Tech Mobile Connect smartphone app authenticator specification

Version 1.2.1

06 December 2022

This is a Non-binding Permanent Reference Document of the GSMA

Security Classification: Non-confidential

Copyright Notice

Disclaimer

Antitrust Notice

Table of Contents

1.0 Introduction 6
1.1 References 7
1.2 Conventions 7
1.3 Definitions 8
2.0 SAA subsystem overview 9
2.1 Mobile Connect architecture recap 9
2.2 SAA subsystem components 10
2.3 SAA Identifiers 11
3.0 Base SAA functional requirements 13
3.1 SAA Client User Interface features 13
3.2 Authentication modes/LoA support 14
3.3 Offline mode 15
3.4 Discovery of Operator logo 15
3.5 User prompts 16
3.6 SAA Client invocation 18
3.7 Mobile Connect lifecycle events 19
3.7.1 SAA activation (SAA + Mobile Connect) 19
3.7.2 Account recovery 21
3.7.3 SAA Client deletion/reinstallation 21
3.7.4 Device change notification 22
3.7.5 Mobile account status change (suspended/reactivation/deletion) 22
3.7.6 User churn 23
3.7.7 Lifecycle event summary 23
3.8 Interface requirements 24
3.9 Base SAA functional requirements summary 25
4.0 Enhanced SAA functional requirements 28
4.1 SAA Client User Interface features 28
4.2 Discovery of SP logo 28
4.3 User prompts 28
4.4 Mobile Connect lifecycle events 29
4.4.1 User churn 29
4.5 SAA Client local invocation (App deep-linking using custom URI scheme) 30
4.6 Extensible support for new authentication methods 31
4.7 SP binding management 32
4.8 Secured messaging feature 33
4.8.1 Message template 33
4.8.2 Secured messaging API support 33
4.9 Security enhancements 34
4.9.1 Network binding 34

4.9.2 Confidence Score 34
4.10 Enhanced SAA functional requirements summary 35

5.0 Technical solution and implementation guidelines 37

5.1 SAA Interface options 37
5.1.1 Option 1: INT1 and INT2 vendor proprietary 38
5.1.2 Option 2: INT1 standardised within Mobile Connect (Preferred approach) 38
5.1.3 Option 3: INT2 standardised via FIDO UAF 38
5.1.4 Option 4: INT1 standardised within Mobile Connect, INT2 standardised via FIDO UAF 39
5.1.5 Pros and cons of the different SAA interface options 39
5.2 Security requirements & guidelines 41
5.2.1 Device security checks 41
5.2.2 Device secure key-store 42
5.2.3 Public key cryptography for signing a challenge 44
5.2.4 Summary of SAA security requirements 45
5.3 SAA Client activation and association 46
5.3.1 MSISDN discovery in SAA Client 46
5.3.2 Technical flow: SAA Client activation and Mobile Connect registration instigated via SAA Client (MSISDN based pairing) 48
5.3.3 Technical flow: SAA Client activation instigated via an Operator self-care portal (Association code based pairing) 51
5.3.4 Technical flow: SAA Client activation instigated via an Operator self-care portal (MSISDN based pairing) 53
5.3.5 Technical flow: Recovery of existing account using recovery information 54
5.4 Authentication flows 55
5.4.1 Remote invocation (network push; separate consumption device) 55
5.4.2 Remote invocation (network push; mobile browser) 61
5.4.3 Local invocation from SP app (deep-linking using custom URI scheme) 62
5.5 Mobile Connect lifecycle events 71
5.5.1 Lifecycle events integration with Mobile Connect 71
5.5.2 Handling of device change notification 72
5.5.3 Handling of Mobile account status notification 72
5.6 SP binding management 76
5.6.1 Technical flow 76
5.7 Secured messaging flows 78
5.7.1 Message template management 78
5.7.2 Send message 78
5.7.3 Read message 79
5.7.4 Delete message 80
5.8 API summary (Base and Enhanced SAAs) 82

6.0 Deployment considerations 84

6.1 SAA subsystem deployment options 84
6.2 SAA Client SDK 86
6.3 Cost-benefit analysis for Base and Enhanced SAA solutions 87

Annex A Further information 89

A.1 Security threats and prevention techniques 89
A.2 Comparison of SAA vs SIM applet from a Security and Fraud perspective 93
A.3 FIDO-enabled SAA 94
A.4 Future options 94
A.4.1 SP lifecycle notifications 94
A.4.2 Utilising the SIM as a secure key-store 95

Annex B Document Management 96

Table of Figures

Figure 1: Mobile Connect Reference Architecture 9
Figure 2: Identity GW Reference Architecture & SAA subsystem 10
Figure 3: SAA identifiers 13
Figure 4: Sample Mobile Connect branded SAA Client home screen 13
Figure 5: SAA Client prompt requirements 18
Figure 6: SAA interface options 24
Figure 7: SAA interface options 37
Figure 8: INT1 and INT2 are both proprietary interfaces 38
Figure 9: Standardising INT1 38
Figure 10: Standardising INT2 (e.g., FIDO UAF) 39
Figure 11: Standardising both INT1 and INT2 39
Figure 12: SAA Security Measures 41
Figure 13: SAA Client activation and Mobile Connect registration instigated via the SAA Client (MSISDN based pairing) 48
Figure 14: SAA Client activation instigated via Operator self-care portal (Association code based pairing) 51
Figure 15: SAA Client activation instigated via an Operator self-care portal (MSISDN based pairing) 53
Figure 16: Recovery of existing account using account recovery code 54
Figure 17: SAA Client remote invocation technical flow (MSISDN prompt in SP App) 59
Figure 18: SAA Client local invocation component view 63
Figure 19: SAA Client local invocation technical flow (MSISDN prompt in SP App) 66
Figure 20: SAA Client local invocation technical flow (without MSISDN prompt in SP App) 69
Figure 21: Lifecycle events integration with Mobile Connect 72
Figure 22: User's account suspension/reactivation/deletion technical flow 74
Figure 23: SP binding management technical flow 76
Figure 24: Message template management technical flow 78
Figure 25: Send message technical flow 79
Figure 26: Read message technical flow 80
Figure 27: Delete message technical flow 81

Tables

Table 1: SAA identifiers 12
Table 2: Summary of SAA Client UI features 14
Table 3: Authentication mode/LoA support 15
Table 4: SAA Client invocation scenarios 19
Table 5: Lifecycle event summary 23
Table 6: Base SAA functional requirements 27
Table 7: Enhanced SAA Client UI features (delta to Base SAA) 28
Table 8: Confidence Score claim (ID Token) 34
Table 9: Enhanced SAA functional requirements 37
Table 10: Comparison of SAA Interface options 40
Table 11: API summary (Base and Enhanced SAAs) 83
Table 12: SAA subsystem deployment options 85
Table 13: Pros and Cons of SAA subsystem deployment options 86
Table 14: Cost-benefit analysis for Base and Enhanced SAA solutions 88
Table 15: Security threats and prevention techniques 92
Table 16: Comparison of SAA vs SIM applet from a security and fraud perspective 93

1.0 Introduction

Mobile Connect is a portfolio of mobile-based secure identity services delivered by mobile operators, that can be integrated into third-party Service Provider's applications to provide authentication, authorisation, and permissioned access to a User's attributes.

