“iAM Smart” Developer Guide
for
Online Services

Version: *2.3.1*

FEBRUARY 2026

Document Revision History

Ver. No.	Release Date	Section Affected	Summary of Changes
1.0.0	30 Sep 2019		Initial release, with reference to eID API Specification Version 1.0.1
1.0.1	31 Oct 2019	Section 3.4.3	Corresponding to eID API Specification Version 1.0.1: 3.4.3: Added in-app browser description
		Section 4	Added Section 4: Reference Application
1.1.0	23 Dec 2019	Please see affected sections as follows:	Corresponding to eID API Specification Version 1.1.0:
		Section 1.1, 1.4, 3.8, 3.9, 4.3.2.1.6, 4.3.2.2.6, 4.3.3.2.6, 4.3.3.3.6, 4.3.4.2.6, 4.3.4.3.6, 4.3.2.1.7, 4.3.2.2.7, 4.3.3.2.7, 4.3.3.3.7, 4.3.4.2.7, 4.3.4.3.7	Added Anonymous Form Filling and Anonymous Digital Signing
		Section 1.1.9, 1.4, 3.6, 4.3.2.1.4, 4.3.2.1.7	Added PDF Digital Signing
		Section 1.1.8, 3.3.1.2	Added "Revoke Content Encryption Key"
		Section 1, 3.7, 4.3.2.1.5, 4.3.3.2.5, 4.3.4.2.5, 4.3.4.3.5	"Step-up authentication" changed to "Re-authentication"
		Section 1.1.4, 3.2.2	Added description "accessToken requested through the anonymous APIs can only be used once"
		Section 4.3.2.1.5, 4.3.2.2.5, 4.3.3.2.5, 4.3.3.3.5, 4.3.4.2.5, 4.3.4.3.5	Changed the title from ‘Step-up Authentication’ to ‘Re-authentication’
		Section 4.3.3.3.1, 4.3.4.2.1, 4.3.4.3.1	Changed the Online Service server part to a reference indication

Ver. No.	Release Date	Section Affected	Summary of Changes
		Section 4.3.1.3	Added “Process API Data for POST Request”
1.2.0	21 Feb 2020	Whole document	Changed "eID" to "iAM Smart" ("internet Access by Mobile in a Smart way")
		Whole document	Changed "Tokenised eID" to "Tokenised ID"
		Whole document	Align changes of the “iAM Smart” API Specification Version 1.2.0
		Section 1.1.10, 3.10	Added Direct Login
		Section 3.4.1.1, 3.4.4.1, 3.8.4.2, 3.9.4.3	Added the descriptions for the parameters (including redirectURI, scope and state) which are required to be URL encoded in the GET request and URLScheme
		Section 3	Added "App_Scheme" and "App_Link" for the source parameter
		Section 3.2.1	Added the description of how “iAM Smart” Mobile App invokes Online Service App based on the source parameter
		Section 3.8, 3.9	Updated the flow diagram based on the revised content in “iAM Smart” API Specification Version 1.2.0
1.2.1	02 Apr 2020	Section 3.10.4	Refined that either URL Scheme or Universal Link/App Link could be used to pass authCode to Online Service app
		Section 3.10.4.1	Added the descriptions of <context> in URL Scheme
		Section 4.3.2.2.8, 4.3.3.4.8, 4.3.4.4.8	Added the Reference Application of Direct Login
		Section 4.3.3.2, 4.3.4.2	Added the description of Online Service configuration to support Universal Link and App Link
1.2.2	27 Oct 2020	Section 1.2, 4.2, 4.3.1, 4.3.2.1, 4.3.2.2.8, 4.3.3.3	Added the description of .NET in Reference Applications.
		Section 3.10.1	Supplemented the implementation of direct login to Online Service website

Ver. No.	Release Date	Section Affected	Summary of Changes
1.2.3	22 Oct 2021	Section 1.1.3	Added description of the new billing address information in the “e-ME profile”
1.2.4	13 Apr 2022	Section 1.3	Add the description of Universal Link (iOS) /App Link (Android) for Online Service App
1.2.5	15 Jul 2022	Section 3.2.3	Update Error Code, D20012
		Section 3.5, 3.8	Update Form Filling scope, requests, response, and callback to V2. Online Service can request both eMEFields and profileFields with these V2 APIs.
1.2.6	14 Sep 2022	Section 1.1.11	Added the description of the support to CCIC holders
		Section 3.4.5	Added the description and workflow to verify whether “iAM Smart” user is a CCIC holder
2.0.0	14 Jun 2023	Whole document	Restructured the format and section of the API Specification Changed " e-Service" to Online Service
		Section 1.1 1.1.1, 1.1.2, 1.1.3	Updated the description of Overview, Introudction of API Functions and “iAM Smart” Account Version and User Profiles and Authorisation Scope and Access Token
		Section 1.1.10, 1.1.11	Added the description of Service Catalogue and Direct Login and Streamline Workflow
		Section 3.5	Added the description of Profiles API for obtaining the User Profiles information
		Section 3.10.1	Added the description of Direct Login v2 for “iAM Smart” user to login to Online Service from Service Catalogue in a simple and swift manner
		Appendix A	Added the desciprtion of API Deprecation
		Appendix C	Added different scenarios to illustrate how to implement “iAM Smart” Streamline Workflow
2.1.0	17 October 2023	Section 1.1.1	Added Bulk Digital Signing

Ver. No.	Release Date	Section Affected	Summary of Changes
		Section 1.1.2	Removed deplicated paragraph
		Section 3.10.1	Added supported browser list for Direct Login v2
		Section 3.11 - 3.15 Section B.3	Added Bulk Digital Signing WorkFlow Description Added Implementation Reference for Bulk Digital Signing
		Section B.3	Updated APP URL scheme from “hk.gov.ogcio” to “hk.gov.iamsmart”
		Whole document	The terms “government”, “government and related organisations” and “GRO” can be used interchangeably in the whole document
2.2.0	24 July 2024	Section 1.3, 3.10.4	Updated Direct Login Workflow descrption, pre-requisites, requests, response, and callback to V2.
		Terms And Conditions Section 1.1, 3.10.1, Appendix A.3.1, B.3.3.1, C	Updated OGCIO to DPO
2.2.1	11 September 2024	Section 3.14,3.15	Deleted the redundant parameter “accessToken”
2.2.2	6 January 2025	Appendix D	Added Essential Tips for App-to-App Direct Login v2
2.3.0	19 December 2025	Section 1.1.1, 1.1.13, 1.4, 1.4.10, 2, 3.1, 3.2.1, 3.16, B.3.2.3	Add in-app browser Javascript APIs related content
		Section 1.1.1, 1.1.3, 1.4, 1.4.11, 1.4.12, 3.17, 3.18	Added Step-up Authentication
2.3.1	12 February 2026	Section 3.2.2	Added rateLimitFactor in common API request header

Table of Contents

TERMS AND CONDITIONS

DPO has the sole discretion to amend or vary the Reference Application and the Developer Guide from time to time.

The Reference Application and the contents of the Developer Guide remain the property of, and shall not be reproduced in whole or in part without the express permission of the Government of the HKSAR ("HKSARG"). Information provided by the Developer Guide and all the associated intellectual property rights are retained by HKSARG.

Government and Related Organisations ("GRO") and their contractors (and sub-contractors, if applicable) (hereafter referred as users) may use the Reference Application and the Developer Guide for the purpose of designing, developing, testing and running of their e-government applications. By using the Reference Application and the Developer Guide, users agree not to sue HKSARG and agree to indemnify, defend and hold harmless HKSARG, its officers and employees from any and all third-party claims, liability, damages and/or costs (including but not limited to, legal fees) arising from the use of the Reference Application and the Developer Guide.

HKSARG will not be liable for any direct, indirect, incidental, special or consequential damages of any kind resulting from the use of or inability to use the Reference Application and information provided by the Developer Guide.

Modifications of the Reference Application by GRO may be required if any third-party libraries or technologies that the Reference Application adopted are no longer available due to product end of life, incompatibility issue, technology deprecation or other reasons.

Users agree to abide the usage terms and conditions specified in this form before releasing the Reference Application and the Developer Guide to their contractors (and sub-contractors, if applicable).

DEFINITIONS

The following terms and abbreviations are used in this document.

Terms / Abbreviations	Definition
"iAM Smart" System	The "iAM Smart" backend services to provide "iAM Smart" APIs and trigger Online Service callback APIs to return results to Online Service applications.
Online Service server	The Online Service backend services make use of "iAM Smart" APIs and provide Online Service callback APIs to receive results from "iAM Smart" System.
Online Service client terminal	The browser (PC or mobile) and/or Online Service mobile application that provide user interface to "iAM Smart" user and backend interface to Online Service server.
Online Service Website	It is the webpage generated by the Online Service server and displayed by a browser (Desktop PC or mobile).
"iAM Smart" e-Cert	Digital certificate issued by Recognized Certification Authority for "iAM Smart" account with digital signing function enabled. The usage and maintenance of the "iAM Smart" e-Cert are managed by "iAM Smart" System for the "iAM Smart" account.
Content Encryption Key or CEK	Symmetric encryption key provided by "iAM Smart" System for API data encryption and decryption.
Key Encryption Key or KEK	The public key certificate of Online Service for encrypting the CEK from the "iAM Smart" System.
Cross-Site Request Forgery or CSRF Attack	CSRF is an attack that forces an end user to execute unwanted actions on a web application in which he/she is currently authenticated.
"iAM Smart" BDSS	The "iAM Smart" Bulk Digital Signing System to support the provision of digital signing for the "iAM Smart" users to perform digital signing on multiple documents with only one digital signing cycle.

1. INTRODUCTION

1.1 OVERVIEW

"iAM Smart" is a one-stop personalised digital services platform which enables users to log in and use online services by their personal mobile phone in a smart and convenient way. "iAM Smart" users can currently conduct authentication, digital signing and "e-ME" form-filling functions via "iAM Smart" mobile app or website services. They can also set up personalised notification services for receiving government service updates with "iAM Smart" mobile app. Users can register "iAM Smart" accounts using the "iAM Smart" mobile app, at the self-registration kiosks and at the registration counters. As of end May 2023, there were over 260 government and commercial online services adopted "iAM Smart" with over 2 million "iAM Smart" registered users.

To build a smart government, which in turn contributes to the development of Hong Kong into a smart city, the Chief Executive announced in the 2022 Policy Address the initiative to turn all government services online in two years by end-2024 and provide one-stop digital services by fully adopting "iAM Smart" within three years by 2025 so as to realise "single portal for online government services (一網通辦)".

To support this initiative of building a smart government, the Digital Policy Office (DPO), formerly known as the Office of the Government Information Officer (OGCIO), will upgrade "iAM Smart" to enable Government and Related Organisations ("GRO") to integrate their existing online services with it in a more convenient and simple manner, simplify the workflows and develop more services that can bring convenience to the public. It also allows citizens to enjoy the use of various online services more conveniently and swiftly, reducing their needs to visit the relevant offices to submit applications and deal with various businesses in person. Moreover, "iAM Smart" will be re-positioned not only for the sole purpose as a digital identity but also a portal for easy access to key government information and online services without the need of account registration.

Online Services shall follow both "iAM Smart" API Specification and "iAM Smart" Developer Guide for "iAM Smart" Adoption. In addition, DPO has appointed a consultant to gauge user needs, expectations and aspiration on the future single portal for online services and will provide new "Reference System Flow and User Interface Specifications" for reference in late 2023.

1.1.1 Introduction of API functions

"iAM Smart" functions are built in the form of RESTful Application Program Interface (API), which could be accessed by registered Online Services upon "iAM Smart" user authorisation. "iAM Smart" makes reference to OAuth 2.0 for authentication and authorisation amongst "iAM Smart" users, Online Services and the "iAM Smart" System. Online services adopting "iAM Smart" are required to provide RESTful callback APIs to receive API responses from the "iAM Smart" System. "iAM Smart" APIs support the following functions:

- Authentication
  Online services can make use of the Authentication API provide by "iAM Smart" to verify the identities of the users in a simple and secure way. The API can be used in various scenarios, such as user login, membership system, booking system, etc.

- Form Filling with Service Login (aka Profiles)
  Online services can make use of the Profiles API to retrieve data of "profileFields" and "eMEFields" in Streamline Workflow. The API can be used in various scenarios, such as account opening, account linkup, form filling, etc.

- Form Filling without Service Login (aka Anonymous Form Filling)
  Online services can make use of the Anonymous Form Filling API to request user's personal information without the need of prior authentication to Online Service with "iAM Smart".

- Digital Signing with Service Login
  Online Services can make use of Digital Signing API to enable "iAM Smart" user to complete online digital signing with legal backing after user has authenticated with Online Service using "iAM Smart". It can be used in many cases, such as digital signing an online application form and digital signing a contract and agreement.

- Digital Signing without Service Login (aka Anonymous Digital Signing)
  Online services can make use of the Anonymous Digital Signing API to enable "iAM Smart" user to complete online digital signing without the need of prior authentication with "iAM Smart".

- Re-authentication with Service Login
  Online services can make use of the Re-authentication API provide by "iAM Smart" to re-confirm the "iAM Smart" user's identity before completing a critical transaction (e.g., confirm submission of tax return).

After a “iAM Smart” user has authenticated Online Service with “iAM Smart”, Online Service can leverage “Request Re-authentication” API to request the same “iAM Smart” user to re-authenticate himself/herself to “iAM Smart” System via “iAM Smart” Mobile App.

- Bulk Digital Signing with Service Login
  Online Services can make use of Bulk Digital Signing API to enable “iAM Smart” user to complete online digital signing for multiple documents with legal backing after user has authenticated with Online Service using “iAM Smart”.

- Bulk Digital Signing without Service Login (aka Anonymous Bulk Digital Signing)
  Online services can make use of the Anonymous Bulk Digital Signing API to enable “iAM Smart” user to complete online digital signing for multiple documents without the need of prior authentication with “iAM Smart”.

- In-App Browser
  Online services can make use of the In-App Browser API to providing seamless experience to users while accessing their websites in “iAM Smart”.

- Step-up Authentication with Service Login
  Online services can make use of the Step-up Authentication API to facilitate users further confirm their identity to “iAM Smart” enabled online services when conducting high value or critical transactions.

- Step-up Authentication without Service Login (aka Anonymous Step-up Authentication)
  Online services can make use of the Anonymous Step-up Authentication API to facilitate users further confirm their identity to “iAM Smart” enabled online services when conducting high value or critical transactions.

1.1.2 “iAM Smart” Account Version and User Profiles

“iAM Smart” Account Version

There are two versions of “iAM Smart”, namely “iAM Smart” and “iAM Smart+”. “iAM Smart” will provide general identity authentication and majority of the functions (such as “e-ME” form filling and personalised notifications), while “iAM Smart+” will provide the additional function of digital signing. Residents may download the mobile app and register for “iAM Smart” or “iAM Smart+” according to their needs.

Registration of “iAM Smart”

- Resident who has the new smart identity (ID) card can use his/her personal mobile phone with NFC (Near Field Communication) function for the remote registration of "iAM Smart+" via latest version of the "iAM Smart" mobile app.
- Remote registration for "iAM Smart" requires applicant to use personal mobile phone for taking photos of the HKIC and selfies for identity verification purpose. The whole process can be completed online using mobile phone.

Registration of "iAM Smart+"

- In-person registration for "iAM Smart+" can be made at self-registration kiosks in specified government premises and public locations by inserting HKIC for data retrieval and taking selfies to perform identity verification.
- In-person registration for "iAM Smart+" can also be made at registration service counters or "iAM Smart" mobile registration teams. The registration staff will check and verify the HKIC of the applicant. No photo-taking of the HKIC or selfie would be required during the registration process.

More information can be found in "iAM Smart" Thematic Website (https://www.iamsmart.gov.hk/reg.html).

"iAM Smart" User Profiles

An "iAM Smart" User Profiles contains both the account information and the personal data provided voluntarily in "e-ME" profile of corresponding "iAM Smart" account. Online Services can request "profileFields" and/or "eMEFields" of "iAM Smart" User Profiles via "iAM Smart" Profiles and Form filling API for the purpose of identity verification and form filling respectively.

- **"profilefields"**

  The "profileFields" consist of the account information including Hong Kong Identity Card (HKIC) number, English Name, Chinese Name (if available), date of birth and gender. The "profileFields" are solely used for the purpose of identity verification, such as account opening, account matching, remote account opening, etc.

- **"eMEFields"**

  In addition to the account information, "iAM Smart" user can voluntarily input and update his/her personal information for the purpose of online form filling in "e-ME profile". All these information for "iAM Smart" form filling are named as "eMEFields"

which include HKIC number, English Name, marital status, phone number, email address, residential address and billing address[1], etc.

Details of the "iAM Smart" "profileFields" and "e-MEFields" can be found in Appendix A of the "iAM Smart API Specification".

Both User Profiles of "iAM Smart" and "iAM Smart+" account version consist of HKIC number, English Name, date of birth and gender. In most cases, the field "Chinese Name" is also available in "iAM Smart" User Profiles, however, only the "iAM Smart+" version has verified the user's "Chinese Name"[2]. Depending on the "iAM Smart" version, the account information would be verified with records stored at Immigration Department (ImmD) during registration. Regular checks for the "iAM Smart" user's account information with ImmD will also be conducted[3].

The account information in "profileFields" and "eMEFields" are the same for same user (e.g., Online Service would get the same HKIC number of specific "iAM Smart" user, no matter the information is from "profileFields" or "eMEFields"). However, both authentication and form filling statistic report in "iAM Smart" system would be updated if Online Service requests both "profileFields" and "eMEFields" in one API request. In addition, unlike "profileFields"which is for the purpose of identity verification, Online Service can allow "iAM Smart" user to modify online form data filled with "eMEFields".

1.1.3 Authorisation Scope and Access Token

"iAM Smart" System makes reference to the OAuth 2.0 authentication framework to enable Online Service to gain access to "iAM Smart" APIs, with "iAM Smart" user's authorisation using "iAM Smart" Mobile App.

---

[1] The billing address information is retrieved from one of the address data providers, including the electricity and gas companies and the Water Supplies Department. The billing address information consists of a PDF e-bill file and the related data, such as the provider name, owner name, service address, postal address, etc. This information could be used by Online Services as a kind of address proof, subject to the need for individual services.

[2] For the "iAM Smart+" account, the value of "chName" is verified, and the additional property "chNameVerified" with ImmD characters code point (with Private Use Areas (PUA), unable to render) will also be returned. Online Service shall use the "chName" for rendering purposes.

[3] When changes are found during regular check, the "iAM Smart" user will be notified to re-register his/her "iAM Smart" account in order to update the profile information. The last modification date ("lastModifiedDate" provided in the POST response of API "Request accessToken & Tokenised ID") of his/her "iAM Smart" Profile will also be updated to the current date upon completion of re-registration in this situation.

Determine Authorisation Scope

To access "iAM Smart" APIs, Online Service should first determine the authorisation scope(s) required for the "iAM Smart" API(s) invoked for its business needs. Online service may refer the table below for the authorisation mapping:

API	Scope Value
Authentication	`eidapi_auth`
Form Filling with Service Login (aka Profiles)	`eidapi_profiles`
Form Filling without Service Login (Anonymous Form Filling)	`eidapi_formFilling`
Digital Signing with Service Login	`eidapi_sign`
Digital Signing without Service Login (Anonymous Digital Signing)	`eidapi_sign`
Re-authentication with Service Login	`eidapi_fr`
Bulk Digital Signing with Service Login	`eidapi_bulksign`
Bulk Digital Signing without Service Login (Anonymous Bulk Digital Signing)	`eidapi_bulksign`
Step-up Authentication with Service Login	`eidapi_sua`
Step-up Authentication without Service Login (Anonymous Step-up Authentication)	`eidapi_sua`

Online Service should combine all authorisation scope(s) required into a single request to get authorisation from "iAM Smart" user through "iAM Smart" System when "iAM Smart" user requests login to Online Service using "iAM Smart" Mobile App. For example, authorisation scope "eidapi_auth", "eidapi_formFilling" and "eidapi_sign" are for "iAM Smart" authentication form filling and digital signing respectively.

Get Access Token

Online Service should invoke "iAM Smart" API "Request QR Page" with the required authorisation scope(s). With successful authorisation of the "iAM Smart" user, "iAM Smart" System will return an Authorisation Code ("authCode") with the required authorisation scopes (i.e., access rights for "iAM Smart" APIs) to Online Service via the Online Service client terminal (e.g., browser). Online Service then exchanges the authCode with "iAM Smart" System for the Access Token ("accessToken"), the Tokenised ID and other metadata of the "iAM Smart" user. The accessToken is bound to the authorisation scope(s) and Online Service must present the accessToken and relevant parameters to access the "iAM Smart" APIs. Online Service should retain the accessToken for access to "iAM Smart" APIs within its

validity period and destroy the accessToken when it expired, become invalid [Note 1] or the "iAM Smart" user left the Online Service (e.g., logout).

The process to get accessToken by Online Service is summarised below:

Step	Description
1	Online Service requests for authCode with required authorisation scope(s) by using Online Service client terminal (e.g., browser or Online Service App) to invoke the "iAM Smart" API "Request QR Page". Depending on the API request parameters, either: - A webpage with QR code will be shown in Online Service client terminal [A]; or - "iAM Smart" Mobile App installed in the device of the Online Service client terminal will be launched [B].
2	"iAM Smart" user logs in his/her "iAM Smart" Mobile App. To synchronise "iAM Smart" System with his/her "iAM Smart" Mobile App: - For [A], "iAM Smart" user will use the "iAM Smart" Mobile App to scan the QR code. - For [B], the synchronisation will be established upon login of "iAM Smart" Mobile App.
3	"iAM Smart" user gives authorisation in "iAM Smart" Mobile App for Online Service's request of authCode.
4	"iAM Smart" System returns authCode to Online Services using Online Service callback API via the same Online Service client terminal in Step 1.
5	Online Service invokes the "iAM Smart" API "Request accessToken & Tokenised ID" to get accessToken using the authCode.
6	"iAM Smart" System returns the accessToken, Tokenised ID and other metadata of the "iAM Smart" user to Online Service.
7	Online Service uses the accessToken and relevant parameters to access "iAM Smart" APIs.

[Note 1]: The accessToken obtained for invoking "iAM Smart" APIs for anonymous form filling and anonymous digital signing on document hash can be used only once. The accessToken obtained for invoking other "iAM Smart" APIs is valid for multiple use within its lifetime as indicated in the response of "iAM Smart" API "Request accessToken & Tokenised ID".

Details of authorisation scope can be found in the "iAM Smart" API Specification.

1.1.4 Authentication for Online Service Login and Authorisation for Anonymous "iAM Smart" APIs

Authentication for Online Service Login

To complete the "Get Access Token" process as stipulated in Section 1.1.3, "iAM Smart" user has to authenticate his/her identity using 2FA (i.e., the mobile device bounded with his/her "iAM Smart" account and a local biometric authentication with the "iAM Smart" Mobile App) to log in "iAM Smart" System to give authorisation. As such, successful completion of this "Get Access Token" process resulting in return of accessToken and Tokenised ID to Online Service should be accepted as successful authentication of the "iAM Smart" user to the Online Service.

Details of authentication can be found in Section 3.4.

Authorisation for Anonymous "iAM Smart" APIs

For Online Service to access "iAM Smart" APIs for anonymous form filling and anonymous digital signing using "iAM Smart", unlike the above Online Service Login process, the successful completion of "Get Access Token" process means that "iAM Smart" user has authorised a one-time access to the relevant "iAM Smart" API. The relevant accessToken should be valid for only one usage. Online Service must go through the same "Get Access Token" process for every access to the same or different anonymous "iAM Smart" APIs.

Details of authorisation for anonymous "iAM Smart" API can be found in Section 3.6 and 3.8.

1.1.5 Callback API and Transient Input Data

Due to the possible unexpected delay (e.g., network latency) and/or action time of the "iAM Smart" user to perform authorisation in the "iAM Smart" Mobile App (e.g., to determine which data items in "e-ME profile" are allowed for retrieval by Online Service), Online Service may experience timeout if an instant API response design is used. Therefore, asynchronised API response is adopted in "iAM Smart" APIs. Online Service is required to implement its own callback APIs to support the return of asynchronised API response from the "iAM Smart" System after invoking the relevant "iAM Smart" APIs. Online Service should also manage "iAM Smart" user's transient input data before invoking any "iAM Smart" APIs.

1.1.6 Data Protection for "iAM Smart" API

All "iAM Smart" and Online Service callback APIs are protected by HTTPS protocol during transmission. Additional API data encryption on API POST request and response body is implemented with encryption key being generated and renewed in "iAM Smart" System using a dedicated "iAM Smart" API "Request/Revoke Symmetric Content Encryption Key". The data encryption key has its validity and registered Online Service must request/renew the key by itself and make use the key to protect the data when invoking the relevant "iAM Smart" APIs.

1.1.7 Identification Code and Result Notification for Digital Signing

For "iAM Smart" user to digitally sign a document in Online Service, Online Service should first generate the hash from the document (or document digest for a PDF document) to be signed ("Document Hash"). The Document Hash will be sent to "iAM Smart" System for digital signing using the private key of "iAM Smart" user's "iAM Smart" e-Cert. A signature bundle including the digital signature, "iAM Smart" e-Cert, etc., will be returned to Online Service for further computing of the signed document. When "iAM Smart" user has logged in the Online Service, both Online Service and "iAM Smart" System will compute and display a 4-digit identification code using the Document Hash and the hash of Tokenised ID of the "iAM Smart" user. For anonymous digital signing, the 4-digit identification code will be computed using the Document Hash, the hash of HKIC number and the hash of client ID of Online Service. "iAM Smart" User should be requested to verify the two identification codes are the same before authorising the digital signing request. After receiving and verifying the signature bundle using the "iAM Smart" e-Cert, Online Service should notify the digital signing result to "iAM Smart" System by calling the "Online Service Acknowledges Digital Signing Result" "iAM Smart" API.

Details of the computation algorithm of the 4-digit identification code can be found in Section 3.7 and 3.8.

For bulk digital signing, Online Service should first generate the hash from the documents (and/or document digest for PDF documents) to be signed ("Document Hash"). The Document Hash will be sent to "iAM Smart" System for Bulk Digital Signing using the private key of "iAM Smart" user's "iAM Smart" e-Cert. A signature bundle including the digital signature, "iAM Smart" e-Cert, etc., will be returned to Online Service for further computing of the signed documents. When "iAM Smart" user has logged in the Online Service, both Online Service and "iAM Smart" System will compute and display a 6-digit identification code

using the Document Hash and the hash of Tokenised ID of the "iAM Smart" user. For anonymous digital signing, the 6-digit identification code will be computed using the Document Hash, the hash of HKIC number and the hash of client ID of Online Service. "iAM Smart" User should be requested to verify the two identification codes are the same before authorising the digital signing request. After receiving and verifying the signature bundle using the "iAM Smart" e-Cert, Online Service should notify the digital signing result to "iAM Smart" System by calling the "Online Service Acknowledges Bulk Digital Signing Result" "iAM Smart" API.

Details of the computation algorithm of the 6-digit identification code can be found in Section 3.11.1.1.

1.1.8 Re-authentication

Re-authentication provides an alternative for Online Service to re-confirm the "iAM Smart" user's identity before completing a critical transaction (e.g., confirm submission of his tax return). After an "iAM Smart" user has authenticated in Online Service using "iAM Smart", Online Service can leverage the "Request Re-authentication" "iAM Smart" API to request the same "iAM Smart" user to re-authenticate himself/herself to "iAM Smart" System via "iAM Smart" Mobile App.

1.1.9 Consular Corps Identity Card ("CCIC")

"iAM Smart" System supports the holders of Consular Corps Identity Card ("CCIC") to register for "iAM Smart+" accounts with effect from November 2022. A dedicated registration team is responsible for the registration of "iAM Smart+" accounts for CCIC holders.

CCIC is one kind of Hong Kong Identity Card issued by Immigration Department ("ImmD") since 20 February 2019 and the registration data of the CCIC holders are recognised as part of the Registration of Persons ("ROP") data. CCICs are issued to consuls, consular staff, the head and members of the Office of the European Union in Hong Kong, their spouses and dependent children of age 11 or above.

More information about CCIC can be found on ImmD website: (https://www.immd.gov.hk/eng/services/hkid/ccic.html).

There are two ways for Online Services to verify whether the specific "iAM Smart" user is a CCIC holder:

1. Since “J” prefix of identity card number is only allotted for the holders of CCIC, Online Services may invoke the “GetProfile” or form filling API and check against the prefix of the identity card number of the “iAM Smart” user so as to distinguish whether it is a CCIC holder.

2. Online Services can also invoke the “verfiyIsCcic” API to verify whether the “iAM Smart” user is a CCIC holder after the user performs the authentication process and obtain the accessToken and Tokenised ID. Details of verification of CCIC holder can be found in Section 0.

1.1.10 Service Catalogue and Direct Login

“iAM Smart” Platform provides a Single Portal for Online Services to enable users to access Online Service directly with one single login through “iAM Smart”. Online Service Catalogue within “iAM Smart” Mobile App offers a list of Online Services which have adopted the features of “iAM Smart”. With Direct Login function, “iAM Smart” users can choose the desired Online Service to launch its web application for logging into the Online Service directly (i.e., without the need of providing further login credentials from user). Authentication is performed at the background with minimal user intervention for better user experience. Currently, Direct Login function is only available for “iAM Smart” enabled web service, but not mobile app service.

Direct Login

To provide a better user experience to “iAM Smart” user, “iAM Smart” provides a simplified Direct Login process, allowing user to initiate the login request from “iAM Smart” service catalogue (i.e., Direct Login v2). The implementation details can be found in Section 3.10.1 of this developer guide.

Direct Access

Similar to Direct Login workflow, “iAM Smart” users can choose the desired Website or App from Service Catalogue to launch its web application with external browser if their business process does not involve “Authentication”. Special approval is required to use Direct Access, and Online Service using the Direct Access function needs to provide sound justification to the Support Team.

1.1.11 Streamline Workflow

The Streamline Workflow is one of the important enhancement to realise "single portal for online government services (一網通辦)", aiming to provide the simplified and seamless "iAM Smart" workflow with a view to enhancing user experience without extra switching back and forth between "iAM Smart" Mobile App and Online Services.

"iAM Smart" provides below two new APIs to simplify the authentication and form filling workflow.

Direct Login v2

The streamlined authentication workflow (i.e., Direct Login v2) allows "iAM Smart" user to login to Online Service from Service Catalogue in a simple and swift manner.

Form Filling with Service Login (aka Profiles)

The streamlined form filling workflow (i.e., Profiles) allows Online Service to obtain "iAM Smart" "profileFields" and "eMEFields" for the purpose of identity verification and form filling respectively in one go after the "Authentication" process, without "iAM Smart" user to give further explicit consent for data passing.

The Streamline Workflow can be adopted in various scenarios. Examples of different scenarios of Streamline Workflow implementation can be found in the Appendix C.

1.1.12 Bulk Digital Signing

The iAM Smart Bulk Digital Signing System (BDSS) is to support the provision of digital signing for the "iAM Smart" users to perform digital signing on multiple documents with only one digital signing cycle.

BDSS is composed of the following segmentations:

1. Bulk Digital Signing Module – provide functions including batch submission & handling and bulk digital signing for digital signing process which is initiated by the Online Service to submit the request for bulk digital signing with "iAM Smart" authorisation by the user. This bulk digital signing supports mix of hash and/or PDF digital signings, normal and anonymous digital signing flow.

2. Integration with Online Service – provide the RESTful APIs for integration with the Online Service.
3. Integration with “iAM Smart” Mobile App – integrate with the existing “iAM Smart” Mobile App for user authorisation of the bulk digital signing request, push notification and related notification function.
4. Integration with “iAM Smart” support functions – implement the system support functions including but not limited to statistical reports, billing reports, audit trails and audit log

1.1.13 In-App Browser

“iAM Smart” has implemented In-App browser solution which serves as a substitute of external mobile browsers (e.g. Safari, Chrome) with the benefits of providing seamless experience to users while accessing Online Service Websites in “iAM Smart”. When the Online Service Websites are running in the In-App browser, they can not only call the existing RESTful APIs to implement the workflows described previously but also invoke the add-on functions described in chapters 1.4.10 and 3.16 to get the settings of the current “iAM Smart” mobile application and to close all the pages and In-app browser directly. More details can be found in the “iAM Smart In-App Browser Integration Technical Reference” document.

1.1.14 Step-up Authentication

The Step-up Authentication function of “iAM Smart” provides additional real-time identity verification for critical online service processes or operations. Online Services can utilize the Hong Kong Identity Card (HKIC) reading capability via Near Field Communication (NFC) and facial recognition provided by “iAM Smart”, further enhancing the security of online services and transactions.

1.2 ASSUMPTIONS AND CONSTRAINTS

1. "iAM Smart" System adopts HTTPS to protect the data in transmission. Additional API data encryption is adopted for API POST request and response body.
2. Application data in the API POST request and response is in JSON format. Online Service Applications are responsible for processing of the returned data and reflecting the result to end users if required.
3. List of browsers supported by "iAM Smart" System is shown in Appendix B of the "iAM Smart" API Specification. Online Service should detect the browser's user agent value and pass respective source value in the list to "iAM Smart" System when necessary.
4. Reference Applications and sample source codes are provided for reference by Online Service to demonstrate the interaction between "iAM Smart" System (core system and Mobile App) and Online Service (including PC and mobile browser, Mobile Apps for Android and iOS, and backend service).
5. The Reference Applications and sample source codes are provided in the form of two technical implementations: Java and .NET. The Java version is built on Spring Boot Release 1.5.19, Spring Cloud Edgware SR5 development framework and is developed with JDK 1.8. It supports Red Hat Linux Server and Windows. The .NET version is built on .NET Framework 3.5 which supports Windows.
6. For Online Service developing Mobile App as client terminal, sample source codes for invoking "iAM Smart" Mobile App through URL Scheme / App Link are provided for reference in Android Studio / Kotlin (Java compatible) and Xcode / Object-C. The minimum Android and iOS version that "iAM Smart" Mobile App will be supported are Android 8.0 and iOS 12.0 respectively.
7. For adoption of the Reference Applications and sample source codes in other development platforms, SDKs and/or application environments, Online Service is responsible to check the compatibility and performs corresponding modification.
8. For Online Service not using Spring Boot, Spring Cloud development platform nor Microservices style for development, Online Service should follow HTTPS standard protocols, "iAM Smart" API Specification and "iAM Smart" Developer Guide to develop the interfaces with "iAM Smart" System.
9. Please also refer to "iAM Smart" API Specifications and other technical documents for other assumptions and possible constraints.

1.3 ONLINE SERVICE REGISTRATION TO "IAM SMART" SYSTEM

Online Service Providers are required to register their Online Services in "iAM Smart" System for accessing "iAM Smart" APIs. Registration information including application name, digital certificate for encipherment, redirect URL, etc., are required to be submitted by Online Services for the registration. A client ID ("clientID") and a secret key ("clientSecret") will be generated by "iAM Smart" System to the Online Service for accessing "iAM Smart" APIs upon the completion of the registration.

Online Service Provider is also required to provide Universal Link (iOS) / App Link (Android) and Package Name (Android) that offer better user experience and greater security for "iAM Smart" Mobile App to launch Online Service App from Service Catalogue. Custom URL scheme is not accepted unless sound justification can be provided.

1.4 POTENTIAL BUSINESS CASES USING "IAM SMART"

Potential business cases using "iAM Smart" are listed in the following sections for reference. For each case, possible high-level steps and the corresponding "iAM Smart" APIs and callback APIs to be used and/or implemented are provided (depending on the interface of Online Service and "iAM Smart" Mobile App as stipulated in Section 3.2.1).

While the involved "iAM Smart" APIs and callback APIs are listed below, details of the "iAM Smart" APIs and callback APIs can be found in the "iAM Smart" API Specification.

"iAM Smart" APIs:

#	**"iAM Smart" API**	**Type**
1	Request Symmetric Content Encryption Key	Data encryption / decryption
2	Revoke Symmetric Content Encryption Key	Data encryption / decryption
3	Request QR Page	Authentication
4	Request accessToken & Tokenised ID	Authentication
5	Open "iAM Smart" Mobile App for Authentication	Authentication
6	Open "iAM Smart" Mobile App for Getting Context	Interface with "iAM Smart" Mobile App for business operation
7	Request Profile	User Registration/Update
8	Request Form Filling	Business operation
9	Request Digital Signing	Business operation
10	Request Re-authentication	Business operation

#	**“iAM Smart” API**	**Type**
11	Online Service Acknowledges Digital Signing Result	Business operation
12	Request PDF Digital Signing	Business operation
13	Request Anonymous Form Filling	Business operation
14	Obtain Anonymous Form Filling Result	Business operation
15	Request Anonymous Digital Signing	Business operation
16	Obtain Anonymous Digital Signing Result	Business operation
17	Request Anonymous PDF Digital Signing	Business operation
18	Obtain Anonymous PDF Digital Signing Result	Business operation
19	Verify CCIC User	Business operation
20	Obtain User Profiles information	Business operation
21	Request Bulk Digital Signing	Business operation
22	Request Anonymous Bulk Digital Signing	Business operation
23	Request BSQC Token	Business operation
24	Online Service Acknowledges Bulk Digital Signing Result	Business operation
25	Enquire Bulk Digital Signing Status	Business operation
26	Cancel Bulk Digital Signing Request	Business operation
27	Request Step-up Authentication	Business operation
28	Request Anonymous Step-up Authentication	Business operation
29	Obtain Anonymous Step-up Authentication Result	Business operation

Callback APIs implemented by Online Services:

#	**Callback API (Implemented by Online Service)**	**Type**
A	Callback with authCode to Online Service App	Authentication
B	Callback with authCode to Online Service Server	Authentication
C	Callback to Receive “iAM Smart” Profile	Business operation
D	Callback to Receive Form Filling Information	Business operation
E	Callback to Receive Digital Signing Result	Business operation
F	Callback to Receive Re-authentication Result	Business operation
G	Callback to Receive PDF Digital Signing Result	Business operation

#	Callback API (Implemented by Online Service)	Type
H	Callback with AuthCode to Online Service Server (Direct Login V2)	Authentication
I	Callback to Receive BSQC Token	Business operation
J	Callback to Receive Bulk Digital Signing Result	Business operation
K	Callback to Receive Step-up Authentication Result	Business operation

"iAM Smart" In-App Browser APIs:

#	"iAM Smart" In-App Browser API	Type
i	Get Language Setting	Business operation
ii	Get Font Setting	Business operation
iii	Get Application Mode	Business operation
iv	Close In-App Browser	Business operation
v	Request Permission Dialogue	Business operation

1.4.1 User Registration

For Online Services requiring user registration, an "iAM Smart" user can either be a new user to the Online Service or an existing Online Service account owner whose account has not yet linked to his/her "iAM Smart" account. Online Service can identify an "iAM Smart" user and proceed the relevant process for user registration with the following steps:

Step	Description	"iAM Smart" API	Callback API
1a	Online Service provides a link for login by "iAM Smart" in the application.		
1b	Alternatively, "iAM Smart" user can login the Online Service via "iAM Smart" service catalogue		
2a	Online Service requests authCode with required authorisation scopes (e.g., "iAM Smart" authentication, form filling authorisation, etc.).	3, 5	A, B
2b	Alternatively, if "iAM Smart" user login the Online Service via "iAM Smart" service catalogue, Online Service would receive the authCode accordingly.		H

Step	Description	"iAM Smart" API	Callback API
3	Online Service gets accessToken and Tokenised ID of the "iAM Smart" user and checks if the Tokenised ID has linked with any existing user account of the Online Service.	4	
4	Online Service can optionally request for "iAM Smart" "Profiles API" when no linkage is found in Step 3 (i.e., a new user) or directly jump to Step 6 for user registration.	6, 7, 20	C
5	Online Service can optionally verify whether the "iAM Smart" user is a CCIC holder by checking "J" prefix of identity card number in "iAM Smart" Profile or invoking the "verfiyIsCcic" API.	19	
6	Online Service checks if the "iAM Smart" user owns an existing user account of Online Service (e.g., check his HKIC information in his/her "iAM Smart" User Profiles against Online Service account information).		
7	Online Service permits login if the "iAM Smart" user owns an existing account, otherwise Online Service could proceed to the new user registration process.		
8	Online Service links the Tokenised ID with the user account of Online Service.		
9	Online Service keeps the accessToken for subsequent access to "iAM Smart" APIs.		

1.4.2 User Authentication

User authentication is similar to user registration. For "iAM Smart" user who has linked his/her "iAM Smart" account (i.e., Tokenised ID) to Online Service user repository, Online Service can authenticate the "iAM Smart" user with the following steps:

Step	Description	"iAM Smart" API	Callback API
1a	Online Service provides a link for login by "iAM Smart" in the application.		
1b	Alternatively, "iAM Smart" user can login the Online Service via "iAM Smart" service catalogue		

Step	Description	"iAM Smart" API	Callback API
2a	Online Service requests for authCode with required authorisation scopes (e.g., "iAM Smart" authentication, form filling authorisation, etc.).	3, 5	A, B
2b	Alternatively, if "iAM Smart" user login the Online Service via "iAM Smart" service catalogue, Online Service would receive the authCode accordingly.		H
3	Online Service gets accessToken and Tokenised ID of "iAM Smart" user and checks if it has linked with any user account in the Online Service.	4	
4	User authentication completes if linkage found in Step 3 and Online Service permits "iAM Smart" user's login.		
5	Online Service keeps the accessToken for subsequent access to "iAM Smart" APIs.		

1.4.3 Form Filling with Service Login

The accessToken of the "iAM Smart" user that possessed by Online Service should include the authorisation scope of Profiles authorisation. Online Service can request for account information and personal information in "e-ME profile" of "iAM Smart" user with the following steps:

Step	Description	"iAM Smart" API	Callback API
1a	Online Service provides a link for form filling in the application.		
1b	Alternatively, "iAM Smart" user can login the Online Service via "iAM Smart" service catalogue		
2a	Online Service requests for authCode with required authorisation scopes (e.g., "iAM Smart" authentication, form filling authorisation, etc.).	3, 5	A, B
2b	Alternatively, if "iAM Smart" user login the Online Service via "iAM Smart" service catalogue, Online Service would receive the authCode accordingly.		H
3	Online Service gets accessToken and Tokenised ID of "iAM Smart" user.	4	

Step	Description	"iAM Smart" API	Callback API
4	Online Service requests "Profile Field" and "e-ME Field" for the purpose of identity verification and form filling respectively from "iAM Smart" System via Profiles API.	20	
5	"iAM Smart" System notes receipt of Online Service's request.		
6	Online Service server receives form filling result response from "iAM Smart" System		

1.4.4 Digital Signing with Service Login

The accessToken of the "iAM Smart" user that possessed by Online Service should include the authorisation scope of digital signing authorisation. Online Service can request for "iAM Smart" digital signing with the following steps:

Step	Description	"iAM Smart" API	Callback API
1	Online Service provides digital signing using "iAM Smart" in Online Service application.		
2	Online Service requests for authCode with required authorisation scopes (e.g., "iAM Smart" authentication, Digital Signing authentication).	3, 5	A, B
3	Online Service gets accessToken and Tokenised ID of "iAM Smart" user.	4	
4	Online Service calculates hash of the document (or digest for pdf document) to be signed and requests digital signing from "iAM Smart" System.	9,12	
5	"iAM Smart" System notes receipt of Online Service's request.		
6	Online Service generates a 4-digit identification code using hash of document (or digest for pdf document) and hash of Tokenised ID.		
7a	When Online Service client terminal and "iAM Smart" Mobile App are in different device, *Online Service client terminal shows the 4-digit identification code with instructions to inform "iAM Smart" user to verify the identification code and complete the "iAM Smart" digital signing in "iAM Smart" Mobile App and polls Online Service server for response from "iAM Smart" System.*		

Step	Description	"iAM Smart" API	Callback API
7b	When Online Service client terminal and "iAM Smart" Mobile App are in same device, *Online Service client terminal shows a 4-digit identification code and invokes "iAM Smart" Mobile App and polls Online Service server for response from "iAM Smart" System.*	6	
8	Online Service server receives signing response from "iAM Smart" System via Online Service callback API and computes the signed document.		E, G
9	Online Service server verifies and acknowledges digital signing result to "iAM Smart" System.	11	
10	Online Service server informs Online Service client terminal to stop polling.		
11	Online Service saves the accessToken for subsequent access to "iAM Smart" APIs.		

1.4.5 Re-authentication with Service Login

The accessToken of the "iAM Smart" user that possessed by Online Service should include the authorisation scope of Re-authentication authorisation. Online Service application can request for re-authentication with the following steps:

Step	Description	"iAM Smart" API	Callback API
1	Online Service provides re-authentication using "iAM Smart" in Online Service application.		
2	Online Service requests for authCode with required authorisation scopes (e.g., "iAM Smart" authentication, Re-authentication authorisation).	3, 5	A, B
3	Online Service gets accessToken and Tokenised ID of "iAM Smart" user.	4	
4	Online Service requests for re-authentication from "iAM Smart" System.	10	
5	"iAM Smart" System notes the receipt of Online Service's request.		

Step	Description	"iAM Smart" API	Callback API
6a	When Online Service client terminal and "iAM Smart" Mobile App are in different devices, *Online Service client terminal shows instruction to inform "iAM Smart" user to complete re-authentication in "iAM Smart" Mobile App and polls Online Service server for response returned from "iAM Smart" System.*		
6b	When Online Service client terminal and "iAM Smart" Mobile App are in the same device, *Online Service client terminal invokes "iAM Smart" Mobile App and polls Online Service server for response returned from "iAM Smart" System.*	6	
7	Online Service server receives Re-authentication response from "iAM Smart" System via Online Service callback API and proceeds its processing.		F
8	Online Service server informs Online Service client terminal to stop polling.		
9	Online Service saves accessToken for subsequent access to "iAM Smart" APIs.		

1.4.6 Anonymous Form Filling

Anonymous form filling describes the situation that user initiates a form filling request without prior authentication to Online Service using "iAM Smart". Online Service can request anonymous form filling with the following steps:

Step	Description	"iAM Smart" API	Callback API
1	Online Service provides a link for anonymous form filling in the application.		
2	Online Service requests "Profile Field" and "e-ME Field" for the purpose of identity verification and form filling respectively from "iAM Smart" System.	13	
3	"iAM Smart" System notes receipt of Online Service's request.		
4a	When Online Service client terminal and "iAM Smart" Mobile App are in different device, *Online Service redirects client terminal to "iAM Smart" QR/App broker page.*	3	

Step	Description	"iAM Smart" API	Callback API
4b	When Online Service client terminal and "iAM Smart" Mobile App are in the same device, *Online Service client terminal invokes "iAM Smart" Mobile App, Online Service App polls Online Service server for response from "iAM Smart" System.*	3, 6	
5	Online Service receives authCode from "iAM Smart" System via Online Service callback API.		A, B
6	Online Service gets accessToken and Tokenised ID of "iAM Smart" user.	4	
7	Online Service server obtains authorised information of "eMEFields" from "iAM Smart" System using accessToken and Tokenised ID and processes the form filling.	14	
8a	When Online Service client terminal is browser, *Online Service server redirects the browser to view the form filling result.*		
8b	When Online Service client terminal is Online Service App, *Online Service server informs Online Service App to stop polling and Online Service App shows the form filling result.*		

1.4.7 Anonymous Digital Signing

Anonymous digital signing describes the situation that user initiates a digital signing request without prior authentication to Online Service using "iAM Smart". Online Service can request anonymous digital signing with the following steps:

Step	Description	"iAM Smart" API	Callback API
1	Online Service provides a link for anonymous digital signing in the application.		
2	Online Service calculates hash of the document (or digest for pdf document) to be signed and hash of the HKIC number, and requests digital signing from "iAM Smart" System.	15, 17	
3	"iAM Smart" System notes receipt of Online Service's request.		
4	Online Service generates a 4-digit identification code using hash of document (or digest for pdf document), hash of the HKIC number and hash of clientID of the Online Service.		

Step	Description	"iAM Smart" API	Callback API
5a	When Online Service client terminal and "iAM Smart" Mobile App are in different devices, *Online Service client terminal shows the 4-digit identification code. For browser, Online Service redirects the browser to "iAM Smart" QR/App broker page. For Online Service App, it polls Online Service server for response from "iAM Smart" System.*	3	
5b	When Online Service client terminal and "iAM Smart" Mobile App are in the same device, *Online Service client terminal shows a 4-digit identification code. For browser, Online Service redirects the browser to "iAM Smart" QR/App broker page. For Online Service App, it invokes "iAM Smart" Mobile App and polls Online Service server for response from "iAM Smart" System.*	3, 6	
6	Online Service receives authCode from "iAM Smart" System via Online Service callback API.		A, B
7	Online Service gets accessToken and Tokenised ID of "iAM Smart" user.	4	
8	Online Service server obtains digital signing result from "iAM Smart" System using accessToken and Tokenised ID and computes the signed document.	16, 18	
9	Online Service server verifies and confirms digital signing result to "iAM Smart" System.	11	
10a	When Online Service client terminal is browser, *Online Service server redirects the browser to view digital signing result.*		
10b	When Online Service client terminal is Online Service App, *Online Service server informs Online Service App to stop polling and Online Service App shows the digital signing result.*		

1.4.8 Bulk Digital Signing with Service Login

The accessToken of the "iAM Smart" user that possessed by Online Service should include the authorisation scope of bulk digital signing authorisation. Online Service can request for "iAM Smart" bulk digital signing with the following steps:

Step	Description	"iAM Smart" API	Callback API
1	Online Service provides bulk digital signing using "iAM Smart" in Online Service application.		
2	Online Service requests for authCode with required authorisation scopes (e.g., "iAM Smart" authentication, Digital Signing authentication).	3, 5	A, B
3	Online Service gets accessToken and Tokenised ID of "iAM Smart" user.	4	
4	Online Service calculates hash of the documents (and/or digest for pdf documents) to be signed and requests bulk digital signing from "iAM Smart" System.	21	
5	"iAM Smart" System notes receipt of Online Service's request.		
6	Online Service generates a 6-digit identification code using hash of documents (and/or digest for pdf documents) and hash of Tokenised ID.		
7a	When Online Service client terminal and "iAM Smart" Mobile App are in different device, *Online Service client terminal shows the 6-digit identification code with instructions to inform "iAM Smart" user to verify the identification code and complete the "iAM Smart" bulk digital signing in "iAM Smart" Mobile App.*		
7b	When Online Service client terminal and "iAM Smart" Mobile App are in same device, *Online Service client terminal shows a 6-digit identification code and invokes "iAM Smart" Mobile App.*	6	
8	Online Service server receives BSQC Token and signing response from "iAM Smart" System via Online Service callback API.		I, J
9	Online Service server verifies and acknowledges bulk digital signing result to "iAM Smart" System if request completed.	24	
10	Online Service saves the accessToken and BSQC Token for subsequent access to "iAM Smart" APIs.		
11	Online Service uses the BSQC Token to enquire bulk digital signing status or cancel bulk digital signing requests.	25, 26	

1.4.9 Anonymous Bulk Digital Signing

Anonymous bulk digital signing describes the situation that user initiates a bulk digital signing request without prior authentication to Online Service using "iAM Smart". Online Service can request anonymous bulk digital signing with the following steps:

Step	Description	"iAM Smart" API	Callback API
1	Online Service provides a link for anonymous bulk digital signing in the application.		
2	Online Service calculates hash of the documents (and/or digest for pdf documents) to be signed and hash of the HKIC number, and requests bulk digital signing from "iAM Smart" System.	22	
3	"iAM Smart" System notes receipt of Online Service's request.		
4	Online Service generates a 6-digit identification code using hash of documents (and/or digest for pdf documents), hash of the HKIC number and hash of clientID of the Online Service.		
5a	When Online Service client terminal and "iAM Smart" Mobile App are in different devices, *Online Service client terminal shows the 6-digit identification code. For browser, Online Service redirects the browser to "iAM Smart" QR/App broker page.*	3	
5b	When Online Service client terminal and "iAM Smart" Mobile App are in the same device, *Online Service client terminal shows a 6-digit identification code. For browser, Online Service redirects the browser to "iAM Smart" QR/App broker page. For Online Service App, it invokes "iAM Smart" Mobile App.*	3, 6	
6	Online Service receives authCode from "iAM Smart" System via Online Service callback API.		A, B
7	Online Service gets accessToken and Tokenised ID of "iAM Smart" user.	4	
8	Online Service server obtains BSQC Token from "iAM Smart" System using accessToken and Tokenised ID.	23	
9	Online Service server receives signing response from "iAM Smart" System via Online Service callback API.		J
10	Online Service server verifies and confirms bulk digital signing result to "iAM Smart" System if request completed.	24	
11	Online Service saves the accessToken and BSQC Token for subsequent access to "iAM Smart" APIs.		

Step	Description	"iAM Smart" API	Callback API
12	Online Service uses the BSQC Token to enquire bulk digital signing status or cancel bulk digital signing requests.		

1.4.10 Requesting In-App Browser to Provide Information or Take Action

Requesting in-app browser to provide information or take action describes the situation that Online Service Websites call the Javascript APIs provided by "iAM Smart" to get its running context or ask "iAM Smart" to execute some actions on behalf of itself. Online Service can request in-app browser to provide information or take action with the following steps:

Step	Description	"iAM Smart" API	Callback API
1	Online Service Website calls the Javascript API	i, ii, iii, iv, or v	
2a	Online Service Website gets the "iAM Smart" information and act accordingly		
2b	"iAM Smart" executes the action (close in-app browser or pop up a window to request permission) based on the called Javascript API		

1.4.11 Step-up Authentication with Service Login

The accessToken of the "iAM Smart" user that possessed by Online Service should include the authorisation scope of step-up authentication authorisation. Online Service can request for "iAM Smart" step-up authentication with the following steps:

Step	Description	"iAM Smart" API	Callback API
1	Online Service provides Step-up Authentication using "iAM Smart" in Online Service application.		
2	Online Service requests for authCode with required authorisation scopes (e.g., "iAM Smart" authentication, Step-up Authentication authentication).	3, 5	A, B

Step	Description	"iAM Smart" API	Callback API
3	Online Service gets accessToken and Tokenised ID of "iAM Smart" user.	4	
4	Online Service requests Step-up Authentication from "iAM Smart" System.	27	
5	"iAM Smart" System notes receipt of Online Service's request.		
6a	When Online Service client terminal and "iAM Smart" Mobile App are in different device: *Online Service client terminal shows the instructions to inform "iAM Smart" user to complete the "iAM Smart" Step-up Authentication in "iAM Smart" Mobile App and polls Online Service server for response from "iAM Smart" System.*		
6b	When Online Service client terminal and "iAM Smart" Mobile App are in same device: *Online Service client terminal invokes "iAM Smart" Mobile App and polls Online Service server for response from "iAM Smart" System.*	6	
7	Online Service server receives signing response from "iAM Smart" System via Online Service callback API and computes the signed document.		K
8	Online Service server informs Online Service client terminal to stop polling.		
9	Online Service saves the accessToken for subsequent access to "iAM Smart" APIs.		

1.4.12 Anonymous Step-up Authentication

Anonymous step-up Authentication describes the situation that user initiates a step-up authentication request without prior authentication to Online Service using "iAM Smart". Online Service can request anonymous step-up authentication with the following steps:

Step	Description	"iAM Smart" API	Callback API
1	Online Service provides a link for anonymous Step-up Authentication in the application.		

Step	Description	"iAM Smart" API	Callback API
2	Online Service calculates hash of the HKIC number, and requests digital signing from "iAM Smart" System.	28	
3	"iAM Smart" System notes receipt of Online Service's request.		
4a	When Online Service client terminal and "iAM Smart" Mobile App are in different devices: *For browser, Online Service redirects the browser to "iAM Smart" QR/App broker page.* *For Online Service App, it polls Online Service server for response from "iAM Smart" System.*	3	
4b	When Online Service client terminal and "iAM Smart" Mobile App are in the same device: *For browser, Online Service redirects the browser to "iAM Smart" QR/App broker page.* *For Online Service App, it invokes "iAM Smart" Mobile App and polls Online Service server for response from "iAM Smart" System.*	3, 6	
5	Online Service receives authCode from "iAM Smart" System via Online Service callback API.		A, B
6	Online Service gets accessToken and Tokenised ID of "iAM Smart" user.	4	
7	Online Service server obtains Step-up Authentication result from "iAM Smart" System using accessToken and Tokenised ID.	29	
8a	When Online Service client terminal is browser, *Online Service server redirects the browser to view Step-up Authentication result.*		
8b	When Online Service client terminal is Online Service App, *Online Service server informs Online Service App to stop polling and Online Service App shows the Step-up Authentication result.*		

2. SYSTEM WORKFLOW USING "IAM SMART"

There are three types of system workflows using "iAM Smart":

1. "Authentication to Online Services using iAM Smart" for Online Service to authenticate "iAM Smart" user who gives authorisation to Online Service to get the accessToken to access the required "iAM Smart" APIs (e.g., user authentication).
2. "Online Service business operations using iAM Smart" for Online Service to access the required "iAM Smart" APIs for completing its business operations (e.g., form filling).
3. "Requesting in-app browser to provide information or take action" for Online Service Websites call the Javascript APIs provided by "iAM Smart" to get its running context or ask "iAM Smart" to execute some actions on behalf of itself.

For details, please refer to corresponding system workflows described in Section 3.

3. IMPLEMENTATION PROCEDURE

3.1 PREREQUISITE

(a) Work out the usage of "iAM Smart" for the Online Service application (e.g., form filling, digital signing, etc.) and determine the authorisation scopes (e.g., form filling authorisation for form filling) required
(b) Design the lifecycle management of the API data encryption/decryption key (e.g., get, renewal)
(c) Apply the digital certificates for encipherment as KEK (i.e., for API data encryption/decryption key)
(d) Study the specifications of relevant "iAM Smart" APIs and Online Service callback APIs
(e) Prepare the redirect URIs and/or URL scheme (or Universal Link/App Link) (for invoking Online Service App to return authorisation code to Online Service server) for each Online Service callback endpoint to be implemented for the Online Service application
(f) Work out and enable the bi-directional network connectivity between Online Service server and "iAM Smart" System
(g) Register Online Service application to "iAM Smart" System and get the corresponding client ID, client secret and/or URL scheme (or Universal Link/App Link) (for invoking "iAM Smart" Mobile App)
(h) Configure client authentication for "iAM Smart" System in Online Service callback endpoint (optional)
(i) Study the relevant system workflows and start implementation
(j) Study the "iAM Smart In-App Browser Integration Technical Reference" to familiar with

the integration of the "iAM Smart" in-app browser

3.2 ONLINE SERVICE AND "IAM SMART" SYSTEM INTERACTION DESIGN

3.2.1 Interaction between Online Service and "iAM Smart" Mobile App

The "iAM Smart" API is designed to provide better user experience when two forms of Online Service client terminal, Online Service App and Online Service Website (i.e., browser, including in-app browser) are interacting with "iAM Smart" Mobile App in the same device and different devices. There are four scenarios for the interactions: (1) ***for authentication using "iAM Smart"*** (2) ***for authorisation of anonymous "iAM Smart" API access*** (e.g., Anonymous Form Filling) (3) ***for performing Online Service business operations with authentication using "iAM Smart"*** (e.g., Form Filling by "iAM Smart") and (4) ***for requesting in-app browser to provide information or take action*** as follows:

1. For authentication using "iAM Smart", Online Service should invoke the "iAM Smart" API "Request QR Page". By default, this API returns a webpage with a QR code for scanning by "iAM Smart" Mobile App in different device.

    - When Online Service detects that the Online Service client terminal is a browser running in personal computer ("PC"), the API request parameter "brokerPage" can be kept as "false" and QR Code webpage will be shown.

    - When Online Service detects that the Online Service client terminal is a browser running in mobile platform, the API request parameter "brokerPage" can be set to "true". The broker page of "iAM Smart" System will show a button to invoke the "iAM Smart" Mobile App. If "iAM Smart" Mobile App is not installed in the same device, QR Code webpage will be shown.

    - When Online Service client terminal is Online Service App, it should detect the existence of "iAM Smart" Mobile App in the device using program code. If the "iAM Smart" Mobile App is installed in the same device, Online Service App should invoke "iAM Smart" Mobile App using URL Scheme with the API parameters. Otherwise, Online Service App should make use the device browser to invoke the "iAM Smart" API with "brokerPage" set to "false" such that QR Code webpage will be shown.

2. For authorisation of anonymous "iAM Smart" API access, Online Service will first invoke the relevant anonymous "iAM Smart" APIs and get the "ticketID". Online Service will then invoke the "iAM Smart" API "Request QR Page" using the "ticketID" to get the accessToken and Tokenised ID for obtaining the result of the relevant anonymous "iAM Smart" API. Same detection for client terminal by Online Service as in authentication using "iAM Smart", is applied.

3. To perform Online Service business operations with authentication using “iAM Smart” such as “Form Filling”, “iAM Smart” System will return the immediate API response with parameters “authByQR” and “ticketID” (optional) to Online Service server before the final result is returned to Online Service using Online Service callback API. Online Service should determine the next action based on “authByQR”:

    - When it is “false”, Online Service client terminal (both browser and Online Service App) should invoke the “iAM Smart” Mobile App using URL Scheme with the “ticketID”. Online Service client terminal should keep polling Online Service server for the result returned to relevant Online Service callback API.
    - Otherwise, Online Service client terminal should show instructions to inform “iAM Smart” user to complete the request in “iAM Smart” Mobile App (“iAM Smart” System will also notify “iAM Smart” Mobile App using push message) and keep polling Online Service server for the result returned to relevant Online Service callback API.

4. To request in-app browser to provide information or take action, Online Service should check if its website is being accessed in the in-app browser and the eIDJSBridge object exists before calling the Javascript APIs that are provided by “iAM Smart”.

The table below summarises the scenarios:

Scenario	**Online Service App/"iAM Smart" Mobile App**	**Online Service Website/"iAM Smart" Mobile App**
Authentication / Authorisation for anonymous "iAM Smart" API (Same device)	• Determine "iAM Smart" Mobile App existence by program code • Invoke "iAM Smart" Mobile App using URL scheme with API parameters	• Determine browser platform by program code • Invoke "Request QR Page" with "brokerPage" set to "true"
Authentication / Authorisation for anonymous "iAM Smart" API (Different device)	• Determine "iAM Smart" Mobile App existence by program code • Invoke "Request QR Page" using device browser with "brokerPage" set to "false"	• Determine browser platform by program code: – If browser in PC, invoke "Request QR Page" with "brokerPage" set to "false" – If browser in mobile, invoke "Request QR Page" with "brokerPage" set to "true"
Business operations with authentication using "iAM Smart" (Same device)	• Determine by "authByQR" is "false" • Invoke "iAM Smart" Mobile App using URL scheme with "ticketID"	
Business operations with authentication using "iAM Smart" (Different device)	• Determine by "authByQR" is "true" • Show instructions to inform "iAM Smart" user to complete the request in "iAM Smart" Mobile App and keep polling Online Service server for result returned to relevant Online Service callback API	
Requesting in-app browser to provide information or take action	• check if its website is being accessed in the in-app browser and the eIDJSBridge object exists • Take actions according to Online Service's own business requirements based on the returned information for the Javascript API call • Note: Chapters 3.2.2, 3.2.3 and 3.3 do not apply to the Javascript APIs. They are only required for HTTPS requests and callbacks.	

Specific for the situation when ***"iAM Smart" Mobile App and Online Service App are in the same device***,

(a) In the scenario (2) when invoking "iAM Smart" Mobile App (i.e., "iAM Smart" API: Open "iAM Smart" Mobile App for Getting Context), Online Service App is required to provide "source" and "redirectURI" parameters. Online Service can specify "source" parameter as "App_Scheme" (for URL Scheme) or "App_Link" (for iOS Universal Link/Android App Link) depending on its implementation method and the information submitted at Online

Service registration. The "source" provides information for how "iAM Smart" Mobile App can invoke Online Service App to return Online Service the authCode for authorisation of anonymous "iAM Smart" API access (i.e., "iAM Smart" API: Callback with authCode to Online Service App):

- If Online Service specify "source" = "App_Scheme", then "iAM Smart" System will consider it as Online Service App and using URL Scheme to invoke the Online Service App. Online Service should specify the corresponding URL Scheme in "redirectURI" and "iAM Smart" System will verify it with the URL Scheme submitted at Online Service registration.
- If Online Service specify "source" = "App_Link", then "iAM Smart" System will consider it as Online Service App and using Universal Link/App Link to invoke the Online Service App. Online Service should specify the corresponding Universal Link/App Link in "redirectURI" and "iAM Smart" System will verify it with the Universal Link/App Link submitted at Online Service registration.

(b) In the scenario (3) when Online Service backend invokes the required "iAM Smart" API (e.g., "iAM Smart" API: Request Form Filling), Online Service backend is required to provide "source" parameter. Online Service can specify "source" parameter as "App_Scheme" (for URL Scheme) or "App_Link" (for iOS Universal Link/Android App Link) depending on its implementation method and the corresponding scheme or link information submitted to "iAM Smart" System during Online Service registration. "iAM Smart" System will query such scheme or link information using the "source" provided for "iAM Smart" Mobile App to invoke Online Service App.

3.2.2 Process of Common API Request and Response Parameters

There are common parameters defined in the HTTP request header, HTTP request body of "iAM Smart" POST APIs. Common parameters are also defined in the "iAM Smart" POST API response body and Online Service callback API request body. This section describes how these common parameters should be prepared and processed by Online Service. Relevant information can be found in the "iAM Smart" API Specification.

Common parameters in the HTTP request header and body of "iAM Smart" POST APIs:

Request	Parameters	Description
Request Header	clientID	The Client Identity provided by "iAM Smart" System during Online Service registration.

Request	Parameters	Description
	signatureMethod	Name of signature algorithm used to compute the POST request signature Currently only support HmacSHA256 and its value must be set to "HmacSHA256", case sensitive.
	timestamp	The timestamp is a number expressed in the number of milliseconds since January 1, 1970 00:00:00 GMT. It must be greater than or equal to the timestamp of previous request submitted by Online Service. "iAM Smart" System compares the timestamp with system time and treats it as valid when the difference is within 60 seconds.
	nonce	A unique, non-repetitive random string for each request submitted by Online Service. It is used with timestamp to prevent Replay Attack. Its value cannot be repeated within 60 seconds.
	rateLimitFactor	Base64 encoded rate limit factor required for specific functions. It will be cross-checked with the content of the request body, if applicable.
	signature	It is a HmacSHA256 digital signature signed by Online Service using its client secret (paired with the clientID) provided by "iAM Smart" System during Online Service registration. The object to be signed includes the parameters concatenating in the following sequence: *clientID + signatureMethod + timestamp + nonce + encrypted request body* All parameters are case sensitive.
Request Body	content	Its value is created using API data encryption. It is BASE64 encoded with the following components concatenating in the following sequence: Content value: *4 bytes IV length + IV + ciphertext* IV is the initialisation vector of the encryption algorithm "AES/GCM/NoPadding" used in API data encryption. The ciphertext is the encrypted textual content of request body of the "iAM Smart" POST API. Details of API data encryption can be referred to Section 3.3.

Common parameters in the "iAM Smart" POST API response body and Online Service callback API request body:

Response/ Callback	Parameters	Description
Response/ Callback Body	txID	It is a unique transaction ID allocated by "iAM Smart" System for the Online Service request. Online Service is suggested to record this transaction ID for future reference or problem diagnostics.

Response/ Callback	Parameters	Description
	code	It is the return code of the processing result of the "iAM Smart" API invoked by Online Service. For details, please refer to "iAM Smart" API Specification.
	message	It is the text description of the "code" parameter.
	secretKey	It exists only in Online Service callback API request body. Its value is the encrypted CEK of Online Service (in AES256 format) using Online Service's KEK based on RSA algorithm. It is BASE64 encoded. Details of CEK and KEK can be referred to Section 6.1 of "iAM Smart" API Specification.
	content	Its value is created using API data encryption. It is BASE64 encoded with the following components concatenating in the following sequence: Content value: *4 bytes IV length + IV + ciphertext* IV is the initialisation vector of the encryption algorithm "AES/GCM/NoPadding" used in API data encryption. The ciphertext is the encrypted textual content of the JSON data inside "content" of the "iAM Smart" POST API response body and Online Service callback APIs request body. Details of API data encryption can be referred to Section 3.3.

Common parameters in JSON data of the "iAM Smart" APIs request body:

Request/ Callback	Parameters	Description
Request Body	businessID	It is a unique identifier generated by Online Service for identifying the business request of Online Service for invoking the "iAM Smart" API. It will be returned to Online Service via the corresponding Online Service callback API. For Online Service, it is used to match the result returned via Online Service callback API and thus should not be repeated. It is recommended to use 36-byte UUID number (ASCII character set) for this parameter.
	accessToken	It is the accessToken Online Service retained after authentication of the "iAM Smart" user and must be still valid. accessToken can be used multiple of times before expiry except that requested through the anonymous "iAM Smart" APIs which can only be used once.
	openID	It is the Tokenised ID of the "iAM Smart" user. Online Service should ensure the accessToken and Tokenised ID are in pair and belongs to the target "iAM Smart" user for the request.

Request/ Callback	Parameters	Description
	source	It can be "App_Scheme" (for URL Scheme), "App_Link" (for iOS Universal Link/Android App Link), or the user agent's name value of the browser. Please refer to Appendix B of the "iAM Smart" API Specification for the list of source values. It is used by "iAM Smart" System to determine how to process the interaction with the Online Service client terminal after receiving the "iAM Smart" API call. Please refer to Section 3.2.1 for details.
	redirectURI	It is the callback URI of Online Service. It is used by "iAM Smart" System to return the result via Online Service callback API and its value must match with the value provided by Online Service during registration to "iAM Smart" System.
	state	It is an optional parameter to prevent CSRF attack. The value of "state" is defined and managed by Online Service. It is recommended to use 36-byte UUID number (ASCII character set) or less with random value. It will be returned to Online Service via the corresponding callback API and Online Service is suggested to verify this value to prevent CSRF attack.

3.2.3 Process of Common Errors Returned from "iAM Smart" APIs

There are two tiers of errors returned from "iAM Smart" System: HTTP error and application error returned from "iAM Smart" APIs (given in parameter "code" or "error_code" in the POST response, Online Service callback API and URL Scheme). For application error, the parameter "message" presents the error description.

Common application errors are listed below:

Error Code	Error Description	Parameters in POST request	Suggested Action
Common Error Code			
D20000	unknown exception	N/A	Unknown error happened in "iAM Smart" System. Retry later or contact "iAM Smart" System administrator if problem persists.
D20001	parameter {%s} is missing	All parameters in request	Check parameter in {%s} is provided

Error Code	Error Description	Parameters in POST request	Suggested Action
D20002	empty parameter {%s}	All parameters in request	Check parameter in {%s} provided has correct value and format
D20003	invalid parameter {%s}	All parameters in request	Check parameter in {%s} provided is defined in the corresponding API
D20004	duplicated request	“timestamp”, “nonce”	Check same “timestamp” and “nonce” are used in different requests submitted
D20005	unsupported signature method	“signatureMethod”	Check signature method is supported by “iAM Smart” System and its value must be “HmacSHA256”, case-sensitive
D20006	signature verification failed	“signature”	1. Check correct client secret is used to calculate signature 2. Check any missing fields or incorrect field sequence when generating signature
D20007	unsupported source	“source”	For Online Service Website, check browser user agent value is one of the browsers supported by “iAM Smart” System (Please refer to Appendix B of “iAM Smart” API Specification for details.) For Online Service App, check the source is “App_Scheme” or “App_Link” (Please refer to Appendix B of “iAM Smart” API Specification for details).
D20008	unregistered redirectURI	“redirectURI”	Check value is registered with “iAM Smart” System
D20009	accessToken not exist or expired	“accessToken”	1. Check correctness of accessToken 2. Check accessToken is not expired
D20010	openID not exist	“openID”	Check validity of openID
D20011	duplicated businessID	“businessID”	Check uniqueness of businessID
D20012	insufficient scope	“scope”	Check authorisation scope(s) requested is/are sufficient for invoking the corresponding “iAM Smart” API
D20012	Forbidden	N/A	Check your endpoint is requested in the application form.
D20015	Scope mismatch	N/A	Online service shall verify the scopes approved for the client id in ESP and the

Error Code	Error Description	Parameters in POST request	Suggested Action
			scopes configured in Online Service catalogue.
Encryption/Decryption Error Code			
D30001	key encryption key not exist or expired	N/A	Check whether Online Service public key certificate (i.e., KEK) has been registered with "iAM Smart" System or it is expired
D30002	content encryption key not exist or expired	N/A	1. Check validity of CEK 2. Check CEK is not expired 3. Check value of "content" is BASE64 encoded
D30003	encryption exception	N/A	Unknown error when performing API data encryption in "iAM Smart" System: please contact "iAM Smart" System administrator if problem persists
D30004	decryption exception	N/A	API data decryption failed: check validity of encrypted "content", encryption process and validity of initialisation vector IV

A list of common HTTP errors which may be returned from "iAM Smart" System:

Error Code	Error Description	Suggested Action
302 Found	The requested resource resides temporarily under a different URI.	Browser directs or initiates request to new address.
400 Bad Request	The request could not be understood by the server due to malformed syntax.	Check syntax and correctness of the HTTP request and its parameters
401 Unauthorised	The request has not been applied because it lacks valid authentication credentials for the target resource.	No authorisation: please contact "iAM Smart" System Administrator
403 Forbidden	The server understood the request but is refusing to fulfil it.	1. "iAM Smart" System verifies the request header anti-replay parameter to be empty or illegal: check parameters and re-request 2. "iAM Smart" System request is detected as Replay: check if timestamp and nonce parameters meets the requirements 3. "iAM Smart" System verifies the signature verification parameter to be

Error Code	Error Description	Suggested Action
		empty or illegal: check the parameters and re-request 4. “iAM Smart” System signature verification fails: check if the signature rule is incorrect and re-request after modification 5. “iAM Smart” System business data decryption fails: check if the encryption rule is incorrect and re-request after modification
404 Not Found	The server has not found anything matching the Request-URI.	Check if interface address is correct
429 Too Many Requests	The user has sent too many requests in a given amount of time.	“iAM Smart” System is busy: resubmit request after a period of time (e.g., 30s) or contact “iAM Smart” System administrator if problem persists
500 Internal Server Error	The server encountered an unexpected condition which prevents it from fulfilling the request.	Retry later, or contact “iAM Smart” System administrator
503 Service Unavailable	The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.	Retry later, or contact “iAM Smart” System administrator

3.3 API DATA ENCYPTION AND DECRYPTION

HTTPS will be used to secure communication channels between Online Services and "iAM Smart" System. However, there are chances that the data transferred by HTTPS could be eavesdropped or manipulated by malicious service providers in some special situations. To protect the data in transit, an additional layer of data encryption will be applied to all APIs POST request and response body (except the API that Online Service request for getting the data encryption key). The response body of all Online Service callback POST APIs invoked by "iAM Smart" System will also be encrypted using the same key and method as described in this section. The process is as below:

1. Online Service requests for data encryption key from "iAM Smart" System;
2. "iAM Smart" System generates an AES256 symmetric data encryption key;
3. "iAM Smart" System stores the generated key and returns to the Online Service who initiated the request in secure manner;
4. For subsequent interactions between Online Service and "iAM Smart" System, business data will be encrypted by the generated key;
5. Both "iAM Smart" System and Online Service will use the same key to decrypt the API data;

   (Note: For the callback to Online Service, considering that CEK may time out when "iAM Smart" System received the request and consequently be regenerated, the encrypted CEK (probably new) will be returned to Online Service in the Online Service callback API. Online Service should use the CEK in callback body for data decryption, but does not retain this CEK.)
6. If the generated key is expired, Online Service have to request a new key and repeat the above process.

3.3.1 Workflow for Encryption and Decryption

The following diagram describes the detailed workflow how Online Service can request data encryption key, perform encryption and decryption with the key, and request for another new encryption key if the existing key is expired:

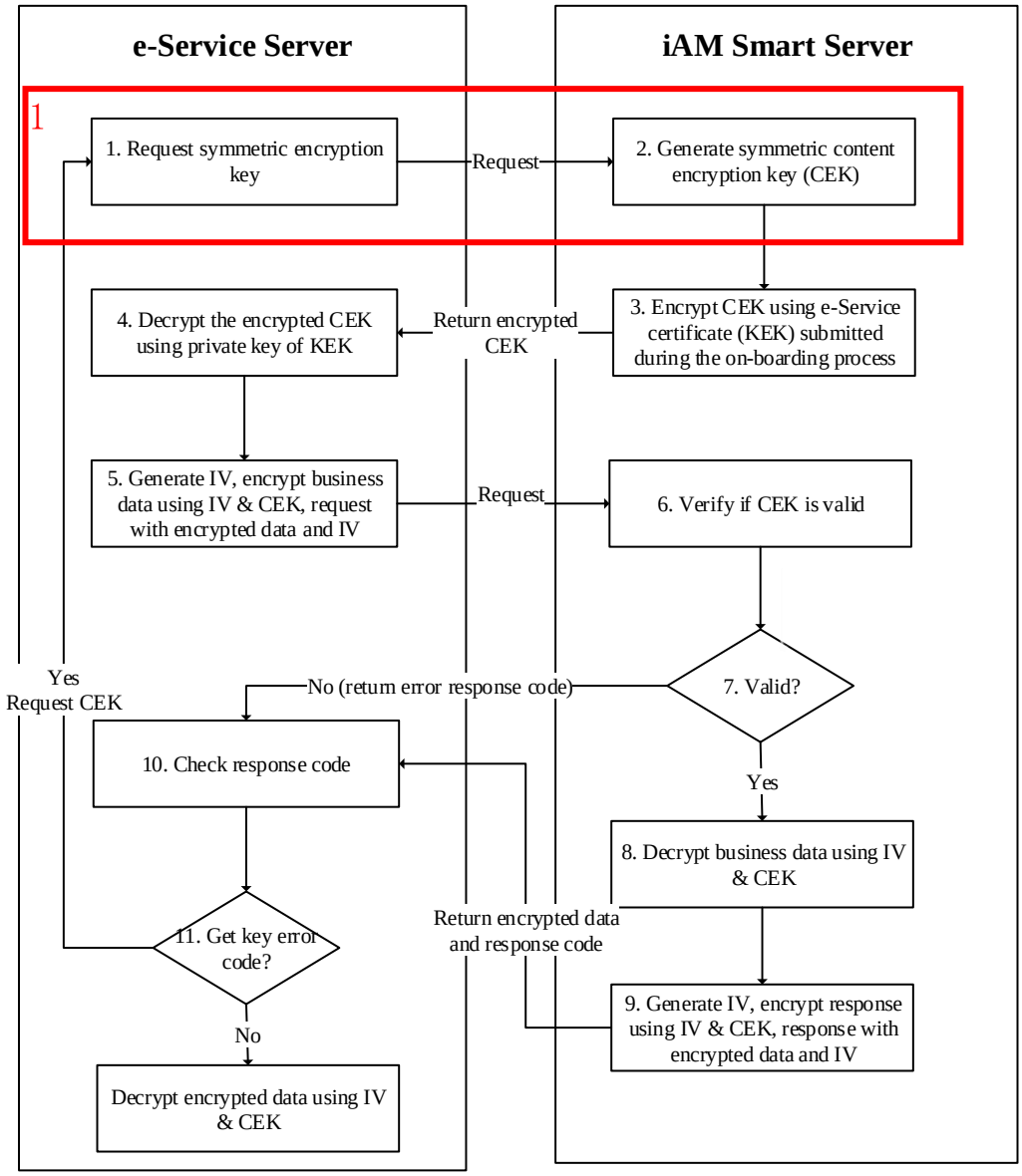


Figure-1 Application of Content Encryption Key and Procedures of Encryption and Decryption (Perform by Online Service)

Step 1. Online Service requests for data encryption key from "iAM Smart" System through invoking "iAM Smart" API;

"iAM Smart" API (POST: Request Symmetric Content Encryption Key);

Step 2. "iAM Smart" System generates an AES256 content encryption key ("CEK") for the Online Service and saves it in "iAM Smart" System;

Step 3. "iAM Smart" System encrypts the CEK using the public key certificate A ("KEK") provided by the Online Service in registration to "iAM Smart" System based on RSA algorithm, and returns the encrypted CEK, key expiry time to the Online Service;

Step 4. Online Service decrypts the encrypted CEK using the private key of its KEK and saves it and its key expiry time;

Step 5. Before Online Service calling APIs of "iAM Smart" System, Online Service should check whether its CEK is still valid and use it to encrypt all the JSON data of the POST request body as ciphertext using encryption algorithm "AES/GCM/NoPadding". This algorithm requires an initialisation vector ("IV") which is provided by Online Service. The ciphertext, IV and length of IV are then concatenated in the following sequence and BASE64 encoded to formulate the value of JSON name/value pair called "content":

Content value: *4 bytes IV length + IV + ciphertext*

Step 6-7. After receiving the Online Service request, "iAM Smart" System first checks whether the KEK, CEK of the Online Service exist or expired and returns with corresponding error code if required;

Step 8. "iAM Smart" System extracts the initialisation vector IV from the Online Service request and decrypts the JSON data of the POST request body using Online Service's CEK and IV. "iAM Smart" System then validates the request parameters and returns error response to Online Service if required;

Step 9. After "iAM Smart" System has processed the request, it generates its initialisation vector IV and encrypts all the JSON data of POST response (except "txID", "code" and "message") as ciphertext using the valid Online Service's CEK (i.e., CEK is not expired) and IV by the same encryption algorithm ("AES/GCM/NoPadding"):

- For "iAM Smart" POST API response, if CEK does not exist or expires, corresponding error code will be returned.
- For Online Service callback, if CEK is expired, "iAM Smart" System will re-generate a new CEK for the Online Service to perform the encryption.

The ciphertext of encrypted POST response / callback JSON data, IV and length of IV are then concatenated in the following sequence and BASE64 encoded to formulate the value of JSON name/value pair called "content":

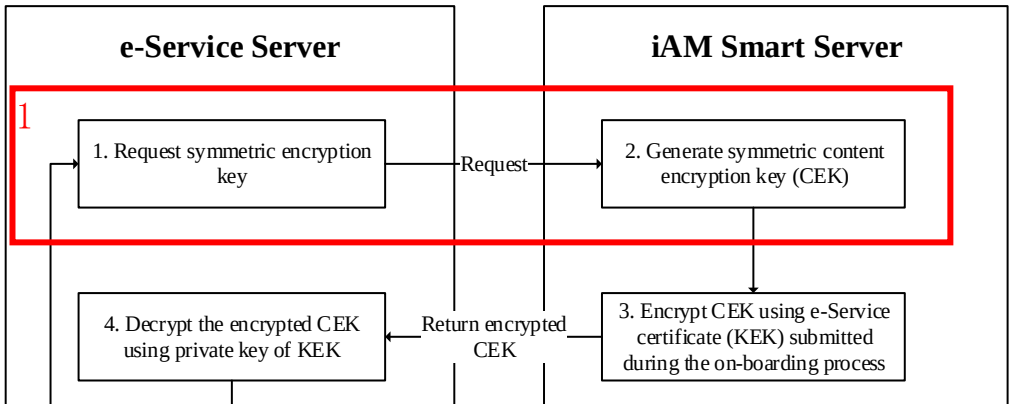
Content value: *4 bytes IV length + IV + ciphertext*

For Online Service callback, the CEK used for this encryption (e.g., new CEK) will also be encrypted using Online Service's KEK and BASE64 encoded, and

will be returned to Online Service as part of the POST callback body as JSON name/value pair called "secretKey";

Step 10-11. For the response received, the Online Service should first check whether the return code contains any key-related error code such as "key does not exist or has been expired". For "iAM Smart" POST API response, if there are no such key-related errors, Online Service should use its CEK and IV (extracted from "content") to decrypt the response data. For Online Service callback, Online Service should decrypt the CEK from "secretKey" parameter in the callback body using its KEK's private key and then use this CEK and IV (extracted from "content") to decrypt the callback data. Online Service should request for a new CEK when it is expired.

3.3.1.1 Implementing (1) POST: Request Symmetric Content Encryption Key



Pre-conditions

- Online Service can access the private key of its KEK.
- Online Service has registered to "iAM Smart" System and uploaded its KEK.

Post-conditions

- Online Service decrypts CEK received from "iAM Smart" System using the private key of its KEK and saves the CEK and its key expiry time.

Error conditions

- For common errors, please refer to Section 3.2.3.

Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- Online Service should ensure the KEK registered to “iAM Smart” System is valid and keeps the private key of KEK. The KEK used for this “iAM Smart” API should be bound to the same clientID allocated to Online Service by “iAM Smart” System.
- If the current CEK for Online Service is not expired, “iAM Smart” System will return the same CEK to Online Service with “issueAt” and “expiresIn” unchanged.
- For common parameters in request and response, please refer to Section 3.2.2.
- Response parameter “secretKey”: It is a CEK generated by “iAM Smart” System for the Online Service and is encrypted using the RSA algorithm and the latest Online Service's KEK. Online Service should use the private key of its KEK to decrypt “secretKey” to get the CEK.
- Response parameter “pubKey”: It is the latest Online Service’s KEK that is used to encrypt the CEK to form value of “secretKey”. Online Service may make use “pubKey” to locate the corresponding private key.
- Response parameter “issueAt”: It is the issue time of CEK expressed in the number of milliseconds since January 1, 1970 00:00:00 GMT.
- Response parameter “expiresIn”: It is the validity period of CEK expressed in milliseconds.

3.3.1.2 Implementing (2) POST: Revoke Symmetric Content Encryption Key

This API allows Online Service to revoke symmetric Content Encryption Key when necessary.

Pre-conditions

- Nil

Post-conditions

- Nil

Error conditions

- For common errors, please refer to Section 3.2.3.

Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Each Online Service can only request for one CEK. When CEK was revoked, Online Service is required to request for a new CEK.

3.3.2 Encryption & Decryption Scope and Examples

Please refer to “iAM Smart” API Specification.

3.3.3 Proper Handling of Encryption and Decryption

Online Service shall handle the following scenarios properly in their system design:

- Due to unexpected reasons, Online Service may receive Encryption/Decryption Error Code. Online service shall perform retry of the “Request Symmetric Content Encryption Key” API specified in section 2.3.6.1 of “iAM Smart” API Specification to obtain the latest CEK and try to decrypt content with such CEK when it is received.
- Due to operational needs, the “expiresIn” response parameter of the “Request Symmetric Content Encryption Key” API specified in section 2.3.6.1 of “iAM Smart” API Specification may dynamically change, Online Service shall calculate the expiry time of the CEK with “issueAt” and “expiresIn” response parameters.
- Under certain circumstances, “iAM Smart” API service may be unavailable for a certain period of time. The Online Service shall prepare the fallback procedures (e.g., switch to paper mode, prevent users from accessing “iAM Smart” functions, etc.)
- When there are too frequent API requests from an Online Service, the Online Service may receive HTTPS status Code 429, “Too Many Requests”. Online service shall store the CEK to avoid unnecessary “Request Symmetric Content Encryption Key” API request (see section 2.3.6.1 of “iAM Smart” API Specification)
- “iAM Smart” API Online Service has to use the old private key to decrypt the CEK until the expiry time. Online service shall match the public key returned from the “Request Symmetric Content Encryption Key” API response to determine which private key should be used for the decryption
- Even a new KEK is configured for effective in a specified period, the existing CEK will still be returned through the “Request Symmetric Content Encryption Key” API specified in section 2.3.6.1 of “iAM Smart” API Specification and encrypted by the old KEK until the expiry of the existing CEK. To use the new KEK, Online Service shall request a new CEK through the “Revoke Symmetric Content Encryption Key” and “Request Symmetric Content Encryption Key” APIs specified in sections 2.3.6.2 and 2.3.6.1 respectively of “iAM Smart” API Specification or to request a new CEK after its expiry through the same APIs.

3.4 WORKFLOWS FOR AUTHENTICATION

3.4.1 Scenario 1: Authentication (Online Service Website in Different Device)

The sequence diagram below shows how Online Service website leverages the “iAM Smart” System to perform user authentication when the “iAM Smart” Mobile App is installed in a separated mobile device.

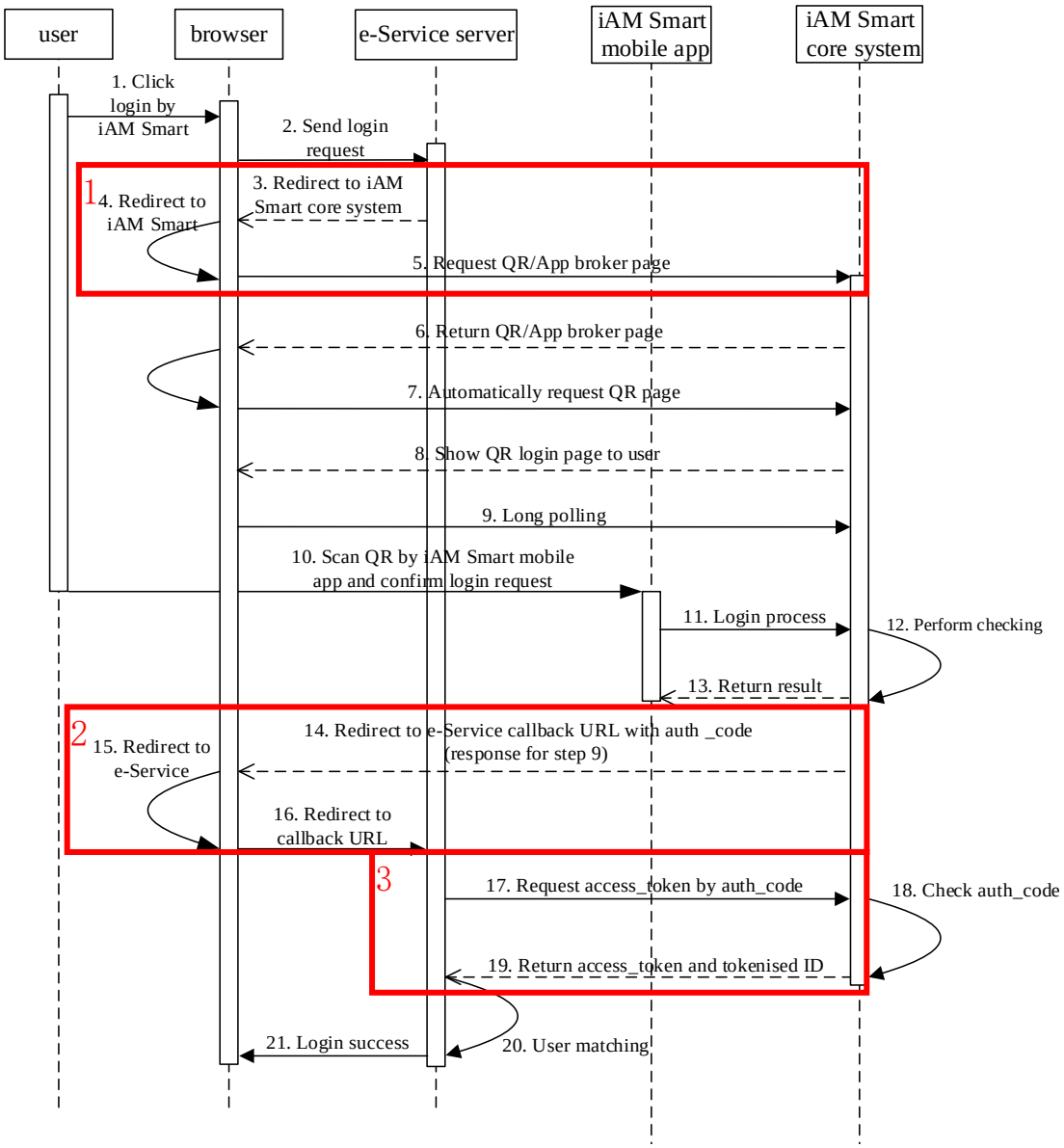


Figure-2 Authentication (Online Service Website in Different Device)

Step 1. User clicks “Login by iAM Smart” button on Online Service Website;

Step 2. Online Service Website initiates login request to Online Service Server with the browser's user agent value (for use as value for request parameter “source”);

Step 3-5. Online Service Server prepares the request parameters such as “clientID”, “redirectURI”, “source”, “scope”, etc. and constructs the GET request to invoke the “iAM Smart” API using browser redirection;

"iAM Smart" API (GET: Request QR page)

Step 6-8. "iAM Smart" System returns the broker page (if Online Service has set "brokerPage" to "true") and after the broker page fails to find the "iAM Smart" Mobile App, it will request QR page to be displayed in browser. If Online Service does not request for broker page (i.e., no broker page will be returned), the browser will directly display QR Code page;

Step 9. QR Code page of "iAM Smart" System polls "iAM Smart" System for the QR Code status;

Step 10-11. "iAM Smart" user logs in "iAM Smart" Mobile App to scan the QR Code and confirms Online Service's login request;

Step12-13. "iAM Smart" System verifies the validity of QR code and other necessary information, and responses the login result to "iAM Smart" Mobile App (e.g., login success and inform "iAM Smart" user to proceed in Online Service, invalid / expired QR code, etc.);

Step 14-16. QR Code page of "iAM Smart" System receives the polling result returned by "iAM Smart" System (i.e., response for the polling in Step 9) which includes authCode or any error code (e.g., "iAM Smart" user rejects the login request), assembles and invokes the Online Service callback API given in "redirectURI" using browser redirection;

Online Service Callback API (GET: Callback with authCode to Online Service Server)

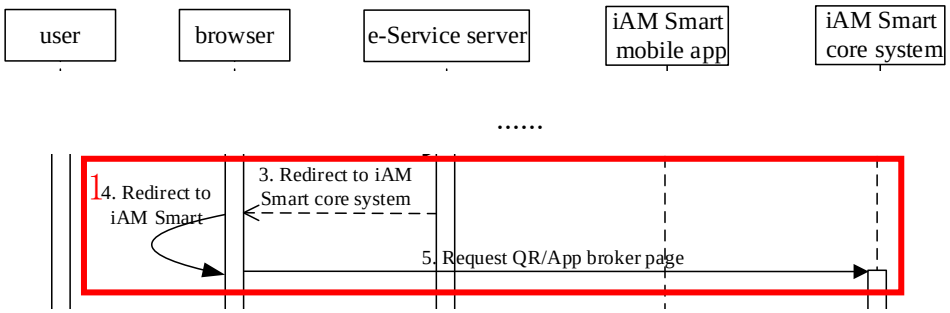
Step 17-19. When Online Service Server gets the authCode, it should invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e., openID) of the "iAM Smart" user. API data encryption is required;

"iAM Smart" API (POST: Request accessToken & Tokenised ID)

Step 20. Online Service Server then performs user matching using the Tokenised ID with its user repository and determines the result of this login request;

Step 21. The Online Service Website shows the login result (e.g., successful when Tokenised ID matched with a user account of Online Service).

3.4.1.1 Implementing (1) GET: Request QR page



Pre-conditions

- Online Service should determine all the authorisation scopes required for this authentication request.
- Online Service should detect the browser's user agent name.
- Online Service Website can use the optional “brokerPage” parameter (Details please refer to Section 3.2.1), or determine whether the browser and “iAM Smart” Mobile App are on different devices or not by using its program code (e.g., the browser's agent name, client configure check). For browser type supported by “iAM Smart” System, please refer to Appendix B of “iAM Smart” API Specification.
- Online Service should determine if it would generate the optional “state” request parameter to prevent CSRF attack. The “state” will be returned to Online Service for checking through the Online Service callback API for receiving the authorisation code from “iAM Smart” System in Step 16.

Post-conditions

- The browser will show the QR Code page of “iAM Smart” System with a “Return” button to allow user to return to Online Service.

Error conditions

- This “iAM Smart” API does not return JSON response (i.e., no application error will return). For common errors, please refer to Section 3.2.3.

Request Parameters

- Please refer to “iAM Smart” API Specification.

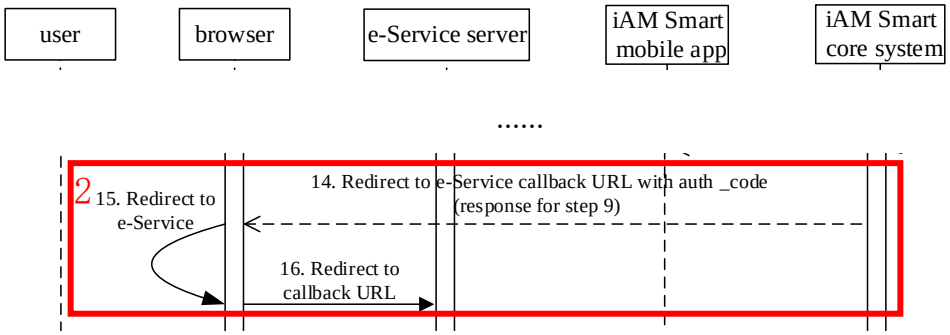
Notes

- For common parameters, please refer to Section 3.2.2.
- Request parameter “responseType”: Value must be 'code'.
- Request parameter “source”: Value should be the browser's user agent value.
- Request parameter “redirectURI”: This is Online Service URL for receiving the authorisation code. The value should be URL encoded. For details, please referred to

"iAM Smart" API Specification "Callback with authCode to Online Service Server". The value must be the same as provided during Online Service registration.

- Request parameter "scope": All authorisation scopes required for this authentication request should be specified. Multiple values should be separated by space (e.g., eidapi_auth ediapi_eMe eidapi_sign). The value should be URL encoded.
- Request parameter "ticketID": Not required in this scenario.
- Request parameter "lang": Display language of the QR code page.
- Request parameter "state": The value should be URL encoded.
- Request parameter "brokerPage": Value should be set to "false" in this scenario. For details, please refer to Section 3.2.1.

3.4.1.2 Implementing (2) GET: Callback with authCode to Online Service Server



Pre-conditions

- "iAM Smart" user has scanned the QR Code using "iAM Smart" Mobile App and authorised Online Service's authentication request.
- "iAM Smart" user can also reject the authentication request in "iAM Smart" Mobile App.

Post-conditions

- authCode will be expired in 1 minute and Online Service can only use the authCode once for requesting the accessToken from "iAM Smart" System.

Error conditions

- This Online Service callback API is a HTTP GET request. If parameter "error_code" is returned, it means the authentication request is failed.

Error Code	Error Description	Suggested Action
error_code - D40000	User cancelled authentication request	Inform user the authentication request is cancelled
error_code - D40001	User rejected authentication request	Inform user the authentication request is rejected
error_code - D40002	Failed to authenticate	Inform user the authentication request is failed and retry later

The error_code D40003 (authentication request timeout) does not appear in this scenario. For the QR code timeout or request confirmation timeout in the "iAM Smart" Mobile App, message will be prompted in the corresponding user interface.

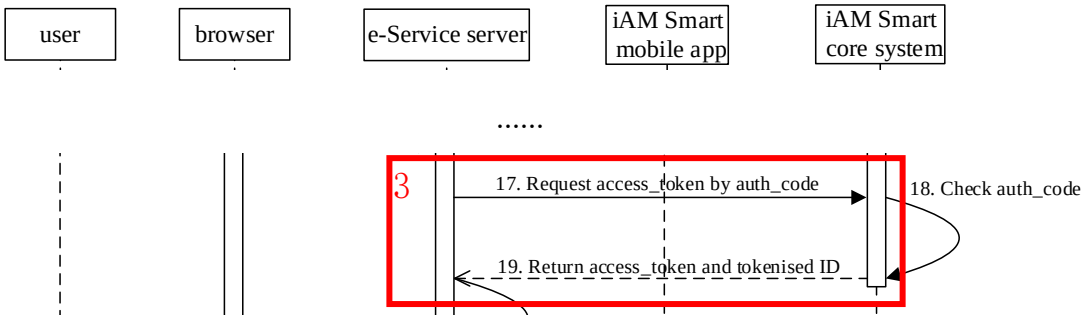
Callback Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- Callback parameter "businessID": Not required in this scenario.
- Callback parameter "code": It is the authCode for requesting the accessToken from "iAM Smart" System. Its validity is 1 minute and can only be used once.
- If "iAM Smart" user rejects the authentication request or other errors occur, "error_code" instead of "code" will be returned.

3.4.1.3 Implementing (3) POST: Request accessToken & Tokenised ID



Pre-conditions

- Online Service should request the accessToken using a valid authCode. authCode is valid for 1 minute and should not be used more than once.
- API data encryption is required.

Post-conditions

- API data decryption is required.
- Online Service should save the accessToken and Tokenised ID for subsequent access to other "iAM Smart" APIs.
- The validity period of accessToken is given in the response parameter "expiresIn".
- The accessToken can be used multiple of times before expiry.
- Depending on its business needs, Online Service may also keep and check other response parameters such as "userType", "scope" to determine subsequent actions to access to other "iAM Smart" APIs.

Error conditions

- For common errors, please refer to Section 3.2.3. Other error for this “iAM Smart” API includes:

Error Code	Error Description	Suggested Action
code - D40004	authCode not exist or expired	Check authCode, or redo authentication

Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- Request parameter “code”: It is the authCode for requesting the accessToken from “iAM Smart” System. Its validity is 1 minute and can only be used once.
- Request parameter “grantType”: Value must be “authorization_code”.
- Response Parameters “accessToken”, “tokenType”, “issueAt”, “expiresIn”: These are the accessToken information of the “iAM Smart” user. The accessToken is issued by “iAM Smart” System at time specified by “issueAt” and valid for the period specified in “expiresIn”. The “tokenType” must be “Bearer”.
- Response parameter “openID”: It is the Tokenised ID of the “iAM Smart” user for the Online Service. The Tokenised ID of an “iAM Smart” user for different Online Services are different. Online Service can use it to verify if the “iAM Smart” user has linked with any user account in Online Service user repository.
- Response parameter “lastModifiedDate”: It is the last update timestamp of “iAM Smart” user's CFD information in his/her “iAM Smart” Profile.
- Response parameter “userType”: It is the type of the “iAM Smart” account owned by the “iAM Smart” user. It is either “default” (i.e., default “iAM Smart”) or “sign” (i.e., “iAM Smart” with digital signing function) and only “userType” with value “sign” is given an “iAM Smart” e-Cert to carry out digital signing operation.
- Response parameter “scope”: It is the authorisation scope of the accessToken and it should match with the authorisation scope requested by Online Service in Step 5.

3.4.2 Scenario 2: Authentication (Online Service Website in Same Device)

The sequence diagrams below show how an Online Service website leverages the "iAM Smart" System to perform user authentication when the "iAM Smart" Mobile App is installed in the same mobile device.

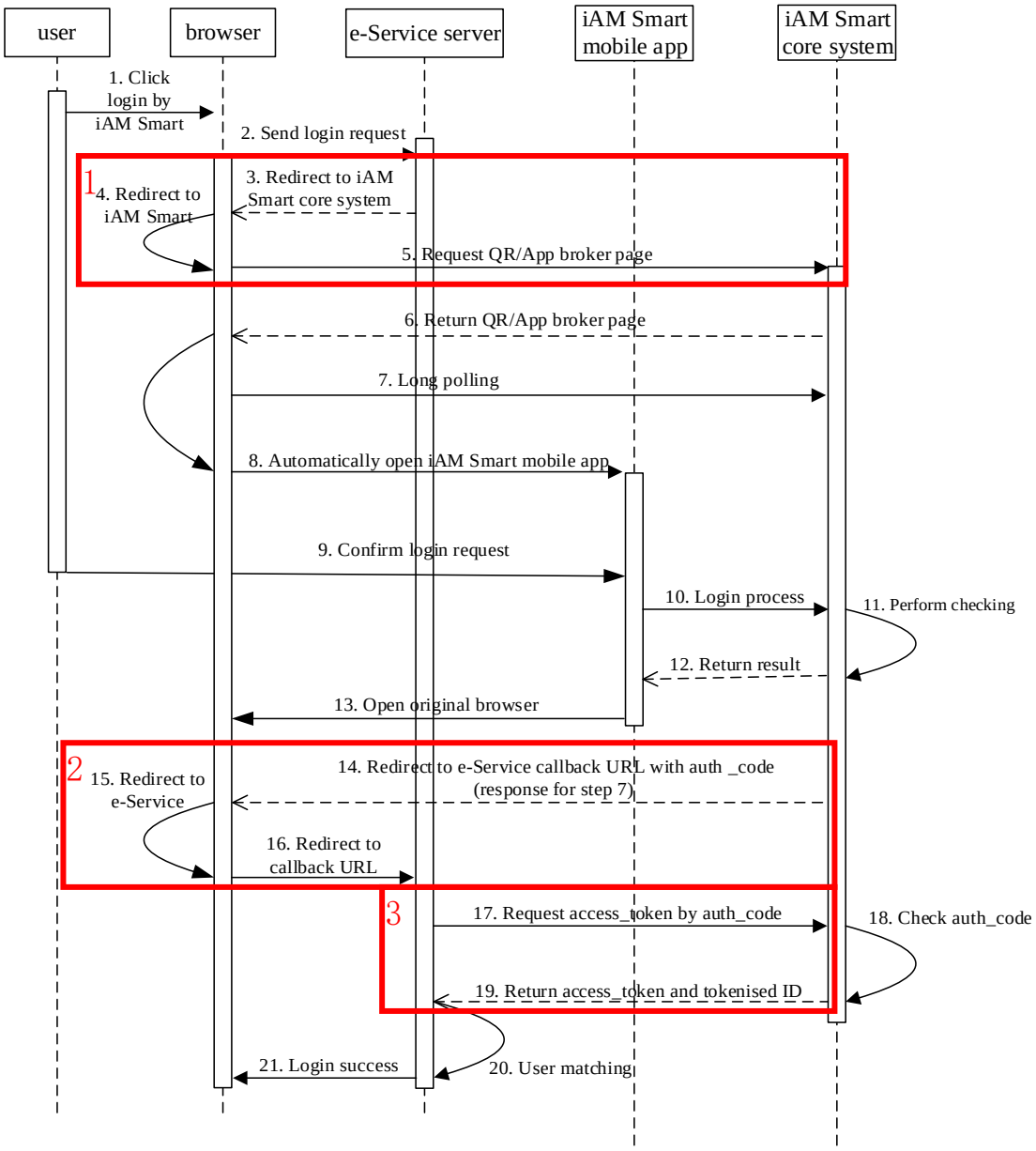


Figure-3 Authentication (Online Service Website in Same Device)

Step 1. User clicks "Login by iAM Smart" button on Online Service Website;

Step 2. Online Service Website initiates login request to Online Service Server with the browser's user agent value (for use as value for request parameter "source");

Step 3-5. Online Service Server prepares the request parameters such as "clientID", "redirectURI", "source" (set as browser's user agent value), "scope", "brokerPage"

(set as "true"). etc. and constructs the GET request to invoke the "iAM Smart" API using browser redirection;

"iAM Smart" API (GET: Request QR Page)

Step 6. "iAM Smart" System returns the broker page to the Online Service Website;

Step 7. Broker page of "iAM Smart" System polls "iAM Smart" System for further action;

Step 8-9. Broker page invokes the "iAM Smart" Mobile App in the same device automatically and sends it the relevant parameters. "iAM Smart" user logs in the "iAM Smart" Mobile App;

Step 10. "iAM Smart" user confirms the Online Service login request in "iAM Smart" Mobile App and "iAM Smart" Mobile App submits the login request to "iAM Smart" System;

Step 11-13. "iAM Smart" System verifies the validity of necessary information, and responses the login result to "iAM Smart" Mobile App. "iAM Smart" Mobile App invokes the original Online Service browser (i.e., using the browser's user agent value submitted);

Step 14-16. Broker page of "iAM Smart" System receives the polling result returned by "iAM Smart" System (i.e., response for the polling in Step 7) which includes authCode or any error code (e.g., "iAM Smart" user rejects the login request), assembles and invokes the Online Service callback API given in "redirectURI" using browser redirection;

Online Service Callback API (GET: Callback with authCode to Online Service Server)

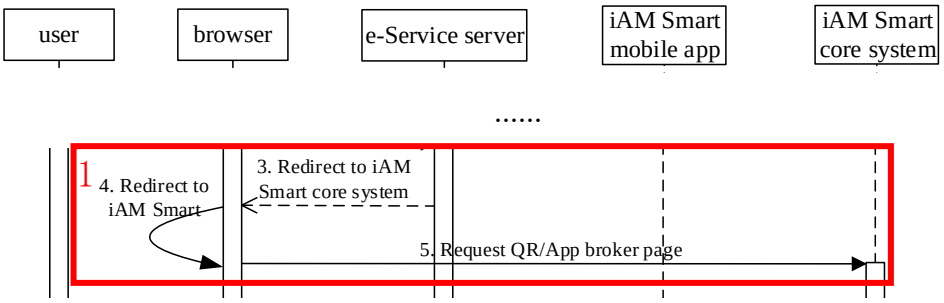
Step 17-19. When Online Service Server gets the authCode, it should invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e., openID) of the "iAM Smart" user. API data encryption is required;

"iAM Smart" API (POST: Request accessToken & Tokenised ID)

Step 20. Online Service Server performs user matching using the Tokenised ID with its user repository and determines the result of this login request;

Step 21. The Online Service Website shows the login result (e.g., successful when Tokenised ID matched with a user account of Online Service).

