-02-12T00:00:00",
        "Gender": "M",
        "MobileNumber": "55555555",
        "EmailAddress": "test@test.com",
        "BloodGroup": "O+",
        "Photo": "iVBORw0KGgoAAAANSUHEUgAAAJMAAAHPCAIAAAApiMxqAAAAAXNSR0IArs4c6QAAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAADSMAAA7DAcdvqGQAAHH+SURBVHhe7Z0HfBTXubfXSe69ubn5ktzcex0nduLETuK4d5vmhvvuOACNmBAwvTeezG9CINNb6J3EIgmBEgISaj3iitUe2+",
        "CardExpiryDate": "2021-05-20T00:00:00",
        "DocumentNumber": "K29999990",
        "PassportNumber": "N010999992",
        "GovData": {
          "CivilIdExpiryDate": "2021-05-20T00:00:00",
          "MOIReference": "109999992",
          "SponsorName": "الشركة المتحدة المقاولات"
        },
        "Address": {
          "Governorate": "المهولة",
          "Area": "المهولة",
          "PaciBuildingNumber": "12345678"
        }
      }
    }
  }
}
    
```

```

"BuildingType": "شقة",
"FloorNumber": " ",
"BuildingNumber": "14",
"BlockNumber": "2",
"UnitNumber": "38",
"StreetName": "5507 شارع",
"UnitType": "منزل",
}},
"Signature": {
"SignatureData": "MIIIGgYJKoZIhvcNAQcCoIIICzCCCAcCAQExDzANBgUglghkgBZQM
EAqEFADAzBgkqhkiG9w0BBWGgJgQkNtdhNDQlZWUuYTNhNS00ZTAxLTlhODctYjllYjhl
M2NiZThioIIFLjCCBSowggQSoAMCAQICEFWva9YyVvKMcjDSI7H3m8towDQYJKoZIhvc
NAQELBQAwYjElMkAkaG1UEBhMCS1cxMzAxBgggZYxCzAJBgNVBAYTAktXMTMwMQYDVQQKD
CpUaGUgUHVhbmGljIEF1dG",
"SigningCertificate": "MIIIFDzCCA/egAwIBAgIQCbWInLKPlxZDokKZmKXdxTANBg
kqhkiG9w0BAQsFADBiMQswCQYDVQQGEWJLVzEzMDEGA1UEChMqVGlhIFB1YmxpYyBBdX
Rob3JpdHkgZm9yYENpdmlsIEluZm9ybWw0aW9uMR4wHAYDVQQDExVQUNJIERlbW9uc3
RyYXRpb24gQ0EwHhcNMjkwMTA5MDAwMDAwWhcNMjAwMTAxMjM0OTU5WjB8MQswCQYDVQ
QGEWJLVzEzMDEGA1UECg"
}}

```

- **Callback Parameters Definition**

Request Details		
Property Name	Data Type	Description
Request ID	String	id of the authentication request
Request Type	String	Authentication in case it's authentication request or Signing in case it's signing request
Service Provider ID	String	ID of the service provider
Additional Data	String	Provided by the service provider with the request
Challenge	String	Optional and can be passed by the service provider in case of authentication request Hash of the document to be signed, must be passed by the service provider in case of signing request
Civil No	String	Civil no of the applicant who's tried to authenticate to the service provider system

Result Details			
Property Name	Data Type	Description	
Result Code	String		
		Result Code	Description
		10	Authenticated
		20	Certificate Revoked
		30	FailedToVerfiy
		40	Declined
		50	Certificate Expired
User Action	String		
		User Action Code	Description
		1	Accept
		2	Decline
User Civil No	String	Civil no of the applicant who's tried to authenticate to the service provider system	
User Certificate	String	PEM string of the user certificate	

Signing Data	String	PKCS7 signature Asn1 Encoded in case authentication request , PKCS1 signature in case signing request
Signature Data	String	
Hash Algorithm	String	
Signing Data type	String	PKCS7 in case of authentication request , PKCS1 signature in case signing request
Transaction Date	Date time	Date and time of the authentication time

Personal Data	
Property Name	Data Type
Civil ID	String
Full Name AR	String