One of the key aspects of the Mobile Connect architecture is its support for "Pluggable Authenticators" such that a range of authenticators can be easily employed to meet different Operator/SP/user needs whilst also ensuring that Mobile Connect is future-proof and can accommodate new authentication mechanisms as they come along (e.g., providing support for advanced biometric authenticators or the inclusion of passive behavioural authentication methods).

With a rapid erosion in the average selling price of smartphones, the number of smartphone connections is predicted to grow threefold over the next six years and reach a 60% tipping point by 2017 (GSMAi). Given this growth, there will be an expectation that Mobile Connect provides authentication solutions that utilise the richer GUI of smartphone devices as well as underlying device features such as gestures and biometric sensors and that such authentication solutions also work over WLAN to support nomadic usage and when the user is out of cellular coverage.

This document therefore provides a set of functional requirements, security and technical implementation guidelines for a Smartphone App Authenticator (SAA) that can be used within the Mobile Connect framework.

Given that there are a number of different approaches that can be taken in the design and deployment of the SAA (e.g., ranging from a simple 'over-the-top' implementation to one more tightly integrated with and using the mobile network) this document identifies two different SAA solution variants:

1. Base SAA — Minimum Viable Product that can be brought to market quickly
2. Enhanced SAA — Tighter integration with the Operator's network to improve user experience and SAA security

The body of the document focuses on the core requirements for the Base SAA; this is then expanded in section 4.0 to reflect the Enhanced SAA.

The document is organised into the following sections:

1. Section 2.0 – Introduction to the SAA subsystem and how it integrates into the existing Mobile Connect architecture
2. Section 3.0 - Functional requirements for the Base SAA including Mobile Connect lifecycle events for the end-end SAA solution including interconnection to and support within the ID GW
3. Section 4.0 – Identification of additional functionality/features that could be incorporated within an Enhanced SAA to improve the user experience and security robustness
4. Section 5.0 – Security guidelines, technical implementation guidelines and technical solutions to the requirements captured in previous sections

5. Section 6.0 – Deployment considerations
6. Annexes – Analysis of key Mobile Connect authenticators, options and considerations for integrating FIDO within the SAA Subsystem to deliver a FIDO-enabled SAA, additional information on security considerations and future options

1.1 References

Ref	Doc Number	Title
[1]	IDY.04	Mobile Connect Technical Architecture and Core Requirements
[2]		OAuth 2.0 for Native Apps[1]
[3]		OWASP Mobile Security Project[2]
[4]		Android Security Tips[3]
[5]		iOS App deep-linking reference[4]
[6]		Android App deep-linking reference[5]
[7]		Push tokens in Android[6]
[8]		Push tokens in iOS[7]
[9]		"Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997
[10]	IDY.16	Mobile Connect Product Manager's Lifecycle Handbook

1.2 Conventions

The key words "must", "must not", "required", "shall", "shall not", "should", "should not", "recommended", "may", and "optional" in this document are to be interpreted as described in RFC2119 [9].

[1] https://tools.ietf.org/html/draft-ietf-oauth-native-apps-00

[2] https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Top_10_Mobile_Risks

[3] http://developer.android.com/training/articles/security-tips.html

[4] https://developer.apple.com/library/IOs/documentation/UIKit/Reference/UIApplicationDelegate_Protocol/Reference/Reference.html

[5] http://developer.android.com/training/app-indexing/deep-linking.html

[6] https://developers.google.com/cloud-messaging/android/client#get-config

[7] https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html

1.3 Definitions

Term	Description
APNS	Apple Push Notification Service – Apple's cloud messaging service supporting push notifications to iOS devices
BSS	Mobile Network Operator's Business Support System i.e., CRM portal etc.,
COTS	Commercial Off The Shelf
FIDO	Fast IDentity Online
FIDO UAF	FIDO's Universal Authentication Framework to support password less experience
FCM	Firebase Cloud Messaging – Google's cloud messaging service supporting push notifications to Android devices
ID GW	Identify Gateway, interfacing between SP and backend authenticator implementations
INT	Interface; to support integration between various system components
IPC	Inter Process Communication
LoA	Level of Assurance
Mobile Connect	Mobile Connect
MCC	Mobile Country Code of the SIM provider
MNC	Mobile Network Code of the SIM provider
MVNO	Mobile Virtual Network Operator
Operator	Mobile Network Operator
OWASP	The Open Web Application Security Project is an online community which creates freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security
PNS	Push Notification Service (e.g., APNS or FCM)
QoS	Quality of Service
SAA	Smartphone App Authenticator
SAA subsystem	Logical entity comprising SAA Adapter, SAA Server and SAA Client
SAA Client	An instance of the SAA application installed on a mobile device
SAA Server	The SAA authentication server provided by the vendor (setup, app lifecycle management, authentication processing etc.)
SAA Adapter	Integration component between Identity Gateway and vendor's SAA Server
SP	Service Provider or Relying Party
TEE	The Trusted Execution Environment (TEE) is a secure area of the main processor of a smart phone (or any connected device including tablets, set-top boxes and televisions). It guarantees code and data loaded inside to be protected with respect to confidentiality and integrity

2.0 SAA subsystem overview

2.1 Mobile Connect architecture recap

Mobile Connect's pluggable approach is achieved through an abstraction layer using a logical component, the Identity Gateway (ID GW) that decouples the interface provided northbound to the Service Provider from the method of authentication used. Taking such an approach allows:

- The SP to indicate the Level of Assurance needed for a particular use case
- The ID GW pluggable arch to support multiple Authenticators and select the most appropriate on a per transaction basis based on the SP's LoA requirements and policy of the Operator

The following diagram depicts the logical architecture of Mobile Connect and illustrates the pluggability of the Identity GW to support multiple Authenticators:

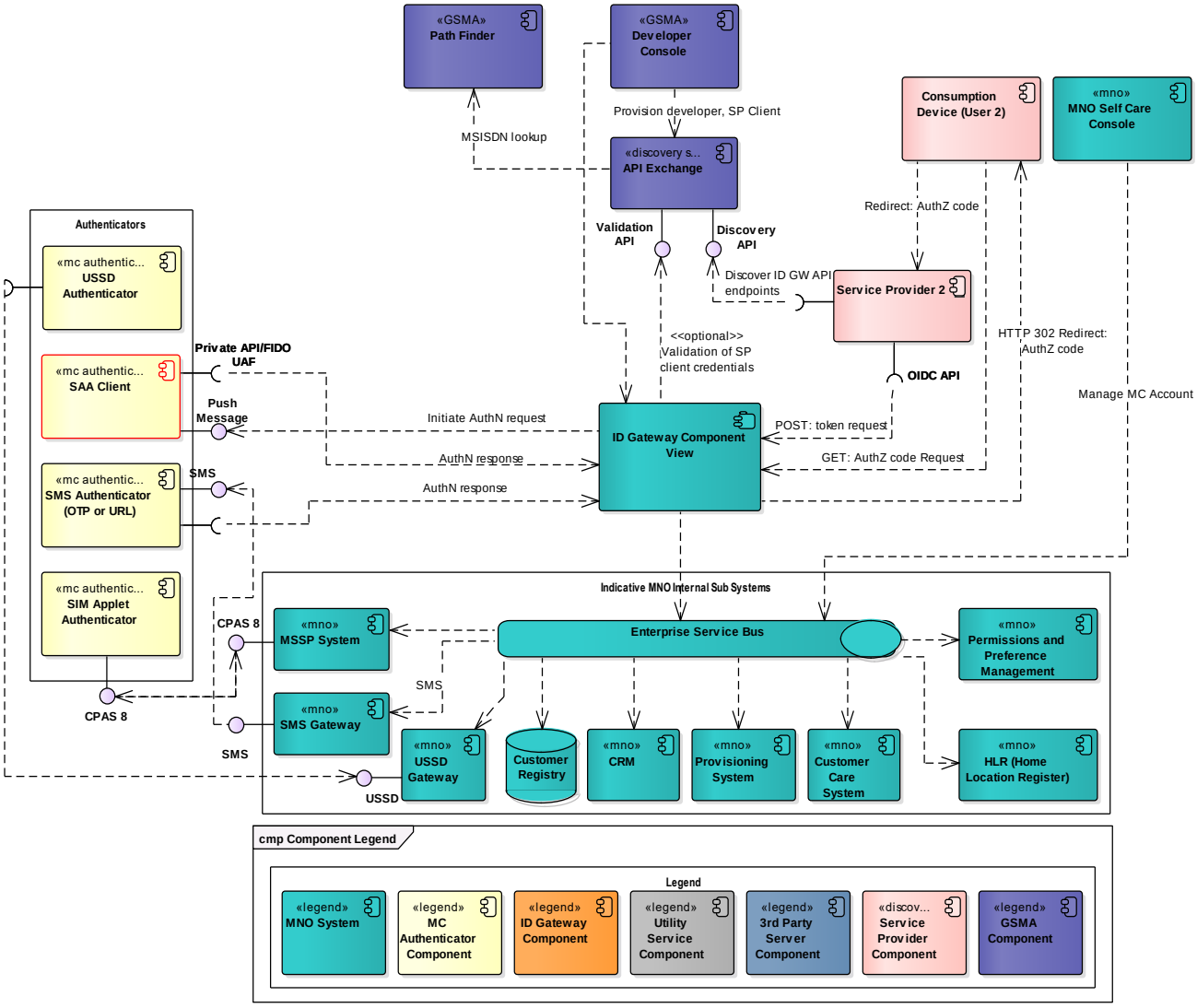

Figure 1: Mobile Connect Reference Architecture

The reference architecture of the Identity GW components is illustrated below, the logical components of the SAA subsystem being highlighted with a RED border:

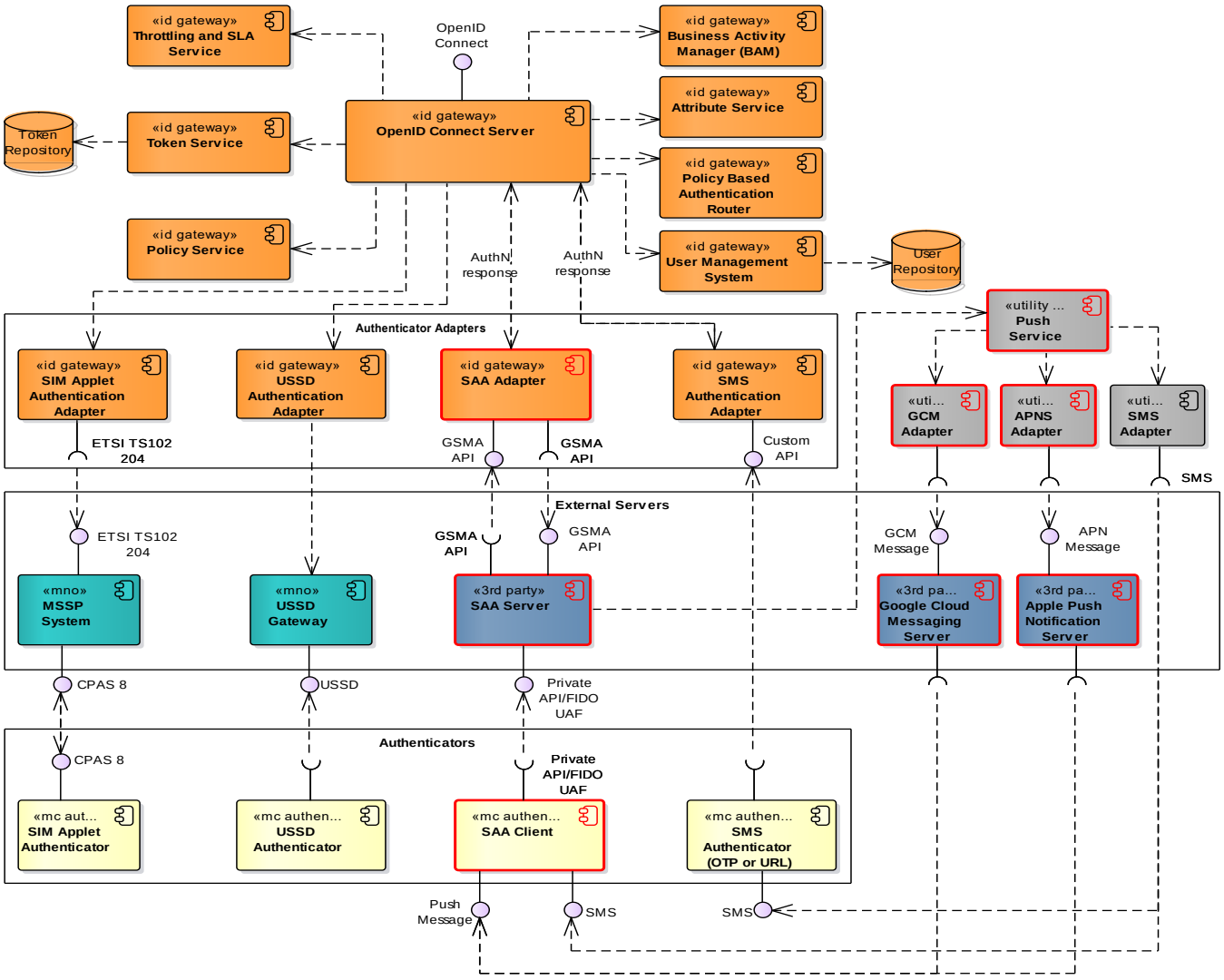

Figure 2: Identity GW Reference Architecture & SAA subsystem

More information on the Identity GW and the overall Mobile Connect architecture can be found in [1].

2.2 SAA subsystem components

The SAA subsystem comprises three components:

- SAA Adapter
- SAA Server
- SAA Client

The SAA subsystem also needs to integrate with a platform-specific push notification service (PNS)[8]. The components are explained in more detail below:

SAA Adapter

The SAA Adapter provides a façade interface between the Identity GW and SAA subsystem in order to support SAA setup, user authentication and any Mobile Connect lifecycle events that impact on the SAA subsystem. The SAA Adapter will be responsible for discovering the MSISDN of the user during the SAA Client activation process.

SAA Server

The SAA Server supports SAA setup, app lifecycle management, authentication/ authorisation processing etc.

SAA Client

The SAA Client is an instance of the Smartphone authenticator app installed on the device.

Push Notification Service

This is a utility service responsible for sending platform-specific push messages (FCM[9] or APNS[10]) for initiating the authentication process of the user via the SAA Client.