3.4.2.1 Implementing (1) GET: Request QR Page



Pre-conditions

- Please refer to Section 3.4.1.1.

Post-conditions

- Online Service browser will display the broker page returned by "iAM Smart" System and "iAM Smart" Mobile App is launched automatically.

Error conditions

- Please refer to Section 3.4.1.1.

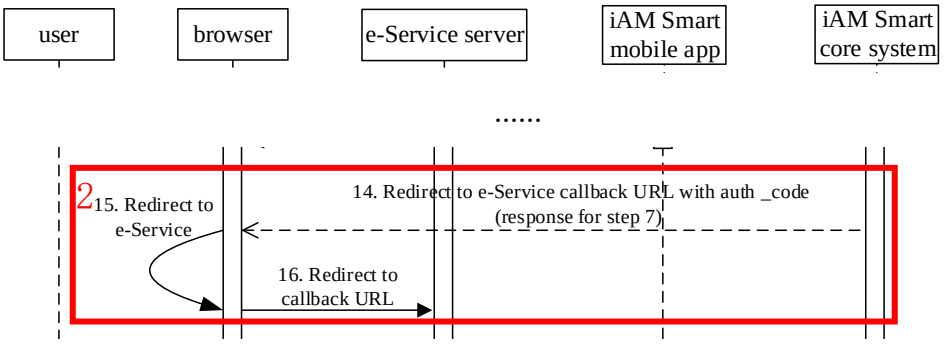
Request Parameters

- Please refer to Section 3.4.1.1.

Notes

- Please refer to Section 3.4.1.1 except the following parameter:
  - Request parameter "brokerPage": Value should be set to "true" in this scenario. Broker page invokes the "iAM Smart" Mobile App in the same device automatically. For details, please refer to Section 3.2.1.

3.4.2.2 Implementing (2) GET: Callback with authCode to Online Service Server



Pre-conditions

- "iAM Smart" user has authorised the authentication request of Online Service in "iAM Smart" Mobile App.
- "iAM Smart" user can also reject the authentication request in "iAM Smart" Mobile App.

Post-conditions

- Please refer to Section 3.4.1.2.

Error Conditions

- Please refer to Section 3.4.1.2.

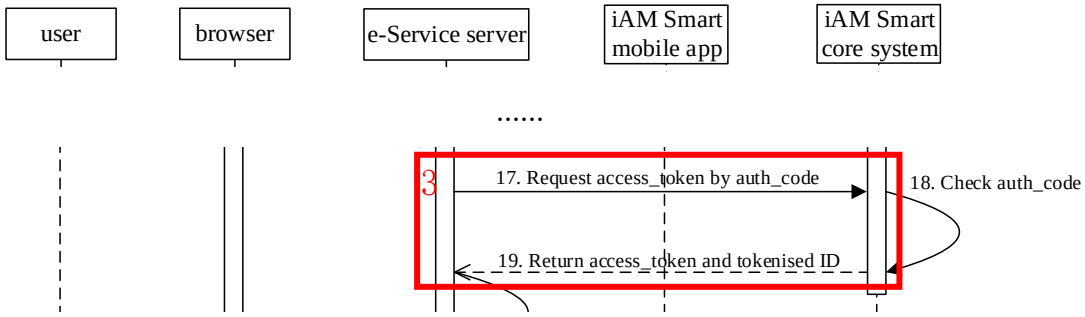
Callback Parameters

- Please refer to Section 3.4.1.2.

Notes

- Please refer to Section 3.4.1.2.

3.4.2.3 Implementing (3) POST: Request accessToken & Tokenised ID



Please refer to Section 3.4.1.3.

3.4.3 Scenario 3: Authentication (Online Service App in Different Device)

The sequence diagram below shows how Online Service mobile application leverages the "iAM Smart" System to perform user authentication when the "iAM Smart" Mobile App is installed in a separated mobile device.

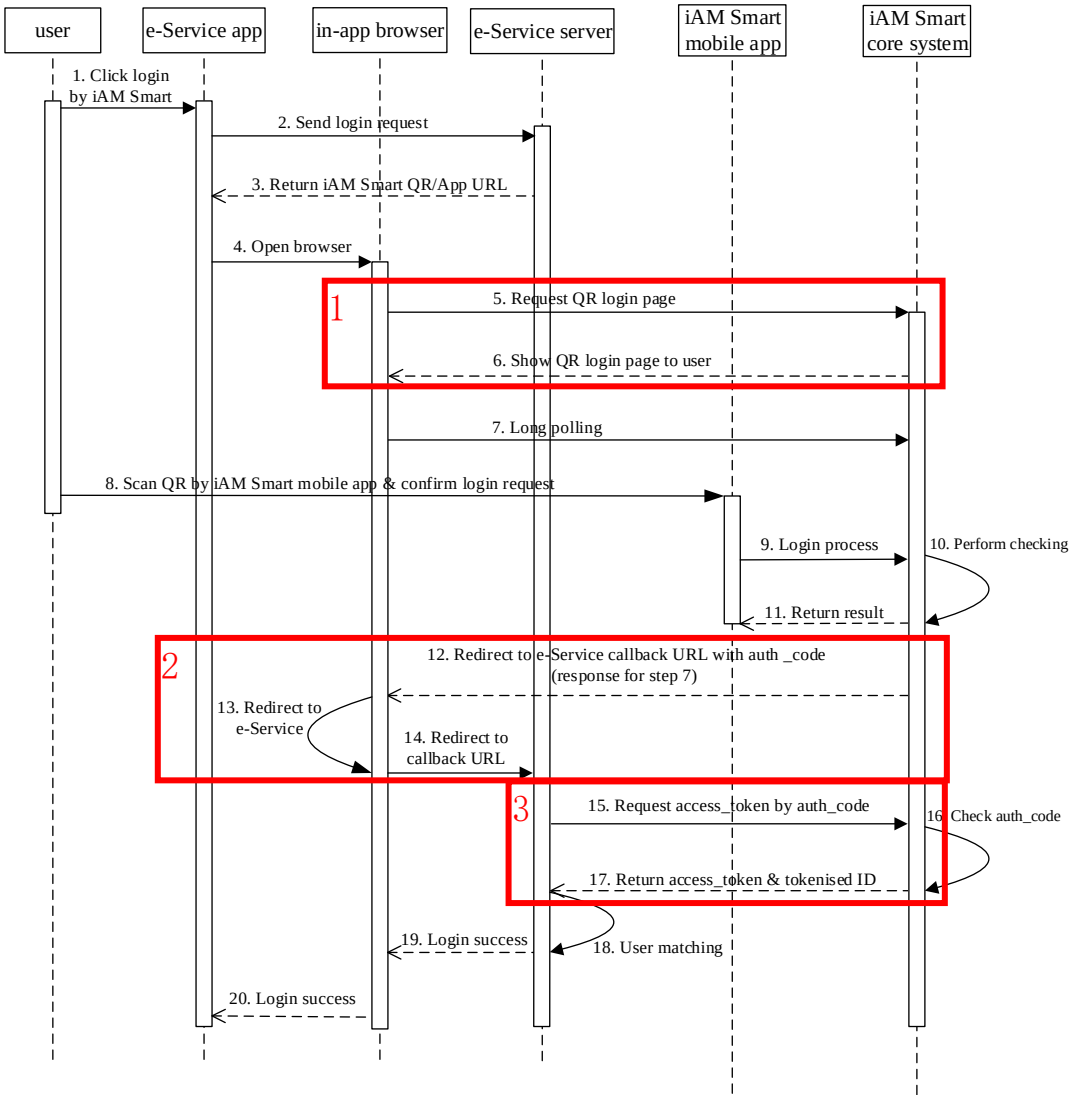


Figure-4 Authentication (Online Service App in Different Device)

Step 1. User clicks "Login by iAM Smart" button in Online Service App;

Step 2-3. Online Service App determines there is no "iAM Smart" Mobile App in the device using program code, requests Online Service Server to prepare the request parameters including "clientID", "redirectURI", "scope", "source" (set as in-app browser's user agent value), "brokerPage" (set as "false"), etc. and constructs the GET request to invoke the "iAM Smart" API by Online Service App;

Step 4. Online Service App invokes in-app browser (Safari for iOS, Chrome for Android) to submit the GET request and keeps synchronise with Online Service server for the status of the login request;

Step 5-6. Browser opens the GET request to invoke the "iAM Smart" API and QR Code page will be shown;

"iAM Smart" API (GET: Request QR page)

Step 7. QR Code page of "iAM Smart" System polls "iAM Smart" System for the QR Code status;

Step 8-9. "iAM Smart" user logs in "iAM Smart" Mobile App to scan the QR Code and confirms Online Service's login request;

Step10-11. "iAM Smart" System verifies the validity of QR code and other necessary information, and responses the login result to "iAM Smart" Mobile App (e.g., login success and inform "iAM Smart" user to proceed in Online Service, invalid / expired QR code, etc.);

Step 12-14. QR Code page of "iAM Smart" System receives the polling result returned by "iAM Smart" System (i.e., response for the polling in Step 7) which includes authCode or any error code (e.g., "iAM Smart" user rejects the login request), assembles and invokes the Online Service callback API given in "redirectURI" using browser redirection;

Online Service Callback API (GET: Callback with authCode to Online Service Server)

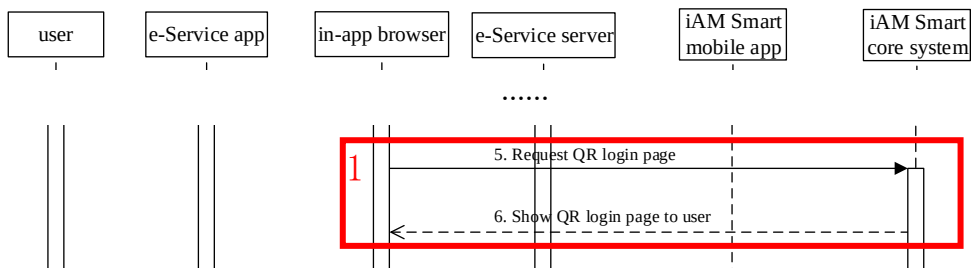
Step 15-17. When Online Service Server gets the authCode, it should invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e., openID) of the "iAM Smart" user. API data encryption is required;

"iAM Smart" API (POST: Request accessToken & Tokenised ID)

Step 18. Online Service Server then performs user matching using the Tokenised ID with its user repository and determines the result of this login request;

Step 19-20. The Online Service App shows the login result (e.g., successful when Tokenised ID matched with a user account of Online Service).

3.4.3.1 Implementing (1) GET: Request QR page



Pre-conditions

- Online Service should determine all the authorisation scopes required for this authentication request.
- Online Service should determine the “iAM Smart” Mobile App is not in the same device of Online Service App using program code.
- Online Service should invoke in-app browser (Safari for iOS, Chrome for Android) to submit the GET request, and transfer the browser's user agent name to “iAM Smart”.
- Online Service should determine if it would generate the optional “state” request parameter to prevent CSRF attack. The “state” will be returned to Online Service for checking through the Online Service callback API for receiving the authorisation code from “iAM Smart” System in Step 14.

Post-conditions

- The browser will show the QR Code page of “iAM Smart” System with a “Return” button to allow user to return to Online Service App.
- Online Service App should keep synchronising with Online Service server for the status of the login request.

Error conditions

- This “iAM Smart” API does not return JSON response (i.e., no application error will be returned). For common errors, please refer to Section 3.2.3.

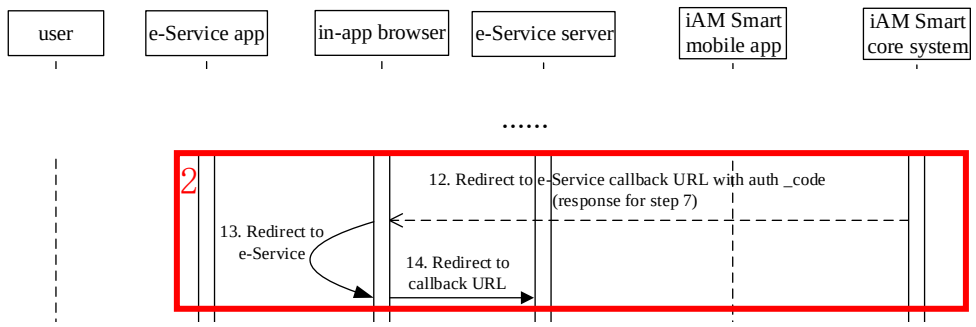
Request Parameters

- Please refer to “iAM Smart” API Specification.

Notes

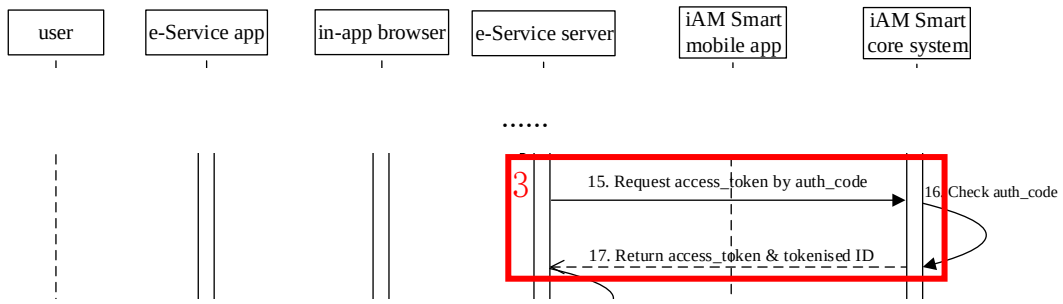
- Please refer to Section 3.4.1.1, except for the following parameter:
  - Request parameter “source”: Value should be the in-app browser's user agent value.

3.4.3.2 Implementing (2) GET: Callback with authCode to Online Service Server



Please refer to Section 3.4.1.2.

3.4.3.3 Implementing (3) POST: Request accessToken & Tokenised ID



Please refer to Section 3.4.1.3.

3.4.4 Scenario 4: Authentication (Online Service App in Same Device)

The sequence diagram below shows how an Online Service mobile application leverages the "iAM Smart" System to perform user authentication when the "iAM Smart" Mobile App is installed in the same mobile device.

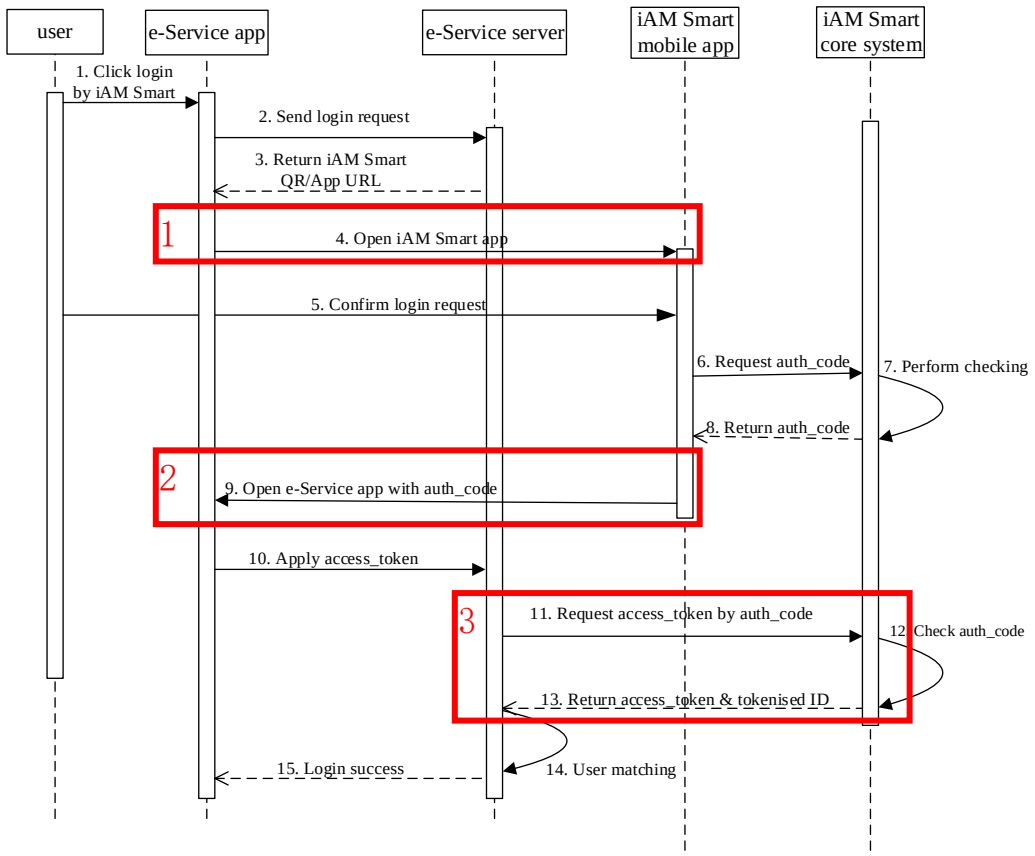


Figure-5 Authentication (Online Service App in Same Device)

Step 1. User clicks "Login by iAM Smart" button on Online Service App;

Step 2-3. Online Service App determines "iAM Smart" Mobile App is installed in the device using program code, requests Online Service Server to prepare the request parameters including "clientID", "redirectURI" (set as Online Service App URL Scheme or Universal Link/App Link depending on "source" parameter), "scope", "source" (set as "App_Scheme" or "App_Link"), etc. and constructs the URL Scheme to invoke "iAM Smart" Mobile App by Online Service App;

Step 4. Online Service App invokes "iAM Smart" Mobile App using URL Scheme with request parameters (set <Context> as "auth" in URL Scheme);

"iAM Smart" API (URL Scheme: Open "iAM Smart" Mobile App for Authentication)

Step 5-7. "iAM Smart" user logs in the "iAM Smart" Mobile App confirms the Online Service login request and "iAM Smart" Mobile App submits the login request to "iAM Smart" System;

Step 8. "iAM Smart" System verifies the necessary information of the login request and returns login result to "iAM Smart" Mobile App;

Step 9. "iAM Smart" Mobile App invokes Online Service App with required parameters using URL Scheme or Universal Link/App Link;

Online Service Callback API (Callback with authCode to Online Service App)

Step 10. Online Service App sends authCode, etc. to Online Service Server;

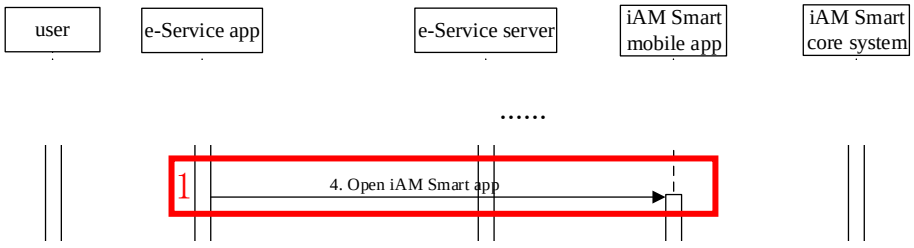
Step 11-13. When Online Service Server gets the authCode, it should invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e., openID) of the "iAM Smart" user. API data encryption is required;

"iAM Smart" API (POST: Request accessToken & Tokenised ID)

Step 14. Online Service then performs user matching using the Tokenised ID with its user repository and determines the result of this login request;

Step 15. Online Service App shows the login result (e.g., successful when Tokenised ID is matched with a user account of Online Service).

3.4.4.1 Implementing (1) URL Scheme: Open "iAM Smart" Mobile App for Authentication



Pre-conditions

- Online Service should determine all the authorisation scopes required for this authentication request.
- Online Service should determine the "iAM Smart" Mobile App is on the same device of Online Service App using program code.
- Online Service should determine if it would generate the optional "state" request parameter to prevent CSRF attack. The "state" will be returned to Online Service for checking through the Online Service callback API for receiving the authorisation code from "iAM Smart" System in Step 9.

Post-conditions

- "iAM Smart" Mobile App will be launched.
- Online Service App should keep synchronising with Online Service server for the status of the login request.

Error conditions

- Nil

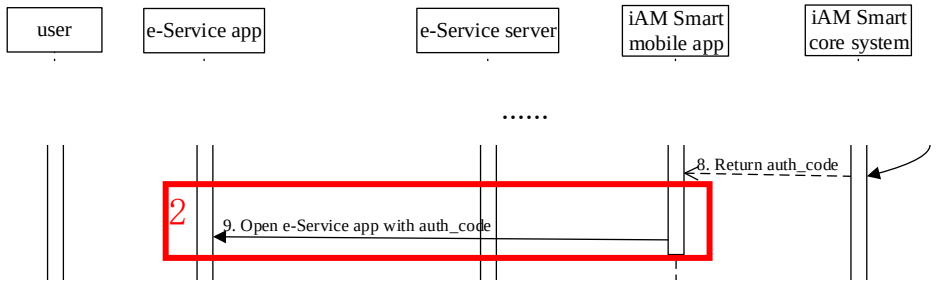
Request Parameters

- This "iAM Smart" API is a URL Scheme. Please refer to "iAM Smart" API Specification.

Notes

- Please refer to Section 3.4.1.1, except for the following parameters:
  - Request parameter "source": Value should be "App_Scheme" (for the Online Service App URL scheme), or "App_Link" (for the Universal Link/App Link).
  - Request parameter "redirectURI": This is Online Service App URL scheme or Universal Link/App Link depending on the "source" parameter, which is used for receiving the authorisation code. The value should be URL encoded. For details, please refer to "iAM Smart" API Specification. The value must be the same as provided during Online Service registration.
- Set <Context> as "auth" in URL Scheme.

3.4.4.2 Implementing (2) Callback with authCode to Online Service App



Pre-conditions

- Please refer to Section 3.4.2.2.

Post-conditions

- Please refer to Section 3.4.2.2.

Error conditions

- This Online Service callback API is a URL Scheme, or Universal Link/App Link. If parameter "error_code" is returned, it means the authentication request is failed.

Error Code	Error Description	Suggested Action
error_code - D40000	User cancelled authentication request	Inform user the authentication request is cancelled
error_code - D40001	User rejected authentication request	Inform user the authentication request is rejected
error_code - D40002	Failed to authenticate	Inform user the authentication request is failed and retry later
error_code - D40003	Authentication request timeout	Inform user the authentication request is timeout, and provide way for user to do re-authentication

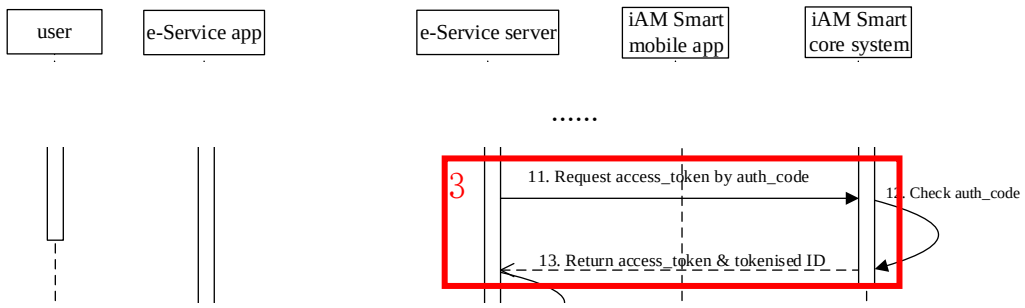
Callback Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Please refer to Section 3.4.2.2.

3.4.4.3 Implementing (3) POST: Request accessToken & Tokenised ID



Please refer to Section 3.4.1.3.

3.4.5 Workflows for verifying CCIC user

3.4.5.1 Scenario : Verify whether the “iAM Smart” user is a CCIC holder

The sequence diagram below shows how Online Services invoke “verfiyIsCcic” API to verify whether the “iAM Smart” user is a CCIC holder after performing authentication process.

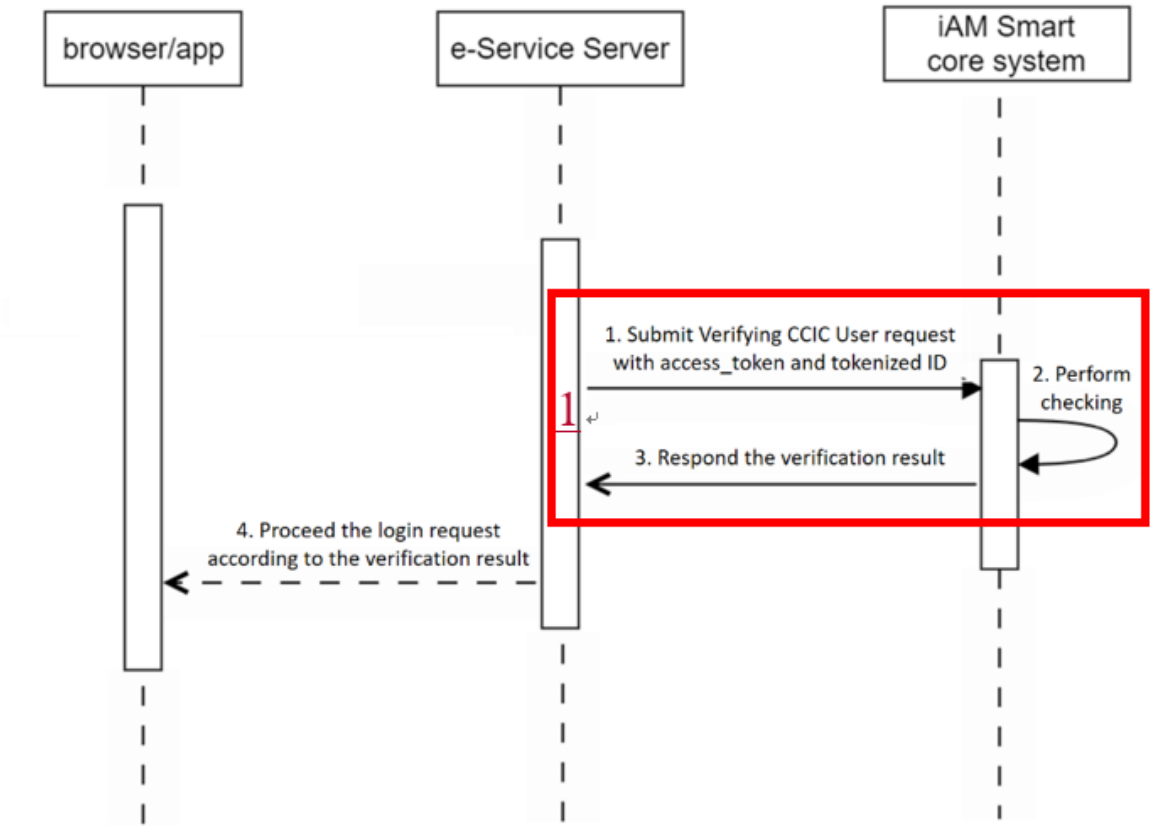


Figure-6 Verify whether the "iAM Smart" user is a CCIC holder

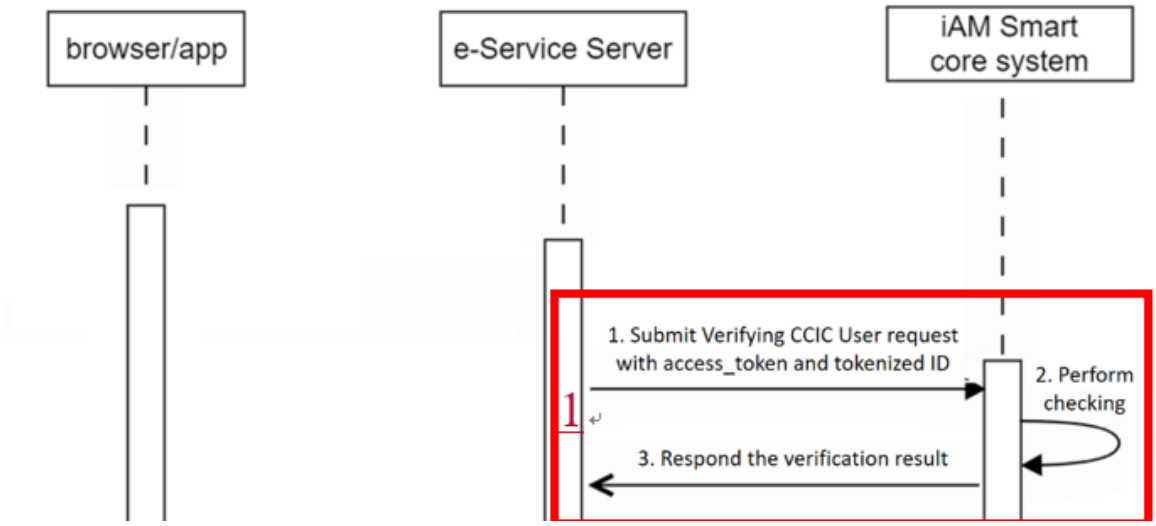
Step 1. Online Service Server initiates a request to invoke the "iAM Smart" *verifyIsCcic* API by sending user's accessToken and openID to "iAM Smart" System. API data encryption is required;

"iAM Smart" API (POST: verifyIsCcic)

Step 2-3. "iAM Smart" core system receives the request and verifies the validity of the accessToken and openID, and then respond the result to Online Service Server. API data encryption is required;

Step 4. The Online Service Server will proceed subsequent login process or display error message according to the verification result.

3.4.5.1.1 Implementing (1) POST: VerifyIsCcic



Pre-conditions

- Online Service must possess a valid accessToken and OpenID of the "iAM Smart" user.
- Online Service must apply the authorisation scope "Authentication" and API Access Control "verfiyIsCcic" in advance.
- API data encryption is required.

Post-conditions

- Online Service should check the response parameters "verifyResult" and determine the next action.
- API data decryption is required.

Error conditions

- Please refer to Section 3.2.3 for common errors.

Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.

3.5 WORKFLOWS FOR FORM FILLING WITH SERVICE LOGIN (AKA PROFILES)

3.5.1 Workflows for obtain User Profiles information after authentication

The sequence diagram below shows how Online Services obtain User Profiles information right after authentication process.

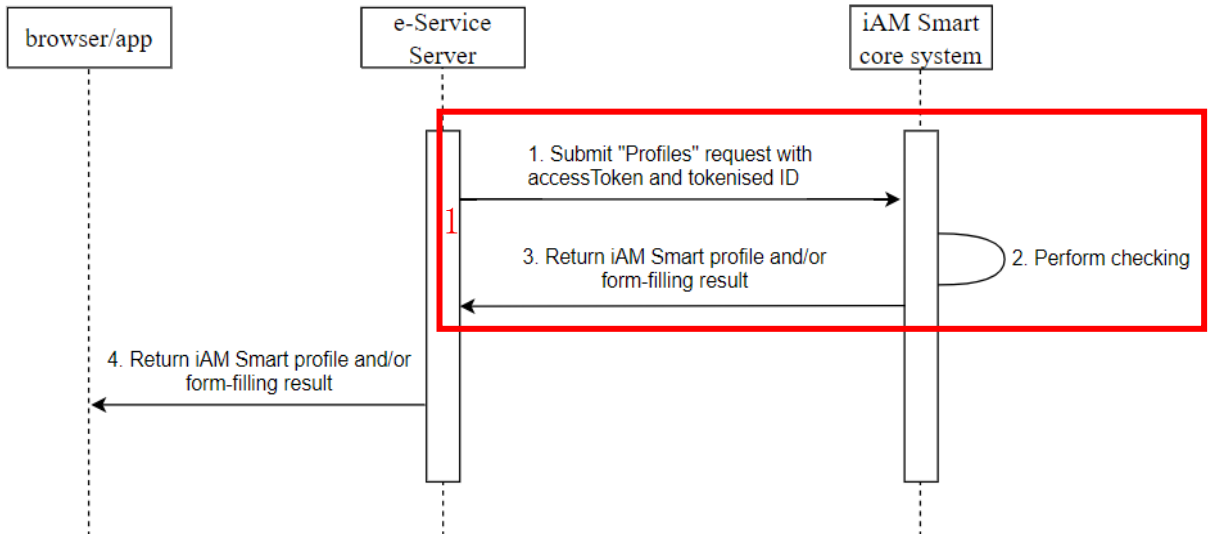


Figure-7 Obtain Profiles Information

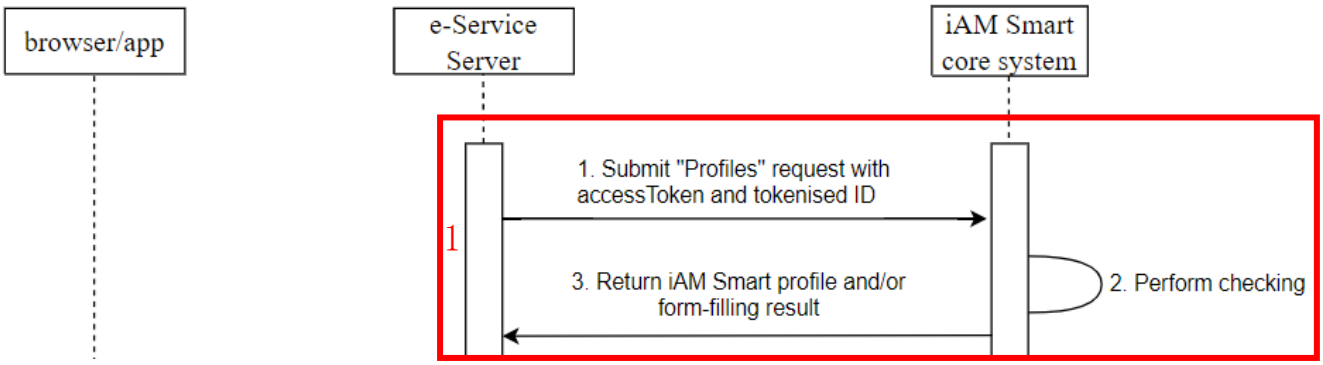
Step 1. Online Service Server initiates a request to invoke the "iAM Smart" *Profiles* API by sending user's accessToken and openID to "iAM Smart" System. API data encryption is required;

"iAM Smart" API (POST: Profiles)

Step 2-3. "iAM Smart" core system receives the request and verifies the validity of the accessToken and openID, and then respond the profile and/or form-filling result to Online Service Server. API data encryption is required;

Step 4. The Online Service Server will proceed subsequent login process or display the form-filling result accordingly.

3.5.1.1 Implementing (1) POST: Profiles



Pre-conditions

- Online Service must possess a valid accessToken and OpenID of the "iAM Smart" user.
- Online Service must apply the authorisation scope "Profiles" and API Access Control "Profiles" in advance.
- API data encryption is required.

Post-conditions

- Online Service should receive the response from the Profiles API and determine the next action.
- API data decryption is required.

Error conditions

- For common errors, please refer to Section 3.2.3. Common error of this API includes:

Code	Description
D20003	Invalid parameter – {accessToken | openID | eMEFields | profileFields}. For this error, please check the ESP for the approved datafields in eMEFields / profileFields in Profiles API. The error occurred may also due to the incorrect accessToken or openID provided.

Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.

3.6 WORKFLOWS FOR FORM FILLING WITHOUT SERVICE LOGIN (AKA ANONYMOUS FORM FILLING)

3.6.1 Scenario 1: Anonymous Form Filling (Online Service Website in Different Device)

The sequence diagram below shows how an anonymous user authorises an Online Service to use his/her "eMEFields" for form filling when Online Service website and the "iAM Smart" Mobile App are running in different devices.

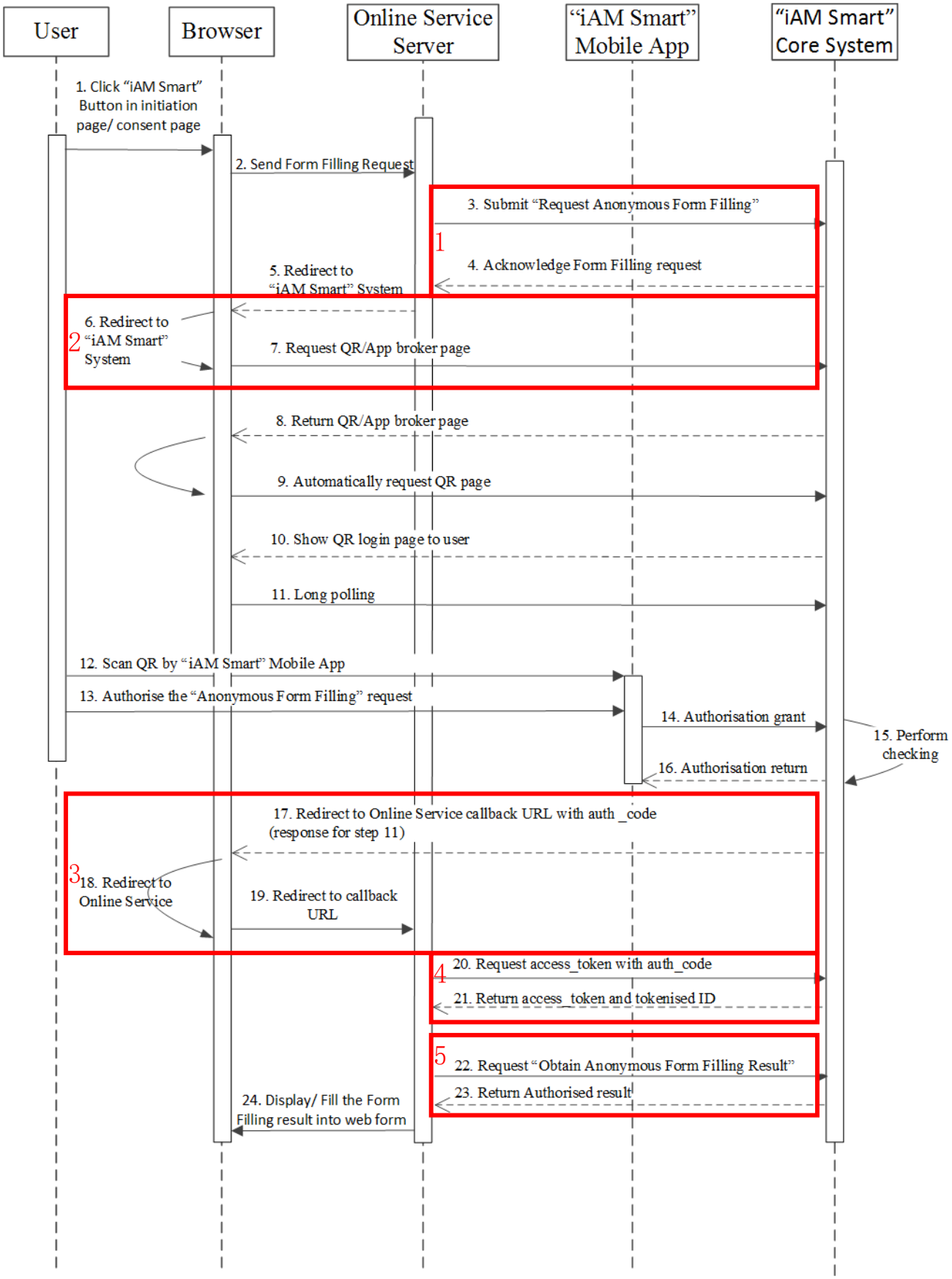


Figure-8 Anonymous Form Filling (Online Service Website in Different Device)

Step 1. User reads the consent page (if “iAM Smart” profile fields requested) and clicks the button in Online Service Website;

Step 2. Online Service Website initiates anonymous form filling request to Online Service Server with browser's user agent name (use as value for request parameter “source”);

Step 3. Online Service Server initiates an anonymous form filling request to invoke the “iAM Smart” API with the “businessID” of this request, required “eMEFields”, etc. API data encryption is required;

“iAM Smart” API (POST: Request Anonymous Form Filling)

Step 4. “iAM Smart” System verifies and confirms receipt of the anonymous Form Filling request to Online Service Server by returning POST response with parameter “ticketID”;

Step 5-7. Online Service Server prepares the request parameters such as “clientID”, “redirectURI”, “source” (set as browser’s user agent value), “scope”, “ticketID”, etc. and constructs the GET request to invoke the “iAM Smart” API using browser redirection;

“iAM Smart” API (GET: Request QR page)

Step 8-10. “iAM Smart” System returns the broker page (if Online Service has set “brokerPage” to “true”) and after the broker page fails to find the “iAM Smart” Mobile App, it will request QR page to be displayed in browser. If Online Service does not request for broker page (i.e., no broker page will be returned), the browser will directly display QR Code page;

Step 11. QR Code page of “iAM Smart” System polls “iAM Smart” System for the QR Code status;

Step 12-14. “iAM Smart” user uses “iAM Smart” Mobile App to scan the QR Code, reviews the form filling authorisation request (e.g., continue or reject the request) and authorises those selected “eMEFields” to Online Services (e.g., authorise only his HKIC number and English name in account information and residential address in “e-ME profile” (unchecked other “eMEFields”));

Step 15-16. "iAM Smart" System verifies the validity of QR code and other necessary information, and return the authorisation result to "iAM Smart" Mobile App;

Step 17-19. QR Code page of "iAM Smart" System receives the polling result returned by "iAM Smart" System (i.e., response for the polling in Step 11) which includes authCode and businessID or any error code (e.g., "iAM Smart" user rejects the request), assembles and invokes the Online Service callback API given in "redirectURI" using browser redirection;

Online Service Callback API (GET: Callback with authCode to Online Service Server)

Step 20-21. When Online Service Server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e., openID) of the "iAM Smart" user. API data encryption is required;

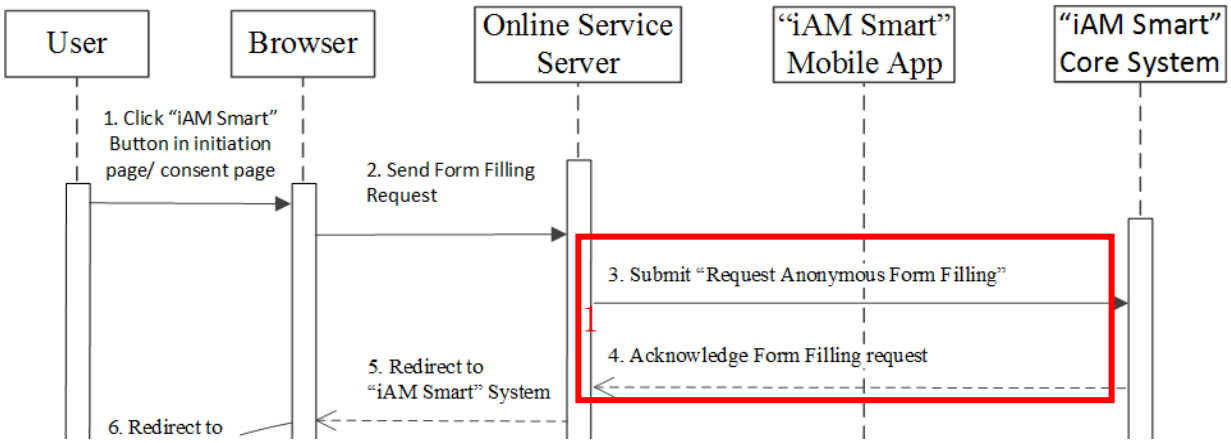
"iAM Smart" API (POST: Request accessToken & Tokenised ID)

Step 22-23. Online Service Server invokes the "iAM Smart" API with the accessToken and openID to obtain the result of anonymous form filling request. API data encryption is required;

"iAM Smart" API (POST: Obtain Anonymous Form Filling Result)

Step 24. Online Service Server processes and shows the result in the corresponding Online Service Website.

3.6.1.1 Implementing (1) POST: Request Anonymous Form Filling



Pre-conditions

- Online Service Website and "iAM Smart" Mobile App are in different devices in this scenario.
- Online Service generates a "businessID" for this request and uses this identifier to match the authCode returned from "iAM Smart" System.
- API data encryption is required.

Post-conditions

- API data decryption is required.

Error conditions

- Please refer to Section 3.5.1.1.

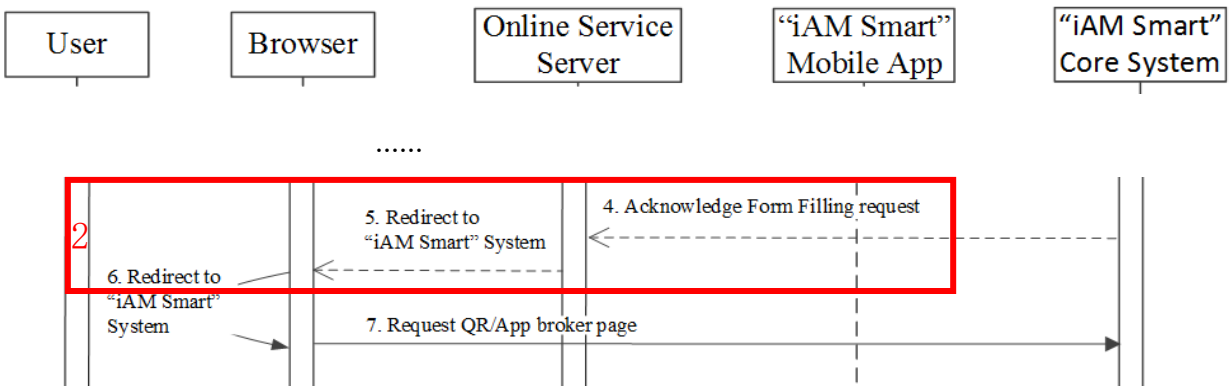
Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- Please refer to Section 3.5.1.1, except for the following parameters:
  - No request parameter "accessToken", "openID", "source", "redirectURI" and "state".
  - No response parameter "authByQR".
  - Response parameter "ticketID": It is returned from "iAM Smart" System for "iAM Smart" APIs for anonymous request. It will be used for invoking subsequent "iAM Smart" APIs "Request QR Page" and "Open "iAM Smart" Mobile App for Getting Context" depending on the scenario. It is a 36-byte (or less) UUID number (ASCII character set).

3.6.1.2 Implementing (2) GET: Request QR Page



Pre-conditions

- Please refer to Section 3.4.1.1 except:
  - Online Service has the "ticketID" provided by "iAM Smart" System for the anonymous request in step 4.
  - Authorisation scope submitted should be "eidapi_formFilling".

Post-conditions

- Please refer to Section 3.4.1.1.

Error conditions

- Please refer to Section 3.4.1.1.

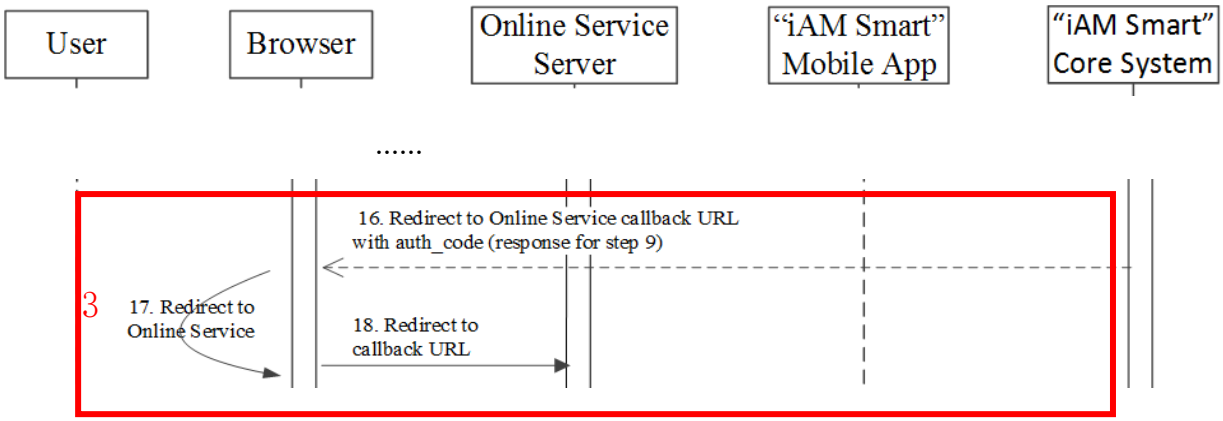
Request Parameters

- Please refer to Section 3.4.1.1.

Notes

- Please refer to Section 3.4.1.1 except the following parameter:
  - Request parameter "ticketID": Value provided by "iAM Smart" System for the anonymous request.
  - Request parameter "scope": It should be "eidapi_formFilling".

3.6.1.3 Implementing (3) GET: Callback with authCode to Online Service Server



Pre-conditions

- Please refer to Section 3.4.1.2.

Post-conditions

- Please refer to Section 3.4.1.2 except:

- Online Service Server should match the callback result with corresponding Online Service client terminal using the “businessID”.

Error conditions

- This Online Service callback API is a HTTP GET request. If parameter “error_code” is returned, it means the request is failed.

Error Code	Error Description	Suggested Action
error_code - D60000	User cancelled form filling request	Inform user the Form Filling request is cancelled
error_code - D60001	User rejected form filling request	Inform user the Form Filling request is rejected
error_code - D60002	Failed to request form filling	Inform user the Form Filling request is failed and retry later

The error_code D60003 (Form Filling request timeout) does not appear in this scenario. For the QR code timeout or request confirmation timeout in the “iAM Smart” Mobile App, message will be prompted in the corresponding user interface.

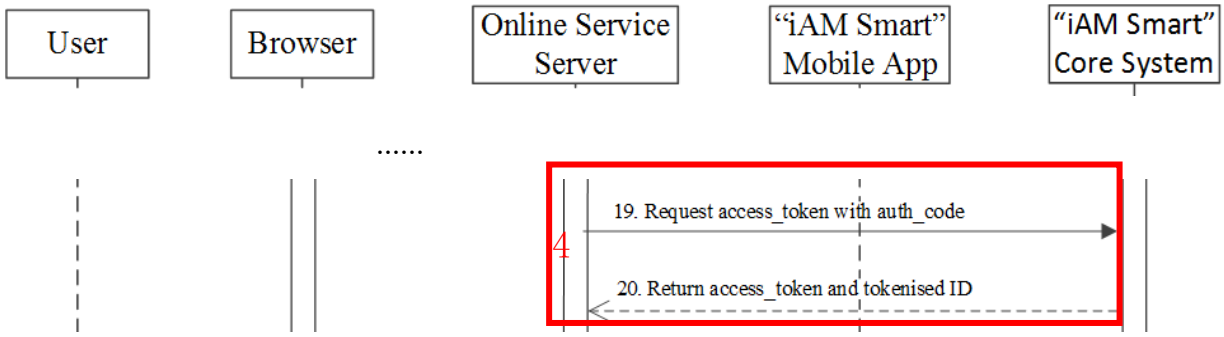
Callback Parameters

- Please refer to Section 3.4.1.2.

Notes

- Please refer to Section 3.4.1.2 except the following parameter:
  - Callback parameter “businessID”: Value returned by “iAM Smart” System should be the same one as Online Service submitted in the anonymous request.

3.6.1.4 Implementing (4) POST: Request accessToken & Tokenised ID



Pre-conditions

- Please refer to Section 3.4.1.3.

Post-conditions

- Please refer to Section 3.4.1.3. except:
  - The accessToken can only be used once before expiry.

Error conditions

- Please refer to Section 3.4.1.3.

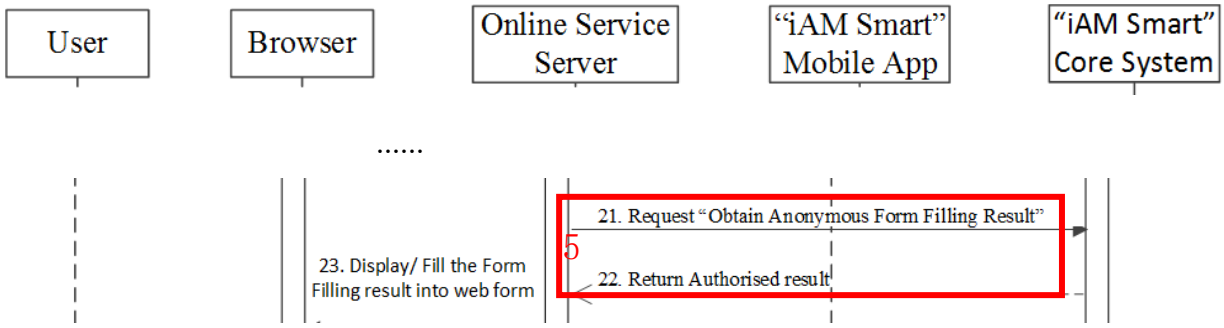
Request and Response Parameters

- Please refer to Section 3.4.1.3.

Notes

- Please refer to Section 3.4.1.3.

3.6.1.5 Implementing (5) POST: Obtain Anonymous Form Filling Result



Pre-conditions

- Online Service must possess a valid accessToken of the "iAM Smart" user with authorisation scope "form filling authorisation".

Post-conditions

- API data decryption is required.
- Online Service Server should check the requested data fields has authorised for the Anonymous Form Filling.
- Online Service should discard the accessToken as it can only be used once.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D60002	Failed to request Form Filling	Inform user the Form Filling request is failed and retry later
code - D60003	Form Filling request timeout	Inform user the Form Filling request is timeout, and provide way for user to retry

Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- "e-ME" items not authorised by "iAM Smart" user will not return in result (i.e., JSON name/value of that item will not exist in result).

3.6.2 Scenario 2: Anonymous Form Filling (Online Service Website in Same Device)

The sequence diagram below shows how an anonymous user authorises an Online Service to use his/her "eMEFields" for form filling when Online

Service website and the “iAM Smart” Mobile App are running in the same device.

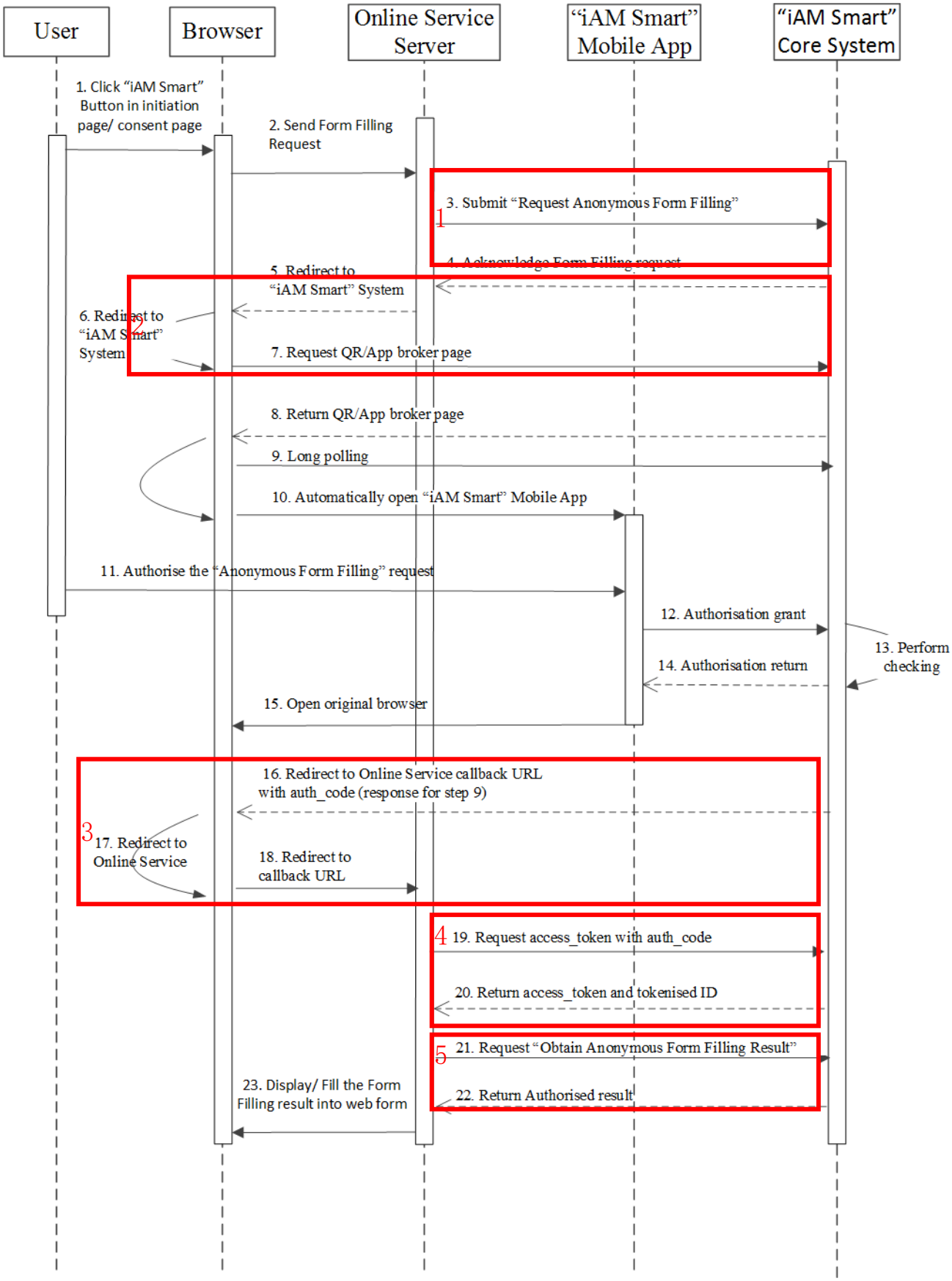


Figure-9 Anonymous Form Filling (Online Service Website in Same Device)

Step 1. User reads the consent page (if "iAM Smart" profile fields requested) and clicks the button in Online Service Website;

Step 2. Online Service Website initiates anonymous form filling request to Online Service Server with browser's user agent name (use as value for request parameter "source");

Step 3. Online Service Server initiates an anonymous Form Filling request to invoke the "iAM Smart" API with the "businessID" of this request, required data fields, etc. API data encryption is required;

"iAM Smart" API (POST: Request Anonymous Form Filling)

Step 4. "iAM Smart" System verifies and confirms receipt of the anonymous Form Filling request to Online Service Server by returning POST response with parameter "ticketID";

Step 5-7. Online Service Server prepares the request parameters such as "clientID", "redirectURI", "source" (set as browser's user agent value), "scope", "brokerPage" (set as "true"), "ticketID", etc. and constructs the GET request to invoke the "iAM Smart" API using browser redirection;

"iAM Smart" API (GET: Request QR page)

Step 8. "iAM Smart" System returns the broker page to the Online Service Website;

Step 9. Broker page of "iAM Smart" System polls "iAM Smart" System for further action;

Step 10-12. Broker page invokes the "iAM Smart" Mobile App in the same device automatically and sends it the relevant parameters. "iAM Smart" user logs in "iAM Smart" Mobile App, reviews the Form Filling authorisation request (e.g., continue or reject the request) and authorises those selected "eMEFields" to Online Service (e.g., authorise only his HKIC number and English name in "iAM Smart" account information and residential address in "e-ME profile" (unchecked other "eMEFields"));

Step 13-15. "iAM Smart" System verifies the validity of necessary information, and return the authorisation result to "iAM Smart" Mobile App.

"iAM Smart" Mobile App invokes the original Online Service browser (i.e., using the browser's user agent value submitted);

Step 16-18. Broker page of "iAM Smart" System receives the polling result returned by "iAM Smart" System (i.e., response for the polling in Step 9) which includes authCode and businessID or any error code (e.g., "iAM Smart" user rejects the request), assembles and invokes the Online Service callback API given in "redirectURI" using browser redirection;

Online Service Callback API (GET: Callback with authCode to Online Service Server)

Step 19-20. When Online Service Server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e., openID) of the "iAM Smart" user. API data encryption is required;

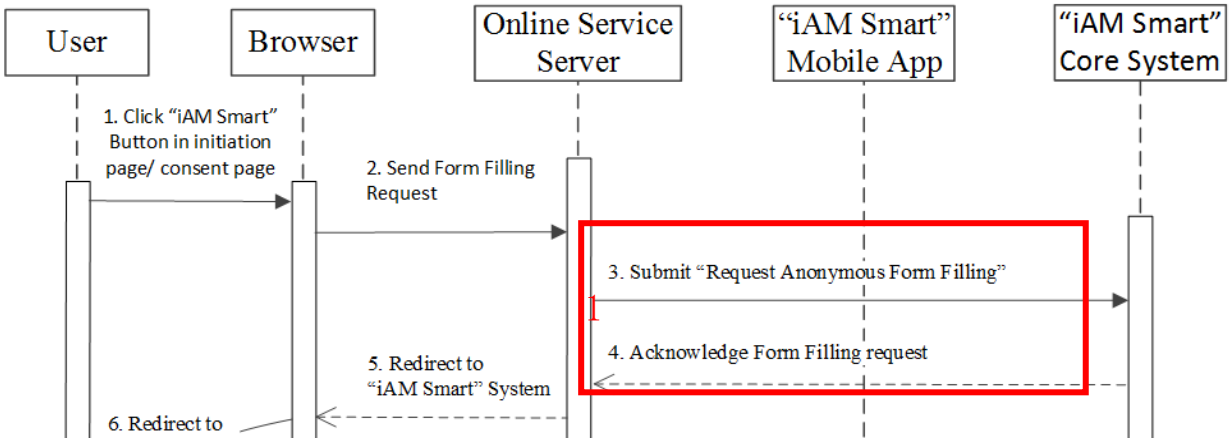
"iAM Smart" API (POST: Request accessToken & Tokenised ID)

Step 21-22. Online Service Server invokes the "iAM Smart" API with the accessToken and openID to obtain the result of anonymous form filling request. API data encryption is required;

"iAM Smart" API (POST: Obtain Anonymous Form Filling Result)

Step 23. Online Service Server processes and shows the result in the corresponding Online Service Website.

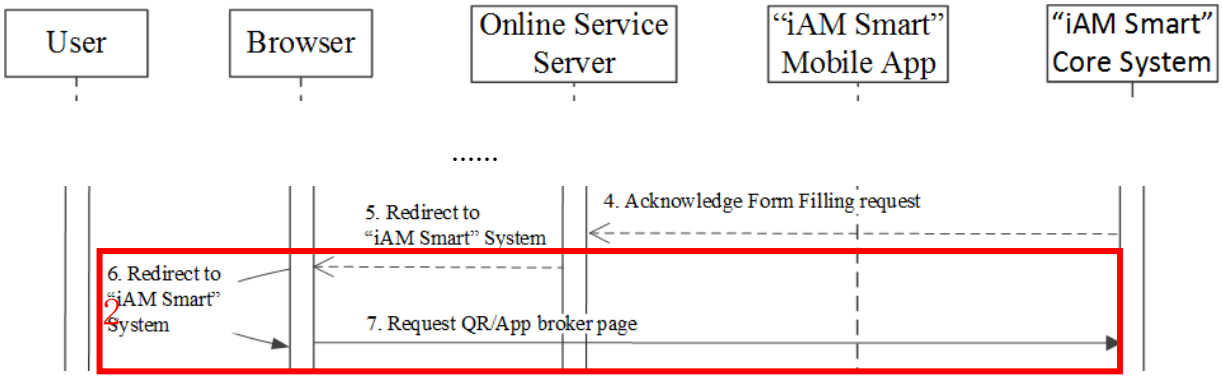
3.6.2.1 Implementing (1) POST: Request Anonymous Form Filling



Please refer to Section 3.6.1.1 except:

- Online Service Website and "iAM Smart" Mobile App are in the same device in this scenario.

3.6.2.2 Implementing (2) GET: Request QR Page



Pre-conditions

- Please refer to Section 3.4.2.1 except:
  - Online Service has the "ticketID" provided by "iAM Smart" System for the anonymous request in step 4.
  - Authorisation scope submitted should be "eidapi_formFilling".

Post-conditions

- Please refer to Section 3.4.2.1.

Error conditions

- Please refer to Section 3.4.2.1.

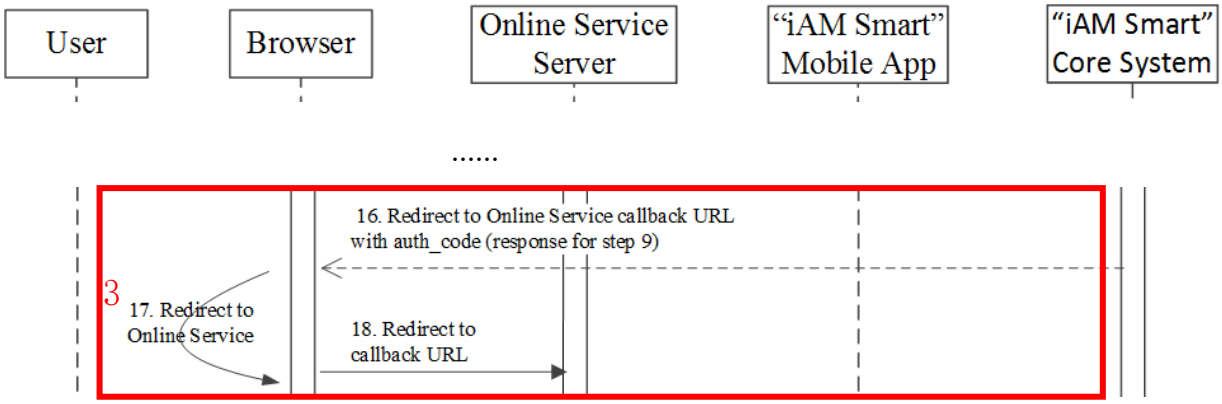
Request Parameters

- Please refer to Section 3.4.2.1.

Notes

- Please refer to Section 3.4.2.1 except the following parameter:
  - Request parameter “ticketID”: provided by “iAM Smart” System for the anonymous request.
  - Request parameter “scope”: It should be “eidapi_formFilling”.

3.6.2.3 Implementing (3) GET: Callback with authCode to Online Service Server



Pre-conditions

- Please refer to Section 3.4.2.2.

Post-conditions

- Please refer to Section 3.4.2.2 except:
  - Online Service Server should match the callback result with corresponding Online Service Website using the “businessID”.

Error conditions

- Please refer to Section 3.4.2.2

Callback Parameters

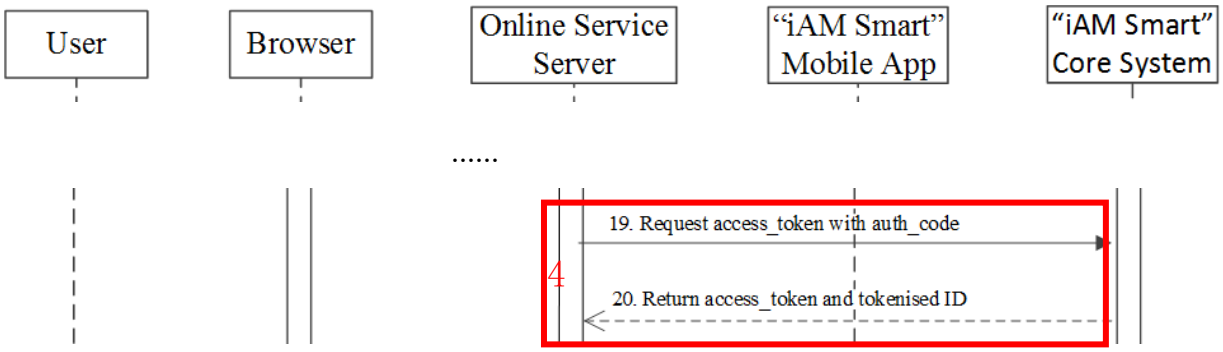
- Please refer to Section 3.4.2.2.

Notes

- Please refer to Section 3.4.2.2 except the following parameter:

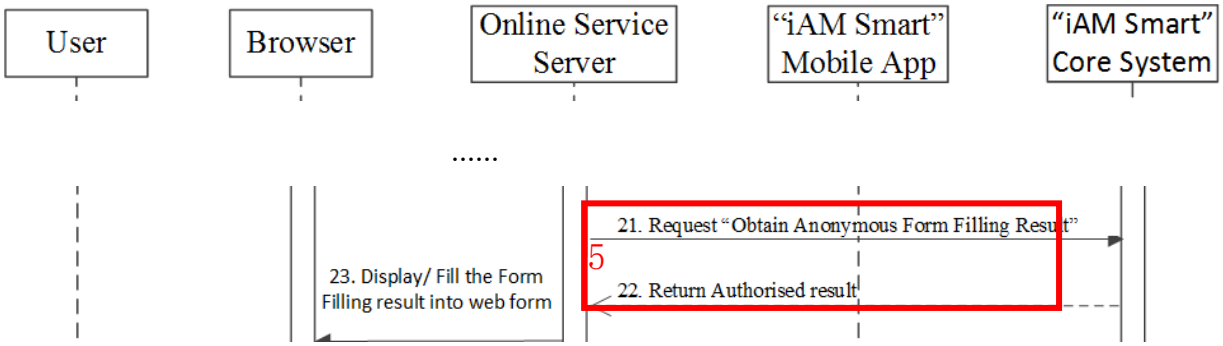
- Callback parameter "businessID": Value returned by "iAM Smart" System should be the same one as Online Service submitted in the anonymous request.

3.6.2.4 Implementing (4) POST: Request accessToken & Tokenised ID



Please refer to Section 3.8.1.5.

3.6.2.5 Implementing (5) POST: Obtain Anonymous Form Filling Result



Please refer to Section 3.8.1.6.

3.6.3 Scenario 3: Anonymous Form Filling (Online Service App in Different Device)

The sequence diagram below shows how an anonymous user authorises an Online Service to use his/her "eMEFields" for form filling when Online

Service App and the "iAM Smart" Mobile App are running in different devices.

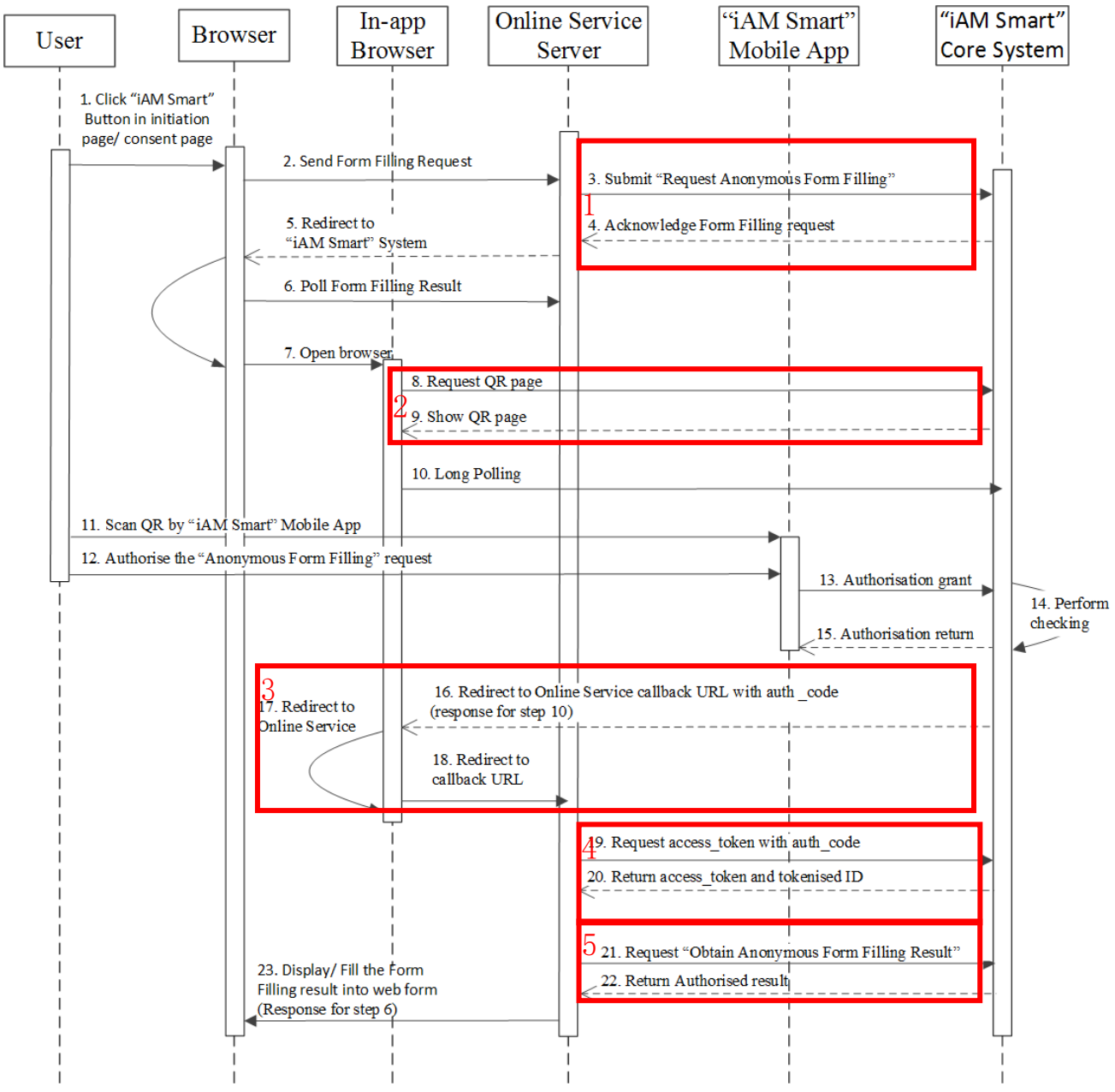


Figure-10 Anonymous Form Filling (Online Service App in Different Device)

Step 1. User reads the consent page (if "iAM Smart" profile fields requested) and clicks the button in Online Service App;

Step 2. Online Service App determines there is no "iAM Smart" Mobile App in the device using program code, initiates anonymous form filling request to Online Service Server with the in-app browser's user agent value (use as value for request parameter "source");

Step 3. Online Service Server initiates an anonymous Form Filling request to invoke the "iAM Smart" API with the "businessID" of this request, required data fields, etc. API data encryption is required;

"iAM Smart" API (POST: Request Anonymous Form Filling)

Step 4. "iAM Smart" System verifies and confirms receipt of the anonymous Form Filling request to Online Service Server by returning POST response with parameter "ticketID";

Step 5. After receiving the response, Online Service Server prepares necessary parameters such as "clientID", "redirectURI", "scope", "source" (set as in-app browser's user agent value), "brokerPage" (set as "false"), "ticketID", etc. and constructs the GET request to invoke the "iAM Smart" API by Online Service App;

Step 6. Online Service App polls Online Service Server for the form filling result from "iAM Smart" System;

Step 7. Online Service App invokes in-app browser (Safari for iOS, Chrome for Android) to submit the GET request;

Step 8-9. Browser opens the GET request to invoke the "iAM Smart" API and QR Code page will be shown;

"iAM Smart" API (GET: Request QR page)

Step 10. QR Code page of "iAM Smart" System polls "iAM Smart" System for the QR Code status;

Step 11-13. "iAM Smart" user logs in "iAM Smart" Mobile App to scan the QR Code, reviews the Form Filling authorisation request (e.g., continue or reject the request) and authorises those selected form filling items to Online Services (e.g., authorise only his HKIC number and residential address but no other items);

Step 14-15. "iAM Smart" System verifies the validity of QR code and other necessary information, and return the authorisation result to "iAM Smart" Mobile App;

Step 16-18. QR Code page of "iAM Smart" System receives the polling result returned by "iAM Smart" System (i.e., response for the polling in Step 10) which includes authCode and businessID or any error code (e.g., "iAM Smart" user rejects the request), assembles and

invokes the Online Service callback API given in "redirectURI" using browser redirection;

Online Service Callback API (GET: Callback with authCode to Online Service Server)

Step 19-20. When Online Service Server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e., openID) of the "iAM Smart" user. API data encryption is required;

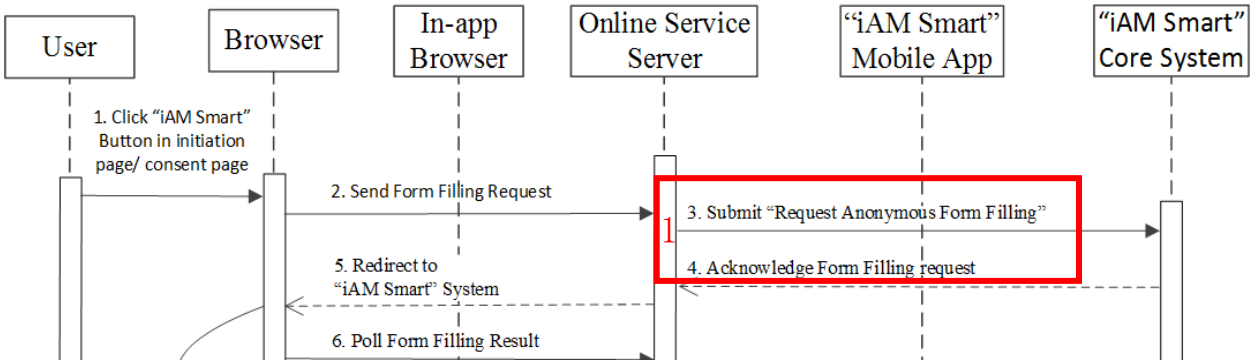
"iAM Smart" API (POST: Request accessToken & Tokenised ID)

Step 21-22. Online Service Server invokes the "iAM Smart" API with the accessToken and openID to obtain the result of anonymous form filling request. API data encryption is required;

"iAM Smart" API (POST: Obtain Anonymous Form Filling Result)

Step 23. Online Service Server processes and shows the result in the corresponding Online Service App (i.e., response for the polling in Step 6).

3.6.3.1 Implementing (1) POST: Request Anonymous Form Filling



Pre-conditions

- Please refer to Section 3.6.1.1 except:
  - Online Service should determine the "iAM Smart" Mobile App is not in the same device of Online Service App using program code.

Post-conditions

- Please refer to Section 3.6.1.1.

Error conditions

- Please refer to Section 3.6.1.1.

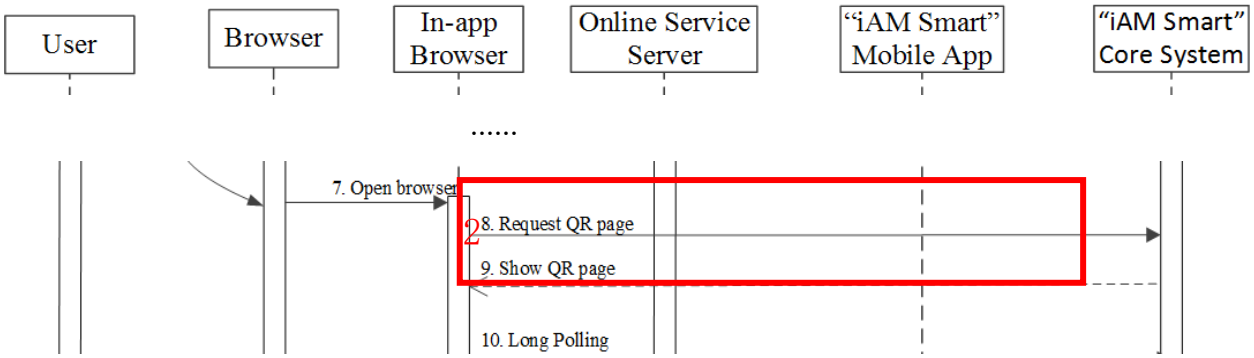
Request and Response Parameters

- Please refer to Section 3.6.1.1.

Notes

- Please refer to Section 3.6.1.1, except for the following parameter:
  - Request parameter “source”: Value should be the in-app browser's user agent value.

3.6.3.2 Implementing (2) GET: Request QR Page



Pre-conditions

- Please refer to Section 3.4.3.1 except:
  - Online Service has the “ticketID” provided by “iAM Smart” System for the anonymous request in step 4.
  - Authorisation scope submitted should be “eidapi_formFilling”.

Post-conditions

- Please refer to Section 3.4.3.1.

Error conditions

- Please refer to Section 3.4.3.1.

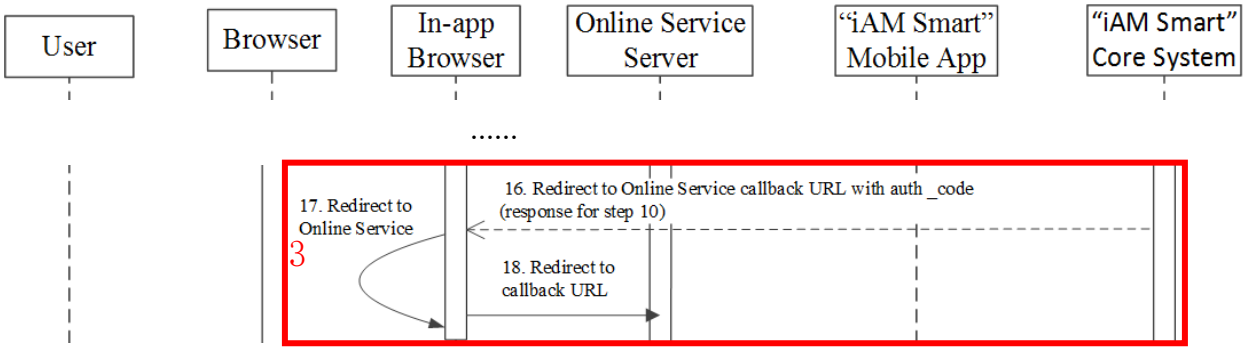
Request Parameters

- Please refer to “iAM Smart” API Specification.

Notes

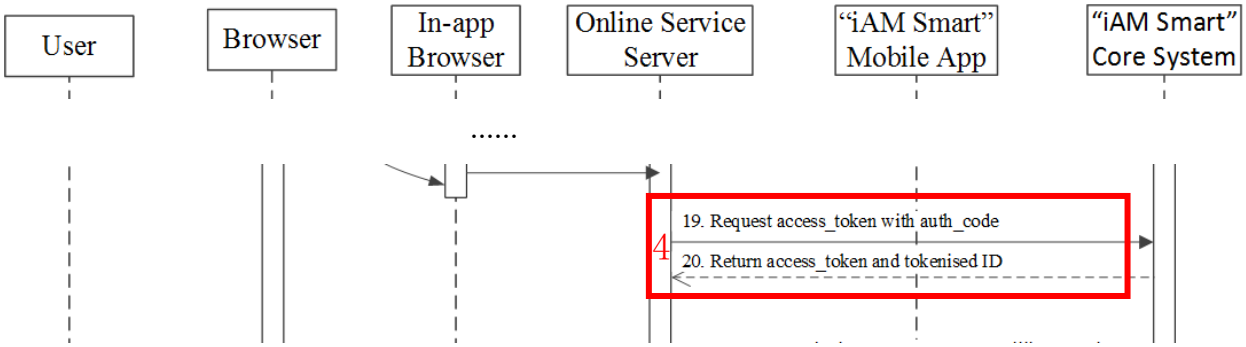
- Please refer to Section 3.4.3.1 except the following parameter:
  - Request parameter “ticketID”: Value provided by “iAM Smart” System for the anonymous request.
  - Request parameter “scope”: It should be “eidapi_formFilling”.

3.6.3.3 Implementing (3) GET: Callback with authCode to Online Service Server



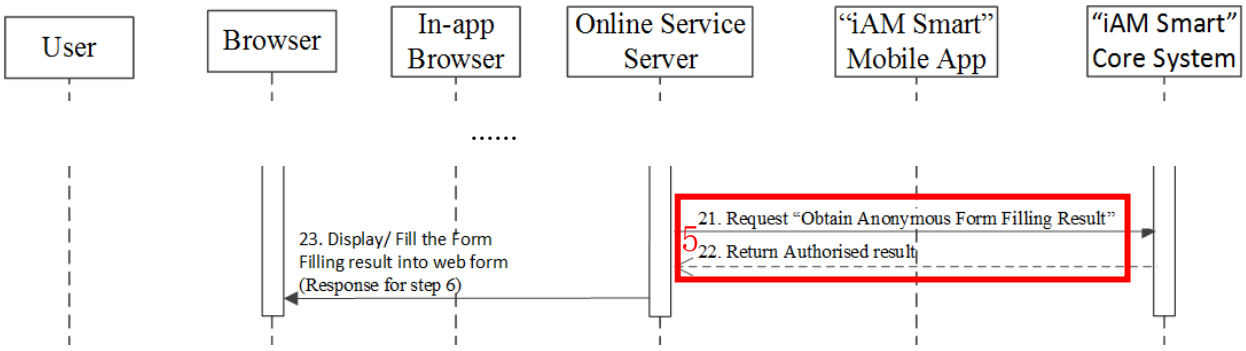
Please refer to Section 3.8.1.4.

3.6.3.4 Implementing (4) POST: Request accessToken & Tokenised ID



Please refer to Section 3.8.1.5.

3.6.3.5 Implementing (5) POST: Obtain Anonymous Form Filling Result



Please refer to Section 3.6.1.5.

3.6.4 Scenario 4: Anonymous Form Filling (Online Service App in Same Device)

The sequence diagram below shows how an anonymous user authorises an Online Service to use his/her "eMEFields" for form filling when Online Service App and the "iAM Smart" Mobile App are running in the same device.

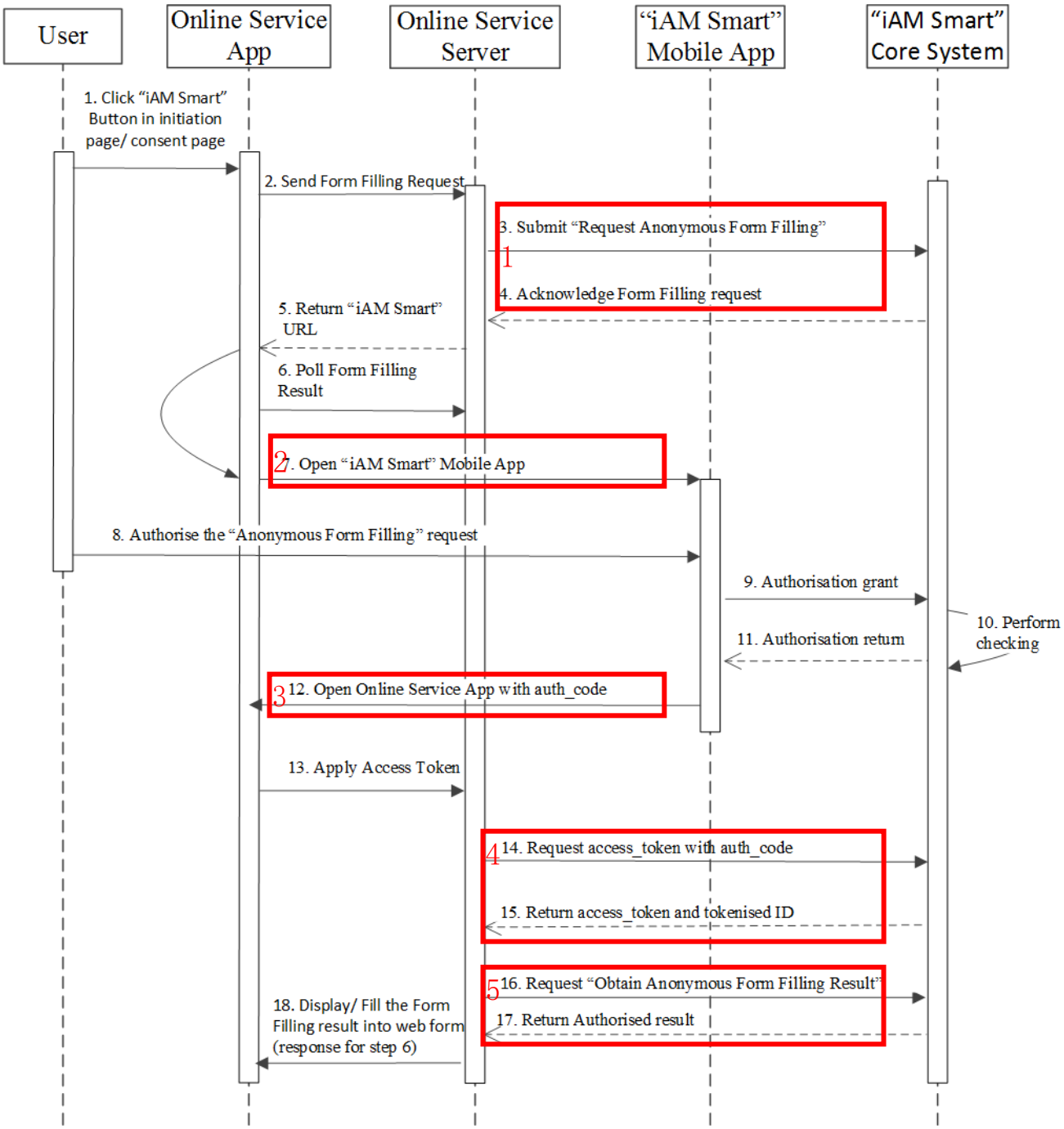


Figure-11 Anonymous Form Filling (Online Service App in Same Device)

Step 1. User reads the consent page (if "iAM Smart" profile fields requested) and clicks the button in Online Service App;

Step 2. Online Service App determines "iAM Smart" Mobile App is installed in the device using program code, initiates anonymous form filling request to Online Service Server with "App_Scheme" or "App_Link" (use as value for request parameter "source");

Step 3. Online Service Server initiates an anonymous Form Filling request to invoke the "iAM Smart" API with the "businessID" of this request, required data fields, etc. API data encryption is required;

"iAM Smart" API (POST: Request Anonymous Form Filling)

Step 4. "iAM Smart" System verifies and confirms receipt of the anonymous Form Filling request to Online Service Server by returning POST response with parameter "ticketID";

Step 5. After receiving the response, Online Service Server prepares necessary parameters such as "clientID", "redirectURI" (set as Online Service App URL Scheme or Universal Link/App Link depending on "source" parameter), "scope", "source" (set as "App_Scheme" or "App_Link"), "ticketID", etc. and constructs the URL Scheme to invoke "iAM Smart" Mobile App by Online Service App;

Step 6. Online Service App polls Online Service Server for the form filling result from "iAM Smart" System;

Step 7. Online Service App invokes "iAM Smart" Mobile App using URL Scheme with request parameters (set <Context> as "anon_form-filling" in URL Scheme);

"iAM Smart" API (URL Scheme: Open "iAM Smart" Mobile App for Getting Context)

Step 8-9. "iAM Smart" user logs in "iAM Smart" Mobile App, reviews the Form Filling authorisation request (e.g., continue or reject the request) and authorises those selected form filling items to Online Service (e.g., authorise only his HKIC number and residential address but no other items);

Step 10-11. "iAM Smart" System verifies the validity of necessary information, and return the authorisation result to "iAM Smart" Mobile App.

Step 12. "iAM Smart" Mobile App invokes Online Service App using URL Scheme or Universal Link/App Link with required parameters such as "authCode", "businessID";

Online Service Callback API (Callback with authCode to Online Service App)

Step 13. Online Service App sends authCode, businessID, etc. to Online Service Server;

Step 14-15. When Online Service Server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e., openID) of the "iAM Smart" user. API data encryption is required;

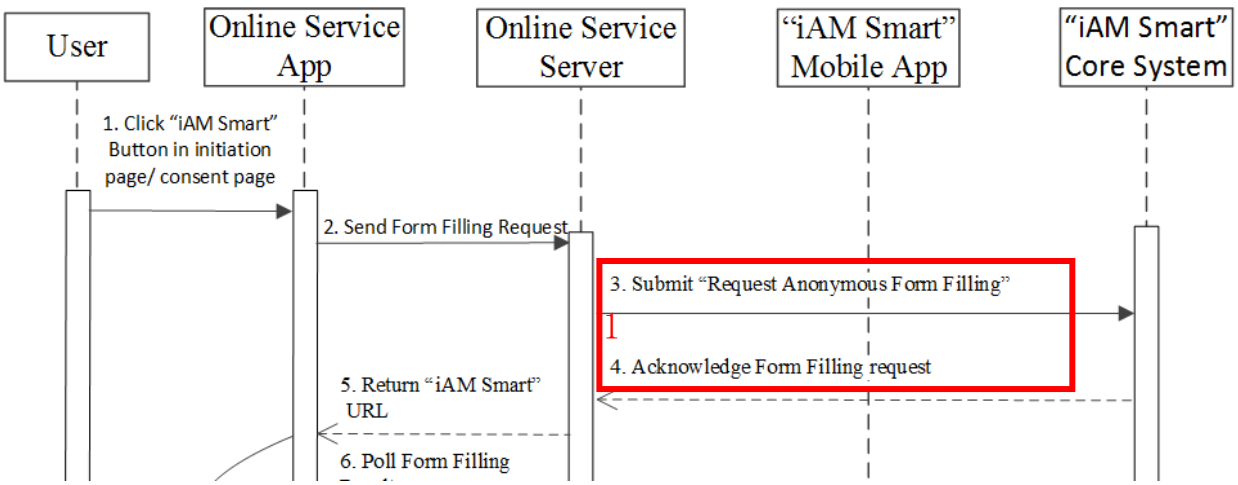
"iAM Smart" API (POST: Request accessToken & Tokenised ID)

Step 16-17. Online Service Server invokes the "iAM Smart" API with the accessToken and openID to obtain the result of anonymous form filling request. API data encryption is required;

"iAM Smart" API (POST: Obtain Anonymous Form Filling Result)

Step 18. Online Service Server processes and shows the result in the corresponding Online Service App (i.e., response for the polling in Step 6).

3.6.4.1 Implementing (1) POST: Request Anonymous Form Filling



Pre-conditions

- Please refer to Section 3.6.1.1 except:
  - Online Service should determine the "iAM Smart" Mobile App is on the same device of Online Service App using program code.

Post-conditions

- Please refer to Section 3.6.1.1.

Error conditions

- Please refer to Section 3.6.1.1.

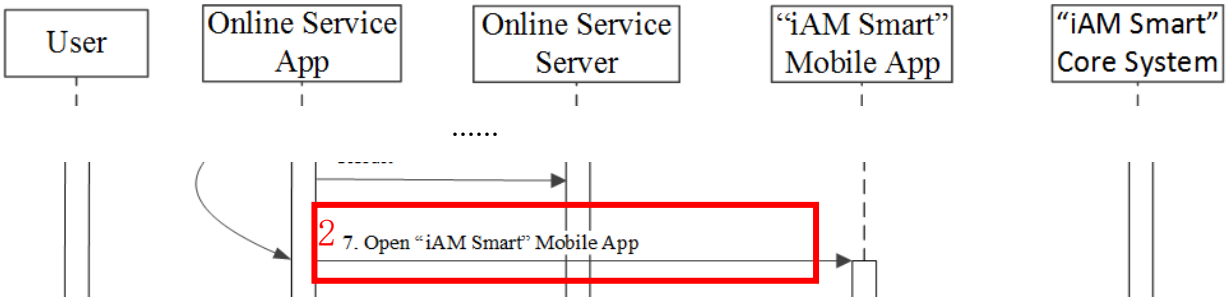
Request and Response Parameters

- Please refer to Section 3.6.1.1.

Notes

- Please refer to Section 3.6.1.1.

3.6.4.2 Implementing (2) URL Scheme: Open "iAM Smart" Mobile App for Getting Context



Pre-conditions

- Online Service App and "iAM Smart" Mobile App are in the same user device.
- Online Service has the "ticketID" provided by "iAM Smart" System for the anonymous request in step 4.
- Online Service should determine if it would generate the optional "state" request parameter to prevent CSRF attack. The "state" will be returned to Online Service for checking through the Online Service callback API for receiving the authorisation code from "iAM Smart" System in Step 12.

Post-conditions

- "iAM Smart" Mobile App will be launched.
- Online Service App should keep synchronising with Online Service server for the result from "iAM Smart" System.

Error conditions

- Nil

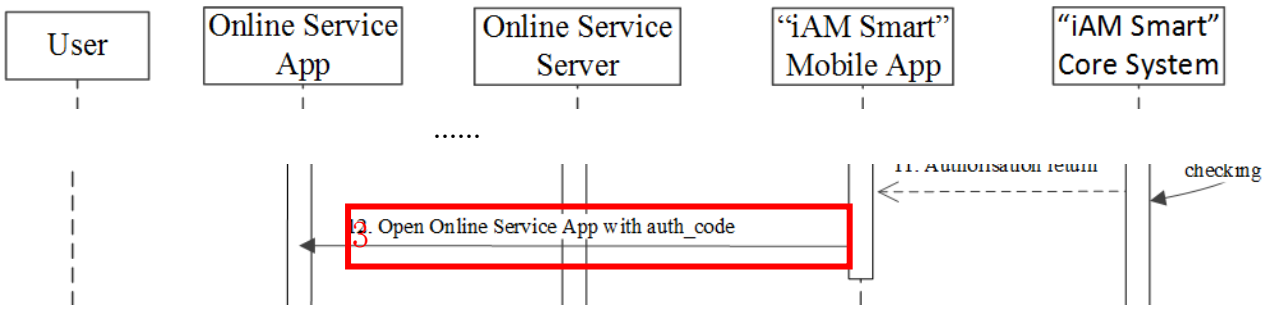
Request Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- Request parameter "ticketID": Value provided by "iAM Smart" System for the anonymous request.
- Request parameter "source": Value should be "App_Scheme" (for Online Service App URL scheme), or "App_Link" (for the Universal Link/App Link).
- Request parameter "redirectURI": This is Online Service App URL scheme or Universal Link/App Link depending on the "source" parameter, which is used for receiving the authorisation code. The value should be URL encoded. For details, please refer to Section 5.5.3 of "iAM Smart" API Specification. The value must be the same as provided during Online Service registration.
- Request parameter "state": The value should be URL encoded.
- Set <Context> as "anon_form-filling" in URL Scheme.

3.6.4.3 Implementing (3) Callback with authCode to Online Service App



Pre-conditions

- Please refer to Section 3.4.2.2.

Post-conditions

- Please refer to Section 3.4.2.2 except:
  - Online Service Server should match the callback result with corresponding Online Service App using the "businessID".

Error conditions

- This Online Service callback API is a URL Scheme, or Universal Link/App Link. If parameter "error_code" is returned, it means the authentication request is failed.

Error Code	Error Description	Suggested Action
error_code - D60000	User cancelled form filling request	Inform user the Form Filling request is cancelled
error_code - D60001	User rejected form filling request	Inform user the Form Filling request is rejected
error_code - D60002	Failed to request form filling	Inform user the Form Filling request is failed and retry later
error_code - D60003	Form Filling request timeout	Inform user the Form Filling request is timeout, and provide way for user to retry

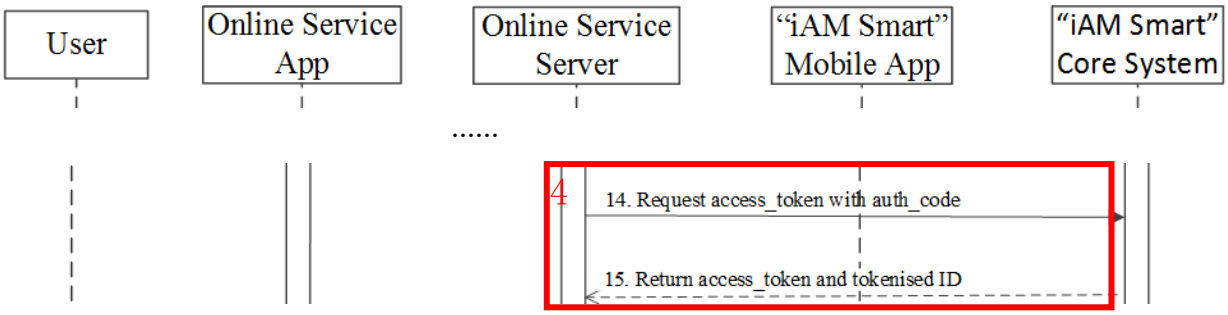
Callback Parameters

- Please refer to "iAM Smart" API Specification.

Notes

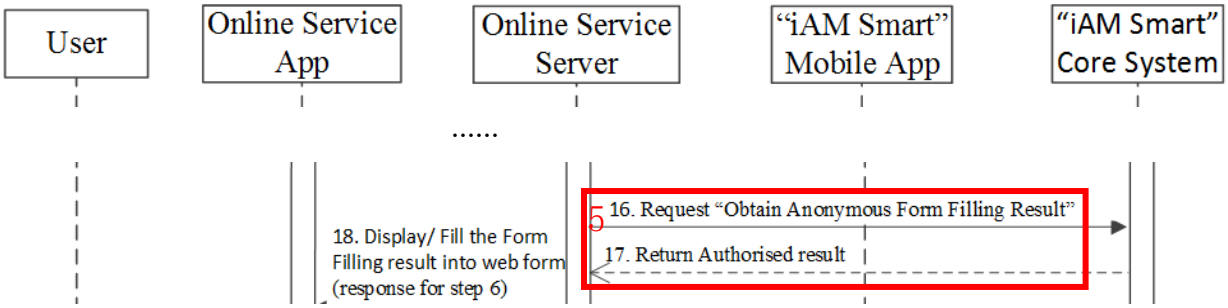
- Please refer to Section 3.4.2.2 except the following parameter:
  - Callback parameter "businessID": Value returned by "iAM Smart" System should be the same one as Online Service submitted in the anonymous request.

3.6.4.4 Implementing (4) POST: Request accessToken & Tokenised ID



Please refer to Section 3.8.1.5.

3.6.4.5 Implementing (5) POST: Obtain Anonymous Form Filling Result



Please refer to Section 3.8.1.6.

3.7 WORKFLOWS FOR DIGITAL SIGNING WITH SERVICE LOGIN

The following process includes non-anonymous hash digital signing and non-anonymous pdf digital signing.

3.7.1 Scenario 1: Digital Signing (Online Service Website/App in Different Device)

The sequence diagram below shows how an authenticated user authorises and signs the Document Hash when Online Service website/App and the "iAM Smart" Mobile App are running in different devices.