Full Name EN	String
Birth Date	String
Gender	String
Email Address	String
Mobile Number	String
Card Expiry Date	Date Time
Passport Number	String
Card Serial	String
Document Number	String
Photo	String / Base64
Blood Group	String
Nationality AR	String
Nationality EN	String
Nationality Flag	Byte[]
Address	Object
Address - Governorate	String
Address - Area	String
Address - PACI Building Number	String
Address - Building Type	String
Address - Floor Number	String
Address - Building Number	String
Address - Block Number	String
Address - Unit Number	String
Address - Street Name	String
Address - Unit Type	String
Gov Id Data	Object
Gov Id Data- Civil Id Expiry Date	Date Time
Gov Id Data- MOI Reference	String
Gov Id Data- Sponsor Name	String

Signature		
Property Name	Data Type	Description
Signature Data	String	id of the authentication request
Signing Certificate	Byte[]	PEM string of the signing certificate

- **Callback Validation:**

The received callback contains signature parameter that hold the SignatureData and SigningCertificate. The SignatureData field that holds the signed value of the other parametes and it can be used to validate the data integrity and nonrepudiation.

Object validation can be achieved by following the below steps:

1. Create MIDAuthSignResponse object that contains the personalData, RequestDetails, and ResultDetails objects (fields ordering should follow the same ordering of the Callback JSON Schema above)
2. Get the SigningCertificate from the signature parameter and compare it to the signing certificate that will be provided by PACI with the package.
3. Get the public key of the SignatureData.
4. Compute hash for the generated object in step 1 using sha256.
5. Use the public key to verify the computed hash against the DataSignature value from the received object.

CONFIDENTIAL

4.4.3 Tools

4.4.3.1 Client Certificate Generator Tool

Allows the service provider administrator to generate a certificate request for an SP client certificate, which is necessary for MIDAuthServiceClient SDK to authenticate to the backend. The generated .x10 request needs to be manually sent to PACI to certify and return the certificate back to be installed using the same tool.

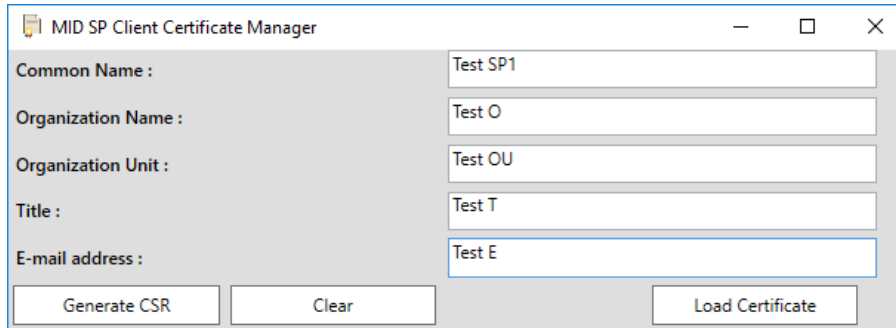


Figure 10 - Client Certificate CSR Generator

4.4.3.2 Certificate Generation Steps

1. Run MID Auth Sign Client Certificate CSR Gen.exe
2. Press on Generate CSR button then file with extension x10 will be created in the same path of the tool
3. Send the CSR to PACI
4. After retrieving the certificate from PACI, go to "Load Certificate" > "Import Certificate" > Select the issued certificate file > View the details > Click "Install Certificate" to install it

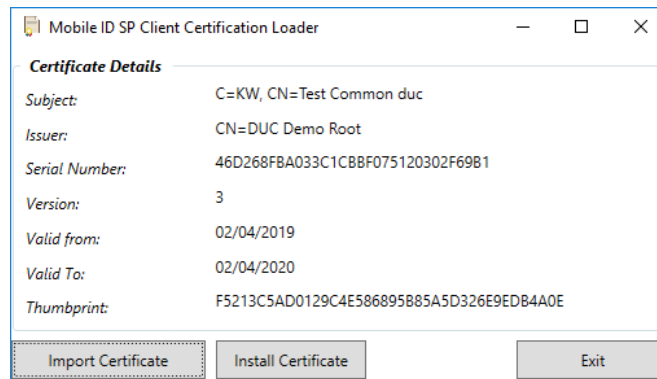


Figure 11 - Client Certificate Loader

4.5 Mobile ID URL Scheme API

4.5.1 Integration Flow

Please refer to section 3.2 for details on the integration flow.

4.5.2 API Documentation

The URL scheme API allows integrating SP mobile based applications (both native and browser-based) to integrate with the Mobile ID App, on both iOS and Android platforms.

- **URL Scheme Format**

The URL Scheme consists of two parts to launch the Kuwait mobile ID app:

```
PACIMID://Send?RID=<Request ID>
```

where:

Scheme Identifier: PACIMID

Parameters:

Request ID: The Request ID returned from the Mobile ID SDK

- **Sample Code (iOS Swift) to call Mobile ID App using the URL Scheme**

```
let urlString = String (format:"PACIMID://Send?RID=%@",reqTF.text ?? "")
    if let url = URL(string:urlString) {
        if UIApplication.shared.canOpenURL(url) {
            UIApplication.shared.open(url, options: [:],
                completionHandler: nil)
        }
    }
```

Appendix I: Code Samples

InitiateAuthenticationRequestQR - C# Code

```

X509Certificate2 cert = new X509Certificate2();
X509Store x509Store = new X509Store(StoreName.My, StoreLocation.LocalMachine);
x509Store.Open(OpenFlags.ReadOnly | OpenFlags.OpenExistingOnly);
var certificate2Collection = x509Store.Certificates
    .Find(X509FindType.FindByThumbprint, certThumbprint, false);
if (certificate2Collection == null || certificate2Collection.Count == 0 ||
    certificate2Collection.Count > 1)
{
    throw new Exception("Certificate not found");
}
cert = certificate2Collection[0];

var authClient = new MIDAuthServiceClient(cert, "PACI-MobileID.paci-scd.kw",
    "7778");

var res = authClient.InitiateAuthRequestQR(new
    MIDAuthSignContract.Entities.AuthenticateRequest
    {
        ServiceProviderId = "1",
        ServiceDescriptionEN = Constants.SERVICE_DESCRIPTION_EN,
        ServiceDescriptionAR = Constants.SERVICE_DESCRIPTION_AR,
        AuthenticationReasonEn = Constants.AUTH_REASON_EN,
        AuthenticationReasonAr = Constants.AUTH_REASON_EN,
        RequestUserDetails = true,
        CallbackURL = ConfigurationManager.AppSettings["ApiUrl"] +
            "auth/AuthenticateUser"
    });
return new AuthenticationQRData
    {
        QRCodeImage = res.Data.QRCodeImage,
        QRCode = res.Data.QRCode
    };
    
```

Overloaded

```

X509Certificate2 cert = new X509Certificate2();
X509Store x509Store = new X509Store(StoreName.My, StoreLocation.LocalMachine);
x509Store.Open(OpenFlags.ReadOnly | OpenFlags.OpenExistingOnly);
var certificate2Collection = x509Store.Certificates
    .Find(X509FindType.FindByThumbprint, certThumbprint, false);
if (certificate2Collection == null || certificate2Collection.Count == 0 ||
    certificate2Collection.Count > 1)
{
    throw new Exception("Certificate not found");
}
cert = certificate2Collection[0];

var authClient = new MIDAuthServiceClient(cert, "PACI-MobileID.paci-scd.kw",
    "7778");

var res = authClient.InitiateAuthRequestQR(
    MIDAuthSignContract.Enums.AssuranceLevel.High, new
    MIDAuthSignContract.Entities.AuthenticateRequest
    {
        ServiceProviderId = "1",
        ServiceDescriptionEN = Constants.SERVICE_DESCRIPTION_EN,
        ServiceDescriptionAR = Constants.SERVICE_DESCRIPTION_AR,
        AuthenticationReasonEn = Constants.AUTH_REASON_EN,
    }
    );
    
```

```
AuthenticationReasonAr = Constants.AUTH_REASON_EN,  
RequestUserDetails = true,  
SPCallbackURL =  
ConfigurationManager.AppSettings["ApiUrl"] + "auth/AuthenticateUser"  
});  
};  
return new AuthenticationQRData  
{  
    QRCodeImage = res.Data.QRCodeImage,  
    QRCode = res.Data.QRCode  
};
```

CONFIDENTIAL

InitiateAuthenticationRequestPN - C#Code