Note that the SAA vendor will typically provide the SAA Server and Client; the SAA Adapter will generally be developed by the Identity GW vendor to interface to the API exposed by the SAA Server.

2.3 SAA Identifiers

Assuming a platform-specific PNS is used, the SAA Client is no longer addressed using MSISDN as is the case for most of the other Mobile Connect Authenticators hence an additional identifier is needed for each specific SAA Client instance (SAA Client ID) which is mapped back to the MSISDN identifier (SAA Client ID <-> MSISDN) used within the Mobile Connect system for identifying a particular user (and Mobile Connect account).

Furthermore, in order to increase security, each SAA Client instance should ideally be bound to the underlying device/SIM identifiers on which it is installed. Hence this introduces an additional identifier (SAA Client device ID) and mapping (SAA Client ID <-> SAA Client device ID).

[8] Note: it is possible to invoke the SAA Client via SMS push to a specific port but the PNS approach is much more reliable, assuming the user has a data tariff/connection. If in some markets there is a high proportion of smartphones without data packages then the SMS approach may need to be used.

[9] Firebase Cloud Messaging

[10] Apple Push Notification Service

The various SAA identifiers are defined as follows:

Identifier	Description
MSISDN	Uniquely identifies a Mobile Connect account in the Identity GW
SAA Client ID	Uniquely identifies an SAA Client instance and is generated by the SAA Server after successful SAA Client activation
SAA Client Device ID	Generated by the SAA Client as a SHA-256 hash of device/SIM identifiers[11]. It uniquely identifies the device where the SAA Client app is installed and activated. SAA Client Device ID is compared every time the app starts/resumes with the hash of device/SIM identifiers to detect any device/SIM change and app clone characteristics. Note: **In Android**, access to some telephony information is permission-protected although it is possible to read IMEI and IMSI using Android's TelephonyManager[12] class. **In iOS**, access to telephony information is restricted and is not available to be read using public functions. It is possible to read UUID that is generated by iOS on a per-app basis. As long as the user doesn't completely delete the app then this identifier will persist between app launches hence enabling the user to be identified using a particular app on a device. Unfortunately, if the user completely deletes and then reinstalls the app the UUID will change.
SAA Client push token	Generated by the SAA Client's OS/platform and registered with the platform-specific push server for sending push notification messages to the device. SAA Client receives the registration push token on registering with the platform specific PNS connection servers[13]. The push token is registered with the platform-specific push server through the SAA Server. Note: The push tokens will have to be refreshed with the SAA Server whenever a new registration token is issued to the SAA Client.

Table 1: SAA identifiers

Note: SAA Identifiers need to be treated as personal data and Operator's privacy policies should reflect the various identifiers that will be stored

[11] **For Android/Windows**: Recommendation - use IMSI and IMEI; **For iOS**: Recommendation - use UUID

[12] TelephonyManager - http://developer.android.com/reference/android/telephony/TelephonyManager.html

[13] For retrieving push tokens in Android [7] - https://developers.google.com/cloud-messaging/android/client#get-config; iOS [8] - https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html

The following diagram illustrates the various identifiers and the mapping between them:

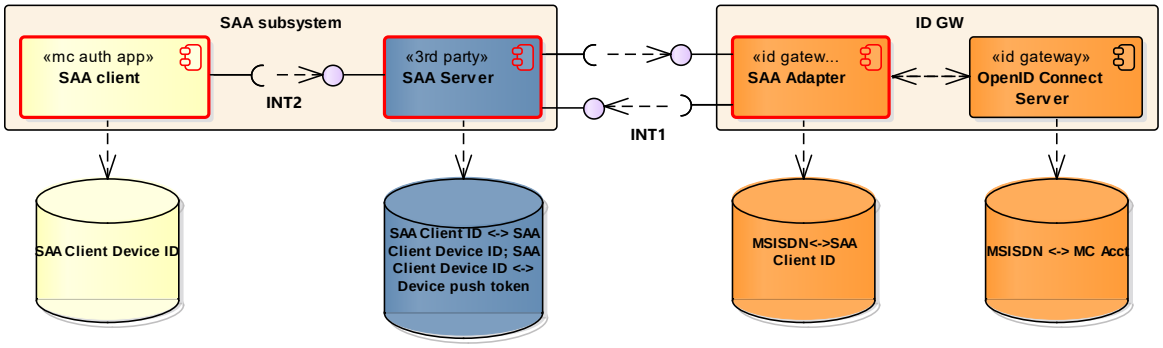

Figure 3: SAA identifiers

3.0 Base SAA functional requirements

This section describes the minimal functional requirements to deliver the Base SAA solution. Section 4.0 then builds on the Base SAA to demonstrate how the Authenticator could be enhanced to improve user experience and security through leveraging Operator network assets.

3.1 SAA Client User Interface features

An example of a Mobile Connect branded SAA Client home screen is shown below:

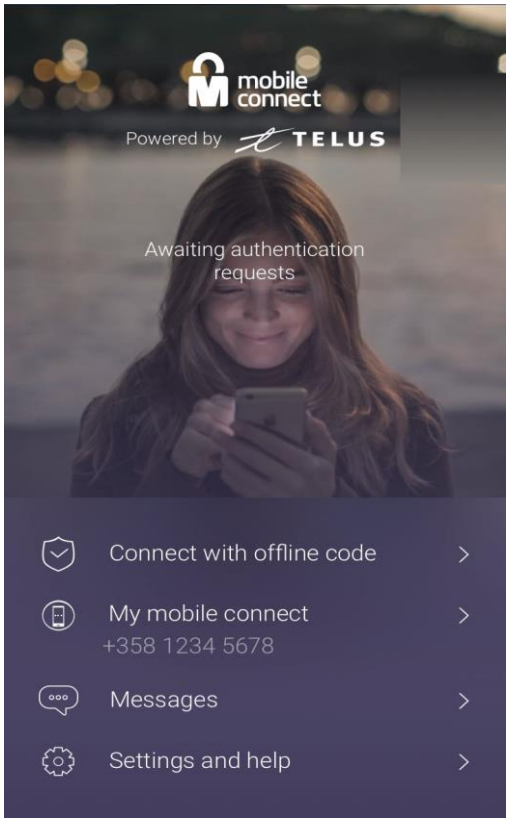

Figure 4: Sample Mobile Connect branded SAA Client home screen

The following table outlines user interface features that should be supported by the SAA Client:

UI Category	UI Feature
Generic	The SAA Client should support localised strings for labels and messages (corresponding to successful and error response codes received from SAA Server)
Generic	The SAA Client should perform device security checks as described in section 5.2.1
Home screen	The SAA Client should display a static Mobile Connect logo on the home screen
Home screen	The SAA Client should display a localised “powered by Operator” on the home screen. Displaying a PNG logo of the Operator brand is highly desirable; options for obtaining the Operator name and logo are covered in section 3.4
Home screen	The SAA Client should display ‘Administration Settings’ and ‘Help’ menu options
Home screen	On opening the app for the first time, the SAA Client should prompt the user to set a local PIN in alignment with Operator policy. This PIN will be SHA-256 hashed and retained on the device and used for both authenticating the user to Mobile Connect as well as granting access to the Administration Settings of the SAA Client
Home screen	On opening the app for the first time, the SAA Client should prompt the user to activate using an association code or recover an existing account using account recovery information
Home Screen	Upon activation, if the MSISDN is determined to be associated to an existing Mobile Connect account, the user should be prompted to recover the account
User prompt screen	The User should be redirected to a prompt screen on receiving authentication, authorisation or attribute sharing consent requests. The SAA Client should support various prompts as described in section 3.5. On completing the transaction request, the user should be redirected back to the screen it was previously on
Administration Settings	Access to the Administration Settings menu should be possible only by authenticating the user against the local PIN registered when the user activated the SAA Client
Administration Settings	It should be possible for the user to manage their PIN (register/change) and on-device biometric[14] profile (e.g., training) where supported
Administration Settings	It should be possible for the user to view account recovery information
Help Menu	Display help contents on ‘Introduction’, ‘How to use?’, ‘Where to use?’, ‘Security tips’, ‘FAQ’, ‘Terms of service’, ‘Privacy policy’. It is expected that the Help information will be generic to Mobile Connect and not curated per Operator

Table 2: Summary of SAA Client UI features

3.2 Authentication modes/LoA support

In alignment with the other Mobile Connect authenticators, the SAA should support both LoA2 and LoA3 although can expand beyond ‘Click OK’ for LoA2 to include Tap, Swipe or other such gestures as supported by smartphone devices and within LoA3 can introduce support for biometrics.

Level of Assurance	Authentication Mode
LoA2	Click OK, Tap or Swipe

LoA3	Enter PIN or use on-device biometric[14]

Table 3: Authentication mode/LoA support

The Base SAA Client can support biometrics using the native Biometrics APIs provided by the underlying platform (Apple TouchID[15], Samsung Pass[16] or Android 6 fingerprint API[17]).

Note that the SAA Server may request a particular authentication mode (e.g., based on SP or Operator policy) or this may be selected in the SAA Client based on user preference (although noting that the Operator/SP policy may need to override any user preference).

Optionally, the SAA may also provide a fall-back OTP mechanism for use when offline – see next section for details.

3.3 Offline mode

In situations where the user is offline for whatever reason or as a fall-back mechanism following a timeout at the ID GW, there is an option for the SAA Client to support the generation of a One-Time Password (OTP) that can be entered manually by the user into the browser of the consumption device for validation by the SAA Server. This is a secured facility and user should be allowed to access One-Time Password generation menu only after successful PIN based local authentication.

The offline code should be generated using a time based mechanism based on pre-synchronized keys/tokens and also be unique to the device (e.g., through using IMEI or MAC address[18]). The pre-synchronized keys/token should have a validity period, be tokenized and be refreshed periodically when the device is online to keep them valid.

Support of an offline mode is optional within the Base SAA but recommended.

3.4 Discovery of Operator logo

There is no standard way of discovering Operator logo metadata dynamically at runtime. Potential options include:

1. Pre-bundling variants of logo resources (mdpi, hdpi, xxhdpi etc.,) of participating Operators in the SAA Client binary and loading them dynamically based on a resource key based on the Operator's MCC + MNC[19]. The SAA Client can read the MCC + MNC of the Operator using platform specific APIs[20]. An application upgrade

[14] Fingerprint, voice recognition, face recognition, iris scan etc.

[15] https://developer.apple.com/library/ios/samplecode/KeychainTouchID/Introduction/Intro.html

[16] http://developer.samsung.com/galaxy#pass

[17] http://developer.android.com/about/versions/marshmallow/android-6.0.html

[18] This information would be provided by the SAA Client to the SAA Server upon initial installation and enrolment of the SAA Client

[19] Mobile Country Code + Mobile Network Code

[20] iOS - https://developer.apple.com/library/ios/documentation/NetworkingInternet/Reference/CTCarrier/index.html; Android - http://developer.android.com/reference/android/telephony/TelephonyManager.html

will be required to load any new participating Operator resources or modify existing resources.

Note however that this functionality cannot be used to support MVNO brands and sub-brands.

2. Pre-bundling variants of logo resources (mdpi, hdpi, xxhdpi etc.,) of participating Operators including MVNOs in the SAA Client binary and loading them dynamically based on a resource key. As part of the SAA Client activation process based on MSISDN, the user's MSISDN is discovered in the SAA Client (see section 3.7.1.1). This MSISDN will be passed to an API exposed by the Operator to discover a key representing the Operator including any MVNO's brand/sub-brand. This key will be used by the SAA Client to load logo resources dynamically. An application upgrade will be required whenever Operator's logo resources needs modification.

3. Instead of pre-bundling logo resources in the SAA Client library, the Operators' resources can be hosted either in the ID GW or SAA Server. These resources can be provisioned in the corresponding server during the Operator on-boarding process. A new logo discovery API exposed by either the ID GW or SAA Server will be responsible for returning a JSON document containing logo metadata including public facing URL's. As part of SAA Client activation process based on MSISDN, the user's MSISDN is discovered in the SAA Client (see section 3.7.1.1) and passed to the logo discovery API to retrieve the logo metadata JSON document, followed by loading the logo resources from the server based on the resource URL. It should be possible for the SAA Client to cache these images to avoid any network latency and improve application performance. No application upgrade will be required if resources needs modification. This approach also allows the ID Gateway to change the branding, links or other personalization on the fly in case something needs to be changed.

3.5 User prompts

Mobile Connect aims to support a range of services encompassing authentication, authorisation and attributes. In doing so it will need to present a range of different prompts to the user via the SAA Client including:

- Authentication prompt - identifying the SP to which a user is being asked to authenticate.
- Authorisation prompt - present the SP-provided details of a transaction that the user is being asked to authorise.
- Attributes prompt - present the Operator-provided attribute labels and associated values that will be shared with an SP, if the user provides their consent.

The SAA Client should support the following requirements for the authorisation prompt:

- SAA Client should support display of SP short name (client_name) to provide context to the user on which service they are authenticating/authorising to[21].

[21] The Service Name will be determined from the SP client_id in the ID GW and passed to the SAA Client.

- In case of Authentication and Authorisation prompts, SAA Client should support display of context value (context) passed by SP to provide context to the user on which service they are authenticating/authorising to[22].
- In case of Attributes prompt, SAA Client should support display of transaction attribute values in the transaction text for seeking user's consent.
- Optionally, SAA Client should support display of a binding message (`binding_message`) if provided; this is a reference number displayed on the consumption device and authorisation device for interlocking purposes.
- The ID GW will communicate various context specific dynamic values (client_name, binding_message, context and transaction attribute values) via the SAA Adapter to the SAA Server. The SAA Server in turn will return these values to the SAA Client for display purpose.
- The authentication mode (Click OK, PIN[23] etc.) will be communicated via the SAA Server; the SAA Client will present the appropriate instruction to the user (e.g., a 'PIN' authentication mode being displayed as 'Enter PIN' on the SAA Client).
- UTF-8 character set must be supported[24].
- SAA Client should provide a cancel option for the user to cancel the transaction.

[22] The Service Name will be determined from the SP client_id in the ID GW and passed to the SAA Client.