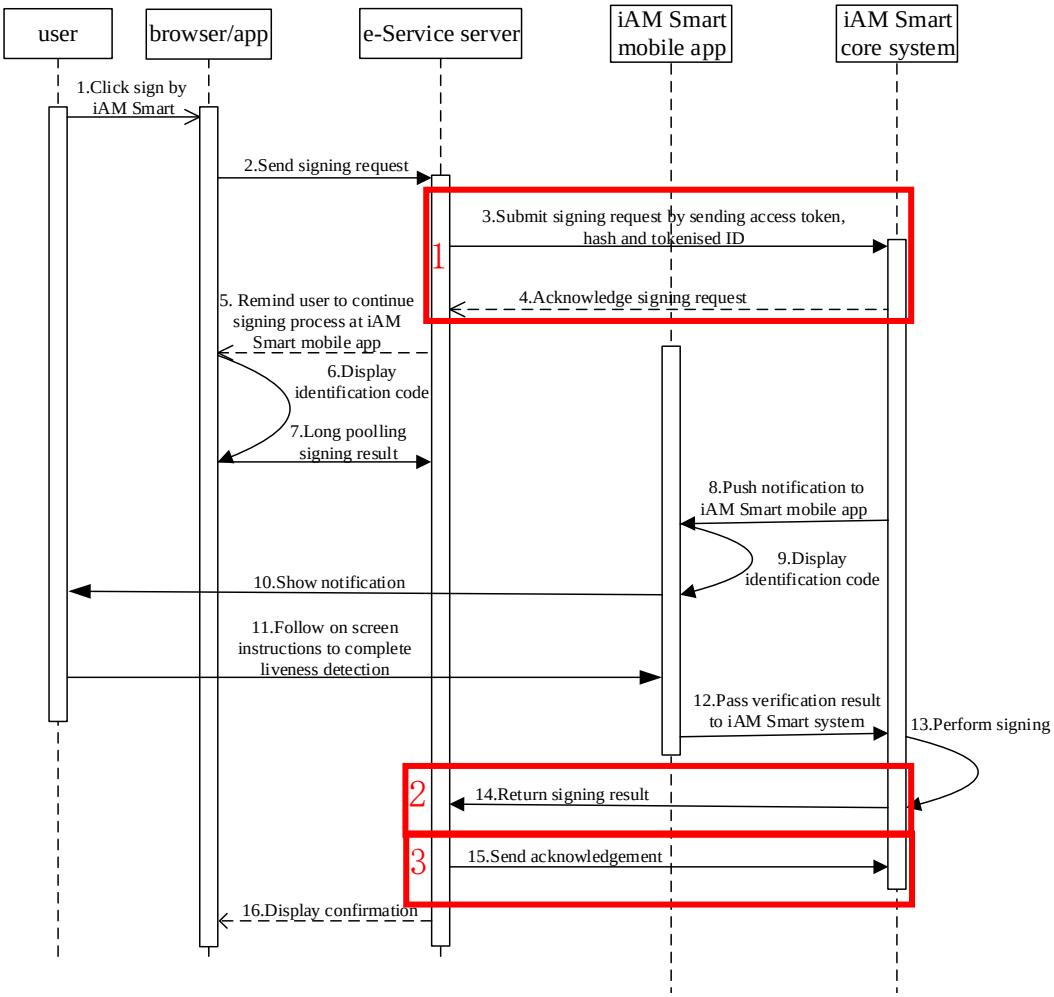


Figure-12 Digital Signing (Online Service Website/App in Different Device)

Step 1. User clicks the "Sign by iAM Smart" button in Online Service Website/App;

Step 2-3. Online Service initiate digital signing request to invoke the "iAM Smart" API with the "businessID" of this request, accessToken,

Tokenised ID, Document Hash / PDF digest, HKICHash, signature algorithm (only for "*Request Digital Signing*"), etc. The request parameter "source" will be the browser's user agent value (for Online Service Website) or "App_Scheme"/"App_Link" (for Online Service App) in this scenario. API encryption is required;

"iAM Smart" API (POST: Request Digital Signing)

"iAM Smart" API (POST: Request PDF Digital Signing)

Step 4. "iAM Smart" System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the digital signing request to Online Service server by returning POST response with parameter "authByQR" (set to "true") in this scenario;

Step 5-6. Online Service Server uses the Document Hash / PDF digest and hash of Tokenised ID to calculate a 4-digit identification code and instructs Online Service Website/App to display the instructions with the 4-digit identification code to inform "iAM Smart" user to process the digital signing authorisation request in "iAM Smart" Mobile App;

Step 7. Online Service Website/App should keep synchronising with Online Service Server for the digital signing processing result (e.g., polling);

Step 8. "iAM Smart" System pushes a notification message to the "iAM Smart" Mobile App based on the Tokenised ID;

Step 9. "iAM Smart" user logs in "iAM Smart" Mobile App, reviews the digital signing authorisation request (e.g., continue or reject the request) and verifies the 4-digit identification code with Online Service Website/App;

Step 10-13. "iAM Smart" user follows the instructions in "iAM Smart" Mobile App to complete digital signing operation;

Step 14. "iAM Smart" System invokes Online Service callback API to return the result with "businessID" of the digital signing request. API data decryption is required;

Online Service Callback API (POST: Callback to Receive Digital Signing Result)

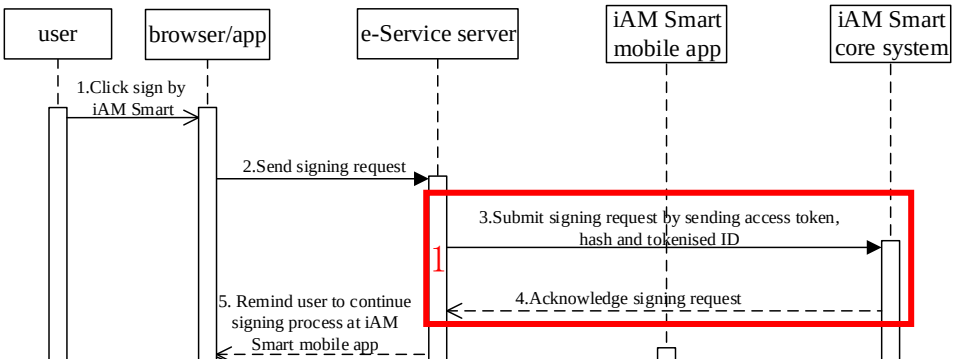
Online Service Callback API (POST: Callback to Receive PDF Digital Signing Result)

Step 11. Online Service Server completes the document digital signing process, verify the result and acknowledges "iAM Smart" System the digital signing result using "iAM Smart" API with the same "businessID" of the digital signing request. API data encryption is required;

"iAM Smart" API (POST: Online Service Acknowledges Digital Signing Result)

Step 12. Online Service Server matches the result using the "businessID" and shows the digital signing result in corresponding Online Service Website/App.

3.7.1.1 Implementing (1) POST: Request Digital Signing



Pre-conditions

- Online Service must possess a valid accessToken of the "iAM Smart" user with authorisation scope "Signing authorisation".
- Online Service Website/App and "iAM Smart" Mobile App are in different devices in this scenario.
- The "iAM Smart" user who signs the document must have a sign version "iAM Smart".
  - Online Service generates a "businessID" for this Digital Signing request and uses this identifier to match the callback result returned from "iAM Smart" System.
- Online Service generates the Document Hash as "hashCode" parameter from the original documents to be signed using a hash algorithm that can suit the

specific business need. It is recommended to use a hash algorithm that is at least SHA-256 or equivalent.

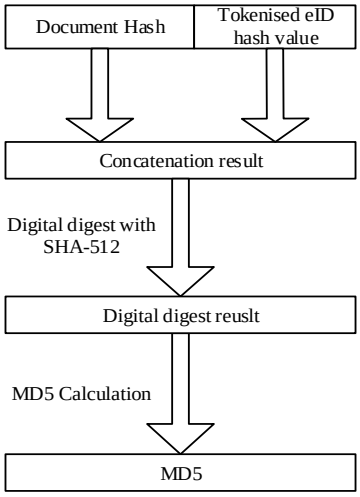
- API data encryption is required.

Post-conditions

- Online Service server should check the response parameter "authByQR" and "ticketID" and determine the next action. For details, please refer to Section 3.2.1. "ticketID" will not be provided by "iAM Smart" System in this scenario.
- Online Service has to calculate a 4-digit identification code using the Document Hash and Tokenised ID hash value. Online Service Website/App should show the 4-digit identification code and instruction to inform "iAM Smart" user to process the digital signing request in "iAM Smart" Mobile App when "authByQR" is "true".

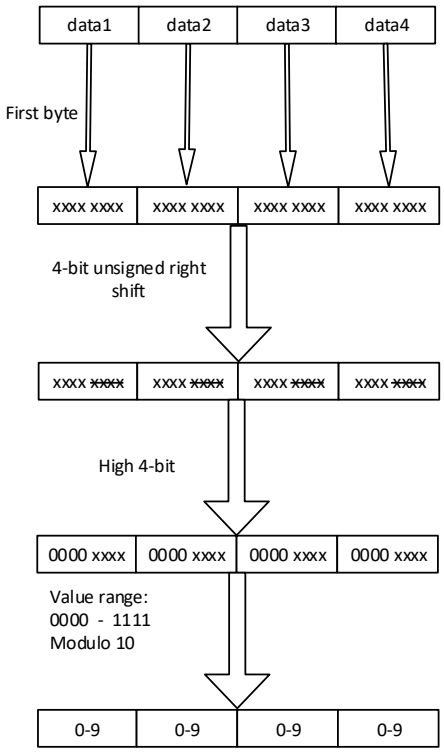
4-digit identification code generation algorithm is as follows:

1) Concatenate the Document Hash and the Tokenised ID hash value (calculated with SHA-512), and perform the digital digest for the concatenation result with SHA-512.
2) Perform the MD5 calculation to obtain a unique MD5 hash value.



3) Divide the 128-bit MD5 hash value into four groups of 4 bytes binary value and then extract the first byte from each group as a data sample.
4) Perform 4-bit unsigned right shift to each byte of the result obtained above for the hexadecimal data.
5) Perform the modulo operation on each byte (modulo 10) to obtain a number of 0-9.

6) Assemble the 4 numbers obtained above as a 4-digit identification code.



Pseudo code for generating 4-digit identification code

```
private static final char hexDigits[] = {'0', '1', '2', '3', '4',
'5', '6', '7', '8', '9'};
public static String getSignCode (String hashCode, String openlD)
{
  // Assume Document Hash is BASE64 encoded and perform decode
  byte[] hashCodeBytes = Base64Util.base64Decode(hashCode);
  // Calculate SHA-512 hashcode of Tokenised ID
  byte[] openlDBytes = DigestUtil.digestToByte(openlD,
Alg.SHA512);
  char code[] = new char [4];
  byte[] source = new byte[hashCodeBytes.length +
openlDBytes.length];
  System.arraycopy(hashCodeBytes, 0, source, 0,
hashCodeBytes.length);
  System.arraycopy(openlDBytes, 0, source, hashCodeBytes.length,
openlDBytes.length);
  // Calculate SHA-512 hashcode from concatenated Document Hash
and
  // SHA-512 hashed Tokenised ID
  byte[] inData = DigestUtil.digestToByte(source, Alg.SHA512);
  // Calculate MD% hashcode
  byte[] digestMD5Data = DigestUtil.digestToByte(inData, Alg.MD5);
  // Convert MD5 value 4-digit identification code
  for (int i = 0; i < 4; i++) {
     code[i] = hexDigits[(digestMD5Data[i * 4] >>> 4 & 0xf) %
10];
  return new String(code);
}
```

- Online Service Website/App should keep synchronising with Online Service Server for the digital signing result returned from "iAM Smart" System.
- API data decryption is required.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D70002	Failed to request digital signing	Inform user the “iAM Smart” digital signing request is failed and retry later
code - D70004	User not allowed to sign	Inform user that “iAM Smart” digital signing is not allowed for default “iAM Smart” and suggest user to upgrade to “iAM Smart” with digital signing function
code - D70005	Inconsistent HKIC number	Hash of HKIC number provided is not matched with “iAM Smart” user. Check hash of HKIC number or inform user to authorise the digital signing using the correct “iAM Smart” account

Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

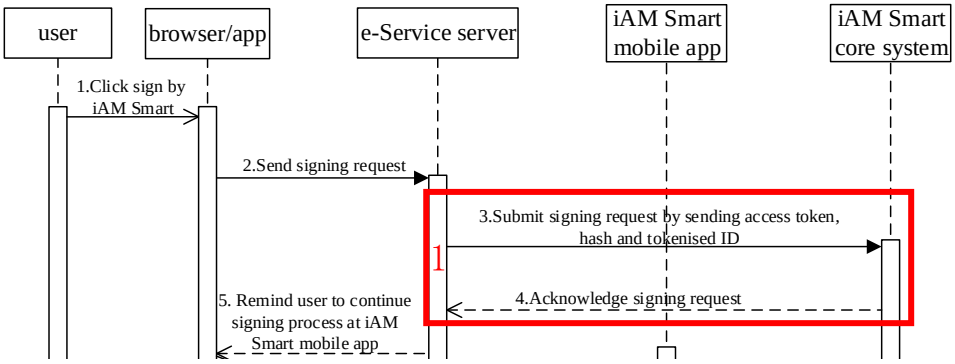
Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- Request parameter “source”: Value should be matched with Online Service client terminal (e.g., “App_Scheme”/“App_Link” or browser’s user agent value).
- Request parameter “openID”: It is the Tokenised ID of the “iAM Smart” user. Online Service should ensure the accessToken and Tokenised ID are in pair and belongs to the target “iAM Smart” user for the request. The target “iAM Smart” user must have a sign version “iAM Smart”.
- Request Parameter - “redirectURI”: Value should equal to URI of “Callback to Receive Digital Signing Result” Online Service callback API. It must be in the same value as provided during Online Service registration.
- Request parameter “hashCode”: It is the Document Hash to be signed. SHA-256 or equivalent is recommended.
- Request parameter “sigAlgo”: It is the signature algorithm to be used by “iAM Smart”. The default value is "SHA256withRSA". While using "NONEwithRSA", hashCode provided must be hashed with SHA-256.
- Request parameter “HKICHash”: “iAM Smart” System will verify the HKIC hash of the “iAM Smart” user with the “HKICHash” provided by Online

Service, and proceed the digital signing only when they matched. Online Service should convert the HKIC number into hash value using SHA-256 before sending to "iAM Smart" System. Only the identifier of HKIC number will be hashed, no check digit is needed.

- Request parameters "department", "serviceName", "documentName": They are the information of the document to be signed and will be shown in "iAM Smart" Mobile App for "iAM Smart" user's reference with the 4-digit identification code.
- Response parameter "authByQR": The value returned by "iAM Smart" System will be "true" in this scenario.
- "iAM Smart" System will not return the response parameter "ticketID" in this scenario.

3.7.1.2 Implementing (1) POST: Request PDF Digital Signing



Pre-conditions

- Please refer to Section 3.7.1.1 except:
  - Online Service generates the digest as the parameter "docDigest" from the pdf document to be signed using the Adobe.PPKLite filter and the adbe.pkcs7.detached subfilter.

Post-conditions

- Please refer to Section 3.7.1.1 except:
  - Computing 4-digit identification code with "docDigest" instead of "hashCode".

Error conditions

- Please refer to Section 3.7.1.1.

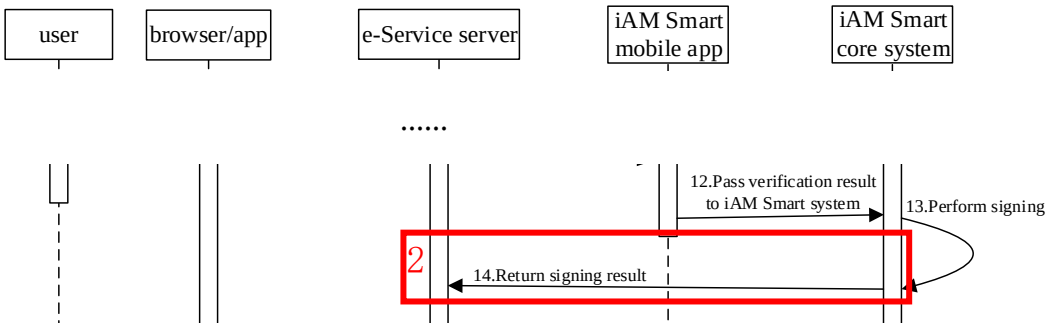
Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Please refer to Section 3.7.1.1, except “docDigest” request parameter is applied instead of “hashCode”.

3.7.1.3 Implementing (2) POST: Callback to Receive Digital Signing Result



Pre-conditions

- “iAM Smart” user can accept and complete the digital signing request.
- “iAM Smart” user can also reject the digital signing request.

Post-conditions

- API data decryption is required.
- Online Service Server should match the callback result with corresponding Online Service Website/App using the “businessID”.
- Online Service server completes the document digital signing process and acknowledges “iAM Smart” System the digital signing result.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D70000	User cancelled signing request	Inform user the “iAM Smart” digital signing request is cancelled
code - D70001	User rejected signing request	Inform user the “iAM Smart” digital signing request is rejected

code - D70002	Failed to request signing	Inform user the "iAM Smart" digital signing request is failed and retry later
code - D70003	Signing request timeout	Inform user the "iAM Smart" digital signing request is timeout and provide way for user to retry

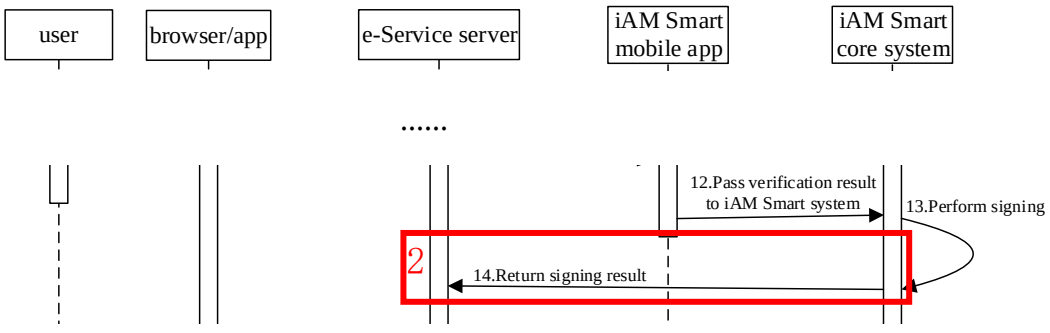
Callback Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- Callback parameter "hashCode": It is the Document Hash provided by Online Service and Online Service may use for checking purpose.
- Callback parameter "timestamp": It is the timestamp returned by "iAM Smart" System when the digital signing operation is successful.
- Callback parameter "signature": It is the RSA signature of the Document Hash submitted by Online Service for digital signing. The value is BASE64 encoded.
- Callback parameter "cert": It is the "iAM Smart" e-Cert of the "iAM Smart" user. It is in X.509 format and the value is BASE64 encoded.

3.7.1.4 Implementing (2) POST: Callback to Receive PDF Digital Signing Result



Pre-conditions

- Please refer to Section 3.6.1.5.

Post-conditions

- Please refer to Section 3.6.1.5.

Error conditions

- Please refer to Section 3.6.1.5.

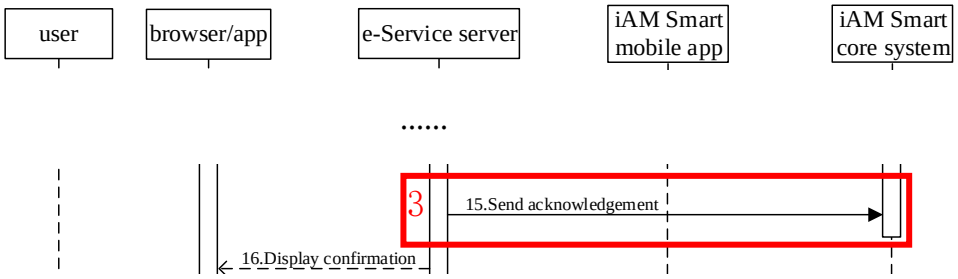
Callback Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- Callback parameter "docDigest": It is the pdf document digest submitted by Online Service and Online Service may use for checking purpose.
- Callback parameter "pdfSignature": It is the Base64-encoded PKCS#7 object that is the actual PDF signature value. It contains signer's certificate, signed hash value, and the digital signing timestamp information. Online Service can embed this value to the PDF document for future verification. "iAM Smart" will provide this to Online Service only when the digital signing is successful.

3.7.1.5 Implementing (3) POST: Online Service Acknowledges Digital Signing Result



Pre-conditions

- Online Service receives the "iAM Smart" digital signing result and completes its document digital signing process (i.e., verify the result).
- API data encryption is required.

Post-conditions

- Online Service may record the digital signing acknowledgement result.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D70006	Failed to process signing acknowledgement	"iAM Smart" digital signing acknowledgement processing failed, retry later

Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- Request parameter "businessID": It is the same "businessID" submitted in the digital signing request.
- Request parameter "signingResult": It is the status of Online Service document digital signing process after receiving the "iAM Smart" digital signing result. Its value must be equal to those specified in this API.

3.7.2 Scenario 2: Digital Signing (Online Service Website in Same Device)

The sequence diagram below shows how an authenticated user authorises and signs the Document Hash when Online Service website and the “iAM Smart” Mobile App are running in the same device.

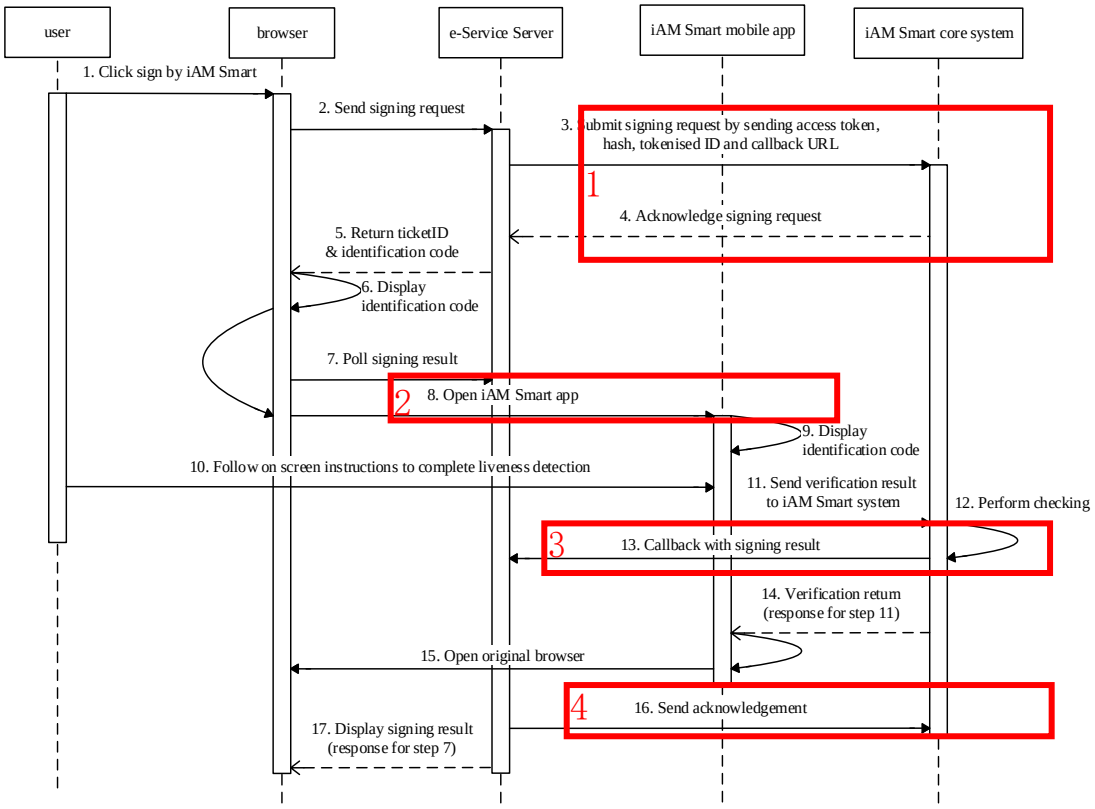


Figure-13 Digital Signing (Online Service Website in Same Device)

Step 1. User clicks the “Sign by iAM Smart” button in Online Service Website;

Step 2-3. Online Service Website initiates digital signing request to Online Service server with browser's user agent name (for use as value for request parameter “source”). Online Service initiates digital signing request to invoke “iAM Smart” API with the “businessID” of this request, accessToken, Tokenised ID, Document Hash / PDF digest, HKICHash, signature algorithm (only for "*Request Digital Signing*"), etc. The request parameter “source” will be browser's user agent value in this scenario. API data encryption is required;

“iAM Smart” API (POST: Request Digital Signing)

"iAM Smart" API (POST: Request PDF Digital Signing)

Step 4. "iAM Smart" System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the digital signing request to Online Service server by returning POST response with parameters "authByQR" (set to "false") and "ticketID";

Step 5-6. Online Service Server uses the Document Hash / PDF digest and hash of Tokenised ID to calculate a 4-digit identification code. Online Service prepares and sends necessary parameters such as "authByQR", "ticketID", 4-digit identification code, etc. to Online Service Website and instructs Online Service Website to display the instruction with the 4-digit identification Code to inform "iAM Smart" user to process the digital signing authorisation request in "iAM Smart" Mobile App;

Step 7. Online Service Website should keep synchronising with Online Service Server for digital signing processing result (e.g., polling);

Step 8. Online Service Website invokes "iAM Smart" Mobile App using URL Scheme with "ticketID" (set <Context> as "hash-sign" or "pdf-sign" in URL Scheme). Other parameters of this API are not used in this scenario.

"iAM Smart" API (URL Scheme: Open "iAM Smart" Mobile App for Getting Context)

Step 9. "iAM Smart" user logs in "iAM Smart" Mobile App, reviews the digital signing authorisation request (e.g., continue or reject the request) and verify the 4-digit identification code with Online Service Website;

Step 10-12. "iAM Smart" user follows the instructions in "iAM Smart" Mobile App to complete digital signing operation;

Step 13-14. "iAM Smart" System invokes Online Service callback API to return the result with "businessID" of the digital signing request to Online Service server. API data decryption is required;

Online Service Callback API (POST: Callback to Receive Digital Signing Result)

Online Service Callback API (POST: Callback to Receive PDF Digital Signing Result)

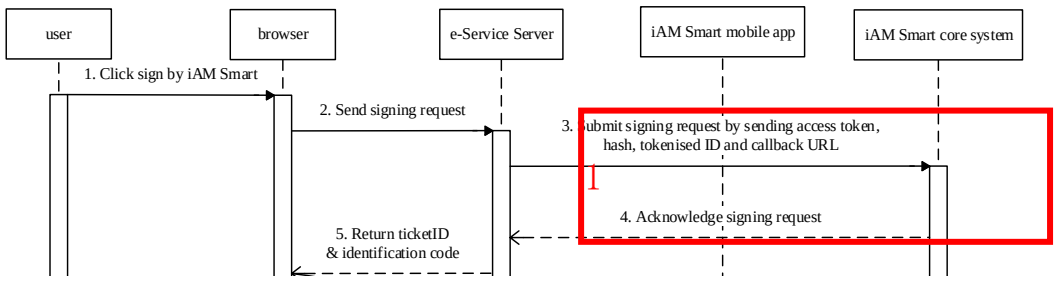
Step 15. "iAM Smart" System instructs "iAM Smart" Mobile App to invoke the original Online Service browser (i.e., using the browser's user agent name submitted in digital signing request in Step 3);

Step 16. Online Service Server completes the document digital signing process, verify the result and acknowledges "iAM Smart" System the digital signing result using "iAM Smart" API with the same "businessID" of the digital signing request. API data encryption is required;]

"iAM Smart" API (POST: Online Service Acknowledges Digital Signing Result)

Step 17. Online Service Server matches the result using the "businessID" and shows the digital signing result in corresponding Online Service Website.

3.7.2.1 Implementing (1) POST: Request Digital Signing



Pre-conditions

- Please refer to Section 3.7.1.1 except:
  - Online Service Website and "iAM Smart" Mobile App are in the same device in this scenario.

Post-conditions

- Please refer to Section 3.7.1.1 except:
  - Response "authByQR" is "false", "ticketID" will be provided by "iAM Smart" System in this scenario.

Error conditions

- Please refer to Section 3.7.1.1.

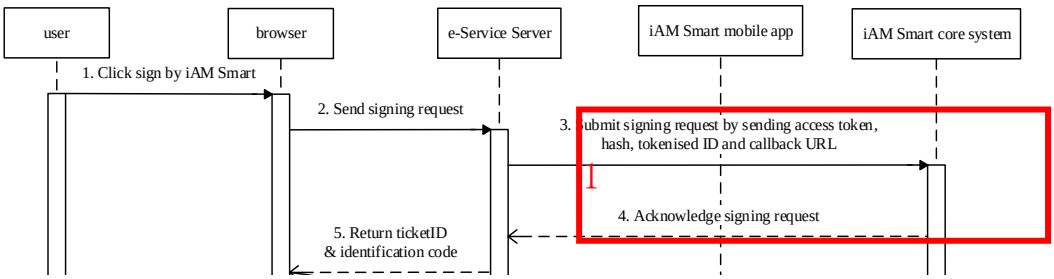
Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Please refer to Section 3.7.1.1, except for the following parameters:
  - Request parameter “source”: Value should be the browser's user agent value.
  - Response parameter “authByQR”: The value is “false” in this scenario.
  - Response parameter “ticketID”: It will be provided by “iAM Smart” System in this scenario. It is a 36-byte (or less) UUID number (ASCII character set).

3.7.2.2 Implementing (1) POST: Request PDF Digital Signing



Pre-conditions

- Please refer to Section 3.7.1.2 except:
  - Online Service generates the digest as the parameter “docDigest” from the pdf document to be signed using the Adobe.PPKLite filter and the adbe.pkcs7.detached subfilter.

Post-conditions

- Please refer to Section 3.7.1.2 except:
  - Computing 4-digit identification code with “docDigest” instead of “hashCode”.

Error conditions

- Please refer to Section 3.7.1.2.

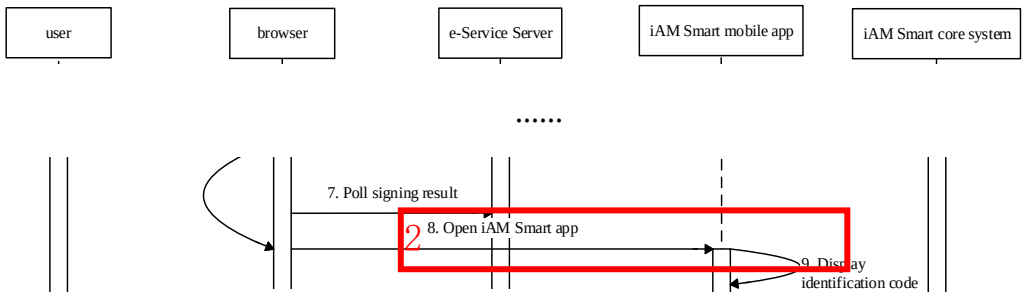
Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Please refer to Section 3.7.2.1, except “docDigest” request parameter is applied instead of “hashCode”.

3.7.2.3 Implementing (2) URL Scheme: Open “iAM Smart” Mobile App for Getting Context



Pre-conditions

- Online Service Website and “iAM Smart” Mobile App are in the same device.
- Online Service has the “ticketID” provided by “iAM Smart” System for the digital signing request;
- Other parameters of this API are not used in this scenario.

Post-conditions

- “iAM Smart” Mobile App will be launched.
- Online Service Website should keep synchronising with Online Service server for the callback response of the digital signing request from “iAM Smart” System.

Error conditions

- Nil

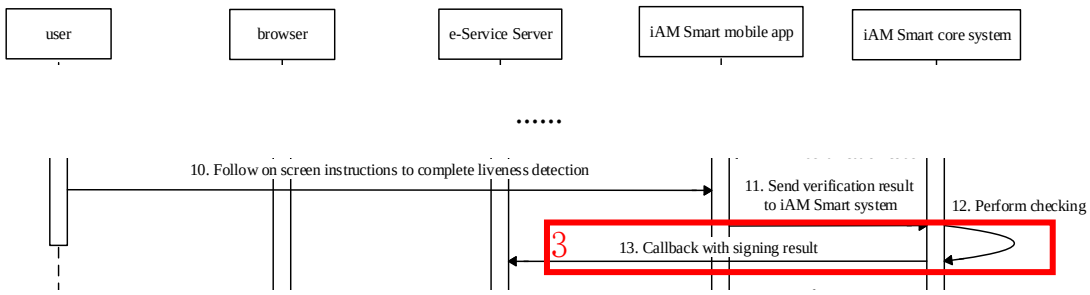
Request Parameters

- Please refer to “iAM Smart” API Specification.

Notes

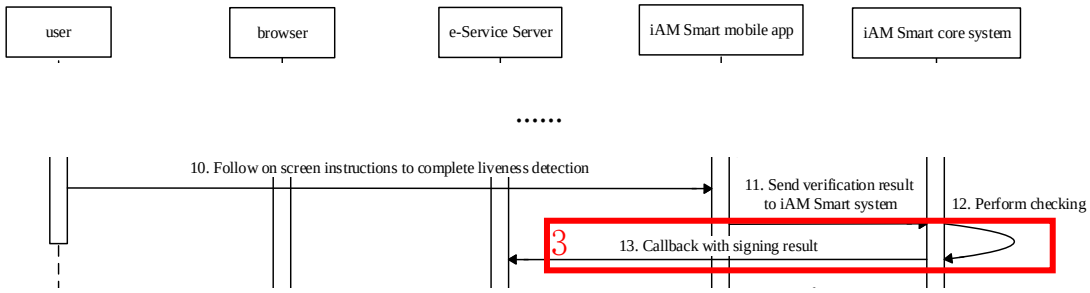
- Set <Context> as “hash-sign” or “pdf-sign” in URL Scheme.

3.7.2.4 Implementing (3) POST: Callback to Receive Digital Signing Result



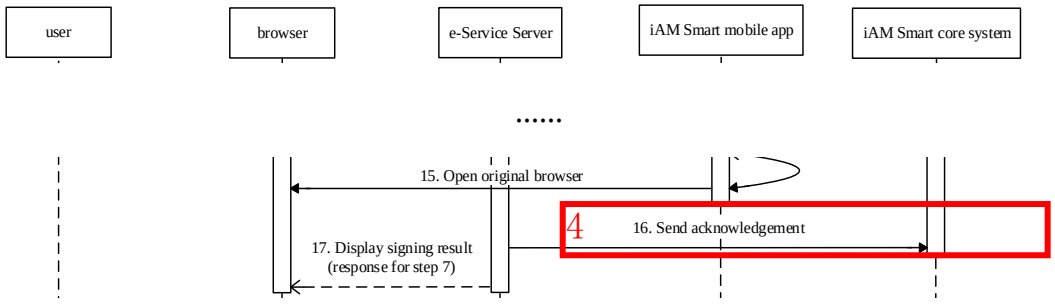
Please refer to Section 3.7.1.3.

3.7.2.5 Implementing (3) POST: Callback to Receive PDF Digital Signing Result



Please refer to Section 3.7.1.4.

3.7.2.6 Implementing (4) POST: Online Service Acknowledges Digital Signing Result



Please refer to Section 3.7.1.5.

3.7.3 Scenario 3: Digital Signing (Online Service App in Same Device)

The sequence diagram below shows how an authenticated user authorises and signs the Document Hash when Online Service App and the "iAM Smart" Mobile App are running in the same device.

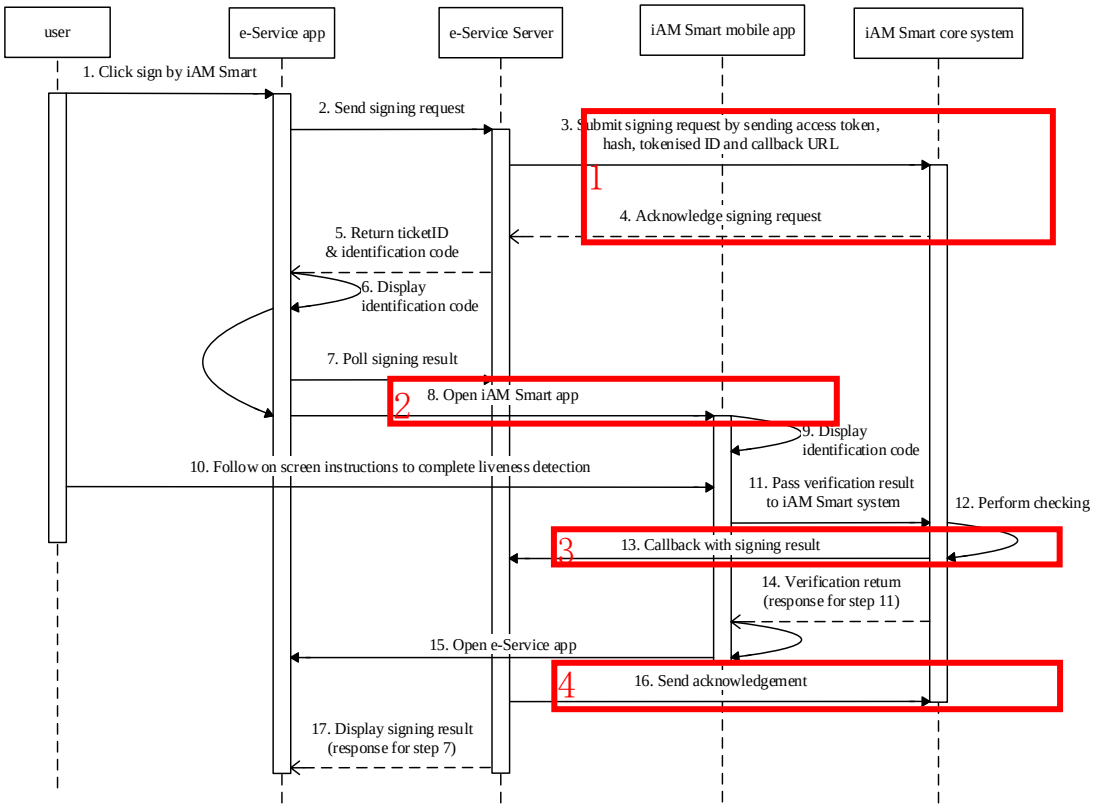


Figure-14 Digital Signing (Online Service App in Same Device)

Step 1. User clicks the "Sign by iAM Smart" button in Online Service App;

Step 2-3. Online Service App initiates digital signing request to Online Service server with "App_Scheme" or "App_Link" (use as the value of request parameter "source"). Online Service initiates digital signing request to invoke "iAM Smart" API with the "businessID" of this request, accessToken, Tokenised ID, Document Hash / PDF digest, HKICHash, signature algorithm (only for "*Request Digital Signing*"), etc. The request parameter "source" will be "App_Scheme" or "App_Link" in this scenario. API data encryption is required;

"iAM Smart" API (POST: Request Digital Signing)

"iAM Smart" API (POST: Request PDF Digital Signing)

Step 4. "iAM Smart" System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the digital signing request to Online Service server by returning POST response with parameters "authByQR" (set to "false") and "ticketID";

Step 5-6. Online Service server uses the Document Hash / PDF digest and hash of Tokenised ID to calculate a 4-digit identification code. Online Service prepares and sends necessary parameters such as "authByQR", "ticketID", 4-digit identification code, etc. to Online Service App. Online Service App displays the instruction with the 4-digit identification Code to inform "iAM Smart" user to process the digital signing authorisation request in "iAM Smart" Mobile App;

Step 7. Online Service App should keep synchronising with Online Service Server for digital signing processing result (e.g., polling);

Step 8. Online Service App invokes "iAM Smart" Mobile App using URL Scheme with "ticketID" (set <Context> as "hash-sign" or "pdf-sign" in URL Scheme). Other parameters of this API are not used in this scenario;

"iAM Smart" API (URL Scheme: Open "iAM Smart" Mobile App for Getting Context)

Step 9. "iAM Smart" user logs in "iAM Smart" Mobile App, reviews the digital signing authorisation request (e.g., continue or reject the request) and verifies the 4-digit identification code with Online Service App;

Step 10-12. "iAM Smart" user follows the instructions in "iAM Smart" Mobile App to complete digital signing operation;

Step 13. "iAM Smart" System invokes Online Service callback API to return the result with "businessID" of the digital signing request to Online Service Server. API data decryption is required;

Online Service Callback API (POST: Callback to Receive Digital Signing Result)

Online Service Callback API (POST: Callback to Receive PDF Digital Signing Result)

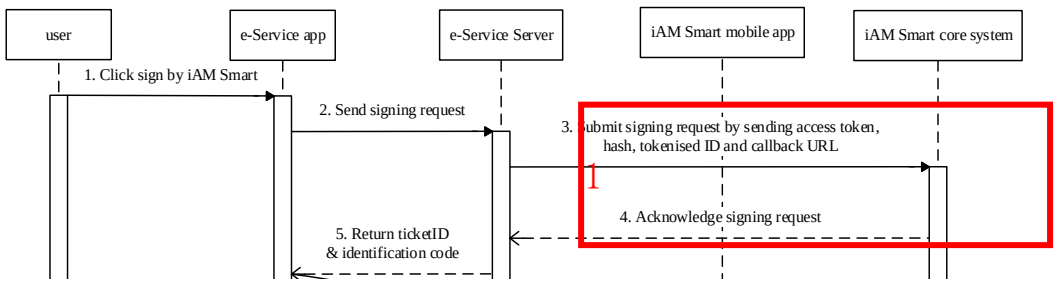
Step 14-15. "iAM Smart" System instructs "iAM Smart" Mobile App to invoke the Online Service App using URL Scheme, or Universal Link/App Link ("iAM Smart" System queries the information registered at Online Service registration depending on the "source" submitted in Step 3);

Step 16. Online Service Server completes the document digital signing process, verify the result and acknowledges "iAM Smart" System the digital signing result using "iAM Smart" API with the same "businessID" of the digital signing request. API data encryption is required;

"iAM Smart" API (POST: Online Service Acknowledges Digital Signing Result)

Step 17. Online Service Server matches the result using the "businessID" and shows the digital signing result in corresponding Online Service App.

3.7.3.1 Implementing (1) POST: Request Digital Signing



Pre-conditions

- Please refer to Section 3.7.2.1 except:
  - Online Service App and "iAM Smart" Mobile App are in the same device in this scenario.

Post-conditions

- Please refer to Section 3.7.2.1.

Error conditions

- Please refer to Section 3.7.2.1.

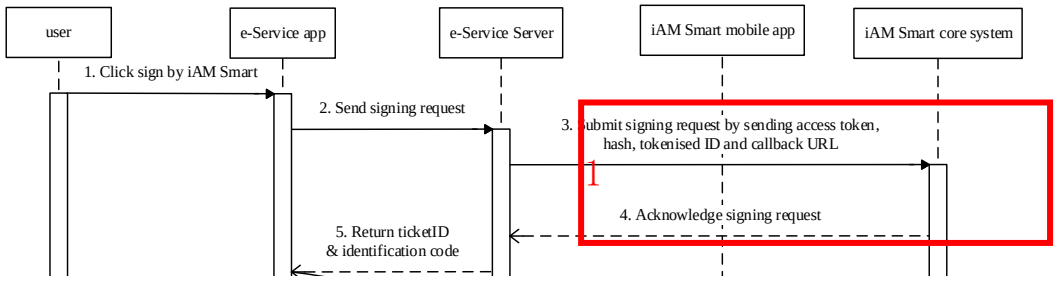
Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Please refer to Section 3.7.2.1, except for the following parameter:
  - Request parameter “source”: Value should be “App_Scheme” (for the Online Service App URL scheme), or “App_Link” (for the Universal Link/App Link).

3.7.3.2 Implementing (1) POST: Request PDF Digital Signing



Pre-conditions

- Please refer to Section 3.7.2.2 except:
  - Online Service generates the digest as the parameter “docDigest” from the pdf document to be signed using the Adobe.PPKLite filter and the adbe.pkcs7.detached subfilter.

Post-conditions

- Please refer to Section 3.7.2.2 except:
  - Computing 4-digit identification code with “docDigest” instead of “hashCode”.

Error conditions

- Please refer to Section 3.7.2.2.

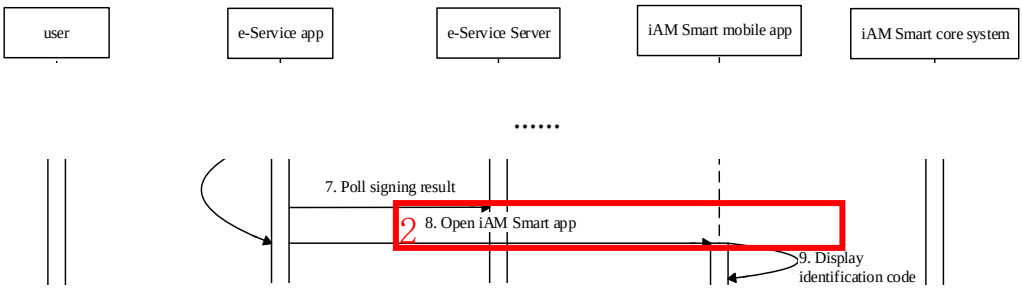
Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

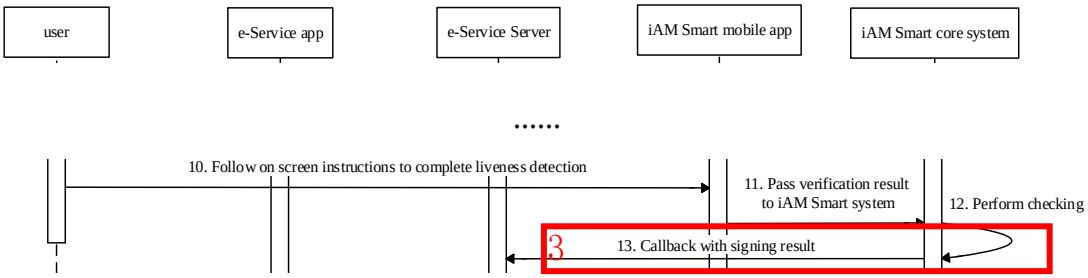
- Please refer to Section 3.7.2.2, except “docDigest” request parameter is applied instead of “hashCode”.

3.7.3.3 Implementing (2) URL Scheme: Open “iAM Smart” Mobile App for Getting Context



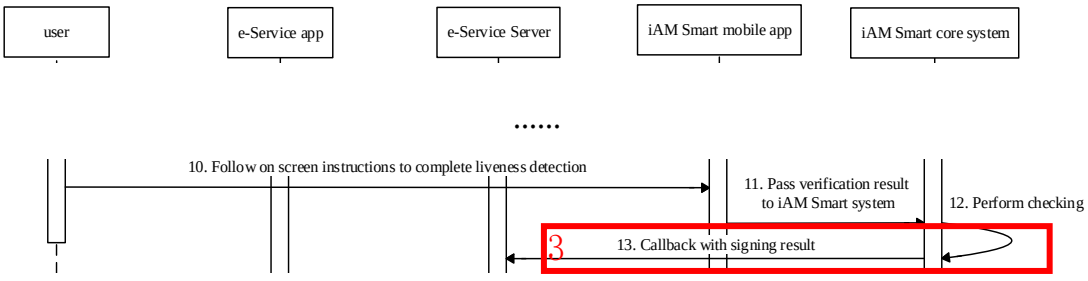
Please refer to Section 3.7.2.3.

3.7.3.4 Implementing (3) POST: Callback to Receive Digital Signing Result



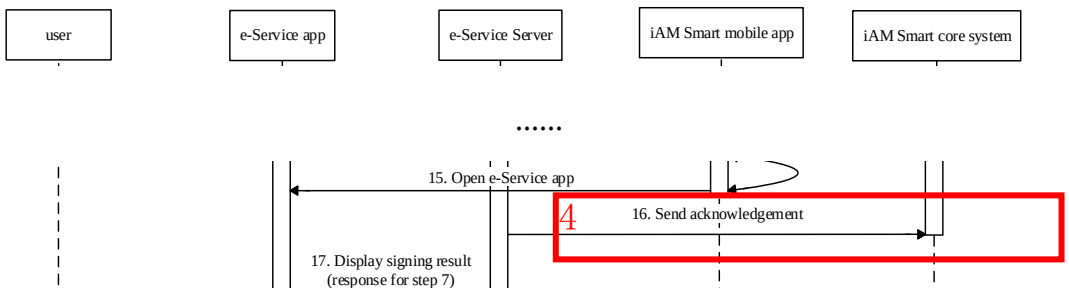
Please refer to Section 3.7.1.3.

3.7.3.5 Implementing (3) POST: Callback to Receive PDF Digital Signing Result



Please refer to Section 3.7.1.4.

3.7.3.6 Implementing (4) POST: Online Service Acknowledges Digital Signing Result



Please refer to Section 3.7.1.5.

3.8 WORKFLOWS FOR DIGITAL SIGNING WITHOUT SERVICE LOGIN (AKA ANONYMOUS DIGITAL SIGNING)

The following process includes anonymous hash digital signing and anonymous pdf digital signing.

3.8.1 Scenario 1: Anonymous Digital Signing (Online Service Website in Different Device)

The sequence diagram below shows how an anonymous user authorises and signs the Document Hash when Online Service website and the "iAM Smart" Mobile App are running in different devices.

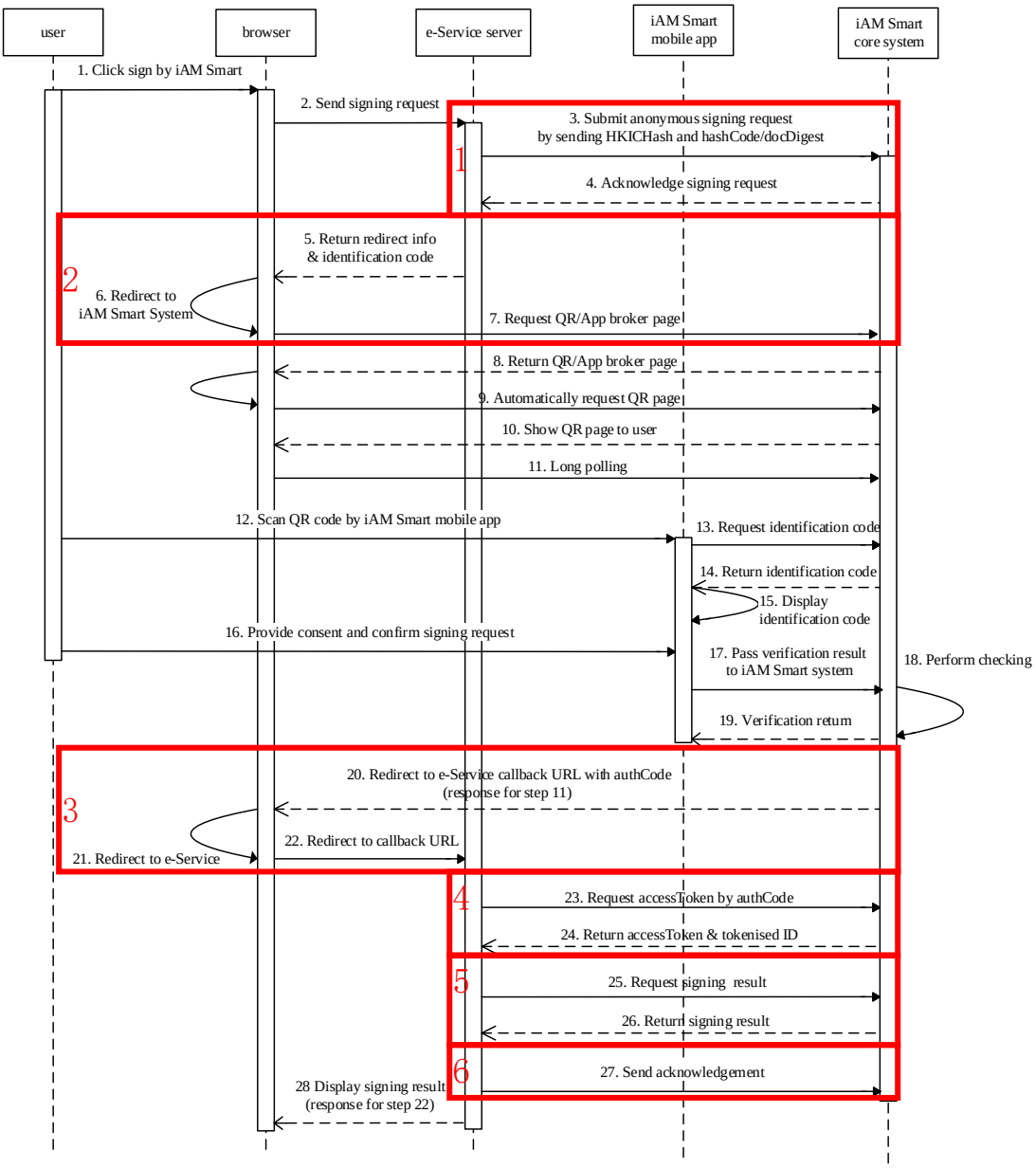


Figure-15 Anonymous Digital Signing (Online Service Website in Different Device)

Step 1. After entering HKIC number, the user clicks the “Anonymous hash digital signing” (or “Anonymous pdf digital signing”) button in Online Service Website;

Step 2. Online Service Website initiates anonymous digital signing request to Online Service Server with browser's user agent name (use as value for request parameter “source”);

Step 3. Online Service Server initiates an anonymous digital signing request to invoke the “iAM Smart” API with the “businessID” of this request, Document Hash / PDF digest, HKICHash, signature algorithm (only for "*Request Anonymous Digital Signing*"), etc. API encryption is required;

“iAM Smart” API (POST: Request Anonymous Digital Signing)

“iAM Smart” API (POST: Request Anonymous PDF Digital Signing)

Step 4. “iAM Smart” System verifies and confirms receipt of the anonymous digital signing request to Online Service Server by returning POST response with parameter “ticketID”;

Step 5-7. Online Service Server uses the Document Hash / PDF digest, HKICHash and hash of clientID to calculate a 4-digit identification code and instructs Online Service Website to display the instructions with the 4-digit identification code to inform “iAM Smart” user to process the digital signing authorisation request in “iAM Smart” Mobile App;

Online Service Server prepares the request parameters such as “clientID”, “redirectURI”, “source” (set as browser’s user agent value), “scope”, “ticketID”, etc. and constructs the GET request to invoke the “iAM Smart” API using browser redirection;

“iAM Smart” API (GET: Request QR page)

Step 8-10. “iAM Smart” System returns the broker page (if Online Service has set “brokerPage” to “true”) and after the broker page fails to find the “iAM Smart” Mobile App, it will request QR page to be displayed in browser. If Online Service does not request for

broker page (i.e., no broker page will be returned), the browser will directly display QR Code page;

Step 11. QR Code page of "iAM Smart" System polls "iAM Smart" System for the QR Code status;

Step 12. "iAM Smart" user logs in "iAM Smart" Mobile App to scan the QR Code;

Step 13-15. "iAM Smart" Mobile App requests and displays 4-digit identification code from "iAM Smart" System. "iAM Smart" user reviews the digital signing authorisation request (e.g., continue or reject the request) and verifies the 4-digit identification code with Online Service Website;

Step 16-17. "iAM Smart" user follows the instructions in "iAM Smart" Mobile App to complete digital signing operation;

Step 18-19. "iAM Smart" System verifies the validity of QR code and other necessary information, and return the authorisation result to "iAM Smart" Mobile App;

Step 20-22. QR Code page of "iAM Smart" System receives the polling result returned by "iAM Smart" System (i.e., response for the polling in Step 11) which includes authCode and businessID or any error code (e.g., "iAM Smart" user rejects the request), assembles and invokes the Online Service callback API given in "redirectURI" using browser redirection;

Online Service Callback API (GET: Callback with authCode to Online Service Server)

Step 23-24. When Online Service Server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e., openID) of the "iAM Smart" user. API data encryption is required;

"iAM Smart" API (POST: Request accessToken & Tokenised ID)

Step 25-26. Online Service Server invokes the "iAM Smart" API with the accessToken and openID to obtain the result of anonymous digital signing. API data encryption is required;

"iAM Smart" API (POST: Obtain Anonymous Digital Signing Result)

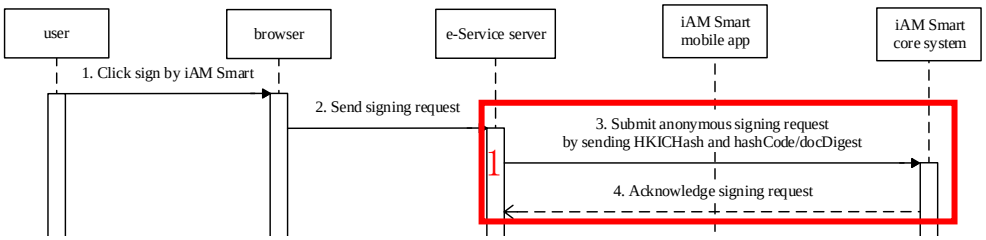
"iAM Smart" API (POST: Obtain Anonymous PDF Digital Signing Result)

Step 27. Online Service Server completes the document digital signing process, verify the result and acknowledges "iAM Smart" System the digital signing result using "iAM Smart" API with the same "businessID" of the digital signing request. API data encryption is required;

"iAM Smart" API (POST: Online Service Acknowledges Digital Signing Result)

Step 28. Online Service Server matches the result using the "businessID" and shows the digital signing result in corresponding Online Service Website.

3.8.1.1 Implementing (1) POST: Request Anonymous Digital Signing



Pre-conditions

- Please refer to Section 3.7.1.1 except:
  - No request parameter "accessToken" exists.
  - Online Service generates a "businessID" for this request and uses this identifier to match the authCode returned from "iAM Smart" System.
  - Online Service should convert the HKIC number (no check digit is needed) into hash value using SHA256.

Post-conditions

- Please refer to Section 3.7.1.1 except:
  - No response parameter "authByQR" exists.
  - Online Service calculates a 4-digit identification code using:

    Document Hash + HKICHash + clientID hash value (calculated with SHA-512)

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D70002	Failed to request signing	Inform user the "iAM Smart" digital signing request is failed and retry later

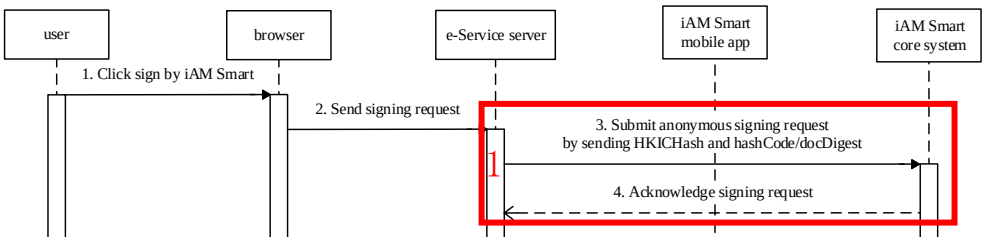
Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- Please refer to Section 3.7.1.1, except for the following parameters:
  - No request parameter "accessToken", "openID", "source", "redirectURI" and "state".
  - No response parameter "authByQR".
  - Response parameter "ticketID": It is returned from "iAM Smart" System for "iAM Smart" APIs for anonymous request. It will be used for invoking subsequent "iAM Smart" APIs "Request QR Page" and "Open "iAM Smart" Mobile App for Getting Context" depending on the scenario. It is a 36-byte (or less) UUID number (ASCII character set).

3.8.1.2 Implementing (1) POST: Request Anonymous Anonymous PDF Digital Signing



Pre-conditions

- Please refer to Section 3.7.3.2.

Post-conditions

- Please refer to Section 3.7.3.2 except:
  - Online Service calculates a 4-digit identification code using:

docDigest of PDF document + HKICHash + clientID hash value (calculated with SHA-512)

Error conditions

- Please refer to Section 3.7.3.2.

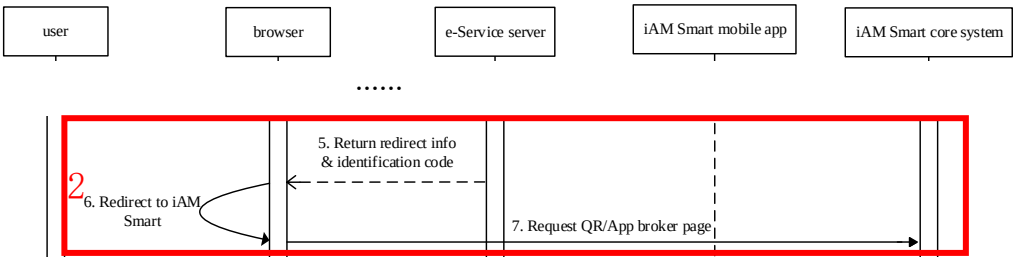
Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- Please refer to Section 3.7.3.2.

3.8.1.3 Implementing (2) GET: Request QR Page



Pre-conditions

- Please refer to Section 3.4.1.1 except:
  - Online Service has the "ticketID" provided by "iAM Smart" System for the anonymous request in step 4.

Post-conditions

- Please refer to Section 3.4.1.1.

Error conditions

- Please refer to Section 3.4.1.1.

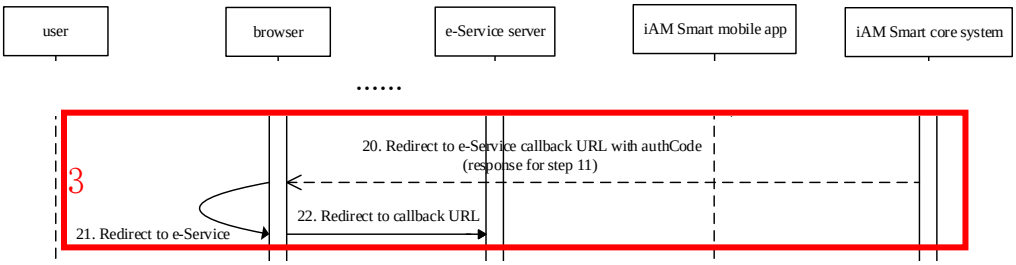
Request Parameters

- Please refer to Section 3.4.1.1.

Notes

- Please refer to Section 3.4.1.1 except the following parameter:
  - Request parameter "ticketID": Value provided by "iAM Smart" System for the anonymous request.

3.8.1.4 Implementing (3) GET: Callback with authCode to Online Service Server



Pre-conditions

- Please refer to Section 3.4.1.2.

Post-conditions

- Please refer to Section 3.4.1.2 except:
  - Online Service Server should match the callback result with corresponding Online Service client terminal using the "businessID".

Error conditions

- This Online Service callback API is a HTTP GET request. If parameter "error_code" is returned, it means the request is failed.

Error Code	Error Description	Suggested Action
error_code - D70000	User cancelled signing request	Inform user the digital signing request is cancelled
error_code - D70001	User rejected signing request	Inform user the digital signing request is rejected
error_code - D70002	Failed to request signing	Inform user the digital signing request is failed and retry later
error_code - D70004	User not allowed to sign	Inform user that "iAM Smart" digital signing is not allowed for default "iAM Smart" and suggest user to upgrade to "iAM Smart" with digital signing function

Error Code	Error Description	Suggested Action
error_code - D70005	Inconsistent HKIC number	Hash of HKIC number provided is not matched with "iAM Smart" user. Check hash of HKIC number or inform user to authorise the digital signing using the correct "iAM Smart" account

The error_code D70003 (signing request timeout) does not appear in this scenario. For the QR code timeout or request confirmation timeout in the "iAM Smart" Mobile App, message will be prompted in the corresponding user interface.

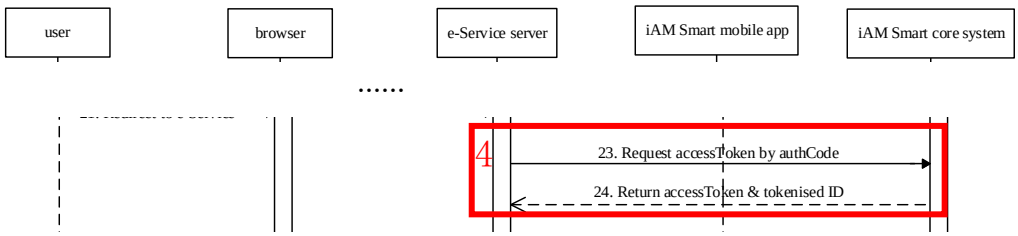
Callback Parameters

- Please refer to Section 3.4.1.2.

Notes

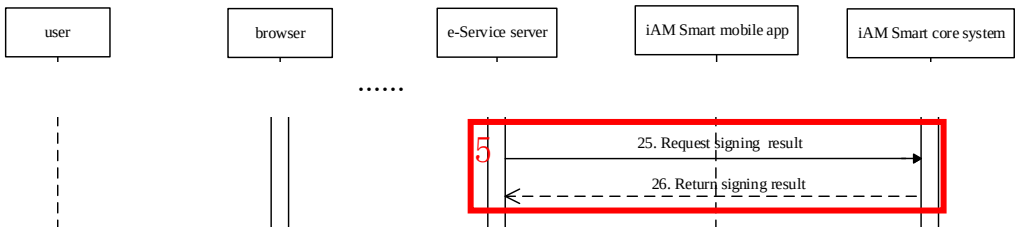
- Please refer to Section 3.4.1.2 except the following parameter:
  - Callback parameter "businessID": Value returned by "iAM Smart" System should be the same one as Online Service submitted in the anonymous request.

3.8.1.5 Implementing (4) POST: Request accessToken & Tokenised ID



Please refer to Section 3.4.1.3

3.8.1.6 Implementing (5) POST: Obtain Anonymous Digital Signing Result



Pre-conditions

- Online Service must possess a valid accessToken of the "iAM Smart" user with authorisation scope "Signing authorisation".
- API data encryption is required.

Post-conditions

- API data decryption is required.
- Online Service Server completes the document digital signing process and acknowledges "iAM Smart" System the digital signing result using "iAM Smart" API with the same "businessID" of the digital signing request.
- Online Service should discard the accessToken as it can only be used once.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D70002	Failed to request signing	Inform user the "iAM Smart" digital signing request is failed and retry later
code - D70003	Signing request timeout	Inform user the "iAM Smart" digital signing request is timeout and provide way for user to retry
code - D70004	User not allowed to sign	Inform user that "iAM Smart" digital signing is not allowed for default "iAM Smart" and suggest user to upgrade to "iAM Smart" with digital signing function

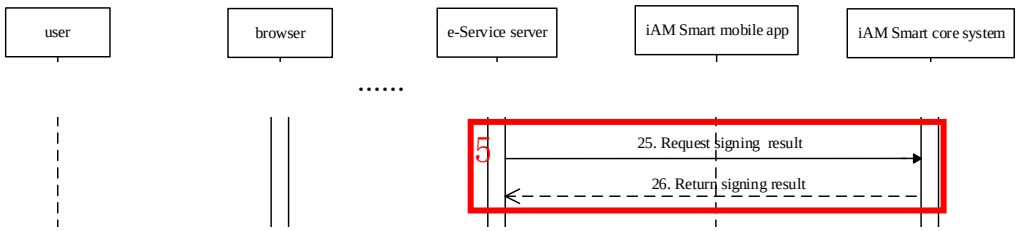
.Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Please refer to Section 3.7.1.3 except:
  - Response parameters “businessID” and “state” do not exist.

3.8.1.7 Implementing (5) POST: Obtain Anonymous PDF Digital Signing Result



Pre-conditions

- Please refer to Section 3.6.4.5

Post-conditions

- Please refer to Section 3.6.4.5.

Error conditions

- Please refer to Section 3.6.4.5.

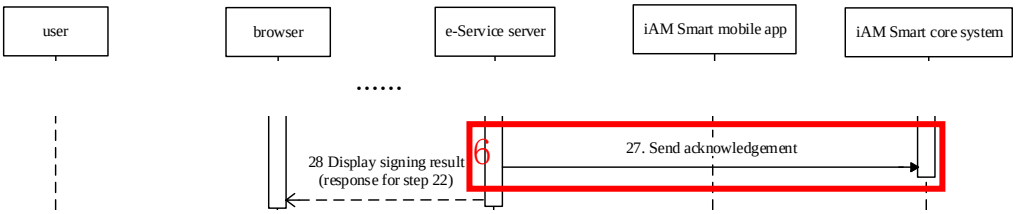
Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Please refer to Section 3.7.1.4 except:
  - Response parameter “businessID” and “state” do not exist.

3.8.1.8 Implementing (6) POST: Online Service Acknowledges Digital Signing Result



Please refer to Section 3.7.1.5.

3.8.2 Scenario 2: Anonymous Digital Signing (Online Service Website in Same Device)

The sequence diagram below shows how an anonymous user authorises and signs the Document Hash when Online Service website and the "iAM Smart" Mobile App are running in the same devices.

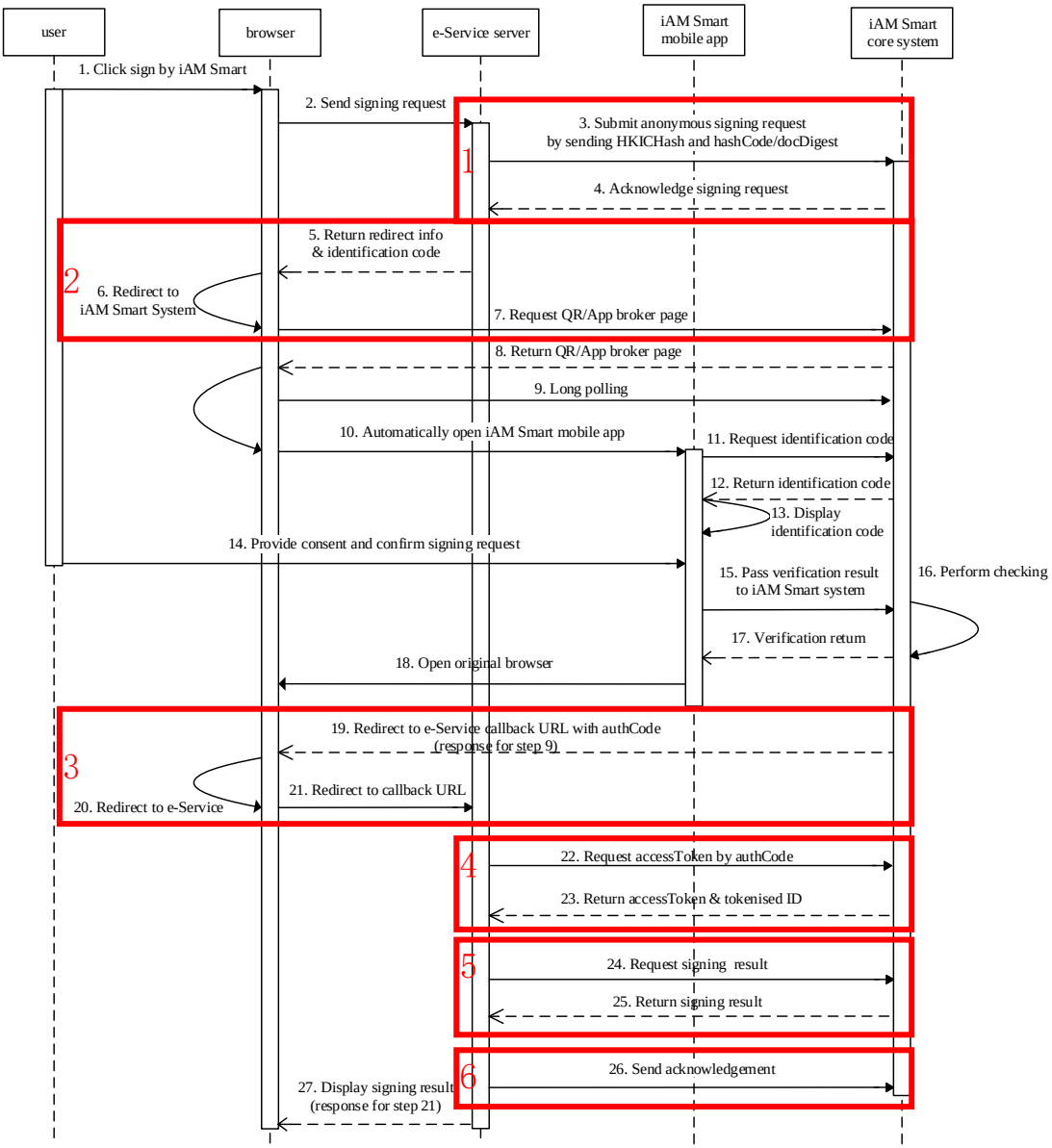


Figure-16 Anonymous Digital Signing (Online Service Website in Same Device)

Step 1. After entering HKIC number, the user clicks the “Anonymous hash digital signing” (or “Anonymous pdf digital signing”) button in Online Service Website;

Step 2. Online Service Website initiates anonymous digital signing request to Online Service Server with browser's user agent name (use as value for request parameter “source”);

Step 3. Online Service Server initiates an anonymous digital signing request to invoke the “iAM Smart” API with the “businessID” of this request, Document Hash / PDF digest, HKICHash, signature algorithm (only for "*Request Anonymous Digital Signing*"), etc. API data encryption is required;

“iAM Smart” API (POST: Request Anonymous Digital Signing)

“iAM Smart” API (POST: Request Anonymous PDF Digital Signing)

Step 4. “iAM Smart” System verifies and confirms receipt of the anonymous digital signing request to Online Service Server by returning POST response with parameter “ticketID”;

Step 5-7. Online Service Server uses the Document Hash / PDF digest, HKICHash and hash of clientID to calculate a 4-digit identification code and instructs Online Service Website to display the instructions with the 4-digit identification code to inform “iAM Smart” user to process the digital signing authorisation request in “iAM Smart” Mobile App;

Online Service Server prepares the request parameters such as “clientID”, “redirectURI”, “source” (set as browser’s user agent value), “scope”, “brokerPage” (set as “true”), “ticketID”, etc. and constructs the GET request to invoke the “iAM Smart” API using browser redirection;

“iAM Smart” API (GET: Request QR page)

Step 8. “iAM Smart” System returns the broker page to the Online Service Website;

Step 9. Broker page of “iAM Smart” System polls “iAM Smart” System for further action;

Step 10. Broker page invokes the “iAM Smart” Mobile App in the same device automatically and sends it the relevant parameters.

Step 11-13. “iAM Smart” user logs in “iAM Smart” Mobile App, “iAM Smart” Mobile App requests and displays 4-digit identification code from “iAM Smart” System. “iAM Smart” user reviews the digital signing authorisation request (e.g., continue or reject the request) and verifies the 4-digit identification code with Online Service Website;

Step 14-15. “iAM Smart” user follows the instructions in “iAM Smart” Mobile App to complete digital signing operation;

Step 16-18. “iAM Smart” System verifies the validity of necessary information, and return the authorisation result to “iAM Smart” Mobile App. “iAM Smart” Mobile App invokes the original Online Service browser (i.e., using the browser's user agent value submitted);

Step 19-21. Broker page of “iAM Smart” System receives the polling result returned by “iAM Smart” System (i.e., response for the polling in Step 9) which includes authCode and businessID or any error code (e.g., “iAM Smart” user rejects the request), assembles and invokes the Online Service callback API given in “redirectURI” using browser redirection;

Online Service Callback API (GET: Callback with authCode to Online Service Server)

Step 22-23. When Online Service Server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invoke the “iAM Smart” API to obtain the accessToken and the Tokenised ID (i.e., openID) of the “iAM Smart” user. API data encryption is required;

“iAM Smart” API (POST: Request accessToken & Tokenised ID)

Step 24-25. Online Service Server invokes the “iAM Smart” API with the accessToken and openID to obtain the result of anonymous digital signing. API data encryption is required;

“iAM Smart” API (POST: Obtain Anonymous Digital Signing Result)

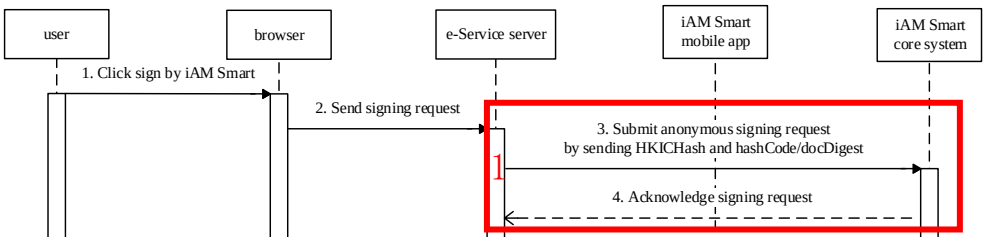
"iAM Smart" API (POST: Obtain Anonymous PDF Digital Signing Result)

Step 26. Online Service Server completes the document digital signing process, verify the result and acknowledges "iAM Smart" System the digital signing result using "iAM Smart" API with the same "businessID" of the digital signing request. API data encryption is required;

"iAM Smart" API (POST: Online Service Acknowledges Digital Signing Result)

Step 27. Online Service Server matches the result using the "businessID" and shows the digital signing result in corresponding Online Service Website.

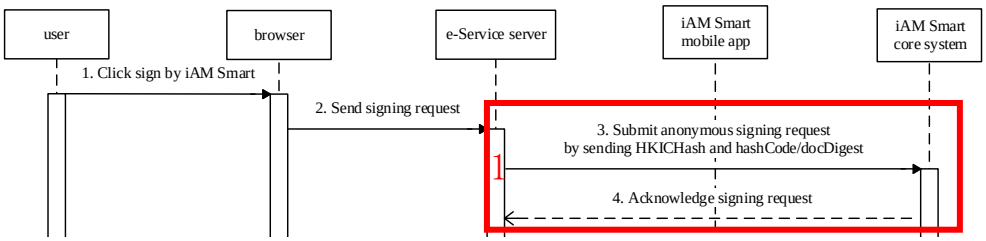
3.8.2.1 Implementing (1) POST: Request Anonymous Digital Signing



Please refer to Section 3.8.1.1 except:

- Online Service Website and "iAM Smart" Mobile App are in the same device in this scenario.

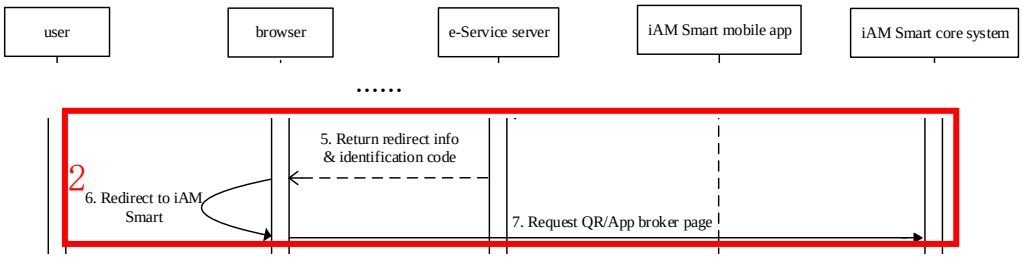
3.8.2.2 Implementing (1) POST: Request Anonymous PDF Digital Signing



Please refer to Section 3.8.1.2 except:

- Online Service Website and "iAM Smart" Mobile App are in the same device in this scenario.

3.8.2.3 Implementing (2) GET: Request QR Page



Pre-conditions

- Please refer to Section 3.4.2.1 except:
  - Online Service has the "ticketID" provided by "iAM Smart" System for the anonymous request in step 4.

Post-conditions

- Please refer to Section 3.4.2.1.

Error conditions

- Please refer to Section 3.4.2.1.

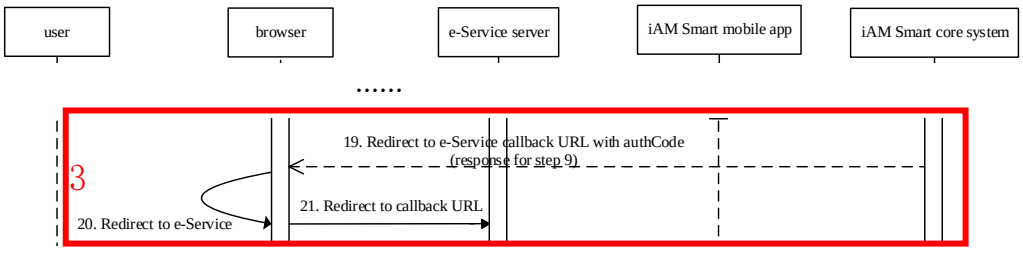
Request Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- Please refer to Section 3.4.2.1 except the following parameter:
  - Request parameter "ticketID": Value provided by "iAM Smart" System for the anonymous request.

3.8.2.4 Implementing (3) GET: Callback with authCode to Online Service Server



Pre-conditions

- Please refer to Section 3.6.2.3.

Post-conditions

- Please refer to Section 3.6.2.3.

Error conditions

- Please refer to Section 3.8.1.4.

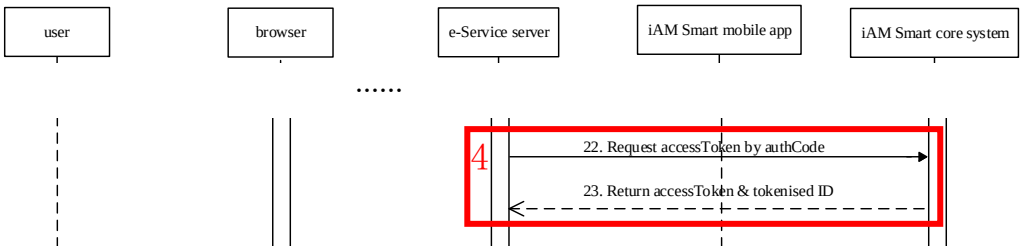
Callback Parameters

- Please refer to Section 3.6.2.3.

Notes

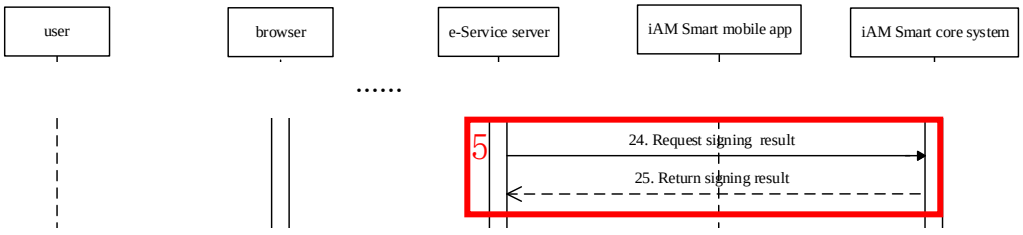
- Please refer to Section 3.6.2.3.

3.8.2.5 Implementing (4) POST: Request accessToken & Tokenised ID



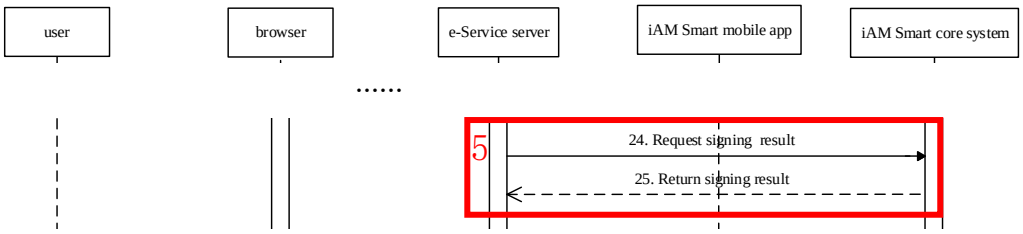
Please refer to Section 3.6.1.4.

3.8.2.6 Implementing (5) POST: Obtain Anonymous Digital Signing Result



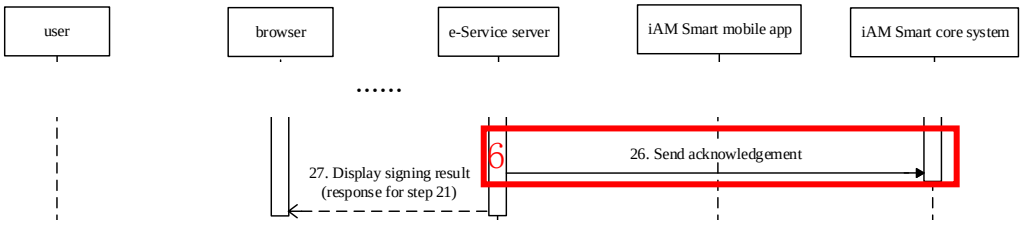
Please refer to Section 3.8.1.6.

3.8.2.7 Implementing (5) POST: Obtain Anonymous PDF Digital Signing Result



Please refer to Section 3.8.1.7.

3.8.2.8 Implementing (6) POST: Online Service Acknowledges Digital Signing Result



Please refer to Section 3.7.1.5.

3.8.3 Scenario 3: Anonymous Digital Signing (Online Service App in Different Device)

The sequence diagram below shows how an anonymous user authorises and signs the Document Hash when Online Service App and the "iAM Smart" Mobile App are running in different devices.

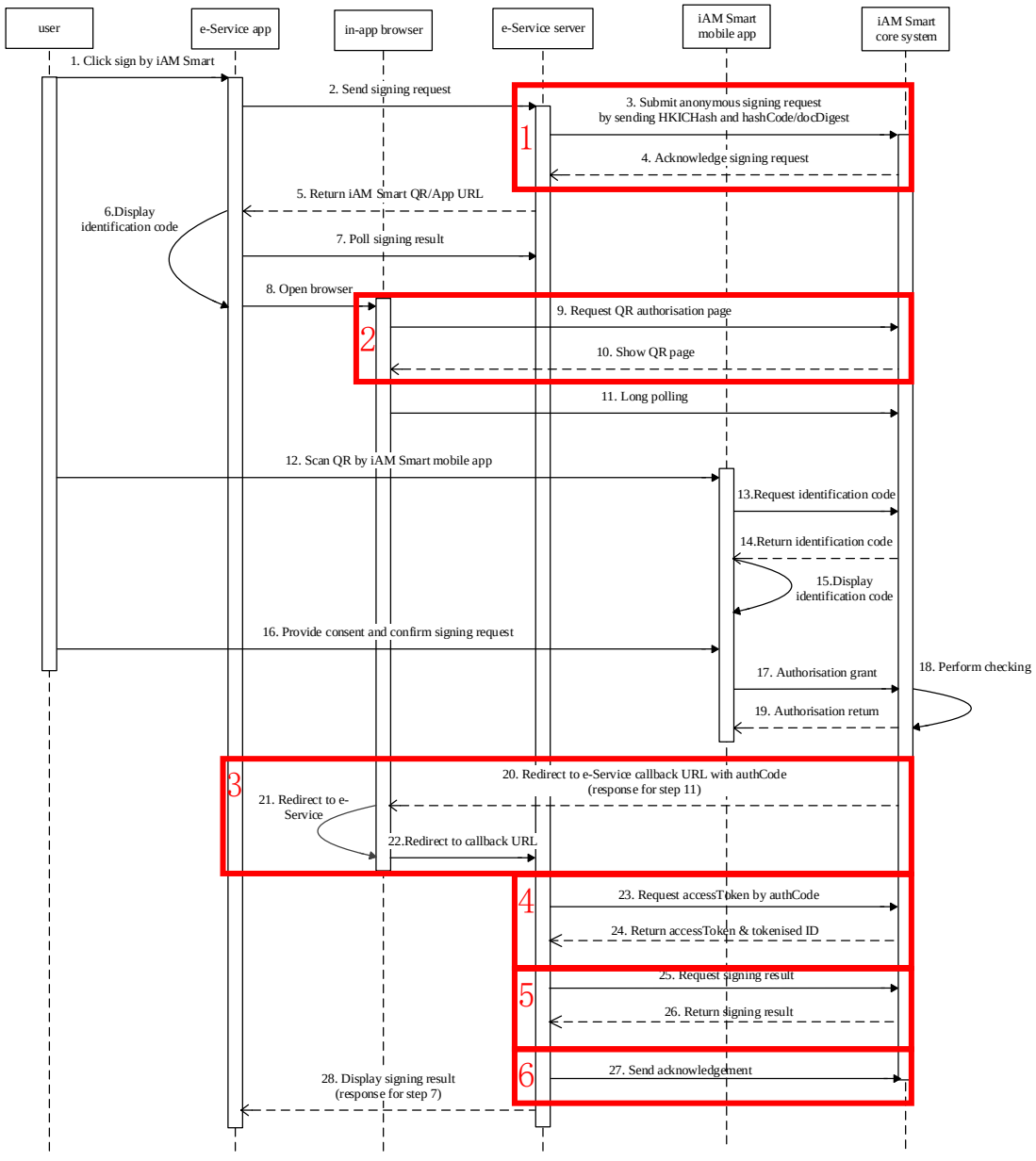


Figure-17 Anonymous Digital Signing (Online Service App in Different Device)

Step 1. After entering HKIC number, the user clicks the “Anonymous hash digital signing” (or “Anonymous pdf digital signing”) button in Online Service App;

Step 2. Online Service App determines there is no “iAM Smart” Mobile App in the device using program code, initiates anonymous digital signing request to Online Service Server with the in-app browser’s user agent value (use as value for request parameter “source”);

Step 3. Online Service Server initiates an anonymous digital signing request to invoke the “iAM Smart” API with the “businessID” of this request, Document Hash / PDF digest, HKICHash, signature algorithm (only for "*Request Anonymous Digital Signing*"), etc. API data encryption is required;

“iAM Smart” API (POST: Request Anonymous Digital Signing)

“iAM Smart” API (POST: Request Anonymous PDF Digital Signing)

Step 4. “iAM Smart” System verifies and confirms receipt of the anonymous digital signing request to Online Service Server by returning POST response with parameter “ticketID”;

Step 5-6. Online Service Server uses the Document Hash / PDF digest, HKICHash and hash of clientID to calculate a 4-digit identification code and instructs Online Service App to display the instructions with the 4-digit identification code to inform “iAM Smart” user to process the digital signing authorisation request in “iAM Smart” Mobile App;

Online Service Server prepares necessary parameters such as “clientID”, “redirectURI”, “scope”, “source” (set as in-app browser’s user agent value), “brokerPage” (set as “false”), “ticketID”, etc. and constructs the GET request to invoke the “iAM Smart” API by Online Service App;

Step 7. Online Service App polls Online Service Server for the digital signing result from “iAM Smart” System;

Step 8. Online Service App invokes in-app browser (Safari for iOS, Chrome for Android) to submit the GET request;

Step 9-10. Browser opens the GET request to invoke the "iAM Smart" API and QR Code page will be shown;

"iAM Smart" API (GET: Request QR page)

Step 11. QR Code page of "iAM Smart" System polls "iAM Smart" System for the QR Code status;

Step 12. "iAM Smart" user logs in "iAM Smart" Mobile App to scan the QR Code;

Step 13-15. "iAM Smart" Mobile App requests and displays 4-digit identification code from "iAM Smart" System. "iAM Smart" user reviews the digital signing authorisation request (e.g., continue or reject the request) and verifies the 4-digit identification code with Online Service Website;

Step 16-17. "iAM Smart" user follows the instructions in "iAM Smart" Mobile App to complete digital signing operation;

Step 18-19. "iAM Smart" System verifies the validity of QR code and other necessary information, and return the authorisation result to "iAM Smart" Mobile App;

Step 20-22. QR Code page of "iAM Smart" System receives the polling result returned by "iAM Smart" System (i.e., response for the polling in Step 11) which includes authCode and businessID or any error code (e.g., "iAM Smart" user rejects the request), assembles and invokes the Online Service callback API given in "redirectURI" using browser redirection;

Online Service Callback API (GET: Callback with authCode to Online Service Server)

Step 23-24. When Online Service Server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e., openID) of the "iAM Smart" user. API data encryption is required;

"iAM Smart" API (POST: Request accessToken & Tokenised ID)

Step 25-26. Online Service Server invokes the "iAM Smart" API with the accessToken and openID to obtain the result of anonymous digital signing. API data encryption is required;

"iAM Smart" API (POST: Obtain Anonymous Digital Signing Result)

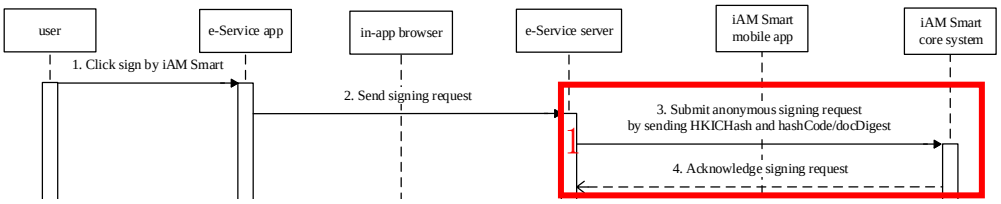
"iAM Smart" API (POST: Obtain Anonymous PDF Digital Signing Result)

Step 27. Online Service Server completes the document digital signing process, verify the result and acknowledges "iAM Smart" System the digital signing result using "iAM Smart" API with the same "businessID" of the digital signing request. API data encryption is required;

"iAM Smart" API (POST: Online Service Acknowledges Digital Signing Result)

Step 28. Online Service Server matches the result using the "businessID" and shows the digital signing result in corresponding Online Service App (i.e., response for the polling in Step 7).

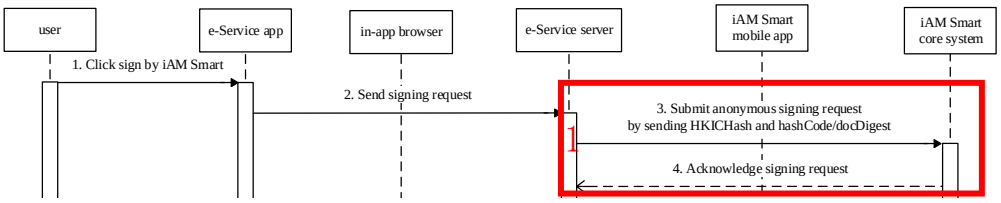
3.8.3.1 Implementing (1) POST: Request Anonymous Digital Signing



Please refer to Section 3.8.1.1 except:

- Online Service should determine the "iAM Smart" Mobile App is not in the same device of Online Service App using program code.

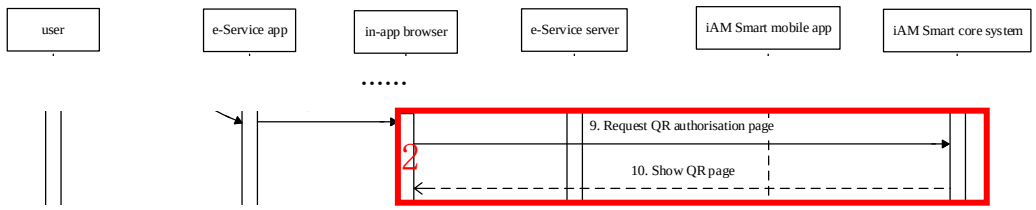
3.8.3.2 Implementing (1) POST: Request Anonymous PDF Digital Signing



Please refer to Section 3.8.1.2 except:

- Online Service should determine the "iAM Smart" Mobile App is not in the same device of Online Service App using program code.

3.8.3.3 Implementing (2) GET: Request QR Page



Pre-conditions

- Please refer to Section 3.4.3.1 except:
  - Online Service has the "ticketID" provided by "iAM Smart" System for the anonymous request in step 4.

Post-conditions

- Please refer to Section 3.4.3.1.

Error conditions

- Please refer to Section 3.4.3.1.

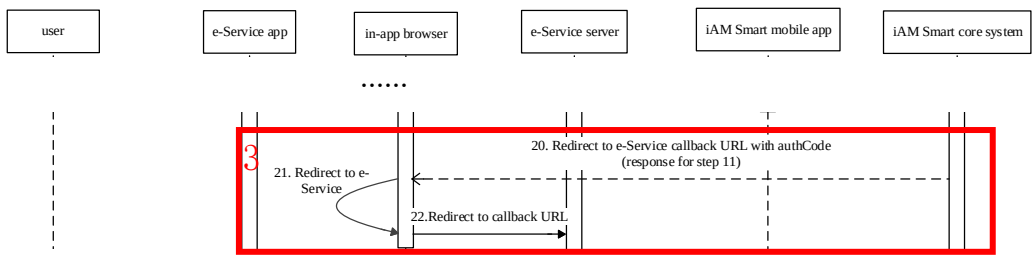
Request Parameters

- Please refer to "iAM Smart" API Specification.

Notes

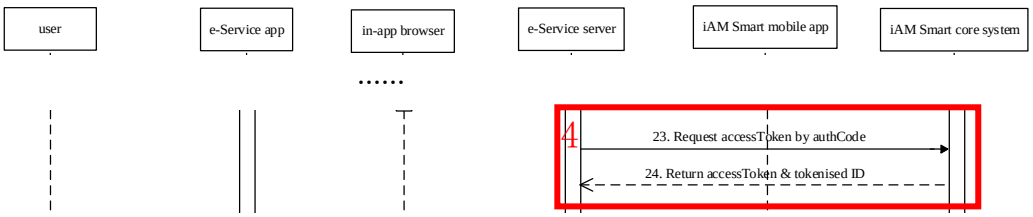
- Please refer to Section 3.4.3.1 except the following parameter:
  - Request parameter "ticketID": Value provided by "iAM Smart" System for the anonymous request.

3.8.3.4 Implementing (3) GET: Callback with authCode to Online Service Server



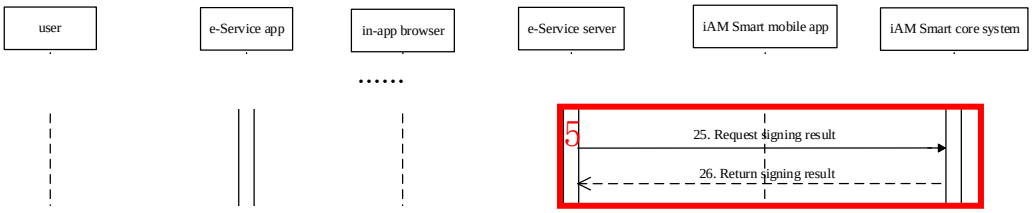
Please refer to Section 3.8.1.4.

3.8.3.5 Implementing (4) POST: Request accessToken & Tokenised ID



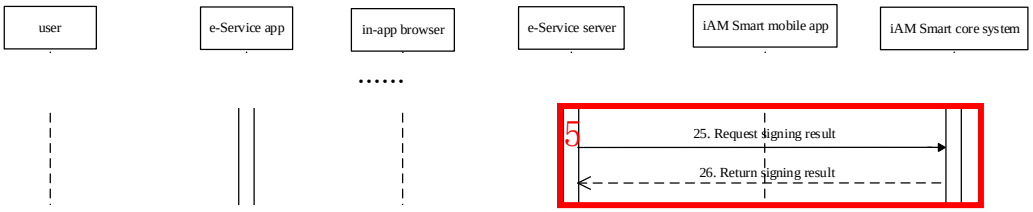
Please refer to Section 3.6.1.4.

3.8.3.6 Implementing (5) POST: Obtain Anonymous Digital Signing Result



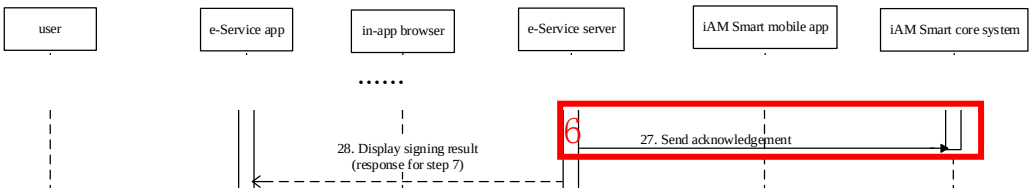
Please refer to Section 3.8.1.6.

3.8.3.7 Implementing (5) POST: Obtain Anonymous PDF Digital Signing Result



Please refer to Section 3.8.1.7.

3.8.3.8 Implementing (6) POST: Online Service Acknowledges Digital Signing Result



Please refer to Section 3.7.1.5.

3.8.4 Scenario 4: Anonymous Digital Signing (Online Service App in Same Device)

The sequence diagram below shows how an anonymous user authorises and signs the Document Hash when Online Service App and the "iAM Smart" Mobile App are running in the same device.

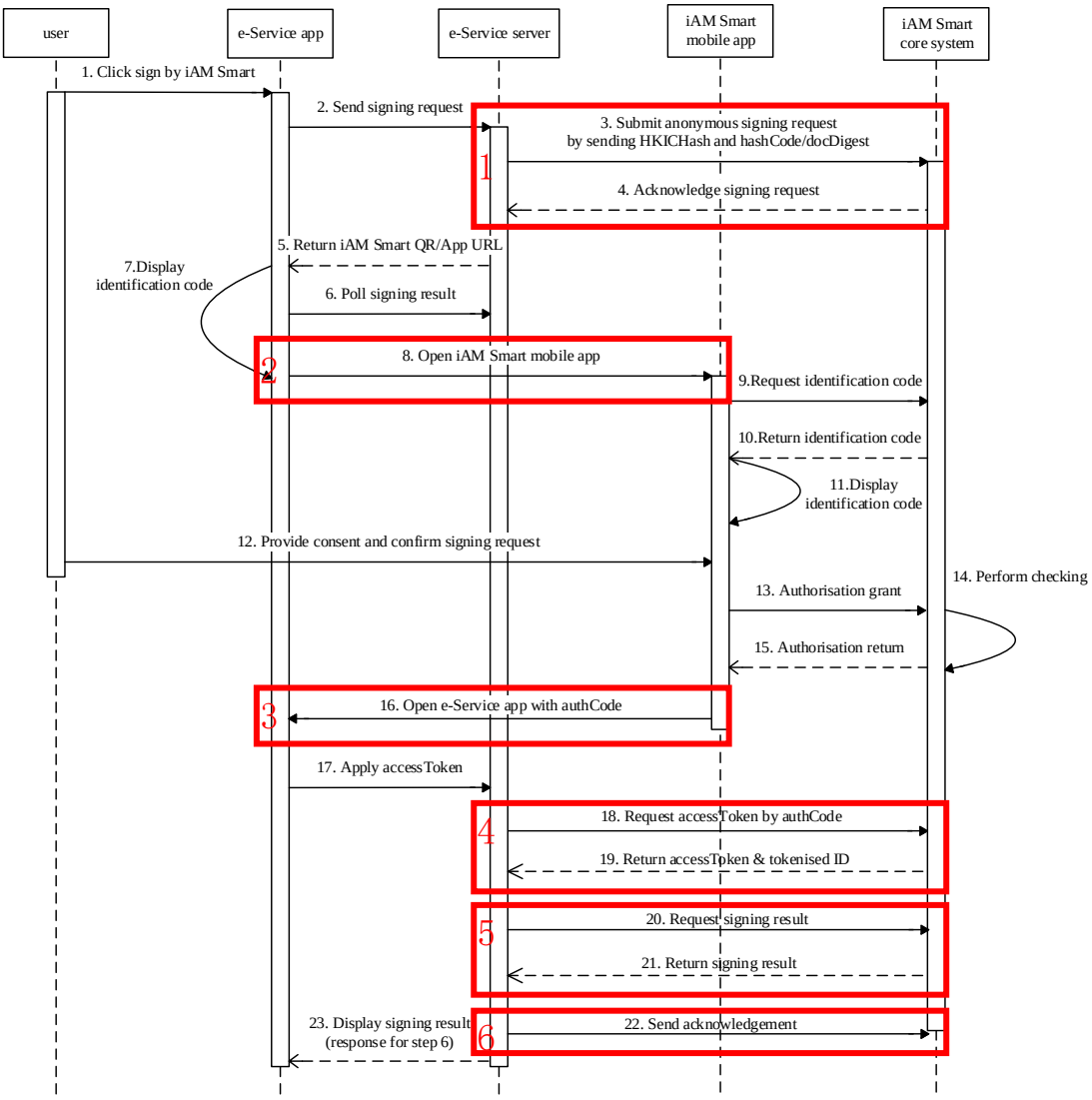


Figure-18 Anonymous Digital Signing (Online Service App in Same Device)

Step 1. After entering HKIC number, the user clicks the "Anonymous hash digital signing" (or "Anonymous pdf digital signing") button in Online Service App;

Step 2. Online Service App determines “iAM Smart” Mobile App is installed in the device using program code, initiates anonymous digital signing request to Online Service Server with “App_Scheme” or “App_Link” (use as value for request parameter “source”);

Step 3. Online Service Server initiates an anonymous digital signing request to invoke the “iAM Smart” API with the “businessID” of this request, Document Hash / PDF digest, HKICHash, signature algorithm (only for "*Request Anonymous Digital Signing*"), etc. API data encryption is required;

“iAM Smart” API (POST: Request Anonymous Digital Signing)

“iAM Smart” API (POST: Request Anonymous PDF Digital Signing)

Step 4. “iAM Smart” System verifies and confirms receipt of the anonymous digital signing request to Online Service Server by returning POST response with parameter “ticketID”;

Step 5&7. Online Service Server uses the Document Hash / PDF digest, HKICHash and hash of clientID to calculate a 4-digit identification code and instructs Online Service App to display the instructions with the 4-digit identification code to inform “iAM Smart” user to process the digital signing authorisation request in “iAM Smart” Mobile App;

Online Service Server prepares necessary parameters such as “clientID”, “redirectURI” (set as Online Service App URL Scheme or Universal Link/App Link depending on “source” parameter), “scope”, “source” (set as “App_Scheme” or “App_Link”), “ticketID”, etc. and constructs the URL Scheme to invoke “iAM Smart” Mobile App by Online Service App;

Step 6. Online Service App polls Online Service Server for the digital signing result from “iAM Smart” System;

Step 8. Online Service App invokes “iAM Smart” Mobile App using URL Scheme with request parameters (set <Context> as “anon_hash-sign” or “anon_pdf-sign” in URL Scheme);

“iAM Smart” API (URL Scheme: Open “iAM Smart” Mobile App for Getting Context)

Step 9-11. "iAM Smart" user logs in "iAM Smart" Mobile App, "iAM Smart" Mobile App requests and displays 4-digit identification code from "iAM Smart" System. "iAM Smart" user reviews the digital signing authorisation request (e.g., continue or reject the request) and verifies the 4-digit identification code with Online Service Website;

Step 12-13. "iAM Smart" user follows the instructions in "iAM Smart" Mobile App to complete digital signing operation;

Step 14-15. "iAM Smart" System verifies the validity of necessary information, and return the authorisation result to "iAM Smart" Mobile App.

Step 16. "iAM Smart" Mobile App invokes Online Service App using URL Scheme or Universal Link/App Link with required parameters such as "authCode", "businessID";

Online Service Callback API (Callback with authCode to Online Service App)

Step 17. Online Service App sends authCode, businessID, etc. to Online Service Server;

Step 18-19. When Online Service Server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e., openID) of the "iAM Smart" user. API data encryption is required;

"iAM Smart" API (POST: Request accessToken & Tokenised ID)

Step 20-21. Online Service Server invokes the "iAM Smart" API with the accessToken and openID to obtain the result of anonymous digital signing. API data encryption is required;

"iAM Smart" API (POST: Obtain Anonymous Digital Signing Result)

"iAM Smart" API (POST: Obtain Anonymous PDF Digital Signing Result)

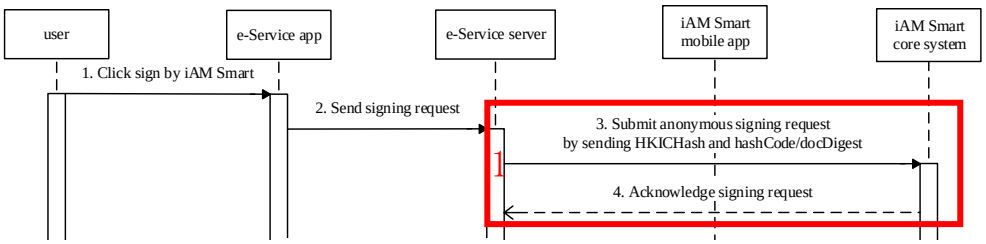
Step 22. Online Service Server completes the document digital signing process, verify the result and acknowledges "iAM Smart" System the digital signing result using "iAM Smart" API with the same

"businessID" of the digital signing request. API data encryption is required;

"iAM Smart" API (POST: Online Service Acknowledges Digital Signing Result)

Step 23. Online Service Server matches the result using the "businessID" and shows the digital signing result in corresponding Online Service App (i.e., response for the polling in Step 6).

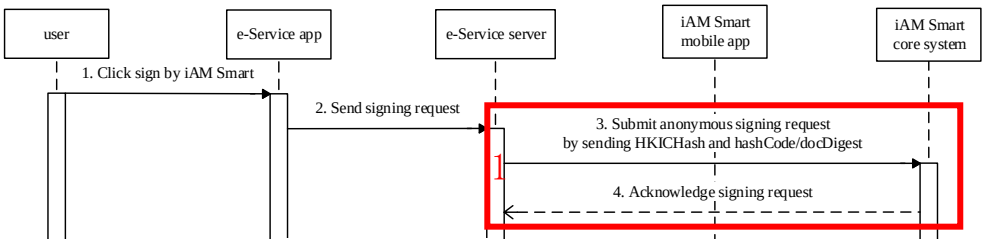
3.8.4.1 Implementing (1) POST: Request Anonymous Digital Signing



Please refer to Section 3.8.1.1 except:

- Online Service should determine the "iAM Smart" Mobile App is on the same device of Online Service App using program code.

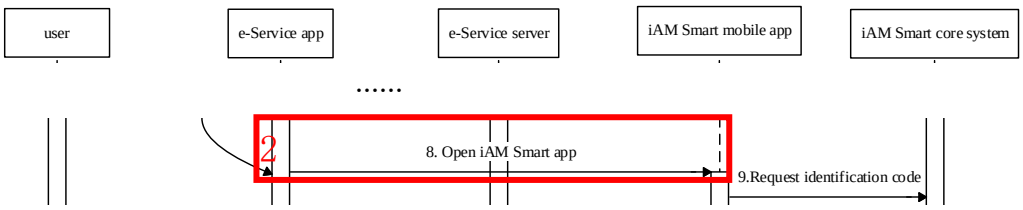
3.8.4.2 Implementing (1) POST: Request Anonymous PDF Digital Signing



Please refer to Section 3.8.1.2 except:

- Online Service should determine the "iAM Smart" Mobile App is on the same device of Online Service App using program code.