```

X509Certificate2 cert = new X509Certificate2();
X509Store x509Store = new X509Store(StoreName.My, StoreLocation.LocalMachine);
x509Store.Open(OpenFlags.ReadOnly | OpenFlags.OpenExistingOnly);
var certificate2Collection = x509Store.Certificates
    .Find(X509FindType.FindByThumbprint, certThumbprint, false);
if (certificate2Collection == null || certificate2Collection.Count == 0 ||
    certificate2Collection.Count > 1)
{
    throw new Exception("Certificate not found");
}
cert = certificate2Collection[0];

var authClient = new MIDAuthServiceClient(cert, "PACI-MobileID.paci-scd.kw",
    "7778");

var res = authClient.InitiateAuthRequestPN(new
    MIDAuthSignContract.Entities.AuthenticateRequest
    {
        ServiceProviderId = "1",
        ServiceDescriptionEN = Constants.SERVICE_DESCRIPTION_EN,
        ServiceDescriptionAR = Constants.SERVICE_DESCRIPTION_AR,
        AuthenticationReasonEN = Constants.AUTH_REASON_EN,
        AuthenticationReasonAR = Constants.AUTH_REASON_EN,
        RequestUserDetails = true,
        SPCallbackURL = ConfigurationManager.AppSettings["ApiUrl"] +
        "auth/AuthenticateUser",
        PersonCivilNo = civilNo
    });
return res.Data ;
    
```

Overloaded

```

X509Certificate2 cert = new X509Certificate2();
X509Store x509Store = new X509Store(StoreName.My, StoreLocation.LocalMachine);
x509Store.Open(OpenFlags.ReadOnly | OpenFlags.OpenExistingOnly);
var certificate2Collection = x509Store.Certificates
    .Find(X509FindType.FindByThumbprint, certThumbprint, false);
if (certificate2Collection == null || certificate2Collection.Count == 0 ||
    certificate2Collection.Count > 1)
{
    throw new Exception("Certificate not found");
}
cert = certificate2Collection[0];
var authClient = new MIDAuthServiceClient(cert, "PACI-MobileID.paci-scd.kw",
    "7778");

var res = authClient.InitiateAuthRequestPN(
    MIDAuthSignContract.Enums.AssuranceLevel.High ,new
    MIDAuthSignContract.Entities.AuthenticateRequest
    {
        ServiceProviderId = "1",
        ServiceDescriptionEN = Constants.SERVICE_DESCRIPTION_EN,
        ServiceDescriptionAR = Constants.SERVICE_DESCRIPTION_AR,
        AuthenticationReasonEN = Constants.AUTH_REASON_EN,
        AuthenticationReasonAR = Constants.AUTH_REASON_EN,
        RequestUserDetails = true,
        SPCallbackURL = ConfigurationManager.AppSettings["ApiUrl"] +
        "auth/AuthenticateUser",
        PersonCivilNo = civilNo
    });
return res.Data ;
    
```

InitiateSigningRequest - C# Code

```
X509Certificate2 cert = new X509Certificate2();
X509Store x509Store = new X509Store(StoreName.My, StoreLocation.LocalMachine);
x509Store.Open(OpenFlags.ReadOnly | OpenFlags.OpenExistingOnly);
var certificate2Collection = x509Store.Certificates
    .Find(X509FindType.FindByThumbprint, certThumbprint, false);
if (certificate2Collection == null || certificate2Collection.Count == 0 ||
    certificate2Collection.Count > 1)
{
    throw new Exception("Certificate not found");
}
cert = certificate2Collection[0];

var authClient = new MIDAuthServiceClient(cert, "PACI-MobileID.paci-scd.kw",
    "7778");

var res = authClient.InitiateSigningRequest(new
MIDAuthSignContract.Entities.SigningRequest
{
    PromptAr = Constants.PROMPT_AR,
    PromptEn = Constants.PROMPT_EN,
    SigningReasonEn = Constants.SIGN_REASON_EN,
    SigningReasonAr = Constants.SIGN_REASON_AR,
    PersonCivilNo = civilNo,
    ServiceProviderId = "1",
    CallbackURL = ConfigurationManager.AppSettings["ApiUrl"] +
    "Signing/DocumentSigned",
    RequestUserDetails = true,
    Challenge = "94YorVEe7+w2Aak3RAB7ttVqcd3t9U9YdoMokgvvgafI=",
    FileName = docName,
    PreviewFile = docData,
    FileType = "application/pdf"
});
return res.Data ;
```

NotifyPerson - C# Code