[23] SAA Clients are likely to support communication of Authentication mode as follows: Any; Time based One Time Passcode; Tap; Swipe; PIN; Fingerprint scan; Face recognition; Voice recognition

[24] Note that the OIDC AuthZ Request has an optional parameter, ui_locales, which is a space separated list of preferred languages as per RFC5646 but that this parameter is for guidance only and the ID GW can override this without prompting any error.

The following diagram provides an example illustration of the different prompt/text requirements:

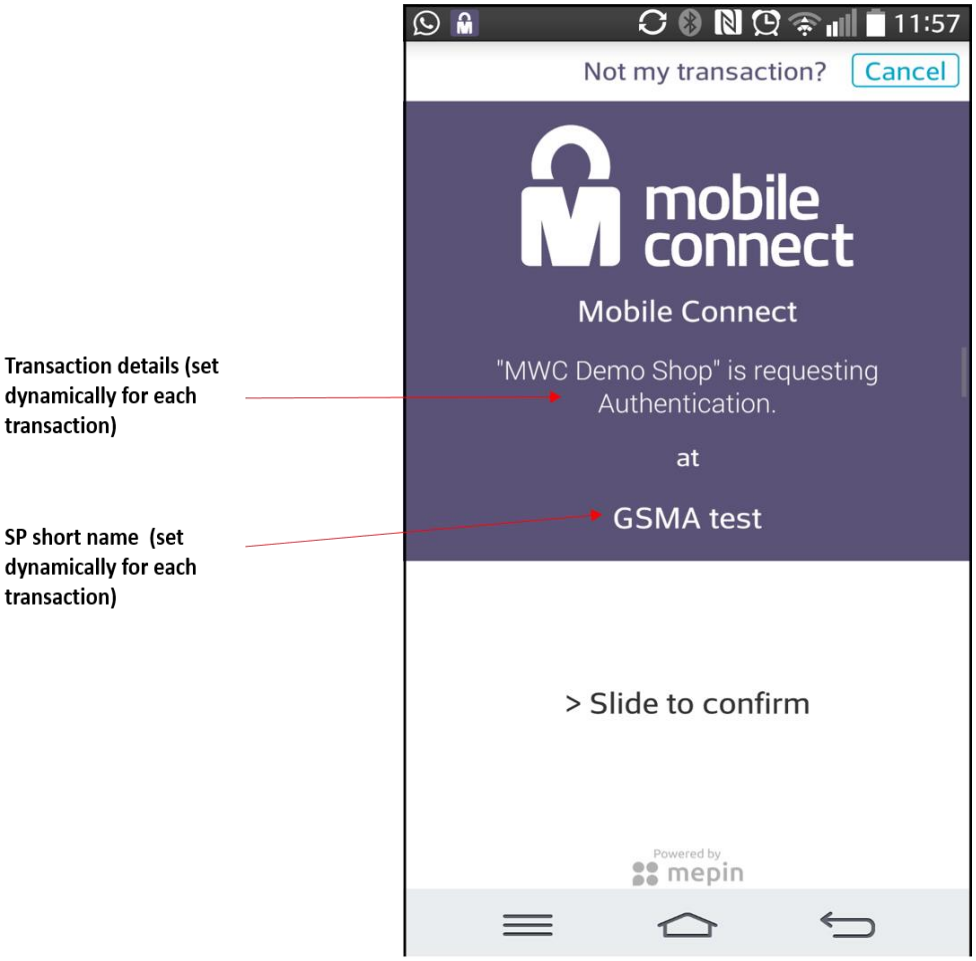

Figure 5: SAA Client prompt requirements

3.6 SAA Client invocation

Mobile Connect takes a 'network initiated' approach in which the user typically receives the authentication challenge on their mobile phone (authentication device) via the network. The same approach should be followed by the SAA authenticator hence the SAA subsystem will need to utilise a platform-specific push notification system such as FCM (Firebase Cloud Messaging) or APNS (Apple Push Notification Service) or alternatively push an SMS to a specific port. Given that the PNS approach results in a better user experience (i.e., reduced latency; ability to use when out of coverage but on WLAN) it is recommended that the Base SAA uses PNS rather than SMS.

In alignment with the existing Mobile Connect authenticators, the SAA should also provide a seamless, intuitive experience to the user when authenticating, and in particular should endeavour to return the smartphone to its prior state after the authentication[25] event has completed. There are three scenarios:

[25] Authentication, Authorisation or Attribute consent interaction

Remote invocation (network push; separate consumption device)	• User initiates authentication via consumption device OR SP service initiates authentication through server-based invocation • SAA Client opens via push notification (with user interaction[26]) and closes or deactivates automatically after completing the authentication process
Remote invocation (network push; mobile browser; same consumption device)	• User initiates authentication via a mobile browser interaction • Mobile browser initiates typical Mobile Connect flow; SAA client invoked remotely by push notification • User will be navigated away from the mobile browser to the SAA Client in order to authenticate • SAA Client closes or deactivates after authentication • Underlying platform should then return focus to the mobile browser so that the user can continue with the SP service
Local invocation from SP app (same consumption device) Enhanced SAA only	• SAA Client opens via platform-specific invocation method initiated by the SP app • After authentication the SAA Client should invoke the custom URI of the SP app to return control back • SAA Client closes or deactivates after returning control • Please see section 4.2 for further reading

Table 4: SAA Client invocation scenarios

Further details on the different invocation scenarios, the accordant technical flows and resultant user experience are covered later in section 5.4.

Note: In case of the device being locked, the user will be required to unlock their device before being able to interact with a Mobile Connect authentication request.

3.7 Mobile Connect lifecycle events

This section provides an overview/guidance on how each of the lifecycle events may impact on the implementation and operation of a Smartphone App Authenticator within Mobile Connect. Section 5.5 then provides a detailed technical walkthrough.

3.7.1 SAA activation (SAA + Mobile Connect)

The activation phase for the SAA solution involves the following steps:

- Download and installation of the SAA Client from the App Store.
- Activation of the SAA Client, setup of an SAA Client account and association with a Mobile Connect user account [optional registration if account not yet available].

The user could potentially discover the SAA Client in a multiple number of ways including:

- Via the App Store.
- SAA Client pre-embedded on the device by the Operator.

[26] Depending on the platform (iOS, Android), there may be a dependency on user interaction in order for the SAA Client to open – this may include unlocking the phone and/or clicking on a notification

- User clicking on a link within the Operator website.
- User clicking on a link within the website of an SP supporting Mobile Connect.
- User clicking on a link received in an SMS pushed to the user by their Operator.
- User clicking on a link within the Operator self-care app.

Note: at this stage, no consideration is being made as to whether the SAA Client exists as a single app on the App Store shared across Operators (and potentially personalised based on Operator) or that there are multiple apps for the user to choose from – distribution options and the pros/cons of each will be considered in more detail in section 6.1.

Once the SAA Client has been downloaded to the device, it needs to be activated[27] and associated with the user's Mobile Connect account (MSISDN), as well as being provisioned to the platform specific push notification service (e.g., APNS or FCM).