3.8.4.3 Implementing (2) URL Scheme: Open "iAM Smart" Mobile App for Getting Context



Pre-conditions

- Online Service Website and "iAM Smart" Mobile App are in the same device.
- Online Service has the "ticketID" provided by "iAM Smart" System for the digital signing request in step 4.
- Online Service should determine if it would generate the optional "state" request parameter to prevent CSRF attack. The "state" will be returned to Online Service for checking through the Online Service callback API for receiving the authorisation code from "iAM Smart" System in Step 16.

Post-conditions

- "iAM Smart" Mobile App will be launched.
- Online Service Website should keep synchronising with Online Service server for the callback response of the digital signing request from "iAM Smart" System.

Error conditions

- Nil

Request Parameters

- Please refer to "iAM Smart" API Specification.

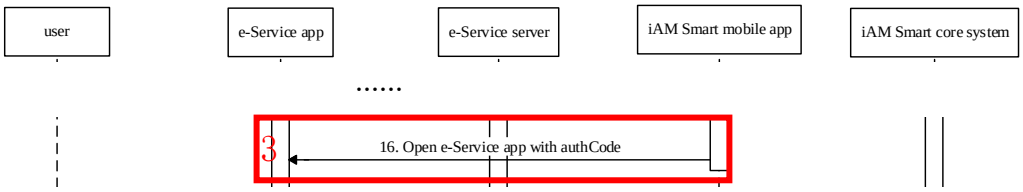
Notes

- Request parameter "ticketID": Value provided by "iAM Smart" System for the anonymous request.
- Request parameter "source": Value should be "App_Scheme" (for Online Service App URL scheme), or "App_Link" (for the Universal Link/App Link).
- Request parameter "redirectURI": This is Online Service App URL scheme or Universal Link/App Link depending on the "source" parameter, which is used for receiving the authorisation code. The value should be URL encoded. For details, please refer to Section 7.5.4 of "iAM Smart" API

Specification. The value must be the same as provided during Online Service registration.

- Request parameter “state”: The value should be URL encoded.
- Set <Context> as “anon_hash-sign” or “anon_pdf-sign” in URL Scheme.

3.8.4.4 Implementing (3) Callback with authCode to Online Service App



Pre-conditions

- Please refer to Section 3.4.2.2.

Post-conditions

- Please refer to Section 3.4.2.2 except:
  - Online Service Server should match the callback result with corresponding Online Service App using the “businessID”.

Error conditions

- Please refer to Section 3.8.1.4 except:
  - This Online Service callback API is a URL Scheme, or Universal Link/App Link.

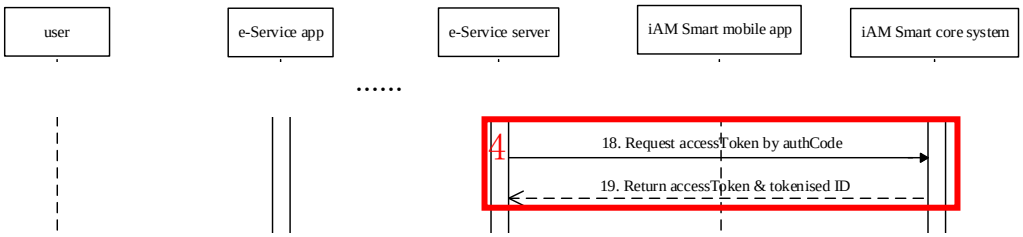
Callback Parameters

- Please refer to “iAM Smart” API Specification.

Notes

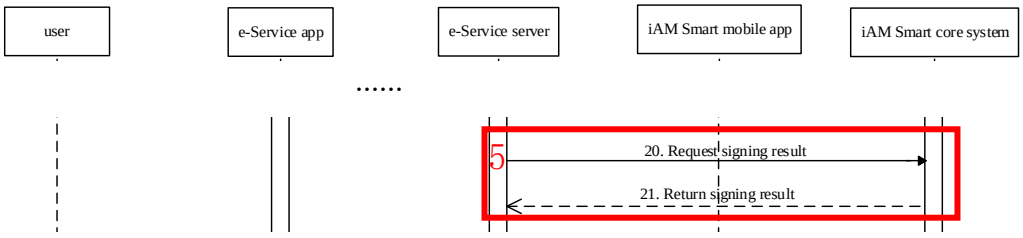
- Please refer to Section 3.4.2.2 except the following parameter:
  - Callback parameter “businessID”: Value returned by “iAM Smart” System should be the same one as Online Service submitted in the anonymous request.

3.8.4.5 Implementing (4) POST: Request accessToken & Tokenised ID



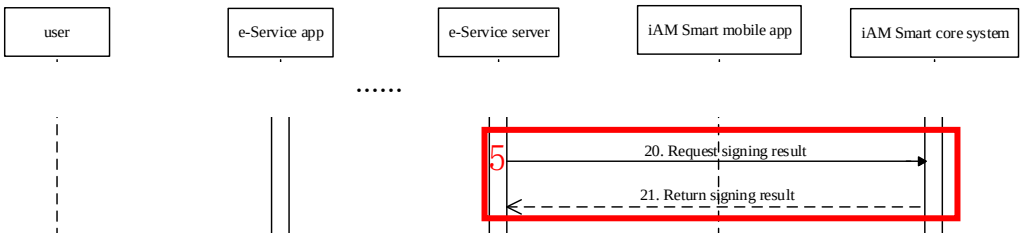
Please refer to Section 3.6.1.4.

3.8.4.6 Implementing (5) POST: Obtain Anonymous Digital Signing Result



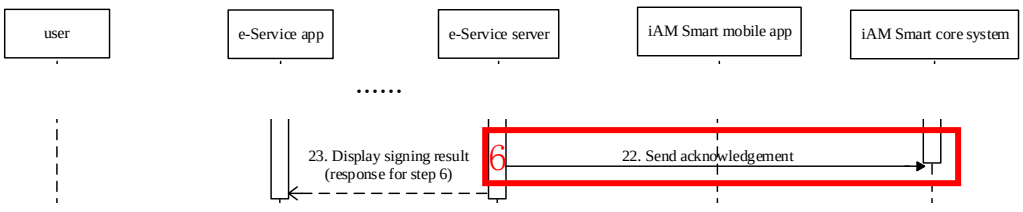
Please refer to Section 3.8.1.6.

3.8.4.7 Implementing (5) POST: Obtain Anonymous PDF Digital Signing Result



Please refer to Section 3.8.1.7.

3.8.4.8 Implementing (6) POST: Online Service Acknowledges Digital Signing Result



Please refer to Section 3.7.1.5.

3.9 WORKFLOWS FOR RE-AUTHENTICATION WITH SERVICE LOGIN

3.9.1 Scenario 1: Re-authentication (Online Service Website/App in Different Device)

The sequence diagram below shows how Online Service performs "iAM Smart" Re-authentication when Online Service and the "iAM Smart" Mobile App are running in different devices.

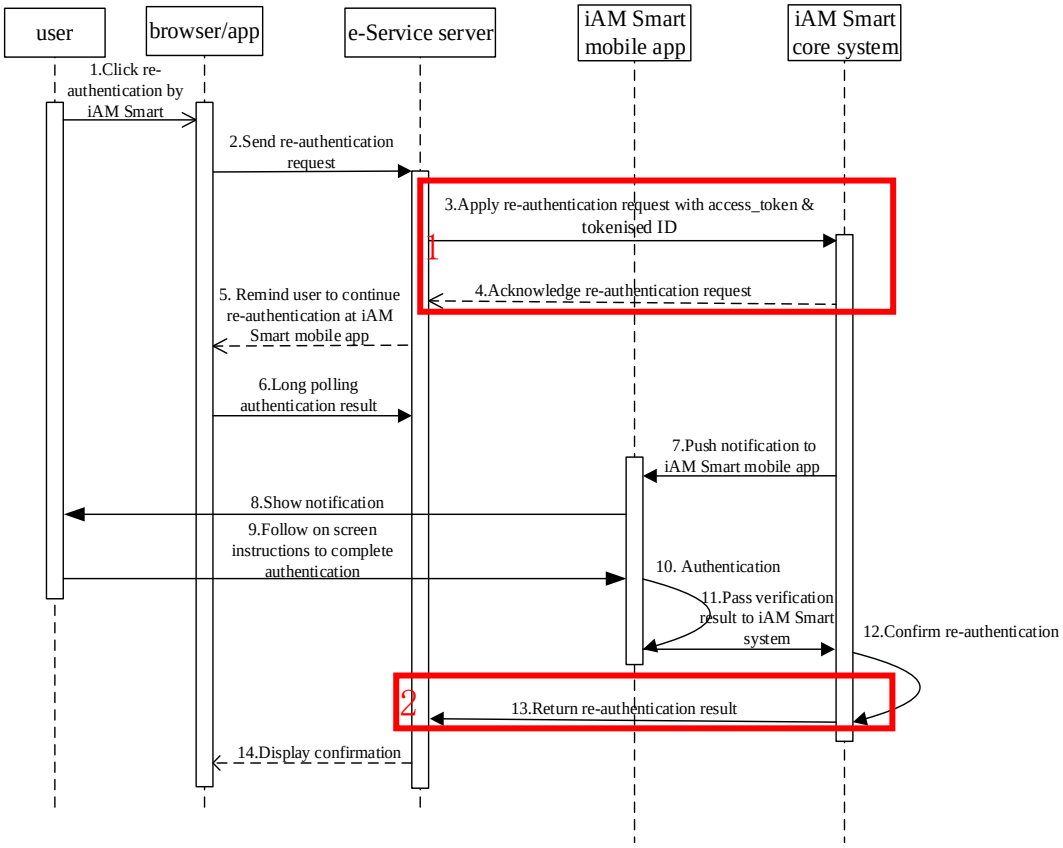


Figure-19 Re-authentication (Online Service Website/App in Different Device)

Step 1. User clicks the "Re-authentication by iAM Smart" button in Online Service Website/App;

Step 2. Online Service Website/App initiates Re-authentication request to Online Service Server with browser's user agent value (for Online Service Website) or "App_Scheme"/"App_Link" (for Online Service App) in this scenario (use as the value of request parameter "source");

Step 3. Online Service Server initiates Re-authentication request to invoke "iAM Smart" API with the "businessID" of this request, accessToken, Tokenised ID, etc;

"iAM Smart" API (POST: Request Re-authentication)

Step 4. "iAM Smart" System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the Re-authentication request to Online Service server by returning POST response with parameter "authByQR" (sets to "true") in this scenario;

Step 5. Online Service Website/App shows instruction to inform "iAM Smart" user to process the Re-authentication authorisation request in "iAM Smart" Mobile App;

Step 6. Online Service Website/App should keep synchronising with Online Service Server for request result (e.g., polling);

Step 7. "iAM Smart" System pushes a notification message to "iAM Smart" Mobile App based on the Tokenised ID;

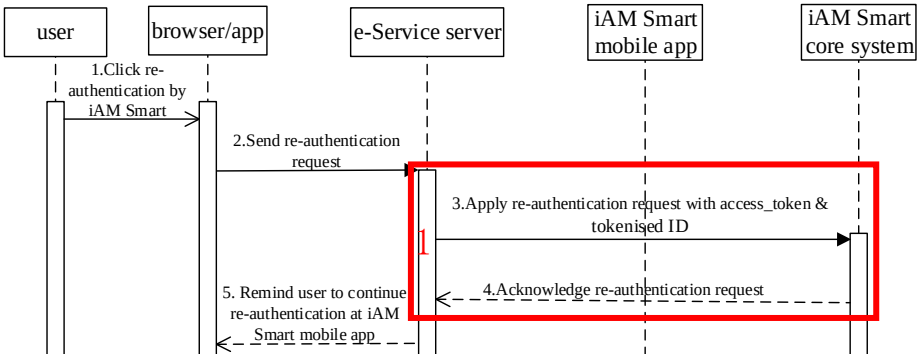
Step 8-12. "iAM Smart" user logs in "iAM Smart" Mobile App, reviews the Re-authentication authorisation request (e.g., continue or reject the request) and authorises the request;

Step 13. "iAM Smart" System invokes Online Service callback API to return the result with the "businessID" of the Re-authentication request. API decryption is required;

Online Service Callback API (POST: Callback to Receive Re-authentication Result)

Step 14. Online Service Server processes and matches the result using the "businessID" and shows the result in corresponding Online Service Website/App.

3.9.1.1 Implementing (1) POST: Request Re-authentication



Pre-conditions

- Online Service must possess a valid accessToken of the "iAM Smart" user with authorisation scope "Re-authentication authorisation".
- Online Service Website/App and "iAM Smart" Mobile App are in different devices in this scenario.
- Online Service generates a "businessID" for this Re-authentication request and use this identifier to match the callback result returned from "iAM Smart" System.
- API data encryption is required.

Post-conditions

- Online Service server should check the response parameter "authByQR" and "ticketID" and determine the next action. For details please refer to Section 3.2.1. "ticketID" will not be provided by "iAM Smart" System in this scenario.
- Online Service Website/App should show instructions to inform "iAM Smart" user to process the Re-authentication request in "iAM Smart" Mobile App when "authByQR" is "true".
- Online Service server should keep synchronising with Online Service Website/App for the result returned from "iAM Smart" System.
- API data decryption is required.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action

code - D80002	Failed to request re-authentication	Inform user the “iAM Smart” re-authentication request is failed and retry later

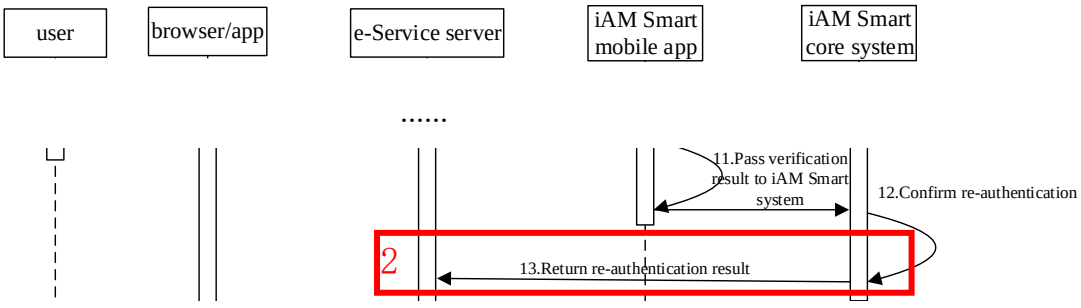
Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- Request parameter “source”: Value should be matched with Online Service client terminal (e.g., “App_Scheme”/“App_Link” or browser’s user agent value).
- Request Parameter - redirectURI: Value should equal to URI of “Callback to Receive Re-authentication Result” Online Service callback API. It has to be the same value as provided during Online Service registration.
- Response parameter “authByQR”: The value returned by “iAM Smart” System will be “true” in this scenario.
- “iAM Smart” System will not return the response parameter “ticketID” in this scenario.

3.9.1.2 Implementing (2) POST: Callback to Receive Re-authentication Result



Pre-conditions

- “iAM Smart” user can accept and complete the Re-authentication request.
- “iAM Smart” user can also reject the Re-authentication request.

Post-conditions

- API data decryption is required.
- Online Service server should match the callback result with corresponding Online Service Website/App using the “businessID”.

Error conditions

For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D80001	User rejected re-authentication request	Inform user the “iAM Smart” re-authentication request is rejected
code - D80002	Failed to request re-authentication	Inform user the “iAM Smart” re-authentication request is failed and retry later
code - D80003	Re-authentication request timeout	Inform user the “iAM Smart” re-authentication request is timeout and provide way for user to retry

Callback Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- Callback parameter “isPassed”: It is the Re-authentication result. If it is the same “iAM Smart” user of the Tokenised ID, its value is “true”. Otherwise, its value is “false”.

3.9.2 Scenario 2: Re-authentication (Online Service Website in Same Device)

The sequence diagram below shows how Online Service performs "iAM Smart" Re-authentication when Online Service website and the "iAM Smart" Mobile App are running in the same device.

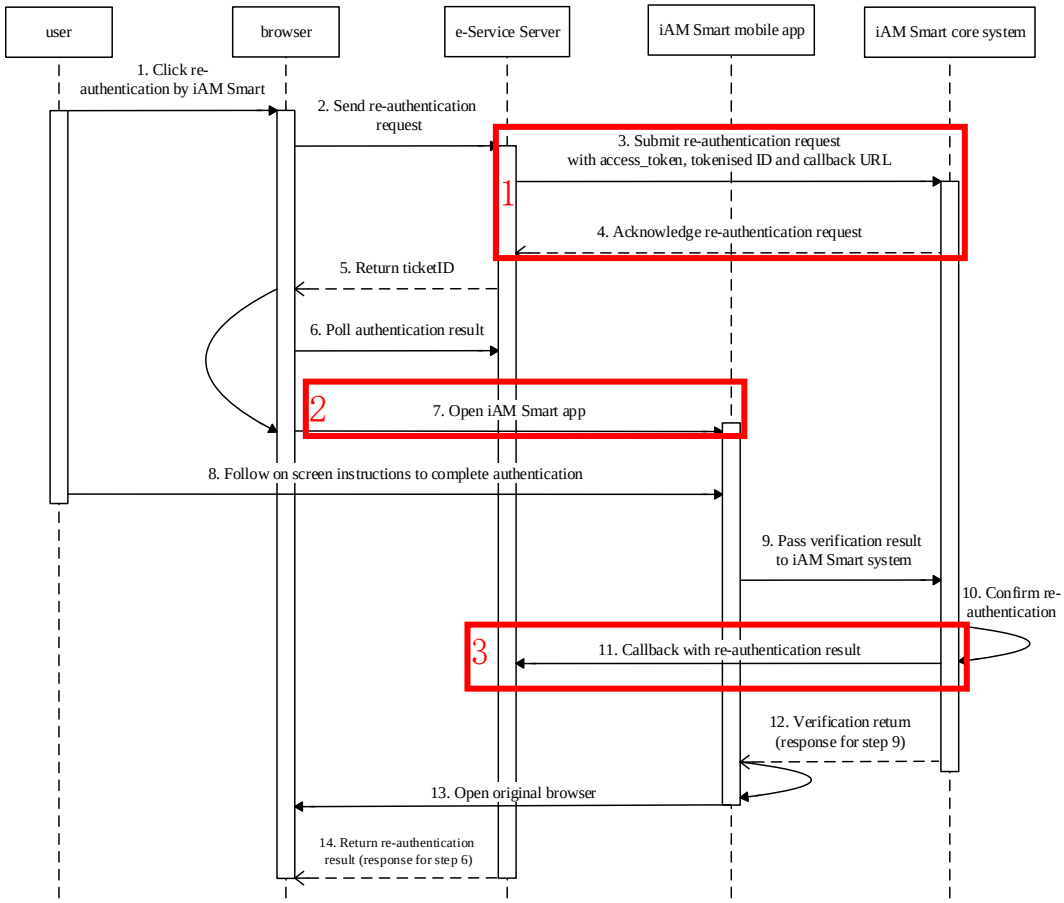


Figure-20 Re-authentication (Online Service Website in Same Device)

Step 1. User clicks the "Re-authentication by iAM Smart" button in Online Service Website;

Step 2. Online Service Website initiates Re-authentication request to Online Service Server with browser's user agent value (for use as value for request parameter "source");

Step 3. Online Service Server initiates Re-authentication request to invoke "iAM Smart" API with the "businessID" of this request, accessToken, Tokenised ID, etc. The request parameter "source" will be the browser's user agent value. API data encryption is required;

"iAM Smart" API (POST: Request Re-authentication)

Step 4. "iAM Smart" System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the Re-authentication request to Online Service server by returning POST response with parameter "authByQR" (set to "false") and "ticketID" in this scenario;

Step 5-6. Online Service Server prepares and sends necessary parameters such as "authByQR", "ticketID", etc. to Online Service Website. Online Service Website should keep synchronising with the Online Service server for the request result (e.g., polling);

Step 7. Online Service Website invokes "iAM Smart" Mobile App using URL Scheme with "ticketID" (set <Context> as "re-auth" in URL Scheme). Other parameters of this API are not used in this scenario;

"iAM Smart" API (URL Scheme: Open "iAM Smart" Mobile App for Getting Context)

Step 8-10. "iAM Smart" user logs in "iAM Smart" Mobile App, reviews the Re-authentication authorisation request (e.g., continue or reject the request) and authorises the request;

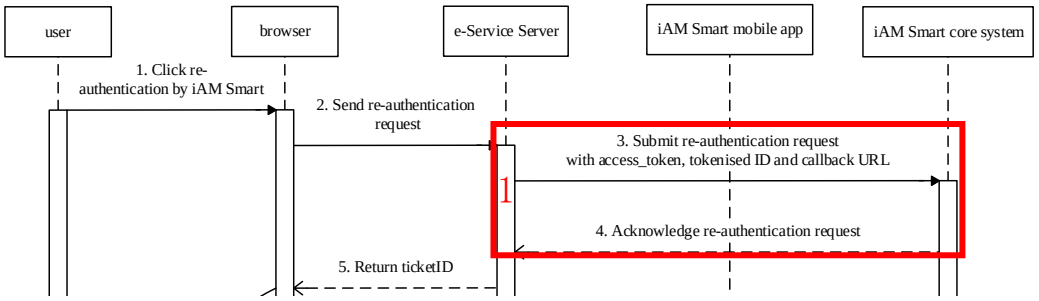
Step 11. "iAM Smart" System invokes Online Service callback API to return the result with "businessID" of the Re-authentication request to Online Service server. API data decryption is required;

Online Service Callback API (POST: Callback to Receive Re-authentication Result)

Step 12-13. "iAM Smart" System instructs "iAM Smart" Mobile App to invoke the original Online Service browser (using the browser's user agent name submitted in the Re-authentication request in Step 3);

Step 14. Online Service Server processes and matches the result using the "businessID" and shows the result in the corresponding Online Service Website.

3.9.2.1 Implementing (1) POST: Request Re-authentication



Pre-conditions

- Please refer to Section 3.9.1.1 except:
  - Online Service Website and “iAM Smart” Mobile App are in the same device in this scenario.

Post-conditions

- Please refer to Section 3.9.1.1 except:
  - Response “authByQR” is “false”, “ticketID” will be provided by “iAM Smart” System in this scenario.

Error conditions

- Please refer to Section 3.9.1.1.

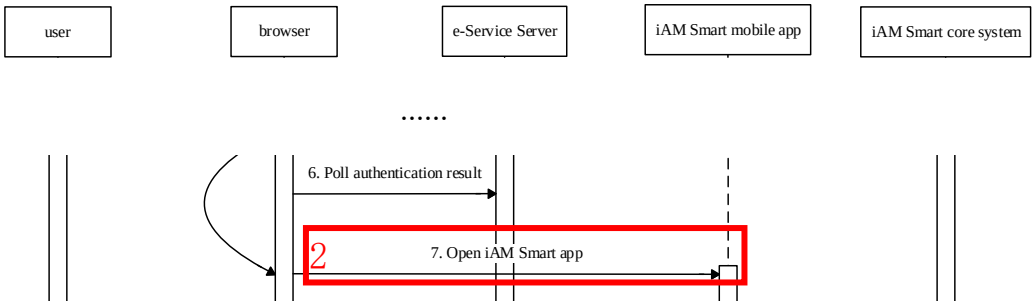
Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Please refer to Section 3.9.1.1, except for the following parameters:
  - Request parameter “source”: Value should be the browser's user agent value.
  - Response parameter “authByQR”: The value is “false” in this scenario.
  - Response parameter “ticketID”: It will be provided by “iAM Smart” System in this scenario. It is a 36-byte (or less) UUID number (ASCII character set).

3.9.2.2 Implementing (2) URL Scheme: Open “iAM Smart” Mobile App for Getting Context



Pre-conditions

- Online Service Website and “iAM Smart” Mobile App are in the same device.
- Online Service has the “ticketID” provided by “iAM Smart” System for the Re-authentication request;
- Other parameters of this API are not used in this scenario.

Post-conditions

- “iAM Smart” Mobile App will be launched.
- Online Service Website should keep synchronising with Online Service server for the callback response of the Re-authentication request from “iAM Smart” System.

Error conditions

- Nil

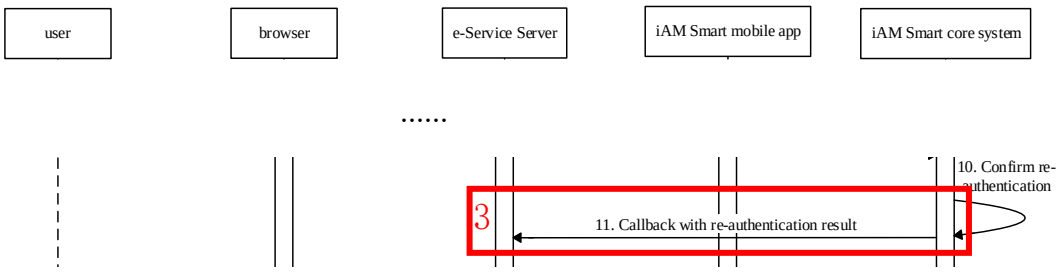
Request Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Set <Context> as “re-auth” in URL Scheme.

3.9.2.3 Implementing (3) POST: Callback to Receive Re-authentication Result



Please refer to Section 3.9.1.2.

3.9.3 Scenario 3: Re-authentication (Online Service App in Same Device)

The sequence diagram below shows how Online Service performs "iAM Smart" Re-authentication when Online Service mobile application and the "iAM Smart" Mobile App are running the same device.

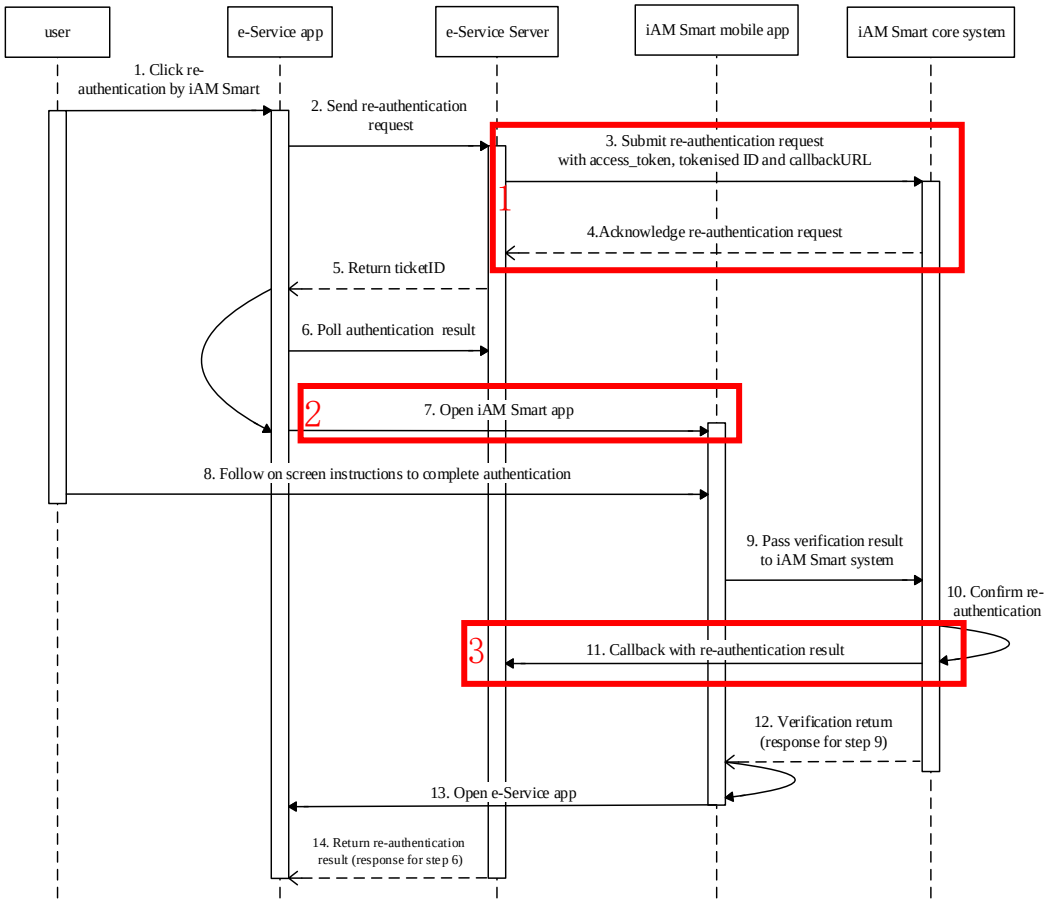


Figure-21 Re-authentication (Online Service App in Same Device)

Step 1. User clicks the "Re-authentication by iAM Smart" button in Online Service App;

Step 2. Online Service App initiates Re-authentication request to Online Service Server with "App_Scheme" or "App_Link" in this scenario (for use as value for request parameter "source");

Step 3. Online Service Server initiates Re-authentication request to invoke "iAM Smart" API with the "businessID" of this request, accessToken, Tokenised ID etc. The request parameter "source" will be "App_Scheme" or "App_Link". API data encryption is required;

"iAM Smart" API (POST: Request Re-authentication)

Step 4. "iAM Smart" System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the Re-authentication request to Online Service server by returning POST response with parameter "authByQR" (set to "false") and "ticketID" in this scenario;

Step 5. Online Service Server prepares and sends the necessary parameters such as "authByQR", "ticketID", etc. to Online Service App. Online Service App should keep synchronising with the Online Service server for the request result (e.g., polling);

Step 7. Online Service App invokes "iAM Smart" Mobile App using URL Scheme with "ticketID" (set <Context> as "re-auth" in URL Scheme). Other parameters of this API are not used in this scenario;

"iAM Smart" API (URL Scheme: Open "iAM Smart" Mobile App for Getting Context)

Step 8-10. "iAM Smart" user logs in "iAM Smart" Mobile App, reviews the Re-authentication authorisation request (e.g., continue or reject the request) and authorises the request;

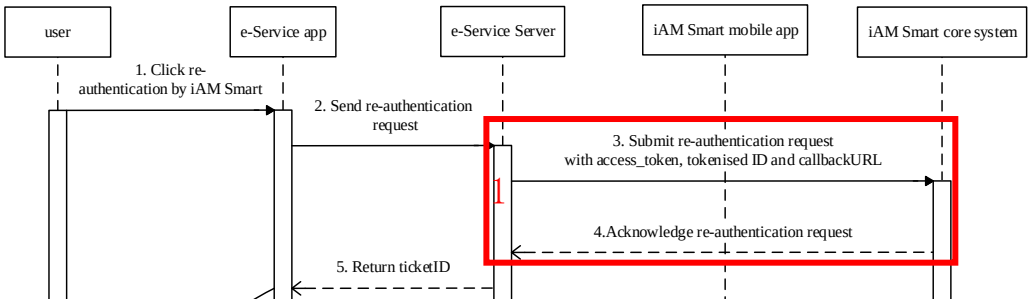
Step 11. "iAM Smart" System invokes Online Service callback API to return the result with "businessID" of the Re-authentication request to Online Service server. API data decryption is required;

Online Service Callback API (POST: Callback to Receive Re-authentication Result)

Step 12-13. "iAM Smart" System instructs "iAM Smart" Mobile App to invoke the Online Service App using URL Scheme, or Universal Link/App Link ("iAM Smart" System queries the information registered at Online Service registration depending on the "source" submitted in Step 3);

Step 14. Online Service Server processes and matches the result using the "businessID" and shows the result in the corresponding Online Service App.

3.9.3.1 Implementing (1) POST: Request Re-authentication



Pre-conditions

- Please refer to Section 3.9.2.1 except:
  - Online Service App and "iAM Smart" Mobile App are in the same device in this scenario.

Post-conditions

- Please refer to Section 3.9.2.1.

Error conditions

- Please refer to Section 3.9.2.1.

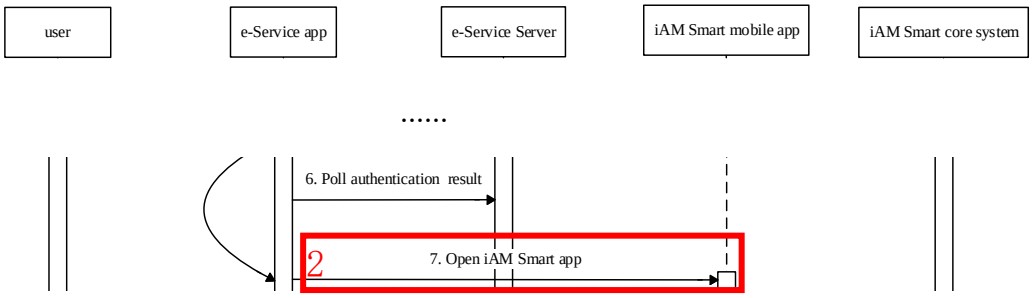
Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

Notes

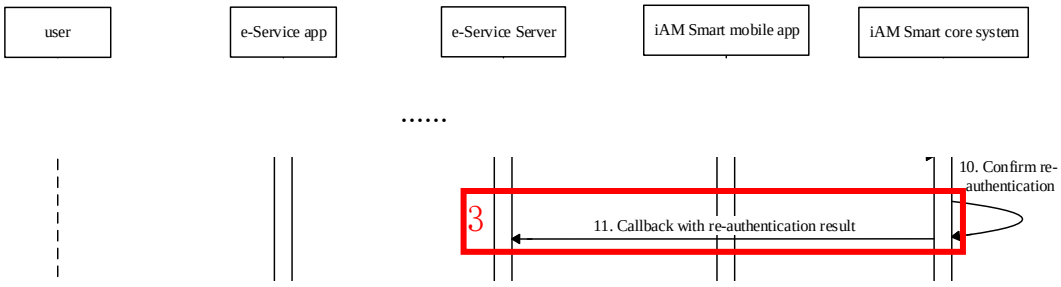
- Please refer to Section 3.9.2.1, except for the following parameter:
  - Request parameter "source": Value should be "App_Scheme" (for the Online Service App URL scheme), or "App_Link" (for the Universal Link/App Link).

3.9.3.2 Implementing (2) URL Scheme: Open "iAM Smart" Mobile App for Getting Context



Please refer to Section 3.9.2.2.

3.9.3.3 Implementing (3) POST: Callback to Receive Re-authentication Result



Please refer to Section 3.9.1.2.

3.10 WORKFLOWS FOR DIRECT LOGIN & DIRECT ACCESS

3.10.1 Scenario 1: Direct Login with Online Service Website (aka Direct Login v2)

The sequence diagram below shows how users initiate Direct Login in that "iAM Smart" Mobile App and then automatically login Online Service website opened in a supported mobile browser.

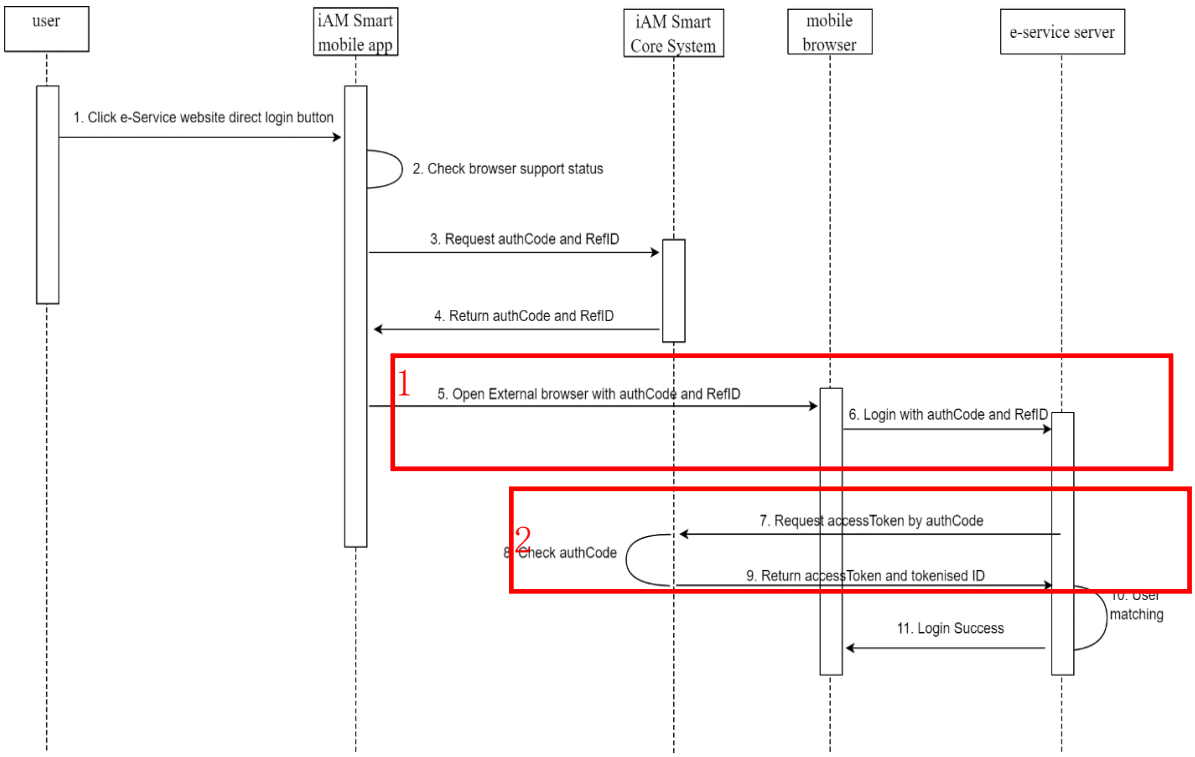


Figure-22 Direct login v2 from "iAM Smart" to Online Service

Step 1. User opens "iAM Smart" Mobile App, browses Online Service catalogue or views the detailed information of Online Service, and clicks the button to invoke the Online Service;

Step 2. "iAM Smart" Mobile App checks if the supported mobile browser and its browser version[4] is installed. If there is no supported

[4] "iAM Smart" will maintain the and review the supported browser list from time to time. Currently the browser priority list for Android is Chrome (highest ranking), Firefox, Samsung

browser found, the default browser in “iAM Smart” mobile device will be opened and redirected to the fallback URL that has been registered in the “iAM Smart” Platform. Please refer to Section 3.10.3 for the fallback details;

Step 3-4. “iAM Smart” Mobile App request “iAM Smart” server for the authCode and RefID;

Step 5-6. “iAM Smart” Mobile App opens the external browser, and requests the direct login callback endpoint with the authCode and RefID according to the current UI display language. Online Service shall provide three direct login endpoints with three different languages (EN/TC/SC) and register in the “iAM Smart” Platform in advance. If there is any exception happened, “iAM Smart” Mobile App will open the fallback URL without the authCode and RefID instead of the direct login URL. In addition, the three fallback URLs of Online Service for three languages (EN/TC/SC) should have been registered so that the “iAM Smart” Mobile App can decide which URL is for fallback with respect to the current UI display language. Online Service should check if any login session exists and perform session management properly according to the business needs;

Online Service Callback API (GET: Callback with AuthCode to Online Service Server (Direct Login V2))

Step 7-9. When the Online Service Server gets the authCode, it should invoke the “iAM Smart” API to obtain the accessToken and the tokenised ID (i.e., openID) of the “iAM Smart” user for the selected Online Service. API data encryption is required;

“iAM Smart” API (POST: Request accessToken & tokenised ID)

Step 10. Online Service server then performs user matching using the tokenised ID with its user repository and determines the result of this login request.

Step 11. The Online Service Website shows the successful login page accordingly.

---

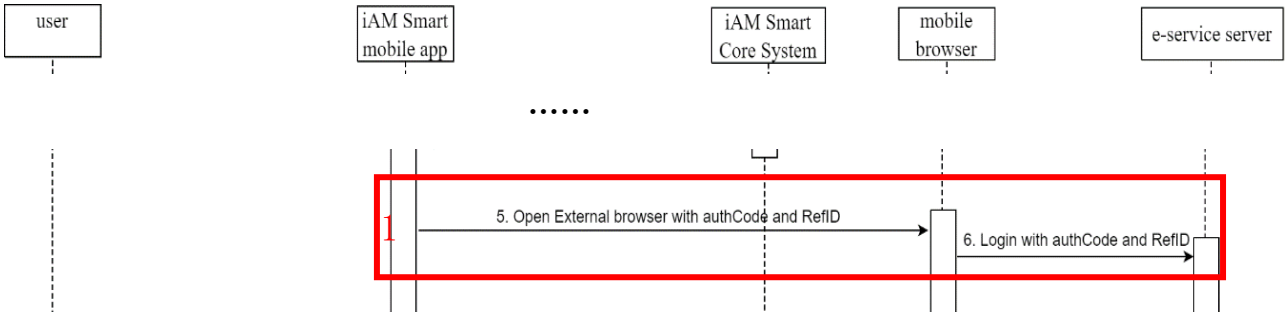
browser, Huawei browser, Xiaomi browser and Edge browser (lowest ranking). iOS in addition supports Safari.

Notes:

Online Services shall provide the following URLs to DPO for setup in the "iAM Smart" Platform:

No.	URL Type	URL Description	Described in
1.	Direct Login (English)	Receive authorisation code of web direct login with English language	Step 5-6
2.	Direct Login (Traditional Chinese)	Receive authorisation code of web direct login with Traditional Chinese language	Step 5-6
3.	Direct Login (Simplified Chinese)	Receive authorisation code of web direct login with Simplified Chinese language	Step 5-6
4.	Fallback / Direct Access (English)	Provide fallback for web direct login with English language when there is no supported browser, i.e., Online Service login page; or provide direct access to the Online Service in English without login requirement	Step 2 Step 5-6
5.	Fallback / Direct Access (Traditional Chinese)	Provide fallback for web direct login with Traditional Chinese language when there is no supported browser, i.e., Online Service login page; or provide direct access to the Online Service in Traditional Chinese without login requirement	Step 2 Step 5-6
6.	Fallback / Direct Access (Simplified Chinese)	Provide fallback for web direct login with Simplified Chinese language when there is no supported browser, i.e., Online Service login page; or provide direct access to the Online Service in Simplified Chinese without login requirement	Step 2 Step 5-6

3.10.1.1 Implementing (1) GET: An Online Service endpoint to receive authCode & RefID from external browser



This endpoint is to be implemented by Online Service. The implementation reuses "Callback with authCode to Online Service Server" (Section 6.4.2 of "iAM Smart" API Specification).

Pre-conditions

- "iAM Smart" user has initiated Online Service website via direct login in "iAM Smart" Mobile App. "iAM Smart" Mobile App has verified that there is a supported browser installed in the same mobile phone.

Post-conditions

- authCode will expire in 30 seconds and Online Service can only use the authCode once for requesting the accessToken from "iAM Smart" System.

Error conditions

- This API is a HTTP GET request. If parameter "error_code" is returned, it means the direct login request is failed.

Error Code	Error Description	Suggested Action
error_code – D20008	unregistered redirectURI	Check value is registered with "iAM Smart" System
error_code – D20012	insufficient scope	Check authorisation scope(s) requested is/are sufficient for invoking the corresponding "iAM Smart" API
error_code – D20015	Scope mismatch	Online service shall verify the scopes approved for the client id in

Error Code	Error Description	Suggested Action
		ESP and the scopes configured in Online Service catalogue.

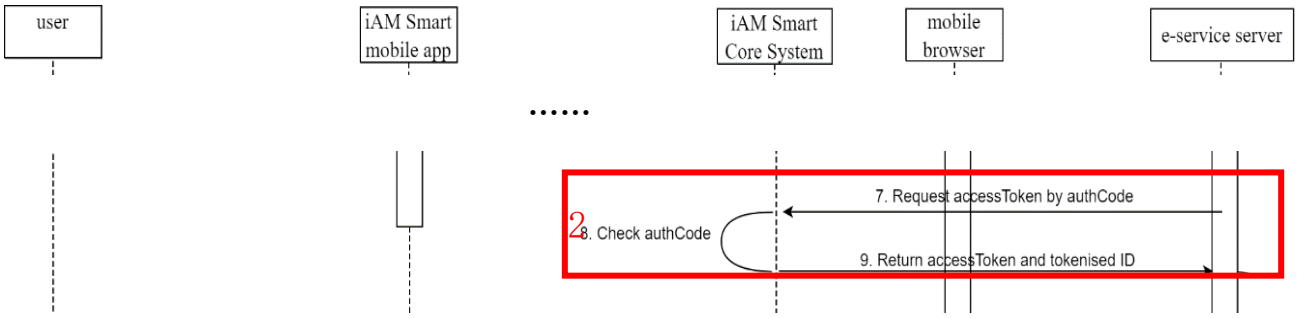
Callback Parameters

- Please refer to Section 6.4.2 of "iAM Smart" API Specification.

Notes

- Callback parameter "code": It is the authCode for requesting the accessToken from "iAM Smart" System. Its validity lasts for 1 minute and can only be used once.
- If errors occur, "error_code" instead of "code" will be returned.
- RefID is a unique value for troubleshooting when necessary.
- Online Service should check if any login session exists and perform session management properly according to business needs.

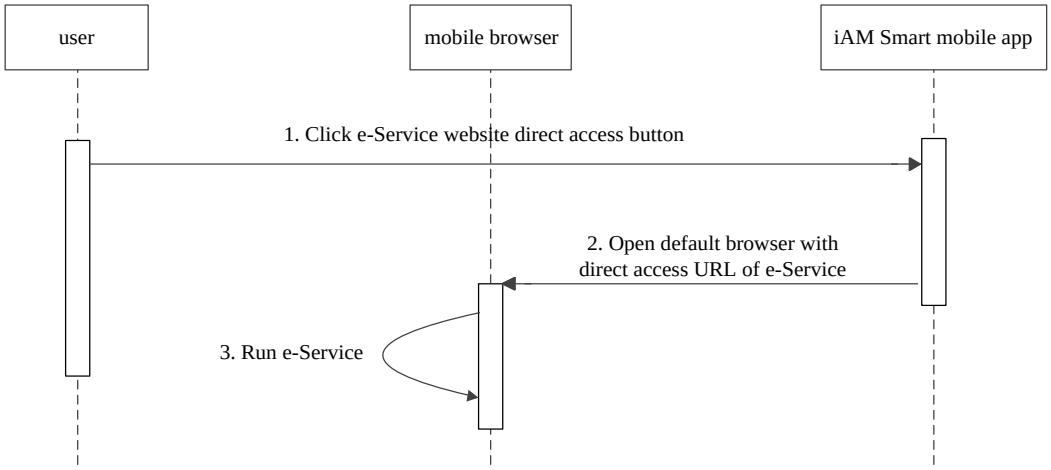
3.10.1.2 Implementing (3) POST: Request accessToken & tokenised ID



Please refer to Section 3.4.1.3.

3.10.2 Scenario 2: Direct Access with Online Service Website

The sequence diagram below shows how users initiate Direct Access in the "iAM Smart" Mobile App.



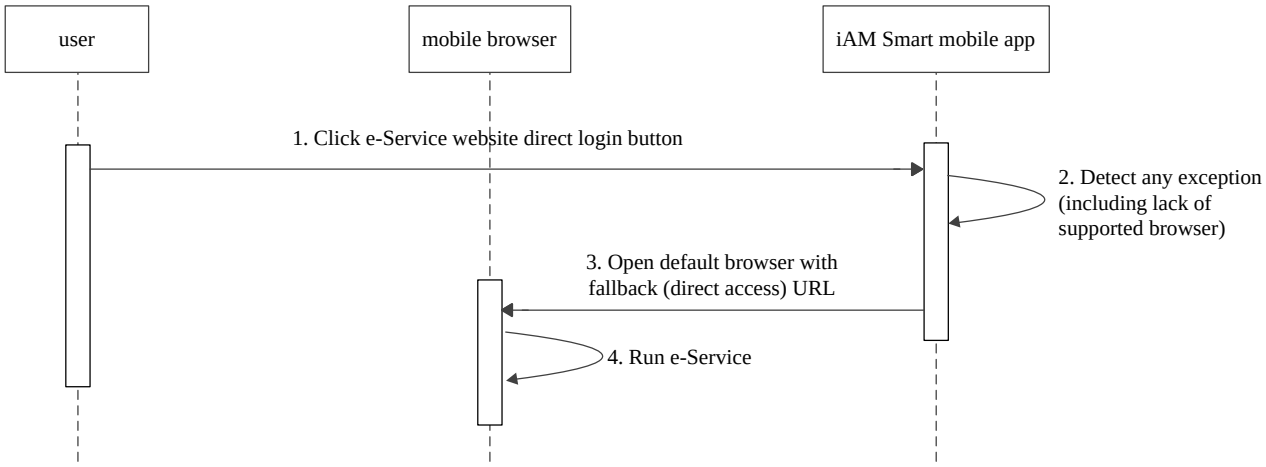
Step 1. User opens "iAM Smart" Mobile App, browses Online Service catalogue or views the detailed information of Online Service and clicks the button to invoke the Online Service.

Step 2. "iAM Smart" Mobile App opens the default browser with direct access URL of Online Service. The three direct access URLs (or fallback URLs, for the English, Traditional Chinese, and Simplified Chinese languages) for the Online Service must be registered in advance in the "iAM Smart" Platform. "iAM Smart" Mobile App chooses the URL based on its current UI display language.

Step 3. User starts using Online Service website in the default browser.

3.10.3 Scenario 3: Direct Login with Online Service Website (Fallback Mechanism)

The sequence diagram below shows how users initiate Direct Login in that "iAM Smart" Mobile App but fallback to direct access because there is no supported browser or exception happens.



Step 1. User opens "iAM Smart" Mobile App, browses Online Service catalogue or views the detailed information of Online Service and clicks the button to invoke the Online Service.

Step 2. "iAM Smart" Mobile App checks if any exception is detected. For example, "iAM Smart" Mobile App checks and does not find browser that support the direct login process.

Step 3. "iAM Smart" Mobile App opens the fallback URL with the default browser. The three fallback URLs (or direct access URLs, for the English, Traditional Chinese, and Simplified Chinese languages) for the Online Service must be registered in advance in the "iAM Smart" Platform. "iAM Smart" Mobile App chooses the URL based on its current UI display language.

Step 4. User starts using Online Service website with the default browser.

3.10.4 Scenario 4: Direct Login with Online Service App (Direct Login v2)

The sequence diagram below shows how users initiate Direct Login in the "iAM Smart" Mobile App and then automatically log in to the Online Service app installed in the same mobile device.

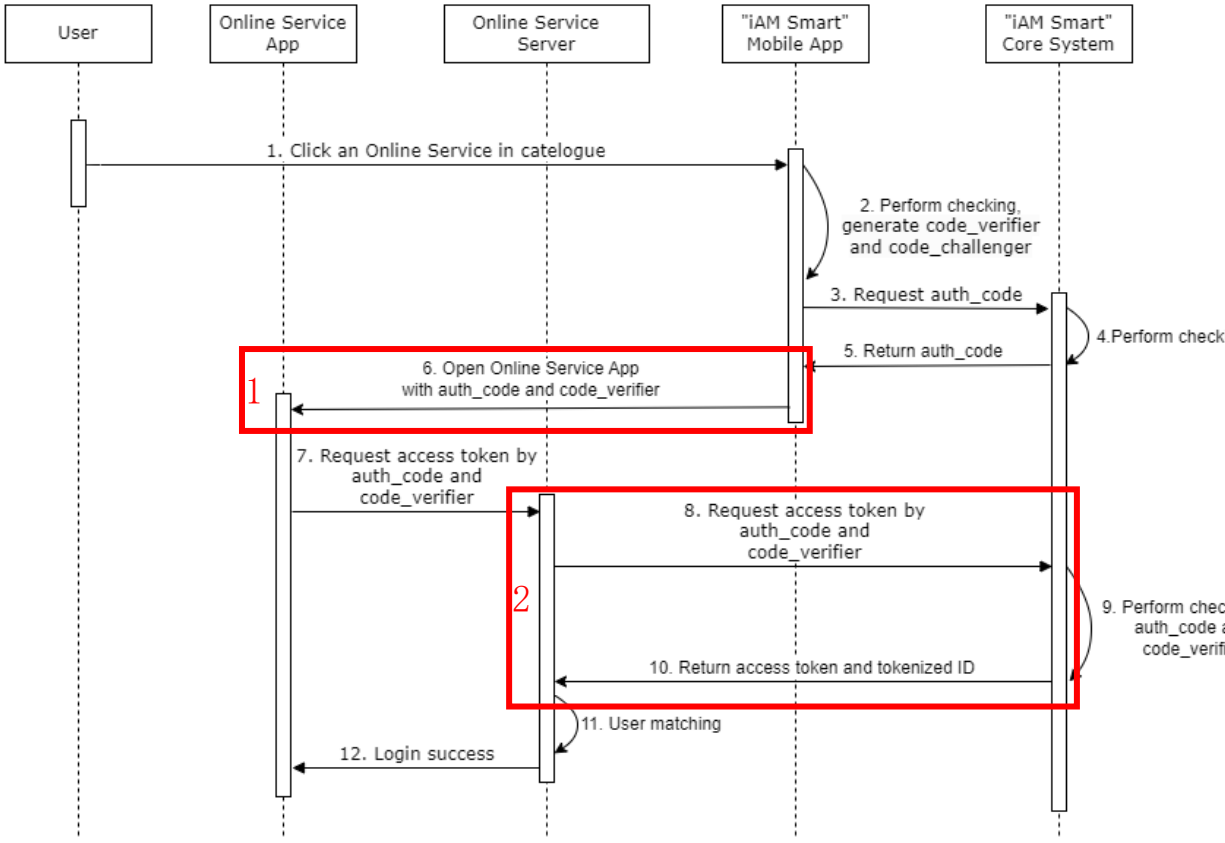


Figure-23 Direct Login with Online Service App

Pre-conditions: The Online Service App is registered to iAM Smart team with the following details,

- Basic information, including the service name, scope
- Package name (Submission should be made 3 months in advance and should be registered in iAM Smart App)
- iOS Universal Link and Android app fingerprint of the signing certificate correspond to package name in ESP

Step 1. User opens "iAM Smart" Mobile App, browses Online Service catalogue or views the detailed information of Online Service and clicks the "(Direct Login) App" button;

Step 2-5. “iAM Smart” Mobile App submits the login request to “iAM Smart” Core System. “iAM Smart” Core System verifies the necessary information of the login request and returns login result and authCode to “iAM Smart” Mobile App;

Step 6. “iAM Smart” Mobile App invokes Online Service App with required parameters using Universal Link (iOS) or package name (Android) according to the operating system;

“iAM Smart” API (URL scheme: Callback with authCode to Online Service App)

For iOS platform, the system browser will open the Universal Link and forward the request to Online Service server. Upon receiving the browser request, Online Service is recommended to instruct user to download and install the Online Service App.

For Android platform, “iAM Smart” will validate the Online Service App. If the Online Service App is installed and both the fingerprint of signing certificate and package name are valid, it will be launched by package name and the login request data, including the authCode and code_verifier will be forwarded. Otherwise, an alert dialog will be shown, the user can open the fallback URL via system browser by choosing the “More” option. Upon receiving the browser request, Online Service is recommended to instruct the user to download and install the Online Service App.

Please note that the Online Service app must check if another login session is already exists and perform session management properly according to the business needs. Online Service is strongly recommended to terminate the existing login session before it proceeds with the rest of the Direct Login workflow. The Online Service is recommended to ask the user if he/she confirms to logout the existing session and use “iAM Smart” Mobile App to login. Please refer to Section 3.10.4.1 for implementation details of the Online Service app on how to receive callback for direct login.

Step 7. Online Service App sends authCode, code_verifier, etc. to Online Service Server;

Step 8. Online Service Server provides the authCode and code_verifier to invokes the "iAM Smart" API to obtain the accessToken and the tokenised ID (i.e., openID) of the "iAM Smart" user with the selected Online Service. API data encryption is required.

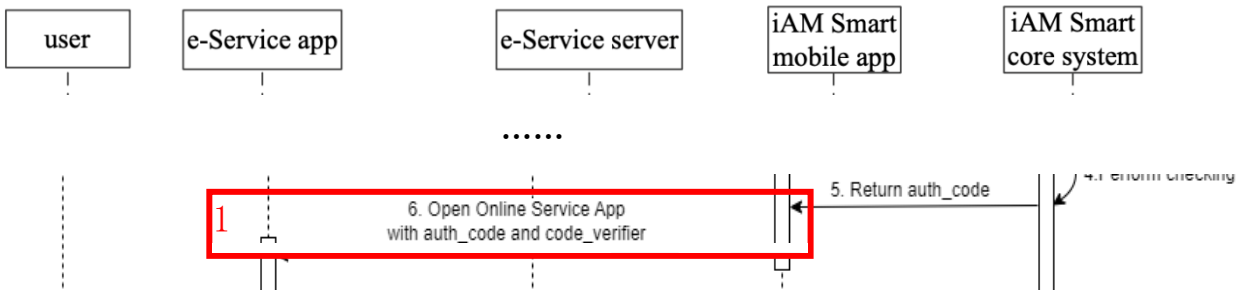
"iAM Smart" API (POST: Request accessToken & Tokenised ID)

Step 9-10. "iAM Smart" Core System verifies if the code_verifier received from Online Service Server and the code_challenge stored on the system are valid. "iAM Smart" Core System return the AccessToken and the tokenised ID (i.e., openID) of the "iAM Smart" user with the selected Online Service to Online Service Server.

Step 11. Online Service then performs user matching using the openID with its user repository and determines the result of this login request;

Step 12. Online Service App shows the login result (e.g., successful when Tokenised ID is matched with a user account of Online Service).

3.10.4.1 Implementing (1) Callback with authCode and code_verifier to Online Service App



Pre-conditions

- "iAM Smart" user has initiated Online Service app via direct login in "iAM Smart" Mobile App.
- Online Service has set up Universal Link and/or Android package name with fingerprint of signing certificate for the Online Service App.

Post-conditions

- authCode will be expired in 30 seconds and Online Service can only use the

authCode once for requesting the accessToken from “iAM Smart” System.

- **Callback Parameters**

Parameter	Type	Presence	Description
code	String	Required (Conditional)	The authorisation code generated by the “iAM Smart” System. The authorisation code will be expired in 30 seconds after issuance. Online service MUST NOT use the authorisation code more than once. An error message will be returned if an authorisation code is expired or re-used.
code_verifier	String	Required	The value of code_verifier is generated by “iAM Smart” System.

- **Example URL Scheme (iOS Only)**

```
// Line breaks are for legibility only.
<Universal Link><landing location>
?code=0ad186353c424c64897fcc00445c9ba1
&code_verifier=3LpK1f7Qs8wNcIxRyO5JvD2Am9Vbhz0Zt6G4BdFgHlTjXeYkSpWqEuMrCnU
iVo1x7ZaQ9s8W7m6D4b3F2h5J9n0B8v7c6x5Z2A1s3D4f7G8h9J0k1L2q3v5B4n6Mcis8
```

- **Example Package Name (Android Only)**

```
Intent ii=new Intent(<package name>, <activity name>);
ii.putExtra("code", "0ad186353c424c64897fcc00445c9ba1");
ii.putExtra("code_verifier", "DxjdFp0vKYCY4F0DE6M3eEadLxcJBTb6k4LZ9J5z");
ii.putExtra ("activityParams", <activity params>);
startActivity(ii);
```

- **Set up Universal Link**

For the steps to set up Universal Link on iOS, please refer to Guides and Reference for iOS in https://developer.apple.com/ios/universal-links/.

Please note that the Universal Link may not be triggered by “iAM Smart” Mobile App only.

- **Set up for Android**

For the steps to enable “iAM Smart” to open Online Service app via Android

Package Name, modification of android files are required.

An activity must be appended in AndroidManifest.xml. Please note that the name of activity can be adjust according to the requirement of Online Service App.

```
<activity
    android:name = ".PackageNameIntentActivity"
    android:launchMode="singleTask" // Example only
    android:exported = "true" // Example only
    android:screenOrientation = "portrait">
</activity>
```

Within the `PackageNameIntentActivity` class, the authCode and code_verifier should be captured and appropriately handled.

```
val authCodeId = intent.getStringExtra("code")
val codeVerifier = intent.getStringExtra("code_verifier")
val activityParams =  intent.getStringExtra("activityParams")
```

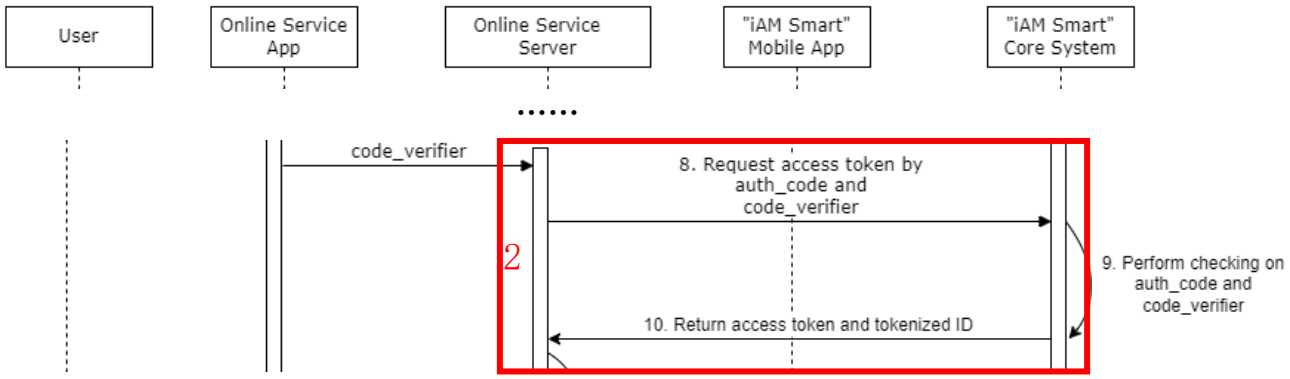
- **Session Management**

The Online Service app must check if another login session already exists and perform session management properly according to the business needs. Online Service is strongly recommended to terminate the existing login session before it proceeds with the rest of the Direct Login workflow. As the Universal Link may not be triggered by "iAM Smart" Mobile App, the Online Service app should take into this consideration to design the session management.

Notes

- Please refer to Section 3.4.2.2.

3.10.4.2 Implementing (2) POST: Request accessToken & tokenised ID



Pre-conditions

- Online Service should request the accessToken using a valid authCode. authCode is valid for 30 seconds and should not be used more than once.
- API data encryption is required

Post-conditions

- Please refer to Section 3.4.1.3.

Error conditions

- Please refer to Section 3.2.3.

- **Request and Response Parameters**

Name	Description
Service Full Name	Request access token and tokenised ID with authCode
URI (as in RESTFUL API)	`https://<iAM_Smart_domain>/api/v1/auth/getToken`
Request Type	POST
Service Version	1.0.0
Description of Service	Online service uses this API to retrieve the access token and Tokenised ID (openID). An authorisation code is necessary during the process. The `accessToken` and `openID` will be used to call corresponding "iAM Smart" services subsequently.

- **Request and Response Parameters**

Parameter	Type	Presence	Description

code	String	Required	The authorisation code is received from the authorisation server. It can only be used once.
code_verifier	String	Required (Conditional)	The same code_verifier value returned by “iAM Smart” Core System should be used. Required for Direct Login v2.
isDirectLoginV2	Boolean	Optional	Online service must configure separate callback endpoint (different from the one for normal login and Direct Login v1) to receive the authorisation code ("authCode") for Direct Login v2. Online service must submit the parameter "isDirectLoginV2" and set this value to *"true"* if the authCode is received by the callback endpoint for Direct Login v2. If the authCode for normal login or Direct Login v1 is received from other callback endpoints, online service can consider omitting this parameter or return a "false" value. The default value is "false".
grantType	String	Required	The value MUST be set to authorization_code.

- **Example Request**

```
// Line breaks are for legibility only.
// Please refer to section 3.2.2 for generating shared common parameters/
header format.
POST
https://<iAM_Smart_domain>/api/v1/auth/getToken
// Request Headers
clientID: "edae2e2529ff46228af1e4d18c8405d1"
signatureMethod: "HmacSHA256"
signature: "5X42Y1B7MEd8Mm%2BonwjiQz9VCZkkrntADskXsYntavU%3D"
timestamp: 1557048906183
nonce: "e893647dc4204eb9b7b8eddd527b687c"
// Unencrypted Request Body (authCode from Direct Login v2)
{
```

```
  "code": "xxxa42e76bf4cb0846a68e6d83d6096",
  "code_verifier":"3LpK1f7Qs8wNcIxRyO5JvD2Am9Vbhz0Zt6G4BdFgHlTjXeYkSpWqEuMr
CnUiVo1x7ZaQ9s8W7m6D4b3F2h5J9n0B8v7c6x5Z2A1s3D4f7G8h9J0k1L2q3v5B4n6Mcis8",
  "isDirectLoginV2": "true"
  "grantType": "authorization_code"
}
```

- **Response Parameters**

Parameter	Type	Presence	Description
accessToken	String	Required	accessToken value can be used multiple of times before expiry.
tokenType	String	Required	Token type, support "Bearer" only.
issueAt	Long	Required	The accessToken issue time is expressed in the number of milliseconds since January 1, 1970 00:00:00 GMT.
expiresIn	Long	Required	The lifetime in milliseconds of the token. The value may vary for different Online Services.
openID	String	Required	Tokenised ID, uniquely generated for each user of each online service website or mobile application.
lastModifiedDate	Long	Required	The datetime of the user complete registration at "iAM Smart" System. The value will be updated when either of the following is valid:- (1) If any one of the following verified data are changed. English name Chinese name (* not applicable if it was marked as unverified during registration) Gender Date of birth (2) User re-register "iAM Smart" after "iAM Smart" de-registration. The modification time will be expressed in the number of milliseconds since January 1, 1970 00:00:00 GMT.

`userType`	String	Required	`default` or `sign` `default`: "iAM Smart" user `sign`: "iAM Smart+" user (digital signing capability)
`scope`	String	Required	The scope of the token. Please refer to the corresponding section specified in each API function.

- **Example Success Response**

```
// The descriptions of txID, code, and message are in Section 3.2.2
// The decrypted body
{
  "txID": "<T=938ffb193b4b4370b6c2584372c6a588>",
  "code": "D00000",
  "message": "SUCCESS",
  "content": {
    "accessToken": "0ad186353c424c64897fcc00445c9ba1",
    "tokenType": "Bearer",
    "issueAt": 1557053922938,
    "expiresIn": 14400000,
    "openID": "liR14%2BvX%2F5hSum5uf4ERczu0KcDnIJA5BM7FoM1ag9c%3D",
    "lastModifiedDate": 1560849218006,
    "userType": "sign",
    "scope": "eidapi_auth eidapi_formFilling"
  }
}
```

- **Example Error Response**

```
// The descriptions of txID, code, and message are in Section 3.2.2
// The decrypted body
{
  "txID": "<T=938ffb193b4b4370b6c2584372c6a588>",
  "code": "D40004",
  "message": "authCode not exist or expired",
}
```

Notes

- Please refer to Section 3.4.1.3, except for the following paramters:
- Request parameter “code_verifier”: code_verifier and hash of “code_verifier” (code_challenge) are used to ensure the request validity and prevent the code interception attack.

3.10.5 Scenario 5: Direct Login with Online Service App (Direct Login v1)

The sequence diagram below shows how users initiate Direct Login in the "iAM Smart" Mobile App and then automatically log in to the Online Service app installed in the same mobile device.

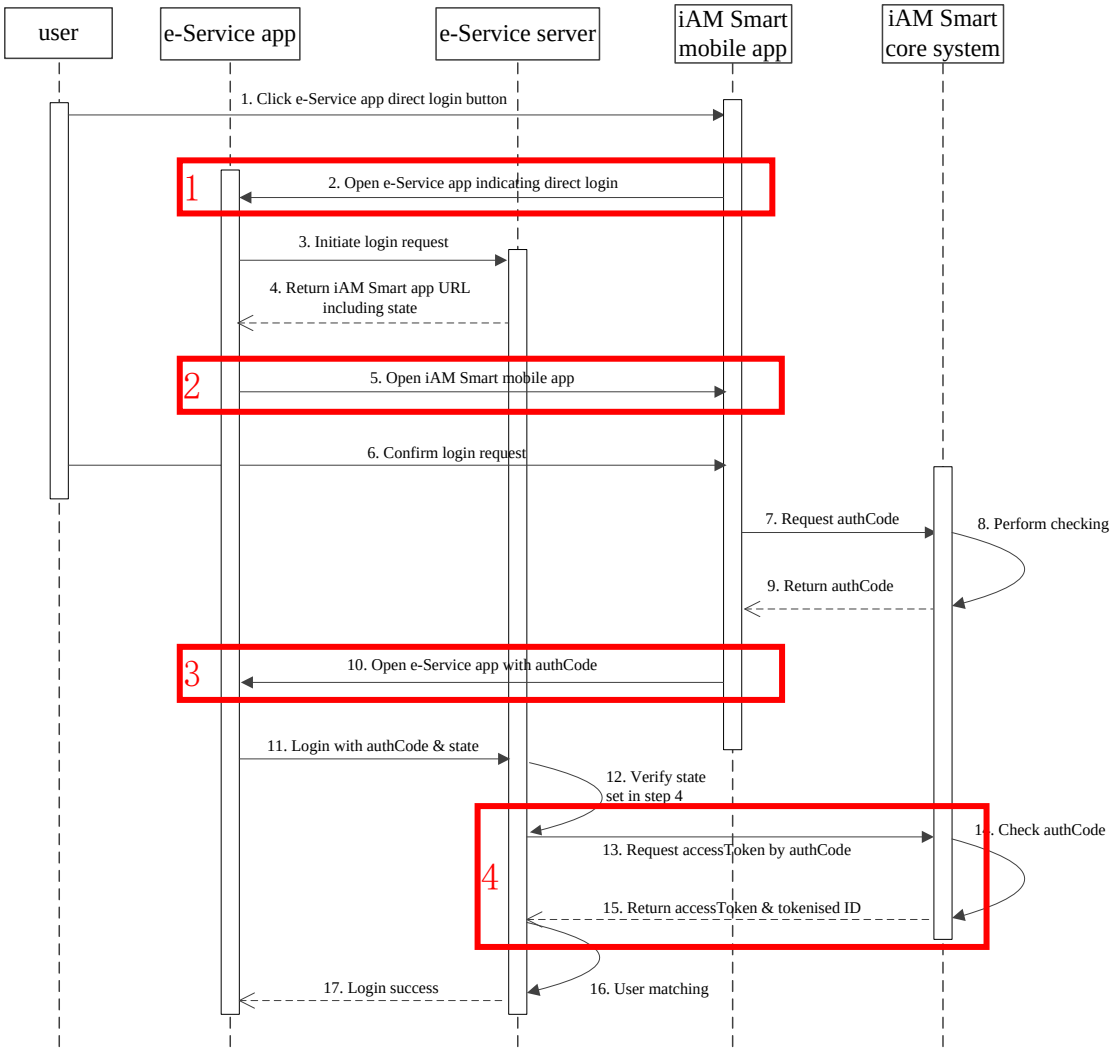


Figure-24 Direct Login with Online Service App

Step 1. User opens "iAM Smart" Mobile App, browses Online Service catalogue or views the detailed information of Online Service and clicks the "(Direct Login) App" button;

Step 2. "iAM Smart" Mobile App opens Online Service app through Universal Link URL (iOS) or App Links URL (Android) that has been predefined in the "iAM Smart" Platform for opening the Online Service app as direct login. If the Online Service app is not installed in the same mobile device, the system browser will

open the URL and forward the request to Online Service server. Upon receiving this browser request, Online Service is recommended to redirect the request to Online Service app detail page in the App Store (iOS) or Play Store (Android).

The Online Service app must check if another login session already exists and perform session management properly according to the business needs. Online Service is strongly recommended to terminate the existing login session before it proceeds with the rest of the Direct Login workflow. The Online Service is recommended to ask the user if he/she confirms to logout existing session and use "iAM Smart" Mobile App to login. Please refer to Section 3.10.4.1 for implementation details of the Online Service app on how to receive request for direct login.

Step 3-4. Online Service App requests Online Service Server to prepare the request parameters including "state", "clientID", "redirectURI", "scope", "source" (set as App_Scheme or App_Link), etc. and constructs the URL Scheme to invoke "iAM Smart" Mobile App by Online Service App;

Step 5. Online Service App invokes "iAM Smart" Mobile App using URL Scheme with request parameters (set <Context> as "auth" in URL Scheme);

"iAM Smart" API (URL Scheme: Open "iAM Smart" Mobile App for Authentication)

Step 6-7. "iAM Smart" user logs in the "iAM Smart" Mobile App and confirms the Online Service login request and then "iAM Smart" Mobile App submits the login request to "iAM Smart" Server;

Step 8-9. "iAM Smart" Server verifies the necessary information of the login request and returns login result to "iAM Smart" Mobile App;

Step 10. "iAM Smart" Mobile App invokes Online Service App with required parameters using URL Scheme or Universal Link/App Link;

Online Service Callback API (Callback with authCode to Online Service App)

Step 11. Online Service App sends authCode, state, etc. to Online Service Server;

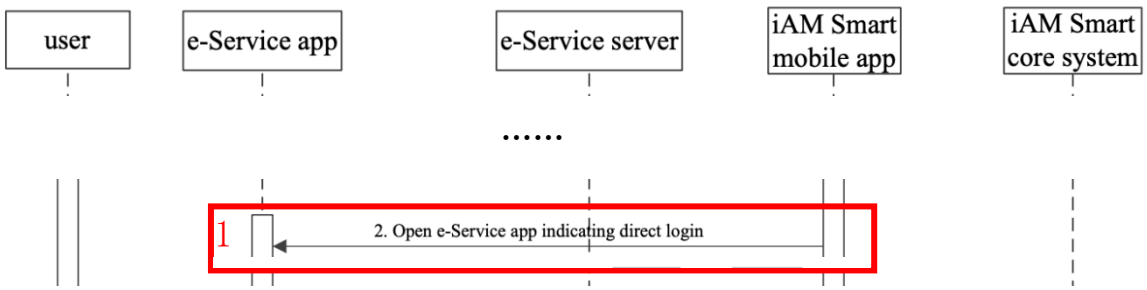
Step 12-15. When Online Service Server gets the authCode, it shall verify the state with the one generated in step 4 and then invokes the "iAM Smart" API to obtain the accessToken and the tokenised ID (i.e., openID) of the "iAM Smart" user with the selected Online Service. API data encryption is required;

"iAM Smart" API (POST: Request accessToken & tokenised ID)

Step 16. Online Service then performs user matching using the openID with its user repository and determines the result of this login request;

Step 17. Online Service App shows the login result (e.g., successful when Tokenised ID is matched with a user account of Online Service).

3.10.5.1 Implementing (1) GET: Online Service app to receive request for direct login



Pre-conditions

- "iAM Smart" user has initiated Online Service app via direct login in "iAM Smart" Mobile App.
- Online Service has set up Universal Link and/or App Links for its app.

Post-conditions

- Online Service app is opened. It will request Online Service server to set a state value that will be used to compare with the state returned with authCode in "iAM Smart" request (i.e., step 10) during verification in step 12.

Error conditions

- Online Service needs to implement this GET request. If Universal Link or App Links has not been set up, Online Service is recommended to redirect the request to Online Service app detail page in the App Store (iOS) or Play Store (Android).

Implementation Details

- **API Description**

Name	Description
Service Full Name	Open Online Service app indicating direct login
Universal Link URL or App Link URL	`https://<Online Service_domain>/<Online Service_context>/<app_direct_login_endpoint>?platform=<platform>&scope=<scope>`
Request Type	GET
Service Version	1.0
Description of Service	Online Service should register this endpoint in the "iAM Smart" Platform and implement it as Universal Link URL (iOS) or App Link URL (Android). If the Online Service app is installed in the same mobile device, the Online Service app will be opened when "iAM Smart" Mobile App invokes this URL. Otherwise, the URL will be opened with system browser and we recommend Online Service to redirect the browser request to the Online Service app detail page in the App Store (iOS) or Play Store (Android).

- **Request Parameters**

Parameter	Type	Condition	Description
`platform`	String	Required	Either `iOS` or `Android`. It helps Online Service to decide where the request should be redirected to (i.e., App Store or Play store) when Online Service app has not been installed in the same mobile device.
`scope`	String	Required	The previously agreed scope (e.g., `eidapi_auth eidapi_sign`). It will be URL encoded.

- **Example Request**

```
GET
https://esd-isit.staging-
eid.gov.hk/eservice/directLogin?platform=iOS&scope
=eidapi_auth
```

- **Set up Universal Link or App Link**

  For the steps to set up Universal Link on iOS, please refer to Guides and Reference for iOS in https://developer.apple.com/ios/universal-links/.

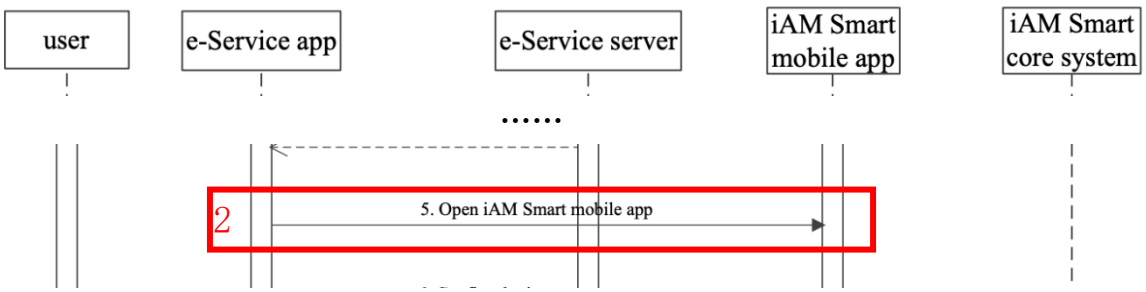
  For the steps to set up App Link on Android, please refer to the Guides for Android in https://developer.android.com/training/app-links.

  Please note that the Universal Link / App Link may not be triggered by "iAM Smart" Mobile App only.

- **Session Management**

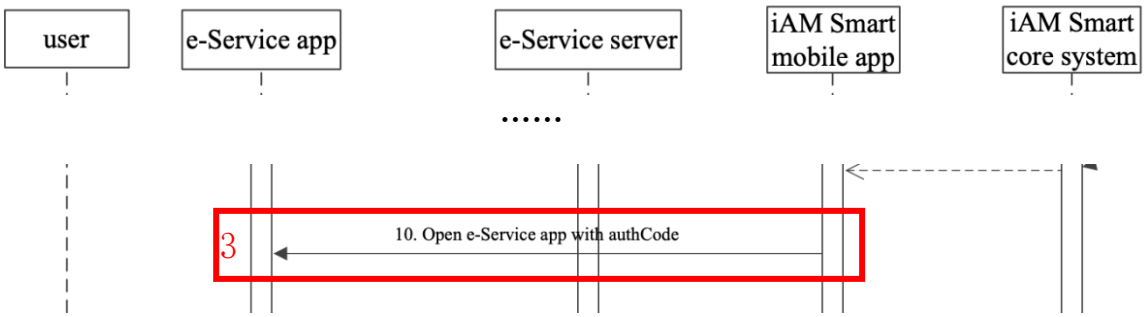
  The Online Service app must check if another login session already exists and perform session management properly according to the business needs. Online Service is strongly recommended to terminate the existing login session before it proceeds with the rest of the Direct Login workflow. As the Universal Link / App Link may not be triggered by "iAM Smart" Mobile App, the Online Service app should take into this consideration to design the session management. Online Service is strongly recommended to ask the user if he/she confirms to logout existing session and use "iAM Smart" Mobile App to login.

3.10.5.2 Implementing (2) URL Scheme: Open "iAM Smart" Mobile App for Authentication



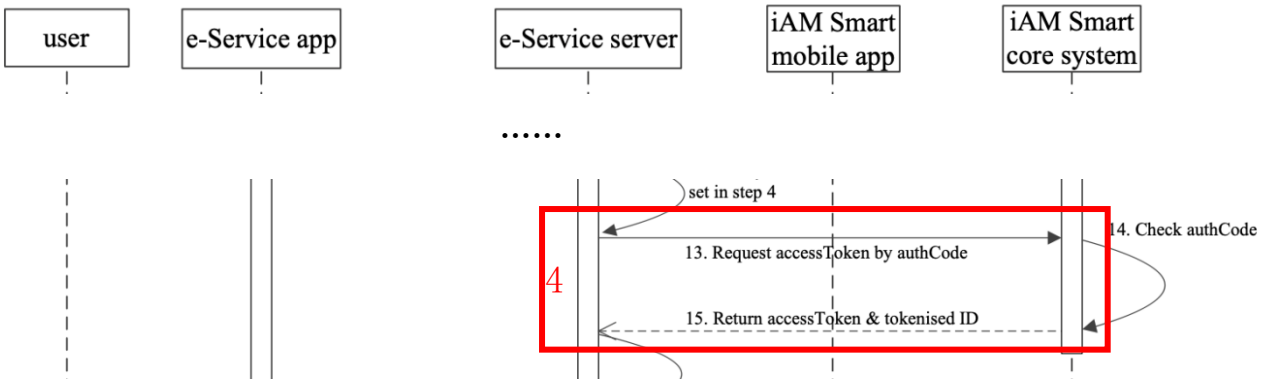
Please refer to Section 3.4.4.1.

3.10.5.3 Implementing (3) Callback with authCode to Online Service App



Please refer to Section 3.4.4.2.

3.10.5.4 Implementing (4) POST: Request accessToken & tokenised ID



Please refer to Section 3.4.4.3.

3.11 WORKFLOWS FOR BULK DIGITAL SIGNING WITH SERVICE LOGIN

There are different use cases for "iAM Smart" System and Online Services. Interactions among user, Online Service and "iAM Smart" System can be summarised into the following sequence diagrams.

3.11.1 Scenario 1: Bulk Digital Signing (Online Service Website/App in Different Device)

The sequence diagram below shows how an authenticated user authorises and signs multiple document hashes and/or digests when Online Service website/app and the "iAM Smart" Mobile App are running in different devices.

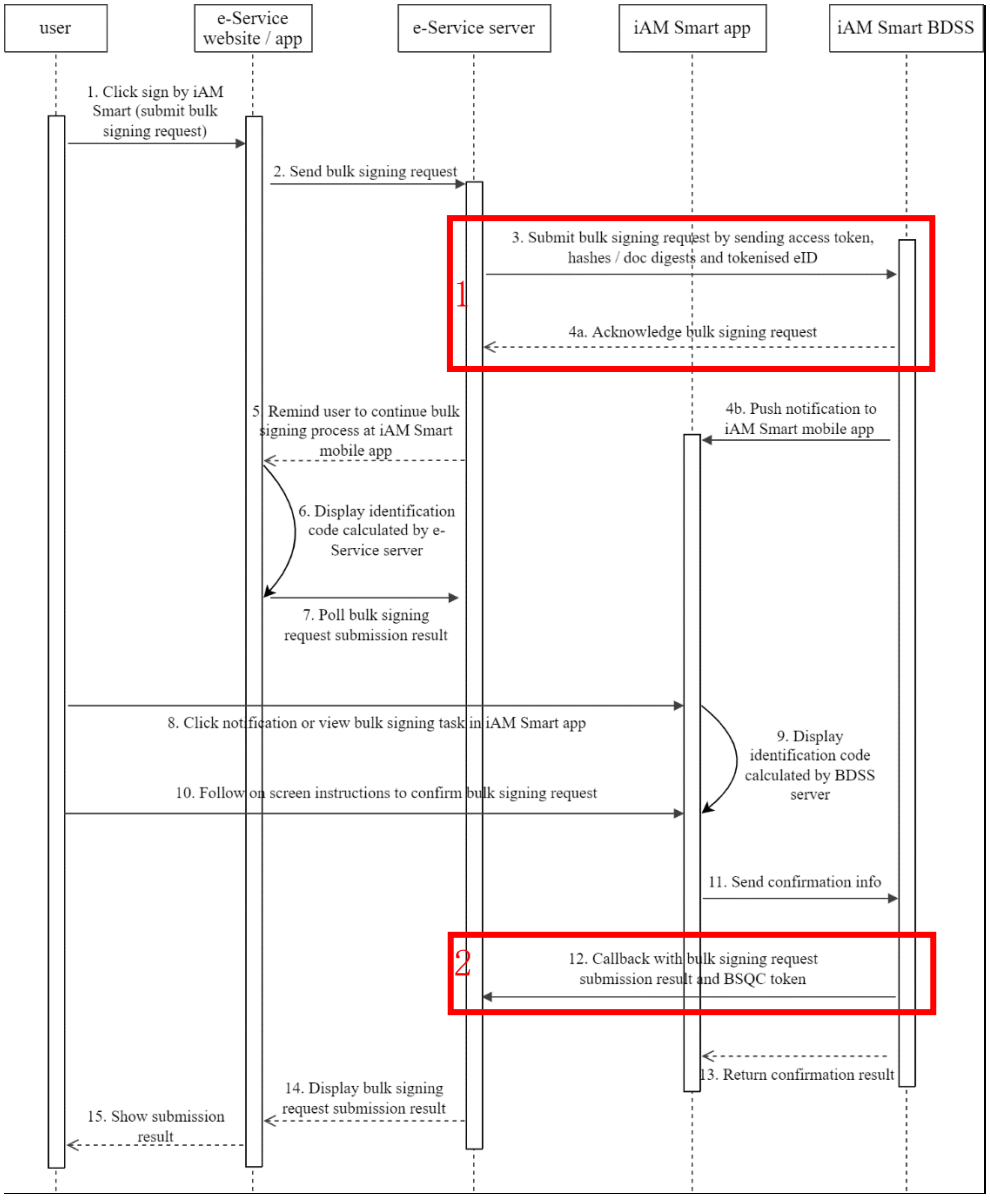


Figure-25 Bulk Digital Signing (Online Service Website/App in Different Device)

Step 1. User clicks the “Sign by iAM Smart” button in Online Service Website/App;

Step 2-3. Online Service initiates bulk digital signing request to invoke the “iAM Smart” API with the “businessID” of this request, accessToken, openID (Tokenised ID), source, redirectURI, callbackResultURI, state, HKICHash, department, serviceName, requestName, maxCallbackSigs, documents, etc.

The request parameter “documents” contains the Document Hash (hashCode) / PDF digest (docDigest) of all documents, signature algorithm (only for document). The request parameter “source” will be the browser's user agent value (for Online Service Website) or “App_Scheme”/“App_Link” (for Online Service App) in this scenario. API encryption is required;

“iAM Smart” API (POST: Request Bulk Signing)

Step 4a. “iAM Smart” System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the digital signing request to Online Service server by returning POST response with parameter “authByQR” (set to “true”) in this scenario;

Step 4b. “iAM Smart” System pushes a notification message to the “iAM Smart” Mobile App based on the Tokenised ID;

Step 5-6. Online Service server concatenates all document hashes and PDF digests and hash of Tokenised ID to calculate a 6-digit identification code and instructs Online Service Website/App to display the instructions with the 6-digit identification code to inform “iAM Smart” user to process the signing authorisation request in “iAM Smart” Mobile App;

Step 7. Online Service Website/App should keep synchronising with Online Service server for the digital signing processing result (e.g. polling);

Step 8. “iAM Smart” user opens “iAM Smart” Mobile App, reviews the digital signing authorisation request (e.g. continue or reject the request).

Step 9. “iAM Smart” user verifies the 6-digit identification code with Online Service Website/App;

Step 10-11. “iAM Smart” user follows the instructions in “iAM Smart” Mobile App to complete digital signing operation;

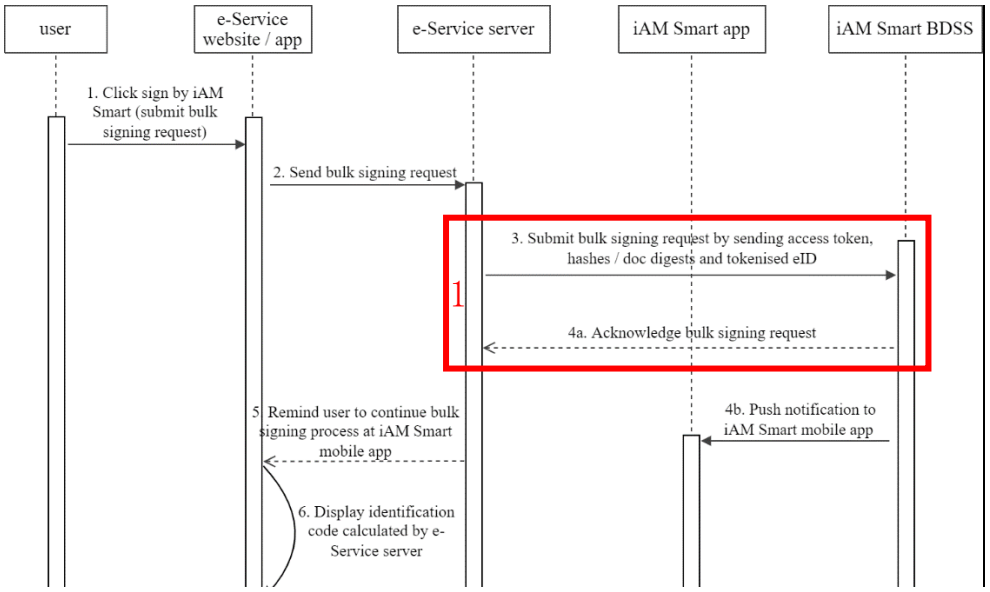
Step 12. “iAM Smart” BDSS invokes Online Service callback API to return the BSQCToken coresponding with openID of the digital signing request. API data encryption by AES;

“iAM Smart” API (POST: Callback to Receive BSQC Token)

Step 13. “iAM Smart” BDSS saves the signing request to MQ queue and returns the result with the same “businessID” of the signing request.

Step 14-15. Online Service server returns the submission result (status of Pending for Digital Signing) to Online Service Website/App, and then Online Service Website/App returns the result to user. Response to Step 7.

3.11.1.1 Implementing (1) POST: Request Bulk Digital Signing



Pre-conditions

- Online Service must possess a valid accessToken of the "iAM Smart" user with authorisation scope "bulk signing authorisation".
- Online Service Website/App and "iAM Smart" Mobile App are in different devices in this scenario.
- The "iAM Smart" user who signs the document must have a sign version "iAM Smart".
- Online Service generates a "businessID" for this bulk signing request and uses this identifier to match the callback result returned from "iAM Smart" BDSS.
- Online Service generates the Document Hash as "hashCode" parameter from the original documents to be signed using a hash algorithm that can suit the specific business need. It is recommended to use a hash algorithm that is at least SHA-256 or equivalent.
- Online Service generates the digest as the parameter "docDigest" from the pdf document to be signed using the Adobe.PPKLite filter and the adbe.pkcs7.detached subfilter.
- API data encryption is required.

Post-conditions

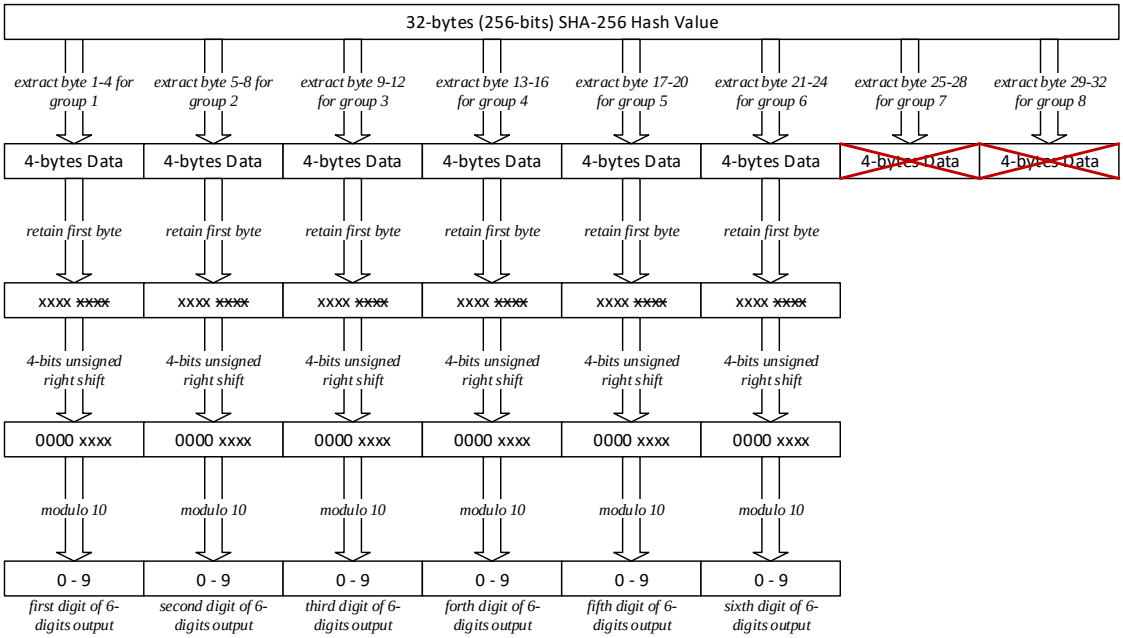
- Online Service server should check the response parameter "authByQR" and "ticketID" and determine the next action. For details, please refer to Section

3.2.1. “ticketID” will not be provided by “iAM Smart” System in this scenario.

- Online Service has to calculate a 6-digit identification code using all Document Hashes and PDF digests and Tokenised ID hash value. Online Service Website/App should show the 6-digit identification code and instruction to inform “iAM Smart” user to process the digital signing request in “iAM Smart” Mobile App when “authByQR” is “true”.

6-digit identification code generation algorithm is as follows:

1. Concatenate all hashes and document digests following the submission order to “iAM Smart” System.
   i.e.: First Hash / Document Digest Bytes + Second Hash / Document Digest Bytes + Third Hash / Document Digest Bytes + …
2. Calculate SHA-512 hash value on the output of combining concatenated value from Step 1 with SHA-512 hash value of Open ID (for non-anonymous digital signing).
   i.e.: For non-anonymous: SHA-512(Concatenated Value from Step 1 + SHA-512(Open ID))
3. Calculate SHA-256 hash value on the output of hashing from Step 2.
   i.e.: SHA-256(Output Value from Step 2)
4. Manipulate the output of hashing from Step 3 as follow to obtain the 6-digits identification code.
   a) Divide the 256-bits SHA-256 hash value into 8 groups of 4-bytes data. Drop the last 2 groups of data (i.e., group 7 and 8).
   b) Retain the only first byte for each group of 4-bytes data for the first 6 groups.
   c) Perform 4-bits unsigned right shift on each byte.
   d) Perform modulo 10 operations on each byte to get a value from 0 to 9.
   e) Combine each 0 to 9 value resulted together to form the 6-digits identification code. The value from group 1 as the first digit, the value from group 2 as the second digit, and so on.


32-bytes (256-bits) SHA-256 Hash Value
extract byte 1-4 for group 1
extract byte 5-8 for group 2
extract byte 9-12 for group 3
extract byte 13-16 for group 4
extract byte 17-20 for group 5
extract byte 21-24 for group 6
extract byte 25-28 for group 7
extract byte 29-32 for group 8
4-bytes Data
4-bytes Data
4-bytes Data
4-bytes Data
4-bytes Data
4-bytes Data
4-bytes Data
4-bytes Data
retain first byte
retain first byte
retain first byte
retain first byte
retain first byte
retain first byte
xxxx ~~xxxx~~
xxxx ~~xxxx~~
xxxx ~~xxxx~~
xxxx ~~xxxx~~
xxxx ~~xxxx~~
xxxx ~~xxxx~~
4-bits unsigned right shift
4-bits unsigned right shift
4-bits unsigned right shift
4-bits unsigned right shift
4-bits unsigned right shift
4-bits unsigned right shift
0000 xxxx
0000 xxxx
0000 xxxx
0000 xxxx
0000 xxxx
0000 xxxx
modulo 10
modulo 10
modulo 10
modulo 10
modulo 10
modulo 10
0 - 9
0 - 9
0 - 9
0 - 9
0 - 9
0 - 9
first digit of 6-digits output
second digit of 6-digits output
third digit of 6-digits output
forth digit of 6-digits output
fifth digit of 6-digits output
sixth digit of 6-digits output

Pseudo code for generating 6-digit identification code

```
private static final char hexDigits[] = {'0', '1', '2', '3', '4',
'5', '6', '7', '8', '9'};
private static String get6DigitsSignCode(byte[] combineHash,
byte[] openID) {
        if (null == combineHash || null == openID) {
            return null;
        }
        char code[] = new char[6];
        byte[] source = new byte[combineHash.length +
openID.length];
        System.arraycopy(combineHash, 0, source, 0,
combineHash.length);
        System.arraycopy(openID, 0, source, combineHash.length,
openID.length);
        byte[] inData =
DigestUtil.digestToByte(DigestUtil.digestToByte(source,
Alg.SHA512), Alg.SHA256);
        try {
            for (int i = 0; i < 6; i++) {
                code[i] = hexDigits[(inData[i * 4] >>> 4 &
0xf) % 10];// 无符号右移 4 位，再取高位
            }
            return new String(code);
        } catch (Exception e) {
            log.error(e.getMessage());
            if (log.isDebugEnabled()) {
                log.debug(e.getMessage(), e);
            }
        }
        return null;
    }
```

- Online Service Website/App should keep synchronising with Online Service server for the bulk digital signing result returned from "iAM Smart" System.
- API data decryption is required.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D71004	user is not allowed to sign	Inform user that "iAM Smart" bulk digital signing is not allowed for default "iAM Smart" and suggest user to upgrade to "iAM Smart" with bulk digital signing function
code - D71005	Inconsistent HKIC number	Hash of HKIC number provided is not matched with "iAM Smart" user. Check hash of HKIC number or inform user to authorise the digital signing using the correct "iAM Smart" account

Request and Response Parameters

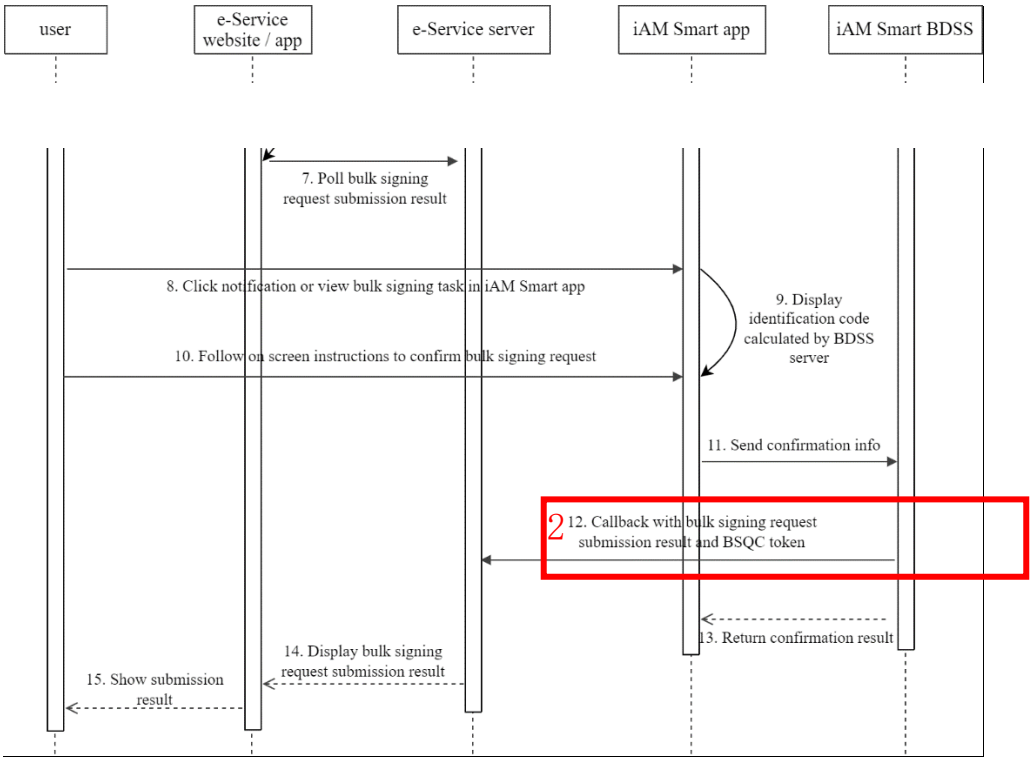
- Please refer to "iAM Smart" API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- Request parameter "source": Value should be matched with Online Service client terminal (e.g. "App_Scheme"/"App_Link" or browser's user agent value).
- Request parameter "openID": It is the Tokenised ID of the "iAM Smart" user. Online Service should ensure the accessToken and Tokenised ID are in pair and belongs to the target "iAM Smart" user for the request. The target "iAM Smart" user must have a sign version "iAM Smart".
- Request parameter "redirectURI": Callback URI for the endpoint that Online Service uses to receive BSQC Token. It must be in the same value as provided during Online Service registration.
- Request parameter "callbackResultURI": Value should equal to URI of "Callback to Receive Bulk Digital Signing Result" Online Service callback API. It must be in the same value as provided during Online Service registration.
- Request parameter "hashCode": It is the Document Hash to be signed. SHA-256 or equivalent is recommended.

- Request parameter “docDigest”: It is the PDF digest to be signed. SHA-256 or equivalent is recommended.
- Request parameter “sigAlgo”: It is the signature algorithm to be used by “iAM Smart”. The default value is "SHA256withRSA". While using "NONEwithRSA", hashCode provided must be hashed with SHA-256.
- Request parameter “HKICHash”: “iAM Smart” System will verify the HKIC hash of the “iAM Smart” user with the “HKICHash” provided by Online Service, and proceed the digital signing only when they matched. Online Service should convert the HKIC number into hash value using SHA-256 before sending to “iAM Smart” System. Only the identifier of HKIC number will be hashed, no check digit is needed.
- Request parameters “department”, “serviceName”, “documents”: They are the information of documents to be signed and will be shown in “iAM Smart” Mobile App for “iAM Smart” user's reference with the 6-digit identification code.
- Response parameter “authByQR”: The value returned by “iAM Smart” System will be “true” in this scenario.

3.11.1.2 Implementing (2) POST: Callback to Receive BSQC Token



Pre-conditions

- "iAM Smart" user accept the bulk digital signing request.
- "iAM Smart" user can also reject the bulk digital signing request.

Post-conditions

- API data decryption is required.
- Online Service server should match the callback result BSQC Token with corresponding Online Service Website/App using the "businessID".

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D71000	User cancelled signing request	Inform user the "iAM Smart" digital signing request is cancelled
code - D71001	User rejected signing request	Inform user the "iAM Smart" digital signing request is rejected

Error Code	Error Description	Suggested Action
code - D71002	Failed to request signing	Inform user the “iAM Smart” digital signing request is failed and retry later
code - D71003	Signing request timeout	Inform user the “iAM Smart” digital signing request is timeout and provide way for user to retry
code - D71005	inconsistent HKIC number	Hash of HKIC number provided is not matched with “iAM Smart” user. Check hash of HKIC number or inform user to authorise the digital signing using the correct “iAM Smart” account

Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- Callback parameter “businessID”: It is a unique identifier for Online Service to differentiate different request. Online Service can use the businessID to relate the BSQC Token with the original request.
- Callback parameter “state”: If the state parameter has been present in the request message, the exact value of state will be returned. It is used to prevent the CSRF attack. The value of state is defined by Online Service and it should be a secure random value.
- Callback parameter “signature”: It can be used by the corresponding Online Service to enquire the status of the submitted bulk digital signing request or cancel the corresponding request.

3.11.2 Scenario 2: Bulk Digital Signing (Online Service Website/App in Same Device)

The sequence diagram below shows how an authenticated user authorises and signs multiple document hashes and/or digests when online service website/app and the "iAM Smart" Mobile App are running in the same device.

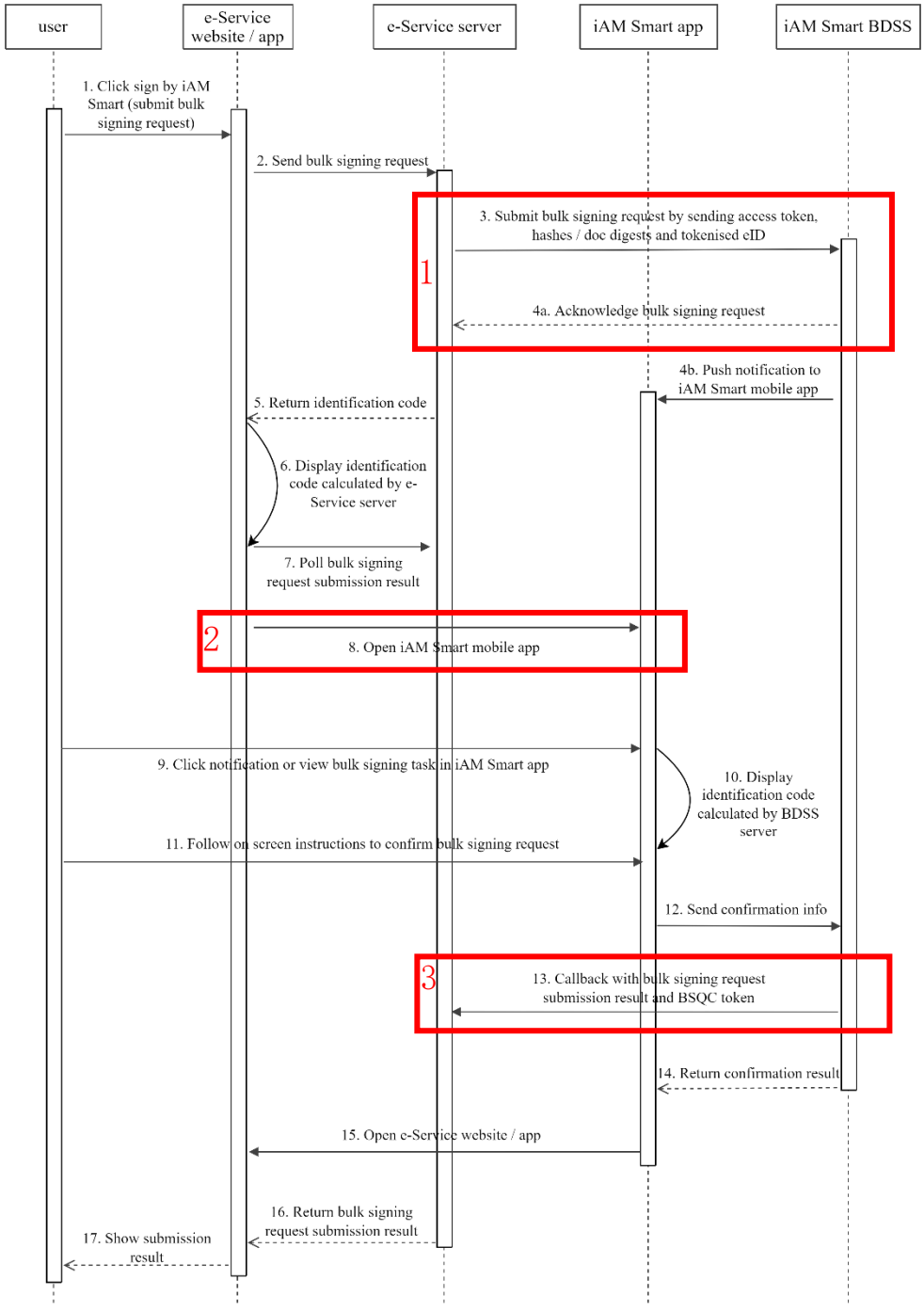


Figure-26 Bulk Digital Signing with Service Login (Online Service Website/App in Same Device)

Step 1. User clicks the "Sign by iAM Smart" button in Online Service Website/App;

Step 2-3. Online Service initiates bulk digital signing request to invoke the "iAM Smart" API with the "businessID" of this request, accessToken, openID (Tokenised ID), source, redirectURI, callbackResultURI, state, HKICHash, department, serviceName, requestName, maxCallbackSigs, documents, etc.