```
X509Certificate2 cert = new X509Certificate2();
X509Store x509Store = new X509Store(StoreName.My, StoreLocation.LocalMachine);
x509Store.Open(OpenFlags.ReadOnly | OpenFlags.OpenExistingOnly);
var certificate2Collection = x509Store.Certificates
    .Find(X509FindType.FindByThumbprint, certThumbprint, false);
if (certificate2Collection == null || certificate2Collection.Count == 0 ||
    certificate2Collection.Count > 1)
{
    throw new Exception("Certificate not found");
}
cert = certificate2Collection[0];

var authClient = new MIDAuthServiceClient(cert, "PACI-MobileID.paci-scd.kw",
    "7778");

var res = authClient.NotifyPerson(
    new MIDAuthSignContract.Entities.NotificationRequest
    {
        ServiceProviderId = serviceProviderID,
        PersonCivilNo = civilNo,
        NotificationSubjectEn = "Subject English",
        NotificationSubjectAr = "Subject Arabic",
        NotificationMessageEn = "Message English",
        NotificationMessageAr = "message Arabic"
    });
```

HasMobileIdentity - C# Code

```
X509Certificate2 cert = new X509Certificate2();
X509Store x509Store = new X509Store(StoreName.My, StoreLocation.LocalMachine);
x509Store.Open(OpenFlags.ReadOnly | OpenFlags.OpenExistingOnly);
var certificate2Collection = x509Store.Certificates
    .Find(X509FindType.FindByThumbprint, certThumbprint, false);
if (certificate2Collection == null || certificate2Collection.Count == 0 ||
    certificate2Collection.Count > 1)
{
    throw new Exception("Certificate not found");
}
cert = certificate2Collection[0];

var authClient = new MIDAuthServiceClient(cert, "PACI-MobileID.paci-scd.kw",
    "7778");

var res = authClient.HasMobileIdentity(serviceProviderID, civilNo);
return new HasMobileIdResponse
{
    HasMobileId = res.Data.HasMobileId,
    CivilNo = res.Data.CivilNo,
    Status = res.Data.Status,
    VerificationLevel = res.Data.VerificationLevel
};
```

CONFIDENTIAL

GetUserCertificate - C# Code

```
X509Certificate2 cert = new X509Certificate2();
X509Store x509Store = new X509Store(StoreName.My, StoreLocation.LocalMachine);
x509Store.Open(OpenFlags.ReadOnly | OpenFlags.OpenExistingOnly);
var certificate2Collection = x509Store.Certificates
    .Find(X509FindType.FindByThumbprint, certThumbprint, false);
if (certificate2Collection == null || certificate2Collection.Count == 0 ||
certificate2Collection.Count > 1)
{
    throw new Exception("Certificate not found");
}
cert = certificate2Collection[0];

var authClient = new MIDAuthServiceClient(cert, "PACI-MobileID.paci-scd.kw",
"7778");

var res = authClient.GetUserCertificate(serviceProviderID, civilNo);
return res.Data;
```

CONFIDENTIAL

VerifyMobileIdQRData- C# Code

```
X509Certificate2 cert = new X509Certificate2();
X509Store x509Store = new X509Store(StoreName.My, StoreLocation.LocalMachine);
x509Store.Open(OpenFlags.ReadOnly | OpenFlags.OpenExistingOnly);
var certificate2Collection = x509Store.Certificates
    .Find(X509FindType.FindByThumbprint, certThumbprint, false);
if (certificate2Collection == null || certificate2Collection.Count == 0 ||
    certificate2Collection.Count > 1)
{
    throw new Exception("Certificate not found");
}
cert = certificate2Collection[0];

var authClient = new MIDAuthServiceClient(cert, "PACI-MobileID.paci-scd.kw",
    "7778");

var res = authClient.VerifyMobileIdQRData(serviceProviderID, QRBase64);
return new MIDAuthSignContract.Entities.IdentityDetails{
    PersonIdentityData = res.Data.PersonIdentityData,
    DataSigningCertificate = res.Data.DataSigningCertificate,
    DataSignature = res.Data.DataSignature
};
```

CONFIDENTIAL