The activation can be initiated either by the SAA Client or by an Operator self-care portal but the aim should be to streamline the user interaction flow as far as possible. As part of the activation process the SAA Client ID needs to be linked with the requisite Mobile Connect account via the use of MSISDN or an association code; there are two options:

- MSISDN based linking
- System generated association code based linking

3.7.1.1 MSISDN based linking

- User downloads SAA Client and initiates activation process
- SAA Client discovers MSISDN of the user during activation phase. The MSISDN can be discovered as described in section 5.3.1
- MSISDN is used as the association code for linking SAA Client ID with Mobile Connect account

This is the preferred approach as the user is not required to remember/enter a system generated association code as part of the activation step; however, supporting this approach would require the Operator to expose an API for determining MSISDN or the use of SMS based MSISDN discovery thereby adding additional complexity.

3.7.1.2 System generated association code based linking

- User receives the association code either by email or Operator's website after successful Mobile Connect registration. The association code will be linked to user's Mobile Connect account.
- The association code should be a temporary one-time code with a very short life span (typically 60 seconds).
- It should be possible for the user to request a new association code from the website in case of association code expiry.
- User downloads the SAA Client and initiates the activation process.
- User enters the association code during activation to link SAA Client ID with his/her Mobile Connect account.

[27] The SAA Client activation is a one-time activity that takes place at the start of usage but may need to be repeated periodically (depending on Mobile Connect lifecycle events).

Note: the above assumes that the user is already registered with a Mobile Connect account and is upgrading to the SAA authenticator – further technical details in section (1).

In the situation where the user is not yet registered for Mobile Connect, this would need to be invoked within the SAA Client activation process – further technical details in section 5.3.2.

3.7.2 Account recovery

For consistency, ease of use and security purposes, a single account recovery mechanism should be established whether the SAA Client has been disabled or deleted or whether the user's device has been lost, stolen or changed. In all cases, the previous SAA Client and the Mobile Connect account should be suspended pending activation of a new SAA Client.

As part of the SAA Client activation process, the ID GW (via the SAA Adapter) will generate a unique account recovery code and make it available for the user to view via the Administration Settings menu item (through SAA Adapter->SAA Server-> SAA Client). The recommended size of the recovery code is 16 alphanumeric characters. This is a secured facility and user should be allowed to access this setting only after successful PIN based local authentication.

3.7.3 SAA Client deletion/reinstallation

It'll be a common occurrence that a user will delete the SAA Client (either by mistake or to free up some space on their device) and then want to reinstall the SAA Client later on. There should be security mechanisms to ensure the deletion/ reinstallation sequence cannot be executed quickly and repeatedly i.e. so that it could be attacked by a bot.

The following guidelines are applicable in this scenario:

1. The SAA Client should prompt the user to set up a new PIN and biometric profile (where applicable).
2. If the user has access to account recovery information from a previous SAA activation process, then it should be possible to activate the SAA Client by entering the account recovery information – further technical details in section 5.3.
3. If the user does not have access to account recovery information, the SAA activation process detailed in section 3.7.1 should be followed. It should not be possible for the user to use the previous association code in this case.
4. If a new SAA Client is successfully activated with an existing Mobile Connect account, then:
    i) As all the SP linkages are maintained with MSISDN it will not be necessary to re-establish these linkages in case of reinstallation of SAA Client.
    ii) Previous SAA Client (MSISDN <-> SAA Client ID) related to a particular Mobile Connect account should be deleted or permanently disabled.

3.7.4 Device change notification

A common occurrence is for a user to upgrade their device or swap their SIM from their primary device to a secondary device that is used for a particular purpose (e.g., a smaller phone for use whilst jogging).

Because the SAA Client is linked to the device and only to the MSISDN indirectly via the Mobile Connect account, any authentication requests for that user will always be routed to their primary (or previous) device until the user downloads an instance of the SAA Client into their new/secondary device and associates it with their SAA account.

The Operator can support the requirement of device change status in the following ways:

1. Pushing notifications from the provisioning systems to the ID GW.
2. Providing an internal API that the ID GW can call every time on a transactional basis.
3. Enabling the user to manually update their Mobile Connect account from the Operator's Mobile Connect website (or similar business process).

Either way, the ID GW must notify the SAA Server of any change in the user's device in order for the SAA Server to act accordingly. The handling of device change status notification events is covered in section 5.5.2.

3.7.5 Mobile account status change (suspended/reactivation/deletion)

The user's mobile account may be suspended for a number of different reasons including lost/stolen or unpaid bills. Because the SAA authenticator works over WLAN, suspending the user's mobile account will not prevent them from using Mobile Connect – in fact a user could register to Mobile Connect on a prepay SIM, throw it away and continue using Mobile Connect services.

It is therefore imperative that the ID GW is aware of the standing of the mobile account associated with the MSISDN being used for Mobile Connect and reject the SP authentication request if that MSISDN is no longer valid.

The Operator can support the requirement of account status change in the following ways:

1. Pushing notifications from the Operator's provisioning systems to the ID GW when a relevant lifecycle event occurs. This in turn should modify the status of the corresponding Mobile Connect account accordingly.
2. Provide an internal API that the ID GW can call every time on a transactional basis.
3. Modify the status of a Mobile Connect account directly from the Operator's BSS (e.g., the CRM portal) by calling an internal API in ID GW **(Base requirement).**

Whichever approach is implemented, the ID GW must notify the SAA Server of the change in user account status in order for the SAA Server to act accordingly. The handling of various account status change notification events is covered in section 5.5.3.

Note that the user must also be able to delete their Mobile Connect account (e.g., via the Operator's self-care portal) – this is a generic requirement of the Mobile Connect solution and not specific to SAA.

3.7.6 User churn

As part of the activation phase, the SAA Server assigns an identifier for the user (SAA Client ID) and this identifier is provided to the ID GW. If the user moves to another Operator, the following processes are applicable:

- User will have to de-register their Mobile Connect account with the old Operator resulting in deactivation of the old SAA Client.
- User will have to register for a Mobile Connect account with the new Operator
  - The user registers for a new Mobile Connect account via the new Operator self-care portal and activates the SAA Client of the new Operator.

Risks:

- Service Providers (SPs) may still use old MSISDN in requests[28]. Hence there is a danger that the SP will continue to use the user's old MSISDN allowing whoever receives this recycled number to authenticate to the user's account with the SP (although MSISDNs typically are quarantined for 90 days[29] hence this situation is unlikely). This risk is mitigated by the previous Operator deleting the user's old Mobile Connect account as part of de-registration process. So any request from the SP to the ID GW for an old MSISDN will be rejected with an appropriate error message. However there is an edge case where a user changes Operator and doesn't bother setting up a new account with their new Operator. Hence the recipient of the old device will be able to authenticate the previous owner's SP accounts. This risk is mitigated by use of PIN/biometric gestures for LoA3 transactions.
- SP uses cached endpoints of the old Operator; however, in such a scenario the PCR used by the SP in the OIDC request will no longer be valid, hence the old Operator will reject the request with an error indicating that the SP needs to call Discovery to determine the new Operator endpoints for the target user.