The request parameter "documents" contains the Document Hash (hashCode) / PDF digest (docDigest) of all documents, signature algorithm (only for document). The request parameter "source" will be the browser's user agent value (for Online Service Website) or "App_Scheme"/"App_Link" (for Online Service App) in this scenario. API encryption is required;

"iAM Smart" API (POST: Request Bulk Signing)

Step 4a. "iAM Smart" System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the digital signing request to Online Service server by returning POST response with parameter "authByQR" (set to "false") in this scenario;

Step 4b. "iAM Smart" System pushes a notification message to the "iAM Smart" Mobile App based on the Tokenised ID;

Step 5-6. Online Service server concatenates all document hashes and PDF digests and hash of Tokenised ID to calculate a 6-digit identification code and instructs Online Service Website/App to display the instructions with the 6-digit identification code to inform "iAM Smart" user to process the signing authorisation request in "iAM Smart" Mobile App;

Step 7. Online Service Website/App should keep synchronising with Online Service server for the digital signing processing result (e.g. polling);

Step 8. Online Service Website invokes "iAM Smart" Mobile App using URL Scheme with "ticketID". Other parameters of this API are not used in this scenario;

"iAM Smart" API (URL Scheme: Open "iAM Smart" Mobile App for Getting Context)

Step 9. "iAM Smart" user opens "iAM Smart" Mobile App, reviews the digital signing authorisation request (e.g. continue or reject the request).

Step 10. "iAM Smart" user verifies the 6-digit identification code with Online Service Website/App;

Step 11-12. "iAM Smart" user follows the instructions in "iAM Smart" Mobile App to complete digital signing operation;

Step 13. "iAM Smart" BDSS invokes Online Service callback API to return the BSQCToken coresponding with openID of the digital signing request. API data encryption by AES;

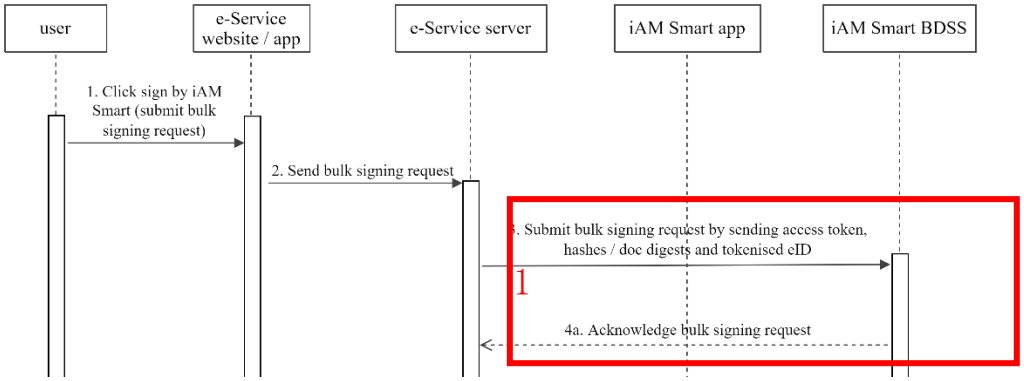
"iAM Smart" API (POST: Callback to Receive BSQC Token)

Step 14. "iAM Smart" BDSS saves the signing request to MQ queue and returns the result with the same "businessID" of the signing request.

Step 15. "iAM Smart" BDSS instructs "iAM Smart" Mobile App to invoke the Online Service App using URL Scheme, or Universal Link/App Link ("iAM Smart" System queries the information registered at Online Service registration depending on the "source" submitted in Step 3);

Step 16-17. Online Service server returns the submission result (status of Pending for Signing) to Online Service Website/App, and then Online Service Website/App returns the result to user. Response to Step 7.

3.11.2.1 Implementing (1) POST: Request Bulk Digital Signing



Pre-conditions

- Please refer to Section 3.7.1.1 except:
  - Online Service Website/App and "iAM Smart" Mobile App are in the same device in this scenario.

Post-conditions

- Please refer to Section 3.7.1.1 except:
  - Response "authByQR" is "false", "ticketID" will be provided by "iAM Smart" System in this scenario.

Error conditions

- Please refer to Section 3.7.1.1.

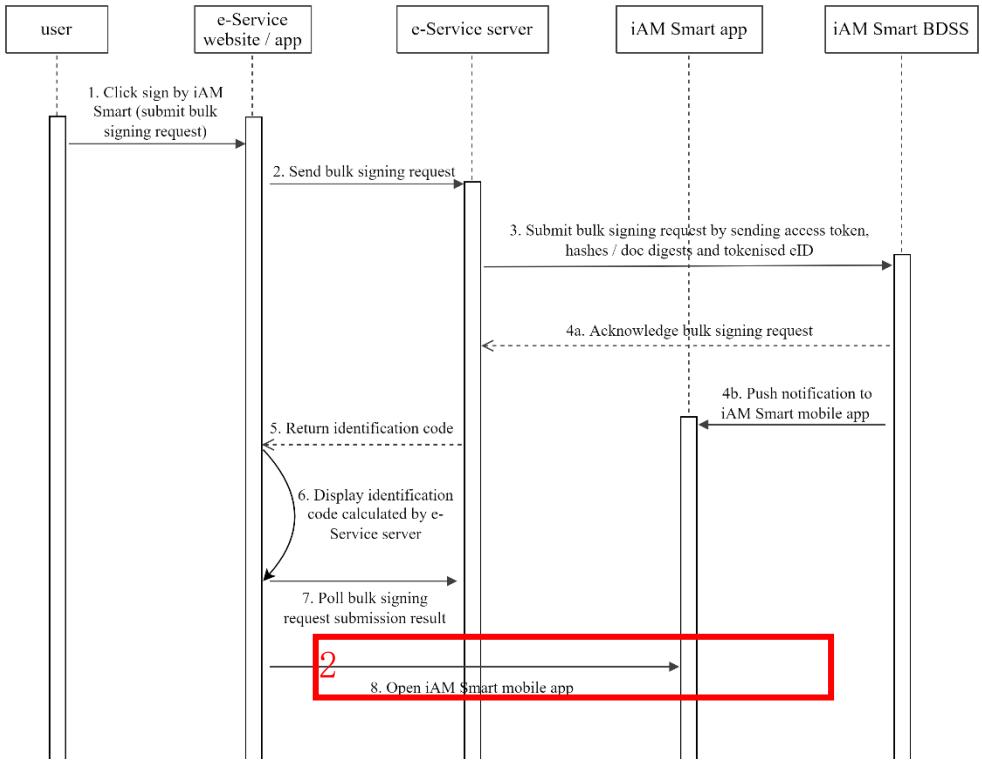
Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- Please refer to Section 3.7.1.1, except for the following parameters:
  - Response parameter "authByQR": The value is "false" in this scenario.
  - Response parameter "ticketID": It will be provided by "iAM Smart" System in this scenario. It is a 36-byte (or less) UUID number (ASCII character set).

3.11.2.2 Implementing (2) URL Scheme: *Open "iAM Smart" Mobile App for Getting Context*



Pre-conditions

- Online Service Website and "iAM Smart" Mobile App are in the same device.
- Online Service has the "ticketID" provided by "iAM Smart" System for the digital signing request;
- Other parameters of this API are not used in this scenario.

Post-conditions

- "iAM Smart" Mobile App will be launched.
- Online Service Website should keep synchronising with Online Service server for the callback response of the digital signing request from "iAM Smart" System.

Error conditions

- Nil

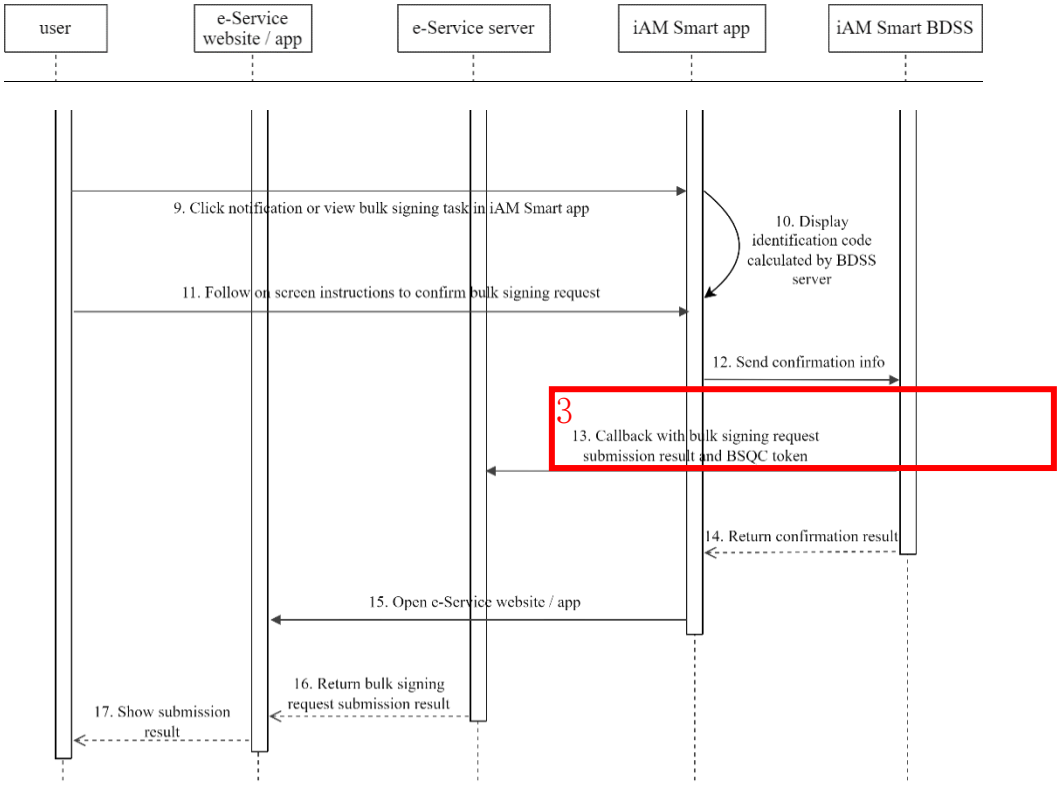
Request Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- Nil

3.11.2.3 Implementing (3) POST: *Callback to Receive BSQC Token*

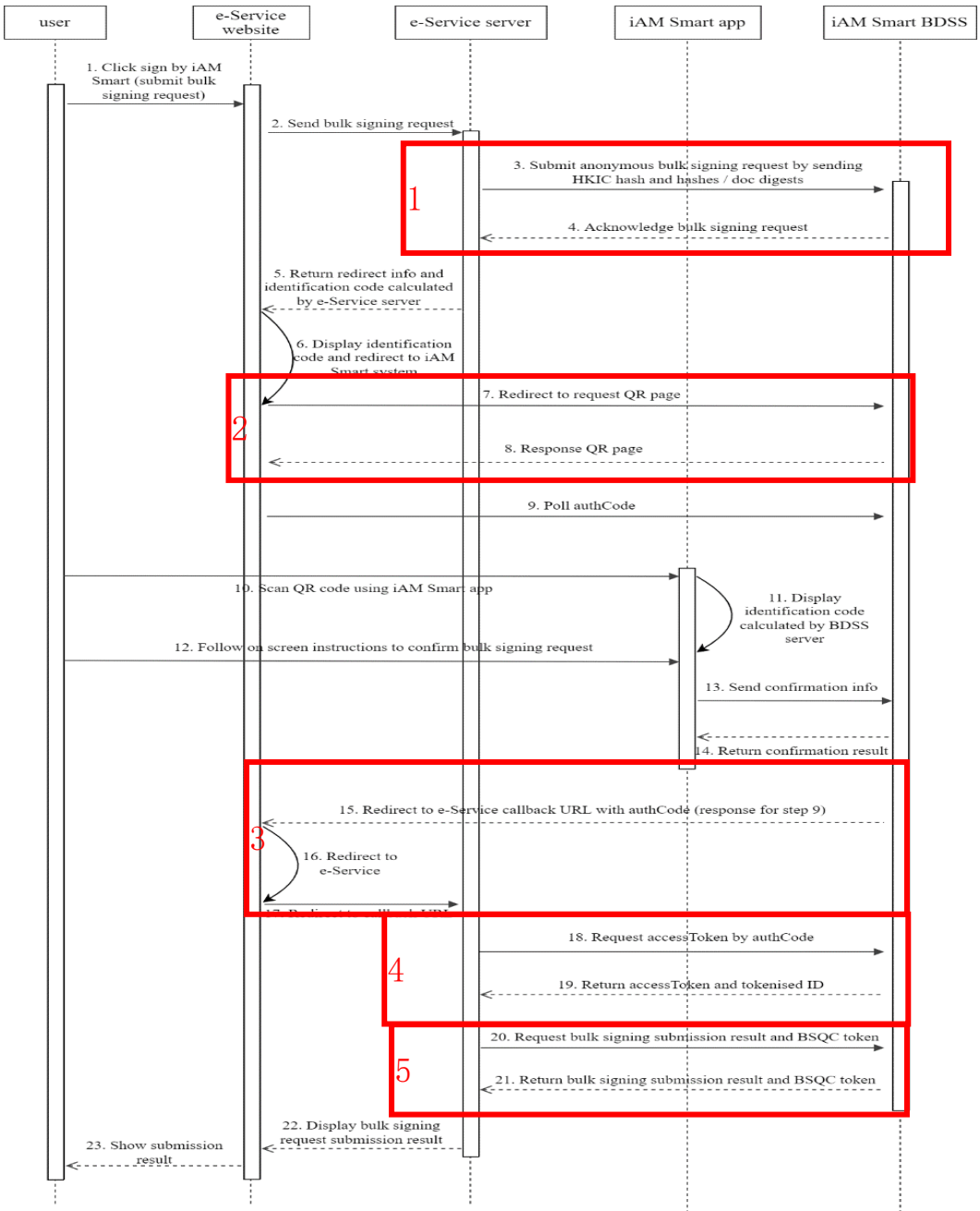


Please refer to Section 3.11.1.2.

3.12 WORKFLOWS FOR BULK DIGITAL SIGNING WITHOUT SERVICE LOGIN (AKA ANONYMOUS BULK DIGITAL SIGNING)

3.12.1 Scenario 1: Anonymous Bulk Digital Signing (Online Service Website in Different Device)

The sequence diagram below shows how an anonymous user authorises and signs multiple document hashes and/or digests when Online Service website and the "iAM Smart" Mobile App are running in different devices.



Step 1. After entering HKIC number and choose documents, the user clicks the “Anonymous bulk digital signing” button in Online Service Website;

Step 2. Online Service Website initiates anonymous bulk digital signing request to Online Service server with browser's user agent name (use as value for request parameter “source”);

Step 3. Online Service server initiates an anonymous bulk digital signing request to invoke the “iAM Smart” API with the “businessID” of this request, Document Hash / PDF digest, HKICHash, signature algorithm (only for hash document), etc. API encryption is required;

”iAM Smart” API (POST: Request Anonymous Bulk Digital Signing)

Step 4. “iAM Smart” BDSS verifies and confirms receipt of the anonymous bulk digital signing request to online service Server by returning POST response with parameter “ticketID”;

Step 5-7. Online Service Server uses the Document Hash / PDF digest, HKICHash and hash of clientID to calculate a 6-digit identification code and instructs Online Service Website to display the instructions with the 6-digit identification code to inform “iAM Smart” user to process the digital signing authorisation request in “iAM Smart” Mobile App;

At the same time, the page polls Online Service for the submission status at step 6, Online Service server prepares the request parameters such as “clientID”, “redirectURI”, “source” (set as browser’s user agent value), “scope”, “ticketID”, etc. and constructs the GET request to invoke the “iAM Smart” API using browser redirection;

“iAM Smart” API (GET: Request QR Page)

Step 8. “iAM Smart” System returns the broker page (if Online Service has set “brokerPage” to “true”) and after the broker page fails to find the “iAM Smart” Mobile App, it will request QR page to be displayed in browser. If Online Service does not request for broker page (i.e. no broker page will be returned), the browser will directly display QR Code page;

Step 9. QR Code page of "iAM Smart" System polls "iAM Smart" System for the QR Code status;

Step 10. "iAM Smart" user logs in "iAM Smart" Mobile App to scan the QR Code;

Step 11. "iAM Smart" Mobile App requests and displays 6-digit identification code from "iAM Smart" BDSS. "iAM Smart" user reviews the signing authorisation request (e.g. continue or reject the request) and verifies the 6-digit identification code with Online Service Website;

Step 12. "iAM Smart" user follows the instructions in "iAM Smart" Mobile App to complete digital signing operation;

Step 13-14. "iAM Smart" BDSS verifies the validity of QR Code and other necessary information, and return the authorisation result to "iAM Smart" Mobile App;

Step 15-17. QR Code page of "iAM Smart" System receives the polling result returned by "iAM Smart" BDSS (i.e. response for the polling in Step 9) which includes authCode and businessID or any error code (e.g. "iAM Smart" user rejects the request), assembles and invokes the Online Service callback API given in "redirectURI" using browser redirection;

Online Service Callback API (GET: Callback with authCode to Online Service Server)

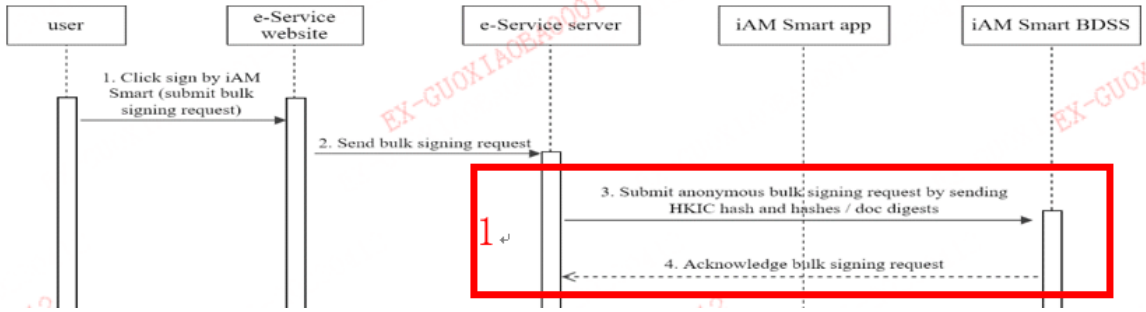
Step 18-19. When Online Service server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e. openID) of the "iAM Smart" user.API data encryption is required;

"iAM Smart" API (POST: Request accessToken and Tokenised ID)

Step 20-21. Online Service server invokes the "iAM Smart" API with the accessToken and openID to obtain BSQCToken, then Online Service invokes the "iAM Smart" API with BSQCToken and openID to obtain the bulk digital signing result. API data encryption is required;

"iAM Smart" API (POST: Request BSQC Token)

3.12.1.1 Implementing (1) POST: Request Anonymous Bulk Digital Signing



Pre-conditions

- Online Service Website/App and "iAM Smart" Mobile App are in different devices in this scenario.
- The "iAM Smart" user who signs the document must have a sign version "iAM Smart".
- Online Service generates a "businessID" for this bulk signing request and uses this identifier to match the callback result returned from "iAM Smart" BDSS.
- Online Service generates the Document Hash as "hashCode" parameter from the original documents to be signed using a hash algorithm that can suit the specific business need. It is recommended to use a hash algorithm that is at least SHA-256 or equivalent, or Online Service generates the Document Hash as "docDigest" parameter from the original documents to be signed using the Adobe.PPKLite filter and the adbe.pkcs7.detached subfilter.
- Online Service should convert the HKIC number (no check digit is needed) into hash value using SHA256.
- API data encryption is required.

Post-conditions

- Online Service server should check the response parameter "ticketID" and determine the next action. "ticketID" will not be provided by "iAM Smart" System in this scenario.
- Online Service has to calculate a 6-digit identification code using the Document Hash and Tokenised ID hash value. Online Service Website/App should show the 6-digit identification code and instruction to inform "iAM Smart" user to process the bulk digital signing request in "iAM Smart" Mobile App when QR Code is scanned.
- 6-digit identification code generation algorithm refer to Section 3.11.1.1.
- Online Service Website/App should keep synchronising with Online Service server for the bulk digital signing result returned from "iAM Smart" BDSS.

- API data decryption is required.

Error conditions

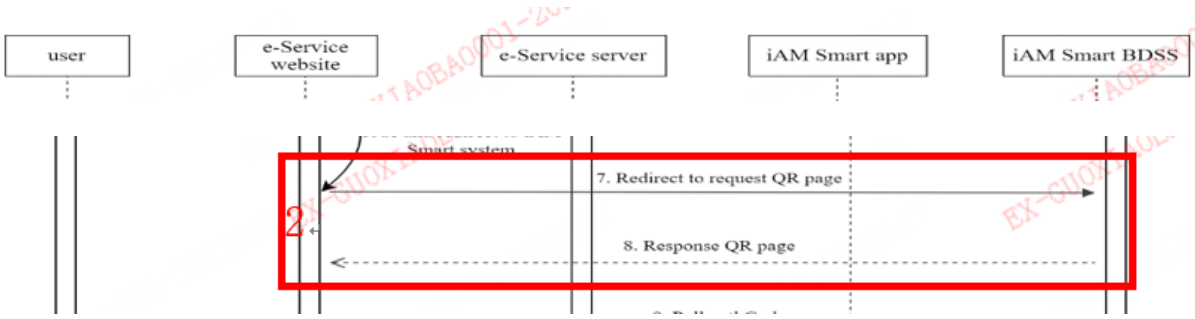
- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D70002	Failed to request signing	Inform user the "iAM Smart" digital signing request is failed and retry later

Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

3.12.1.2 Implementing (2) GET: Request QR Page



Pre-conditions

- Please refer to Section 3.4.1.1 except:
  - Online Service has the "ticketID" provided by "iAM Smart" System for the anonymous request in step 4.

Post-conditions

- Please refer to Section 3.4.1.1.

Error conditions

- Please refer to Section 3.4.1.1.

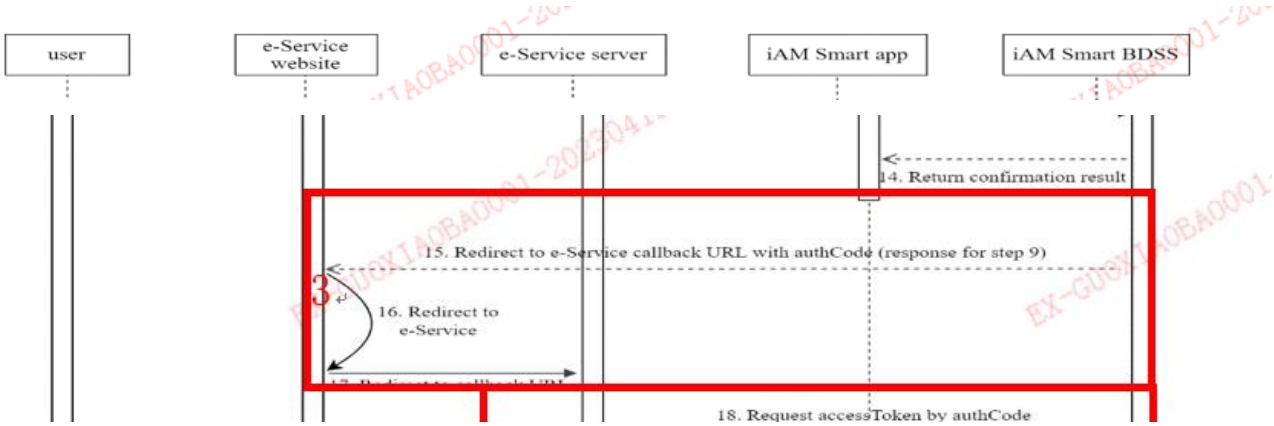
Request Parameters

- Please refer to Section 3.4.1.1.

Notes

- Please refer to Section 3.4.1.1 except the following parameter:
  - Request parameter "ticketID": Value provided by "iAM Smart" System for the anonymous bulk digital signing request.

3.12.1.3 Implementing (3) GET: Callback with authCode to Online Service Server



Pre-conditions

- Please refer to Section 3.4.1.2.

Post-conditions

- Please refer to Section 3.4.1.2 except:
  - Online Service server should match the callback result with corresponding Online Service client terminal using the "businessID".

Error conditions

- This Online Service callback API is a HTTP GET request. If parameter "error_code" is returned, it means the request is failed.

Error Code	Error Description	Suggested Action
error_code - D71001	User rejected signing request	Inform user the digital signing request is rejected
error_code - D71002	Failed to request signing	Inform user the digital signing request is failed and retry later
error_code - D71004	User not allowed to sign	Inform user that "iAM Smart" digital signing is not allowed for default "iAM Smart" and suggest user to upgrade to "iAM Smart" with digital signing function
error_code - D71005	Inconsistent HKIC number	Hash of HKIC number provided is not matched with "iAM Smart" user. Check hash of HKIC number or inform user to authorise the digital signing using the correct "iAM Smart" account

The error_code D71003 (signing request timeout) does not appear in this scenario. For the QR Code timeout or request confirmation timeout in the "iAM Smart" Mobile App, message will be prompted in the corresponding user interface.

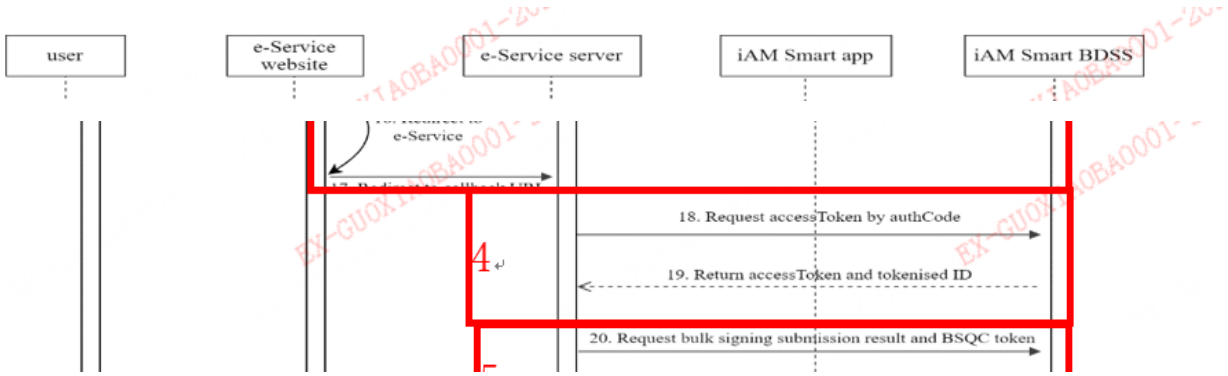
Callback Parameters

- Please refer to Section 3.4.1.2.

Notes

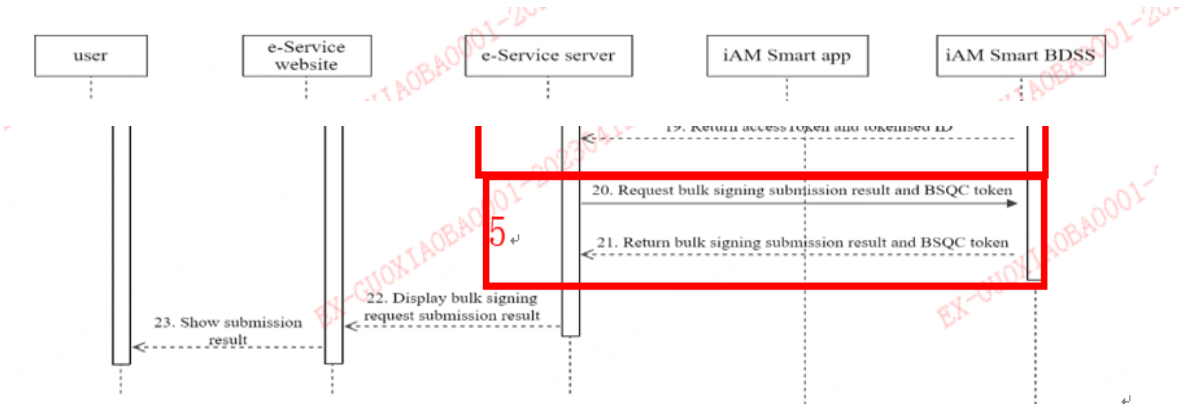
- Please refer to Section 3.4.1.2 except the following parameter:
  - Callback parameter "businessID": Value returned by "iAM Smart" System should be the same one as Online Service submitted in the anonymous request.

3.12.1.4 Implementing (4) POST: Request accessToken and Tokenised ID



Please refer to Section 3.6.1.4

3.12.1.5 Implementing (5) POST: Request BSQC Token



Pre-conditions

- Online Service provides parameters "accessToken" and "openID".
- API data encryption is required.

Post-conditions

- Online Service has obtained "accessToken" and "openID" by previous request

Error conditions

Error Code	Error Description	Suggested Action
error_code - D20012	insufficient scope	Check authorisation scope(s) requested is/are sufficient for invoking the corresponding "iAM Smart" API

Request Parameters

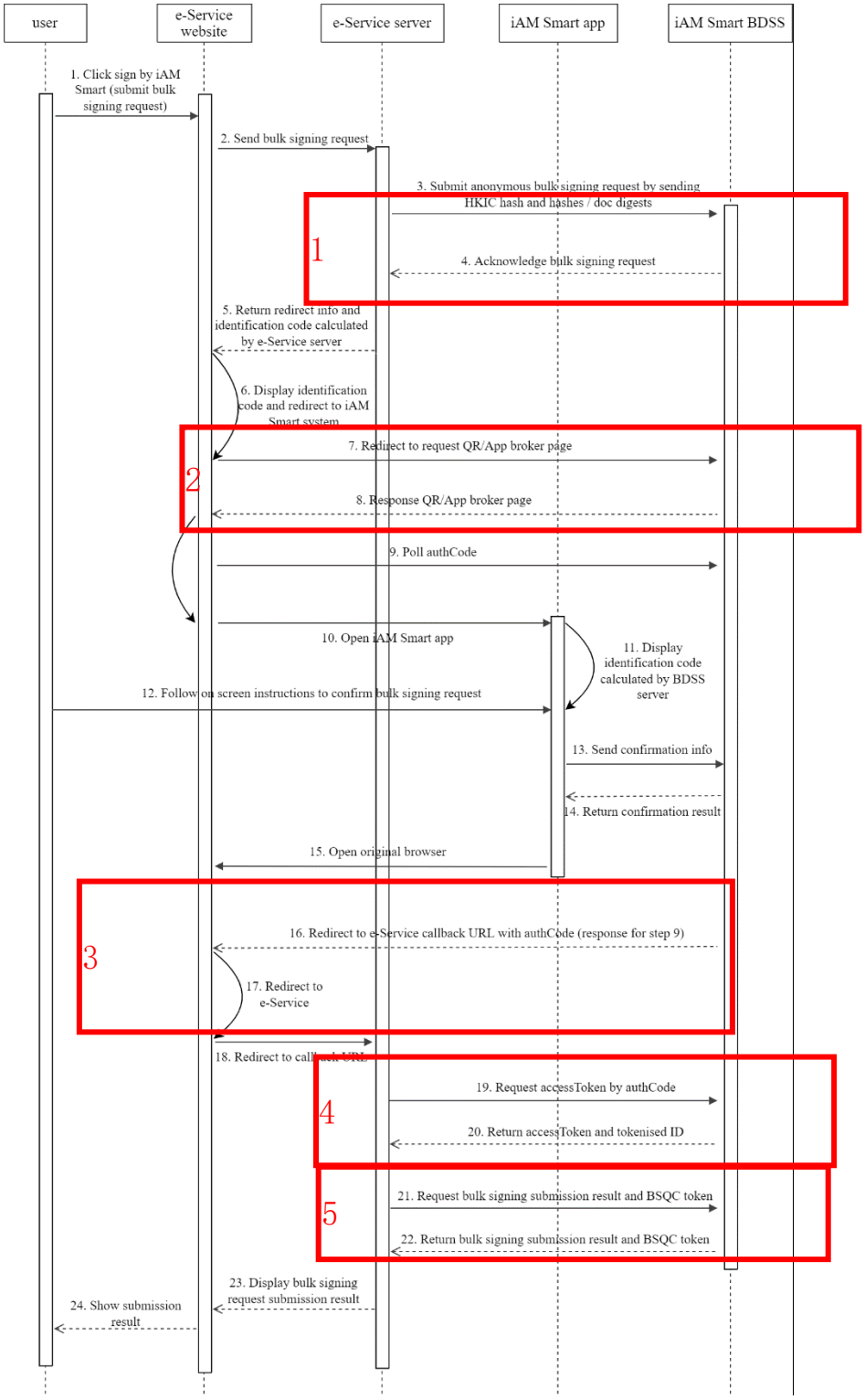
Parameter	Type	Presence	Description
`accessToken`	String	Required	`accessToken` value
`openID`	String	Required	Tokenised ID value

Response Parameters

Parameter	**Type**	**Presence**	**Description**
`BSQCToken`	String	Required	`BSQCToken` can be used by the corresponding Online Service to enquire or cancel the submitted bulk digital signing request

3.12.2 Scenario 2: Anonymous Bulk Digital Signing (Online Service Website in Same Device)

The sequence diagram below shows how an anonymous user authorises and signs multiple document hashes and/or digests when Online Service website and the "iAM Smart" Mobile App are running in the same devices.



Step 1. After entering HKIC number and choose documents, the user clicks the “Anonymous bulk digital signing” button in Online Service Website;

Step 2. Online Service Website initiates anonymous bulk digital signing request to Online Service server with browser's user agent name (use as value for request parameter “source”);

Step 3. Online Service server initiates an anonymous bulk digital signing request to invoke the “iAM Smart” API with the “businessID” of this request, Document Hash / PDF digest, HKICHash, signature algorithm (only for hash document), etc. API encryption is required;

”iAM Smart” API (POST: Request Anonymous Bulk Digital Signing)

Step 4. “iAM Smart” BDSS verifies and confirms receipt of the anonymous bulk digital signing request to Online Service server by returning POST response with parameter “ticketID”;

Step 5-7. Online Service server uses the Document Hash / PDF digest, HKICHash and hash of clientID to calculate a 6-digit identification code and instructs Online Service Website to display the instructions with the 6-digit identification code to inform “iAM Smart” user to process the digital signing authorisation request in “iAM Smart” Mobile App;

At the same time, the page polls Online Service for the submission status at step 6, Online Service server prepares the request parameters such as “clientID”, “redirectURI”, “source” (set as browser’s user agent value), “scope”, “ticketID”, etc. and constructs the GET request to invoke the “iAM Smart” API using browser redirection;

“iAM Smart” API (GET: Request QR Page)

Step 8. “iAM Smart” System returns the broker page to the Online Service Website;

Step 9. Broker page of “iAM Smart” System polls “iAM Smart” System for further action;

Step 10. Broker page invokes the “iAM Smart” Mobile App in the same device automatically and sends it the relevant parameters.

Step 11. "iAM Smart" Mobile App requests and displays 6-digit identification code from "iAM Smart" BDSS. "iAM Smart" user reviews the signing authorisation request (e.g. continue or reject the request) and verifies the 6-digit identification code with Online Service Website;

Step 12. "iAM Smart" user follows the instructions in "iAM Smart" Mobile App to complete digital signing operation;

Step 13-15. "iAM Smart" BDSS verifies the validity of QR Code and other necessary information, and returns the authorisation result to "iAM Smart" Mobile App, "iAM Smart" Mobile App invokes the original Online Service browser;

Step 16-18. Broker page of "iAM Smart" System receives the polling result returned by "iAM Smart" BDSS (i.e. response for the polling in Step 9) which includes authCode and businessID or any error code (e.g. "iAM Smart" user rejects the request), assembles and invokes the Online Service callback API given in "redirectURI" using browser redirection;

Online Service Callback API (GET: Callback with authCode to Online Service Server)

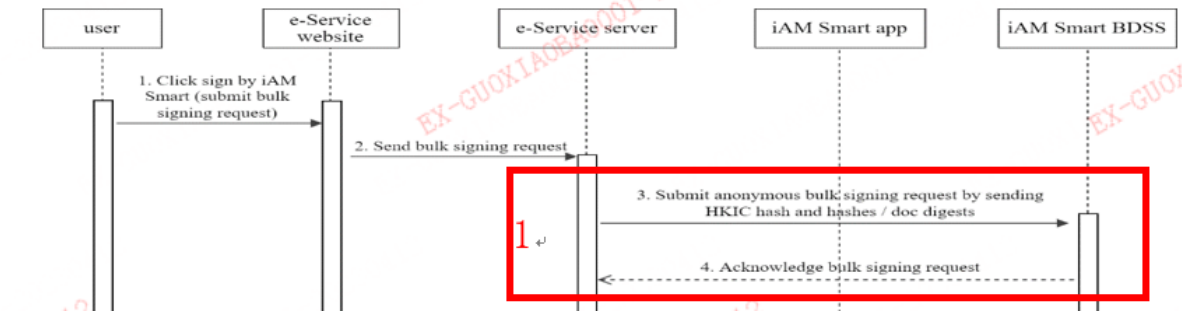
Step 18-19. When Online Service server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invokes the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e. openID) of the "iAM Smart" user. API data encryption is required;

"iAM Smart" API (POST: Request accessToken and Tokenised ID)

Step 20-21. Online Service server invokes the "iAM Smart" API with the accessToken and openID to obtain BSQCToken, then Online Service invokes the "iAM Smart" API with BSQCToken and openID to obtain the bulk digital signing result. API data encryption is required;

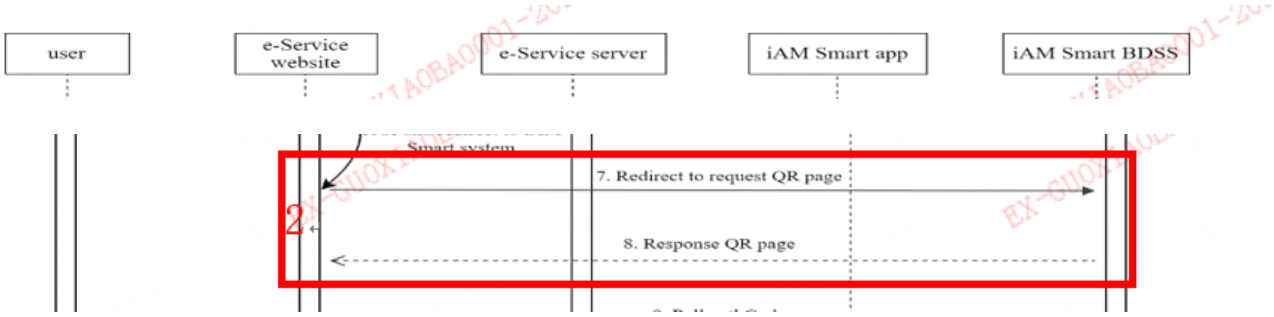
"iAM Smart" API (POST: Request BSQC Token)

3.12.2.1 Implementing (1) POST: Request Anonymous Bulk Digital Signing



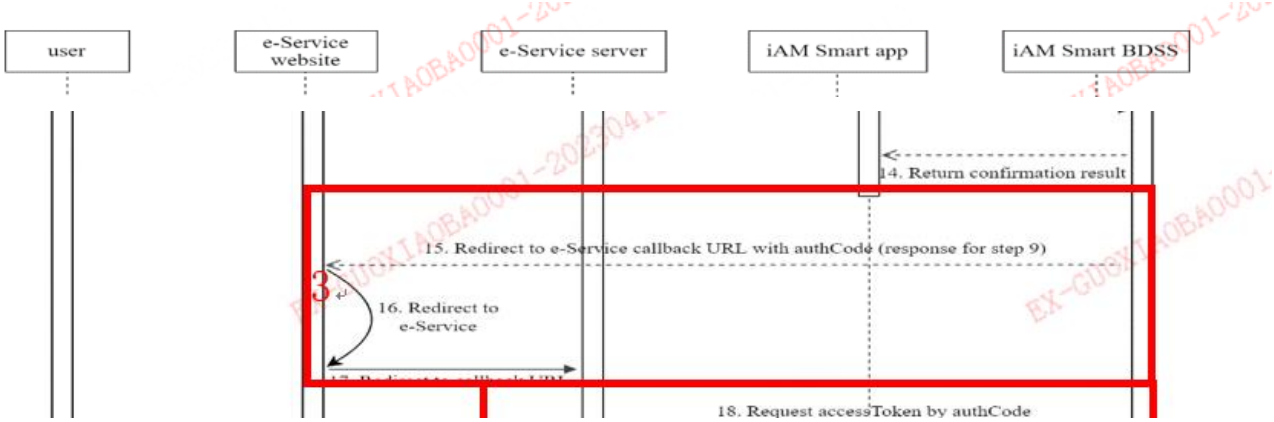
Please refer to Section 3.12.1.1

3.12.2.2 Implementing (2) GET: Request QR Page



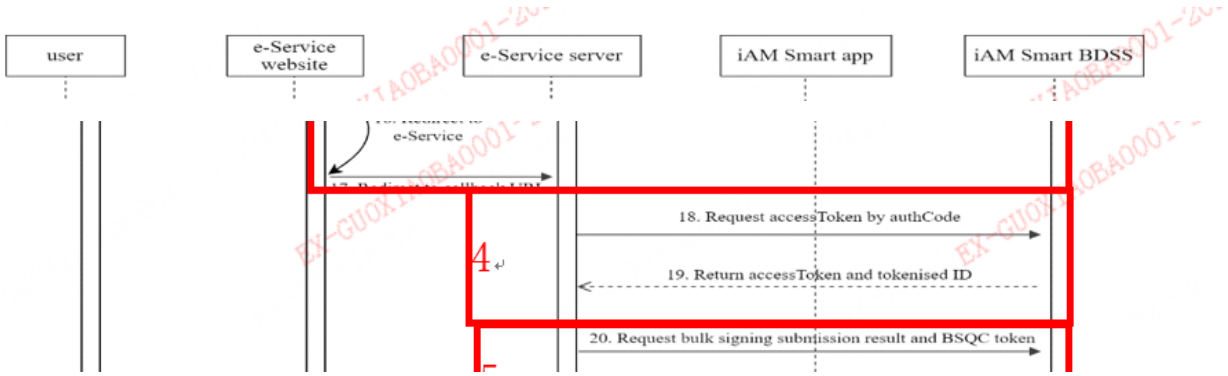
Please refer to Section 3.12.1.2

3.12.2.3 Implementing (3) GET: Callback with authCode to Online Service Server



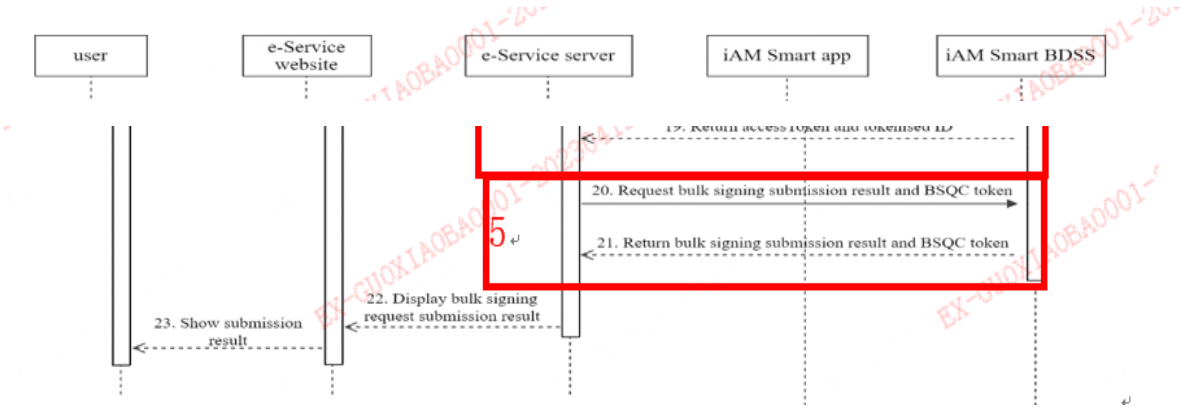
Please refer to Section 3.12.1.3

3.12.2.4 Implementing (4) POST: Request accessToken and Tokenised ID



Please refer to Section 3.6.1.4

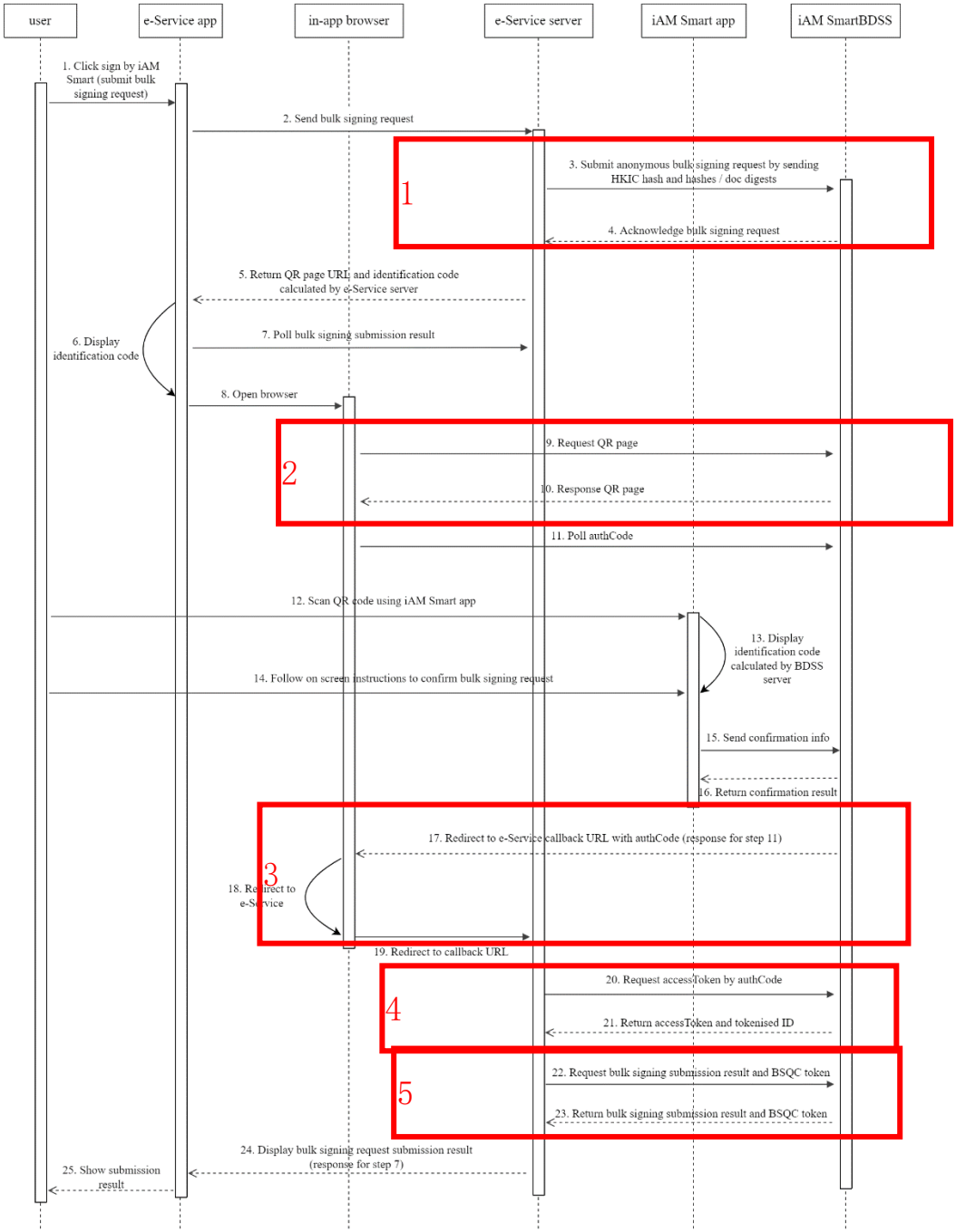
3.12.2.5 Implementing (5) POST: Request BSQC Token



Please refer to Section 3.12.1.5

3.12.3 Scenario 3: Anonymous Bulk Digital Signing (Online Service App in Different Device)

The sequence diagram below shows how an anonymous user authorises and signs multiple document hashes and/or digests when Online Service App and the "iAM Smart" Mobile App are running in different devices.



Step 1. After entering HKIC number and choose documents, the user clicks the “Anonymous bulk digital signing” button in Online Service Website;

Step 2. Online Service Website initiates anonymous bulk digital signing request to Online Service server with browser's user agent name (use as value for request parameter “source”);

Step 3. Online Service server initiates an anonymous bulk digital signing request to invoke the “iAM Smart” API with the “businessID” of this request, Document Hash / PDF digest, HKICHash, signature algorithm (only for hash document), etc. API encryption is required;

”iAM Smart” API (POST: Request Anonymous Bulk Digital Signing)

Step 4. “iAM Smart” BDSS verifies and confirms receipt of the anonymous bulk digital signing request to Online Service server by returning POST response with parameter “ticketID”;

Step 5-6. Online Service server uses the Document Hash / PDF digest, HKICHash and hash of clientID to calculate a 6-digit identification code and instructs Online Service Website to display the instructions with the 6-digit identification code to inform “iAM Smart” user to process the digital signing authorisation request in “iAM Smart” Mobile App;

At the same time, the page polls Online Service for the submission status at step 6, Online Service server prepares the request parameters such as “clientID”, “redirectURI”, “source” (set as browser’s user agent value), “scope”, “ticketID”, etc. and constructs the GET request to invoke the “iAM Smart” API using browser redirection;

“iAM Smart” API (GET: Request QR Page)

Step 7. Online Service App polls Online Service server for the digital signing result from “iAM Smart” BDSS;

Step 8. Online Service App invokes in-app browser (Safari for iOS, Chrome for Android) to submit the GET request;

Step 9-10. Browser opens the GET request to invoke the “iAM Smart” API and QR Code page will be shown;

"iAM Smart" API (GET: Request QR page)

Step 11. QR Code page of "iAM Smart" System polls "iAM Smart" System for the QR Code status;

Step 12. "iAM Smart" user logs in "iAM Smart" Mobile App to scan the QR Code;

Step 13. "iAM Smart" Mobile App requests and displays 6-digit identification code from "iAM Smart" BDSS. "iAM Smart" user reviews the signing authorisation request (e.g. continue or reject the request) and verifies the 6-digit identification code with Online Service Website;

Step 14. "iAM Smart" user follows the instructions in "iAM Smart" Mobile App to complete digital signing operation;

Step 15-16. "iAM Smart" BDSS verifies the validity of QR Code and other necessary information, and return the authorisation result to "iAM Smart" Mobile App;

Step 17-19. QR Code page of "iAM Smart" System receives the polling result returned by "iAM Smart" System (i.e. response for the polling in Step 9) which includes authCode and businessID or any error code (e.g. "iAM Smart" user rejects the request), assembles and invokes the Online Service callback API given in "redirectURI" using browser redirection;

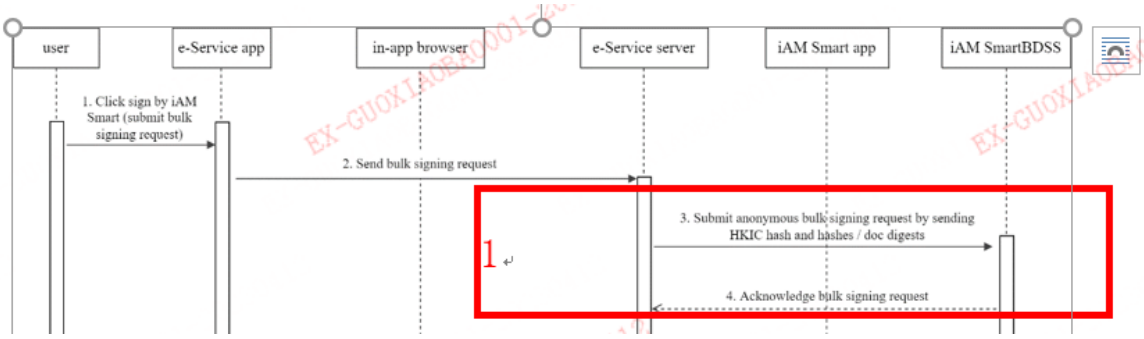
Online Service Callback API (GET: Callback with authCode to Online Service Server)

Step 20-21. When Online Service server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invokes the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e. openID) of the "iAM Smart" user. API data encryption is required;

"iAM Smart" API (POST: Request accessToken and Tokenised ID)

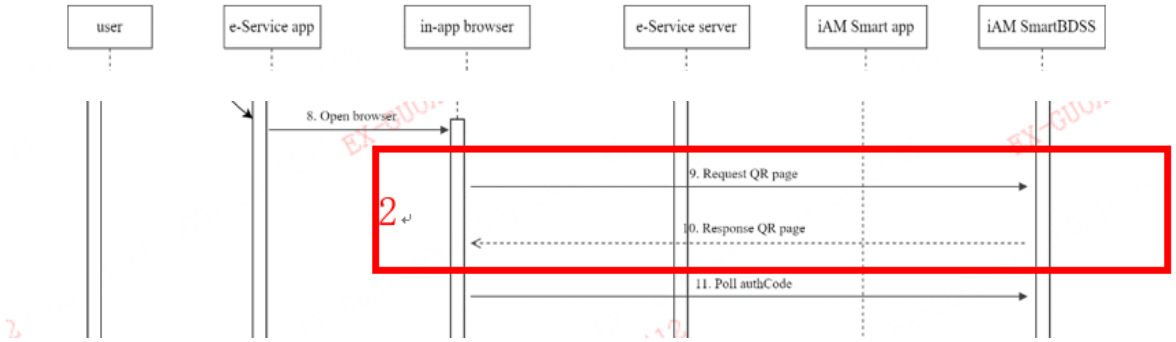
Step 22-23. Online Service server invokes the "iAM Smart" API with the accessToken and openID to obtain BSQCToken, then Online Service invokes the "iAM Smart" API with BSQCToken and openID to obtain the bulk digital signing result. API data encryption is required;

3.12.3.1 Implementing (1) POST: Request Anonymous Bulk Digital Signing



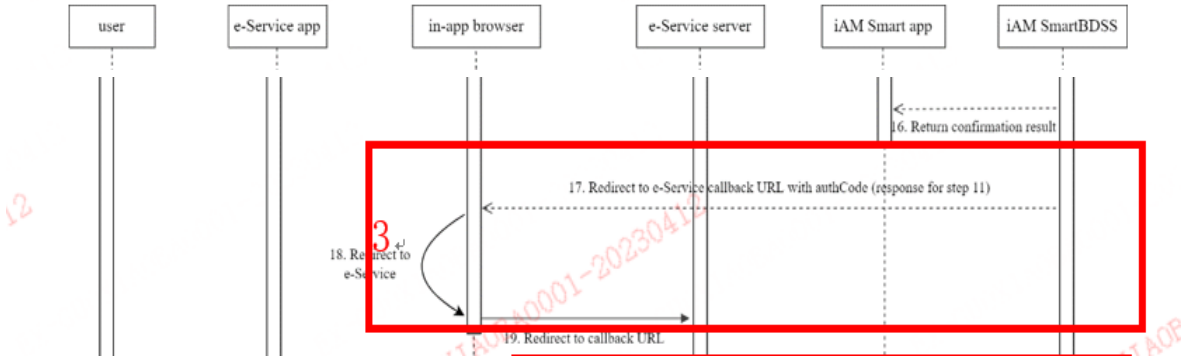
Please refer to Section 3.12.1.1

3.12.3.2 Implementing (2) GET: Request QR Page



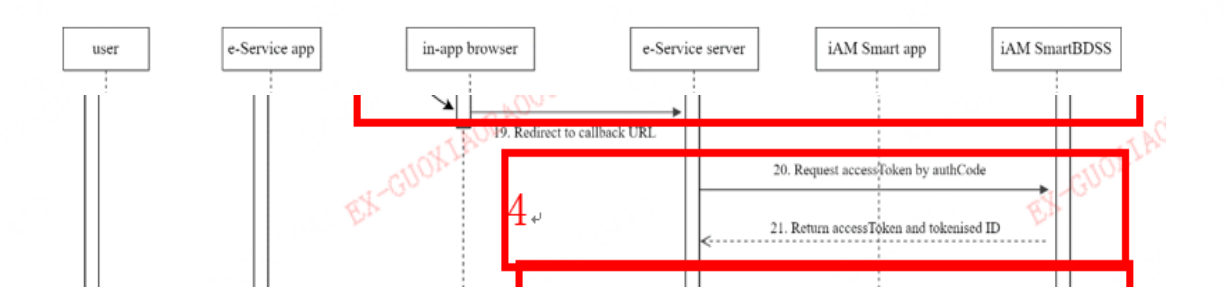
Please refer to Section 3.12.1.2

3.12.3.3 Implementing (3) GET: Callback with authCode to Online Service Server



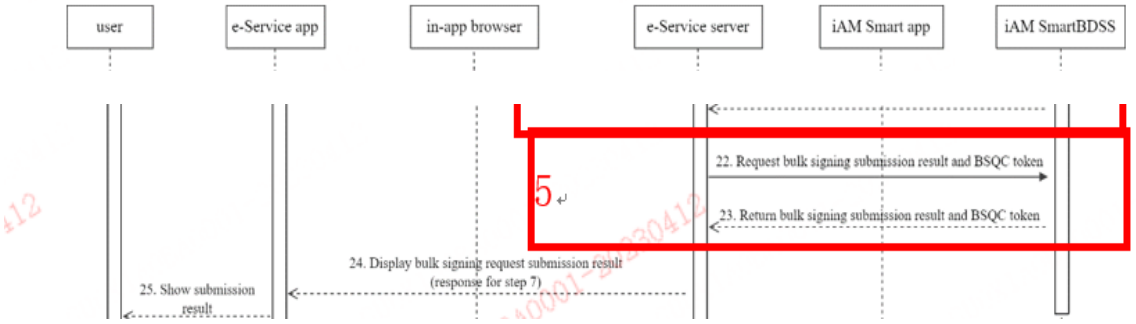
Please refer to Section 3.12.1.3

3.12.3.4 Implementing (4) POST: Request accessToken and Tokenised ID



Please refer to Section 3.6.1.4

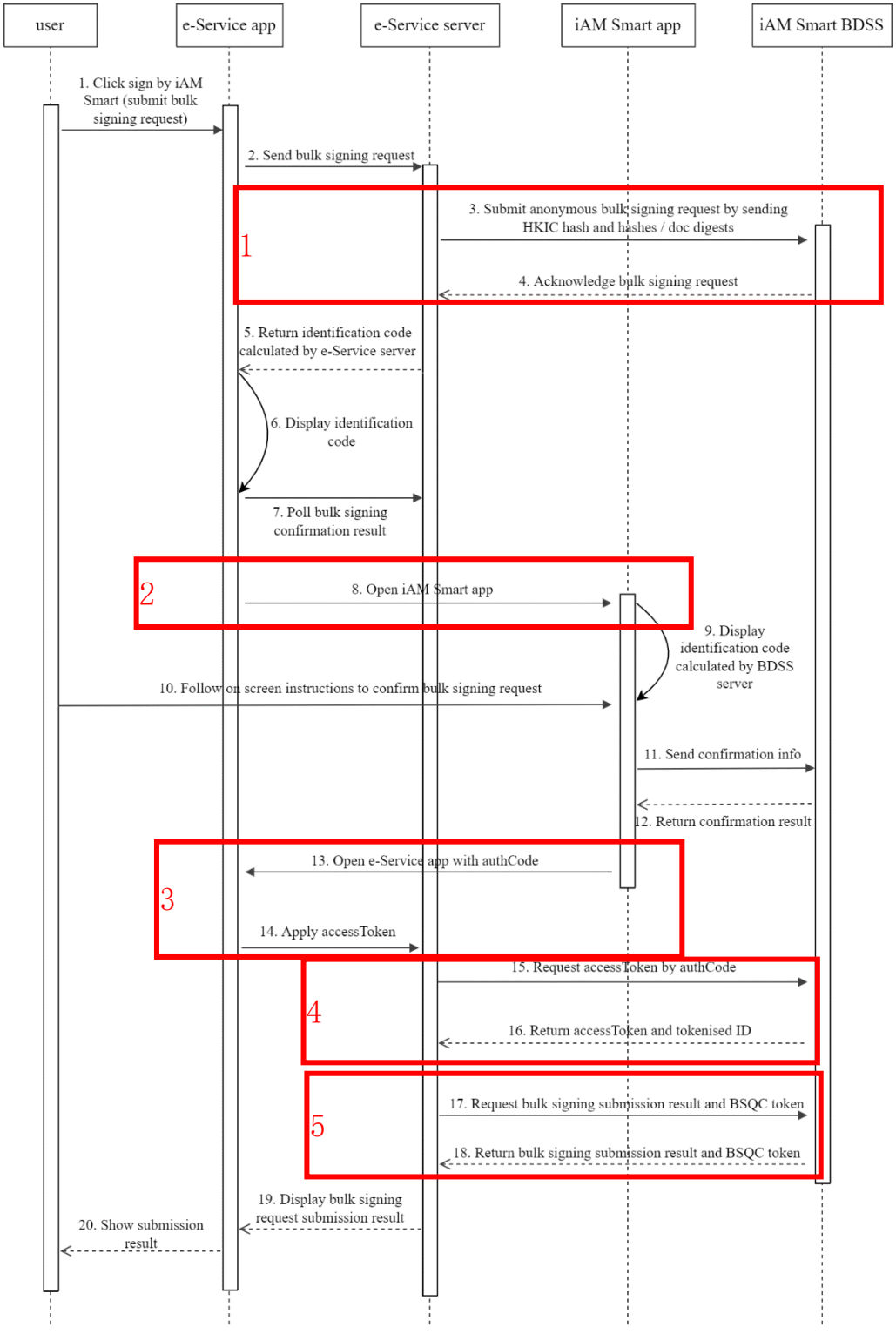
3.12.3.5 Implementing (5): POST: Request BSQC Token



Please refer to Section 3.12.1.5

3.12.4 Scenario 4: Anonymous Bulk Digital Signing (Online Service App in Same Device)

The sequence diagram below shows how an anonymous user authorises and signs multiple document hashes and/or digests when Online Service App and the "iAM Smart" Mobile App are running in the same device.



Step 1. After entering HKIC number and choose documents, the user clicks the “Anonymous bulk digital signing” button in Online Service Website;

Step 2. Online Service Website initiates anonymous bulk digital signing request to Online Service server with browser's user agent name (use as value for request parameter “source”);

Step 3. Online Service server initiates an anonymous bulk digital signing request to invoke the “iAM Smart” API with the “businessID” of this request, Document Hash / PDF digest, HKICHash, signature algorithm (only for hash document), etc. API encryption is required;

”iAM Smart” API (POST: Request Anonymous Bulk Digital Signing)

Step 4. “iAM Smart” BDSS verifies and confirms receipt of the anonymous bulk digital signing request to Online Service server by returning POST response with parameter “ticketID”;

Step 5&6. Online Service server uses the Document Hash / PDF digest, HKICHash and hash of clientID to calculate a 6-digit identification code and instructs Online Service App to display the instructions with the 6-digit identification code to inform “iAM Smart” user to process the digital signing authorisation request in “iAM Smart” Mobile App;

Online Service server prepares necessary parameters such as “clientID”, “redirectURI” (set as Online Service App URL Scheme or Universal Link/App Link depending on “source” parameter), “scope”, “source” (set as “App_Scheme” or “App_Link”), “ticketID”, etc. and constructs the URL Scheme to invoke “iAM Smart” Mobile App by Online Service App;

Step 7. Online Service App polls Online Service server for the digital signing result from “iAM Smart” BDSS;

Step 8. Online Service App invokes “iAM Smart” Mobile App using URL Scheme with request parameters (set <Context> as “anon_bulk-sign” in URL Scheme);

“iAM Smart” API (URL Scheme: Open “iAM Smart” Mobile App for Getting Context)

Step 9. "iAM Smart" user logs in "iAM Smart" Mobile App, "iAM Smart" Mobile App requests and displays 6-digit identification code from "iAM Smart" BDSS. "iAM Smart" user reviews the digital signing authorisation request (e.g. continue or reject the request) and verifies the 6-digit identification code with Online Service Website;

Step 10. "iAM Smart" user follows the instructions in "iAM Smart" Mobile App to complete digital signing operation;

Step 11-12. "iAM Smart" BDSS verifies the validity of QR Code and other necessary information, and return the authorisation result to "iAM Smart" Mobile App;

Step 13. "iAM Smart" Mobile App invokes Online Service App using URL Scheme or Universal Link/App Link with required parameters such as "authCode", "businessID";

Online Service Callback API (GET: Callback with authCode to Online Service App)

Step 14. Online Service App sends authCode, businessID, etc. to Online Service server;

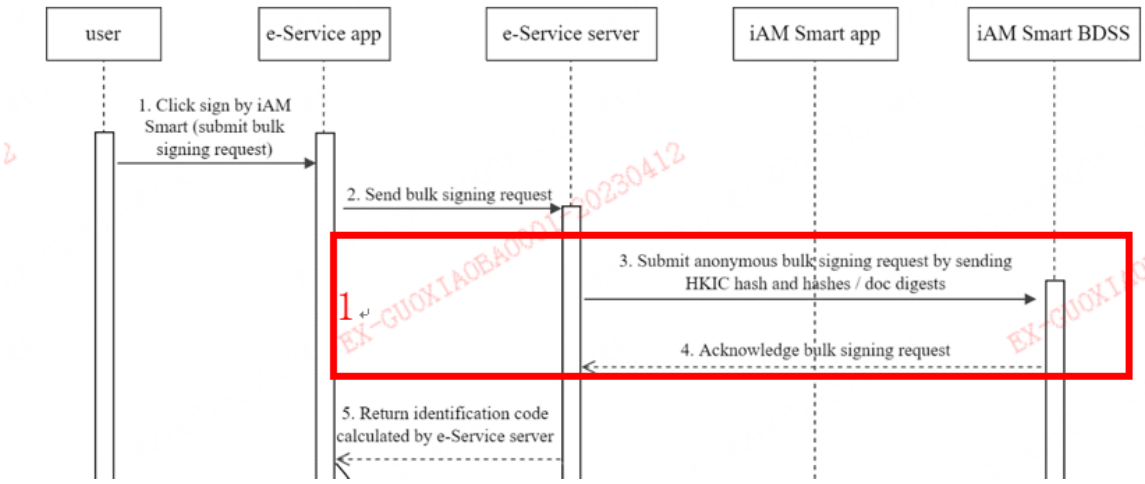
Step 15-16. When Online Service server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e. openID) of the "iAM Smart" user. API data encryption is required;

"iAM Smart" API (POST: Request accessToken & Tokenised ID)

Step 17-8. Online Service server invokes the "iAM Smart" API with the accessToken and openID to obtain BSQCToken, then Online Service invokes the "iAM Smart" API with BSQCToken and openID to obtain the bulk digital signing result. API data encryption is required;

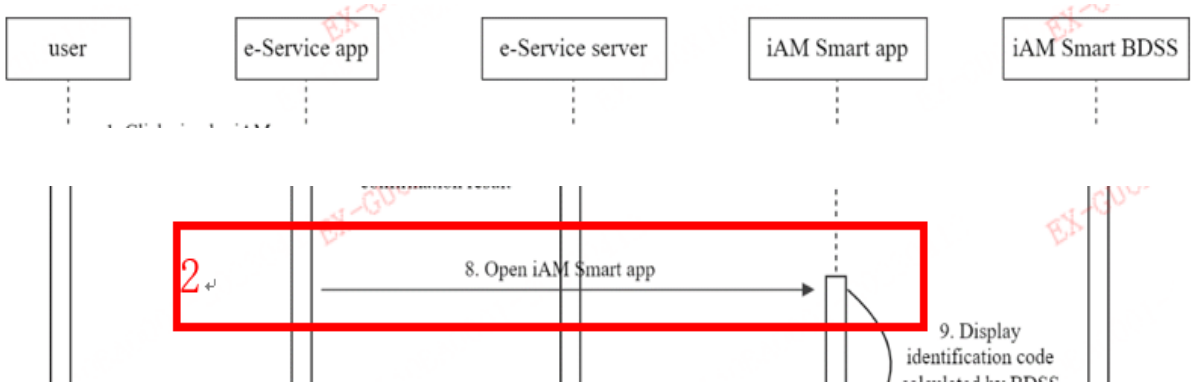
"iAM Smart" API (POST: Request BSQC Token)

3.12.4.1 Implementing (1) POST: Request Anonymous Bulk Digital Signing



Please refer to Section 3.12.1.1

3.12.4.2 Implementing (2) URL Scheme: Open "iAM Smart" Mobile App for Getting Context



Pre-conditions

- Please refer to Section 3.4.1.2.

Post-conditions

- Please refer to Section 3.4.1.2.

Error conditions

- Please refer to Section 3.4.1.2.

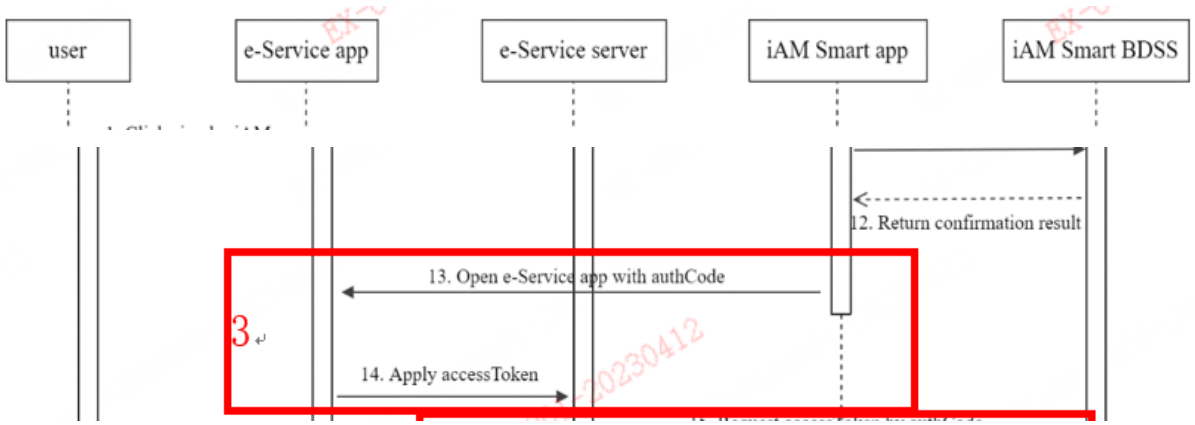
Request Parameters

- Please refer to Section 3.4.1.2.

Notes

- Please refer to Section 3.4.1.2, except:
  - Set <Context> as "anon_bulk-sign" in URL Scheme.

3.12.4.3 Implementing (3) GET: Callback with authCode to Online Service Server



Pre-conditions

- Please refer to Section 3.4.1.2.

Post-conditions

- Please refer to Section 3.4.1.2 except:
  - Online Service server should match the callback result with corresponding Online Service client terminal using the "businessID".

Error conditions

- This Online Service callback API is a HTTP GET request. If parameter "error_code" is returned, it means the request is failed.

Error Code	Error Description	Suggested Action
error_code - D71001	User rejected signing request	Inform user the digital signing request is rejected
error_code - D71002	Failed to request signing	Inform user the digital signing request is failed and retry later
error_code - D71004	User not allowed to sign	Inform user that "iAM Smart" digital signing is not allowed for default "iAM Smart" and suggest user to upgrade to "iAM Smart" with digital signing function

Error Code	Error Description	Suggested Action
error_code - D71005	Inconsistent HKIC number	Hash of HKIC number provided is not matched with "iAM Smart" user. Check hash of HKIC number or inform user to authorise the digital signing using the correct "iAM Smart" account

The error_code D71003 (signing request timeout) does not appear in this scenario. For the QR Code timeout or request confirmation timeout in the "iAM Smart" Mobile App, message will be prompted in the corresponding user interface.

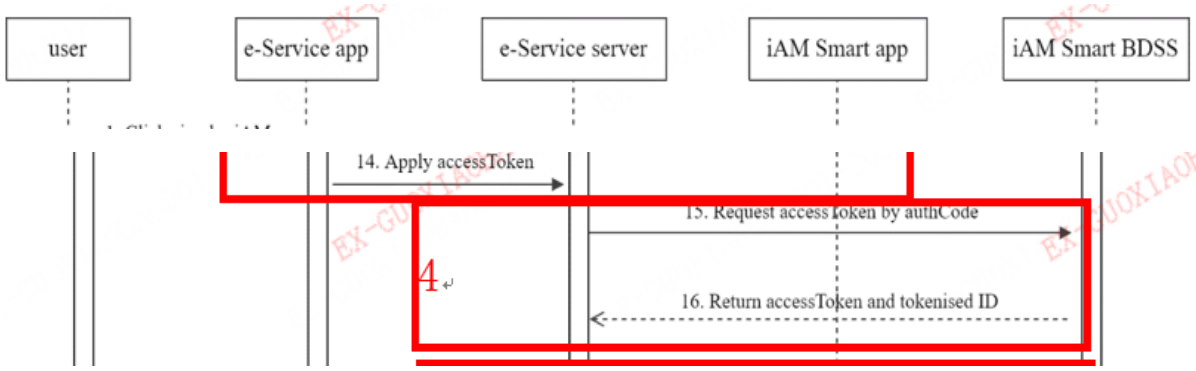
Callback Parameters

- Please refer to Section 3.4.1.2.

Notes

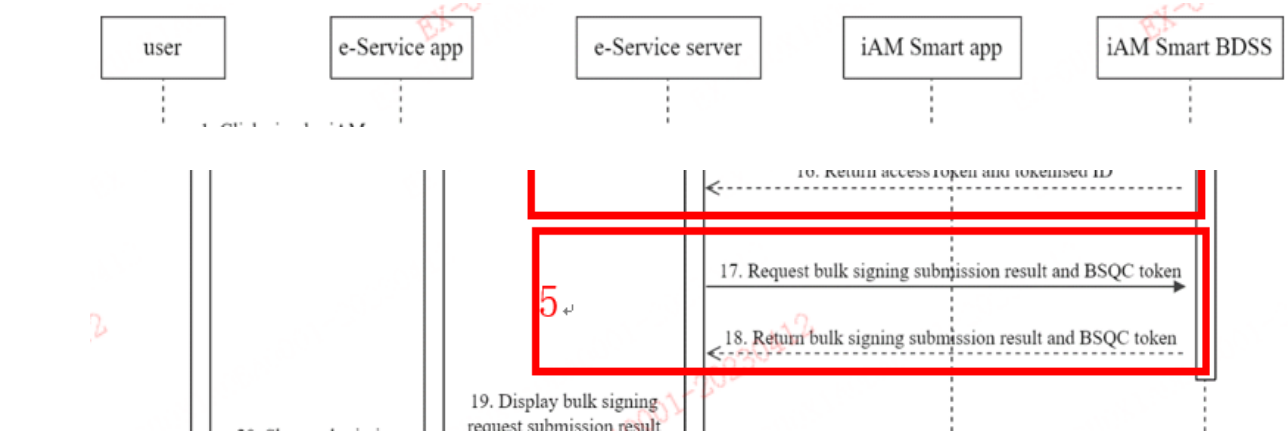
- Please refer to Section 3.4.1.2 except the following parameter:
  - Callback parameter "businessID": Value returned by "iAM Smart" System should be the same one as Online Service submitted in the anonymous request.

3.12.4.4 Implementing (4) POST: Request accessToken and Tokenised ID



Please refer to Section 3.6.1.4

3.12.4.5 Implementing (5) POST: Request BSQC Token

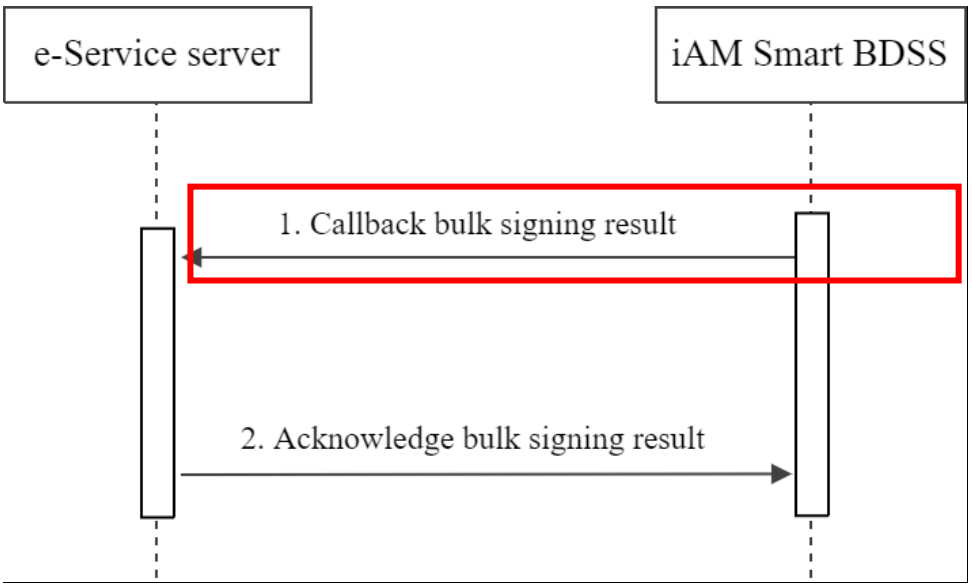


Please refer to Section 3.12.1.5

3.13 WORKFLOWS FOR CALLBACK BULK DIGITAL SIGNING RESULT

3.13.1 Workflows for Callback Bulk Digital Signing Result

3.13.1.1 Implementing (1) POST: Callback to Receive Bulk Digital Signing Result



Pre-conditions

- "iAM Smart" user can accept and complete the digital signing request.
- "iAM Smart" user can also reject the digital signing request.

Post-conditions

- API data decryption is required.
- Online Service server should match the callback result with corresponding Online Service Website/App using the "businessID".

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D71000	User cancelled signing request	Inform user the "iAM Smart" digital signing request is cancelled

code - D71001	User rejected signing request	Inform user the “iAM Smart” digital signing request is rejected
code - D71002	Failed to request signing	Inform user the “iAM Smart” digital signing request is failed and retry later
code - D71003	Signing request timeout	Inform user the “iAM Smart” digital signing request is timeout and provide way for user to retry

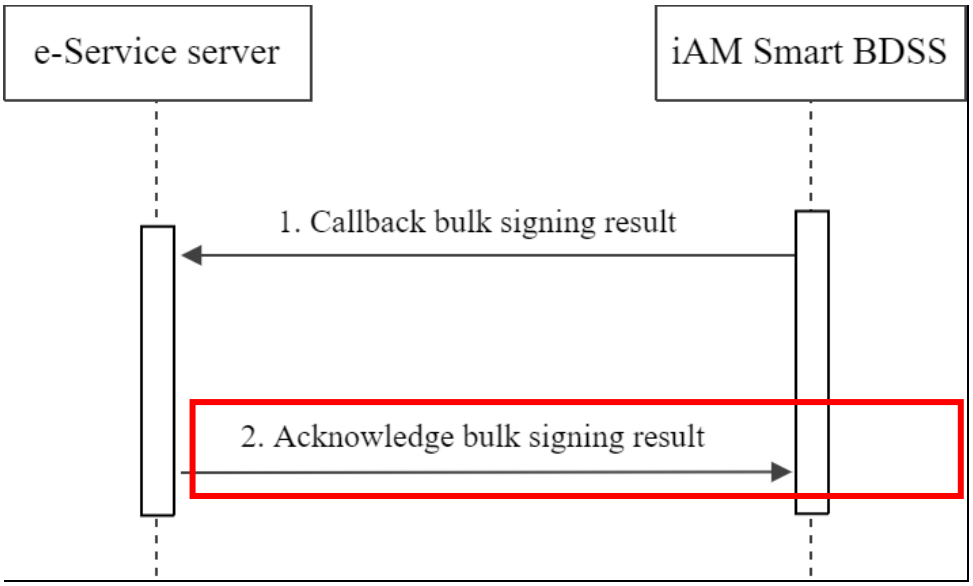
Callback Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- Callback parameter “hashCode”: It is the Document Hash provided by Online Service and Online Service may use for checking purpose.
- Callback parameter “timestamp”: It is the timestamp returned by “iAM Smart” System when the digital signing operation is successful.
- Callback parameter “signature”: It is the RSA signature of the Document Hash submitted by Online Service for digital signing. The value is BASE64 encoded.
- Callback parameter “cert”: It is the “iAM Smart” e-Cert of the “iAM Smart” user. It is in X.509 format and the value is BASE64 encoded.

3.13.1.2 Implementing (2) POST: Online Service Acknowledges Bulk Digital Signing Result



Pre-conditions

- Online Service receives the "iAM Smart" bulk digital signing result and completes its document digital signing process.
- API data encryption is required.

Post-conditions

- Online Service may record the bulk digital signing acknowledgement result.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D71006	Failed to process digital signing acknowledgement	"iAM Smart" digital signing acknowledgement processing failed, retry later

Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

Notes

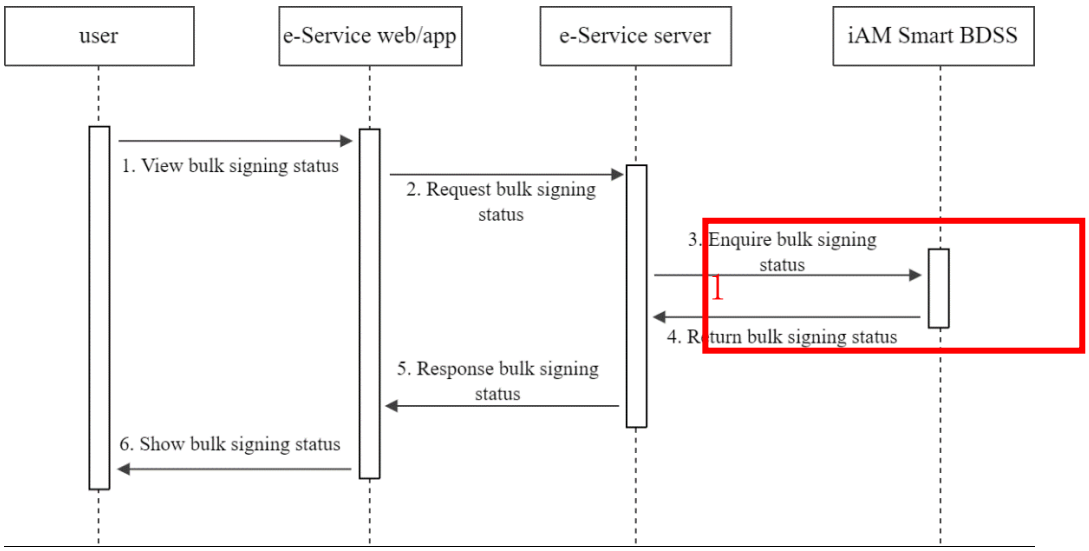
- For common parameters in request and response, please refer to Section 3.2.2.

- Request parameter “businessID”: It is the same “businessID” submitted in the digital signing request.
- Request parameter “signingResult”: It is the status of Online Service document digital signing process after receiving the “iAM Smart” digital signing result. Its value must be equal to those specified in this API.

3.14 WORKFLOWS FOR ENQUIRE BULK DIGITAL SIGNING RESULT

3.14.1 Workflows for Enquire Bulk Digital Signing Result

3.14.1.1 Implementing (1) POST: Enquire Bulk Digital Signing Status



Pre-conditions

- Online Service provides parameters “openID” and “BSQCToken”.
- API data encryption is required.

Post-conditions

- API data encryption is required.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D71008	BSQC Token not found	BSQC Token may expire

Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

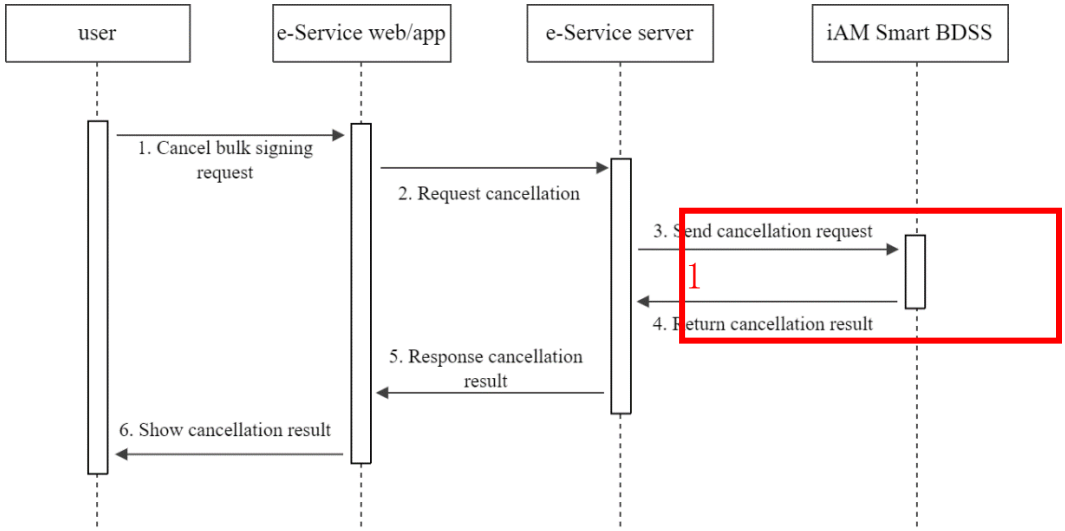
Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- The BSQC Token that’s used to enquire digital signing status or cancel signing request is valid for 1 day.

3.15 WORKFLOWS FOR CANCEL BULK DIGITAL SIGNING REQUEST

3.15.1 Workflows for Cancel Bulk Digital Signing Request

3.15.1.1 Implementing (1) POST: Cancel Bulk Digital Signing Request



Pre-conditions

- Online Service provides parameters “openID” and “BSQCToken”.
- The request of which status is “Pending for Signing” can be cancelled.
- API data encryption is required.

Post-conditions

- API data encryption is required.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D71008	BSQC Token not found	BSQC Token may expire

Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

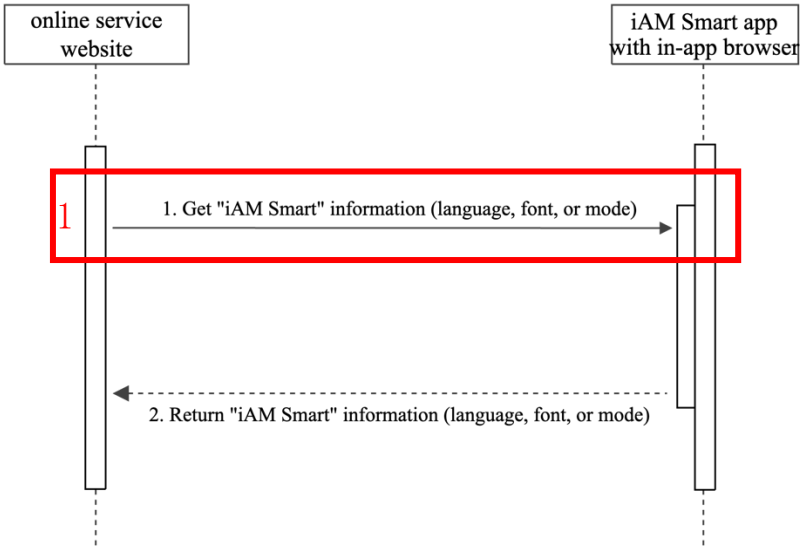
Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- Only the request of which status is “Pending for Signing” can be cancelled.

- The BSQC Token that’s used to enquire digital signing status or cancel signing request is valid for 1 day.

3.16 WORKFLOWS FOR IN-APP BROWSER JS API CALLS

3.16.1 Workflows for Requesting In-App Browser to Provide Information



Pre-conditions

- Online Service Website is loaded in In-app browser.
- the `eIDJSBridge` object exists.

Post-conditions

- The corresponding information is returned.

Example Calls and Request and Response Parameters

- `getAppLanguage()`

```
<script>
  function onClick() {
    var appLang = window.eIDJSBridge.getAppLanguage();
    /*
    * use the appLang value
    */
  }
</script>
```

The function does not need request parameter but returns "`tc`", "`en`" or "`sc`".

- `getAppFontSize()`

```
<script>
```

```
  function onClick() {
    var appFontSize = window.eIDJSBridge.getAppFontSize();
    /*
    * use the appFontSize value
    */
  }
</script>
```

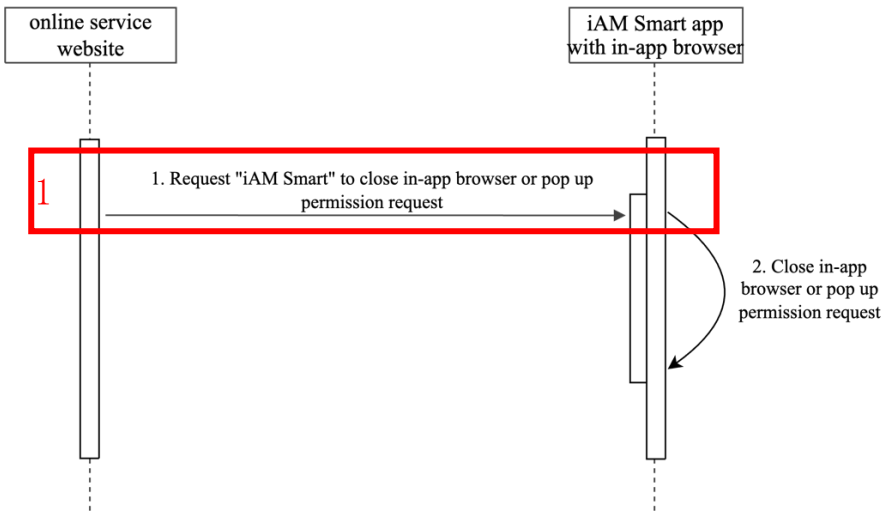
The function does not need request parameter but returns "`small`", "`medium`" or "`large`".

- `getAppMode()`

```
<script>
  function onClick() {
    var appMode = window.eIDJSBridge.getAppMode();
    /*
    * use the appMode value
    */
  }
</script>
```

The function does not need request parameter but returns "`normal`" or "`lite`"

3.16.2 Workflows for Requesting In-App Browser to Take Action



Pre-conditions

- Online Service Website is loaded in In-app browser.
- the `eIDJSBridge` object exists.

Post-conditions

- The corresponding action is taken.

Example Calls and Request and Response Parameters

- `closeInAppBrowser()`

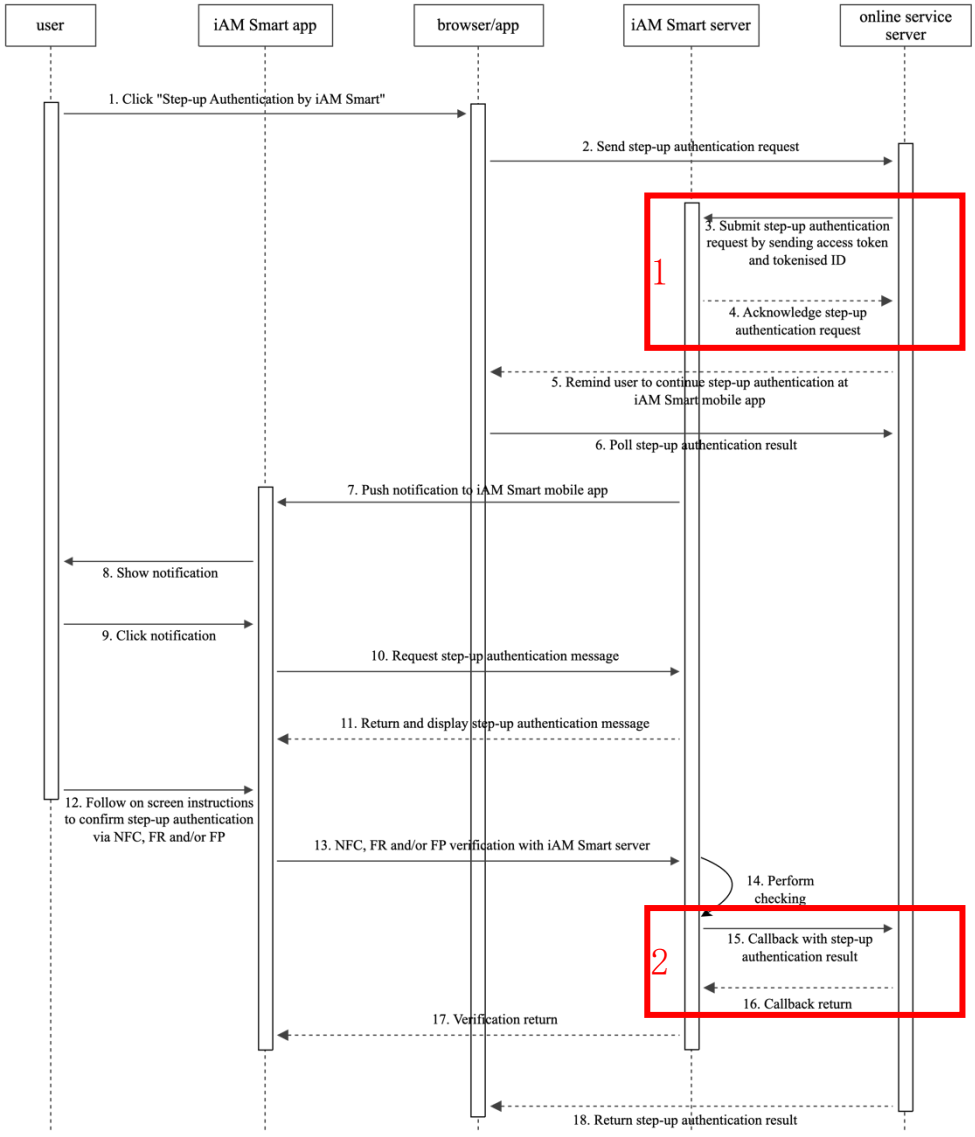
```
<script>
  function onClick() {
    window.eIDJSBridge.closeInAppBrowser();
  }
</script>
```

No request and response parameter.

3.17 WORKFLOWS FOR STEP-UP AUTHENTICATION WITH SERVICE LOGIN

3.17.1 Scenario 1: Step-up Authentication (Online Service Website/App in Different Device)

The sequence diagram below shows how online service performs "iAM Smart" Step-up Authentication when online service and the "iAM Smart" Mobile App are running in different devices.



Step 1. User clicks the "Authenticate with iAM Smart" button in Online Service Website/App;

Step 2-3. Online Service initiate Step-Up Authentication request to invoke the "iAM Smart" API with the "businessID" of this request,

accessToken, Tokenised ID, etc. The request parameter "source" will be the browser's user agent value (for Online Service Website) or "App_Scheme"/"App_Link" (for Online Service App) in this scenario. API encryption is required;

Step 4. "iAM Smart" System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the Step-Up Authentication request to Online Service server by returning POST response with parameter "authByQR" (set to "true") in this scenario;

Step 5-6. Online Service Website/App should keep synchronising with Online Service Server for the Step-up authentication processing result (e.g., polling);

Step 7. "iAM Smart" System pushes a notification message to the "iAM Smart" Mobile App based on the Tokenised ID;

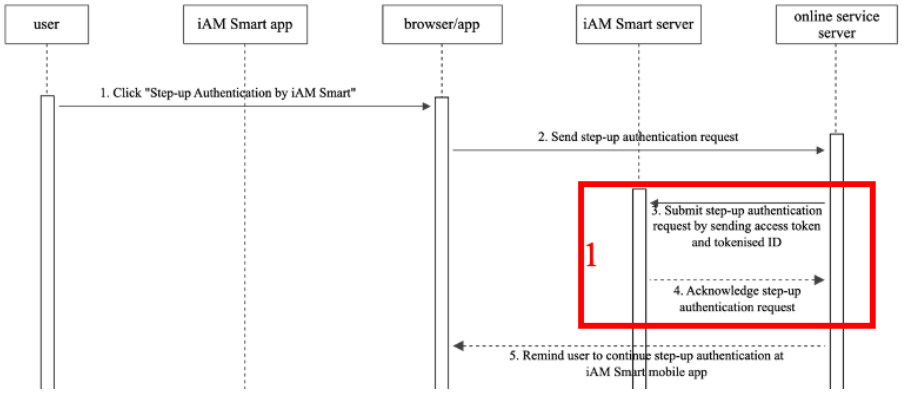
Step 8-11. "iAM Smart" user logs in "iAM Smart" Mobile App, reviews the Step-up authentication request (e.g., continue or reject the request);

Step 12-14. "iAM Smart" user follows the instructions in "iAM Smart" Mobile App to complete Step-up Authentication operation;

Step 15-17. "iAM Smart" System invokes Online Service callback API to return the result with "businessID" of the Step-up Authentication request. API data decryption is required;

Step 18. Online Service Server matches the result using the "businessID" and shows the Step-up Authentication result in corresponding Online Service Website/App.

3.17.1.1 Implementing (1) POST: Request Step-up Authentication



Pre-conditions

- Online Service must possess a valid accessToken of the “iAM Smart” user with authorisation scope “Step-up Authentication”.
- Online Service Website/App and “iAM Smart” Mobile App are in different devices in this scenario.
- Online Service generates a “businessID” for this Step-up Authentication request and uses this identifier to match the callback result returned from “iAM Smart” System.
- API data encryption is required.

Post-conditions

- Online Service server should check the response parameter “authByQR” and “ticketID” and determine the next action. For details, please refer to Section 3.2.1. “ticketID” will not be provided by “iAM Smart” System in this scenario.
- Online Service Website/App should keep synchronising with Online Service Server for the Step-up Authentication result returned from “iAM Smart” System.
- API data decryption is required.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code – D41006	user is still in waiting period	Inform user that “iAM Smart” step-up authentication is not allowed for users who create iAM Smart users within certain period
code – D41007	user is not allowed to step-up authentication	Inform user that “iAM Smart” step-up authentication is not allowed for those ineligible users. E.g. CCIC users

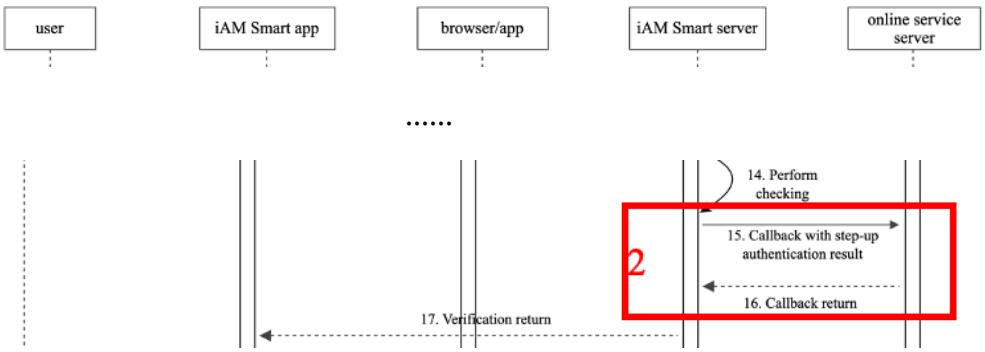
Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- Request parameter “source”: Value should be matched with Online Service client terminal (e.g., “App_Scheme”/“App_Link” or browser’s user agent value).
- Request parameter “openID”: It is the Tokenised ID of the “iAM Smart” user. Online Service should ensure the accessToken and Tokenised ID are in pair and belongs to the target “iAM Smart” user for the request.
- Request Parameter - “redirectURI”: Value should equal to URI of “Callback to Receive Step-up Authentication Result” Online Service callback API. It must be in the same value as provided during Online Service registration.
- Response parameter “authByQR”: The value returned by “iAM Smart” System will be “true” in this scenario.
- “iAM Smart” System will not return the response parameter “ticketID” in this scenario.

3.17.1.2 Implementing (2) POST: Callback to Receive Step-up Authentication Result



Pre-conditions

- “iAM Smart” user can accept and complete the step-up authentication request.
- “iAM Smart” user can also reject the step-up authentication request.

Post-conditions

- API data decryption is required.
- Online Service Server should match the callback result with corresponding Online Service Website/App using the “businessID”.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code – D41001	user rejected step-up authentication request	Inform user the “iAM Smart” step-up authentication request is rejected
code – D41500	Failed to step-up authenticate	Inform user the “iAM Smart” step-up authentication request is failed and retry later
code – D41503	device not supported for NFC	Inform user the “iAM Smart” step-up authentication request is not allowed for devices not support NFC
code – D41002	Step-up authentication request timeout	Inform user the “iAM Smart” step-up authentication request is timeout and provide way for user to retry

Callback Parameters

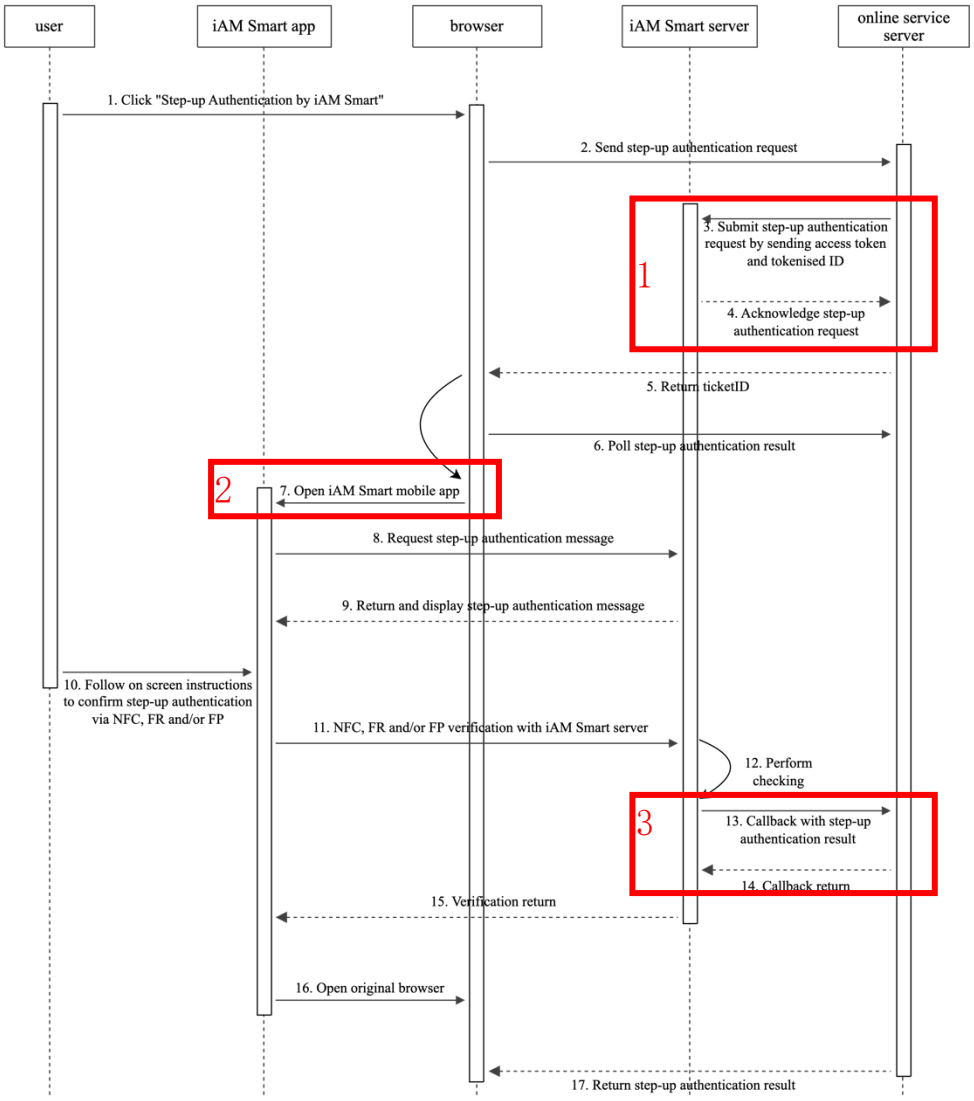
- Please refer to “iAM Smart” API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.

3.17.2 Scenario 2: Step-up Authentication (Online Service Website in Same Device)

The sequence diagram below shows how online service performs “iAM Smart” Step-up Authentication when the online service website and the “iAM Smart” Mobile App are running in the same device.



Step 1. User clicks the “Authenticate with iAM Smart” button in Online Service Website;

Step 2-3. Online Service Website initiates Step-Up Authentication request to Online Service server with browser's user agent name (for use as value for request parameter “source”). Online Service initiates Step-Up Authentication request to invoke “iAM Smart” API with the “businessID” of this request, accessToken, Tokenised ID, etc.

The request parameter “source” will be browser's user agent value in this scenario. API data encryption is required;

Step 4. “iAM Smart” System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the Step-Up Authentication request to Online Service server by returning POST response with parameters “authByQR” (set to “false”) and “ticketID”;

Step 5-6. Online Service Website should keep synchronising with Online Service Server for Step-Up Authentication processing result (e.g., polling);

Step 7. Online Service Website invokes “iAM Smart” Mobile App using URL Scheme with “ticketID”.

Step 8-9. “iAM Smart” user logs in “iAM Smart” Mobile App, reviews the Step-Up Authentication authorisation request (e.g., continue or reject the request);

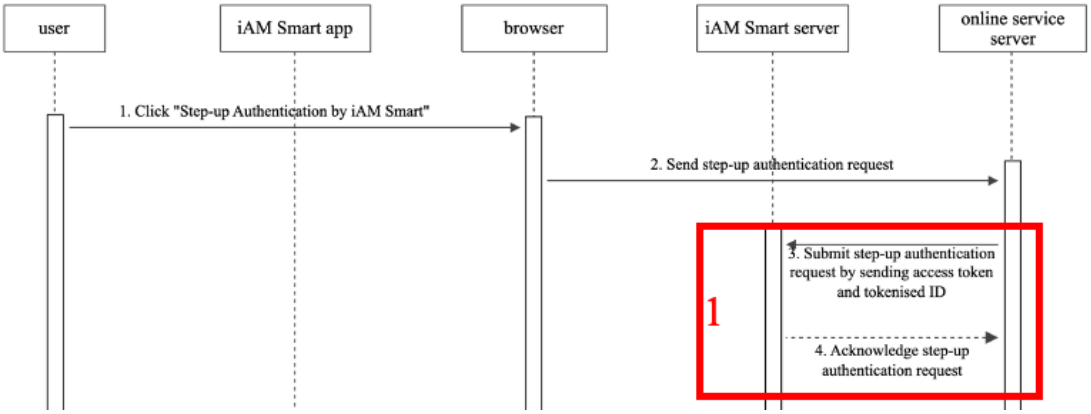
Step 10-12. “iAM Smart” user follows the instructions in “iAM Smart” Mobile App to complete Step-Up Authentication operation;

Step 13-14. “iAM Smart” System invokes Online Service callback API to return the result with “businessID” of the Step-Up Authentication request to Online Service server. API data decryption is required;

Step 15-16. “iAM Smart” System instructs “iAM Smart” Mobile App to invoke the original Online Service browser (i.e., using the browser's user agent name submitted in Step-Up Authentication request in Step 3);

Step 17. Online Service Server matches the result using the “businessID” and shows the Step-Up Authentication result in corresponding Online Service Website.

3.17.2.1 Implementing (1) POST: Request Step-Up Authentication



Pre-conditions

- Please refer to Section 3.17.1.1 except:
  - Online Service Website and “iAM Smart” Mobile App are in the same device in this scenario.

Post-conditions

- Please refer to Section 3.17.1.1 except:
  - Response “authByQR” is “false”, “ticketID” will be provided by “iAM Smart” System in this scenario.

Error conditions

- Please refer to Section 3.17.1.1.

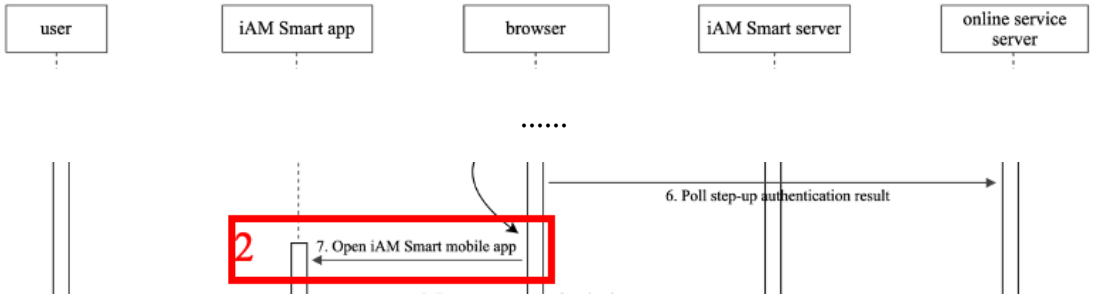
Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Please refer to Section 3.17.1.1, except for the following parameters:
  - Request parameter “source”: Value should be the browser's user agent value.
  - Response parameter “authByQR”: The value is “false” in this scenario.
  - Response parameter “ticketID”: It will be provided by “iAM Smart” System in this scenario. It is a 36-byte (or less) UUID number (ASCII character set).

3.17.2.2 Implementing (2) URL Scheme: Open “iAM Smart” Mobile App



Pre-conditions

- Online Service Website and “iAM Smart” Mobile App are in the same device.
- Online Service has the “ticketID” provided by “iAM Smart” System for the Step-up Authentication request;
- Other parameters of this API are not used in this scenario.

Post-conditions

- “iAM Smart” Mobile App will be launched.
- Online Service Website should keep synchronising with Online Service server for the callback response of the Step-up Authentication request from “iAM Smart” System.

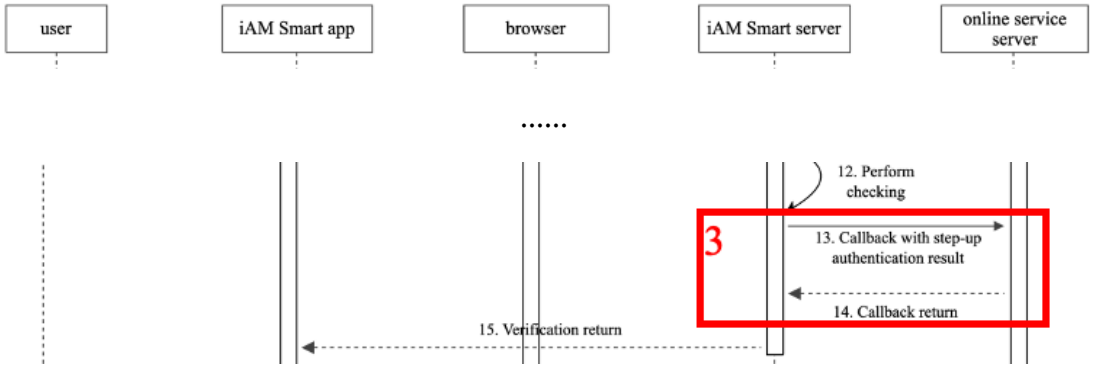
Error conditions

- Nil

Request Parameters

- Please refer to “iAM Smart” API Specification.

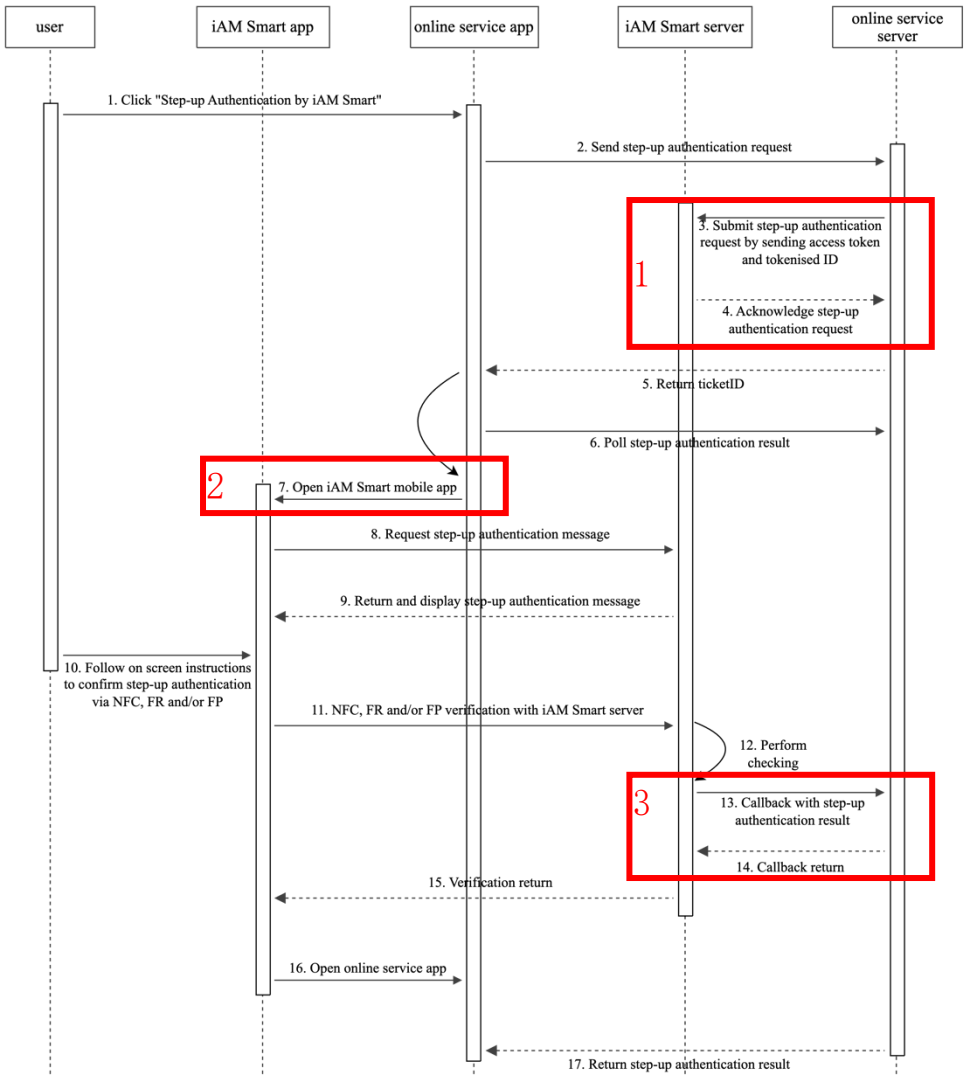
3.17.2.3 Implementing (3) POST: Callback to Receive Step-up Authentication Result



Please refer to Section 3.17.1.2.

3.17.3 Scenario 3: Step-up Authentication (Online Service App in Same Device)

The sequence diagram below shows how online service performs "iAM Smart" Step-up Authentication when the online service mobile application and the "iAM Smart" Mobile App are running in the same device.



Step 1. User clicks the "Authenticate with iAM Smart" button in Online Service App;

Step 2. Online Service App initiates step-up authentication request to Online Service server with "App_Scheme" or "App_Link" (use as the value of request parameter "source"). Online Service initiates Step-up Authentication request to invoke "iAM Smart" API with the "businessID" of this request, accessToken,

Tokenised ID, etc. The request parameter "source" will be "App_Scheme" or "App_Link" in this scenario. API data encryption is required;

Step 3-4. "iAM Smart" System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the step-up authentication request to Online Service server by returning POST response with parameters "authByQR" (set to "false") and "ticketID";

Step 5. Online Service prepares and sends necessary parameters such as "authByQR", "ticketID", etc. to Online Service App. Online Service App displays the instruction to inform "iAM Smart" user to process the Step-up Authentication authorisation request in "iAM Smart" Mobile App;

Step 6. Online Service App should keep synchronising with Online Service Server for Step-up Authentication processing result (e.g., polling);

Step 7. Online Service App invokes "iAM Smart" Mobile App using URL Scheme with "ticketID".

Step 8. "iAM Smart" user logs in "iAM Smart" Mobile App, reviews the Step-up Authentication authorisation request (e.g., continue or reject the request);

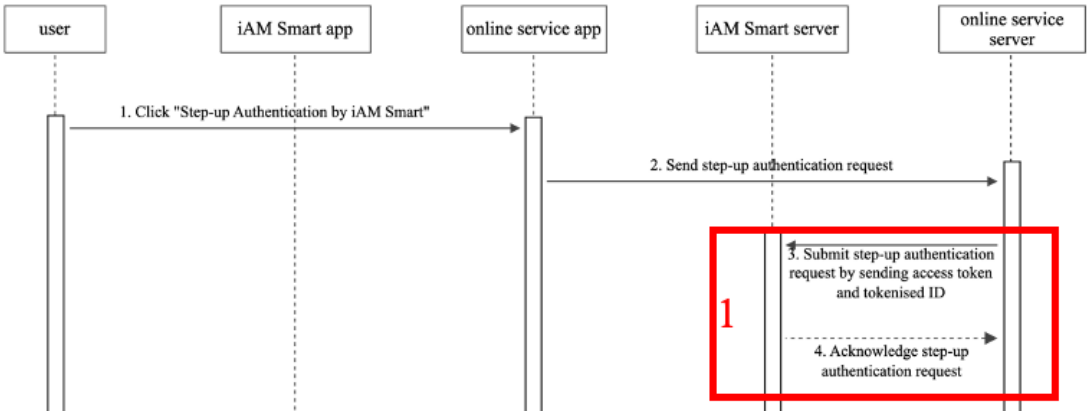
Step 10-12. "iAM Smart" user follows the instructions in "iAM Smart" Mobile App to complete Step-up Authentication operation;

Step 13-14. "iAM Smart" System invokes Online Service callback API to return the result with "businessID" of the Step-up Authentication request to Online Service Server. API data decryption is required;

Step 15-16. "iAM Smart" System instructs "iAM Smart" Mobile App to invoke the Online Service App using URL Scheme, or Universal Link/App Link ("iAM Smart" System queries the information registered at Online Service registration depending on the "source" submitted in Step 2);

Step 17. Online Service Server matches the result using the "businessID" and shows the Step-up Authentication result in corresponding Online Service App.

3.17.3.1 Implementing (1) POST: Request Step-up Authentication



Pre-conditions

- Please refer to Section 3.17.2.1 except:
  - Online Service App and "iAM Smart" Mobile App are in the same device in this scenario.

Post-conditions

- Please refer to Section 3.17.2.1.

Error conditions

- Please refer to Section 3.17.2.1.

Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

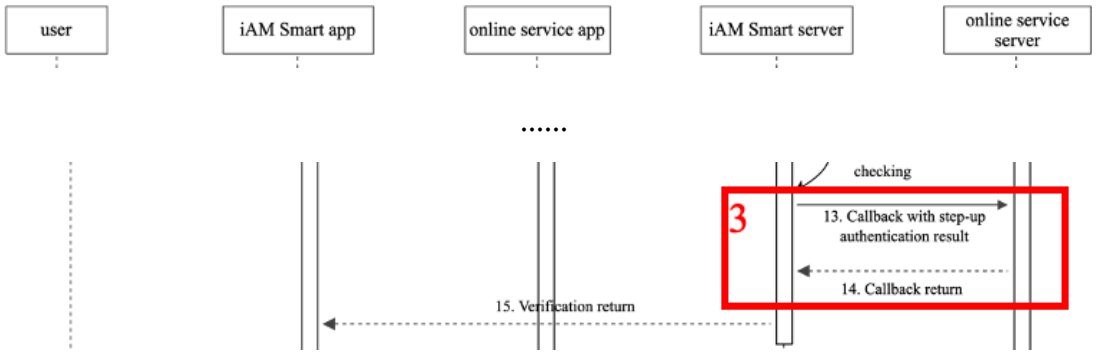
Notes

- Please refer to Section 3.17.2.1, except for the following parameter:
  - Request parameter "source": Value should be "App_Scheme" (for the Online Service App URL scheme), or "App_Link" (for the Universal Link/App Link).

3.17.3.2 Implementing (2) URL Scheme: Open “iAM Smart” Mobile App

Please refer to Section 3.17.2.2.

3.17.3.3 Implementing (3) POST: Callback to Receive Step-up Authentication Result

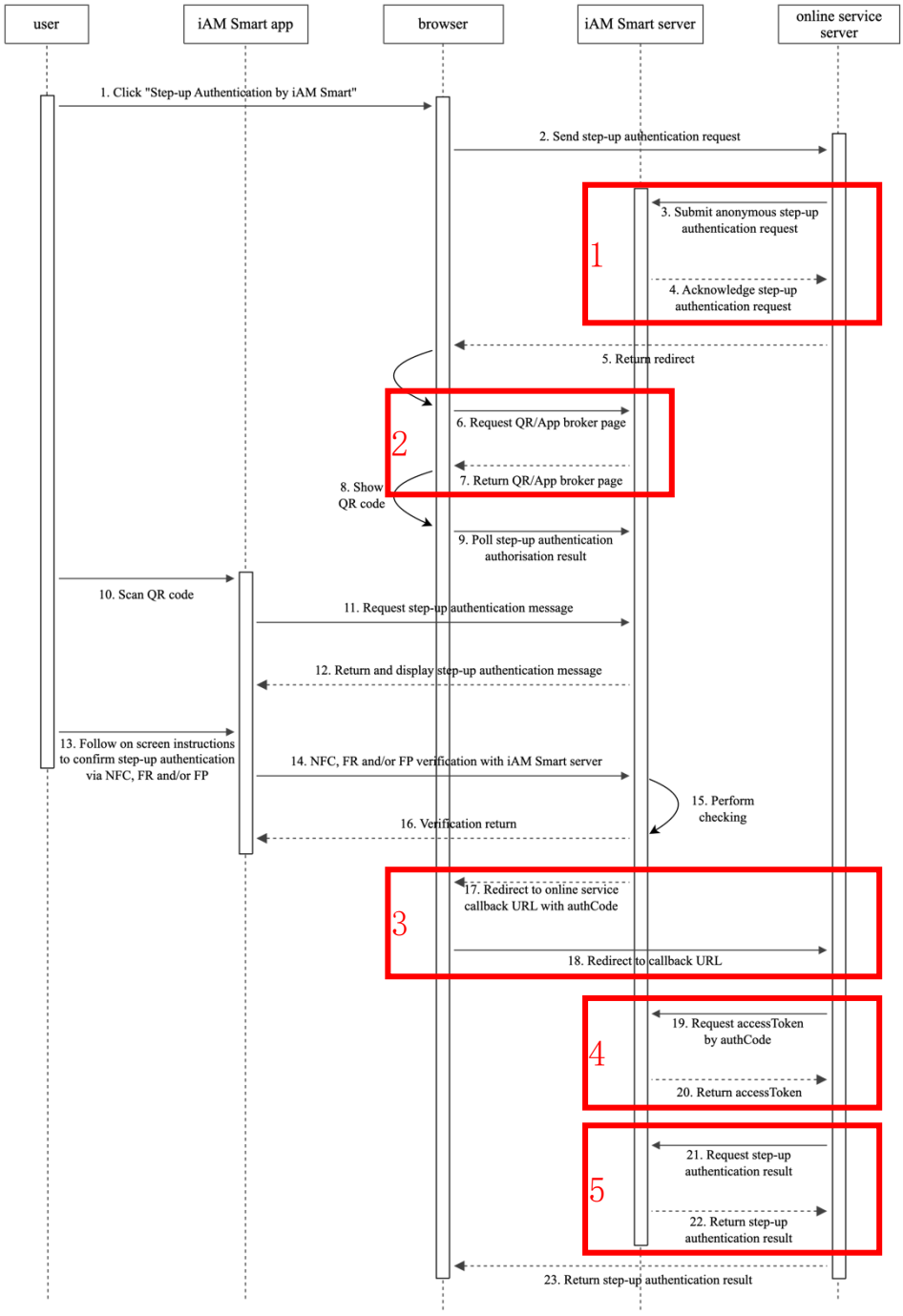


Please refer to Section 3.17.2.3.

3.18 WORKFLOWS FOR STEP-UP AUTHENTICATION WITHOUT SERVICE LOGIN (AKA ANONYMOUS STEP-UP AUTHENTICATION)

3.18.1 Scenario 1: Anonymous Step-up Authentication (Online Service Website in Different Device)

The sequence diagram below shows how an anonymous user performs step-up authentication when online service website and the "iAM Smart" Mobile App are running in different devices.



Step 1. After entering HKIC number, the user clicks the “Authenticate with iAM Smart” button in Online Service Website;

Step 2. Online Service Website initiates anonymous Step-up Authentication request to Online Service Server with browser's user agent name (use as value for request parameter “source”);

Step 3. Online Service Server initiates an anonymous Step-up Authentication request to invoke the “iAM Smart” API with the “businessID” of this request, HKICHash, etc. API encryption is required;

Step 4. “iAM Smart” System verifies and confirms receipt of the anonymous Step-up Authentication request to Online Service Server by returning POST response with parameter “ticketID”;

Step 5. Online Service Server instructs Online Service Website to display the instructions to inform “iAM Smart” user to process the Step-up Authentication authorisation request in “iAM Smart” Mobile App;

Online Service Server prepares the request parameters such as “clientID”, “redirectURI”, “source” (set as browser’s user agent value), “scope”, “ticketID”, etc. and constructs the GET request to invoke the “iAM Smart” API using browser redirection;

Step 6-8. “iAM Smart” System returns the broker page (if Online Service has set “brokerPage” to “true”) and after the broker page fails to find the “iAM Smart” Mobile App, it will request QR page to be displayed in browser. If Online Service does not request for broker page (i.e., no broker page will be returned), the browser will directly display QR Code page;

Step 9. QR Code page of “iAM Smart” System polls “iAM Smart” System for the QR Code status;

Step 10. “iAM Smart” user logs in “iAM Smart” Mobile App to scan the QR Code;

Step 11-12. “iAM Smart” user reviews the Step-up Authentication authorisation request (e.g., continue or reject the request);

Step 13-14. “iAM Smart” user follows the instructions in “iAM Smart” Mobile App to complete Step-up Authentication operation;

Step 15-16. “iAM Smart” System verifies the validity of QR code and other necessary information, and return the authorisation result to “iAM Smart” Mobile App;

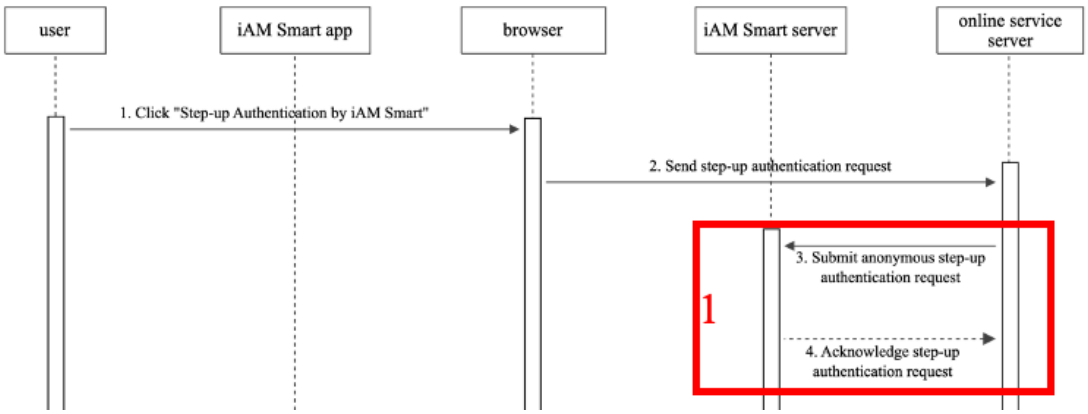
Step 17-18. QR Code page of “iAM Smart” System receives the polling result returned by “iAM Smart” System (i.e., response for the polling in Step 9) which includes authCode and businessID or any error code (e.g., “iAM Smart” user rejects the request), assembles and invokes the Online Service callback API given in “redirectURI” using browser redirection;

Step 19-20. When Online Service Server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invoke the “iAM Smart” API to obtain the accessToken and the Tokenised ID (i.e., openID) of the “iAM Smart” user. API data encryption is required;

Step 21-22. Online Service Server invokes the “iAM Smart” API with the accessToken and openID to obtain the result of anonymous Step-up Authentication. API data encryption is required;

Step 23. Online Service Server matches the result using the “businessID” and shows the Step-up Authentication result in corresponding Online Service Website.

3.18.1.1 Implementing (1) POST: Request Anonymous Step-up Authentication



Pre-conditions

- Please refer to Section 3.17.1.1 except:
  - No request parameter “accessToken” exists.
  - Online Service generates a “businessID” for this request and uses this identifier to match the authCode returned from “iAM Smart” System.
  - Online Service should convert the HKIC number (no check digit is needed) into hash value using SHA256.

Post-conditions

- Please refer to Section 3.17.1.1 except:
  - No response parameter “authByQR” exists.

Error conditions

For common errors, please refer to Section 3.2.3.

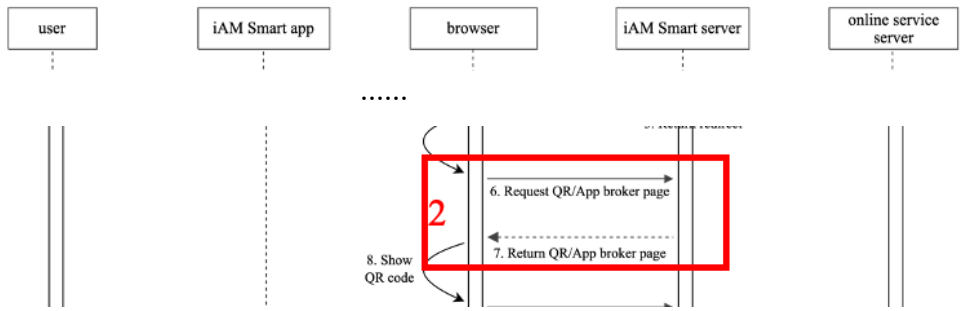
Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Please refer to Section 3.17.1.1, except for the following parameters:
  - No request parameter “accessToken”, “openID”, “source”, “redirectURI” and “state”.
  - No response parameter “authByQR”.
  - Response parameter “ticketID”: It is returned from “iAM Smart” System for “iAM Smart” APIs for anonymous request. It will be used for invoking subsequent “iAM Smart” APIs “Request QR Page” and “Open “iAM Smart” Mobile App for Getting Context” depending on the scenario. It is a 36-byte (or less) UUID number (ASCII character set).

3.18.1.2 Implementing (2) GET: Request QR Page



Pre-conditions

- Please refer to Section 3.4.1.1 except:

- Online Service has the “ticketID” provided by “iAM Smart” System for the anonymous request in step 4.

Post-conditions

- Please refer to Section 3.4.1.1.

Error conditions

- Please refer to Section 3.4.1.1.

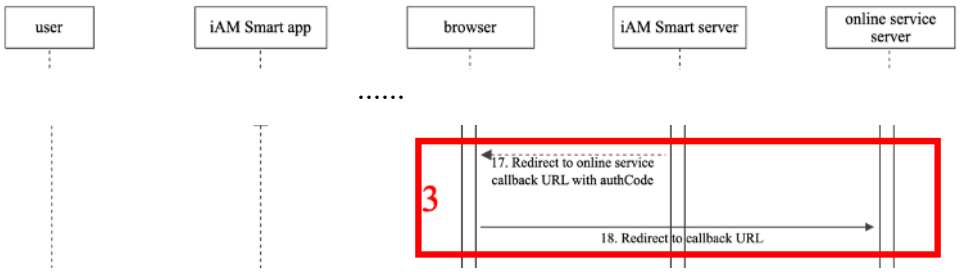
Request Parameters

- Please refer to Section 3.4.1.1.

Notes

- Please refer to Section 3.4.1.1 except the following parameter:
  - Request parameter “ticketID”: Value provided by “iAM Smart” System for the anonymous request.

3.18.1.3 Implementing (3) GET: Callback with authCode to Online Service Server



Pre-conditions

- Please refer to Section 3.4.1.2.

Post-conditions

- Please refer to Section 3.4.1.2 except:
  - Online Service Server should match the callback result with corresponding Online Service client terminal using the “businessID”.

Error conditions

- This Online Service callback API is a HTTP GET request. If parameter “error_code” is returned, it means the request is failed.

Error Code	Error Description	Suggested Action
code – D41001	user rejected step-up authentication request	Inform user the "iAM Smart" step-up authentication request is rejected
code – D41003	Inconsistent HKIC number	Hash of HKIC number provided is not matched with "iAM Smart" user. Check hash of HKIC number or inform user to authorise the step-up authentication using the correct "iAM Smart" account
code – D41006	user is still in waiting period	Inform user that "iAM Smart" step-up authentication is not allowed for users who create account within certain period
code – D41007	user is not allowed to step-up authentication	Inform user that "iAM Smart" step-up authentication is not allowed for those ineligible users. E.g. CCIC users
code – D41500	Failed to step-up authenticate	Inform user the "iAM Smart" step-up authentication request is failed and retry later
code – D41503	device not supported for NFC	Inform user the "iAM Smart" step-up authentication request is not allowed for devices not support NFC

The error_code D41002 (step-up authentication request timeout) does not appear in this scenario. For the QR code timeout or request confirmation timeout in the "iAM Smart" Mobile App, message will be prompted in the corresponding user interface.

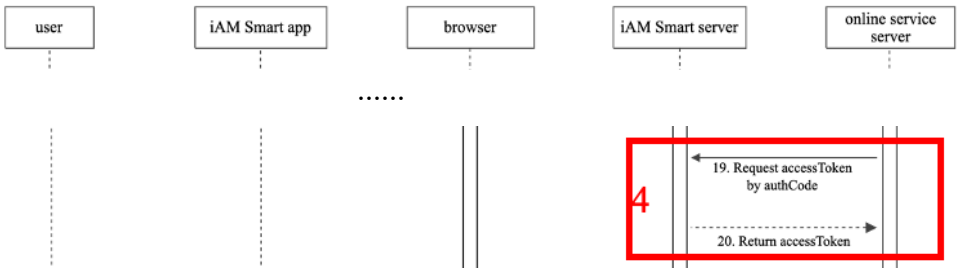
Callback Parameters

- Please refer to Section 3.4.1.2.

Notes

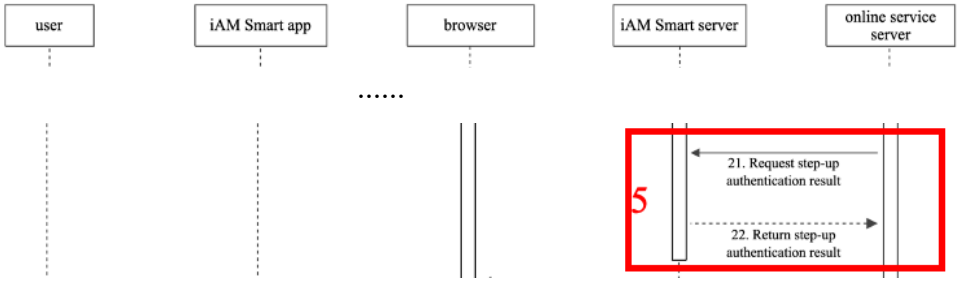
- Please refer to Section 3.4.1.2 except the following parameter:
  - Callback parameter "businessID": Value returned by "iAM Smart" System should be the same one as Online Service submitted in the anonymous request.

3.18.1.4 Implementing (4) POST: Request accessToken & Tokenised ID



Please refer to Section 3.4.1.3

3.18.1.5 Implementing (5) POST: Obtain Anonymous Step-up Authentication Result



Pre-conditions

- Online Service must possess a valid accessToken of the "iAM Smart" user with authorisation scope "Anonymous Step-up Authentication".
- API data encryption is required.

Post-conditions

- API data decryption is required.
- Online Service should discard the accessToken as it can only be used once.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code – D41001	user rejected step-up authentication request	Inform user the "iAM Smart" step-up authentication request is rejected

code – D41500	Failed to step-up authenticate	Inform user the “iAM Smart” step-up authentication request is failed and retry later
code – D41002	Step-up authentication request timeout	Inform user the “iAM Smart” step-up authentication request is timeout and provide way for user to retry

.Request and Response Parameters

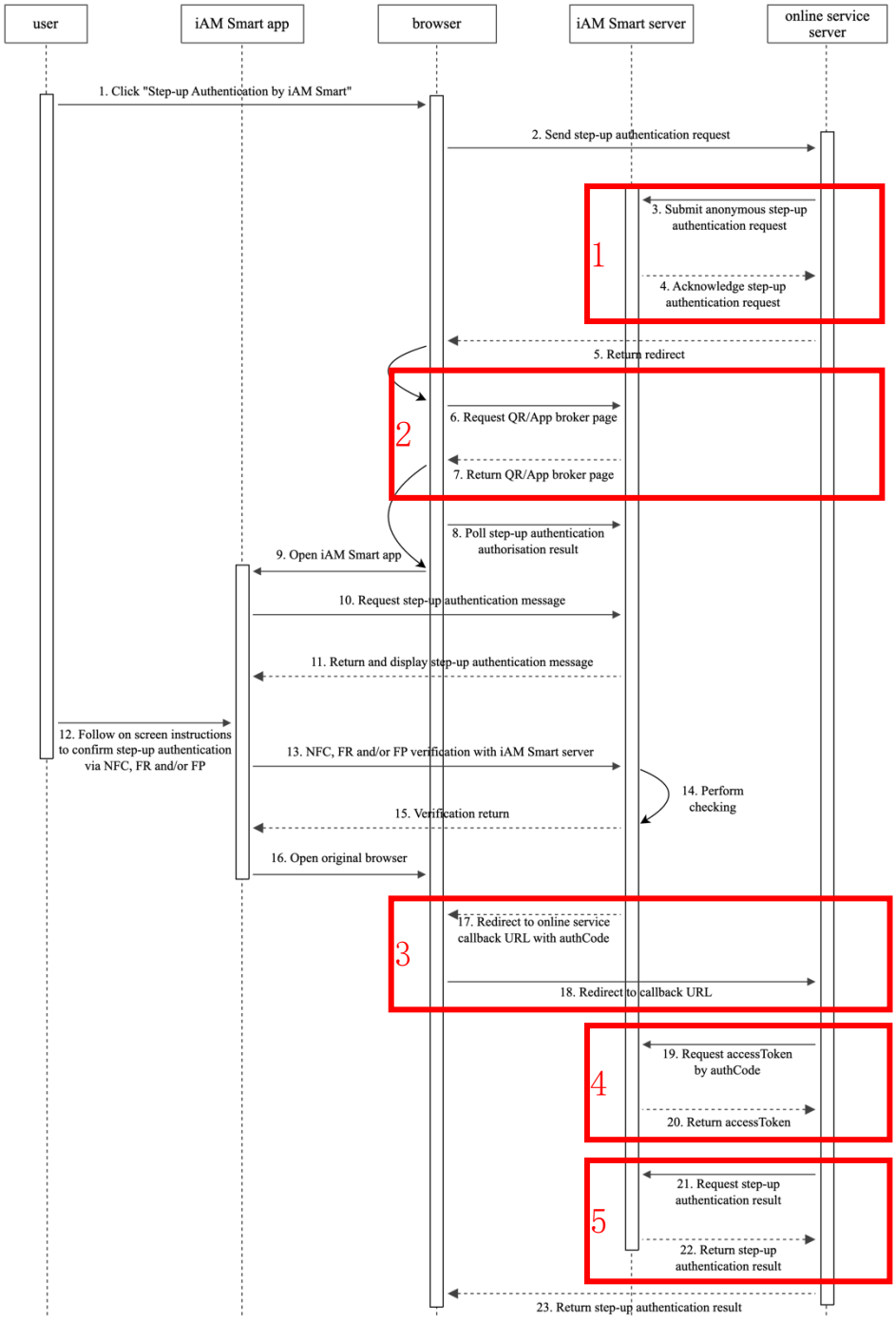
- Please refer to “iAM Smart” API Specification.

Notes

- Please refer to Section 3.17.1.2 except:
  - Response parameters “businessID” and “state” do not exist.

3.18.2 Scenario 2: Anonymous Step-up Authentication (Online Service Website in Same Device)

The sequence diagram below shows how an anonymous user performs step-up authentication when online service website and the "iAM Smart" Mobile App are running in the same device.



Step 1. After entering HKIC number, the user clicks the “Authenticate with iAM Smart” button in Online Service Website;

Step 2. Online Service Website initiates anonymous Step-up Authentication request to Online Service Server with browser's user agent name (use as value for request parameter “source”);

Step 3. Online Service Server initiates an anonymous Step-up Authentication request to invoke the “iAM Smart” API with the “businessID” of this request, HKICHash, etc. API data encryption is required;

Step 4. “iAM Smart” System verifies and confirms receipt of the anonymous Step-up Authentication request to Online Service Server by returning POST response with parameter “ticketID”;

Step 5. Online Service Server instructs Online Service Website to display the instructions to inform “iAM Smart” user to process the Step-up Authentication authorisation request in “iAM Smart” Mobile App;

Online Service Server prepares the request parameters such as “clientID”, “redirectURI”, “source” (set as browser’s user agent value), “scope”, “brokerPage” (set as “true”), “ticketID”, etc. and constructs the GET request to invoke the “iAM Smart” API using browser redirection;

Step 6-7. “iAM Smart” System returns the broker page to the Online Service Website;

Step 8. Broker page of “iAM Smart” System polls “iAM Smart” System for further action;

Step 9. Broker page invokes the “iAM Smart” Mobile App in the same device automatically and sends it the relevant parameters.

Step 10-11. “iAM Smart” user logs in “iAM Smart” Mobile App. “iAM Smart” user reviews the Step-up Authentication authorisation request (e.g., continue or reject the request);

Step 12-13. “iAM Smart” user follows the instructions in “iAM Smart” Mobile App to complete Step-up Authentication operation;

Step 14-16. “iAM Smart” System verifies the validity of necessary information, and return the authorisation result to “iAM Smart” Mobile App.

"iAM Smart" Mobile App invokes the original Online Service browser (i.e., using the browser's user agent value submitted);

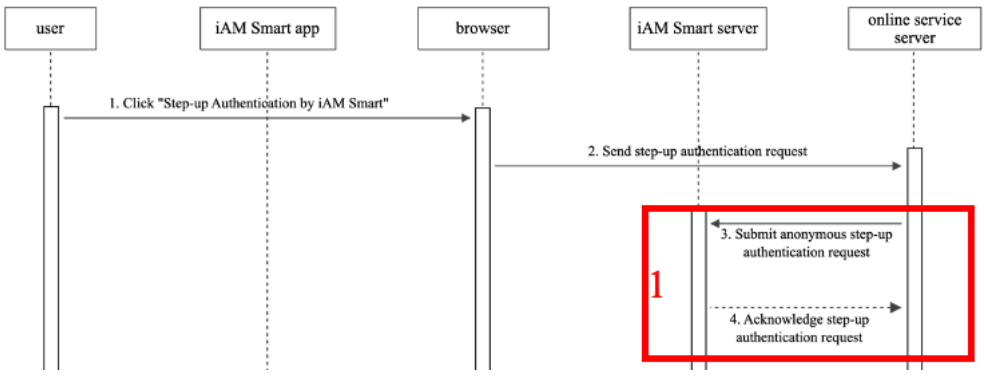
Step 17-18. Broker page of "iAM Smart" System receives the polling result returned by "iAM Smart" System (i.e., response for the polling in Step 9) which includes authCode and businessID or any error code (e.g., "iAM Smart" user rejects the request), assembles and invokes the Online Service callback API given in "redirectURI" using browser redirection;

Step 19-20. When Online Service Server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e., openID) of the "iAM Smart" user. API data encryption is required;

Step 21-22. Online Service Server invokes the "iAM Smart" API with the accessToken and openID to obtain the result of anonymous Step-up Authentication. API data encryption is required;

Step 23. Online Service Server matches the result using the "businessID" and shows the Step-up Authentication result in corresponding Online Service Website.

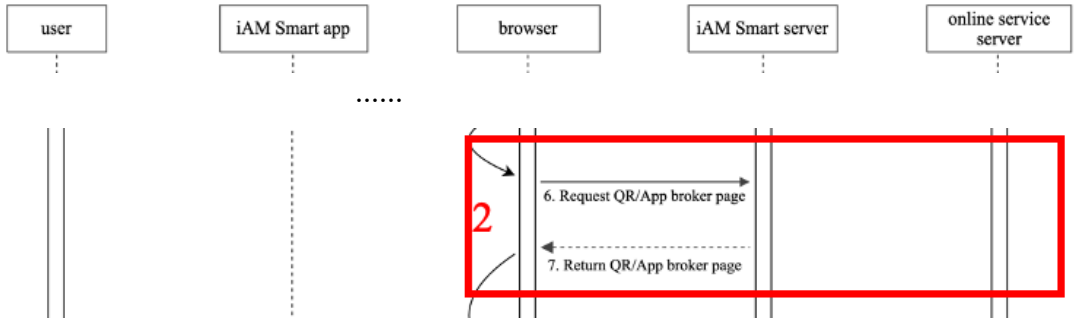
3.18.2.1 Implementing (1) POST: Request Anonymous Step-up Authentication



Please refer to Section 3.18.1.1 except:

- Online Service Website and "iAM Smart" Mobile App are in the same device in this scenario.

3.18.2.2 Implementing (2) GET: Request QR Page



Pre-conditions

- Please refer to Section 3.4.2.1 except:
  - Online Service has the "ticketID" provided by "iAM Smart" System for the anonymous request in step 4.

Post-conditions

- Please refer to Section 3.4.2.1.

Error conditions

- Please refer to Section 3.4.2.1.

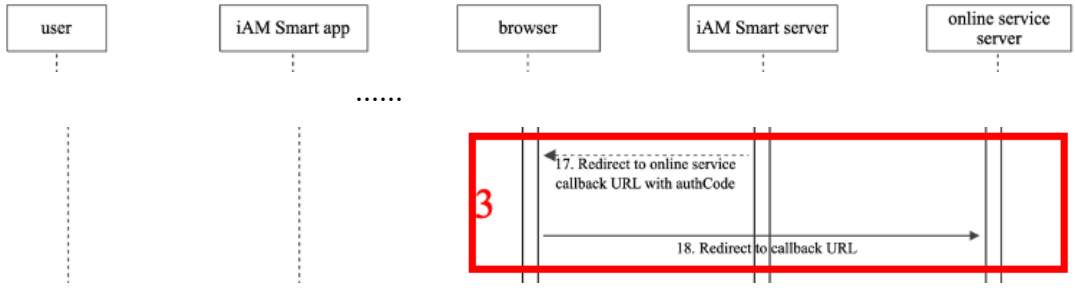
Request Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- Please refer to Section 3.4.2.1 except the following parameter:
  - Request parameter "ticketID": Value provided by "iAM Smart" System for the anonymous request.

3.18.2.3 Implementing (3) GET: Callback with authCode to Online Service Server



Pre-conditions

- Please refer to Section 3.6.2.3.

Post-conditions

- Please refer to Section 3.6.2.3.

Error conditions

- Please refer to Section 3.18.1.3.

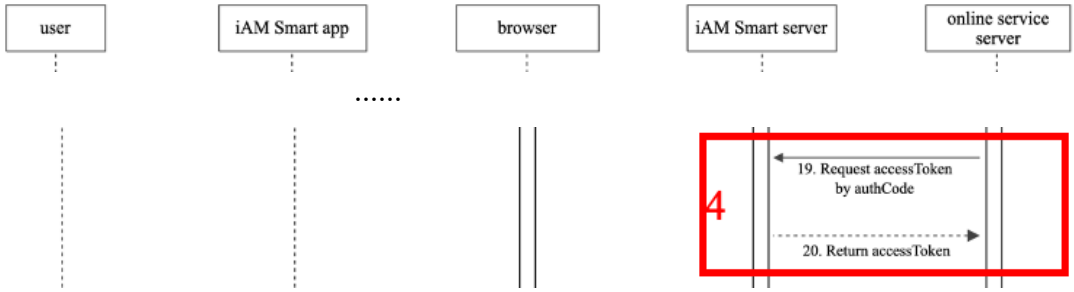
Callback Parameters

- Please refer to Section 3.6.2.3.

Notes

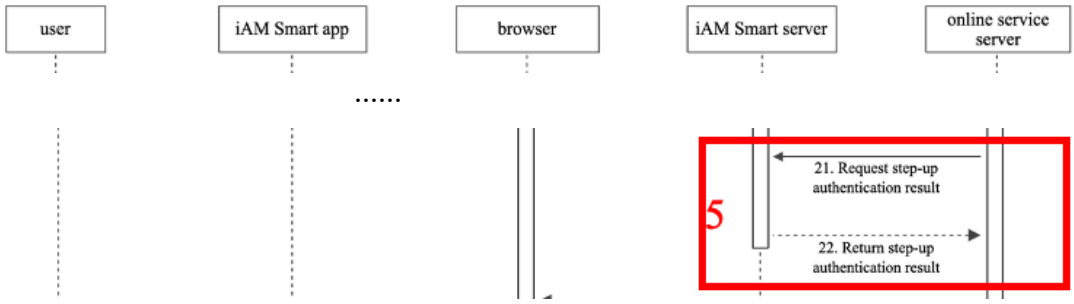
- Please refer to Section 3.6.2.3.

3.18.2.4 Implementing (4) POST: Request accessToken & Tokenised ID



Please refer to Section 3.6.1.4.

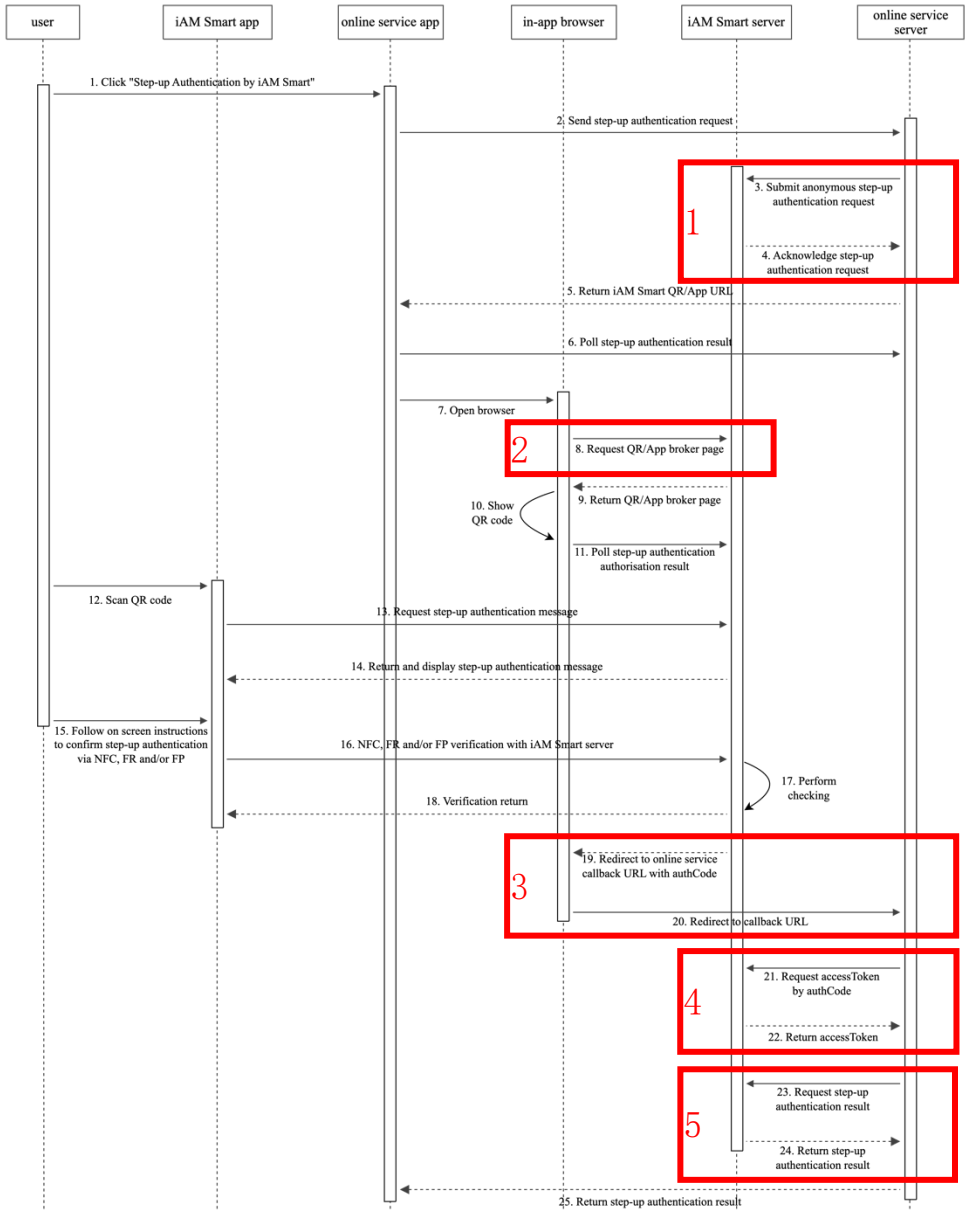
3.18.2.5 Implementing (5) POST: Obtain Anonymous Step-up Authentication Result



Please refer to Section 3.18.1.5.

3.18.3 Scenario 3: Anonymous Step-up Authentication (Online Service App in Different Device)

The sequence diagram below shows how an anonymous user performs step-up authentication when online service App and the "iAM Smart" Mobile App are running in different devices.



Step 1. After entering HKIC number, the user clicks the "Authenticate with iAM Smart" button in Online Service App;

Step 2. Online Service App determines there is no "iAM Smart" Mobile App in the device using program code, initiates anonymous Step-up Authentication request to Online Service Server with the in-app

browser's user agent value (use as value for request parameter "source");

Step 3. Online Service Server initiates an anonymous Step-up Authentication request to invoke the "iAM Smart" API with the "businessID" of this request, HKICHash, etc. API data encryption is required;

Step 4. "iAM Smart" System verifies and confirms receipt of the anonymous Step-up Authentication request to Online Service Server by returning POST response with parameter "ticketID";

Step 5. Online Service Server instructs Online Service App to display the instructions to inform "iAM Smart" user to process the Step-up Authentication authorisation request in "iAM Smart" Mobile App;

Online Service Server prepares necessary parameters such as "clientID", "redirectURI", "scope", "source" (set as in-app browser's user agent value), "brokerPage" (set as "false"), "ticketID", etc. and constructs the GET request to invoke the "iAM Smart" API by Online Service App;

Step 6. Online Service App polls Online Service Server for the Step-up Authentication result from "iAM Smart" System;

Step 7. Online Service App invokes in-app browser (Safari for iOS, Chrome for Android) to submit the GET request;

Step 8. Browser opens the GET request to invoke the "iAM Smart" API and QR Code page will be shown;

Step 9-11. QR Code page of "iAM Smart" System polls "iAM Smart" System for the QR Code status;

Step 12. "iAM Smart" user logs in "iAM Smart" Mobile App to scan the QR Code;

Step 13-15. "iAM Smart" user reviews the Step-up Authentication authorisation request (e.g., continue or reject the request);

Step 16. "iAM Smart" user follows the instructions in "iAM Smart" Mobile App to complete Step-up Authentication operation;

Step 17-18. "iAM Smart" System verifies the validity of QR code and other necessary information, and return the authorisation result to "iAM Smart" Mobile App;

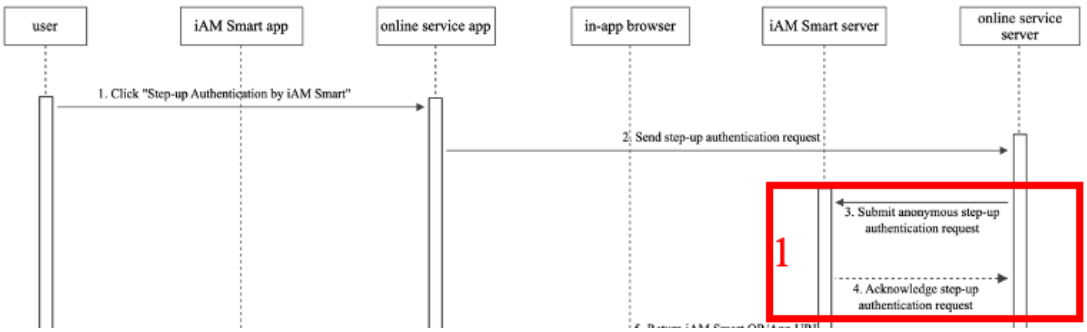
Step 19-20. QR Code page of "iAM Smart" System receives the polling result returned by "iAM Smart" System (i.e., response for the polling in Step 11) which includes authCode and businessID or any error code (e.g., "iAM Smart" user rejects the request), assembles and invokes the Online Service callback API given in "redirectURI" using browser redirection;

Step 21-22. When Online Service Server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e., openID) of the "iAM Smart" user. API data encryption is required;

Step 23-24. Online Service Server invokes the "iAM Smart" API with the accessToken and openID to obtain the result of anonymous Step-up Authentication. API data encryption is required;

Step 25. Online Service Server matches the result using the "businessID" and shows the step-up authentication result in corresponding Online Service App (i.e., response for the polling in Step 6).

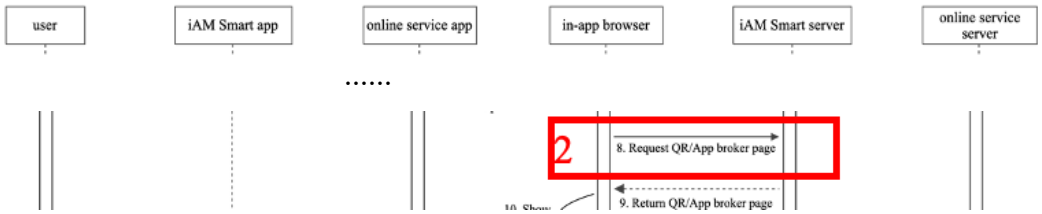
3.18.3.1 Implementing (1) POST: Request Anonymous Step-up Authentication



Please refer to Section 3.18.1.1 except:

- Online Service should determine the "iAM Smart" Mobile App is not in the same device of Online Service App using program code.

3.18.3.2 Implementing (2) GET: Request QR Page



Pre-conditions

- Please refer to Section 3.4.3.1 except:
  - Online Service has the "ticketID" provided by "iAM Smart" System for the anonymous request in step 4.

Post-conditions

- Please refer to Section 3.4.3.1.

Error conditions

- Please refer to Section 3.4.3.1.

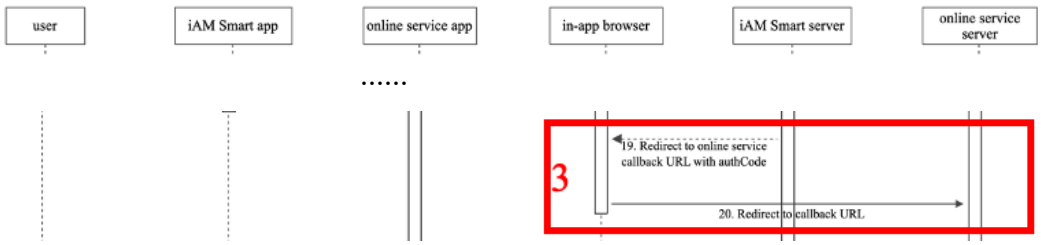
Request Parameters

- Please refer to "iAM Smart" API Specification.

Notes

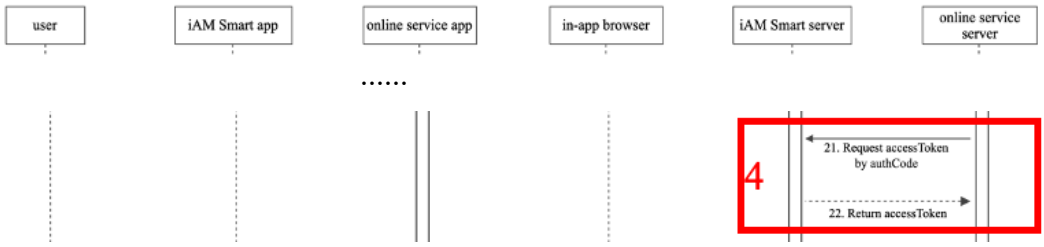
- Please refer to Section 3.4.3.1 except the following parameter:
  - Request parameter "ticketID": Value provided by "iAM Smart" System for the anonymous request.

3.18.3.3 Implementing (3) GET: Callback with authCode to Online Service Server



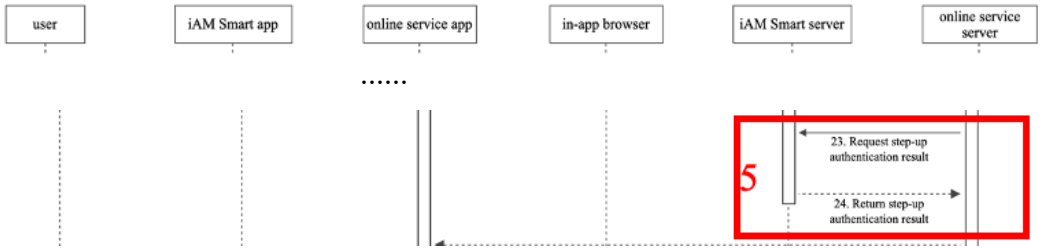
Please refer to Section 3.18.1.3.

3.18.3.4 Implementing (4) POST: Request accessToken & Tokenised ID



Please refer to Section 3.6.1.4.

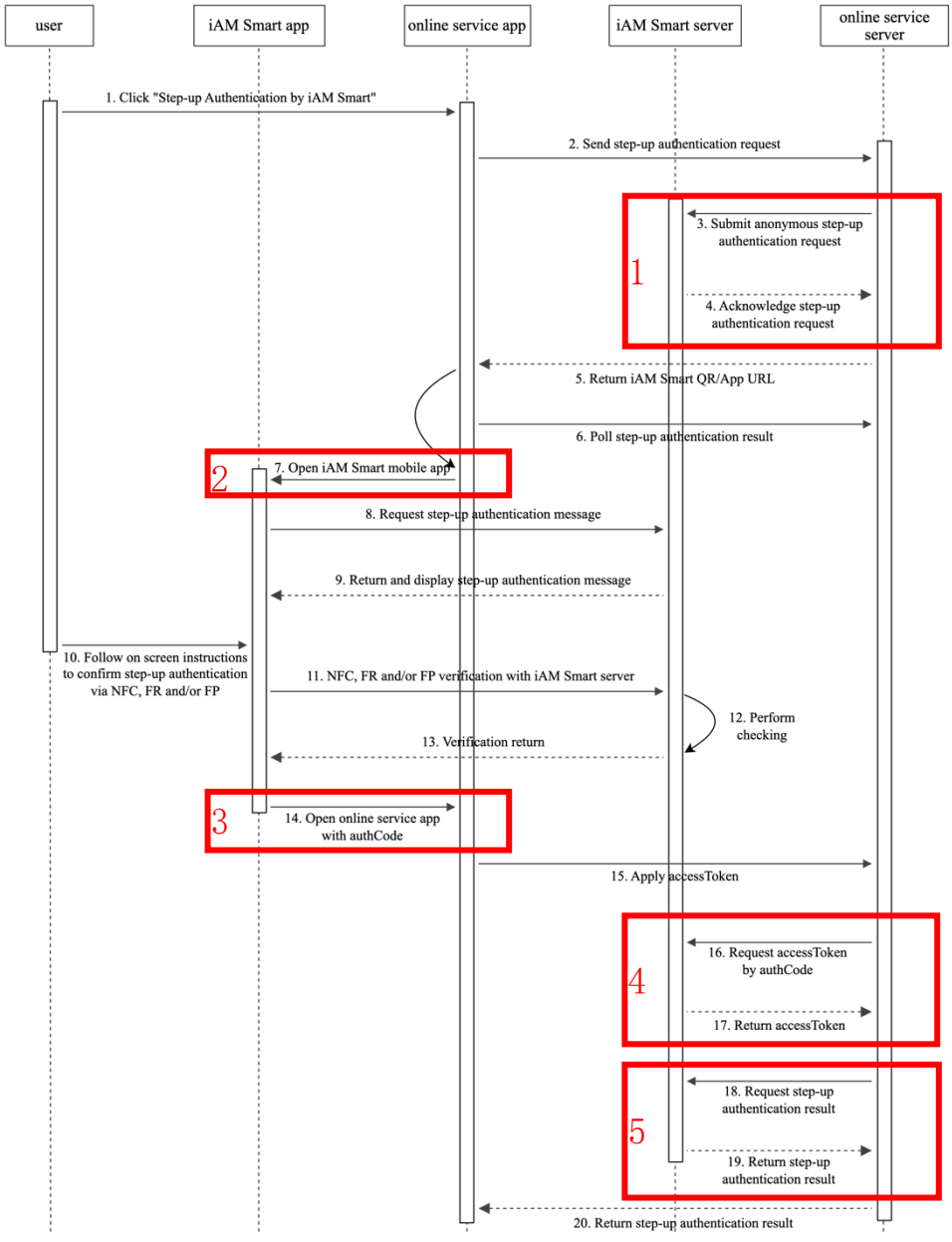
3.18.3.5 Implementing (5) POST: Obtain Anonymous Step-up Authentication Result



Please refer to Section 3.18.1.5.

3.18.4 Scenario 4: Anonymous Step-up Authentication (Online Service App in Same Device)

The sequence diagram below shows how an anonymous user performs step-up authentication when online service App and the "iAM Smart" Mobile App are running in the same device.



Step 1. After entering HKIC number, the user clicks the "Authenticate with iAM Smart" button in Online Service App;

Step 2. Online Service App determines "iAM Smart" Mobile App is installed in the device using program code, initiates anonymous

Step-up Authentication request to Online Service Server with "App_Scheme" or "App_Link" (use as value for request parameter "source");

Step 3. Online Service Server initiates an anonymous Step-up Authentication request to invoke the "iAM Smart" API with the "businessID" of this request, HKICHash, etc. API data encryption is required;

Step 4. "iAM Smart" System verifies and confirms receipt of the anonymous Step-up Authentication request to Online Service Server by returning POST response with parameter "ticketID";

Step 5. Online Service Server instructs Online Service App to display the instructions to inform "iAM Smart" user to process the Step-up Authentication authorisation request in "iAM Smart" Mobile App;

Online Service Server prepares necessary parameters such as "clientID", "redirectURI" (set as Online Service App URL Scheme or Universal Link/App Link depending on "source" parameter), "scope", "source" (set as "App_Scheme" or "App_Link"), "ticketID", etc. and constructs the URL Scheme to invoke "iAM Smart" Mobile App by Online Service App;

Step 6. Online Service App polls Online Service Server for the Step-up Authentication result from "iAM Smart" System;

Step 7. Online Service App invokes "iAM Smart" Mobile App using URL Scheme;

Step 8-9. "iAM Smart" user logs in "iAM Smart" Mobile App. "iAM Smart" user reviews the Step-up Authentication authorisation request (e.g., continue or reject the request);

Step 10-11. "iAM Smart" user follows the instructions in "iAM Smart" Mobile App to complete Step-up Authentication operation;

Step 12-13. "iAM Smart" System verifies the validity of necessary information, and return the authorisation result to "iAM Smart" Mobile App.

Step 14. "iAM Smart" Mobile App invokes Online Service App using URL Scheme or Universal Link/App Link with required parameters such as "authCode", "businessID";

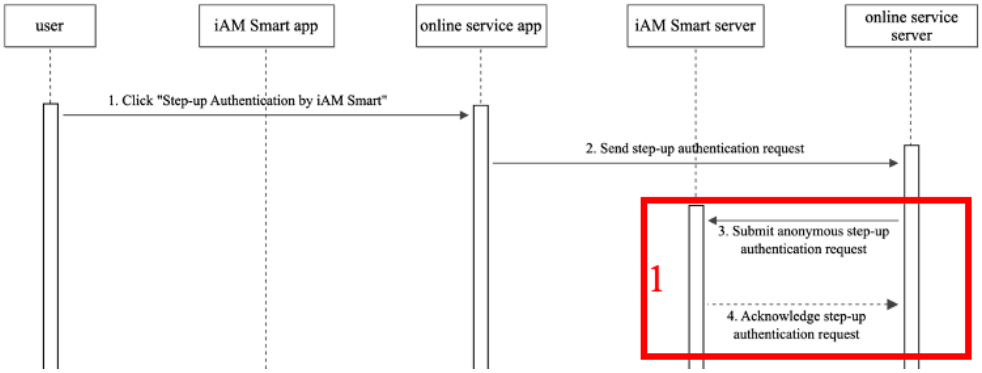
Step 15. Online Service App sends authCode, businessID, etc. to Online Service Server;

Step 16-17. When Online Service Server gets the authCode and businessID, it verifies businessID as generated in Step 3 and should then invoke the "iAM Smart" API to obtain the accessToken and the Tokenised ID (i.e., openID) of the "iAM Smart" user. API data encryption is required;

Step 18-19. Online Service Server invokes the "iAM Smart" API with the accessToken and openID to obtain the result of anonymous Step-up Authentication. API data encryption is required;

Step 20. Online Service Server matches the result using the "businessID" and shows the Step-up Authentication result in corresponding Online Service App (i.e., response for the polling in Step 6).

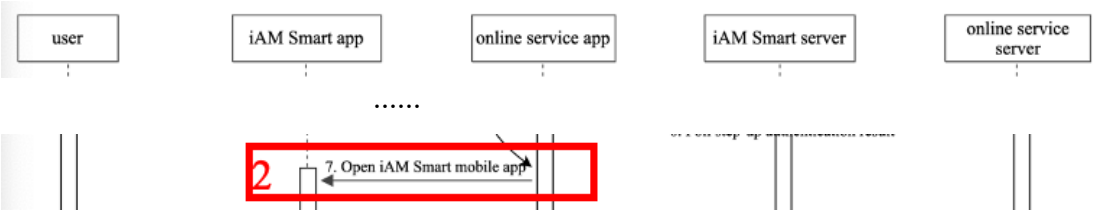
3.18.4.1 Implementing (1) POST: Request Anonymous Step-up Authentication



Please refer to Section 3.18.1.1 except:

- Online Service should determine the "iAM Smart" Mobile App is on the same device of Online Service App using program code.

3.18.4.2 Implementing (2) URL Scheme: Open "iAM Smart" Mobile App



Pre-conditions

- Online Service Website and “iAM Smart” Mobile App are in the same device.
- Online Service has the “ticketID” provided by “iAM Smart” System for the Step-up Authentication request in step 4.
- Online Service should determine if it would generate the optional “state” request parameter to prevent CSRF attack. The “state” will be returned to Online Service for checking through the Online Service callback API for receiving the authorisation code from “iAM Smart” System in Step 16.

Post-conditions

- “iAM Smart” Mobile App will be launched.
- Online Service Website should keep synchronising with Online Service server for the callback response of the Step-up Authentication request from “iAM Smart” System.

Error conditions

- Nil

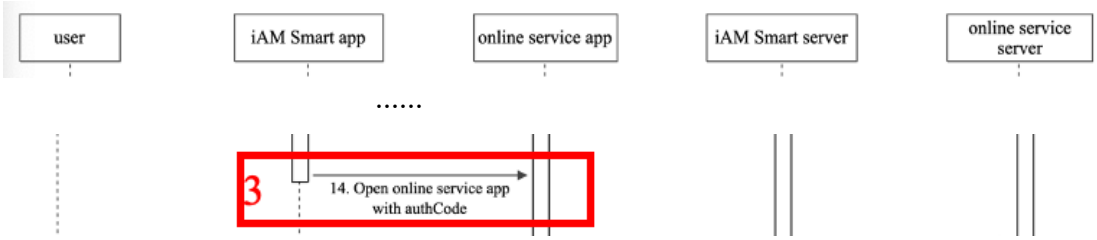
Request Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Request parameter “ticketID”: Value provided by “iAM Smart” System for the anonymous request.
- Request parameter “source”: Value should be “App_Scheme” (for Online Service App URL scheme), or “App_Link” (for the Universal Link/App Link).
- Request parameter “redirectURI”: This is Online Service App URL scheme or Universal Link/App Link depending on the “source” parameter, which is used for receiving the authorisation code. The value should be URL encoded. For details, please refer to Section 12.5.3 of “iAM Smart” API Specification. The value must be the same as provided during Online Service registration.
- Request parameter “state”: The value should be URL encoded

3.18.4.3 Implementing (3) Callback with authCode to Online Service App



Pre-conditions

- Please refer to Section 3.4.2.2.

Post-conditions

- Please refer to Section 3.4.2.2 except:
  - Online Service Server should match the callback result with corresponding Online Service App using the "businessID".

Error conditions

- Please refer to Section 3.18.1.3 except:
  - This Online Service callback API is a URL Scheme, or Universal Link/App Link.

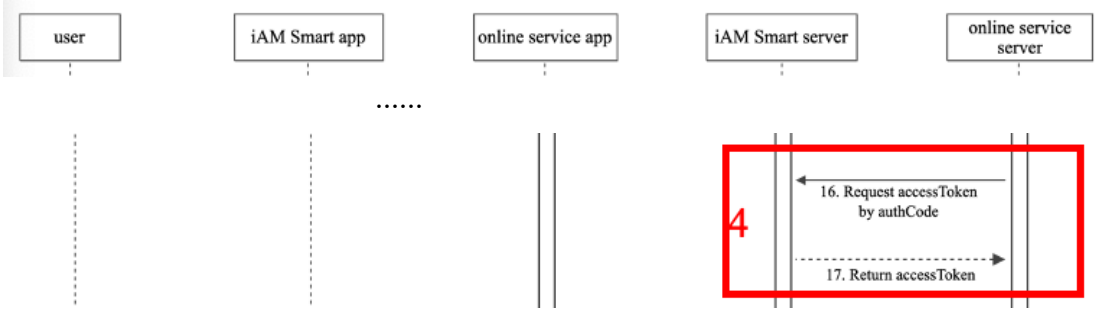
Callback Parameters

- Please refer to "iAM Smart" API Specification.

Notes

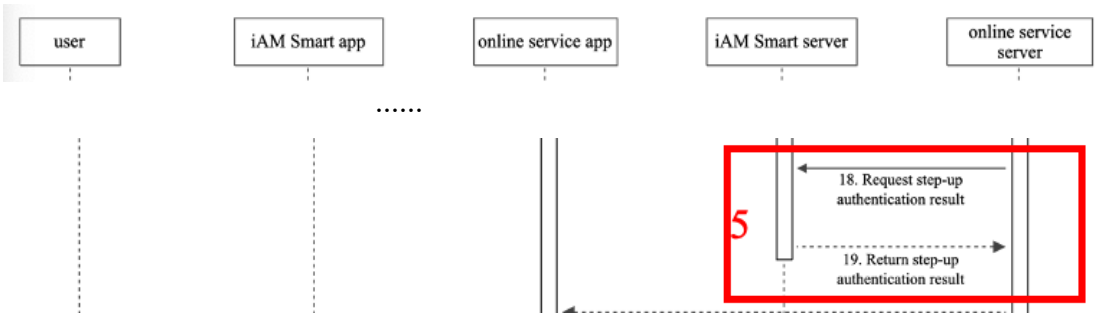
- Please refer to Section 3.4.2.2 except the following parameter:
  - Callback parameter "businessID": Value returned by "iAM Smart" System should be the same one as Online Service submitted in the anonymous request.

3.18.4.4 Implementing (4) POST: Request accessToken & Tokenised ID



Please refer to Section 3.6.1.4.

3.18.4.5 Implementing (5) POST: Obtain Anonymous Step-up Authentication Result



Please refer to Section 3.18.1.5.

APPENDICES

A. DEPRECATION

Online Service shall refer to the deprecation period of the respective APIs in "iAM Smart" API Specification. The Deprecation Date is the start date of the deprecation period, while Shutdown Date is the end date of the deprecation period. After the Shutdown Date, the feature will no longer be available.

A.1 Workflows for Getting Profile

A.1.1. Scenario 1: Get "iAM Smart" Profile (Online Service Website/App in Different Device)

The sequence diagram below shows how Online Service gets user additional profile information right after authentication when Online Service and the "iAM Smart" Mobile App are running different devices.

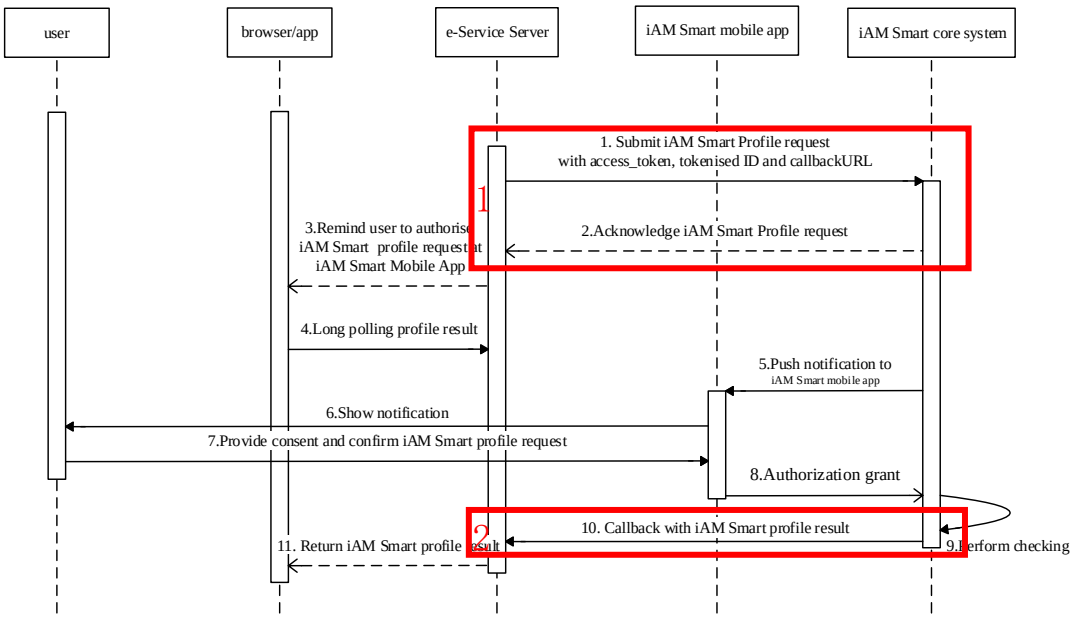


Figure-27 Get "iAM Smart" Profile (Online Service Website/App in Different Device)

Step 1. Online Service Server initiates an "iAM Smart" Profile request to invoke the "iAM Smart" API with the "businessID" of this request, accessToken, Tokenised ID and required "iAM Smart" Profile items. The request parameter "source" can be either the browser's user agent value (for Online Service Website) or "App_Scheme"/"App_Link" (for Online Service App) in this scenario. API data encryption is required;

"iAM Smart" API (POST: Request Profile)

Step 2. "iAM Smart" System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the "iAM Smart" Profile request to Online Service Server by returning POST response with parameter "authByQR" (set to "true") in this scenario;

Step 3. Online Service Website/App shows instructions to inform "iAM Smart" user to process the "iAM Smart" Profile authorisation request in "iAM Smart" Mobile App;

Step 4. The Online Service Website/App should keep synchronising with the Online Service server for the request result (e.g., polling);

Step 5. The "iAM Smart" System pushes a notification message to the "iAM Smart" Mobile App based on the Tokenised ID;

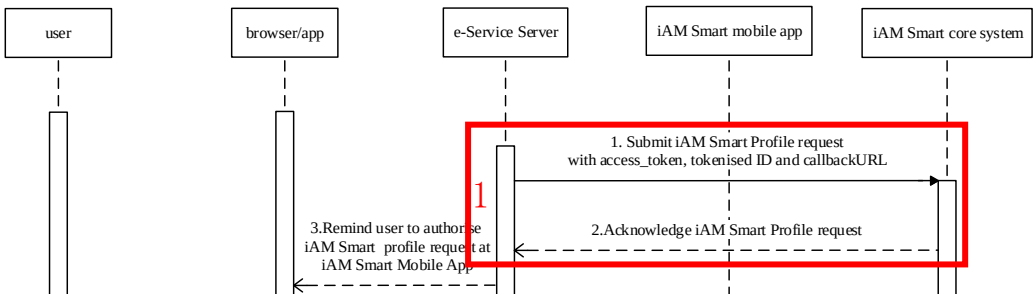
Step 6-8. "iAM Smart" user logs in "iAM Smart" Mobile App, reviews the "iAM Smart" Profile authorisation request (e.g., continue or reject the request) and authorises those selected "iAM Smart" Profile items to Online Service (e.g., authorise only his HKIC number but no other items);

Step 8-9. "iAM Smart" System invokes Online Service callback API to return the result with the "businessID" of the "iAM Smart" Profile request. API data decryption is required;

Online Service Callback API (POST: Callback to Receive "iAM Smart" Profile)

Step 10. The Online Service Server processes and matches the result using the "businessID" and shows the result in the corresponding Online Service Website/App.

A.1.1.1 Implementing (1) POST: Request Profile



Pre-conditions

- Online Service must possess a valid accessToken of the "iAM Smart" user with authorisation scope "Authentication";
- Online Service Website/App and "iAM Smart" Mobile App are in different devices in

this scenario;

- Online Service generates the "businessID" for this "iAM Smart" Profile request and uses this identifier to match the callback result returned from "iAM Smart" System;
- API data encryption is required.

Post-conditions

- Online Service server should check the response parameters "authByQR" and "ticketID" and determine the next action. For details, please refer to Section 3.2.1. "ticketID" will not be provided by "iAM Smart" System in this scenario.
- Online Service Website/App should show instructions to inform "iAM Smart" user to process the "iAM Smart" Profile request in "iAM Smart" Mobile App when "authByQR" is "true" is returned.
- Online Service Website/App should keep synchronising with Online Service server for the result returned from "iAM Smart" System.
- API data decryption is required.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D50002	Failed to request profile	Inform user the "iAM Smart" Profile request is failed and retry later

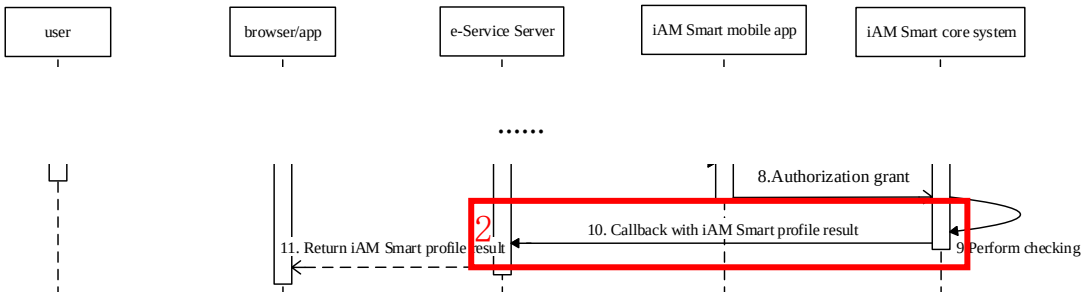
Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- Request parameter "source": Value should be matched with Online Service client terminal (e.g., "App_Scheme"/"App_Link" or browser's user agent value).
- Request Parameter - "redirectURI": Value should equal to URI of "Callback to Receive "iAM Smart" Profile" Online Service callback API. It must be in the same value as provided during Online Service registration.
- Request parameter "profileFields": It specifies the "iAM Smart" Profile items required by Online Service. The item name must match with the specification of this "iAM Smart" API.
- Response parameter "authByQR": The value returned by "iAM Smart" System will be "true" in this scenario.
- "iAM Smart" System will not return the response parameter "ticketID" in this scenario.

A.1.1.2 Implementing (2) POST: Callback to Receive “iAM Smart” Profile



Pre-conditions

- “iAM Smart” user can accept the request and authorise selected “iAM Smart” Profile items to Online Service.
- “iAM Smart” user can also reject the “iAM Smart” Profile request.

Post-conditions

- API data decryption is required.
- Online Service Server should match the callback result with corresponding Online Service Website/App using the “businessID”.
- Online Service Server should check the “iAM Smart” Profile items that “iAM Smart” user has authorised.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D50001	User rejected profile request	Inform user the “iAM Smart” Profile request is rejected
code - D50002	Failed to request profile	Inform user the “iAM Smart” Profile request is failed and retry later
code - D50003	Profile request timeout	Inform user the “iAM Smart” Profile request is timeout and provide way to user to retry

Callback Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.

- "iAM Smart" Profile items that is not authorised by "iAM Smart" user will not be returned in the result (i.e., JSON name/value of that item will not exist in result).

A.1.2 Scenario 2: Get "iAM Smart" Profile (Online Service Website in Same Device)

The sequence diagram below shows how an authenticated user authorises an Online Service to use his/her "eMEFields" for form filling when Online Service website and the "iAM Smart" Mobile App are running in the same device.

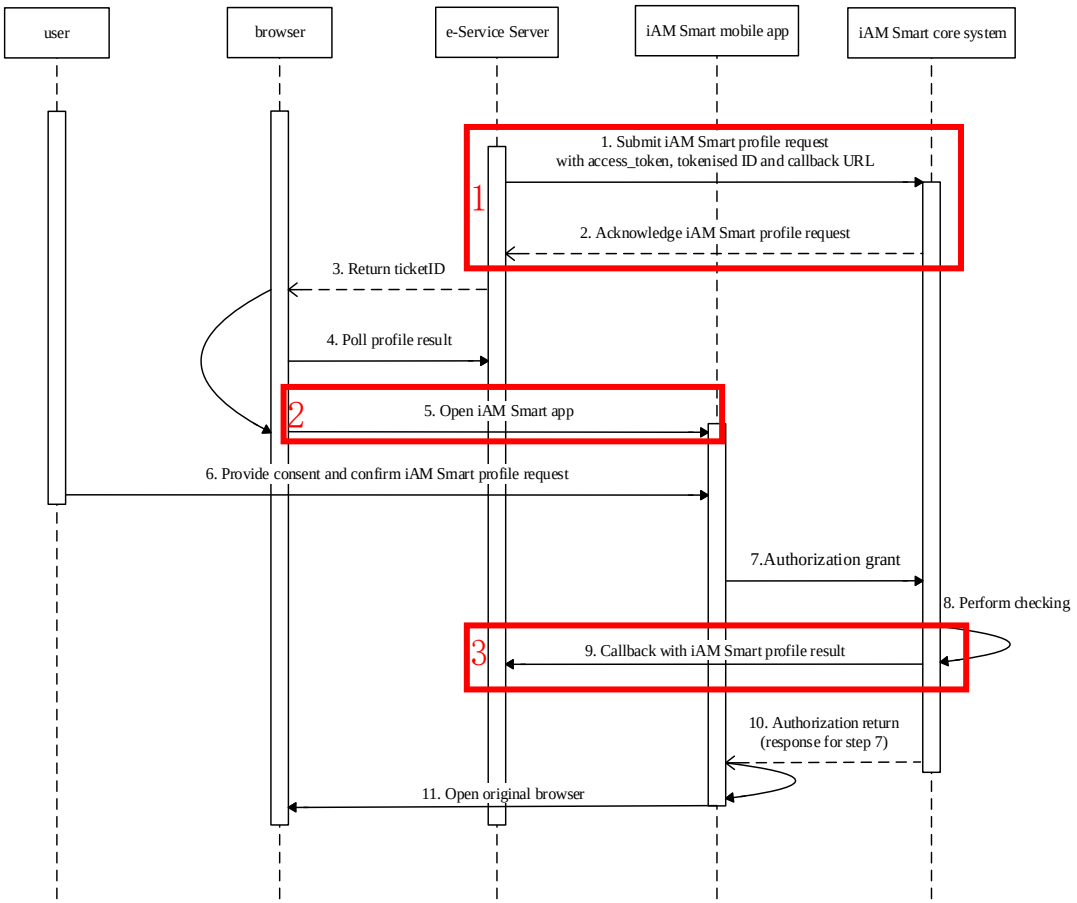


Figure-28 Get "iAM Smart" Profile (Online Service Website in Same Device)

Step 1. Online Service Server initiates an "iAM Smart" Profile request to invoke the "iAM Smart" API with the "businessID" of this request, accessToken, Tokenised ID and required "iAM Smart" Profile items. The request parameter "source" is browser's user agent value. API data encryption is required;

"iAM Smart" API (POST: Request Profile)

Step 2. "iAM Smart" System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the "iAM Smart" Profile request to Online Service Server by returning POST response with parameter "authByQR" (set to "false") and "ticketID";

Step 3. After receiving the response, Online Service Server prepares and sends necessary parameters such as "authByQR", "ticketID", etc. to Online Service Website;

Step 4. Online Service Website should keep synchronising with the Online Service server for the request result (e.g., polling);

Step 5. Online Service Website invokes "iAM Smart" Mobile App using URL Scheme with "ticketID" (set <Context> as "profile" in URL Scheme). Other parameters of this API are not used in this scenario;

"iAM Smart" API (URL Scheme: Open "iAM Smart" Mobile App for Getting Context)

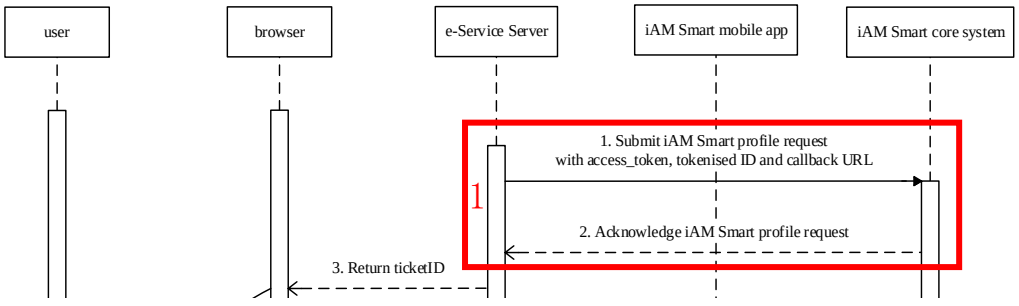
Step 6-7. "iAM Smart" user logs in "iAM Smart" Mobile App, reviews the "iAM Smart" Profile authorisation request (e.g., continue or reject the request) and authorises those selected "iAM Smart" Profile items to Online Service (e.g., authorise only his HKIC number but no other items);

Step 8-9. "iAM Smart" System invokes Online Service callback API to return the result with the "businessID" of the "iAM Smart" Profile request. API decryption is required;

Online Service Callback API (POST: Callback to Receive "iAM Smart" Profile)

Step 10-11. "iAM Smart" System instructs "iAM Smart" Mobile App to invoke the original Online Service browser (i.e., using the browser's user agent value submitted in "iAM Smart" Profile request in Step 1).

A.1.2.1 Implementing (1) POST: Request Profile



Pre-conditions

- Please refer to Section 20.9338264.1455428365.509163592A.1.1.1 except:
  - Online Service Website and “iAM Smart” Mobile App are in the same device in this scenario.

Post-conditions

- Please refer to Section 20.9338264.1455428365.509163592A.1.1.1 except:
  - Response “authByQR” is “false”, “ticketID” will be provided by “iAM Smart” System in this scenario.

Error conditions

- Please refer to Section 20.9338264.1455428365.509163592A.1.1.1.

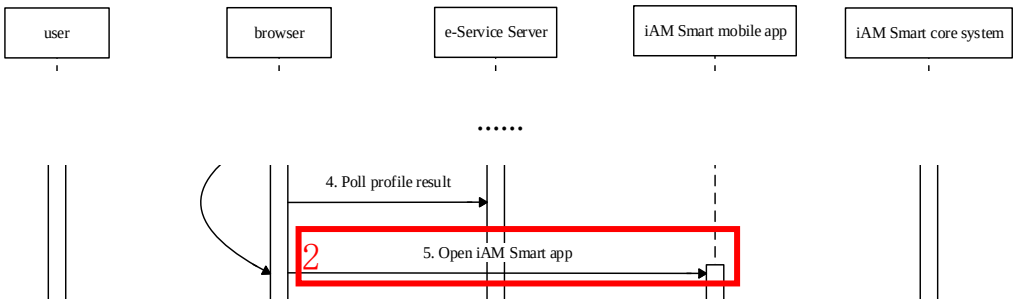
Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Please refer to Section 20.9338264.1455428365.509163592A.1.1.1, except for the following parameters:
  - Request parameter “source”: Value should be the browser's user agent value.
  - Response parameter “authByQR”: The value is “false” in this scenario.
  - Response parameter “ticketID”: It will be provided by “iAM Smart” System in this scenario. It is a 36-byte (or less) UUID number (ASCII character set).

A.1.2.2 Implementing (2) URL Scheme: Open “iAM Smart” Mobile App for Getting Context



Pre-conditions

- Online Service Website/App and “iAM Smart” Mobile App are in the same device;
- Online Service has the “ticketID” provided by “iAM Smart” System for the “iAM Smart” Profile request;
- Other parameters of this API are not used in this scenario.

Post-conditions

- “iAM Smart” Mobile App will be launched.
- Online Service Website/App should keep synchronising with Online Service server for the callback response of the “iAM Smart” Profile request from “iAM Smart” System.

Error conditions

- Nil

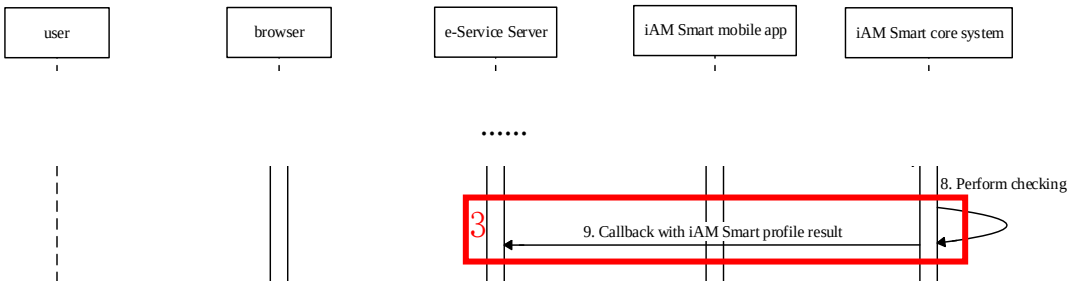
Request Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Set <Context> as “profile” in URL Scheme.

A.1.2.3 Implementing (3) POST: Callback to Receive “iAM Smart” Profile



Please refer to Section 20.9338264.1455428365.509163592A.1.1.2.

A.1.3 Scenario 3: Get "iAM Smart" Profile (Online Service App in Same Device)

The sequence diagram below shows how Online Service gets user additional profile information right after authentication when Online Service mobile application and the "iAM Smart" Mobile App are running the same device.

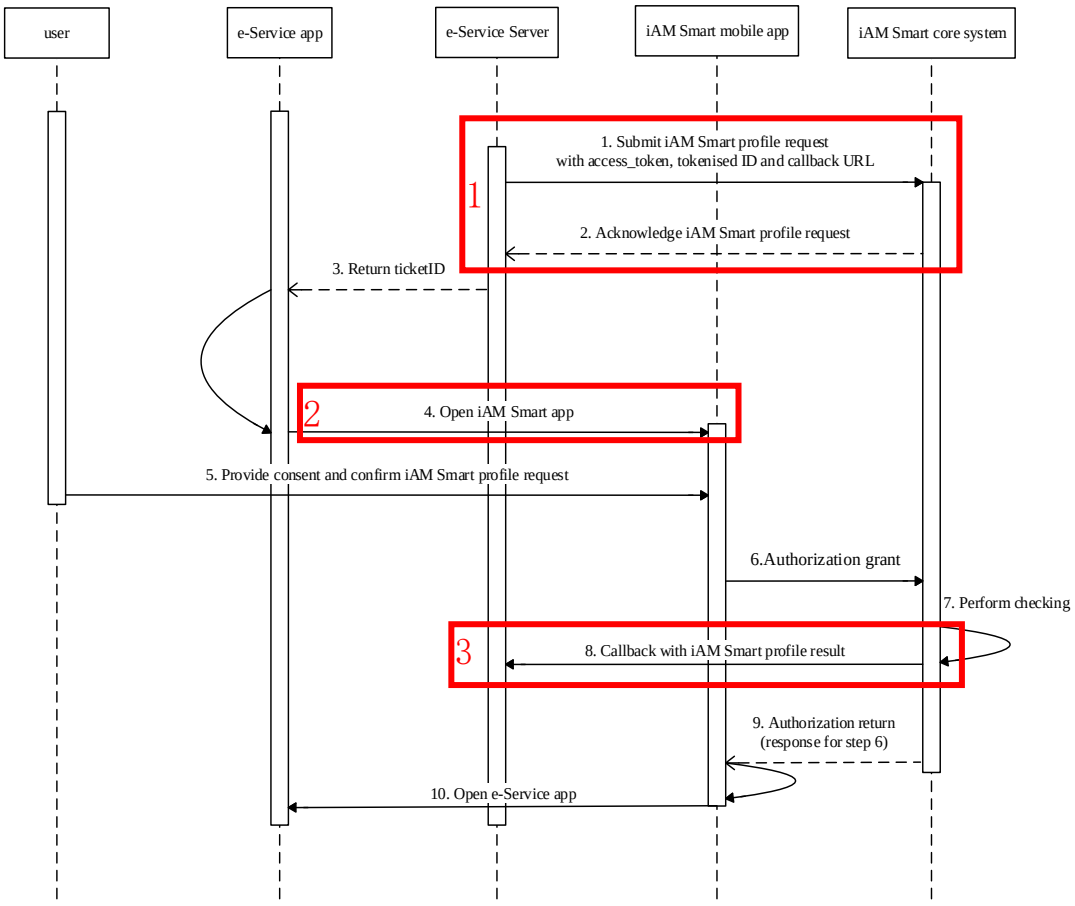


Figure-29 Get "iAM Smart" Profile (Online Service App in Same Device)

Step 1. Online Service Server initiates "iAM Smart" Profile request to invoke the "iAM Smart" API with the "businessID" of this request, accessToken, Tokenised ID and required "iAM Smart" Profile items. The request parameter "source" is "App_Scheme" or "App_Link". API data encryption is required;

"iAM Smart" API (POST: Request Profile)

Step 2. "iAM Smart" System verifies the accessToken, Tokenised ID, other parameters and confirms receipt of the "iAM Smart" Profile request to Online Service Server by returning POST response with parameter "authByQR" (set to "false") and "ticketID";

Step 3. After receiving the response, Online Service Server prepares and sends necessary parameters such as “authByQR”, “ticketID”, etc. to Online Service App;

Step 4. Online Service App determines “authByQR” to be false, or uses program code to check if “iAM Smart” Mobile App exists in the same device, then invokes “iAM Smart” Mobile App using URL Scheme with “ticketID” (set <Context> as “profile” in URL Scheme). Other parameters of this API are not used in this scenario. Online Service App should keep synchronising with the Online Service server for the request result (e.g., polling);

“iAM Smart” API (URL Scheme: Open “iAM Smart” Mobile App for Getting Context)

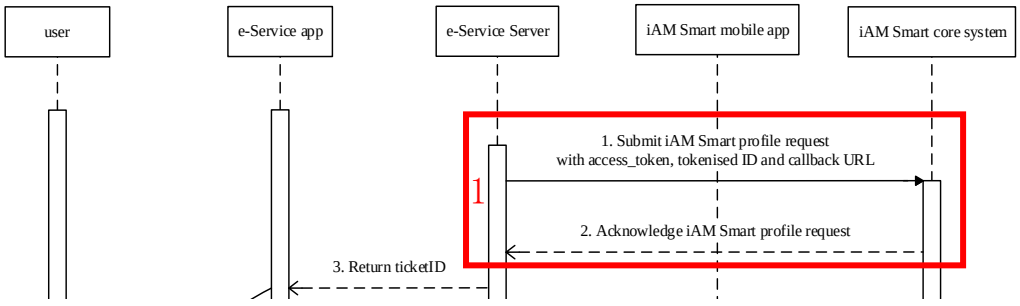
Step 5-6. “iAM Smart” user logs in “iAM Smart” Mobile App, reviews the “iAM Smart” Profile authorisation request (e.g., continue or reject the request) and authorises those selected “iAM Smart” Profile items to Online Service (e.g., authorise only his HKIC number but no other items);

Step 7-8. “iAM Smart” System invokes Online Service callback API to return the result with the “businessID” of the “iAM Smart” Profile request. API decryption is required;

Online Service Callback API (POST: Callback to Receive “iAM Smart” Profile)

Step 9-10. “iAM Smart” System instructs “iAM Smart” Mobile App to invoke Online Service App using URL Scheme, or Universal Link/App Link (“iAM Smart” System queries the information registered at Online Service registration depending on the “source” submitted in Step 1).

A.1.3.1 Implementing (1) POST: Request Profile



Pre-conditions

- Please refer to Section 20.9338264.1455428365.509163592A.1.1.1 except:
  - Online Service App and “iAM Smart” Mobile App are in the same device in this scenario.

Post-conditions

- Please refer to Section 20.9338264.1455428365.509163592A.1.1.1.

Error conditions

- Please refer to Section 20.9338264.1455428365.509163592A.1.1.1.

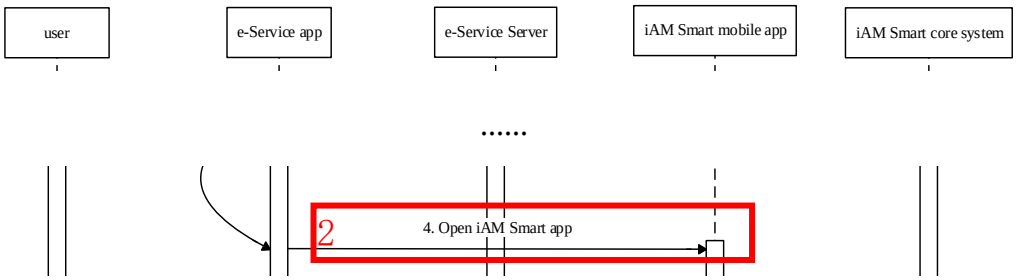
Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

Notes

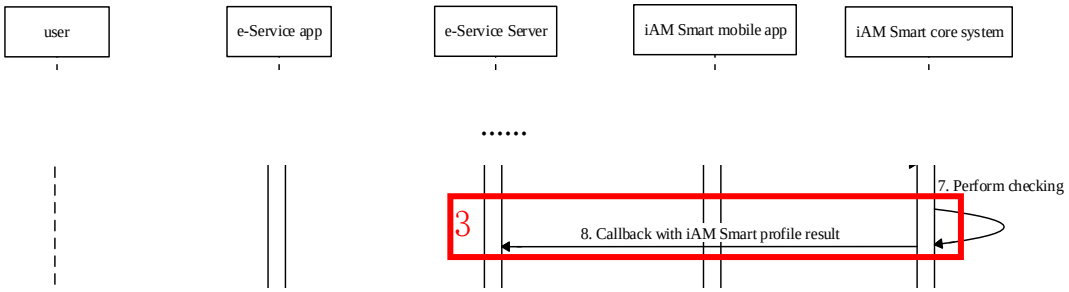
- Please refer to Section 20.9338264.1455428365.509163592A.1.1.1, except for the following parameter:
  - Request parameter "source": Value should be "App_Scheme" (for the Online Service App URL scheme), or "App_Link" (for the Universal Link/App Link).

A.1.3.2 Implementing (2) URL Scheme: Open "iAM Smart" Mobile App for Getting Context



Please refer to Section 20.9338264.1455428365.509163592A.1.2.2.

A.1.3.3 Implementing (3) POST: Callback to Receive "iAM Smart" Profile



Please refer to Section 20.9338264.1455428365.509163592A.1.1.2.

A.2 Workflows for Form Filling with Service Login (v1 and v2)

A.2.1. Scenario 1: Form Filling (Online Service Website/App in Different Device)

The sequence diagram below shows how an authenticated user authorises an Online Service to use his/her "eMEFields" for form filling when Online Service and the "iAM Smart" Mobile App are running in different devices.

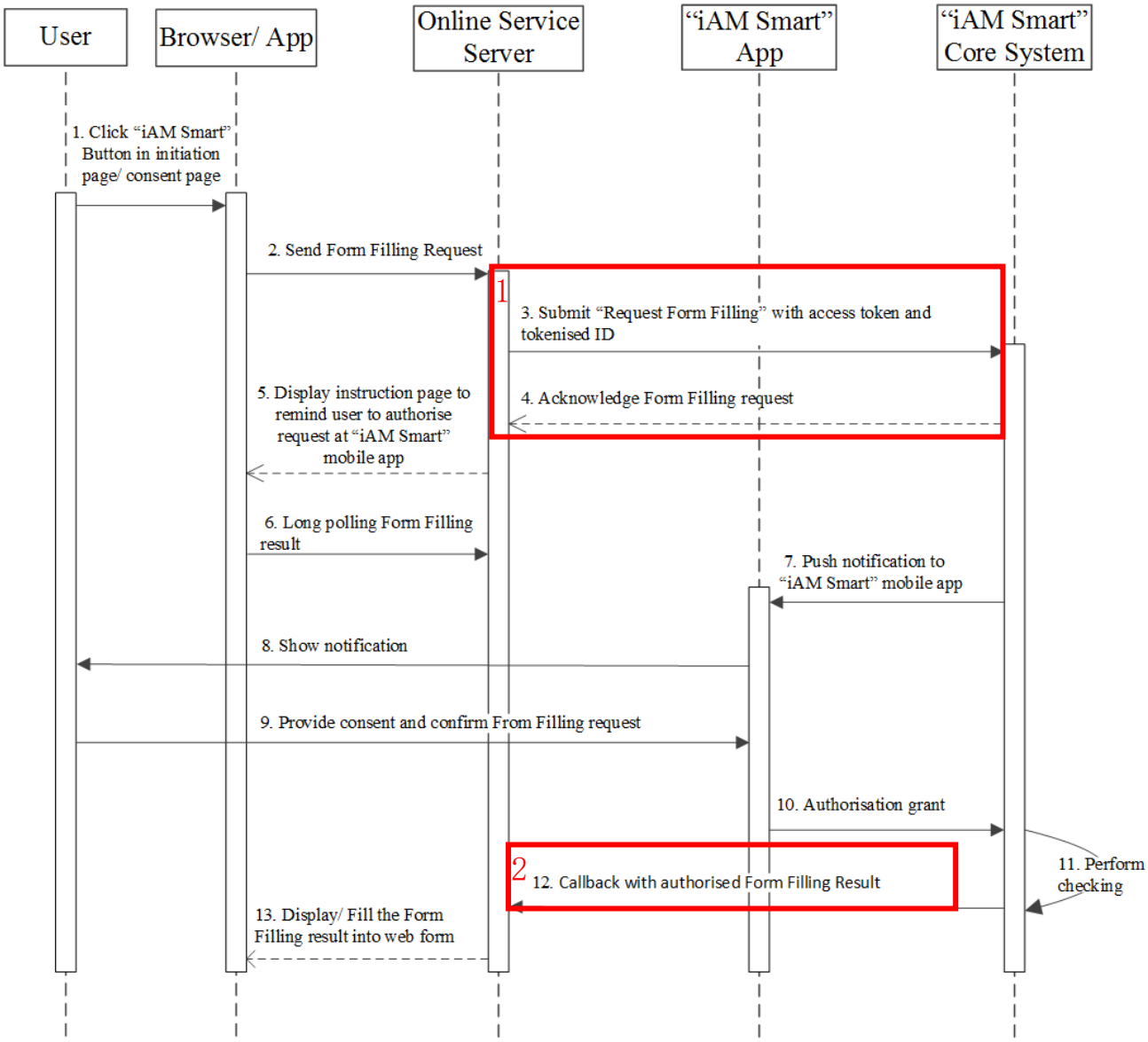


Figure-30 Form Filling (Online Service Website/App in Different Device)

Step 1. User reads the consent page (if "iAM Smart" profile fields requested) and clicks the button in Online Service Website/App;

Step 2. Online Service Website/App initiates form filling request to Online Service Server with browser's user agent name (for Online Service Website) or

"App_Scheme"/"App_Link" (for Online Service App) in this scenario (i.e., for use as value for request parameter "source");

Step 3. Online Service Server initiates a Form Filling request to invoke the "iAM Smart" API with the "businessID" of this request, accessToken, Tokenised ID, required "eMEFields", etc. The request parameter "source" will be the browser's user agent value or "App_Scheme"/"App_Link" in this scenario. API encryption is required;

"iAM Smart" API (POST: Request Form Filling)

Step 4. "iAM Smart" System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the Form Filling request to Online Service Server by returning POST response with parameter "authByQR" (set to "true") in this scenario;

Step 5-6. Online Service Website/App shows instruction to inform "iAM Smart" user to process the Form Filling authorisation request in "iAM Smart" Mobile App. The Online Service Website/App should keep synchronising with the Online Service server for the request result (e.g., polling);

Step 7. "iAM Smart" System pushes a notification message to the "iAM Smart" Mobile App based on the Tokenised ID;

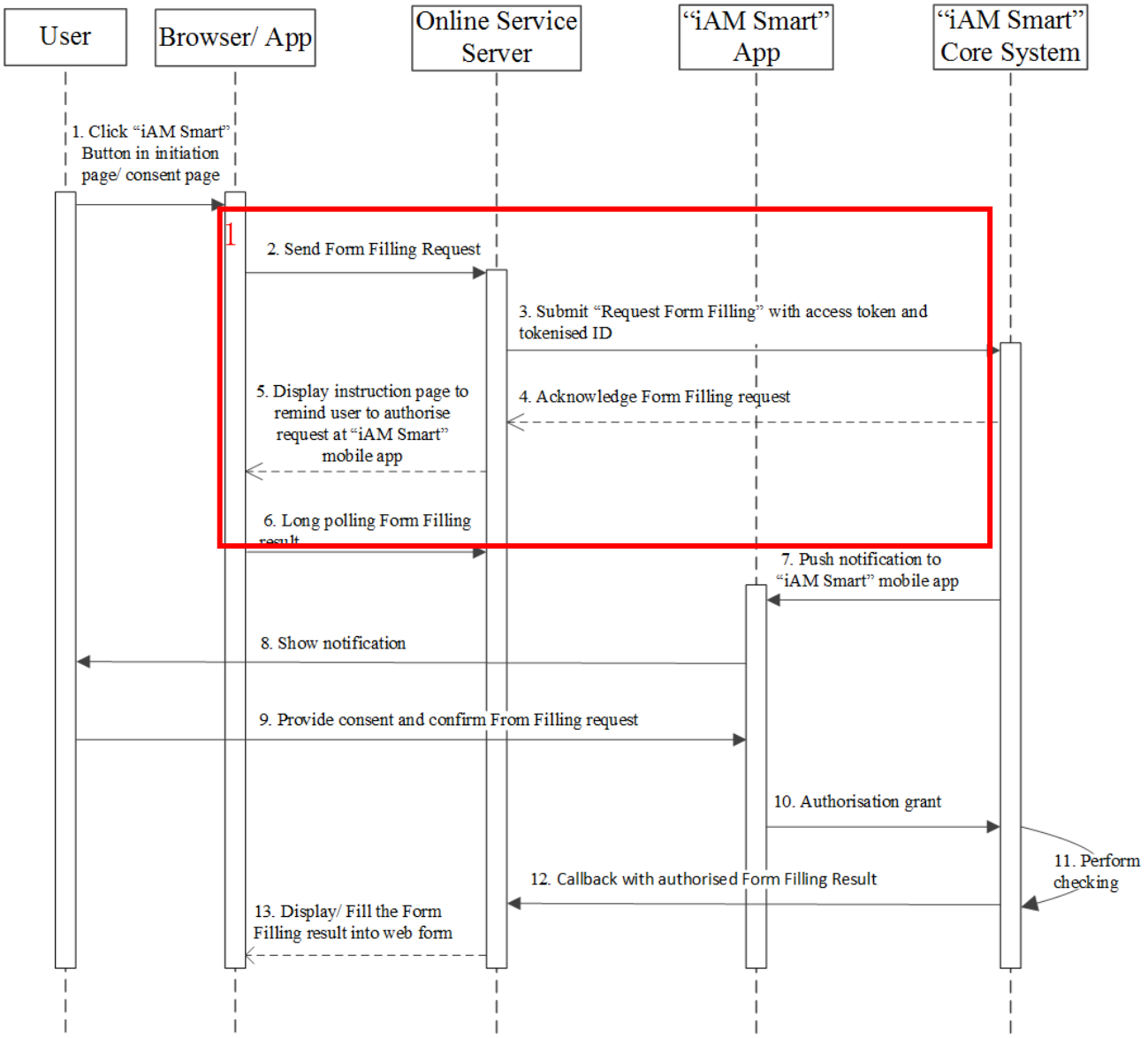
Step 8-10. "iAM Smart" user logs in "iAM Smart" Mobile App, reviews the Form Filling authorisation request (e.g. continue or reject the request) and authorises those selected "eMEFields" to Online Services (e.g. authorise HKIC number, English name, residential address in "e-ME profile", etc.);

Step 11-12. "iAM Smart" System invokes Online Service callback API to return the result with the "businessID" of the form filling request. API decryption is required;

Online Service Callback API (POST: Callback to Receive Form Filling Information)

Step 13. Online Service Server processes and matches the result using the "businessID" and shows the result in the corresponding Online Service Website/App.

A.2.1.1 Implementing (1) POST: Request Form Filling



Pre-conditions

- Online Service must possess a valid accessToken of the "iAM Smart" user with authorisation scope "Form Filling authorisation".
- Online Service Website/App and "iAM Smart" Mobile App are in different devices in this scenario.
- Online Service generates a "businessID" for this Form Filling request and uses this identifier to match the callback result returned from "iAM Smart" System.
- API data encryption is required.

Post-conditions

- Online Service server should check the response parameter "authByQR" and "ticketID" and determine the next action. For details, please refer to Section 3.2.1. "ticketID"

will not be provided by “iAM Smart” System in this scenario.

- Online Service Website/App should show instructions to inform “iAM Smart” user to process the Form Filling request in “iAM Smart” Mobile App when “authByQR” is “true” is returned.
- Online Service Website/App should keep synchronising with Online Service server for the result returned from “iAM Smart” System.
- API data decryption is required.

Error conditions

- For common errors, please refer to Section 3.2.3. Other error of this API includes:

Error Code	Error Description	Suggested Action
code - D60002	Failed to request Form Filling	Inform user the Form Filling request is failed and retry later

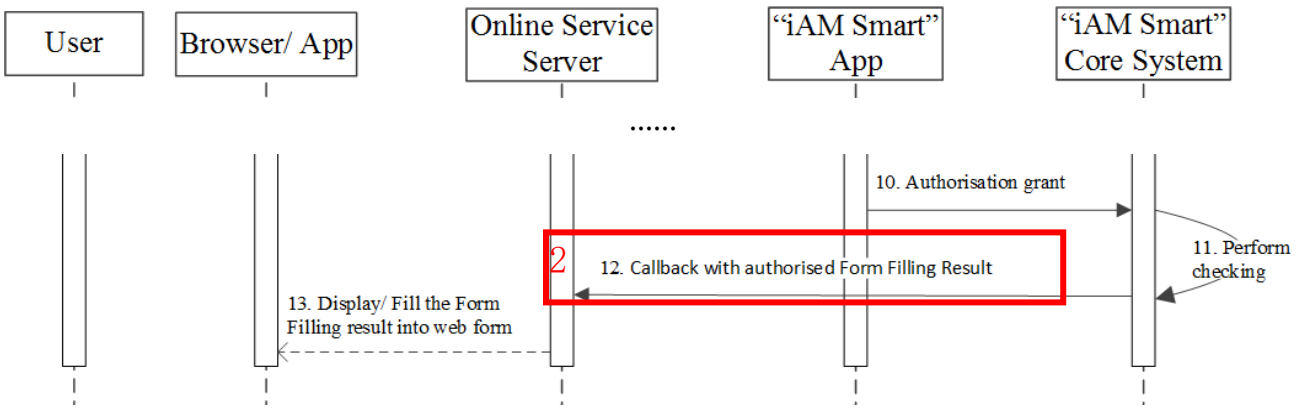
Request and Response Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- Request parameter “source”: Value should be matched with Online Service client terminal (e.g., “App_Scheme”/“App_Link” or browser’s user agent value).
- Request Parameter “redirectURI”: Value should equal to URI of “Callback to Receive Form Filling Information” Online Service callback API. It must be in the same value as provided during Online Service registration.
- Request parameters “formName”, “formNum”, “formDesc”: They are the information of the Online Service's form and will be shown in “iAM Smart” Mobile App for “iAM Smart” user's reference. Online Service should determine the language of these items.
- Request parameter “eMEFields”: It specifies the account information and data voluntarily provided in “e-ME profile” required by Online Service. The item name must match with the specification of this “iAM Smart” API. Otherwise, error code for the invalid parameter will be returned.
- Request parameter “profileFields”: It specifies the “iAM Smart” profile fields required by Online Service. The item name must match with the specification of this “iAM Smart” API. Otherwise, error code for the invalid parameter will be returned.
- Response parameter “authByQR”: The value returned by “iAM Smart” System will be “true” in this scenario.
- “iAM Smart” System will not return the response parameter “ticketID” in this scenario.

A.2.1.2 Implementing (2) POST: Callback to Receive Form Filling Information



Pre-conditions

- "iAM Smart" user can accept the request and authorise data fields to Online Service.
- "iAM Smart" user can also reject the form filling request.

Post-conditions

- API data decryption is required.
- Online Service Server should match the callback result with corresponding Online Service Website/App using the "businessID".
- Online Service Server should check the requested data fields has authorised for the Anonymous Form Filling.

Error conditions

- For common errors, please refer to Section 3.2.3. Other errors of this API include:

Error Code	Error Description	Suggested Action
code - D60000	User cancelled Form Filling request	Inform user the Form Filling request is cancelled
code - D60001	User rejected Form Filling request	Inform user the Form Filling request is rejected
code - D60002	Failed to request Form Filling	Inform user the Form Filling request is failed and retry later
code - D60003	Form Filling request timeout	Inform user the Form Filling request is timeout, and provide way for user to retry

Callback Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- For common parameters in request and response, please refer to Section 3.2.2.
- “eMEFields” that not authorised by “iAM Smart” user will not return in result (i.e., JSON name/value of that item will not exist in result).

A.2.2 Scenario 2: Form Filling (Online Service Website in Same Device)

The sequence diagram below shows how an authenticated user authorises an Online Service to use his/her "eMEFields" for form filling when Online Service website and the "iAM Smart" Mobile App are running in the same device.

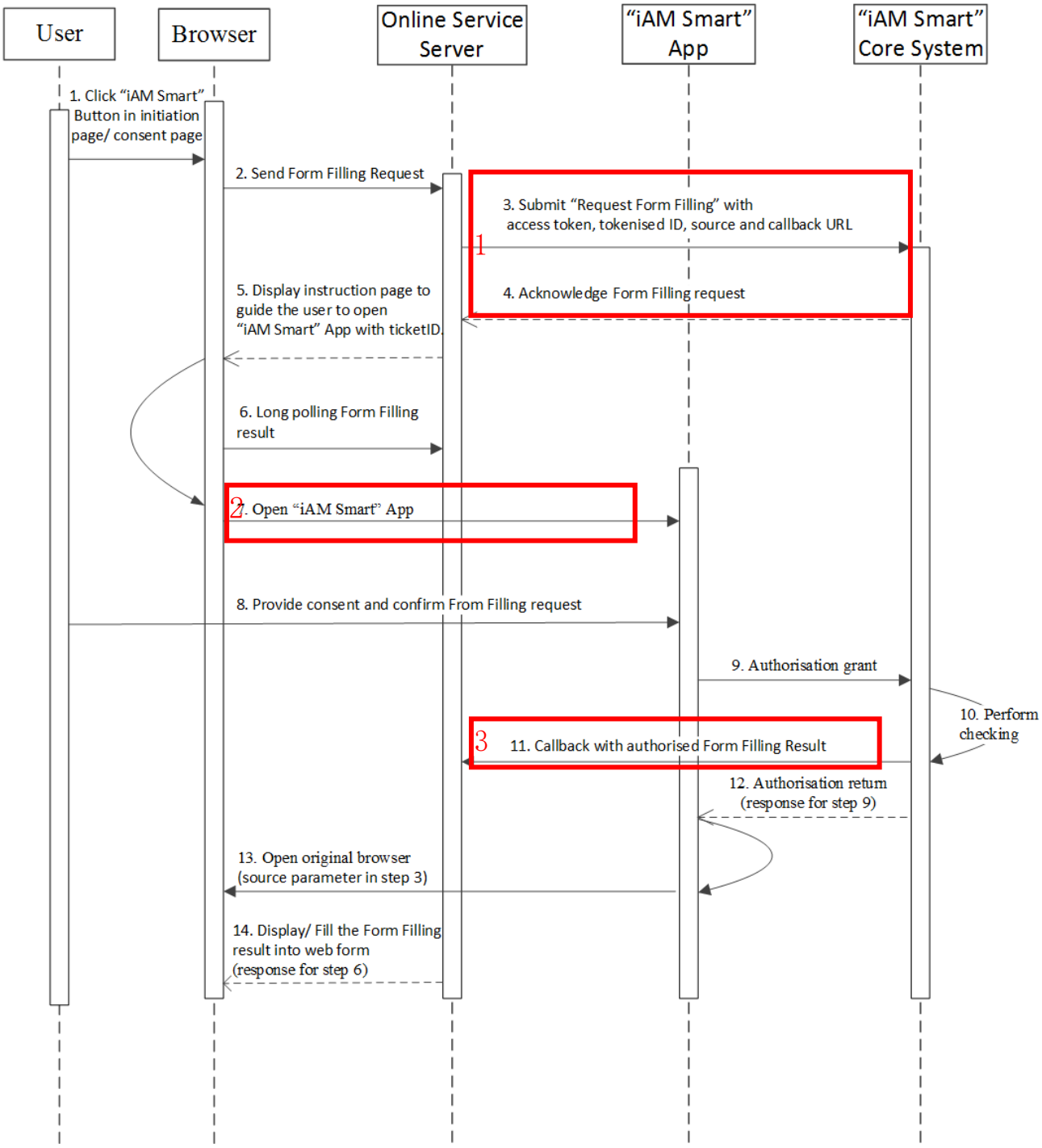


Figure-31 Form Filling (Online Service Website in Same Device)

Step 1. User reads the consent page (if "iAM Smart" profile fields requested) and clicks the button in Online Service Website;

Step 2. Online Service Website initiates form filling request to Online Service Server with browser's user agent name (i.e., for use as value for request parameter "source");

Step 3. Online Service Server initiates Form Filling request to invoke the "iAM Smart" API with the "businessID" of this request, accessToken, Tokenised ID, required "eMEFields", etc. The request parameter "source" will be the browser's user agent value. API data encryption is required;

"iAM Smart" API (POST: Request Form Filling)

Step 4. "iAM Smart" System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the Form Filling request to Online Service Server by returning POST response with parameter "authByQR" (set to "false") and "ticketID" in this scenario;

Step 5. After receiving the response, Online Service Server prepares and sends necessary parameters such as "authByQR", "ticketID", etc. to Online Service Website;

Step 6-7. Online Service Website invokes "iAM Smart" Mobile App using URL Scheme with "ticketID" (set <Context> as "form-filling" in URL Scheme). Other parameters of this API are not used in this scenario. The Online Service Website should keep synchronising with the Online Service Server for the request result (e.g., polling). API data encryption is required;

"iAM Smart" API (URL Scheme: Open "iAM Smart" Mobile App for Getting Context)

Step 8-10. "iAM Smart" user logs in "iAM Smart" Mobile App, reviews the Form Filling authorisation request (e.g. continue or reject the request) and authorises those selected "eMEFields" to Online Services (e.g. authorise HKIC number and English name in "iAM Smart" account information and residential address in "e-ME profile" (unchecked other "eMEFields"));

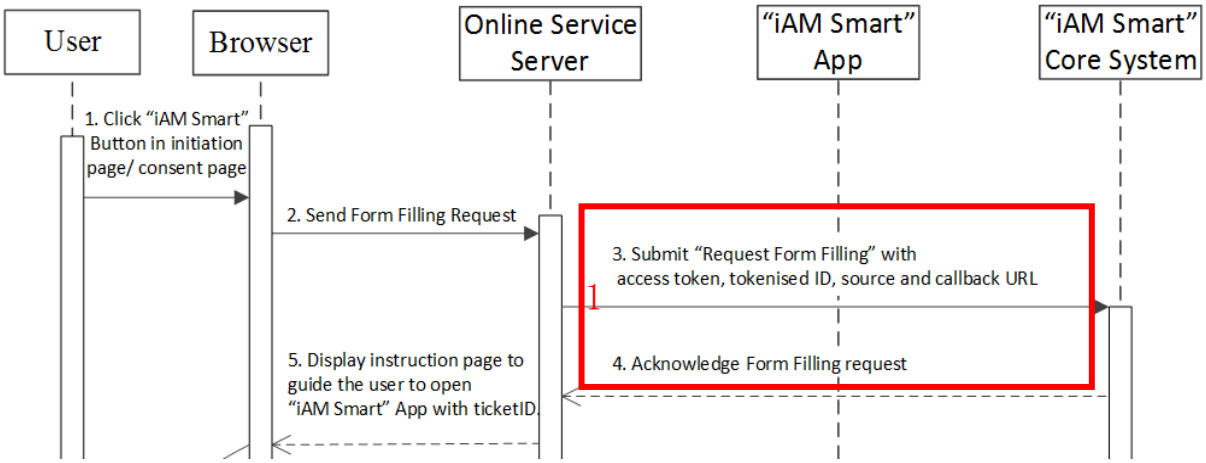
Step 11. "iAM Smart" System invokes Online Service callback API to return the result with the "businessID" of the Form Filling request. API decryption is required;

Online Service callback API (POST: Callback to Receive Form Filling Information)

Step 12-13. "iAM Smart" System instructs "iAM Smart" Mobile App to invoke the original Online Service browser (i.e., using the browser's user agent value submitted in form filling request in Step 3);

Step 14. Online Service server processes and matches the result using the "businessID" and shows the result in the corresponding Online Service Website.

A.2.2.1 Implementing (1) POST: Request Form Filling



Pre-conditions

- Please refer to Section 20.9338264.1455428365.509163592A.2.1.1 except:
  - Online Service Website and "iAM Smart" Mobile App are in the same device in this scenario.

Post-conditions

- Please refer to Section 20.9338264.1455428365.509163592A.2.1.1 except:
  - Response "authByQR" is "false", "ticketID" will be provided by "iAM Smart" System in this scenario.

Error conditions

- Please refer to Section 20.9338264.1455428365.509163592A.2.1.1

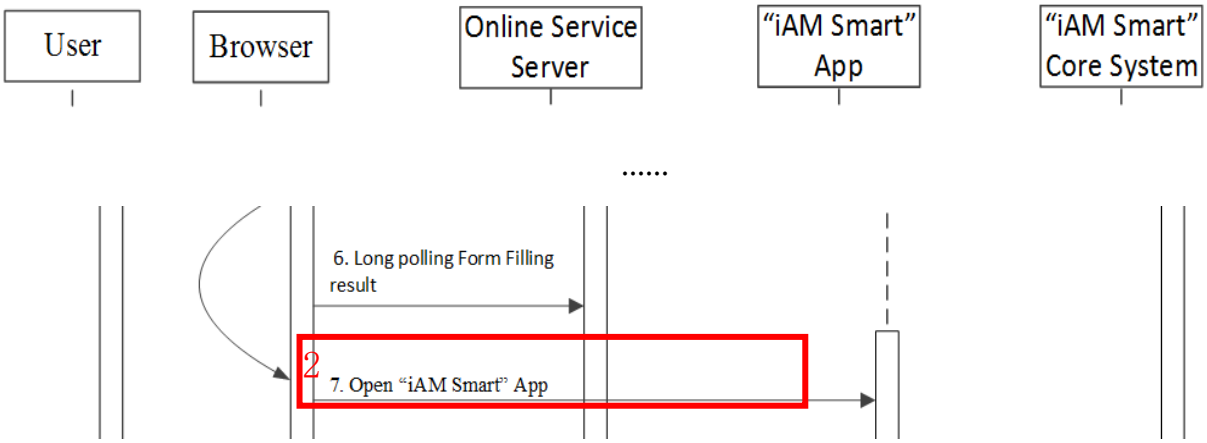
Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- Please refer to Section 20.9338264.1455428365.509163592A.2.1.1, except for the following parameters:
  - Request parameter "source": Value should be the browser's user agent value.
  - Response parameter "authByQR": The value is "false" in this scenario.
  - Response parameter "ticketID": It will be provided by "iAM Smart" System in this scenario. It is a 36-byte (or less) UUID number (ASCII character set).

A.2.2.2 Implementing (2) URL Scheme: Open “iAM Smart” Mobile App for Getting Context



Pre-conditions

- Online Service Website and “iAM Smart” Mobile App are in the same user device.
- Online Service has the “ticketID” provided by “iAM Smart” System for the form filling request;
- Other parameters of this API are not used in this scenario.

Post-conditions

- “iAM Smart” Mobile App will be launched.
- Online Service Website should keep synchronising with Online Service server for the callback response of the form filling request from “iAM Smart” System.

Error conditions

- Nil

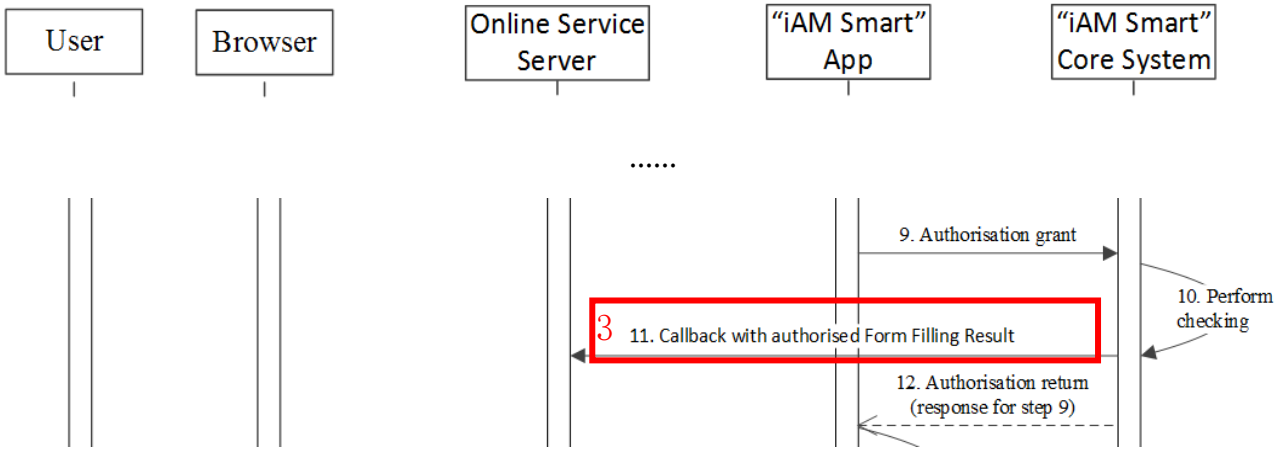
Request Parameters

- Please refer to “iAM Smart” API Specification.

Notes

- Set <Context> as “form-filling” in URL Scheme.

A.2.2.3 Implementing (3) POST: Callback to Receive Form Filling Information



Please refer to Section 20.9338264.1455428365.509163592A.2.1.2.

A.2.3 Scenario 3: Form Filling (Online Service App in Same Device)

The sequence diagram below shows how an authenticated user authorises an Online Service to use his/her "eMEFields" for form filling when Online Service App and the "iAM Smart" Mobile App are running in the same device.

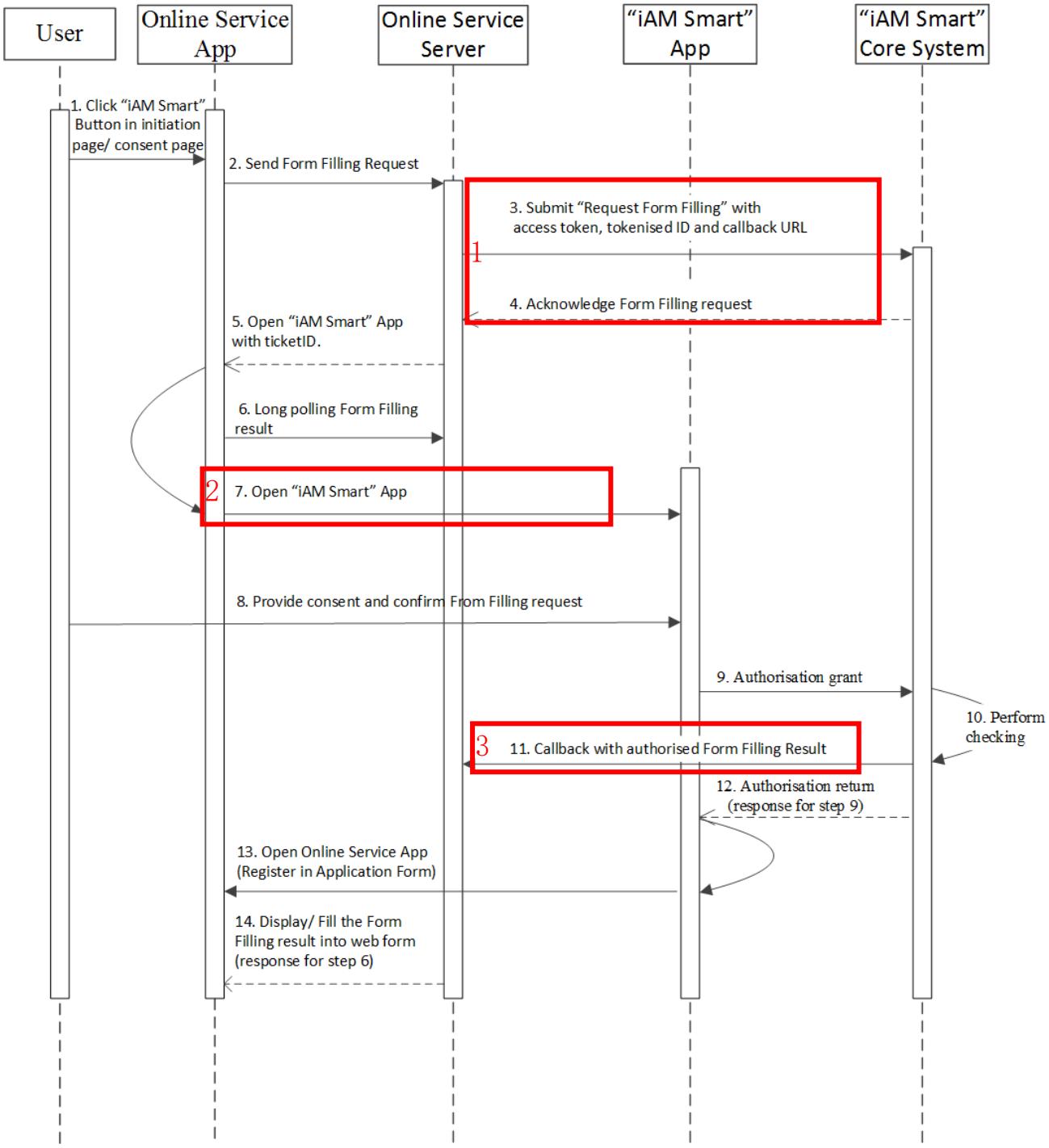


Figure-32 Form Filling (Online Service App in Same Device)

Step 1. User reads the consent page (if "iAM Smart" profile fields requested) and clicks the button in Online Service App;

Step 2. Online Service App initiates form filling request to Online Service Server with "App_Scheme" or "App_Link" (for use as value for request parameter "source");

Step 3. Online Service Server initiates form filling request to invoke the "iAM Smart" API with the "businessID" of this request, accessToken, Tokenised ID, required "e-ME" items, etc. The request parameter "source" will be "App_Scheme" or "App_Link". API data encryption is required;

"iAM Smart" API (POST: Request Form Filling)

Step 4. "iAM Smart" System verifies the validity of accessToken, Tokenised ID, other parameters and confirms receipt of the form filling request to Online Service Server by returning POST response with parameter "authByQR" (set to "false") and "ticketID" in this scenario;

Step 5. After receiving the response, Online Service Server prepares necessary parameters such as "authByQR", "ticketID", etc. to Online Service App;

Step 6-7. Online Service App determines "authByQR" to be false, or uses program code to check if "iAM Smart" Mobile App exists in the same device, then invokes "iAM Smart" Mobile App using URL Scheme with "ticketID" (set <Context> as "form-filling" in URL Scheme). Other parameters of this API are not used in this scenario. The Online Service App should keep synchronising with the Online Service Server for the request result (e.g., polling);

"iAM Smart" API (URL Scheme: Open "iAM Smart" Mobile App for Getting Context)

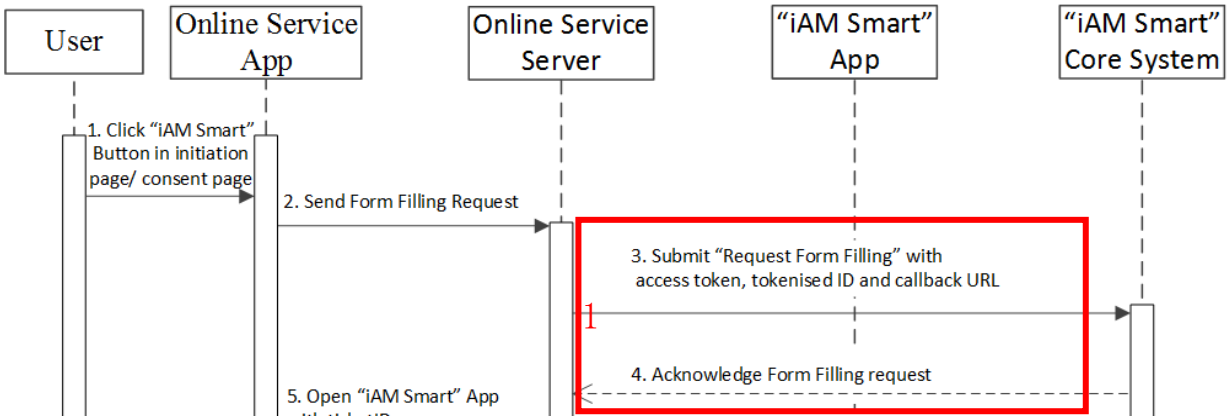
Step 8-9. "iAM Smart" user logs in "iAM Smart" Mobile App, reviews the Form Filling authorisation request (e.g. continue or reject the request) and authorises those selected "eMEFields" to Online Services (e.g. authorise HKIC number and English name in "iAM Smart" account information and residential address in "e-ME profile" (unchecked other "eMEFields));

Step 10-11. "iAM Smart" System invokes Online Service callback API to return the result with the "businessID" of the form filling request. API decryption is required;

Online Service Callback API (POST: Callback to Receive Form Filling Information)

Step 12-14. "iAM Smart" System instructs "iAM Smart" Mobile App to invoke the Online Service App using URL Scheme, or Universal Link/App Link ("iAM Smart" System queries the information registered at Online Service registration depending on the "source" submitted in Step 3).

A.2.3.1 Implementing (1) POST: Request Form Filling



Pre-conditions

- Please refer to Section 20.9338264.1455428365.509163592A.2.1.1 except:
  - Online Service App and "iAM Smart" Mobile App are in the same device in this scenario.

Post-conditions

- Please refer to Section 20.9338264.1455428365.509163592A.2.1.1.

Error conditions

- Please refer to Section 20.9338264.1455428365.509163592A.2.1.1.

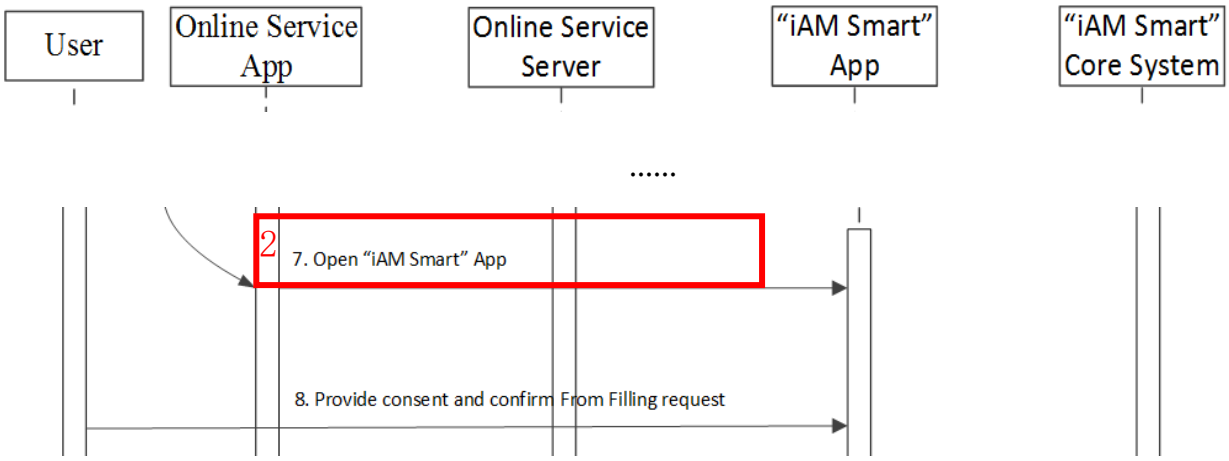
Request and Response Parameters

- Please refer to "iAM Smart" API Specification.

Notes

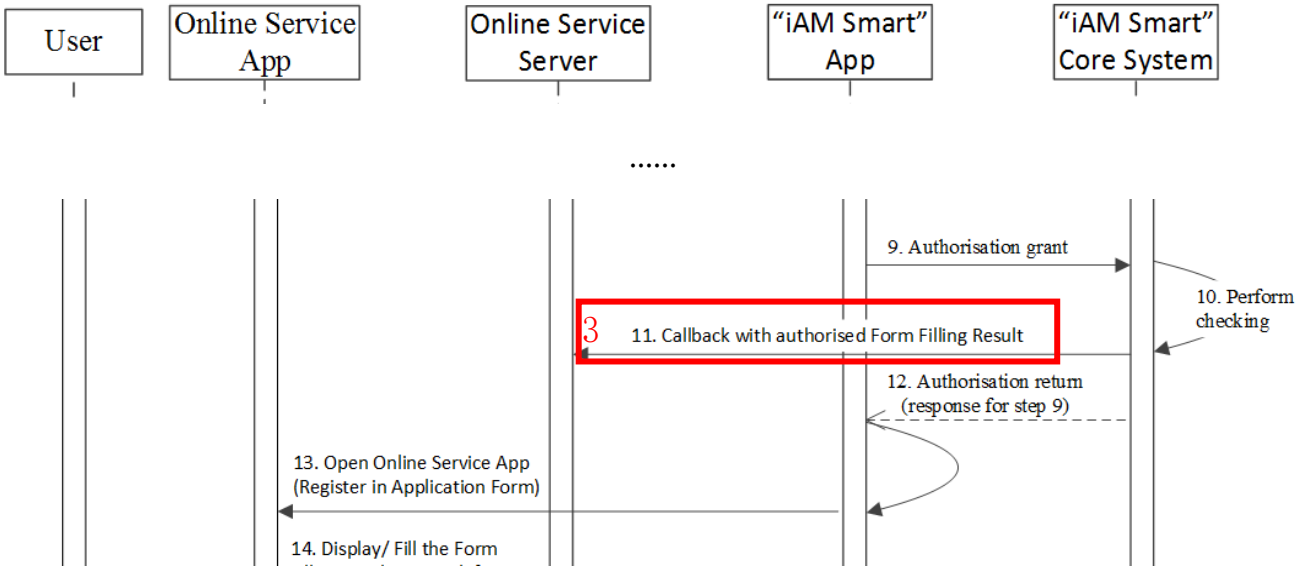
- Please refer to Section 20.9338264.1455428365.509163592A.2.1.1, except for the following parameter:
  - Request parameter "source": Value should be "App_Scheme" (for the Online Service App URL scheme), or "App_Link" (for the Universal Link/App Link).

A.2.3.2 Implementing (2) URL Scheme: Open "iAM Smart" Mobile App for Getting Context



Please refer to Section 20.9338264.1455428365.509163592A.2.2.2.

A.2.3.3 Implementing (3) POST: Callback to Receive Form Filling Information



Please refer to Section 20.9338264.1455428365.509163592A.2.2.3.

A.3 Workflows for Direct Login v1

A.3.1 Scenario 1: Direct Login with Online Service Website

The sequence diagram below shows how users initiate Direct Login in that "iAM Smart" Mobile App and then automatically login Online Service website opened in a supported mobile browser.

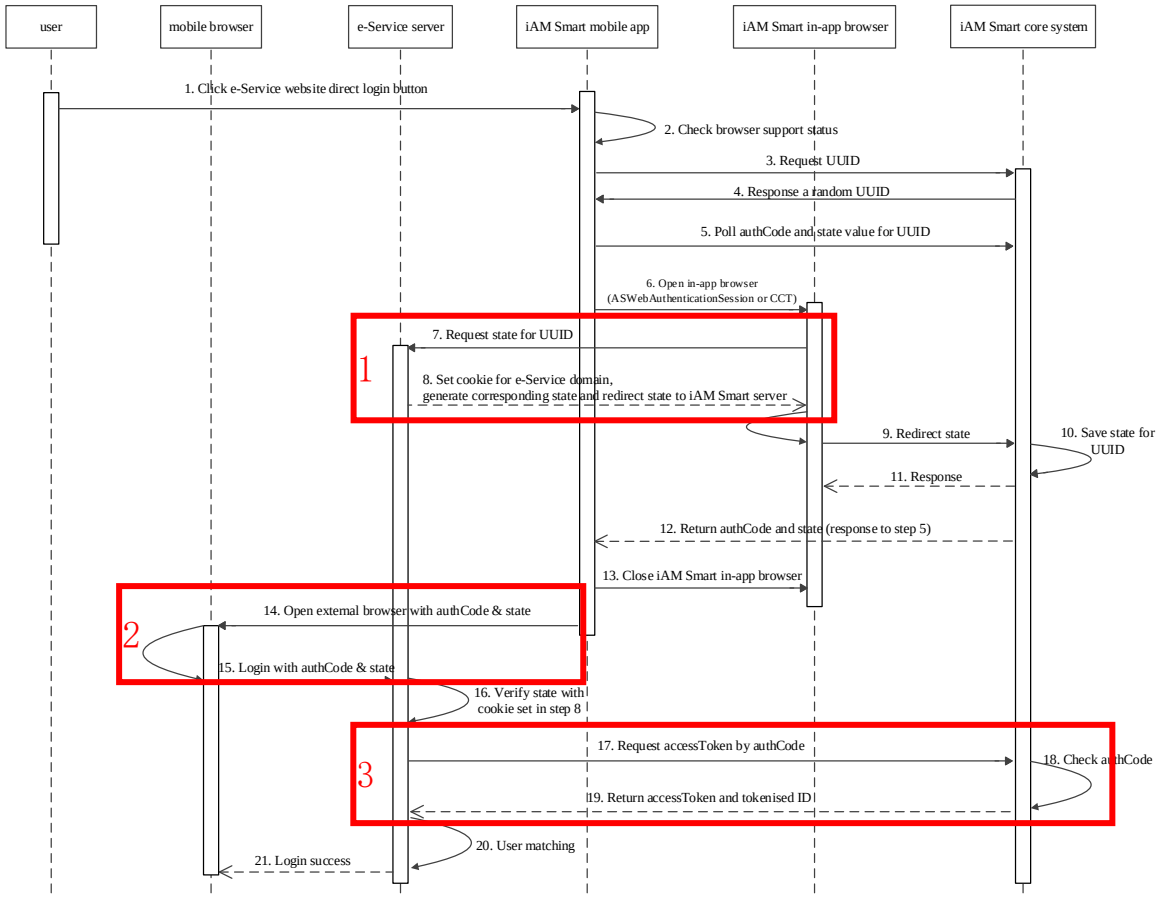


Figure-33 Direct Login with Online Service Website

Step 1. User opens "iAM Smart" Mobile App, browses Online Service catalogue or views the detailed information of Online Service, and clicks the button to invoke the Online Service;

Step 2. "iAM Smart" Mobile App checks if any of the supported mobile browsers (please refer to the table below) is installed. If there is no supported browser, the default browser will be opened and redirected to the fallback URL that has been registered in the "iAM Smart" Platform. Please refer to Section 3.10.3 for the fallback details;

	Android[5]	iOS
Supported Browser of Direct Login	Chrome	Safari

Step 3-4. "iAM Smart" Mobile App requests a random universal unique ID (UUID) from "iAM Smart" server;

Step 5. "iAM Smart" Mobile App polls "iAM Smart" server for the authCode and state corresponding to the UUID;

Step 6. "iAM Smart" Mobile App brings up its in-app browser;

Step 7-8. The request is redirected to an Online Service endpoint which should perform the following:

- ensure that the domain in the callback URI of the request belongs to "iAM Smart";
- set a unique persistent cookie for the current in-app browser session. For Android devices, Online Service could adopt an optional approach, which uses session cookie (e.g., JSESSIONID) instead of persistent cookie for session identification.
- generate a state (i.e., CSRF token) corresponding to the cookie. Online Service shall generate the state with the method suggested by Open Web Application Security Project (OWASP) based on HMAC Based Token Pattern, i.e., the state should be generated using HMAC function based on the cookie value and a timestamp. Specifically, the cookie value refers to the value of persistent cookie if Online Service sets a unique persistent cookie for the in-app browser session as mentioned in the second bullet above. If Online Service adopts the optional approach for Android devices based on session cookie, the value refers to the session ID. The timestamp is to prevent replay attack.
- redirect this state to the "iAM Smart" server.

Please refer to Section 3.10.1.1 for implementation details of the above Online Service endpoint, including the details of HMAC based token pattern for CSRF protection, security hardening for setting the state and cookie, and security protection for the Online Service endpoint. Online Service is strongly recommended to follow these practices. However, Online Service could adopt better or equivalent measures to mitigate the CSRF attack if feasible.

[5] More browsers will be supported by "iAM Smart" Plaform in the future in case Chrome is not available on the Android device.

Step 9-11. The previously redirected endpoint will save the state value for the UUID and prompts a webpage that indicates “iAM Smart” is directly logging into Online Service;

Step 12. Once the state for the given UUID is ready, the polling started in Step 5 will get its value and the corresponding authCode, and the UUID in the “iAM Smart” server will be discarded;

Step 13. “iAM Smart” Mobile App closes the in-app browser;

Step 14-15. “iAM Smart” Mobile App gets its current UI display language, opens the external browser, and requests one of the three direct login endpoints with the authCode and state. These three direct login endpoints must be registered in the “iAM Smart” Platform in advance. “iAM Smart” Mobile App chooses the endpoint corresponding to its current UI language. If there is any exception during Step 5 to Step 13, “iAM Smart” Mobile App will open the fallback URL without the authCode and state instead of the direct login URL. In addition, the three fallback URLs of Online Service for three languages (EN/TC/SC) should have been registered so that the “iAM Smart” Mobile App can decide which URL is for fallback with respect to the current UI display language. Online Service should check if any login session exists and perform session management properly according to the business needs;

Online Service Callback API (GET: Callback with authCode to Online Service Server)

Step 16. In response to the direct login request from the external browser (Step 15), the Online Service server receives authCode, state and the corresponding persistent/session cookie set in Step 7-8. In addition, Online Service follows the method suggested by OWASP (i.e., re-generate the state using the same method adopted in Step 8) to verify the session and the state value. Online Service shall check if the generated state matches the one passed from the direct login request, and verify if timestamp received is less than the defined expiry time of state. After checking, the state can be safely invalidated;

Step 17-19. When the Online Service Server gets the authCode, it should invoke the “iAM Smart” API to obtain the accessToken and the tokenised ID (i.e., openID) of the “iAM Smart” user for the selected Online Service. API data encryption is required;

“iAM Smart” API (POST: Request accessToken & tokenised ID)

Step 20. Online Service server then performs user matching using the tokenised ID with its user repository and determines the result of this login request. Online Service must establish another session for maintaining the login status of the application. It is different from the session setup and used in Step 8 and Step 15 which is for Direct Login process only and will automatically expire after timeout. Online Service shall invalidate the session used for Direct Login process and ensure this session and its corresponding session data will never be reused for the subsequent business flow of the Online Service;

Step 21. The Online Service Website shows the page with login state.

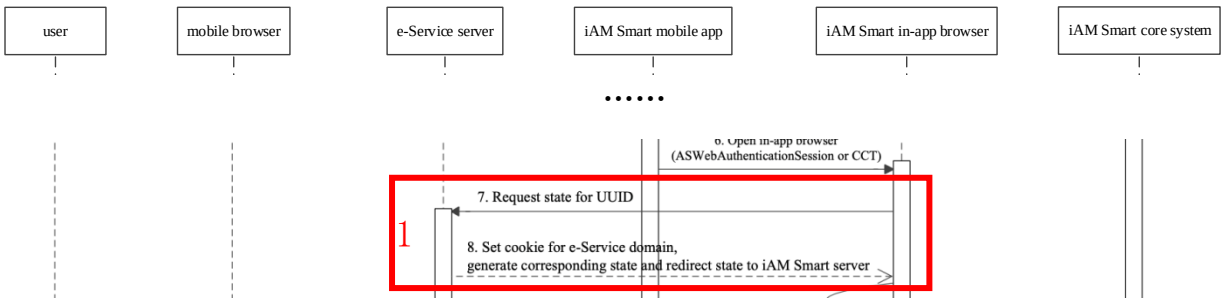
Notes:

Online Services shall provide the following URLs to DPO for setup in the "iAM Smart" Platform:

No.	URL Type	URL Description	Described in
7.	Generate State	Set cookie and generate state	Step 6
8.	Direct Login (English)	Receive authorisation code of web direct login with English language	Step 14-15
9.	Direct Login (Traditional Chinese)	Receive authorisation code of web direct login with Traditional Chinese language	Step 14-15
10.	Direct Login (Simplified Chinese)	Receive authorisation code of web direct login with Simplified Chinese language	Step 14-15
11.	Fallback / Direct Access (English)	Provide fallback for web direct login with English language when there is no supported browser, i.e., Online Service login page; or provide direct access to the Online Service in English without login requirement	Step 14-15
12.	Fallback / Direct Access (Traditional Chinese)	Provide fallback for web direct login with Traditional Chinese language when there is no supported browser, i.e., Online Service login page;	Step 14-15

		or provide direct access to the Online Service in Traditional Chinese without login requirement	
13.	Fallback / Direct Access (Simplified Chinese)	Provide fallback for web direct login with Simplified Chinese language when there is no supported browser, i.e., Online Service login page; or provide direct access to the Online Service in Simplified Chinese without login requirement	Step 14-15

A.3.1.1 Implementing (1) GET: An Online Service endpoint to set cookie and redirect state



This endpoint is to be implemented by Online Service.

Pre-conditions

- "iAM Smart" user has initiated Online Service website via direct login in "iAM Smart" Mobile App.
- "iAM Smart" Mobile App has verified that there is a supported browser installed in the same mobile phone.

Post-conditions

- The state redirected to "iAM Smart" server (in step 8) will be sent to Online Service as described in "Open Online Service website with authCode & state in external browser" (Implementation (2)). Online Service should verify in step 16 whether the state matches with the cookie sent from the external browser request.

Error conditions

- Online Service needs to implement this GET request. If errors occur, Online Service should just do the redirection without the state parameter.

Implementation Details of the Online Service Endpoint:

- **API Description**

Name	Description
Service Full Name	Online Service endpoint to set cookie and redirect state
URI (as in RESTFUL API)	`https://<Online Service_domain>/<Online Service_context>/<eservice_get_state_endpoint>?redirectURI=<redirect_URI>&eserviceName=<eservice_name>&clientID=<eservice_clientID>&scope=<scope>`
Request Type	GET
Service Version	1.0
Description of Service	Online Service should implement this endpoint to allow for direct login to Online Service website in the "iAM Smart" Mobile App. When "iAM Smart" calls this endpoint `<eservice_get_state_endpoint>`, which will be registered in the "iAM Smart" Platform, Online Service should set a cookie value in the `<Online Service_domain>`, generate a `state` that is uniquely related to the cookie value, and redirect the `state` to the URL specified in the `redirectURI` parameter (i.e. after setting cookie and state, Online Service should redirect the state to this location and the order of the parameters doesn't matter): `https://<redirect_URI>?eserviceName=<eservice_name>&clientID=<eservice_clientID>&scope=<scope>&state=<state>` When "iAM Smart" send the `authCode` to Online Service in a later step (step 15), it will also send back the `state` generated by this API. Online Service shall use the `state` to verify that the `authCode` is for the request it initiated previously.

- **Request Parameters**

Parameter	Type	Condition	Description
`redirectURI`	String	Required	A redirect URI (e.g., `https://<eid_domain>/saveState/<UUID>`). It will be URL encoded.
`eserviceName`	String	Required	The Online Service name (e.g., `test_eservice`). It will be URL encoded.

clientID	String	Required	The Online Service clientID (e.g., 0bf73be70f844fdb8a4da2f86cfeaddc). It will be URL encoded.
scope	String	Required	The previously agreed scope (e.g., eidapi_auth eidapi_sign). It will be URL encoded.

- **Example Request**

```
// For legibility only. Parameters should be URL encoded.
GET
https://esd-isit.staging-
eid.gov.hk/eservice/getState?redirectURI=https://core-isit.staging-
eid.gov.hk/api/v1/auth/saveState/7FCC9EC5CD7A4D5093F9ADFFCFDF0C4D&eserviceName
=test_eservice&clientID=0bf73be70f844fdb8a4da2f86cfeaddc&scope=eidapi_sign
eidapi_auth
```

- **Response Parameters**

Parameter	Type	Condition	Description
state	String	Required	A value that uniquely relates to the cookie value Online Service set in the <Online Service_domain>; It is an ASCII string less than or equal to 128 bytes (Note: this size of state only applies in the flow of direct login with Online Service website)
eserviceName	String	Required	The eserviceName as provided in the request parameter
clientID	String	Required	The clientID as provided in the request parameter
scope	String	Required	The scope as provided in the request parameter

- **Example Success Response**

```
// For legibility only. Parameters should be URL encoded.
302
Location=https://core-isit.staging-
eid.gov.hk/api/v1/auth/saveState/7FCC9EC5CD7A4D5093F9ADFFCFDF0C4D?state=c500fe
ce71ff4a419cf46f2c840c947d&eserviceName=test_eservice&clientID=0bf73be70f844fd
b8a4da2f86cfeaddc&scope=eidapi_sign eidapi_auth
```

- **HMAC Based Token Pattern Mitigation for CSRF Protection**

This defence is one of the most popular and recommended methods to mitigate CSRF with reference to the OWASP on "Cross-Site Request Forgery Prevention Cheat Sheet".
HMAC Based Token Pattern mitigation is achieved without maintaining any state at the server. HMAC based CSRF protection works similar to encryption based CSRF[6] protection with a couple of minor differences 1. It uses a strong HMAC function (SHA-256 or better is recommended) instead of an encryption function to generate the token. 2. The token consists of a HMAC and timestamp. Below are the steps for the proper implementation of the HMAC based CSRF protection: 1. Generate the token Using key K, Online Service server generates HMAC (cookie value + timestamp) and appends the same timestamp value to it which results in your CSRF token. Specifically, the cookie value refers to the value of persistent cookie if Online Service sets a unique persistent cookie for the in-app browser session as mentioned in the second bullet of the step 7-8 in Section 3.10.1. If Online Service adopts the optional approach for Android devices based on session cookie, the value refers to the session ID. 2. Include the token (*i.e.,* HMAC + timestamp) Include token as the state parameter as mentioned in "Response Parameters" of the "Implementation Details of the Online Service Endpoint" in this section. 3. Validating the token When the request is received at the Online Service server, the token is re-generated with same key K (parameters are the cookie value from the request and timestamp in the received token). If the HMAC in the received token and the one generated in this step match, verify if timestamp received is less than defined token expiry time. If both of them are successful, then request is treated as

[6] Encryption based CSRF protection encrypts token which comprises the user's session ID and a timestamp using a unique key available only on the server.

legitimate and can be allowed. If not, the Online Service server blocks the request and logs the attack for incident response purposes.

Security Hardening for Setting State

- Online Service is strongly recommended to conduct the following security hardening for setting the state:
  a) Set a short timeout for the state (e.g., 30 seconds).

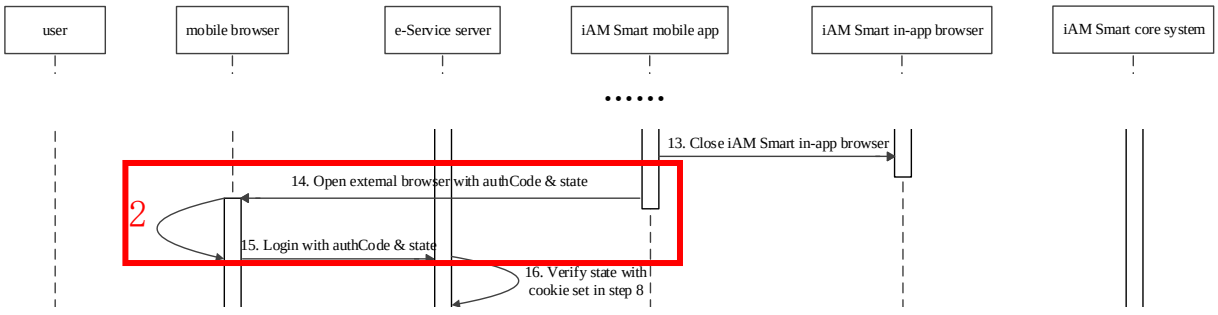
Security Hardening for Setting Cookie

- Online Service is strongly recommended to conduct the following security hardening for setting the cookie:
  a) Set limited values for the Attribute "Domain" and "Path";
  b) Set the Attribute "Secure" to true;
  c) Set the Attribute "HttpOnly" to true;
  d) Set the Attribute "SameSite" to Strict;
  e) Set Cookie Name Prefixes (i.e., the "_Secure-" Prefix and the "_Host-" Prefix);
  f) Set a short expiry time (e.g., 30 seconds); and
  g) Ensure the cookie cannot be re-used.

Security Protection of the Online Service Endpoint:

- Online Service should implement intrusion detection scheme to avoid any attack on the Online Service endpoint, e.g., reviewing any abnormal failure of direct login.

A.3.1.2 Implementing (2) GET: An Online Service endpoint to receive authCode & state from external browser



This endpoint is to be implemented by Online Service. The implementation reuses "Callback with authCode to Online Service Server" (Please refer to "iAM Smart" API Specification).

Pre-conditions

- "iAM Smart" user has initiated Online Service website via direct login in "iAM Smart" Mobile App.
- "iAM Smart" Mobile App has verified that there is a supported browser installed in the same mobile phone.

Post-conditions

- authCode will expire in 1 minute and Online Service can only use the authCode once for requesting the accessToken from "iAM Smart" System.

Error conditions

- This API is a HTTP GET request. If parameter "error_code" is returned, it means the direct login request is failed.

Error Code	Error Description	Suggested Action
error_code - D40002	Failed to authenticate	Inform Online Service the direct login request failed, and Online Service should fallback to direct access.
error_code – D40003	authentication request timeout	Inform Online Service that it took more than 30 seconds to "Set Cookie and Redirect State" (Implementation (1))

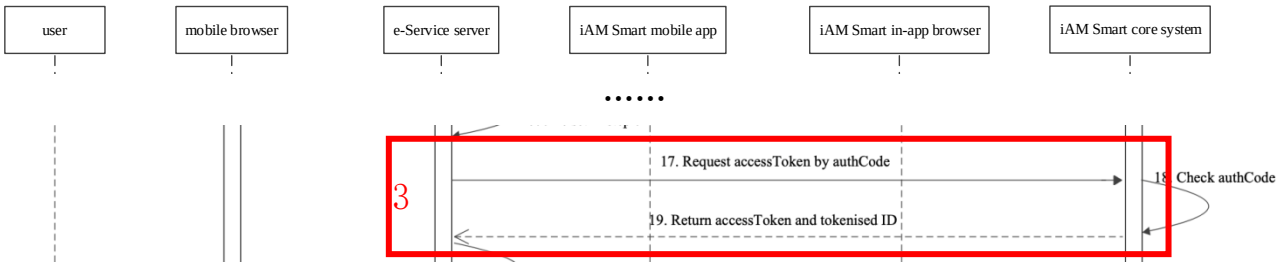
Callback Parameters

- Please refer to "iAM Smart" API Specification.

Notes

- Callback parameter "state": Online Service shall compare this state (i.e., carried in Step 15 of 3.10.1) with the state associated with the current cookie (i.e., based on the cookie and state pair saved in Step 8 of 3.10.1). If they do not match, Online Service is recommended to terminate the login process and prompt its user accordingly.
- Callback parameter "businessID": Not required in this scenario.
- Callback parameter "code": It is the authCode for requesting the accessToken from "iAM Smart" System. Its validity lasts for 1 minute and can only be used once.
- If errors occur, "error_code" instead of "code" will be returned.
- Online Service should check if any login session exists and perform session management properly according to business needs.

A.3.1.3 Implementing (3) POST: Request accessToken & tokenised ID



Please refer to Section 3.4.1.3.

B. REFERENCE APPLICATIONS

B.1 Overview

The Reference Application ("RA") is a set of sample Online Service programs to demonstrate how to make use of "iAM Smart" APIs and "iAM Smart" Mobile App to provide functions such as authentication, form filling and digital signing. The RA consists of a simulated Online Service application system ("Online Service server") which could be accessed through desktop/mobile browser (i.e., Online Service Website) or mobile application (i.e., Online Service App).

The RA is intended only for a reference of application development and it is not a mandatory way of implementation nor a standard package to be included in the Online Service application. Not all return codes of "iAM Smart" APIs are processed by the RA. Online Services should design and implement their own business logic and handle different "iAM Smart" API return codes according to their business requirements. Online Services developers must ensure that their applications should be implemented with sufficient security measures on handling the data, according to their departmental security policy as well as their application security requirements.

B.2 Software Directory Structure

The RA package contains a Java version and a .NET version of Online Service server / website, an Android Online Service App and an iOS Online Service App. These components can be found in the following directories of the RA package:

Directory	Description
server-java	Java source code for Online Service server
server-java\vue	Vue code for Online Service website
server-.net	.NET source code for Online Service server
mobile-android	Android source code for Online Service App
mobile-ios	iOS source code for Online Service App

B.3 Implementation Reference

This section provides a directory of functions to be demonstrated by the RA and the corresponding location of the program code in the RA package.

B.3.1 API Data Encryption and Decryption

B.3.1.1 Request CEK from "iAM Smart" System

Reference Component		Online Service server (Java)
1	Program of RA	*demo.eservice.utils.HttpUtil* - refreshAESKey(...)
	Demonstration	• Invoke "iAM Smart" API "Request Symmetric Content Encryption Key"
2	Program of RA	*demo.eservice.utils.EncryptRSAUtil* - privateDecodeToString(...)
	Demonstration	• Decrypt CEK returned from "iAM Smart" API "Request Symmetric Content Encryption Key" using private key of KEK

Reference Component		Online Service server (.NET)
1	Program of RA	*eID.Bussiness.EncryptService* - RequestBizAESKey(...)
	Demonstration	• Invoke "iAM Smart" API "Request Symmetric Content Encryption Key"
2	Program of RA	*eService.Common.EncryptUtils* - DecryptByPrivateKey(...)
	Demonstration	• Decrypt CEK returned from "iAM Smart" API "Request Symmetric Content Encryption Key" using private key of KEK

B.3.1.2 API Data Encryption and Decryption for POST Request and Response

Reference Component		Online Service server (Java)
1	Program of RA	*demo.eservice.utils.HttpUtil* - PostdataWithEncrypt(...) Invoke the following method to encrypt requested data in AES standard: *demo.eservice.utils.EncryptionForAES* - encryptGCMString(...)
	Demonstration	• Encrypt POST request body using CEK
2	Program of RA	*demo.eservice.utils.HttpUtil* - decryptCallBackData(...) Invoke the following method to decrypt "iAM Smart" returned data in AES standard:

		demo.eservice.utils.EncryptionForAES - decryptGCMSBase64 (...)
	Demonstration	• Decrypt POST response / Online Service callback API body using CEK

Reference Component		Online Service server (.NET)
1	Program of RA	*eID.Bussiness.EncryptService* - BizPostWithEncrypt(...) Invoke the following method to encrypt requested data in AES standard: *eService.Common.EncryptUtils* - AESEncrypt(...)
	Demonstration	• Encrypt POST request body using CEK
2	Program of RA	*eID.Bussiness.EncryptService* - DecryptCallbackContent(...) Invoke the following method to decrypt "iAM Smart" returned data in AES standard: *eService.Common.EncryptUtils* - AESDecrypt(...)
	Demonstration	• Decrypt POST response / Online Service callback API body using CEK

B.3.1.3 Prosess API Data for POST Request

Reference Component		Online Service server (Java)
1	Program of RA	*demo.eservice.utils.SignUtil* - genHMACSHA256Sign(signoriginalStr, clientSecret) Parameters: signoriginalStr: clientID+signatureMethod+timestamp+nonce+ encrypted requestbody clientSecret : SecretKey retrieved from "iAM Smart"
	Demonstration	• There are common parameters defined in the HTTP request header. Please refer to Section 3.2.2 for more details.

Reference Component		Online Service server (.NET)
1	Program of RA	*eService.Common.SignUtils* - GenHMACSHA1Sign (parameterStr, secretKey) Parameters:

		signoriginalStr: clientID+signatureMethod+timestamp+nonce+ encrypted requestbody clientSecret : SecretKey retrieved from “iAM Smart”
	Demonstration	• There are common parameters defined in the HTTP request header. Please refer to Section 3.2.2 for more details.

B.3.2 Web Application Development

B.3.2.1Online Service Website and “iAM Smart” Mobile App are in different devices

B. 3. 2. 1. 1 **Authentication**

Reference component		Online Service server (Java)
1	Program of RA	*demo.eservice.controller.LoginController* - getQR(...) *uri: /m/getQR*
	Demonstration	• Prepare parameters for “iAM Smart” API “Request QR Page” (assembles the Request QR Page URL)
2	Program of RA	*demo.eservice.controller.LoginController* - authCallBack(...) *uri: /authcallback*
	Demonstration	• Online Service callback API for receiving authorisation code from “iAM Smart” System (callback API used by Online Service to receive auth_code)
3	Program of RA	*demo.eservice.controller.LoginController* - webSiteLogin(...) Invoke the following method to request “iAM Smart” for accessToken: *demo.eservice.service.LoginService* - getAccessToken(...)
	Demonstration	• Online Service gets accessToken from “iAM Smart” System using authorisation code (Online Service uses auth_code to request “iAM Smart” System for accessToken)

Reference component		Online Service server (.NET)
1	Program of RA	*eService_Demo.RESTfulService.svc* - GetQR(...) *uri: /m/*getQR
	Demonstration	• Prepare parameters for “iAM Smart” API “Request QR Page” (assembles the obtained URL of QR Page)
2	Program of RA	*eService_Demo.RESTfulService.svc* - AuthCallBack(...) *uri: /authcallback*
	Demonstration	• Online Service callback API for receiving authorisation code

		from “iAM Smart” System (callback API used by Online Service to receive auth_code)
3	Program of RA	*eService_Demo.RESTfulService.svc* - WebsiteLogin(...) Invoke the following method to request “iAM Smart” for accessToken: *eID.Bussiness.Impl.LoginServiceImpl* - GetAccessToken(...)
	Demonstration	• Online Service gets accessToken from “iAM Smart” System using authorisation code (Online Service uses auth_code to request “iAM Smart” System for accessToken)

Reference component		Online Service Website
1	Program of RA	For mobile browser */src/pages/Eservice/Mobile/Login.vue* Method: • continueBtn() - Initiate login call request to Online Service server • browserType() - Provide browser information to Online Service server • getQR() - Redirect browser to invoke “iAM Smart” API “Request QR Page” to show QR Code For PC browser */src/pages/Eservice/PC/Login.vue* Method: • continueBtn() - Initiate login request to Online Service server • PC terminal sets to PC_Browser - Provide browser information to Online Service server • getQR() - Redirect browser to invoke “iAM Smart” API “Request QR Page” to show QR Code
	Demonstration	• Initiate login request to Online Service server • Provide browser information to Online Service server • Redirect browser to invoke “iAM Smart” API “Request QR Page” to show QR Code
2	N/A	• Browser shows QR Page of “iAM Smart” System and after completion of authentication process in “iAM Smart” System, browser will be redirect back to Online Service callback API for receiving authorisation code.

3	Program of RA	*/src/router/index.js* Method: router.beforeEach() For success case */src/pages/Eservice/Mobile/LoginSuc.vue* (For mobile browser) */src/pages/Eservice/PC/LoginSuc.vue* (For PC browser) Method: created() Cancelled, timeout or other conditions */src/pages/Eservice/Mobile/Login.vue* (For mobile browser) */src/pages/Eservice/PC/Login.vue* (For PC browser) Method: created()
	Demonstration	• Online Service Website shows the login result

B. 3. 2. 1. 2 **Get Profile**

Reference component		Online Service server (Java)
1	Program of RA	*demo.eservice.controller.ProfileController* - requestProfile(...) *uri: /v1/api/*requestProfile Invoke the following method to encrypt business data and request "iAM Smart" profile. *demo.eservice.utils.HttpUtil* - PostdataWithEncrypt(...)
	Demonstration	• Online Service Server invokes the "iAM Smart" API with accessToken, Tokenised ID and callback URL
2	Program of RA	*demo.eservice.controller.ProfileController* - getProfile(...) *uri: /v1/api/getProfile*
	Demonstration	• The Online Service server provides polling API for its Online Service Website to wait for callback result
3	Program of RA	*demo.eservice.controller.ProfileController* - profileCallBack(...) *uri: /v1/api/profileCallBack* Invoke the following method to decrypt the encrypted data returned from "iAM Smart": *demo.eservice.utils.HttpUtil* - decryptCallBackData(...)

	Demonstration	• Online Service callback API for receiving “iAM Smart” profile

Reference component		Online Service server (.NET)
1	Program of RA	*eService_Demo.RESTfulService.svc* - RequestProfile(...) *uri: /v1/api/requestProfile* Invoke the following method to encrypt business data and request “iAM Smart” profile. *eID.Bussiness.EncryptService* - BizPostWithEncrypt(...)
	Demonstration	• Online Service Server invokes the ”iAM Smart” API with accessToken, Tokenised ID and callback URL
2	Program of RA	*eService_Demo.RESTfulService.svc* - GetProfile(...) *uri: /v1/api/getProfile*
	Demonstration	• The Online Service server provides polling API for its Online Service Website to wait for callback result
3	Program of RA	*eService_Demo.RESTfulService.svc* - ProfileCallBack(...) *uri: /v1/api/profileCallBack* Invoke the following method to decrypt the encrypted data returned from “iAM Smart”: *eID.Bussiness.EncryptService* - DecryptCallbackContent(...)
	Demonstration	• Online Service callback API for receiving “iAM Smart” profile

Reference component		Online Service Website
1	Program of RA	For mobile browser */src/pages/Eservice/Mobile/Profile.vue* Method: profileBtn() - initiate Get “iAM Smart” Profile request requestProfile() - request Get “iAM Smart” Profile API, process response returned check() - check if device is Android or iOS device openApp() - open “iAM Smart” Mobile App *Note:* *“iAM Smart” API fields returned: "code": "D00000", authByQR: if value is "false" then it is same device scenario on the mobile terminal, it will open “iAM Smart” Mobile App* For PC browser

		/src/pages/Eservice/PC/Profile.vue Method: profileBtn() - invoke Get “iAM Smart” Profile request requestProfile() - request Get “iAM Smart” Profile API, process response returned
	Demonstration	• Online Service Website initiates Get “iAM Smart” Profile request to Online Service server • Mobile terminal opens “iAM Smart” Mobile App depending on the “iAM Smart” API response parameters returned
2	Program of RA	*/src/pages/Eservice/Mobile/Profile.vue* (For mobile browser) */src/pages/Eservice/PC/Profile.vue* (For PC browser) Method: profileResult() - poll Online Service API for Get “iAM Smart” Profile result getProfile() - Get “iAM Smart” Profile result from Online Service server and show the result. *Note:* *Online Service API field returned: "code": if value is "D8002" then delay 3 seconds before invoking profileResult() to get polling result*
	Demonstration	• The Online Service Website should keep synchronising with the Online Service server for the request result using polling • The Online Service Server processes and matches the result and shows the result in the Online Service Website

B. 3. 2. 1. 3 **Form Filling**

Reference component		Online Service server (Java)
1	Program of RA	*demo.eservice.controller.EmeInfoController* - requestEMEInfo(...) *uri: /v1/api/requestEMEInfo* Invoke the following method to encrypt business data and request to “iAM Smart” for form filling information: *demo.eservice.utils.HttpUtil* - PostdataWithEncrypt(...)
	Demonstration	• Online Service Server initiates a Form Filling request to invoke the “iAM Smart” API with accessToken, Tokenised ID, required “iAM Smart” “eMEFields” (Online Service server initiates form filling request to “iAM Smart” server)

2	Program of RA	*demo.eservice.controller.EmeInfoController* - getEMEInfo(...) *uri: /v1/api/getEMEInfo*
	Demonstration	• The Online Service server provides polling API for its Online Service Website to wait for callback result (Online Service server provides form filling result polling API)
3	Program of RA	*demo.eservice.controller.EmeInfoController* - eMEInfoCallBack(...) *uri: /v1/api/eMEInfoCallBack* Invoke the following method to decrypt the encrypted "profileFields" and "eMEFields" returned from "iAM Smart": *demo.eservice.utils.HttpUtil* - decryptCallBackData(...)
	Demonstration	• Callback to Receive form filling information (Online Service provides callback API to receive form filling form filling result)

Reference component		Online Service server (.NET)
1	Program of RA	*eService_Demo.RESTfulService.svc* - RequestEMEInfo(...) *uri: /v1/api/requestEMEInfo* Invoke the following method to encrypt business data and request to "iAM Smart" for form filling information: *eID.Bussiness.EncryptService* - BizPostWithEncrypt(...)
	Demonstration	• Online Service Server initiates a Form Filling request to invoke the "iAM Smart" API with accessToken, Tokenised ID, required form filling items (Online Service server initiates form filling form filling request to "iAM Smart" server)
2	Program of RA	*eService_Demo.RESTfulService.svc* - GetEMEInfo(...) *uri: /v1/api/getEMEInfo*
	Demonstration	• The Online Service server provides polling API for its Online Service Website to wait for callback result (Online Service server provides form filling result polling API)
3	Program of RA	*eService_Demo.RESTfulService.svc - EmeInfoCallBack* *uri: /v1/api/eMEInfoCallBack* Invoke the following method to decrypt the encrypted form filling profile and/or "iAM Smart" profilereturned from iAM Smart: *eID.Bussiness.EncryptService* - DecryptCallbackContent(...)
	Demonstration	• Callback to Receive form filling information (Online Service provides callback API to receive form filling form filling result)

Reference component		Online Service Website
1	Program of RA	For mobile browser */src/pages/Eservice/Mobile/PreFillForm.vue* Method: preFillForm() - initiate form filling request to Online Service server requestEMEInfo() - process form filling request API, process response returned check() - check if device is Android or iOS device openApp() - open “iAM Smart” Mobile App *Note:* *“iAM Smart” API fields returned: "code": "D00000", authByQR: if valiue is "false" then it is same device scenario on the mobile terminal, it will open “iAM Smart” Mobile App* For PC browser */src/pages/Eservice/PC/PreFillForm.vue* Method: preFillForm() - initiates form filling request requestEMEInfo() - process form filling request API, process response returned
	Demonstration	• Online Service Website initiates form filling request to Online Service server • Mobile terminal opens “iAM Smart” Mobile App depending on the API response parameters returned
2	Program of RA	*/src/pages/Eservice/Mobile/PreFillForm.vue* (For mobile browser) */src/pages/Eservice/PC/PreFillForm.vue* (For PC browser) Method: getUserInfo() - poll Online Service API for the form filling result getEmeInfo() - enquire form filling result from Online Service server and show result. *Note:* *Online Service API field returned: "code": if value is "D8001" then delay 3 seconds before invoking getUserInfo () to enquire result*
	Demonstration	• Online Service Website initiates polling request to Online Service for getting form filling result data

		• Online Service Website shows the complete form filling resulting page

B. 3. 2. 1. 4 **Signing**

Reference component		Online Service server (Java)
1	Program of RA	1) Invokes the following function to request Hash/PDF signature: *demo.eservice.controller.SignController* - hashSign(...) *Hash digital signing uri: /v1/api/signing/signature* *PDF digital signing uri: /v1/api/pdfsigning/signature* A. **Call "iAM Smart" to initiate Hash digital signing** - *EidHashSignServiceImpl.signature()*; Steps for specific function calls: 1. Parameter verification [tokenID, xmlData, source] - *StringUtils.isEmpty()*; 2. Generate businessID & csrfState - *UUID.randomUUID().toString()*; 3. Get openID: 3.1 Get AccessTokenDTO from redis according to tokenID - *redisTemplate.opsForValue().get(RedisKeyUtil.getAccessTokenKey(tokenID))*; 3.2 Obtain openID from AccessTokenDTO - *accessTokenDTO.getOpenID()*; 4. Get hkicHash: 4.1 Get Profile from redis according to openID - *redisTemplate.opsForValue().get(RedisKeyUtil.getProfileKey(openID))*; 4.2 Obtain hkic from Profile and calculate hkic Hash with SHA-256 - *CAUtil.getHKICHashByProfile(...)*; 5. Calculate Signature hash - *CAUtil.getHashSignSourceData(…)* 5.1 If there is attachment(s) uploaded:

		a) Upload the files and call the interface - *demo.eservice.controller.SignFileUploadController - upload(...)*; b) Add the attachment file name to the XML structure; c) Save the XML file; d) Perform SHA-256 calculation of hash on both the XML file and the uploaded files, merge the hashes, and then perform SHA-256 on the merged hash; 5.2 If no attachments are uploaded: a) Save the XML file; b) Perform SHA-256 calculation of hash on the XML file; 6. Calculate four-digit signature code based on hashCode and openID - *CodeUtil.getSignCode(...)*; 7. Call "iAM Smart" to initiate digital signing request - *HttpUtil.PostdataWithEncrypt(...)*; 8. If the request returns successfully from "iAM Smart", return the businessID, signCode, authByQR, ticketID and other parameters to the eService frontend. **B. Call "iAM Smart" to initiate PDF digital signing** - *EidPDFSignServiceImpl.signature()*; Steps for specific function calls: 1. Parameter verification [tokenID, xmlData, source] - *StringUtils.isEmpty()*; 2. Generate businessID, csrfState - *UUID.randomUUID().toString()*; 3. Get openID: 3.1 Get AccessTokenDTO from redis according to tokenID - *redisTemplate.opsForValue().get(RedisKeyUtil.getAccessTokenKey(tokenID))*; 3.2 Obtain openID from AccessTokenDTO – *accessTokenDTO.getOpenID()*; 4. Get hkicHash:

		4.1 Get Profile from redis according to openID - *redisTemplate.opsForValue().get(RedisKeyUtil.getProfileKey(openID))*; 4.2 Obtain hkic from Profile and calculate hkic Hash with SHA-256 - *CAUtil.getHKICHashByProfile(...)*; 5. Calculate PDF signature hash - *CAUtil.xmlToPdf(…)*; 5.1 Save XML file; 5.2 Generate PDF file(s) based on XML node(s); 5.3 Use the *iText* sdk to generate temporary PDF file(s) with a signed field - *MakeSignature.signExternalContainer(…)* (signature reserved size is 20000); 5.4 SHA-256 calculation of the generated temporary PDF file(s); 6. Calculate four-digit signature code based on docDigest and openID - *CodeUtil.getSignCode(…)*; 7. Call "iAM Smart" to initiate digital signing request - *HttpUtil.PostdataWithEncrypt(…)*; 8. If the request returned successfully from "iAM Smart", return the businessID, signCode, authByQR, ticketID and other parameters to the eService frontend.
	Demonstration	• Online Service server initiates a digital signing request to invoke the "iAM Smart" API with accessToken, Tokenised ID, Document Hash (Online Service server initiates a digital signing request to "iAM Smart" server).
2	Program of RA	Invokes the following function to receive the Hash digital signing result: *demo.eservice.controller.SignController* - receiveHashSign(...) *Hash digital signing uri: /v1/api/signing/receive* *PDF digital signing uri: /v1/api/pdfsigning/receive* **A. Hash digital signing receives the signature result ("iAM Smart" Callback interface)** - *EidHashSignServiceImpl.receiveSignatureResult()*;

		Steps for specific function calls: 1. Decrypt Callback content - *HttpUtil.decryptCallBackData(…)*; 2. Parameter verification [businessID] - *StringUtils.isEmpty()*; 3. Store the signature result in redis - *redisTemplate.opsForValue().set(redisKeyUtil.getSignResultKey(businessID), signResult.toJSONString())*; **B. PDF digital signing receiving signature results ("iAM Smart" callback interface)** - *EidPDFSignServiceImpl.receiveSignatureResult()*; Steps for specific function calls: 1. Decrypt Callback content - *HttpUtil.decryptCallBackData(…)*; 2. Parameter verification [businessID] - *StringUtils.isEmpty()*; 3. Store the signature result in redis - *redisTemplate.opsForValue().set(redisKeyUtil.getSignResultKey(businessID), signResult.toJSONString())*;
	Demonstration	• The Online Service server provides callback API for receiving hash digital signing result.
3	Program of RA	Invoke the following function to get the Hash digital signing result: *demo.eservice.controller.SignController* - getHashSignResult(...) *Hash digital signing uri: /v1/api/signing/getSignatureResult* *PDF digital signing uri: /v1/api/pdfsigning/getSignatureResult* **A. Hash digital signing to get the digital signing result** - *EidHashSignServiceImpl.getSignatureResult()*; Steps for specific function calls: 1. Parameter verification [businessID] - *StringUtils.isEmpty()*; 2. Get the signature result from redis - *redisTemplate.opsForValue().get(redisKeyUtil.getSignResultKey(businessID))*;

		3. Verify that the digital signing result is correct if the digital signing is successful - *CAUtil.verify(...)*; 4. Call the ack. interface - *HttpUtil.PostdataWithEncrypt(...)* 5. If the request returned successfully from "iAM Smart", return the Base64 XML file to frontend for downloading; **B. PDF digital signing to get the signature result** - *EidPDFSignServiceImpl.getSignatureResult()*; Steps for specific function calls: 1. Parameter verification [businessID] - *StringUtils.isEmpty()*; 2. Get the signature result from redis - *redisTemplate.opsForValue().get(redisKeyUtil.getSignResultKey(businessID))*; 3. PDF signature needs to be synthesized when digital signing is successful - *MakeSignature.signDeferred(...)*; 4. If LTV is enabled, PDF file for enabling LTV is to be generated based on the PDF synthesized with signature. 5. Call the ack. Interface - *HttpUtil.PostdataWithEncrypt(...)*; 6. If the request is returned successfully from "iAM Smart", return the Base64 PDF file to frontend for downloading; If LTV is enabled, the LTV PDF is returned, otherwise the PDF synthesized with signature will be returned.
	Demonstration	• Verify digital signing result and return digital signing result acknowledgement to "iAM Smart" System.

Reference component		Online Service server (.NET)
1	Program of RA	1) Invokes the following function to request Hash/PDF signature: *eService_Demo.RESTfulService.svc* - HashSign(...) *Hash digital signing uri: /v1/api/signing/signature* *eService_Demo.RESTfulService.svc* - PdfSign(...) *PDF digital signing uri: /v1/api/pdfsigning/signature*

		A. **Call "iAM Smart" to initiate Hash digital signing** - *eID.Bussiness.Impl.SignServiceImpl.RequestHashSign()*; Steps for specific function calls: 1. Parameter verification [tokenID, xmlData, source] - *string.IsNullOrEmpty()*; 2. Generate businessID & csrfState - *UUIDUtils.GetUUIDString()*; 3. Get openID: 3.1 Get AccessTokenDTO from redis according to tokenID - *RedisHelper.Get<string>(RedisKeyUtils.GetAccessTokenKey(tokenID))*; 3.2 Obtain openID from AccessTokenDTO - *AccessTokenDTO.OpenID*; 4. Get hkicHash: 4.1 Get Profile from redis according to openID - *RedisHelper.Get<string>(RedisKeyUtils.GetProfileKey(accessTokenDTO.OpenID))*; 4.2 Obtain hkic from Profile and calculate hkic Hash with SHA-256 - *CAUtils.GetHKICHashByProfile(…)*; 5. Calculate Signature hash - *CAUtils.GetHashSignSourceData(…)* 5.1 If there is attachment(s) uploaded: a) Upload the files and call the interface - *eService_Demo.RESTfulService.svc - Upload(...)*; b) Add the attachment file name to the XML structure; c) Save the XML file; d) Perform SHA-256 calculation of hash on both the XML file and the uploaded files, merge the hashes, and then perform SHA-256 on the merged hash; 5.2 If no attachments are uploaded: a) Save the XML file; b) Perform SHA-256 calculation of hash on the XML file;

		6. Calculate four-digit signature code based on hashCode and openID - *CodeUtils.GetSignCode(…)*; 7. Call "iAM Smart" to initiate digital signing request - *EncryptService.BizPostWithEncrypt(…)*; 8. If the request returns successfully from "iAM Smart", return the businessID, signCode, authByQR, ticketID and other parameters to the eService frontend. **B. Call "iAM Smart" to initiate PDF digital signing** - *eID.Bussiness.Impl.SignServiceImpl.RequestPdfSign()*; Steps for specific function calls: 1. Parameter verification [tokenID, xmlData, source] - *string.IsNullOrEmpty()*; 2. Generate businessID, csrfState - *UUIDUtils.GetUUIDString()*; 3. Get openID: 3.1 Get AccessTokenDTO from redis according to tokenID - *RedisHelper.Get<string>(RedisKeyUtils.GetAccessTokenKey(tokenID))*; 3.2 Obtain openID from AccessTokenDTO – *AccessTokenDTO.OpenID*; 4. Get hkicHash: 4.1 Get Profile from redis according to openID - *RedisHelper.Get<string>(RedisKeyUtils.GetProfileKey(accessTokenDTO.OpenID))*; 4.2 Obtain hkic from Profile and calculate hkic Hash with SHA-256 - *CAUtils.GetHKICHashByProfile(…)*; 5. Calculate PDF signature hash - *CAUtils.XmlToPdf(…)*; 5.1 Save XML file; 5.2 Generate PDF file(s) based on XML node(s); 5.3 Use the *iText* sdk to generate temporary PDF file(s) with a signed field - *MakeSignature.SignExternalContainer(…)* (signature reserved size is 20000);

		5.4 SHA-256 calculation of the generated temporary PDF file(s); 6. Calculate four-digit signature code based on docDigest and openID - *CodeUtils.GetSignCode(…)*; 7. Call "iAM Smart" to initiate digital signing request - *EncryptService.BizPostWithEncrypt(…)*; 8. If the request returned successfully from "iAM Smart", return the businessID, signCode, authByQR, ticketID and other parameters to the eService frontend.
	Demonstration	• Online Service server initiates a digital signing request to invoke the "iAM Smart" API with accessToken, Tokenised ID, Document Hash (Online Service server initiates a digital signing request to "iAM Smart" server).
2	Program of RA	Invokes the following function to receive the Hash digital signing result: *eService_Demo.RESTfulService.svc* - HashSignCallBack(...) *Hash digital signing uri: /v1/api/signing/receive* *eService_Demo.RESTfulService.svc* - PdfSignCallBack(...) *PDF digital signing uri: /v1/api/pdfsigning/receive* **A. Hash digital signing receives the signature result ("iAM Smart" Callback interface)** - Steps for specific function calls: 1. Decrypt Callback content - *EncryptService.DecryptCallBackContent(…)*; 2. Parameter verification [businessID] - *string.IsNullOrEmpty()*; 3. Store the signature result in redis - *RedisHelper.Add(RedisKeyUtils.GetSignResultKey(businessID), callBackData);* **B. PDF digital signing receiving signature results ("iAM Smart" callback interface)** – Steps for specific function calls:

		1. Decrypt Callback content - *EncryptService.DecryptCallBackContent(…)*; 2. Parameter verification [businessID] - *string.IsNullOrEmpty()*; 3. Store the signature result in redis - *RedisHelper.Add(RedisKeyUtils.GetSignResultKey(businessID), callBackData);*
	Demonstration	• The Online Service server provides callback API for receiving hash digital signing result.
3	Program of RA	Invoke the following function to get the Hash digital signing result: *eService_Demo.RESTfulService.svc* - GetHashSignResult(...) *Hash digital signing uri: /v1/api/signing/getSignatureResult* *eService_Demo.RESTfulService.svc* - GetPdfSignResult(...) *PDF digital signing uri: /v1/api/pdfsigning/getSignatureResult* **A. Hash digital signing to get the digital signing result** - *eID.Bussiness.Impl.SignServiceImpl.GetHashAckResult()*; Steps for specific function calls: 1. Parameter verification [businessID] - *string.IsNullOrEmpty()*; 2. Get the signature result from redis - *RedisHelper.Get<string>(RedisKeyUtils.GetSignResultKey(businessID))*; 3. Verify that the digital signing result is correct if the digital signing is successful - *CAUtils.VerifySignature(…)*; 4. Call the ack. interface - *EncryptService.BizPostWithEncrypt(…)* 5. If the request returned successfully from "iAM Smart", return the Base64 XML file to frontend for downloading; **B. PDF digital signing to get the signature result** - *eID.Bussiness.Impl.SignServiceImpl.GetPdfAckResult()*; Steps for specific function calls: 1. Parameter verification [businessID] - *string.IsNullOrEmpty()*;

		2. Get the signature result from redis - *RedisHelper.Get<string>(RedisKeyUtils.GetSignResultKey(businessID))*; 3. PDF signature needs to be synthesized when digital signing is successful - *MakeSignature.SignDeferred(…)*; 4. If LTV is enabled, PDF file for enabling LTV is to be generated based on the PDF synthesized with signature. 5. Call the ack. Interface - *EncryptService.BizPostWithEncrypt(…)*; 6. If the request is returned successfully from "iAM Smart", return the Base64 PDF file to frontend for downloading; If LTV is enabled, the LTV PDF is returned, otherwise the PDF synthesized with signature will be returned.
	Demonstration	• Verify digital signing result and return digital signing result acknowledgement to "iAM Smart" System.

Reference component		Online Service Website
1	Program of RA	For mobile browser */src/pages/Eservice/Mobile/Signature.vue* Method: eIDSign() - initiate digital signing request, assemble xml data signatureHash() - Hash Sign signature request API signaturePdf() - pdf Sign signature request API getSignatureRes() - process signature request API data returned, shows the 4-digit identification code check() - check if device is Android or iOS device openApp() - opens "iAM Smart" Mobile App *Note:* *"iAM Smart" API returned field: "code": "D00000", authByQR: if value is "false" then it is same device scenario on the mobile terminal, it will show the "open "iAM Smart" Mobile App" button, clicks to open "iAM Smart" Mobile App.*

		For PC browser */src/pages/Eservice/PC/Signature.vue* Hash Sign Method: uploadFile() - upload signature file progressFunction() - upload in progress uploadComplete() - upload complete cancleUploadFile() - cancel upload delUploadFile() - delete uploaded file uploadFailed() - upload failed eIDSign() - initiate digital signing request, assemble xml data signature() - Hash Sign signature request API getSignatureRes() - Signature result request API, process returned data and shows 4-digit identification code Pdf Sign Method: eIDSign() - initiate digital signing request, assemble xml data signaturePdf() - pdf Sign signature request API getSignatureRes() - Signature result request API, process returned data and shows 4-digit identification code
	Demonstration	• Hash digital signing requires upload file for signature • Online Service Website sends digital signing request to Online Service server and shows the 4-digit identificaiton code • Mobile terminal determine action to display or hide the "open "iAM Smart" Mobile App" depending on the "iAM Smart" API parameters returned
2	Program of RA	*/src/pages/Eservice/Mobile/Signature.vue* (For mobile browser) */src/pages/Eservice/PC/Signature.vue* (For PC browser) Method: getSignRes() - poll signing result from Online Service server API getSignResult() - Hash Sign digital signing result request API getSignResultRes() - process digital signing result API return data and shows result */src/pages/Eservice/Mobile/SignResult.vue* (For mobile browser) */src/pages/Eservice/PC/SignResult.vue* (For PC browser) Method: downLoadFile() - download digital signing result file *Note:*

		Online Service server API fields returned: "code": if value is "D79401" then delay 3 seconds before invoking getSignRes() to enquire result
	Demonstration	• Online Service Website sends polling request to Online Service server, gets and shows the digital signature result

B. 3. 2. 1. 5 **Re-authentication**

Reference component		Online Service server (Java)
1	Program of RA	*demo.eservice.controller.StepUpAuthController* - requestStepUpAuth(...) *uri: /v1/api/requestStepUpAuth* Invoke the following method to encrypt business data and send Re-authentication request to "iAM Smart" System: *demo.eservice.utils.HttpUtil* - PostdataWithEncrypt(...)
	Demonstration	• Online Service Server initiates Re-authentication request to invoke the "iAM Smart" API with accessToken, Tokenised ID (Online Service server initiates Re-authentication request to "iAM Smart" server)
2	Program of RA	*demo.eservice.controller.StepUpAuthController* - getStepUpAuth (...) *uri: /v1/api/getStepUpAuth*
	Demonstration	• The Online Service server should provide polling API for the Online Service Website (Online Service server provides Re-authentication result polling API)
3	Program of RA	*demo.eservice.controller.StepUpAuthController* - stepUpAuthCallBack (...) *uri: /v1/api/stepUpAuthCallBack* Invoke the following method to decrypt the Re-authentication encrypted data returned from "iAM Smart" server: *demo.eservice.utils.HttpUtil* - decryptCallBackData(...)
	Demonstration	• Callback to receive Re-authentication result (Online Service provides callback API to receive the Re-authentication result)

Reference component		Online Service server (.NET)
1	Program of RA	*eService_Demo.RESTfulService.svc* - RequestStepUpAuth(...) *uri: /v1/api/requestStepUpAuth*

		Invoke the following method to encrypt business data and send Re-authentication request to “iAM Smart” System: *eID.Bussiness.EncryptService* - BizPostWithEncrypt(...)
	Demonstration	• Online Service Server initiates Re-authentication request to invoke the “iAM Smart” API with accessToken, Tokenised ID (Online Service server initiates Re-authentication request to “iAM Smart” server)
2	Program of RA	*eService_Demo.RESTfulService.svc* - GetStepUpAuth(...) *uri: /v1/api/getStepUpAuth*
	Demonstration	• The Online Service server should provide polling API for the Online Service Website (Online Service server provides Re-authentication result polling API)
3	Program of RA	*eService_Demo.RESTfulService.svc* - StepUpAuthCallBack(...) *uri: /v1/api/stepUpAuthCallBack* Invoke the following method to decrypt the Re-authentication encrypted data returned from “iAM Smart” server: *eID.Bussiness.EncryptService* - DecryptCallbackContent(...)
	Demonstration	• Callback to receive Re-authentication result (Online Service provides callback API to receive the Re-authentication result)

Reference component		Online Service Website
1	Program of RA	For mobile browser */src/pages/Eservice/Mobile/Authentic.vue* Method: eidBtn() - initiate Re-authentication request requestStepUpAuth() - Re-authentication request API, processes data returned check() - check if device is Android or iOS device openApp() - open “iAM Smart” Mobile App *Note:* *“iAM Smart” API fields returned: "code": "D00000", authByQR: if valiue is "false" then it is same device scenario on the mobile terminal, it will open “iAM Smart” Mobile App* For PC browser */src/pages/Eservice/PC/Authentic.vue* Method: eidBtn() - initiate Re-authentication request

		requestStepUpAuth() - Re-authentication request API, processes return
	Demonstration	• Online Service Website initiates Re-authentication request to Online Service server • Mobile terminal opens “iAM Smart” Mobile App depending on the “iAM Smart” API parameters returned
2	Program of RA	*/src/pages/Eservice/Mobile/Authentic.vue* (For mobile browser) */src/pages/Eservice/PC/Authentic.vue* (For PC browser) Method: authResult() - poll Re-authentication result with Online Service API getStepUpAuth() - Re-authentication result request API, process data returned and shows result *Note:* *Online Service API field returned: "code": if value is "D8003" then delay 3 seconds before invoking getStepUpAuth() to enquire result*
	Demonstration	• Online Service Website initiates polling request to Online Service server, gets and shows Re-authentication result

B. 3. 2. 1. 6 **Anonymous Form Filling**

Reference component		Online Service server (Java)
1	Program of RA	*demo.eservice.controller.EmeInfoController* - requestAnonymousEMEInfo(...) *uri: /v1/api/requestAnonymousEMEInfo* Encrypt business data and request form filling information from “iAM Smart”: *demo.eservice.utils.HttpUtil* - PostdataWithEncrypt(...)
	Demonstration	Online Service server receives request from web page and initiates an anonymous form filling request to “iAM Smart” server to get ticketID with parameters businessID, source, redirectURI and eMEFields
2	Program of RA	*demo.eservice.utils.GetQRUtil* - getQRUrl(...)
	Demonstration	Online Service server assembles QR Page URL including ticketID, and return QR Page URL to web page to redirect
3	Program of RA	*demo.eservice.controller.LoginController* - authCallBack(...) *uri: /authcallback*

	Demonstration	Online Service server receives businessID and its relevant authCode transferred by web page (redirectURI)
4	Program of RA	1. Require to process anonymous digital signing request in case of authCallBack API returning businessID *demo.eservice.service.impl.LoginServiceImpl* - handleAnonymousEme(...) 2. Check parameters *demo.eservice.service.impl.LoginServiceImpl* - getAccessTokenByAuthCode(...) 3. Get accessToken and openID with authCode, and store the returned accessToken and openID in redis for subsequent use *demo.eservice.service.impl.LoginServiceImpl* - getAccessToken(...)
	Demonstration	Online Service server requests accessToken and openID from "iAM Smart" server with authCode
5	Program of RA	*demo.eservice.controller.EmeInfoController* - getAnonymousEMEInfo(...) *uri: /v1/api/getAnonymousEMEInfo* 1. Request Form Filling information from "iAM Smart" server with accessToken (single-use only) and openID *demo.eservice.service.impl.AnonymousEmeInfoServiceImpl* - anonymousResult(...) 2. Encrypt business data and request form filling information from "iAM Smart" *demo.eservice.utils.HttpUtil* - PostdataWithEncrypt(...)
	Demonstration	Online Service server requests form filling information from "iAM Smart" server with accessToken (single-use only) and openID

Reference component		Online Service server (.NET)
1	Program of RA	*eService_Demo.RESTfulService.svc* - RequestAnonymousEMEInfo(...) *uri: /v1/api/requestAnonymousEMEInfo* Encrypt business data and request form filling information from "iAM Smart": *eID.Bussiness.EncryptService* -BizPostWithEncrypt(...)
	Demonstration	Online Service server receives request from web page and initiates an anonymous form filling request to "iAM Smart" server to get

		ticketID with parameters businessID, source, redirectURI and eMEFields
2	Program of RA	*eService.Common.GetQRUtils* - GetQRUrl(...)
	Demonstration	Online Service server assembles QR Page URL including ticketID, and return QR Page URL to web page to redirect
3	Program of RA	*eService_Demo.RESTfulService.svc* - AuthCallBack(...) *uri: /authcallback*
	Demonstration	Online Service server receives businessID and its relevant authCode transferred by web page (redirectURI)
4	Program of RA	1. Require to process anonymous digital signing request in case of authCallBack API returning businessID *eService_Demo.RESTfulService.svc* - AuthCallBack(...) 2. Check parameters *eService_Demo.RESTfulService.svc* - AuthCallBack(...) 3. Get accessToken and openID with authCode, and store the returned accessToken and openID in redis for subsequent use *eID.Bussiness.Impl.LoginServiceImpl* - GetAccessToken(...)
	Demonstration	Online Service server requests accessToken and openID from "iAM Smart" server with authCode
5	Program of RA	*eService_Demo.RESTfulService.svc* - GetAnonymousEMEInfo(...) *uri: /v1/api/getAnonymousEMEInfo* 1. Request Form Filling information from "iAM Smart" server with accessToken (single-use only) and openID *eID.Bussiness.Impl.EMeInfoImpl* - GetAnonymousEMeDetail(...) 2. Encrypt business data and request form filling information from "iAM Smart" *eID.Bussiness.EncryptService* - BizPostWithEncrypt(...)
	Demonstration	Online Service server requests form filling information from "iAM Smart" server with accessToken (single-use only) and openID

Reference component		Online Service Website
1	Program of RA	For mobile browser */src/pages/Eservice/Mobile/AnonymousFilling.vue* Method: anonymousFillForm() - initiate anonymous form filling request requestAnonymousEMEInfo() - request anonymous form filling API and process return data

		For PC browser */src/pages/Eservice/PC/AnonymousFilling.vue* Method: anonymousFillForm() - initiate anonymous form filling request requestAnonymousEMEInfo() - request anonymous form filling API and process return data
	Demonstration	• Online Service web page initiates anonymous form filling request to Online Service server • Open "iAM Smart" QR Code page
2-6	Program of RA	*/src/pages/Eservice/Mobile/AnonymousFilling.vue* (For mobile browser) */src/pages/Eservice/PC/AnonymousFilling.vue* (For PC browser) Method: getAnonymousServiceInfo() - enquiry anonymous form filling result getAnonymousServiceInfo() - enquiry anonymous form filling result API, process return data, and show data resetForm() - clear input box *Note:* *API return field: "code": if value is "D8001", then delay 3 seconds before invoking getAnonymousServiceInfo() to enquiry result*
	Demonstration	• Online Service Website initiates polling request to Online Service server for getting and showing anonymous form filling result data

B. 3. 2. 1. 7 **Anonymous Digital Signing**

Reference component		Online Service server (Java)
1	Program of RA	Invoke the following function to request Hash/PDF digital signing: Anonymous Hash digital signing: *demo.eservice.controller.SignController* - anonymousHashSign(...) Anonymous PDF digital signing: *demo.eservice.controller.SignController* - anonymousPdfSign(...)

		Anonymous Hash digital signing uri: */v1/api/anonymous/signing/signature* *Anonymous PDF digital signing uri:* */v1/api/anonymous/pdfsigning/signature* **A. Call "iAM Smart" to initiate Anonymous Hash sign** - *EidAnonymousHashSignServiceImpl.signature()*; Steps for specific function calls: 1. Parameter check [hkicNumber, xmlData, source] - *StringUtils.isEmpty();* 2. Parameter validity check [hkicNumber] - *CAUtil.checkHKIC(…);* 3. Generate businessID & csrfState - *UUID.randomUUID().toString()*; 4. SHA-256 calculation of hkicHash according to hkicNumber - *CodeUtil.digestToByte(…)* 4. Calculate Anonymous Hash Signature Hash - *CAUtil.getHashSignSourceData(…)*; 4.1 If there is upload attachment(s): a) Upload file(s) and call the interface *demo.eservice.controller.SignFileUploadController - upload(...)* b) Add the attachment file name(s) to the XML structure; c) Save the XML file; d) Perform SHA-256 calculation of hash on XML file and the uploaded file(s), merge the hashes, and then perform SHA-256 on the merged hash; 4.2 If no attachment(s) is uploaded: a) Save the XML file; b) Perform SHA-256 calculation of hash on XML file ; 5. Calculate four-digit signature code based on hashCode, hkicHash & clientID - *CodeUtil.getAnonymousSignCode(…)*

		6. Call “iAM Smart” to initiate an anonymous Hash request - *HttpUtil.PostdataWithEncrypt(…)*; 7. If the request is successful, return the businessID, redirectURI, signCode, ticketID and other parameters to the eService frontend. **B. Call “iAM Smart” to initiate anonymous PDF digital signing** - *EidAnonymousPDFSignServiceImpl.signature();* Steps for specific function calls: 1. Parameter check [hkicNumber, xmlData, source] - *StringUtils.isEmpty();* 2. Parameter validity check [hkicNumber] - *CAUtil.checkHKIC(…);* 3. Generate businessID & csrfState - *UUID.randomUUID().toString()*; 4. SHA-256 calculation of hkicHash according to hkicNumber - *CodeUtil.digestToByte(…)* 5. Calculate anonymous PDF signature hash - *CAUtil.xmlToPdf(…)*; 4.1 Save the XML file; 4.2 Generate PDF file(s) based on XML node(s); 4.3 Use *iText* SDK to generate a temporary PDF with a signed field -*MakeSignature.signExternalContainer(…)*; 4.4 Calculate Hash with SHA-256 for temporary PDF file(s); 6. Calculate four-digit signature code based on docDigest, hkicHash, clientID - *CodeUtil.getAnonymousSignCode(…)*; 7. Call “iAM Smart” to initiate anonymous PDF request - *HttpUtil.PostdataWithEncrypt(…)*; 8. If the request returns successfully from “iAM Smart”, return the businessID, redirectURI, signCode, ticketID and other parameters to the eService frontend.

	Demonstration	Online Service server receives request from web page and initiates an anonymous digital signing request to "iAM Smart" server to get ticketID with parameters businessID, source, redirectURI and eMEFields
2	Program of RA	*demo.eservice.utils.GetQRUtil* - getQRUrl(...)
	Demonstration	Online Service server assembles QR Page URL including ticketID, and return QR Page URL to web page to redirect
3	Program of RA	*demo.eservice.controller.LoginController* - authCallBack(...) *uri: /authcallback*
	Demonstration	Online Service server receives businessID transferred by web page (redirectURI) and its relevant authCode
4	Program of RA	1. Require to process anonymous digital signing request in case of authCallBack API returning businessID Anonymous Hash digital signing: *demo.eservice.controller.SignController* - anonymousHashSign (...) Anonymous PDF digital signing: *demo.eservice.controller.SignController* - anonymousPdfSign (...) 2. Check parameter *demo.eservice.service.impl.LoginServiceImpl* - getAccessTokenByAuthCode(...) 3. Get accessToken and openID with authCode, and store the returned accessToken and openID in redis for future use *demo.eservice.service.impl.LoginServiceImpl* - getAccessToken(...) 4. If "iAM Smart" API response is successfully returned, use hkicHash + ClientID + hashCode/docDigest to invoke to generate 4-digit identification code: *CodeUtil. getAnonymousSignCode ()* - generate a 4-digit identification code
	Demonstration	Online Service server requests accessToken and openID from "iAM Smart" server with authCode
5	Program of RA	Invoke the following function to request getting Hash/PDF digital signing result:

		Anonymous Hash digital signing: *demo.eservice.controller.SignController* - getAnonymousHashSignResult (...) Anonymous PDF digital signing: *demo.eservice.controller.SignController* - getAnonymousPDFSignResult (...) *Anonymous Hash digital signing uri:* */v1/api/anonymous/signing/getSignatureResult* *Anonymous PDF digital signing uri:* */v1/api/anonymous/pdfsigning/getSignatureResult* **A. Get the signature result** - *EidAnonymousHashSignServiceImpl.getSignatureResult()*; The specific steps of the function call: 1. Parameter verification [businessID] - *StringUtils.isEmpty()*; 2. Get accessToken and openID: 2.1 Get AccessTokenDTO from redis according to tokenID - *redisTemplate.opsForValue().get(RedisKeyUtil.getAccessTokenKey(tokenID))*;; 2.2 Get accessToken from AccessTokenDTO - *accessTokenDTO.getAccessToken()*; 2.3 Get openID from AccessTokenDTO - *accessTokenDTO.getOpenID()*; 3. Call "iAM Smart" to get anonymous Hash signature result - *HttpUtil.PostdataWithEncrypt(...)*; 4. Verify that the digital signing result is correct if the digital signing is successful - *CAUtil.verify(...)*; 5. Call the ack. Interface - *HttpUtil.PostdataWithEncrypt(...)* 6. When calling "iAM Smart" successfully, return the Base64 XML files for frontend to download; **B. Get the signature result** - *EidAnonymousPDFSignServiceImpl.getSignatureResult()*;

		The specific steps of the function call: 1. Parameter verification [businessID] - *StringUtils.isEmpty()*; 2. Get accessToken and openID: 2.1 Get AccessTokenDTO from redis according to tokenID - *redisTemplate.opsForValue().get(RedisKeyUtil.getAccessTokenKey(tokenID))*;; 2.2 Get accessToken from AccessTokenDTO - *accessTokenDTO.getAccessToken()*; 2.3 Get openID from AccessTokenDTO - *accessTokenDTO.getOpenID()*; 3. Call "iAM Smart" to get anonymous PDF signature results - *HttpUtil.PostdataWithEncrypt(…)*; 4. PDF signature needs to be synthesized if digital signing is successful - *MakeSignature.signDeferred(…)*; 5. Call the ack. interface - *HttpUtil.PostdataWithEncrypt(…)*; 6. If calling "iAM Smart" successfully, return Base64 PDF files for frontend to download;
	Demonstration	Online Service server requests getting digital signing result from "iAM Smart"

Reference component		Online Service server (.NET)
1	Program of RA	Invoke the following function to request Hash/PDF digital signing: Anonymous Hash digital signing: *eService_Demo.RESTfulService.svc* - AnonymousHashSign(...) Anonymous PDF digital signing: *eService_Demo.RESTfulService.svc* - AnonymousPdfSign(...) *Anonymous Hash digital signing uri:* */v1/api/anonymous/signing/signature* *Anonymous PDF digital signing uri:* */v1/api/anonymous/pdfsigning/signature*

		A. Call "iAM Smart" to initiate Anonymous Hash sign - *eID.Bussiness.Impl.SignServiceImpl.RequestAnonymousHashSign()*; Steps for specific function calls: 1. Parameter check [hkicNumber, xmlData, source] - *string.IsNullOrEmpty()*; 2. Parameter validity check [hkicNumber] - *CAUtils.CheckHKIC(…);* 3. Generate businessID & csrfState - *UUIDUtils.GetUUIDString()*; 4. SHA-256 calculation of hkicHash according to hkicNumber - *CodeUtils.GetAnonymousSignCode(…)* 4. Calculate Anonymous Hash Signature Hash - *CAUtils.GetHashSignSourceData(…)*; 4.1 If there is upload attachment(s): a) Upload file(s) and call the interface *eService_Demo.RESTfulService.svc - Upload(...)*; b) Add the attachment file name(s) to the XML structure; c) Save the XML file; d) Perform SHA-256 calculation of hash on XML file and the uploaded file(s), merge the hashes, and then perform SHA-256 on the merged hash; 4.2 If no attachment(s) is uploaded: a) Save the XML file; b) Perform SHA-256 calculation of hash on XML file ; 5. Calculate four-digit signature code based on hashCode, hkicHash & clientID - *CodeUtils.GetAnonymousSignCode(…)* 6. Call "iAM Smart" to initiate an anonymous Hash request - *EncryptService.BizPostWithEncrypt(…)*; 7. If the request is successful, return the businessID, redirectURI, signCode, ticketID and other parameters to the eService frontend.

		B. Call "iAM Smart" to initiate anonymous PDF digital signing - *eID.Bussiness.Impl.SignServiceImpl.RequestAnonymousPdfSign()*; Steps for specific function calls: 1. Parameter check [hkicNumber, xmlData, source] - *string.IsNullOrEmpty()*; 2. Parameter validity check [hkicNumber] - *CAUtils.CheckHKIC(…);* 3. Generate businessID & csrfState - *UUIDUtils.GetUUIDString()*; 4. SHA-256 calculation of hkicHash according to hkicNumber - *CodeUtils.GetAnonymousSignCode(…)* 5. Calculate anonymous PDF signature hash - *CAUtils.XmlToPdf(…)*; 4.1 Save the XML file; 4.2 Generate PDF file(s) based on XML node(s); 4.3 Use *iText* SDK to generate a temporary PDF with a signed field -*MakeSignature.SignExternalContainer(…)*; 4.4 Calculate Hash with SHA-256 for temporary PDF file(s); 6. Calculate four-digit signature code based on docDigest, hkicHash, clientID - *CodeUtils.GetAnonymousSignCode(…)*; 7. Call "iAM Smart" to initiate anonymous PDF request - *EncryptService.BizPostWithEncrypt(…)*; 8. If the request returns successfully from "iAM Smart", return the businessID, redirectURI, signCode, ticketID and other parameters to the eService frontend.
	Demonstration	Online Service server receives request from web page and initiates an anonymous digital signing request to "iAM Smart" server to get ticketID with parameters businessID, source, redirectURI and eMEFields

2	Program of RA	*eService.Common.GetQRUtils* - GetQRUrl(...)
	Demonstration	Online Service server assembles QR Page URL including ticketID, and return QR Page URL to web page to redirect
3	Program of RA	*eService_Demo.RESTfulService.svc* - AuthCallBack(...) *uri: /authcallback*
	Demonstration	Online Service server receives businessID transferred by web page (redirectURI) and its relevant authCode
4	Program of RA	1. Require to process anonymous digital signing request in case of authCallBack API returning businessID Anonymous Hash digital signing: *eService_Demo.RESTfulService.svc* - AnonymousHashSign(...) Anonymous PDF digital signing: *eService_Demo.RESTfulService.svc* - AnonymousPdfSign(...) 2. Check parameter 3. Get accessToken and openID with authCode, and store the returned accessToken and openID in redis for future use *eID.Bussiness.Impl.LoginServiceImpl* - GetAccessToken(...) 4. If "iAM Smart" API response is successfully returned, use hkicHash + ClientID + hashCode/docDigest to invoke to generate 4-digit identification code: *CodeUtils.GetAnonymousSignCode ()* - generate a 4-digit identification code
	Demonstration	Online Service server requests accessToken and openID from "iAM Smart" server with authCode
5	Program of RA	Invoke the following function to request getting Hash/PDF digital signing result: Anonymous Hash digital signing: *eService_Demo.RESTfulService.svc* - GetAnonymousHashSignResult(...) Anonymous PDF digital signing: *eService_Demo.RESTfulService.svc* - GetAnonymousPDFSignResult(...) *Anonymous Hash digital signing uri:* */v1/api/anonymous/signing/getSignatureResult*

		Anonymous PDF digital signing uri: */v1/api/anonymous/pdfsigning/getSignatureResult* **A. Get the signature result** - *eID.Bussiness.Impl.GetAnonymousHashAckResult ()*; The specific steps of the function call: 1. Parameter verification [businessID] - *string.IsNullOrEmpty()*; 2. Get accessToken and openID: 2.1 Get AccessTokenDTO from redis according to tokenID - *RedisHelper.Get<string>(RedisKeyUtils.GetBussAccessTokenKey(businessID))*; 2.2 Get accessToken from AccessTokenDTO - *AccessTokenDTO.AccessToken ()*; 2.3 Get openID from AccessTokenDTO - *AccessTokenDTO.OpenID()*; 3. Call "iAM Smart" to get anonymous Hash signature result - *eID.Bussiness.EncryptService* - BizPostWithEncrypt(...); 4. Verify that the digital signing result is correct if the digital signing is successful - *CAUtils.VerifySignature(…)*; 5. Call the ack. Interface - *eID.Bussiness.EncryptService* - BizPostWithEncrypt(...); 6. When calling "iAM Smart" successfully, return the Base64 XML files for frontend to download; **B. Get the signature result** - *eID.Bussiness.Impl.GetAnonymousPdfAckResult ()*; The specific steps of the function call: 1. Parameter verification [businessID] - *string.IsNullOrEmpty()*; 2. Get accessToken and openID: 2.1 Get AccessTokenDTO from redis according to tokenID - *RedisHelper.Get<string>(RedisKeyUtils.GetBussAccessTokenKey(businessID))*;

		2.2 Get accessToken from AccessTokenDTO - *AccessTokenDTO.AccessToken()*; 2.3 Get openID from AccessTokenDTO - *AccessTokenDTO.OpenID()*; 3. Call "iAM Smart" to get anonymous PDF signature results - *eID.Bussiness.EncryptService* - BizPostWithEncrypt(...); 4. PDF signature needs to be synthesized if digital signing is successful - *MakeSignature.SignDeferred(…)*; 5. Call the ack. interface - *eID.Bussiness.EncryptService* - BizPostWithEncrypt(...); 6. If calling "iAM Smart" successfully, return Base64 PDF files for frontend to download;
	Demonstration	Online Service server requests getting digital signing result from "iAM Smart"

Reference component		Online Service Website
1	Program of RA	For mobile browser */src/pages/Eservice/Mobile/AnonymousSignature.vue* Method: eIDSign() - initiates anonymous digital signing request, assemble xml data signatureAmsHash() - anonymous Hash digital signing request API signatureAmsPdf() - anonymous PDF digital signing request API getSignatureResult() - request anonymous digital signing API, process return data, and show identification code showQRCode() - opens "iAM Smart" QR Code page For PC browser */src/pages/Eservice/PC/AnonymousSignature.vue* Hash Sign Method: uploadFile() - uploads digital signing document progressFunction() - upload in progress uploadComplete() - upload completed

		cancleUploadFile() - cancel upload delUploadFile() - delete uploaded document uploadFailed() - upload failed eIDSign() - initiate anonymous digital signing request, assemble xml data signatureAmsHash() - anonymous Hash digital signing request API getSignatureResult() - request anonymous digital signing API, process return data, and show identification code showQRCode() - open “iAM Smart” QR Code page PDF Sign Method: eIDSign() - initiate anonymous digital signing request, assemble xml data signatureAmsPdf() - PDF Sign anonymous digital signing request API getSignatureResult() - request anonymous digital signing API, process return data, and show identification code showQRCode() - open “iAM Smart” QR Code page
	Demonstration	• Signing document needs to be uploaded first for Hash Sign • Online Service Website initiates applying digital signing request and shows 4-digit identification code • Click "Open iAM Smart" to open “iAM Smart” QR Code page
2-6	Program of RA	*/src/pages/Eservice/Mobile/AnonymousSignature.vue* (For mobile browser) */src/pages/Eservice/PC/AnonymousSignature.vue* (For PC browser) Method: getSignRes() - enquiry anonymous digital signing result getSignResultAmsHash() - get anonymous Hash digital signing result getSignResultAmsPdf() - get anonymous PDF digital signing result getSignResultRes() - request anonymous digital signing result, process return data, and show result downLoadFile() - download digital signing result file *Note:* *API return field: "code": if value is "D79404", then delay 3 seconds before invoking getSignRes() to enquiry result*

	Demonstration	• Online Service Website initiates polling request to Online Service server for getting and showing anonymous digital signing result

B. 3. 2. 1. 8 **Bulk Digital Signing**

Reference component		Online Service server (Java)
1	Program of RA	Invokes the following function to request Bulk Digital Signing: *demo.eservice.bulksign.controller - bulkSigning(...)* /v1/api/bulksigning/signature *demo.eservice.bulksign.service.impl.EidBulkSigningServiceImpl#signature* Steps for specific function calls: 1. Parameter verification [tokenID, source] - *StringUtils.isEmpty();* 2. Generate businessID & csrfState - *UUID.randomUUID().toString();* 3. Get openID: 3.1 Get AccessTokenDTO from redis according to tokenID - *redisTemplate.opsForValue().get(RedisKeyUtil.getAccessTokenKey(tokenID));* 3.2 Obtain openID from AccessTokenDTO - *accessTokenDTO.getOpenID();* 4. Get hkicHash: 4.1 Get Profile from redis according to openID - *redisTemplate.opsForValue().get(RedisKeyUtil.getProfileKey(openID));* 4.2 Obtain hkic from Profile and calculate hkic Hash with SHA-256 - *CAUtil.getHKICHashByProfile(...);* 5. Initialize the request object:

		5.1 build requestTDO - *demo.eservice.bulksign.service.impl.EidBulkSigningServiceImpl#buildInitiateRequestDTO* 5.2 Create documents - *demo.eservice.bulksign.service.impl.AbstractBulkSigningService#createDocuments* a) if pdf document SHA-256 calculation of the generated temporary PDF file. - *demo.eservice.bulksign.service.impl.AbstractBulkSigningService#getPdfFile* b) if hash document Perform SHA-256 calculation of hash. - *demo.eservice.bulksign.service.impl.AbstractBulkSigningService#getHashFile* 6. Calculate six-digit signature code - *demo.eservice.bulksign.service.impl.AbstractBulkSigningService#calc6DigitsCode* 7. Call "iAM Smart" to initiate digital signing request - *demo.eservice.utils.HttpUtil#PostdataWithEncrypt* 8. If the request returns successfully from "iAM Smart", return the businessID, signCode, authByQR, ticketID and other parameters to the Online Service.
	Demonstration	• Online Service server initiates a digital signing request to invoke the "iAM Smart" API with accessToken, Tokenised ID, Documents (Online Service server initiates a bulk digital signing request to "iAM Smart" server).
2	Program of RA	Invokes the following function to receive the digital signing result: *demo.eservice.bulksign.controller.BulkSigningController#receiveResult* *receive result uri: /v1/api/bulksigning/receiveResult* receives the bulk digital signing result ("iAM Smart" Callback interface) -

		demo.eservice.bulksign.service.impl.AbstractBulkSigningService# receiveResult Steps for specific function calls: 1. Decrypt Callback content - *demo.eservice.utils.HttpUtil#decryptCallBackData;* 2. Parameter verification [businessID] - *StringUtils.isEmpty();* 3. Store the signature result in redis - *demo.eservice.bulksign.service.impl.AbstractBulkSigningService#saveSignatureData* 4. Call "iAM Smart" ack interface to notify sign result - *demo.eservice.bulksign.service.impl.AbstractBulkSigningService#ackResult*
	Demonstration	• The Online Service server provides callback API for receives the bulk digital signing result.
3	Program of RA	Invokes the following function to receive the BSQC Token: *demo.eservice.bulksign.controller.BulkSigningController#receiveBSQCToken* receive BSQC Token uri: */v1/api/bulksigning/receiveBSQCToken* receive BSQC Token ("iAM Smart" Callback interface) - *demo.eservice.bulksign.service.impl.EidBulkSigningServiceImpl#receiveBSQCToken* Steps for specific function calls: 1. Decrypt Callback content - *demo.eservice.utils.HttpUtil#decryptCallBackData;* 2. Parameter verification [businessID & BSQCToken] - *StringUtils.isEmpty();* 3. Store the BSQC Token in redis - *redisTemplate.opsForValue().set(RedisKeyUtil.getBSQCtokenKey(businessID), BSQCToken);*
	Demonstration	• Receive BSQC Token for the bulk digital signing status query and cancellation.
4	Program of RA	Invokes the following function to cancel:

		demo.eservice.bulksign.controller.BulkSigningController#cancelBulkSigning cancel uri: /v1/api/bulksigning/cancel demo.eservice.bulksign.service.impl.AbstractBulkSigningService#cancelBulksigning Steps for specific function calls: 1. Check and sign the task through businessID - demo.eservice.utils.HttpUtil#decryptCallBackData; 2. Parameter verification [BSQCToken, OpenID] 3. Build the request object for BSQCToken and OpenID 4. Call "iAM Smart" to cancel signing request - *demo.eservice.utils.HttpUtil#PostdataWithEncrypt*
	Demonstration	Call "iAM Smart" to cancel signing request

B. 3. 2. 1. 9 **Anonymous Bulk Digital Signing**

Reference component		Online Service server (Java)
1	Program of RA	Invoke the following function to request bulk digital signing: Anonymous bulk digital signing uri: */v1/api/anonymous/bulksigning/signature* Call "iAM Smart" to initiate anonymous bulk digital signing - *demo.eservice.bulksign.service.EidBulkSigningService#signature* Steps for specific function calls: 1. Parameter check [hkicNumber, source] - *StringUtils.isEmpty();* 2. Parameter validity check [hkicNumber] - *CAUtil.checkHKIC(...);*

		3. Generate businessID & csrfState - *UUID.randomUUID().toString();* 4. Initialize the request object: 4.1 Build requestTDO - *demo.eservice.bulksign.service.impl.EidAnonymousBulkSigningServiceImpl#buildInitiateRequestDTO* 4.2 Create documents - *demo.eservice.bulksign.service.impl.AbstractBulkSigningService#createDocuments* 5. Calculate six-digit signature code - *demo.eservice.bulksign.service.impl.AbstractBulkSigningService#calc6DigitsCode* 6. Call "iAM Smart" to initiate anonymous request if request type is multipart/form-data call *demo.eservice.utils.HttpUtil#PostdataWithEncryptMultipart* if request type is application/json call *demo.eservice.utils.HttpUtil#PostdataWithEncrypt* 7. If the request returns successfully from "iAM Smart", return the businessID, redirectURI, signCode, ticketID and other parameters to the Online Service.
	Demonstration	Online Service server receives request from web page and initiates an anonymous digital signing request to "iAM Smart" server to get ticketID with parameters businessID, source, redirectURI and eMEFields
2	Program of RA	*demo.eservice.utils.GetQRUtil* - getQRUrl(...)
	Demonstration	Online Service server assembles QR Page URL including ticketID, and return QR Page URL to web page to redirect
3	Program of RA	*demo.eservice.bulksign.controller.BulkSigningController#authCallBack* *uri: /bulksigning/authcallback*
	Demonstration	Online Service server receives businessID transferred by web page (redirectURI) and its relevant authCode
4	Program of RA	Invokes the following function to receive the digital signing result:

		demo.eservice.bulksign.controller.BulkSigningController#receiveResult *receive result uri: /v1/api/bulksigning/receiveResult* receives the bulk digital signing result ("iAM Smart" Callback interface) - *demo.eservice.bulksign.service.impl.AbstractBulkSigningService#receiveResult* Steps for specific function calls: 1. Decrypt Callback content - *demo.eservice.utils.HttpUtil#decryptCallBackData;* 2. Parameter verification [businessID] - *StringUtils.isEmpty();* 3. Store the signature result in redis - *demo.eservice.bulksign.service.impl.AbstractBulkSigningService#saveSignatureData* 4. Call "iAM Smart" ack interface to notify sign result - *demo.eservice.bulksign.service.impl.AbstractBulkSigningService#ackResult*
	Demonstration	• The Online Service server provides callback API for receives the bulk digital signing result.

B.3.2.3Online Service Website and "iAM Smart" Mobile App are in same device

B. 3. 2. 2. 1 **Authentication**

Please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.1.1

B. 3. 2. 2. 2 **Get Profile**

Please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.1.2

B. 3. 2. 2. 3 **Form Filling**

Please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.1.3

B. 3. 2. 2. 4 **Signing**

Please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.1.4

B. 3. 2. 2. 5 **Re-authentication**

Please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.1.5

B. 3. 2. 2. 6 **Anonymous Form Filling**

Please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.1.6

B. 3. 2. 2. 7 **Anonymous Digital Signing**

Please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.1.7

B. 3. 2. 2. 8 **Direct Login**

Reference component		Online Service server (Java)
1	Program of RA	*demo.eservice.controller.LoginController* -getState (...) *uri: /getState*
	Demonstration	• Online Service should implement this endpoint to allow user to access the Online Service website via direct login in the "iAM Smart" app. When "iAM Smart" calls this endpoint, Online Service should set a cookie value in the <Online Service_domain>, generate a random state that is uniquely related to the cookie value, save *the cookie and state pair* in DB or cache for future use, and redirect the state to the URL specified in the redirectURI parameter. When "iAM Smart" sends the authCode to Online Service in a later step, it will also send back the state generated by this API. Online Service then could use the state to verify that the authCode is for the request it initiated previously.
2	Program of RA	*demo.eservice.controller.LoginController* - authCallBack(...) *uri: /authcallback*
	Demonstration	• Online Service callback API for receiving authorisation code from "iAM Smart" System (callback API used by Online Service to receive auth_code) • Online Service must compare the state corresponding to the cookie in the <Online Service_domain> (Online Service can get the state value from *the previously saved cookie and state pair* based on the current cookie) with the one in this callback request parameter. If the comparison result shows that they are inconsistent, Online Service is recommended to terminate the login process and prompt its user accordingly.

3	Program of RA	*demo.eservice.controller.LoginController* - webSiteLogin(...) Invoke the following method to request "iAM Smart" for accessToken: *demo.eservice.service.LoginService* - getAccessToken(...)
	Demonstration	• Online Service gets accessToken from "iAM Smart" System using authorisation code (Online Service uses auth_code to request "iAM Smart" System for accessToken) • After getting the tokenised ID, if there is an existing user who has currently logged in the Online Service, Online Service needs to verify the tokenised ID with the one of the existing user. The login remains only when the verification matches. Otherwise, Online Service is recommended to prompt user for confirmation before forcing the logout of the current account (if logged in) first, and then use the tokenised ID to log back in.

Reference component		Online Service server (.NET)
1	Program of RA	*eService_Demo.RESTfulService.svc* - GetState (...) *uri: /getState*
	Demonstration	• Online Service should implement this endpoint to allow user to access the Online Service website via direct login in the "iAM Smart" app. When "iAM Smart" calls this endpoint, Online Service should set a cookie value in the <Online Service_domain>, generate a random state that is uniquely related to the cookie value, save *the cookie and state pair* in DB or cache for future use, and redirect the state to the URL specified in the redirectURI parameter. When "iAM Smart" sends the authCode to Online Service in a later step, it will also send back the state generated by this API. Online Service then could use the state to verify that the authCode is for the request it initiated previously.
2	Program of RA	*eService_Demo.RESTfulService.svc* - AuthCallBack(...) *uri: /authcallback*
	Demonstration	• Online Service callback API for receiving authorisation code from "iAM Smart" System (callback API used by Online Service to receive auth_code) • Online Service must compare the state corresponding to the cookie in the <Online Service_domain> (Online Service can get the state value from *the previously saved cookie and state pair*

		based on the current cookie) with the one in this callback request parameter. If the comparison result shows that they are inconsistent, Online Service is recommended to terminate the login process and prompt its user accordingly.
3	Program of RA	*eService_Demo.RESTfulService.svc* - *W*ebSiteLogin(...) Invoke the following method to request "iAM Smart" for accessToken: *eID.Bussiness.Impl.LoginServiceImpl* - GetAccessToken(...)
	Demonstration	• Online Service gets accessToken from "iAM Smart" System using authorisation code (Online Service uses auth_code to request "iAM Smart" System for accessToken) • After getting the tokenised ID, if there is an existing user who has currently logged in the Online Service, Online Service needs to verify the tokenised ID with the one of the existing user. The login remains only when the verification matches. Otherwise, Online Service is recommended to prompt user for confirmation before forcing the logout of the current account (if logged in) first, and then use the tokenised ID to log back in.

B. 3. 2. 2. 9 **Bulk Digital Signing**

Please refer to Section B.3.2.1.8

B. 3. 2. 2. 10 **Anonymous Bulk Digital Signing**

Please refer to Section B.3.2.1.9

B.3.2.3Requesting In-App Browser to Provide Information or Take Action

Reference component		Online Service backend esdemo (Java)
1	Program of RA	*eid-backend-esdemo-java/vue/src/pages/EService/Mobile/JSAPI.vue* - getAppSet(…)
	Demonstration	• Online Service calls in-app browser Javascript APIs to get "iAM Smart" lanague, font and mode settings and close in-app browser.
2	Program of RA	*eid-backend-esdemo-java/vue/src/pages/EService/Mobile/JSAPI.vue* - getAppSystemSet(...)

	Demonstration	• Online Service calls in-app browser Javascript API to pop up system permission request dialogue.

B.3.3 iOS Application Development

B.3.3.1 URL scheme Setup in Online Service App

Prerequisites: Online Service developers are required to register an Online Service access account from DPO, Online Service developers have to provide DPO the Online Service App URL scheme, “iAM Smart” will add the Online Service URL Scheme in the white list. Developers can follow the following 2 steps to add their URL Scheme to the Online Service App URL Types configuration.

Step 1: In Online Service APP, open *Targets->info->LSApplicationQueriesSchemes* and add “iAM Smart” Mobile App URL scheme: “iAM Smart” Mobile App’s only URL Scheme (e.g., hk.gov.iamsmart), means allow Online Service App to pull-up “iAM Smart” Mobile App.

Step 2: In Online Service APP, open *Targets->info->URL Types* and add the Online Service URL scheme that was provided to DPO during developer account registration, e.g., hk.gov.eservice001

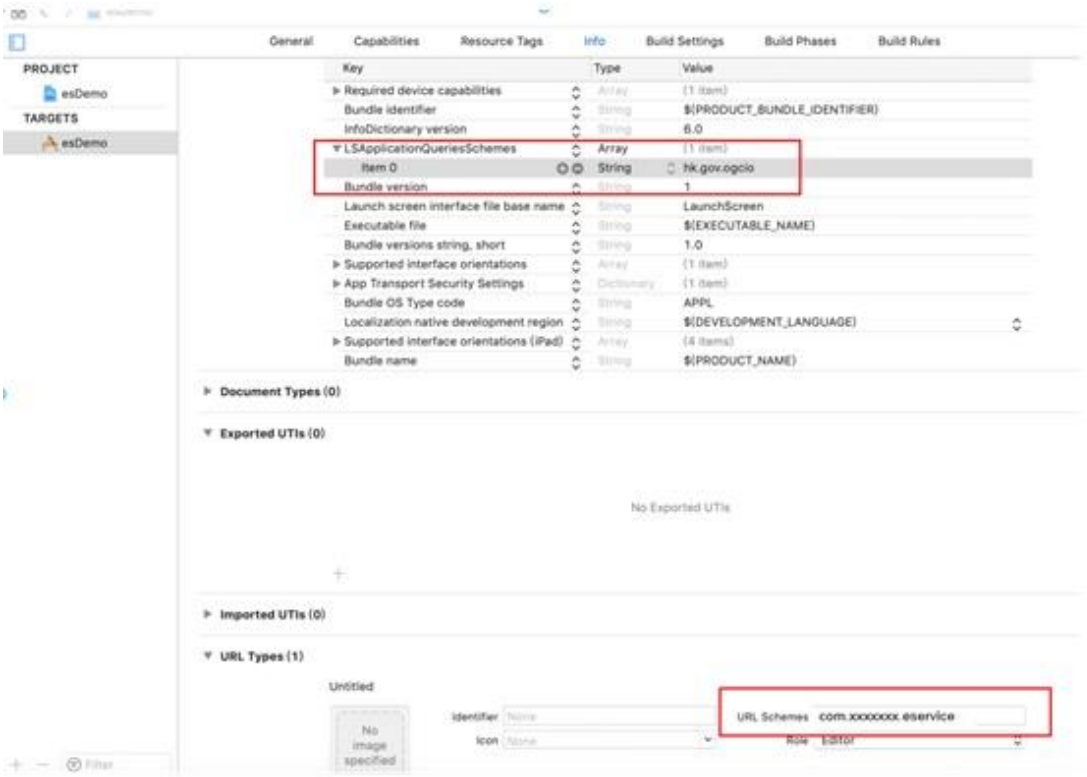
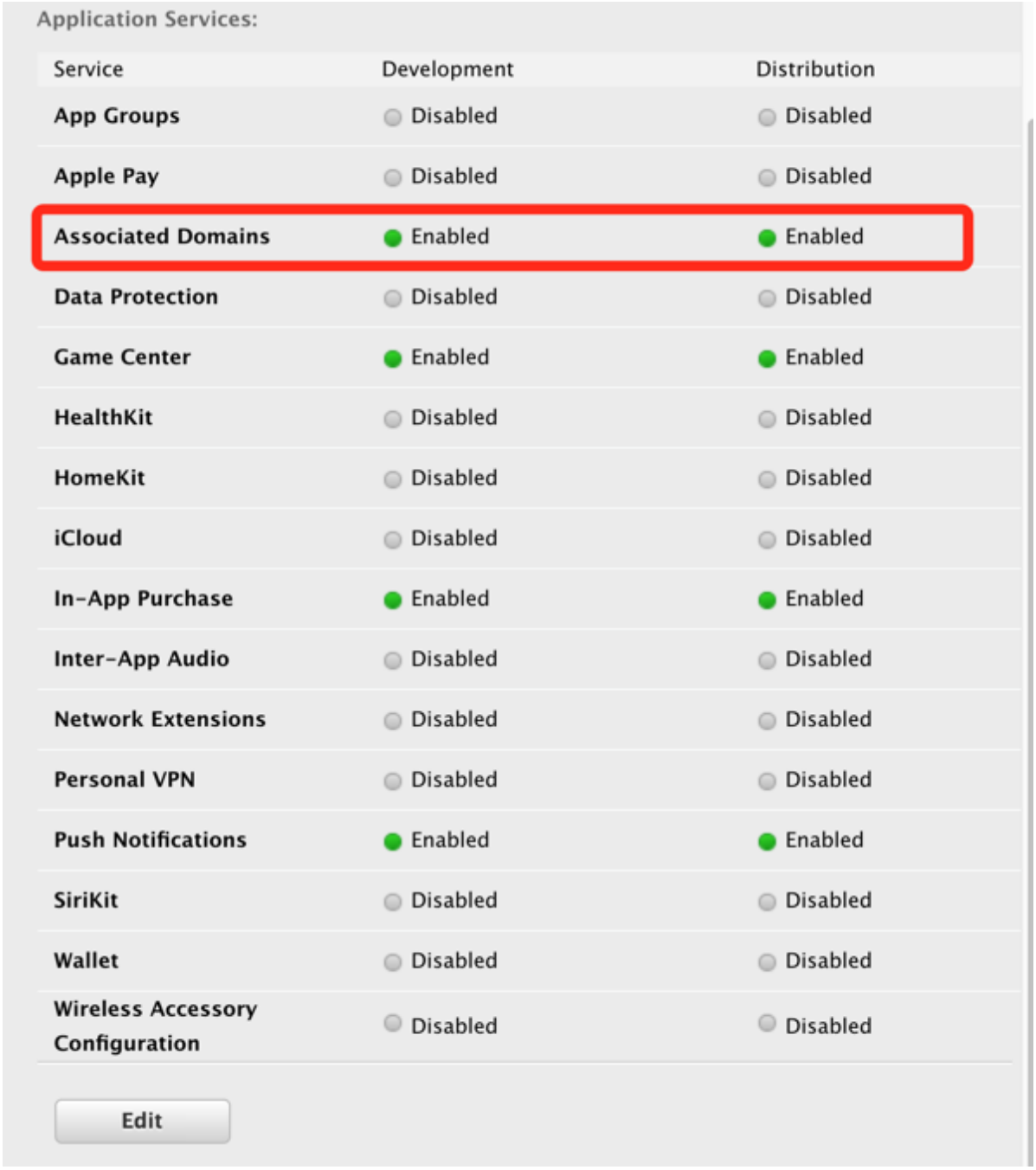


Figure-34 Configuring URL scheme in Xcode

B.3.3.2 Notes for Universal Links Setup in Online Service App

1. There must be a domain that supports HTTPS, and a permission of uploading to root directory under the domain (for uploading Apple specified file);

2. Apple developer center configuration: Find the corresponding App ID, locate the Associated Domains in the Application Services list, and turn it into Enabled;

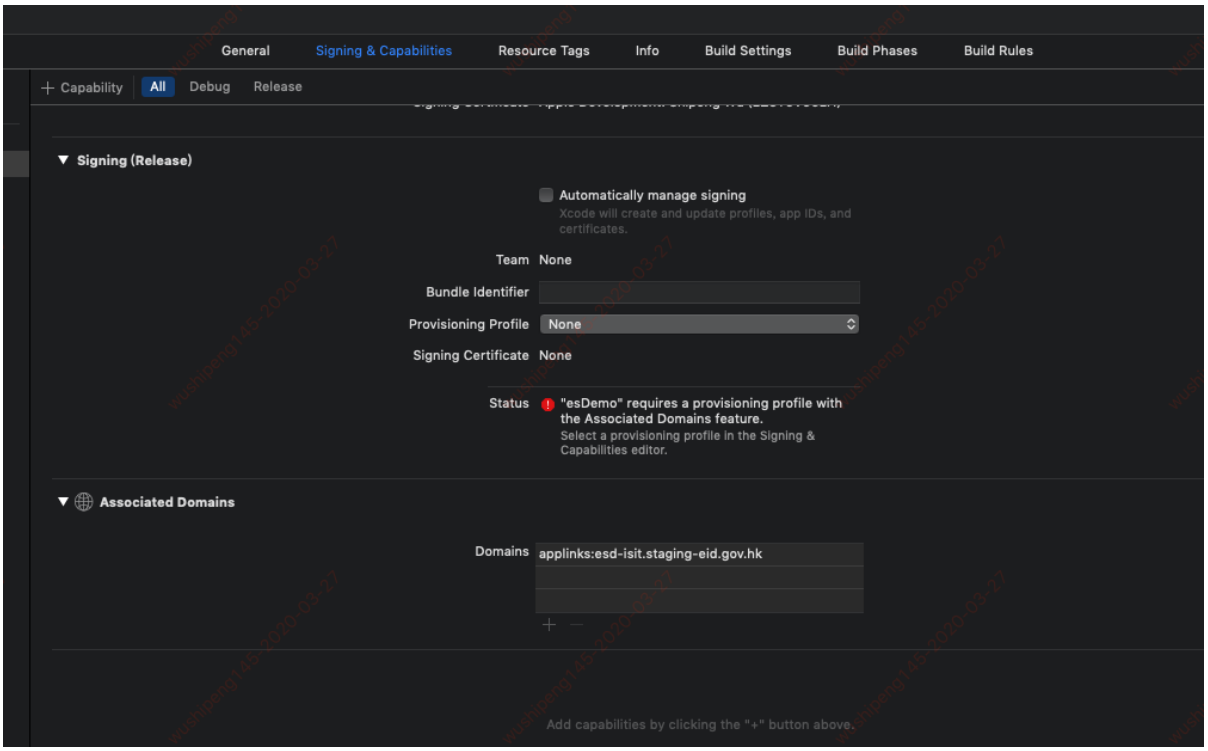


Configuring Associated Domains in Apple Developer Center

3. Project configuration:

Project configuration of relevant function for the version Xcode 11.0 and above:

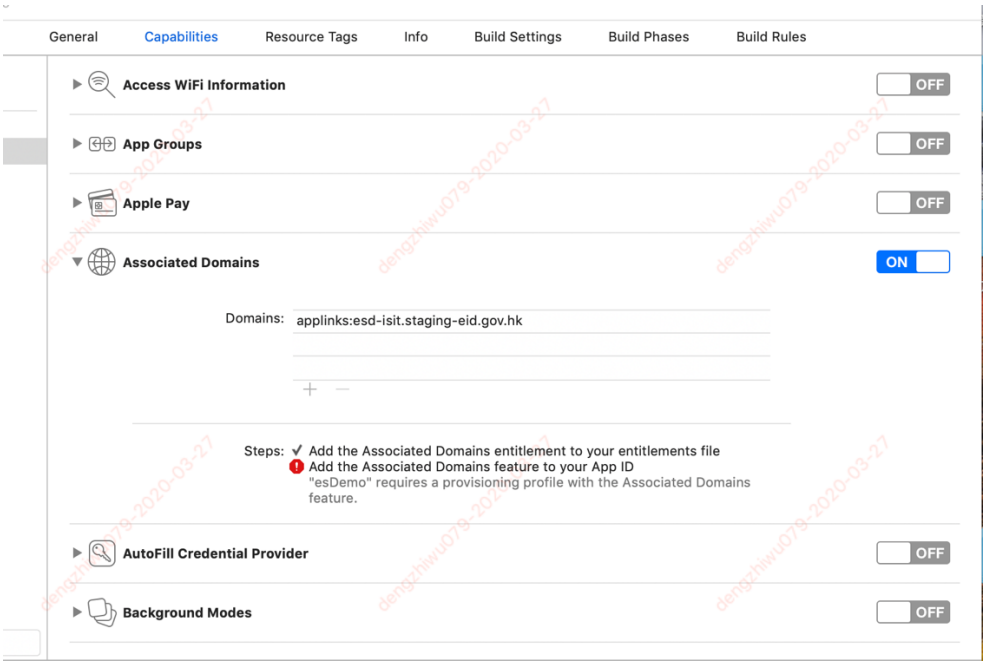
Fill in the expected domain name to be supported in the Domains under targets->Signing&Capabilites->Capability->Associated Domains, and the prefix must be applinks:



Configuring Associated Domains in Xcode 11.0 and above

Project configuration of relevant function for the version below Xcode 11.0:

Fill in the expected domain name to be supported in the Domains under targets->Signing&Capabilites->Capability->Associated Domains, and the prefix must be applinks: as well.



Configuring Associated Domains in Xcode below 11.0

4. Configure specified file: Create a file with its content in the format of json. Apple will request for this file based on the domain name filled in the project. The file name must be apple-app-site-association, and have no suffix (special concern). The file content is likely as follows:

```
{
   "applinks": {
        "apps": [],
        "details": [
            {
                "appID": "M3EM4YGUHH.com.example.demo",
                "paths":
["/directLogin","/directLogin/","/eservice/directLogin","/eservice/directLogin/"]
            }
        ]
    }
}
```

Explanations:

appID：The composition method is teamId.your app's bundle identifier. The teamId is like M3EM4YGUHH mentioned above, and can be found under Account -> Membership after the login to the Apple Developer Center.

paths：Set a list of paths supported by your App. Only links to these specified paths can be processed by the App.

5. Upload specified file: Upload the file to the root directory or .well-know directory corresponding to your domain name so that Apple can get the file you uploaded. After uploading, take a visit on the browser by yourself to see if you can get it.

https://domain/apple-app-site-association

6. Verify whether is effective: Enter a link that your App can recognize in a memo on your iOS device, and click on it to directly jump to your App, or long press the link and select the second item in the popped up menu to open it with 'XXX', which also represents a success;

Note:

Front-end development always faces cross-domain issues, and the cross-domain is mandatory required.

Universal Link will only take effect if the URL domain name of current webview does not match that of jump target.

B.3.3.3 Online Service App and "iAM Smart" Mobile App are in different devices

B.3.3.3.1 Authentication

Reference component		Online Service server (Java)
1	Program of RA	demo.eservice.controller.*LoginController* - getQR(...) *uri: /m/getQR*
	Demonstration	• Prepare parameters for "iAM Smart" API "Request QR Page" (assembles the URL of get QR Page)
2	Program of RA	demo.eservice.controller.*LoginController* - mobileLogin(...) uri: /m/login
	Demonstration	• Online Service callback API for receiving authorisation code from Online Service App (Online Service callback API to receive auth_code, it is different from 4.3.2.1.1)
3	Program of RA	*demo.eservice.service.LoginService* - getAccessToken(...)
	Demonstration	• Online Service gets accessToken from "iAM Smart" System using authorisation code (Online Service uses auth_code to request accessToken from "iAM Smart" server)

Reference component		Online Service server (.NET)
1	Program of RA	eService_Demo.*RESTfulService.svc* - GetQR(...) *uri: /m/*getQR
	Demonstration	• Prepare parameters for “iAM Smart” API “Request QR Page” (assembles the URL of get QR Page)
2	Program of RA	eService_Demo.*RESTfulService.svc* - MobileLogin(...) uri: /m/login
	Demonstration	• Online Service callback API for receiving authorisation code from Online Service App (Online Service callback API to receive auth_code, it is different from 4.3.2.1.1)
3	Program of RA	eService_Demo.*RESTfulService.svc* - GetAccessToken(...)
	Demonstration	• Online Service gets accessToken from “iAM Smart” System using authorisation code (Online Service uses auth_code to request accessToken from “iAM Smart” server)

Reference component		Online Service App
1	Program of RA	*ELoginViewController.m* - loginAction() canOpenURL: determine if “iAM Smart” Mobile App is installed in the device loadModelData: request redirect URL Schema
	Demonstration	• Initiate login request to Online Service server • Determined that “iAM Smart” Mobile App is not installed in the device • Provide URL scheme of Online Service App to Online Service server
2	Program of RA	*ELoginViewController.m* - loadModelData() Request Browser PageUrlWithParams: request web login page address ESafariViewController: in-app-browser to pull-up web browser.
	Demonstration	• Online Service App invokes in-app browser to show QR Page request (browser opens get QR Page URL)
3	Program of RA	*ELoginHelp.m*-handlerLoginWithDic() paramsDic: parameter received from “iAM Smart” QR Page redirection requestOpenIDByWithParams: Request tokenID from server handlerLoginWithDic: withBlock shows login result goToProfileOrFinish within block: Redirect page for successful login

	Demonstration	• Online Service App receives the authorisation code that was transferred from “iAM Smart” QR Page • Invoke Online Service server API to request accessToken and Tokenised ID • shows login result page after receiving result from Online Service server

B. 3. 3. 3. 2 **Get Profile**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.1.2.

Reference component		Online Service App
1	Program of RA	*EApplyProfileViewController.m* - onGoToProvideBtn() applyProfileSuccessBlock: initiate get Profile request to server
	Demonstration	• Online Service App initiates get Profile request to Online Service server
2	Program of RA	*EApplyProfileViewController.m* - checkGetResult() getResult: poll Online Service server
	Demonstration	• The Online Service App should keep synchronising with the Online Service server for the request result using polling
3	Program of RA	*EApplyProfileViewController.m* - getResult() getProfileSuccessBlock: process return result from Online Service server
	Demonstration	• The Online Service Server processes and matches the result and shows the result in the Online Service App

B. 3. 3. 3. 3 **Form Filling**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.1.3

Reference component		Online Service App
1	Program of RA	*FormFillViewController.m* - onFillBtn() requestFormFilling: initiate form filling
	Demonstration	• Online Service App initiates form filling request to Online Service server
2	Program of RA	*FormFillViewController.m* - onFillBtn()

		getRequestResult: poll Online Service server canOpenURL: determine if "iAM Smart" Mobile App is installed in the device
	Demonstration	• The Online Service App should keep synchronising with the Online Service server for the request result using polling • The Online Service App detected "iAM Smart" Mobile App is installed in the device, use URL Scheme to pull-up "iAM Smart" Mobile App
3	Program of RA	*FormFillViewController.m* - getRequestResult() requestFillingResult: Process server returned result showMyToast: show result
	Demonstration	• The Online Service Server processes and matches the result and shows the result in the Online Service App

B. 3. 3. 3. 4 **Signing**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.1.4.

Reference component		Online Service App
1	Program of RA	*SignPageViewController.m* - onSignBtn() requestSign: initiate digital signing request
	Demonstration	• Online Service App initiates digital signing request to Online Service server
2	Program of RA	*SignPageViewController.m* - getSignResult() requestSignResult: poll Online Service server *ShowCodeViewController.m* - openEIDBtnClick() canOpenURL: determine if "iAM Smart" Mobile App is installed in the device
	Demonstration	• The Online Service App should keep synchronising with the Online Service server for the request result using polling • The Online Service App detected "iAM Smart" Mobile App is installed in the device, use URL Scheme to pull-up "iAM Smart" Mobile App
3	Program of RA	*SignPageViewController.m* - requestSignResult() showMyToast: show result showSignSuccessVCWithData: redirect to signature successful page

	Demonstration	• The Online Service Server processes and matches the result and shows the result in the Online Service App

B. 3. 3. 3. 5 **Re-authentication**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.1.5.

Reference component		Online Service App
1	Program of RA	*ReAuthViewController.m* - sendEidAuthRequest() sendAuthRequestWithParams: initiates request to Online Service server
	Demonstration	• Online Service App initiates Re-authentication request to Online Service server
2	Program of RA	*ReAuthViewController.m* - sendEidAuthRequest() getAuthResultRequest: Online Service server polling canOpenURL: determine if "iAM Smart" Mobile App is installed in the device
	Demonstration	• The Online Service App should keep synchronising with the Online Service server for the request result using polling • The Online Service App detected "iAM Smart" Mobile App is installed in the device, use URL Scheme to pull-up "iAM Smart" Mobile App
3	Program of RA	*ReAuthViewController.m* - getAuthResultRequestWithParams() updateUI: updates UI with server returned result
	Demonstration	• The Online Service Server processes and matches the result and shows the result in the Online Service App

B. 3. 3. 3. 6 **Anonymous Form Filling**

Reference component		Online Service server (Java)
1	Program of RA	*demo.eservice.controller.EmeInfoController* - requestAnonymousEMEInfo(...) *uri: /v1/api/requestAnonymousEMEInfo* Encrypt business data and request form filling information from "iAM Smart": *demo.eservice.utils.HttpUtil* - PostdataWithEncrypt(...)

	Demonstration	Online Service server receives request from web page and initiates an anonymous form filling request to "iAM Smart" server to get ticketID with parameters businessID, source, redirectURI and eMEFields
2	Program of RA	*demo.eservice.controller.EmeInfoController* - getAnonymousEMEInfo(...) *uri: /v1/api/getAnonymousEMEInfo*
	Demonstration	Online Service App initiates polling request for anonymous form filling processing status
3	N/A	
4	Program of RA	*demo.eservice.controller.EmeInfoController* - applyAccessToken(...) *uri: /v1/api/applyAccessToken* 1. Check parameters *demo.eservice.service.impl.LoginServiceImpl* - getAccessTokenByAuthCode(...) 2. Get accessToken and openID with authCode, and store the returned accessToken and openID in redis for subsequent use *demo.eservice.service.impl.LoginServiceImpl* - getAccessToken(...)
	Demonstration	Online Service App transfers authorisation code to Online Service server, and then Online Service server requests accessToken and openID from "iAM Smart" server with received authorisation code
5	Program of RA	1. Request Form Filling information from "iAM Smart" server with accessToken (single-use only) and openID *demo.eservice.service.impl.AnonymousEmeInfoServiceImpl* - anonymousResult(...) 2. Encrypt business data and request form filling information from "iAM Smart" *demo.eservice.utils.HttpUtil* - PostdataWithEncrypt(...)
	Demonstration	Online Service server requests form filling information from "iAM Smart" server with accessToken (single-use only) and openID

Reference component		Online Service server (.NET)
1	Program of RA	*eService_Demo.RESTfulService.svc* - RequestAnonymousEMEInfo(...)

		uri: /v1/api/requestAnonymousEMEInfo Encrypt business data and request form filling information from "iAM Smart": *eID.Bussiness.EncryptService* - BizPostWithEncrypt(...)
	Demonstration	Online Service server receives request from web page and initiates an anonymous form filling request to "iAM Smart" server to get ticketID with parameters businessID, source, redirectURI and eMEFields
2	Program of RA	*eService_Demo.RESTfulService.svc* - GetAnonymousEMEInfo(...) *uri: /v1/api/getAnonymousEMEInfo*
	Demonstration	Online Service App initiates polling request for anonymous form filling processing status
3	N/A	
4	Program of RA	*eService_Demo.RESTfulService.svc* - ApplyAccessToken(...) *uri: /v1/api/applyAccessToken* 1. Check parameters 2. Get accessToken and openID with authCode, and store the returned accessToken and openID in redis for subsequent use *eID.Bussiness.Impl.LoginServiceImpl* - GetAccessToken(...)
	Demonstration	Online Service App transfers authorisation code to Online Service server, and then Online Service server requests accessToken and openID from "iAM Smart" server with received authorisation code
5	Program of RA	1. Request Form Filling information from "iAM Smart" server with accessToken (single-use only) and openID *eID.Bussiness.Impl.EMeInfoImpl* - GetAnonymousEMeDetail(...) 2. Encrypt business data and request form filling information from "iAM Smart" *eID.Bussiness.EncryptService* - BizPostWithEncrypt(...)
	Demonstration	Online Service server requests form filling information from "iAM Smart" server with accessToken (single-use only) and openID

Reference component		Online Service App
1	Program of RA	*FormFillViewController.m* - onFillBtn() requestFormFilling: initiate anonymous form filling request

	Demonstration	Online Service App initiates anonymous form filling request to Online Service server and provides App URL Scheme information
2	Program of RA	*FormFillViewController.m* - onFillBtn() requestQRCodeUrlSuccessBlock: get URL assembled by server getRequestResult: poll server enquiry result
	Demonstration	Online Service App receives QR Page URL assembled by Online Service server and opens it with the in-app browser In the meantime, Online Service App starts polling result of server enquiry request
3	Program of RA	AppDelegate.m - application:openURL:options paramsDic: transferred parameter
	Demonstration	In-app browser transfers authorisation code to Online Service App
4	Program of RA	FormFillModel.m - onNotification() applyAccess: transfer authorisation code to Online Service server
	Demonstration	Online Service App transfers authorisation code to Online Service server
5	Program of RA	*FormFillViewController.m* - getRequestResult () requestFillingResult: poll server enquiry result reloadData: refresh displayed result
	Demonstration	Online Service App displays form filling result

B. 3. 3. 3. 7 **Anonymous Digital Signing**

Reference component		Online Service server (Java)
1	Program of RA	Invoke the following function to request Hash/PDF digital signing: Anonymous Hash digital signing: *demo.eservice.controller.SignController* - anonymousHashSign (...) Anonymous PDF digital signing: *demo.eservice.controller.SignController* - anonymousPdfSign (...) *Anonymous Hash digital signing uri:* */v1/api/anonymous/signing/signature* *Anonymous PDF digital signing uri:* */v1/api/anonymous/pdfsigning/signature*

		Encrypt business data and request digital signing information from "iAM Smart": *demo.eservice.utils.HttpUtil* - PostdataWithEncrypt(...)
	Demonstration	Online Service server receives request from web page and initiates an anonymous digital signing request to "iAM Smart" server to get ticketID with parameters businessID, source, redirectURI and eMEFields
2	Program of RA	*demo.eservice.controller.SignController* - getAnonymousHashSignResult (...) *uri: v1/api/anonymous/signing/getSignatureResult* *demo.eservice.controller.SignController* - getAnonymousPDFSignResult (...) *uri: v1/api/anonymous/pdfsigning/getSignatureResult*
	Demonstration	Online Service App initiates polling request for anonymous digital signing processing status
3	N/A	
4	Program of RA	1. Require to process anonymous digital signing request in case of authCallBack API returning businessID Anonymous Hash digital signing: *demo.eservice.controller.SignController* - anonymousHashSign (...) Anonymous PDF digital signing: *demo.eservice.controller.SignController* - anonymousPdfSign (...) 2. Check parameter *demo.eservice.service.impl.LoginServiceImpl* - getAccessTokenByAuthCode(...) 3. Get accessToken and openID with authCode, and store the returned accessToken and openID in redis for future use *demo.eservice.service.impl.LoginServiceImpl* - getAccessToken(...)
	Demonstration	Online Service server requests accessToken and openID from "iAM Smart" server with authCode
5	Program of RA	Invoke the following function to request getting Hash/PDF digital signing result: Anonymous Hash digital signing: *demo.eservice.controller.SignController* - getAnonymousHashSignResult (...)

		Anonymous PDF digital signing: *demo.eservice.controller.SignController* - getAnonymousPDFSignResult (...) *Anonymous Hash digital signing uri:* */v1/api/anonymous/signing/getSignatureResult* *Anonymous PDF digital signing uri:* */v1/api/anonymous/pdfsigning/getSignatureResult* Method: PostdataWithEncrypt(...) - request getting digital signing result
	Demonstration	Online Service server requests getting digital signing result from "iAM Smart"
6	Program of RA	Invoke the following function to request Hash/PDF digital signing: Anonymous Hash digital signing: *demo.eservice.controller.SignController* - getAnonymousHashSignResult (...) Anonymous PDF digital signing: *demo.eservice.controller.SignController* - getAnonymousPDFSignResult (...) *Anonymous Hash digital signing uri:* */v1/api/anonymous/signing/getSignatureResult* *Anonymous PDF digital signing uri:* */v1/api/anonymous/pdfsigning/getSignatureResult* Method: PostdataWithEncrypt(...) - send confirmation of digital signing result reception
	Demonstration	Online Service server initiates digital signing confirmation request to "iAM Smart" server with businessID transferred to "iAM Smart" server in step 1

Reference component		Online Service server (.NET)
1	Program of RA	Invoke the following function to request Hash/PDF digital signing: Anonymous Hash digital signing: *eService_Demo.RESTfulService.svc* - AnonymousHashSign (...) Anonymous PDF digital signing: *eService_Demo.RESTfulService.svc* - AnonymousPdfSign (...)

		Anonymous Hash digital signing uri: */v1/api/anonymous/signing/signature* *Anonymous PDF digital signing uri:* */v1/api/anonymous/pdfsigning/signature* Encrypt business data and request digital signing information from "iAM Smart": *eID.Bussiness.EncryptService* - BizPostWithEncrypt(...)
	Demonstration	Online Service server receives request from web page and initiates an anonymous digital signing request to "iAM Smart" server to get ticketID with parameters businessID, source, redirectURI and eMEFields
2	Program of RA	*eService_Demo.RESTfulService.svc* - GetAnonymousHashSignResult (...) *uri: v1/api/anonymous/signing/getSignatureResult* *eService_Demo.RESTfulService.svc* - GetAnonymousPDFSignResult (...) *uri: v1/api/anonymous/pdfsigning/getSignatureResult*
	Demonstration	Online Service App initiates polling request for anonymous digital signing processing status
3	N/A	
4	Program of RA	1. Require to process anonymous digital signing request in case of authCallBack API returning businessID Anonymous Hash digital signing: *eService_Demo.RESTfulService.svc* - AnonymousHashSign (...) Anonymous PDF digital signing: *eService_Demo.RESTfulService.svc* - AnonymousPdfSign (...) 2. Check parameter 3. Get accessToken and openID with authCode, and store the returned accessToken and openID in redis for future use *eID.Bussiness.Impl.LoginServiceImpl* - GetAccessToken(...)
	Demonstration	Online Service server requests accessToken and openID from "iAM Smart" server with authCode
5	Program of RA	Invoke the following function to request getting Hash/PDF digital signing result: Anonymous Hash digital signing: *eService_Demo.RESTfulService.svc* - GetAnonymousHashSignResult (...) Anonymous PDF digital signing:

		eService_Demo.RESTfulService.svc - GetAnonymousPDFSignResult (...) *Anonymous Hash digital signing uri:* */v1/api/anonymous/signing/getSignatureResult* *Anonymous PDF digital signing uri:* */v1/api/anonymous/pdfsigning/getSignatureResult* Method: *eID.Bussiness.EncryptService* - BizPostWithEncrypt(...) - requests digital signing result
	Demonstration	Online Service server requests getting digital signing result from "iAM Smart"
6	Program of RA	Invoke the following function to request Hash/PDF digital signing: Anonymous Hash digital signing: *eService_Demo.RESTfulService.svc* - GetAnonymousHashSignResult (...) Anonymous PDF digital signing: *eService_Demo.RESTfulService.svc* - GetAnonymousPDFSignResult (...) *Anonymous Hash digital signing uri:* */v1/api/anonymous/signing/getSignatureResult* *Anonymous PDF digital signing uri:* */v1/api/anonymous/pdfsigning/getSignatureResult* Method: *eID.Bussiness.EncryptService* - BizPostWithEncrypt(...) - send confirmation of digital signing result reception
	Demonstration	Online Service server initiates digital signing confirmation request to "iAM Smart" server with businessID transferred to "iAM Smart" server in step 1

Reference component		Online Service App
1	Program of RA	*AMSignViewController.m* - sendSignAction() *sendAMSignRequestWithParams:* initiate anonymous digital signing request
	Demonstration	Online Service App initiates anonymous digital signing request to Online Service server with the in-app browser's user agent value
2	Program of RA	*AMSignViewController.m* - getSignResultRequest ()

		fetchAMSignReqeustWithParams: poll server of getting anonymous digital signing result *AMSignContentViewController.m* - openeIDAction() canOpenURL: determine if “iAM Smart” Mobile App is installed
	Demonstration	Online Service App receives QR Page URL and 4-digit identification code assembled by Online Service server and opens QR Page URL with in-app browser, meanwhile starts initiating polling request to Online Service server to enquiry result of this digital signing request
3	N/A	
4	N/A	
5	Program of RA	*AMSignViewController.m* - getSignResultRequest () showMyToast: show result showSignSuccessVCWithData: redirect to success page
	Demonstration	Online Service App gets digital signing request completion status from Online Service server, and shows anonymous digital signing result
6	N/A	

B. 3. 3. 3. 8 **Bulk Digital Signing**

Reference component		Online Service App
1	Program of RA	*SignPageBDSSVC.m* - onSignBtn () bdssSendNAMRequest: initiate bulk digital signing request
	Demonstration	Online Service App initiates bulk digital signing request to Online Service server with the in-app browser’s user agent value
2	Program of RA	*SignPageBDSSVC.m* - fetchAuthResult () bdssQueryAuth: poll server of getting bulk digital signing result *BDSSCodeVC.m* - openEIDBtnClick () canOpenURL: determine if “iAM Smart” Mobile App is installed
	Demonstration	Online Service App receives QR Page URL and 6-digit identification code assembled by Online Service server and opens QR Page URL with in-app browser, meanwhile starts initiating polling request to Online Service server to enquiry result of this digital signing request
3	N/A	
4	N/A	

5	Program of RA	*SignPageBDSSVC.m* - fetchAuthResult () showMyToast: shows result if failed, otherwise redirect to page AMBDSSAuthedVC
	Demonstration	Online Service App gets digital signing request completion status from Online Service server, and shows bulk digital signing result
6	N/A	

B. 3. 3. 3. 9 **Anonymous Bulk Digital Signing**

Reference component		Online Service App
1	Program of RA	*AMBDSSSignVC.m* - sendSignAction() bdssSendAMRequest: initiate anonymous bulk digital signing request
	Demonstration	Online Service App initiates anonymous bulk digital signing request to Online Service server with the in-app browser's user agent value
2	Program of RA	*AMBDSSSignVC.m* - fetchAuthResult () bdssQueryAuth: poll server of getting anonymous bulk digital signing result *AMBDSSCodeVC.m* - openEIDBtnClick () canOpenURL: determine if "iAM Smart" Mobile App is installed
	Demonstration	Online Service App receives QR Page URL and 6-digit identification code assembled by Online Service server and opens QR Page URL with in-app browser, meanwhile starts initiating polling request to Online Service server to enquiry result of this digital signing request
3	N/A	
4	N/A	
5	Program of RA	*AMBDSSSignVC.m* - fetchAuthResult () 1. SignPageErrorView – showFrom： shows result if failed with code of D71003, D71004 or D71005 2. showMyToast: shows result if failed with other error code 3. If succeed, redirect to page AMBDSSAuthedVC
	Demonstration	Online Service App gets digital signing request completion status from Online Service server, and shows anonymous bulk digital signing result

6	N/A	

B.3.3.4 Online Service App and "iAM Smart" Mobile App are in same device

B. 3. 3. 4. 1 **Authentication**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.1.

Reference component		Online Service App
1	Program of RA	*ELoginViewController.m* - loginAction() canOpenURL: determine if "iAM Smart" Mobile App is installed in the device launchUrl: parameter sent to "iAM Smart" Mobile App
	Demonstration	• Initiate login request to Online Service server • Determined that "iAM Smart" Mobile App is installed in the device • Provide URL scheme of Online Service App to Online Service server
2	Program of RA	*ELoginViewController.m* - loginAction () openURL: pull-up "iAM Smart" Mobile App
	Demonstration	• Online Service App invokes "iAM Smart" Mobile App using URL Scheme to submit the request parameters (uses URL Scheme to pull-up "iAM Smart" Mobile App and sends request parameters to "iAM Smart" Mobile App)
3	Program of RA	*ELoginHelp.m* - handlerLoginWithDic(...) authCode: received parameter from "iAM Smart" Mobile App
	Demonstration	• Online Service App uses callback URL Scheme to receive authorisation code from "iAM Smart" Mobile App
4	Program of RA	*ELoginHelp.m* - handlerLoginWithDic() requestOpenIDByWithParams: send authCode to Online Service server then server returns with tokenID handlerLoginWithDic: withBlock: shows login result goToProfileOrFinish within block: Redirect page for successful login
	Demonstration	• Online Service App sends back authorisation code to Online Service server
5	Program of RA	*ELoginHelp.m* - handlerLoginWithDic()

		handlerLoginWithDic: withBlock: shows login result goToProfileOrFinish within block: Redirect page for successful login
	Demonstration	• Online Service App shows authorisation successful page

B. 3. 3. 4. 2 **Get Profile**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.2.

Reference component		Online Service App
1	Program of RA	*EApplyProfileViewController.m* - onGoToProvideBtn() applyProfileSuccessBlock(): initiate get Profile request to server
	Demonstration	• Online Service App initiates get Profile request to Online Service server
2	Program of RA	*EApplyProfileViewController.m* - checkGetResult() getResult: poll Online Service server *EApplyProfileViewController.m* - onGoToProvideBtn() canOpenURL: determine if "iAM Smart" Mobile App is installed in the device openURL: pull-up "iAM Smart" Mobile App
	Demonstration	• The Online Service App should keep synchronising with the Online Service server for the request result using polling • The Online Service App detected "iAM Smart" Mobile App is installed in the device, uses URL Scheme to pull-up "iAM Smart" Mobile App
3	Program of RA	*EApplyProfileViewController.m* - getProfileSuccessBlock() getToNext: redirect to result page
	Demonstration	• The Online Service Server processes and matches the result and shows the result in the Online Service App

B. 3. 3. 4. 3 **Form Filling**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.3.
For Online Service App, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.3.3.3.

B. 3. 3. 4. 4 **Signing**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.4.
For Online Service App, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.3.3.4.

B. 3. 3. 4. 5 **Re-authentication**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.5.
For Online Service App, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.3.3.5.

B. 3. 3. 4. 6 **Anonymous Form Filling**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.6.

Reference component		Online Service App
1	Program of RA	*FormFillViewController.m* - onFillBtn() requestFormFilling: initiate anonymous form filling request
	Demonstration	Online Service App initiates anonymous form filling request to Online Service server and provides Online Service App URL Scheme information
2	Program of RA	*FormFillViewController.m* - onFillBtn() openURL: pull-up “iAM Smart” Mobile App
	Demonstration	Online Service App pulls up “iAM Smart” Mobile App with URL Scheme
3	Program of RA	AppDelegate.m - application:openURL:options paramsDic: transferred parameter
	Demonstration	“iAM Smart” Mobile App pulls up Online Service App with URL Scheme, and transfers authorisation code to Online Service App
4	Program of RA	FormFillModel.m - onNotification() applyAccess: transfer authorisation code to Online Service server
	Demonstration	Online Service App transfers authorisation code to Online Service server
5	Program of RA	*FormFillViewController.m* - getRequestResult () requestFillingResult: poll server enquiry result

		reloadData: refresh result in UI
	Demonstration	Online Service App displays form filling result

B. 3. 3. 4. 7 **Anonymous Digital Signing**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.7.

Reference component		Online Service App
1	Program of RA	*AMSignViewController.m* - sendSignAction() *sendAMSingRequestWithParams:* initiate anonymous digital signing request
	Demonstration	Online Service App initiates anonymous digital signing request to Online Service server and provides Online Service App URL Scheme information
2	Program of RA	*AMSignViewController.m* - getSignResultRequest () *fetchAMSignReqeustWithParams:* poll server of getting anonymous digital signing result *AMSignContentViewController.m* - openeIDAction() canOpenURL: determine if "iAM Smart" is installed *self.signType:* determine if it is self anonymous digital signing
	Demonstration	Online Service App pulls up "iAM Smart" Mobile App with URL Scheme
3	Program of RA	*Appdelegate.m* - appdelegate:openURL:options: get authorisation code transferred from "iAM Smart" Mobile App Transfer authorisation code to *AMSignViewController.m* via *NSNotificationCenter*
	Demonstration	"iAM Smart" Mobile App pulls up Online Service App with URL Scheme, and transfers authorisation code to Online Service App
4	Program of RA	*AMSignViewController.m* - updateAMSNotify() updateAMSTaskTypeWithParams: transfer authorisation code to server
	Demonstration	Online Service App transfers authorisation code to Online Service server
5	N/A	
6	Program of RA	*AMSignViewController.m* - getSignResultRequest () showMyToast: show result showSignSuccessVCWithData: redirect to success page

	Demonstration	Online Service App shows digital signing confirmation result

B. 3. 3. 4. 8 **Direct Login**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.8.

Reference component		Online Service App
1	Program of RA	*AppDelegate.m* - application:continueUserActivity:restorationHandler() paramsDic: parameters from “iAM Smart” requestQRCodeUrlWithParams: get parameters. eidUrl: parameter sent to “iAM Smart” Mobile App
	Demonstration	• Parse arguments from link • Initiate login request to Online Service server • Provide parameter sent to “iAM Smart” Mobile App
2	Program of RA	*AppDelegate.m* - application:continueUserActivity:restorationHandler() openURL: pull-up “iAM Smart” Mobile App
	Demonstration	• Online Service App invokes “iAM Smart” Mobile App using URL Scheme to submit the request parameters
3	Program of RA	*ELoginHelp.m* - handlerLoginWithDic(...) authCode: received parameter from “iAM Smart” Mobile App
	Demonstration	• Online Service App receives authorisation code from “iAM Smart” Mobile App
4	Program of RA	*ELoginHelp.m* - handlerLoginWithDic() requestOpenIDByWithParams: send authCode to Online Service server then server returns with tokenID handlerLoginWithDic: withBlock: shows login result goToProfileOrFinish within block: Redirect page for successful login
	Demonstration	• Online Service App sends back authorisation code to Online Service server
5	Program of RA	*ELoginHelp.m* - handlerLoginWithDic() handlerLoginWithDic: withBlock: shows login result goToProfileOrFinish within block: Redirect page for successful login
	Demonstration	• Online Service App shows authorisation successful page

B. 3. 3. 4. 9 **Bulk Digital Signing**

For Online Service server, please refer to Section B.3.2.2.9.

Reference component		Online Service App
1	Program of RA	*SignPageBDSSVC.m* - onSignBtn () bdssSendNAMRequest: initiate bulk digital signing request
	Demonstration	Online Service App initiates anonymous digital signing request to Online Service server and provides Online Service App URL Scheme information
2	Program of RA	*SignPageDSSVC.m* - fetchAuthResult () bdssQueryAuth: poll server of getting bulk digital signing result *BDSSCodeVC.m* - openEIDBtnClick () canOpenURL: determine if "iAM Smart" Mobile App is installed
	Demonstration	Online Service App receives 6-digit identification code assembled by Online Service server and opens "iAM Smart" Mobile App, meanwhile starts initiating polling request to Online Service server to enquiry result of this digital signing request
3	N/A	
4	N/A	
5	Program of RA	*SignPageBDSSVC.m* - fetchAuthResult () showMyToast: shows result if failed, otherwise redirect to page AMBDSSAuthedVC
	Demonstration	Online Service App gets digital signing request completion status from Online Service server, and shows bulk digital signing result
6	N/A	

B. 3. 3. 4. 10 **Anonymous Bulk Digital Signing**

For Online Service server, please refer to Section B.3.2.2.10.

Reference component		Online Service App
1	Program of RA	*AMBDSSSignVC.m* - sendSignAction() bdssSendAMRequest: initiate anonymous bulk digital signing request

	Demonstration	Online Service App initiates anonymous digital signing request to Online Service server and provides Online Service App URL Scheme information
2	Program of RA	*AMBDSSSignVC.m* - fetchAuthResult () bdssQueryAuth: poll server of getting anonymous bulk digital signing result *AMBDSSCodeVC.m* - openEIDBtnClick () canOpenURL: determine if "iAM Smart" Mobile App is installed
	Demonstration	Online Service App receives 6-digit identification code assembled by Online Service server and opens "iAM Smart" Mobile App, meanwhile starts initiating polling request to Online Service server to enquiry result of this digital signing request
3	N/A	
4	N/A	
5	Program of RA	*AMBDSSSignVC.m* - fetchAuthResult () 1. SignPageErrorView – showFrom：shows result if failed with code of D71003, D71004 or D71005 2. showMyToast: shows result if failed with other error code 3. If succeed, redirect to page AMBDSSAuthedVC
	Demonstration	Online Service App gets digital signing request completion status from Online Service server, and shows anonymous bulk digital signing result
6	N/A	

B.3.4 Android Application Development

B.3.4.1 URL scheme Setup in Online Service App

Prerequisites: Online Service developers are required to provide the URL Scheme of Online Service App when register an Online Service access account from DPO. After registered, the "iAM Smart" App URL scheme, e.g., hk.gov.iamsmart, will be provided to Online Service.

Step 1: In Online Service App, open *LoginActivity.kt->toStartLogin() ->Uri.parse()* and add "iAM Smart" Mobile App URL scheme, e.g. Url.parse(hk.gov.iamsmart://params?key=value).

Step 2: In file AndroidManifest.xml, add the corresponding scheme to the list, this is the ingress-controller pulled up by "iAM Smart" Mobile App, e.g. adds hk.gov.eservice001 to EidEnterActivity.

B.3.4.2 Notes for AppLink Setup in Online Service App

Step 1: Add the file with the configuration name assetlinks.json to the Online Service server. The format is as follow, where the package_name is the package name of Online Service App and the sha256_cert_fingerprints is the SHA value:

```
 [{
  "relation": ["delegate_permission/common.handle_all_urls"],
  "target": {
    "namespace": "android_app",
    "package_name": "com.example.demo",
    "sha256_cert_fingerprints":
    ["EE:04:5B:7B:63:FB:04:B7:21:0A:44:70:1D:2A:F7:2A:53:F6:A3:F9:92:B1
:8F:1F:10:EF:66:9A:1D:E1:61:F4"]
  }
}]
```

Step 2: Put assetlinks.json in the server directory:

https://domain/.well-known/assetlinks.json

Step 3: Perform the following configuration in AndroidManifest.xml:

```
<activity
    android:name=".AppLinkActivity">
    <intent-filter android:autoVerify="true">
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data
            android:host="domain"
            android:scheme="https" />
    </intent-filter>
</activity>
```

Step 4: Enable the mobile network access to the google service.

B.3.4.3 Online Service App and "iAM Smart" Mobile App are in different devices

B. 3. 4. 3. 1 Authentication

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.1.

Reference component		Online Service App
1	Program of RA	*LoginActivity.kt* Method:toStartLogin() *AppUtils*.checkPackInfo(): checks if "iAM Smart" Mobile App is installed in the device
	Demonstration	• Initiate login request to Online Service server • Determined that "iAM Smart" Mobile App is not installed in the device • Provide URL scheme of Online Service App to Online Service server
2	Program of RA	*LoginActivity.kt* Method: toStartLogin() - function for Online Service App to pull-up in-app browser in else block
	Demonstration	• Online Service App invokes in-app browser to submit the get QR Page request (pulls-up in-app browser and opens get QR Page URL)
3	Program of RA	*LoginActivity.kt* - initData(): resolves the URL sent by the in-app browser Resolves query parameter to obtain the authCodeId. After getTokenID() get the tokenID from server API with authCodeId, enters ProvideInformationActivity.kt to shows login successful page
	Demonstration	• Online Service App receives the authorisation code from "iAM Smart" QR Page • Invoke Online Service server API to request accessToken and Tokenised ID • Show the login result page after receiving Online Service response

B. 3. 4. 3. 2 **Get Profile**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.2.

Reference component		Online Service App
1	Program of RA	*ProvideInformationActivity.kt* - initClick() is the click event of mBinding.tvOpenEid
	Demonstration	• Online Service App initiates get Profile request to Online Service server
2	Program of RA	*NoEidNoticeActivity.kt* - getProfile() Invoke polling result API
	Demonstration	• The Online Service App should keep synchronising with the Online Service server for the request result using polling
3	Program of RA	Redirect after polling succeeded *ProvideInformationSuccessActivity.kt* shows the successful page
	Demonstration	• The Online Service Server processes and matches the result and shows the result in the Online Service App

B. 3. 4. 3. 3 **Form Filling**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.3.

Reference component		Online Service App
1	Program of RA	*AllowFillFormActivity.kt* - initClick() is the click event of mBinding.tvUseEidFill, Invoke sendEmeReq() to initiate request
	Demonstration	• Online Service App initiates form filling request to Online Service server
2	Program of RA	*NoEidNoticeActivity.kt* - getFillFormResult() Invoke polling result API
	Demonstration	• The Online Service App should keep synchronising with the Online Service server for the request result using polling
3	Program of RA	Redirects after result polling succeeded *AllowFillFromActivity.kt* shows successful page
	Demonstration	• The Online Service Server processes and matches the result and shows the result in the Online Service App

B. 3. 4. 3. 4 **Signing**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.4.

Reference component		Online Service App
1	Program of RA	*SignatureOkActivity.kt* - initClick()is the click event of mBinding.tvOpenEid
	Demonstration	• Online Service App initiates digital signing request to Online Service server
2	Program of RA	AllowSignatureActivity.kt - getSignatureResult() Invoke polling result API
	Demonstration	• The Online Service App should keep synchronising with the Online Service server for the request result using polling
3	Program of RA	Redirects after polling succeeded AllowSignatureActivity.kt shows successful page
	Demonstration	• The Online Service Server processes and matches the result and shows the result in the Online Service App • CommonOkActivity.kt - downloadFile() is the click event of mBinding.tvDownload

B. 3. 4. 3. 5 **Re-authentication**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.5.

Reference component		Online Service App
1	Program of RA	*StrongCertifiedActivity.kt* - getRequestId() is the click event of mBinding.tvUseEidCertified
	Demonstration	• Online Service App initiates Re-authentication request to Online Service server
2	Program of RA	*NoEidNoticeActivity.kt* - getStrongCertifiedResult(): Invoke polling result API
	Demonstration	• The Online Service App should keep synchronising with the Online Service server for the request result using polling
3	Program of RA	Redirects after polling succeeded *StrongCertifiedActivity.kt* - certifiedSuccess(): shows success page
	Demonstration	• The Online Service Server processes and matches the result and shows the result in the Online Service App

B. 3. 4. 3. 6 **Anonymous Form Filling**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.6.

Reference component		Online Service App
1	Program of RA	*AnonymousAutoFillingActivity.kt* - requestAnonymousEMEInfo(): invoke API to initiate request
	Demonstration	Online Service App initiates anonymous form filling request to Online Service server and provides App URL Scheme information
2	Program of RA	*AnonymousAutoFillingActivity.kt* - toWebView(): invoke this method with URL to open QR Code page after getting parameter redirectURI from server *AnonymousAutoFillingActivity.kt* - getFillFormResult(): invoke this method to start polling result from server if step 1 is succeed
	Demonstration	Online Service App receives QR Page URL assembled by Online Service server and opens it with in-app browser, meanwhile starts polling result of server enquiry request
3	N/A	
4	N/A	
5	Program of RA	*AnonymousAutoFillingActivity.kt* - applyAccessToken(): After QR Code page gets result in polling, transfer authorisation code to server and uses authorisation code and other parameters to pull-up Online Service App *AnonymousAutoFillingActivity.kt* - getFillFormResult(): process the result get in polling *Superclass AllowFillFromActivity.kt of AnonymousAutoFillingActivity.kt* - showContent(): show result data
	Demonstration	Online Service web page displays form filling result

B. 3. 4. 3. 7 **Anonymous Digital Signing**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.7.

Reference component	Online Service App

1	Program of RA	*AnonymousSignActivity.kt* - requestSign(): initiate anonymous digital signing request
	Demonstration	Online Service App initiates anonymous digital signing request to Online Service server and provides App URL Scheme information
2	Program of RA	*AnonymousSignActivity.kt* - getSignatureResult(): invoke this method to start polling result from server if step 1 succeed *AnonymousSignOkActivity.kt* - initClick(): Invoke this method with polling result as the parameter and process the click event to show QR Code with in-app browser
	Demonstration	Online Service App receives QR Page URL and 4-digit identification code assembled by Online Service server and opens QR Page URL with in-app browser, meanwhile starts initiating polling request to Online Service server to enquiry result of this digital signing request
3	N/A	
4	N/A	
5	Program of RA	*AnonymousSignActivity.kt* - onAnonymousFormFillingEvent(): After QR Code page gets result in polling, transfer authorisation code to server and uses authorisation code and other parameters to pull-up Online Service App *AnonymousSignActivity.kt* - getSignatureResult (): get result in polling after authorisation code is transferred to server *AnonymousSignActivity.kt:* show result
	Demonstration	Online Service App gets digital signing request completion status from Online Service server, and shows form filling result *CommonOkActivity.kt* - downloadFile(): click event of mBinding.tvDownload
6	N/A	

B. 3. 4. 3. 8 **Bulk Digital Signing**

For Online Service server, please refer to Section B.3.2.2.9.

Reference component		Online Service App
1	Program of RA	*SignatureOkActivity.kt* - initClick() is the click event of mBinding.tvOpenEid
	Demonstration	• Online Service App initiates digital signing request to Online Service server

2	Program of RA	AllowSignatureActivity.kt - getSignatureResult() Invoke polling result API
	Demonstration	• The Online Service App should keep synchronising with the Online Service server for the request result using polling
3	Program of RA	Redirects after polling succeeded AllowSignatureActivity.kt shows successful page
	Demonstration	• The Online Service Server processes and matches the result and shows the result in the Online Service App • CommonOkActivity.kt - downloadFile() is the click event of mBinding.tvDownload

B. 3. 4. 3. 9 **Anonymous Bulk Digital Signing**

For Online Service server, please refer to Section B.3.2.2.10.

Reference component		Online Service App
1	Program of RA	*BulkSignActivity.kt* - sendSignatureRequest(): initiate anonymous digital signing request
	Demonstration	Online Service App initiates anonymous digital signing request to Online Service server and provides App URL Scheme information
2	Program of RA	*BulkSignActivity.kt* - getSignatureResult(): invoke this method to start polling result/ from server if step 1 succeed *AnonymousSignOkActivity.kt* - initClick(): Invoke this method with polling result as the parameter and process the click event to show QR Code with in-app browser
	Demonstration	Online Service App receives QR Page URL and 6-digit identification code assembled by Online Service server and opens QR Page URL with in-app browser, meanwhile starts initiating polling request to Online Service server to enquiry result of this digital signing request
3	N/A	
4	N/A	
5	Program of RA	*BulkSignActivity.kt* - onAnonymousSignEvent (): After QR Code page gets result in polling, transfer authorisation code to server and uses authorisation code and other parameters to pull-up Online Service App *BulkSignActivity.kt* - getSignatureResult (): get result in polling after authorisation code is transferred to server

		BulkSignActivity.kt: show result
	Demonstration	Online Service App gets digital signing request completion status from Online Service server, and shows form filling result *CommonOkActivity.kt* - downloadFile(): click event of mBinding.tvDownload
6	N/A	

B.3.4.4 Online Service App and "iAM Smart" Mobile App are in same device

B. 3. 4. 4. 1 **Authentication**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.1

Reference component		Online Service App
1	Program of RA	*LoginActivity.kt* Method: toStartLogin() *AppUtils*.checkPackInfo(): checks if "iAM Smart" Mobile App is installed in the device
	Demonstration	• Initiate login request to Online Service server • Determined "iAM Smart" Mobile App is installed in the device • Provide URL scheme of Online Service App to Online Service server
2	Program of RA	*LoginActivity.kt* Method: toStartLogin() - pull-up "iAM Smart" Mobile App and send the request parameter in the if block
	Demonstration	• Online Service App invokes "iAM Smart" Mobile App using URL Scheme to submit the request parameters (uses URL Scheme to pull-up "iAM Smart" Mobile App and sends request parameters to "iAM Smart" Mobile App)
3	Program of RA	*LoginActivity.kt* - initData(): resolves the callback URL sent from "iAM Smart" Mobile App
	Demonstration	• Online Service App receives callback from "iAM Smart" Mobile App using URL Scheme, receives authorisation code
4	Program of RA	Invoke API *LoginActivity.kt* - getTokenID(): gets tokenID with the authorisation code

	Demonstration	• Online Service App returns authorisation code to Online Service server
5	Program of RA	Gets the tokenID to login *ProvideInformationActivity.kt:* shows login successful page
	Demonstration	• Online Service App shows authorisation successful page

B. 3. 4. 4. 2 **Get Profile**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.2.

Reference component		Online Service App
1	Program of RA	*ProvideInformationActivity.kt* - initClick() is the click event of mBinding.tvOpenEid
	Demonstration	• Online Service App initiates get Profile request to Online Service server
2	Program of RA	*ProvideInformationActivity.kt* - getProfile() invokes polling result API
	Demonstration	• The Online Service App should keep synchronising with the Online Service server for the request result using polling • The Online Service App detected “iAM Smart” Mobile App is installed in the device, uses URL Scheme to pull-up “iAM Smart” Mobile App
3	Program of RA	After polling succeeded *ProvideInformationActivity.kt* shows successful page
	Demonstration	• The Online Service Server processes and matches the result and shows the result in the Online Service App

B. 3. 4. 4. 3 **Form Filling**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.3.

Reference component		Online Service App
1	Program of RA	*AllowFillFromActivity.kt* - initClick() is the click event of mBinding.tvUseEidFill, Invoke sendEmeReq() to initiate request
	Demonstration	• Online Service App initiates form filling request to Online Service server

2	Program of RA	*AllowFillFromActivity.kt* - getFillFormResult() Invoke polling result API
	Demonstration	• The Online Service App should keep synchronising with the Online Service server for the request result using polling • The Online Service App detected "iAM Smart" Mobile App is installed in the device, uses URL Scheme to pull-up "iAM Smart" Mobile App
3	Program of RA	After polling result succeeded *AllowFillFromActivity.kt* shows successful page
	Demonstration	• The Online Service Server processes and matches the result and shows the result in the Online Service App

B. 3. 4. 4. 4 **Signing**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.4.

Reference component		Online Service App
1	Program of RA	*SignatureOkActivity.kt* - initClick() is the click event of mBinding.tvOpenEid
	Demonstration	• Online Service App initiates digital signing request to Online Service server
2	Program of RA	*AllowSinatureActivity.kt* - getSignatureResult() Invoke polling result API
	Demonstration	• The Online Service App should keep synchronising with the Online Service server for the request result using polling • The Online Service App detected "iAM Smart" Mobile App is installed in the device, uses URL Scheme to pull-up "iAM Smart" Mobile App
3	Program of RA	After polling result succeeded *AllowSinatureActivity.kt* shows result successful page
	Demonstration	• The Online Service Server processes and matches the result and shows the result in the Online Service App

B. 3. 4. 4. 5 **Re-authentication**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.5.

Reference component		Online Service App
1	Program of RA	*StrongCertifiedActivity.kt* - getRequestId(): click event of mBinding.tvUseEidCertified
	Demonstration	• Online Service App initiates Re-authentication request to Online Service server
2	Program of RA	*StrongCertifiedActivity.kt* - getStrongCertifiedResult(): Invoke polling result API
	Demonstration	• The Online Service App should keep synchronising with the Online Service server for the request result using polling • The Online Service App detected "iAM Smart" Mobile App is installed in the device, uses URL Scheme to pull-up "iAM Smart" Mobile App
3	Program of RA	After receive polling result and its succeeded *StrongCertifiedActivity.kt* shows successful page
	Demonstration	• The Online Service Server processes and matches the result and shows the result in the Online Service App

B.3.4.4.6 **Anonymous Form Filling**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.6.

Reference component		Online Service App
1	Program of RA	*AnonymousAutoFillingActivity.kt* - requestAnonymousEMEInfo(): invoke API to initiate form filling request
	Demonstration	Online Service App initiates anonymous form filling request to Online Service server and provides Online Service App URL Scheme information
2	Program of RA	*AnonymousAutoFillingActivity.kt* - toMySelfAnonymous(): pull up "iAM Smart" Mobile App if AppUtils.checkPackInfo() is true after getting redirectURI, ticketID and other parameters from server
	Demonstration	Online Service App pulls up "iAM Smart" Mobile App with URL Scheme
3	Program of RA	*EidEnterActivity.kt* - initDatas(): After "iAM Smart" Mobile App pulls up Online Service App with URL schema, get authorisation code transferred from "iAM

		Smart" Mobile App and transfer authorisation code to AnonymousAutoFillingActivity.kt by EventBus
	Demonstration	"iAM Smart" Mobile App pulls up Online Service App with URL Scheme, and transfers authorisation code to Online Service App
4	Program of RA	*AnonymousAutoFillingActivity.kt* - applyAccessToken(): invoke this method to transfer authorisation code to server
	Demonstration	Online Service App transfers authorisation code to Online Service server
5	Program of RA	*AnonymousAutoFillingActivity.kt* -getFillFormResult(): invoke this method to get result if step 4 is succeed *Superclass AllowFillFromActivity.kt of AnonymousAutoFillingActivity.kt* - showContent(): show result data
	Demonstration	Online Service App displays form filling result

B. 3. 4. 4. 7 **Anonymous Digital Signing**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.7.

Reference component		Online Service App
1	Program of RA	*AnonymousSignActivity.kt* - requestSign(): initiate anonymous digital signing request
	Demonstration	Online Service App initiates anonymous digital signing request to Online Service server and provides Online Service App URL Scheme information
2	Program of RA	*AnonymousSignActivity.kt* - getSignatureResult(): invoke this method to start polling result from server if step 1 is succeed *AnonymousSignOkActivity.kt* - initClick(): After step 1 is succeed, pull up "iAM Smart" Mobile App if AppUtils.checkPackInfo() is true. Get redirectURI to process click event
	Demonstration	Online Service App pulls up "iAM Smart" Mobile App with URL Scheme
3	Program of RA	EidEnterActivity.kt - initDatas(): After QR Code page gets result in polling, Online Service App gets authorisation code transferred from "iAM Smart" Mobile App and uses authorisation code and

		other parameters to pull up Online Service App, and to transfer authorisation code to *AnonymousSignActivity.kt* by EventBus
	Demonstration	"iAM Smart" Mobile App pulls up Online Service App with URL Scheme, and transfers authorisation code to Online Service App
4	Program of RA	*AnonymousSignActivity.kt* - onAnonymousFormFillingEvent(): invoke this method to transfer authorisation code to server
	Demonstration	Online Service App transfers authorisation code to Online Service server
5	N/A	
6	Program of RA	*AnonymousSignActivity.kt* - getSignatureResult (): get result by polling if process of step 4 succeeds, and show result on AnonymousSignActivity.kt page
	Demonstration	Online Service App shows digital signing confirmation result

B. 3. 4. 4. 8 **Direct Login**

For Online Service server, please refer to Section 20.9338264.1455428365.509163592.9338560B.3.2.2.8.

Reference component		Online Service App
1	Program of RA	*LoginActivity.kt* Method: onCreate() - Determine whether the tag is AppLink: If yes, directly call the applyQrCode() request API, and with a success, call the toStartLogin() API.
	Demonstration	• Parse arguments from link • Initiate login request to Online Service server • Provide parameter sent to "iAM Smart" Mobile App
2	Program of RA	*LoginActivity.kt* Method: toStartLogin() - pull-up "iAM Smart" Mobile App and send the request parameter if "iAM Smart" Mobile App is installed in the device
	Demonstration	• Online Service App invokes "iAM Smart" Mobile App using URL Scheme to submit the request parameters
3	Program of RA	*LoginActivity.kt* - initData(): resolves the callback URL sent from "iAM Smart" Mobile App
	Demonstration	• Online Service App receives authorisation code from "iAM Smart" Mobile App
4	Program of RA	Invoke API

		LoginActivity.kt - getTokenID(): gets tokenID with the authorisation code
	Demonstration	• Online Service App returns authorisation code to Online Service server
5	Program of RA	Gets the tokenID to login *ProvideInformationActivity.kt:* shows login successful page
	Demonstration	• Online Service App shows authorisation successful page

B. 3. 4. 4. 9 **Bulk Digital Signing**

For Online Service server, please refer to Section B.3.2.2.9.

Reference component		Online Service App
1	Program of RA	*SignatureOkActivity.kt* - initClick() is the click event of mBinding.tvOpenEid
	Demonstration	• Online Service App initiates digital signing request to Online Service server
2	Program of RA	*BulkSignActivity.kt* - getSignatureResult() Invoke polling result API
	Demonstration	• The Online Service App should keep synchronising with the Online Service server for the request result using polling • The Online Service App detected “iAM Smart” Mobile App is installed in the device, uses URL Scheme to pull-up “iAM Smart” Mobile App
3	Program of RA	After polling result succeeded *AllowSinatureActivity.kt* shows result successful page
	Demonstration	• The Online Service Server processes and matches the result and shows the result in the Online Service App

B. 3. 4. 4. 10 **Anonymous Bulk Digital Signing**

For Online Service server, please refer to Section B.3.2.2.10.

Reference component		Online Service App
1	Program of RA	*BulkSignActivity.kt* - sendSignatureRequest (): initiate anonymous digital signing request

	Demonstration	Online Service App initiates anonymous digital signing request to Online Service server and provides Online Service App URL Scheme information
2	Program of RA	*BulkSignActivity.kt* - getSignatureResult(): invoke this method to start polling result from server if step 1 is succeed *AnonymousSignOkActivity.kt* - initClick(): After step 1 is succeed, pull up "iAM Smart" Mobile App if AppUtils.checkPackInfo() is true. Get redirectURI to process click event
	Demonstration	Online Service App pulls up "iAM Smart" Mobile App with URL Scheme
3	Program of RA	EidEnterActivity.kt - initDatas(): After QR Code page gets result in polling, Online Service App gets authorisation code transferred from "iAM Smart" Mobile App and uses authorisation code and other parameters to pull up Online Service App, and to transfer authorisation code to *BulkSignActivity.kt* by EventBus
	Demonstration	"iAM Smart" Mobile App pulls up Online Service App with URL Scheme, and transfers authorisation code to Online Service App
4	Program of RA	*BulkSignActivity.kt* - onAnonymousSignEvent (): invoke this method to transfer authorisation code to server
	Demonstration	Online Service App transfers authorisation code to Online Service server
5	N/A	
6	Program of RA	*BulkSignActivity.kt* - getSignatureResult (): get result by polling if process of step 4 succeeds, and show result on BulkSignActivity.kt page
	Demonstration	Online Service App shows digital signing confirmation result

C. STREAMLINE WORKFLOW SCENARIOS

The Streamline Workflow is one of the important enhancement to realise "single portal for online government services (一網通辦)", aiming to provide the simplified and seamless "iAM Smart" workflow with a view to enhancing user experience without extra switching back and forth between "iAM Smart" mobile app and Online Services.

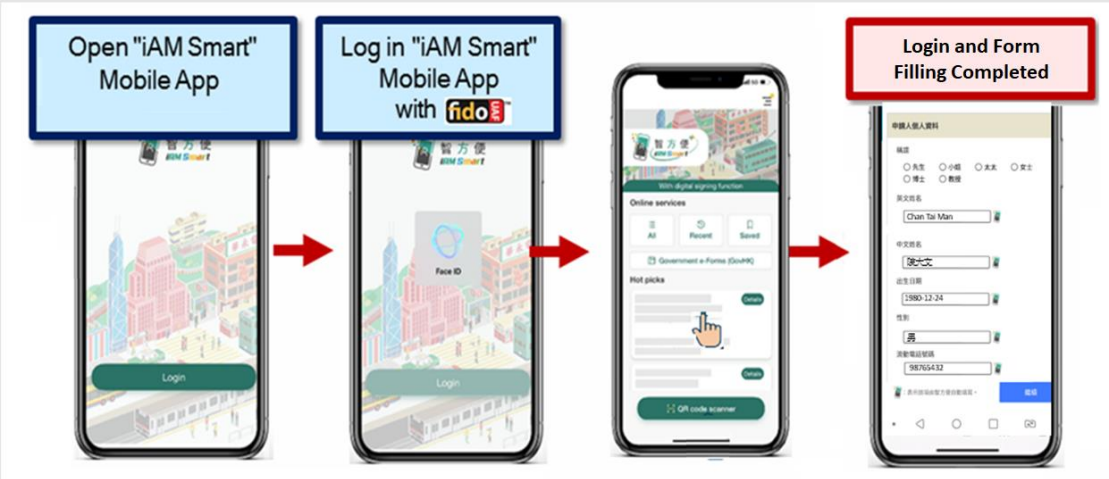
Online Services shall follow both "iAM Smart API Specification" and "iAM Smart Developer Guide" for "iAM Smart" Adoption. In addition, DPO has appointed a Consultant to gauge user needs, expectations and aspiration on the future single portal for online government services and will provide new "Reference System Flow and User Interface Specifications" for reference in late 2023.

The below examples illustrate how the Streamline Workflow can be fully utilised among different business scenarios.

Remark: Online Services can request or collect personal data of users from "iAM Smart" in accordance with the requirements under the Personal Data (Privacy) Ordinance ("PD(P)O") (Cap. 486). After the personal data is transferred to Online Services (data users), Online Services undertake to ensure the personal data are held, processed and used in accordance with the PD(P)O, including but not limited to ***giving a personal information collection statement ("PICS") stating clearly how they will collect, use and process the personal data of the users from the "iAM Smart" system.***

C.1 Scenario 1

User directly login to Online Service Website via "iAM Smart" service catalogue and then retrieve "eMEFields" for form filling. User can modify form data if required and then submit the form.



Online Services should refer to below table for above implementation.

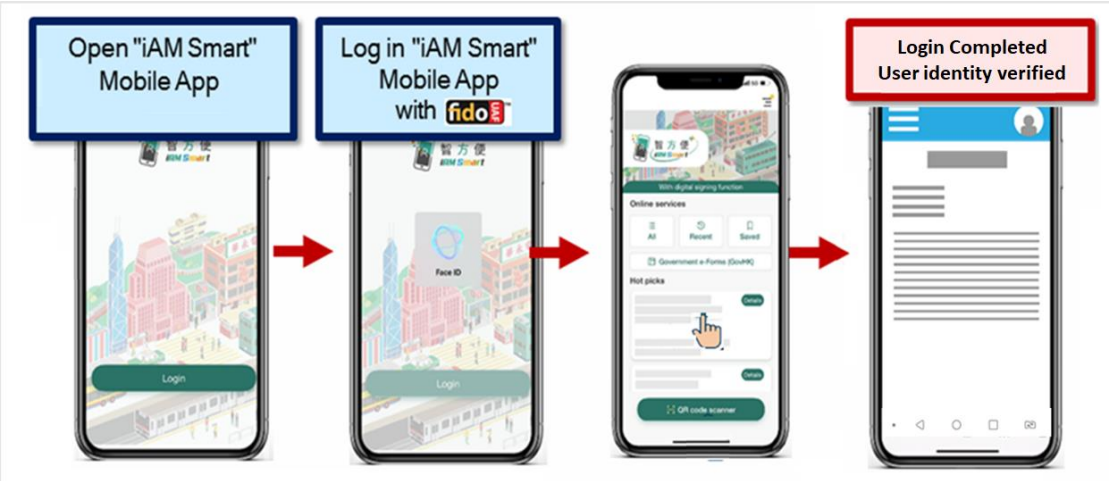
	Developer Guide	API Specification
Direct Login v2	Section 3.10.1	Section 3.3.2.2
Form Filling with Service Login (i.e., Profiles)	Section 3.5	Section 5

Remark:
Existing Online Service hosted in Service Catalogue should also refer below table to take necessary upgrade action.

For Online Service adopting below API	Impact	Action
Direct Login v1	Yes	Migrate to Direct Login v2
Anonymous Form Filling	Yes	Implement Authentication process Migrate to Profiles
Form Filling with Service Login v1 or v2	Yes	Migrate to Profiles

C.2 Scenario 2

User directly login to Online Service Website via “iAM Smart” service catalogue and then Online Service requires “profileFields” for user identity verification, account link up or account on boarding. User is not allowed to update “profileFields” in the Online Service Website.



Online Services should refer to below table for above implementation.

	Developer Guide	API Specification
Direct Login v2	Section 3.10.1	Section 3.3.2.2
Form Filling with Service Login (i.e., Profiles)	Section 3.5	Section 5

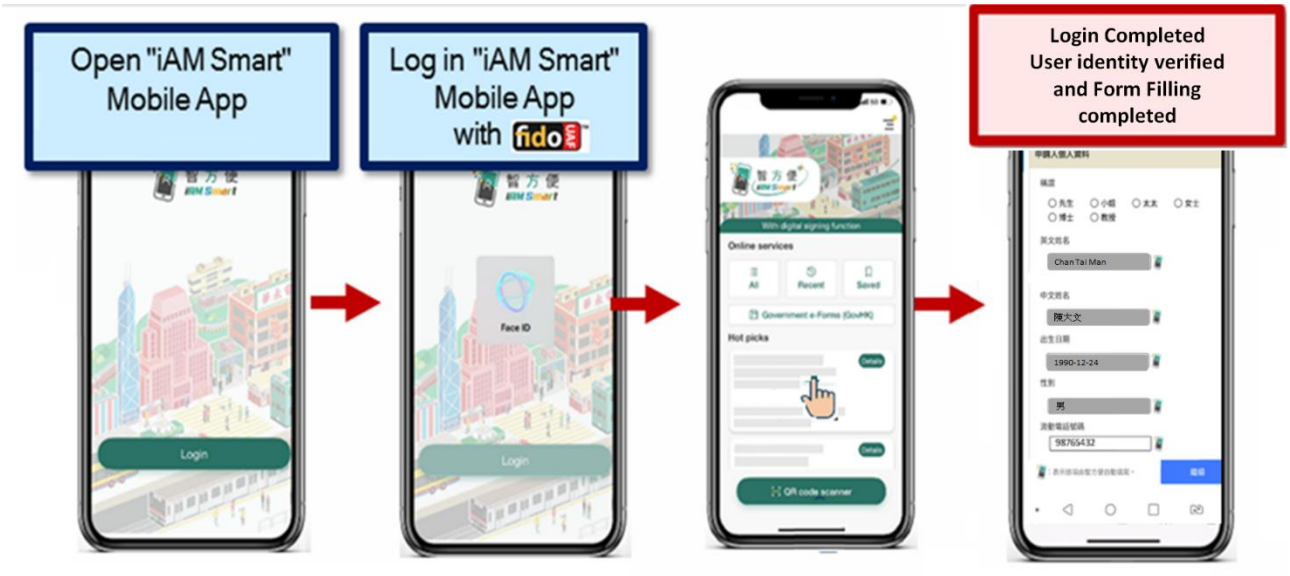
Remark:
Existing Online Service hosted in Service Catalogue should also refer below table to take necessary upgrade action.

For Online Service adopting below API	Impact	Action
Direct Login v1	Yes	Migrate to Direct Login v2
getProfile	Yes	Migrate to Profiles

C.3 Scenario 3

User directly login to Online Service Website via “iAM Smart” service catalogue and then Online Service requires both “profileFields” and “eMEFields” for user identity verification and form filling respectively. (Scenario 1 + 2)

The account information in “profileFields” and “eMEFields” are the same for same user (e.g., Online Service would get the same HKIC number of specific “iAM Smart” user, no matter the information is from “profileFields” or “eMEFields”). However, both authentication and form filling statistic report in “iAM Smart” system would be updated if Online Service requests both “profileFields” and “eMEFields” in one Profiles API call. In addition, unlike “profileFields” which is for the purpose of identity verification, Online Service can allow “iAM Smart” user to modify online form data filled with “eMEFields”.



Online Services should refer to below table for above implementation.

	Developer Guide	API Specification
Direct Login v2	Section 3.10.1	Section 3.3.2.2
Form Filling with Service Login (i.e., Profiles)	Section 3.5	Section 5

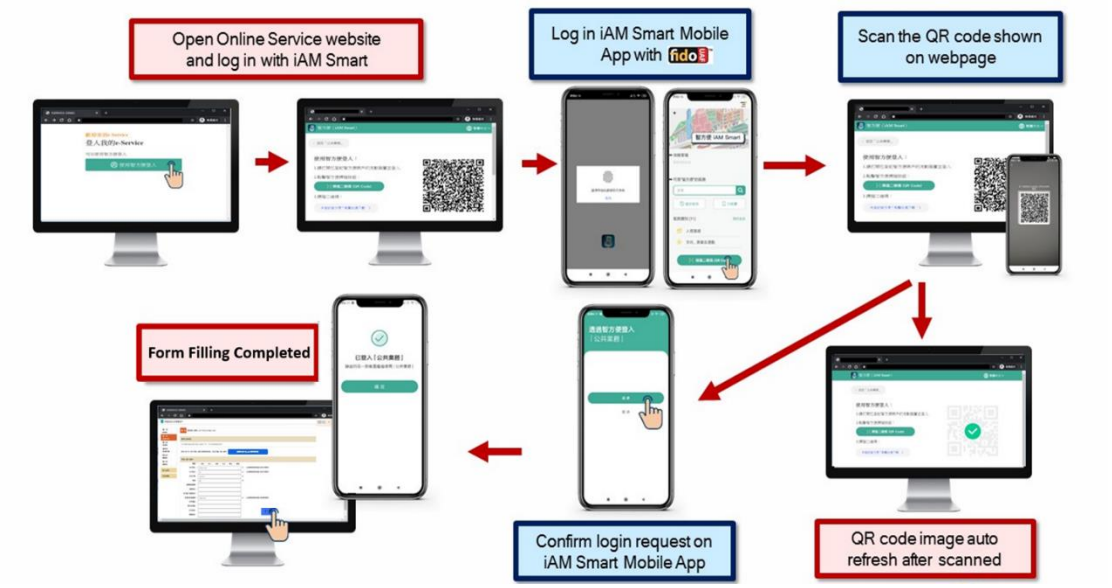
Remark:
Existing Online Service hosted in Service Catalogue should also refer below table to take

necessary upgrade action.

For Online Service adopting below API	Impact	Action
Direct Login v1	Yes	Migrate to Direct Login v2
getProfile	Yes	Migrate to Profiles
Form Filling with Service Login v1 or v2	Yes	Migrate to Profiles

C.4 Scenario 4

User opens Online Service Website via web browser (different device) and then login Online Service with “iAM Smart”. Upon successful of Authentication process, Online Service retrieves “eMEFields” for form filling.



Online Services should refer to below table for above implementation.

	Developer Guide	API Specification
Authentication	Section 3.4.1	Section 3.3.2.1

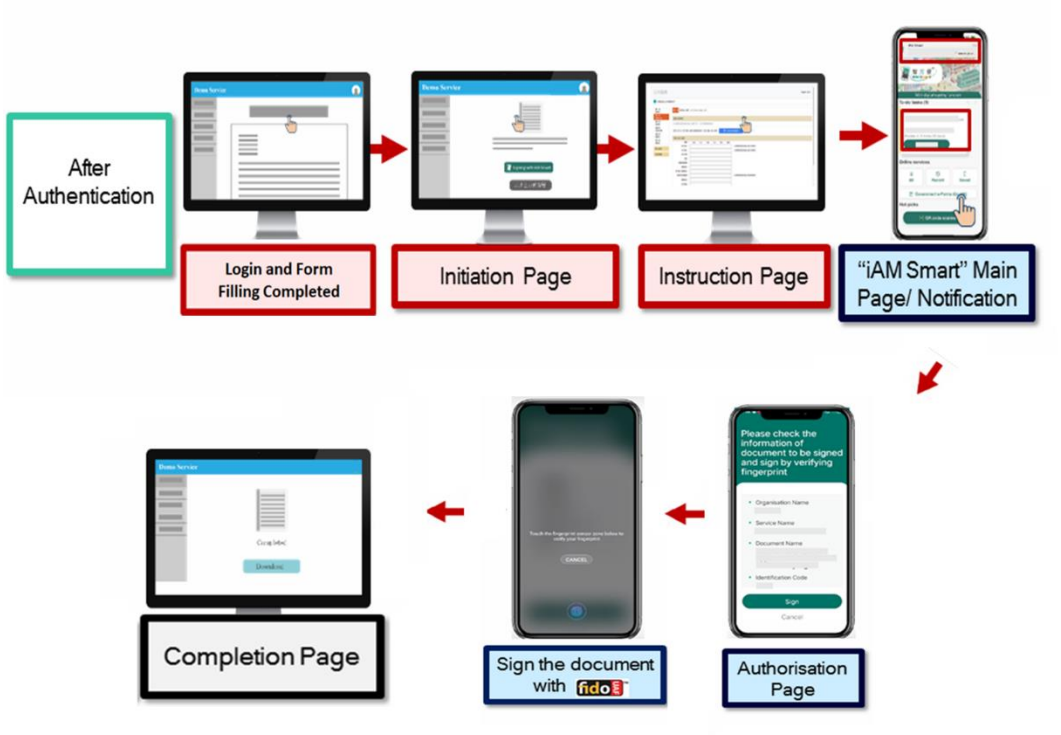
Form Filling with Service Login (i.e., Profiles)	Section 3.5	Section 5

Remark:
Existing Online Service should also refer below table to take necessary upgrade action.

For Online Service adopting below API	Impact	Action
getProfile	Yes	Migrate to Profiles
Form Filling with Service Login v1 or v2	Yes	Migrate to Profiles
Anonymous Form Filling v1 (support multiple "iAM Smart" users to fill the same online form)	Yes	Migrate to Anonymous Form Filling v2

C.5 Scenario 5

User opens Online Service Website via web browser (different device) and then login Online Service with "iAM Smart". Upon successful of Authentication process, Online Service retrieves "eMEFields" for form filling and perform digital signing.



	Developer Guide	API Specification
Authentication	Section 3.4.1	Section 3.3.2.1
Form Filling with Service Login (i.e., Profiles)	Section 3.5	Section 5
Digital Signing with Service Login	Section 3.7	Section 8

Remark:
Existing Online Service should also refer below table to take necessary upgrade action.

For Online Service adopting below API	Impact	Action

getProfile	Yes	Migrate to Profiles
Form Filling with Service Login v1 or v2	Yes	Migrate to Profiles
Anonymous Form Filling v1 (support multiple “iAM Smart” users to fill the same online form)	Yes	Migrate to Anonymous Form Filling v2
Anonymous Digital Signing (single user digital signing)	Yes	Migrate to Digital Signing with Service Login
Anonymous Digital Signing (support multiple “iAM Smart” users to sign the same application)	No	N/A

D. ESSENTIAL TIPS FOR APP-TO-APP DIRECT LOGIN V2

Direct Login v2 has adopted a new approach on Android for launching the online service app. The `authCode` and related parameters are passed in the form of an Android Intent. This section outlines essential tips and common pitfalls associated with Direct Login v2, including issues where the online service app does not launch successfully and the `authCode` cannot be retrieved from the "iAM Smart" Android Intent.

D.1 Prerequisites to launch Android App via Package Name

In order to launch the online service app from the "iAM Smart" App catalogue successfully, the following configuration shall be valid:

- Package name
- Activity name
- Signing certificate fingerprint

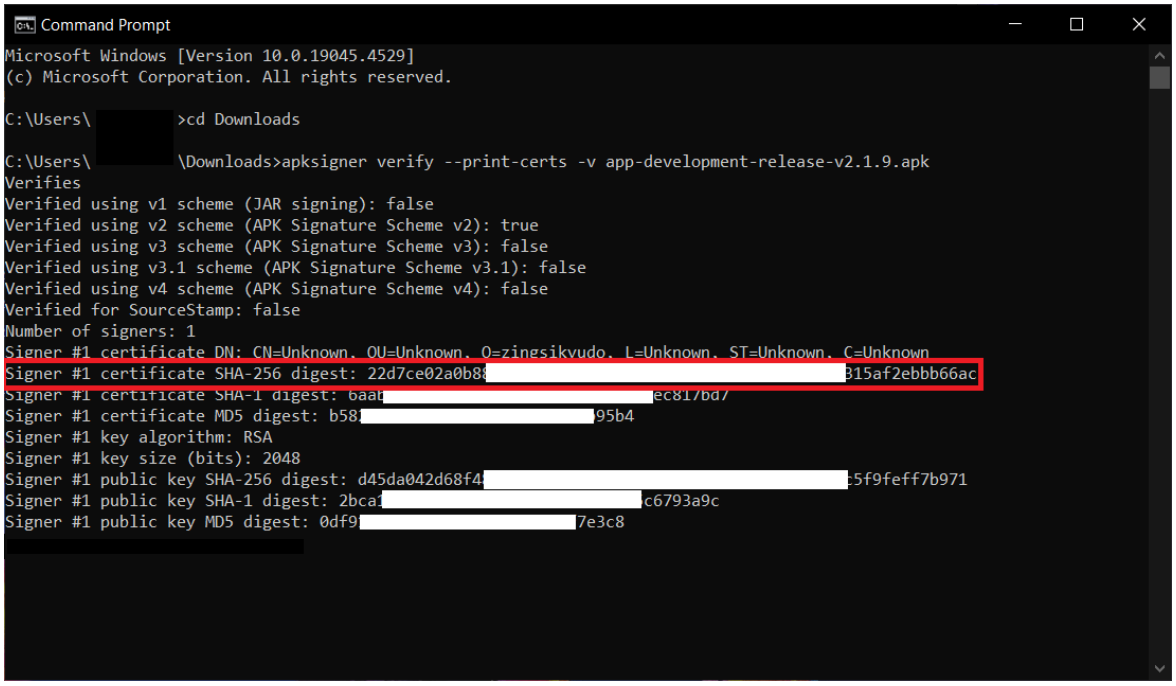
An error message stating "An error occurred, unable to launch the mobile application" will be displayed if otherwise. Information can be gathered from the ESP, the online service app, and the submitted "iAM Smart" catalogue application form to ensure that the above configurations are **aligned and correct**.

Item	Item to check	Retrieve information from online service app
Certificate Fingerprint	ESP	Refer to D.1.1
	APK	
Package name	ESP	Refer to D.1.2
	APK	
	Application form	
Activity class name	APK	Refer to D.1.3
	Application form	

D.1.1 Retrieve signing certificate from online service app

To retrieve the certificate from the online service app, the Android SDK Platform-Tools are required. After installing the Android SDK Platform-Tools, execute the command below to obtain the signing fingerprint of the app. Please note that if different certificates are used for signing the production and development apps, the fingerprints will differ.

```
apksigner verify –print-certs –v <path-to-apk-file>
```

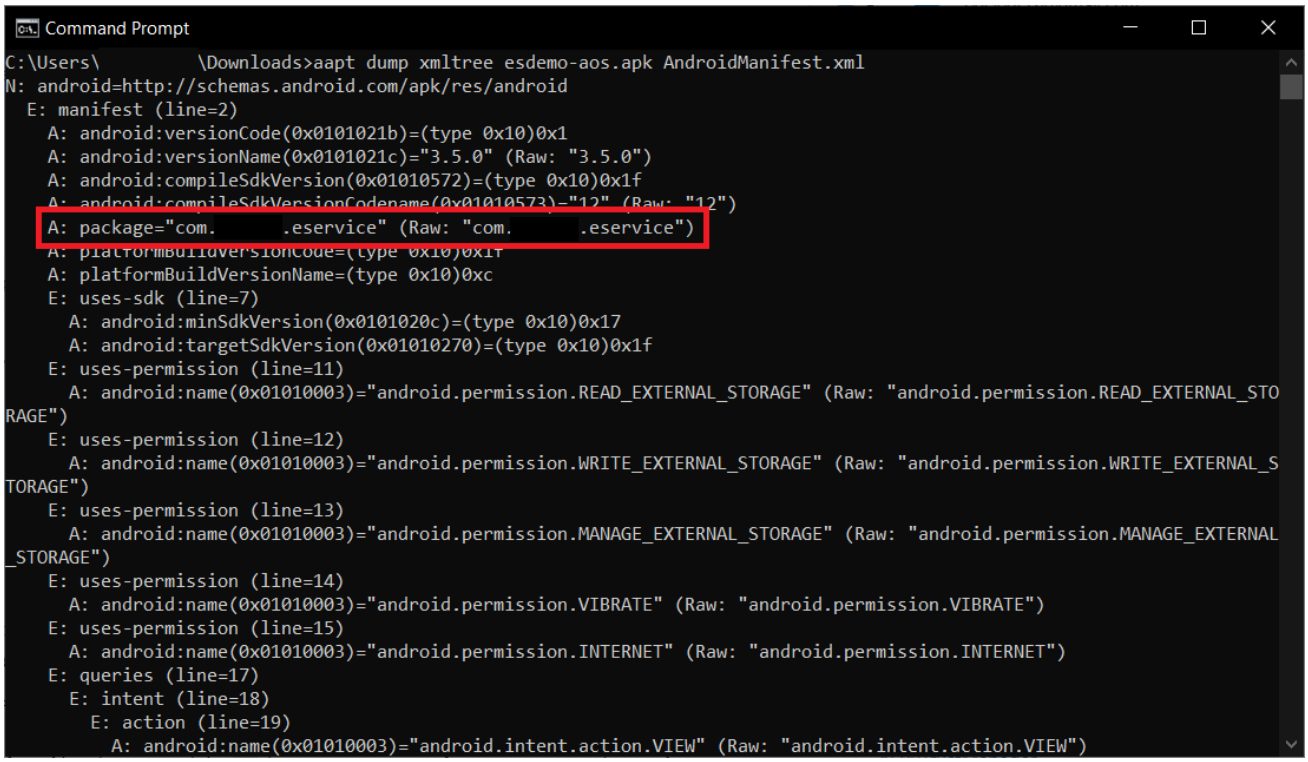


The fingerprint can be found in the Signer certificate SHA-256 digest. Please note that the format may differ from what is required in the ESP. For example, the format in the ESP is `22:D7:CE...66:AC`, whereas it may appear as `22d7ce...66ac` in the terminal.

D.1.2 Retrieve package name from online service app

To retrieve the package name from the online service app, the Android SDK Platform-Tools are required. After installing the Android SDK Platform-Tools, execute the command below to obtain the package name of the app. Please note that the package names for the production and development apps may differ depending on the configuration of the online service app.

```
aapt dump xmltree <path-to-apk-file> AndroidManifest.xml
```



The package name can be found in "`A: package=...`".

D.1.3 Retrieve activity name from online service app

Online service provider should have decided which activity to handle the incoming intent from "iAM Smart". To retrieve the full activity name (e.g. `com.example.dev.MainActivity` instead of `MainActivity`) from the app, the Android SDK Platform-Tools are required. After installing the Android SDK Platform-Tools, execute the command below to obtain the activity name of the app.

```
aapt dump xmltree <path-to-apk-file> AndroidManifest.xml
```

The contents of the Android Manifest file will be displayed in the terminal, including the activity name that handles the intent from "iAM Smart".

D.2 Use correct method to obtain the authCode and code_verifier from Android Intent

Since the iAM Smart has adopted a new approach on Android to pass the authCode, online service app is required to retrieve the authCode and code_verifier from the received intent using the following code.

```
val authCodeId = intent.getStringExtra("code")
val codeVerifier = intent.getStringExtra("code_verifier")
```

```
val activityParams =  intent.getStringExtra("activityParams") //
optional
```

Please note that the code uses `getStringExtra()` method, not the `getData()` method.

For more details, refer to the section 3.10.4.1.