3.7.7 Lifecycle event summary

The following table summarises the lifecycle events:

		Changes				Impacts			
	Use case	Device	MSISDN	MNO	Account	SAA Client ID	SAA Client Device ID	MC a/c	SP
1	User deletes and reinstalls SAA client (app)	No	No	No	No	Yes	No	No	No
2	User swaps SIM to different device (perm or temp)	Yes	No	No	No	Yes	Yes	No	No
3	Mobile account is suspended (unpaid; lost/stolen)	No	No	No	Yes	No	No	Yes	Maybe
4	Mobile account is reactivated	No	No	No	Yes	No	No	Yes	Maybe
5	Mobile account is deleted (e.g., unpaid; inactive)	No	No	No	Yes	No	No	Yes	Maybe
6	User changes MSISDN (same MNO)	No	Yes	No	No	No	No	Yes	Yes
7	User churns; new number, same device	No	Yes	Yes	Yes	Maybe	No	Yes	Yes
8	User churns; new number, new device	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
9	User churns; same number; same device	No	No	Yes	Yes	Maybe	No	Yes	Yes
10	User churns; same number; new device	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes

Table 5: Lifecycle event summary

[28] Trusted SPs are able to stipulate the target MSISDN as a login_hint in their service requests

[29] The period for which numbers are quarantined differs from jurisdiction to jurisdiction

3.8 Interface requirements

The following diagram illustrates the interfaces between:

- SAA Server <-> ID GW (INT1)
- SAA Server <-> SAA Client (INT2)

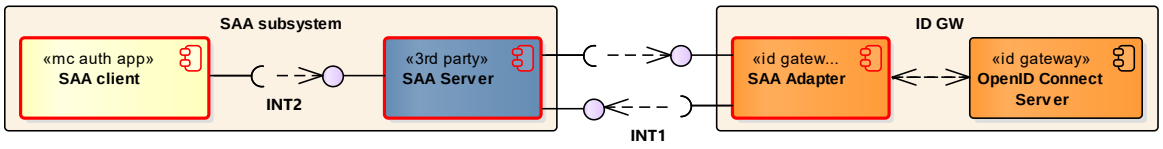

Figure 6: SAA interface options

The following sections identify the requirements for each interface. Detailed discussions of options to meet these requirements are covered in section 5.1

INT1 requirements

- The interface SHOULD be simple, lightweight and extensible.
- The interface SHOULD be RESTful.
- The interface SHOULD use TLS.
- The interface MAY use message layer security, like encrypted payload.
- The parameters passed within INT1 MUST be sufficient for the SAA Server to register the SAA Client, securely identify the specific instance of the SAA Client, send secure messages to the SAA Client etc.
- The interface MUST not restrict the deployment options for the SAA subsystem, e.g. it SHOULD be possible for SAA subsystem components such as the SAA Server to be deployed in the public/private/hybrid Cloud, on premise and also as multi-instance or multi-tenant deployments.
- The interface SHOULD use PKI for integrity.
- Use of Nonce is recommended to ensure the threat of replay attacks is minimised.

INT2 requirements

- The interface MUST use secure communications between the Device and the SAA Server.
- The interface MUST allow the SAA Server to initiate the Authentication challenge to the SAA Client.
- The payload used in the interface SHOULD be minimal, such that the latency of interaction is minimised and optimal bandwidth is used.
- The interface SHOULD use TLS.
- The transport used SHOULD be IP based and transports like SMS, USSD SHOULD be avoided.
- The interface SHOULD use signed messages and PKI for integrity.
- Use of Nonce is recommended to ensure the threat of replay attacks is minimised.

3.9 Base SAA functional requirements summary

The following table summarises the functional requirements for the Base SAA solution:

Requirement	Description
App instance	Recommendation: • Single SAA Client app that is branded Mobile Connect and used by all Operators within a market/region. • Section 6.1 provides more analysis on the different SAA subsystem and SAA Client distribution options
App discovery/ download	Multiple options: • SAA Client pre-embedded on the device by the Operator . • User clicking on a link within the Operator website. • User clicking on a link within the website of an SP supporting Mobile Connect. • User clicking on a link received in an SMS pushed to the user by their Operator. • User clicking on a link within the Operator self-care app
SAA subsystem Identifiers	• As per section 2.3
SAA Client invocation	• SAA Client remote invocation (Network push model) using platform-specific (Android, iOS, Windows) push messaging mechanisms over an IP bearer. • See section 5.4.1 for more details.
Biometrics support	• SAA Client should support biometrics using the native Biometrics APIs provided by the underlying platform (Apple TouchID[30], Samsung Pass[31] or Android 6 fingerprint API[32]).
Authentication modes	• As per section 3.2
User prompt composition & format	• SAA Client should support prompt text on a per transactional basis. • SAA Client should support display of SP short name (client_name) to provide context to the user on which service they are authenticating/authorising to[33]. • In case of Authentication and Authorisation prompts, SAA Client should support display of context value (context) passed by SP to provide context to the user on which service they are authenticating/authorising to[34]. • In case of Attributes prompt, SAA Client should support display of transaction attribute values in the transaction text for seeking user's consent.

[30] https://developer.apple.com/library/ios/samplecode/KeychainTouchID/Introduction/Intro.html

[31] http://developer.samsung.com/galaxy#pass

[32] http://developer.android.com/about/versions/marshmallow/android-6.0.html

[33] The Service Name will be determined from the SP client_id in the ID GW and passed to the SAA Client.

[34] The Service Name will be determined from the SP client_id in the ID GW and passed to the SAA Client.

	• Optionally, SAA Client should support display of binding message (binding_message), a reference number on consumption device and authorisation device for interlocking purposes. • The ID GW will communicate various context specific dynamic values (client_name, binding_message, context and transaction attribute values) via the SAA Adapter to the SAA Server. The SAA Server in turn will return these values to SAA Client for display purpose. • The authentication mode (Click OK, PIN[35] etc.) will be communicated via the SAA Server; the SAA Client will present the appropriate instruction to the user (e.g., a 'PIN' authentication mode being displayed as 'Enter PIN' on the SAA Client). • UTF-8 character set must be supported[36]. • SAA Client should provide a cancel option for the user to cancel the transaction.
SAA Interfaces	• INT1 (ID GW <-> SAA Server) vendor specific (ID GW develops SAA Adapter as required). • INT2 (SAA Server <-> SAA Client) vendor specific. • See section 5.1 for more details.
Security Guidelines	• As per section 5.2
Lifecycle: SAA Client activation	• User receives the association code either by email or the Operator's Mobile Connect or self-care website after successful Mobile Connect registration. • The association code should be a temporary one-time code with a very short life span (typically 60 seconds). • It should be possible for the user to request a new association code from the Operator's website in case of association code expiry. • User downloads SAA Client and initiates activation process. • User enters the association code during activation to link SAA Client ID with his/her Mobile Connect account. • See section 3.7.1.2 for more details.
Lifecycle: Account recovery	• On successful activation, the ID GW (SAA Adapter) must generate a 16 character account recovery code and make it available for the user to view via the SAA Client (Administration Settings menu). • It should be possible for the user to recover an existing account using the account recovery code. • See section 5.3.5 for more details.
Lifecycle: SAA Client deletion/reinstallation	• SAA Client should prompt the user to set up new PIN and also allow the user to set up a biometric profile (where appropriate). • If the user has access to account recovery information from a previous SAA activation process, it should be possible to activate the SAA Client

[35] SAA Clients are likely to support communication of Authentication mode as follows: Any; Time based One Time Passcode; Tap; Swipe; PIN; Fingerprint scan; Face recognition; Voice recognition

[36] Note that the OIDC AuthZ Request has an optional parameter, ui_locales, which is a space separated list of preferred languages as per RFC5646 but that this parameter is for guidance only and the ID GW can override this without prompting any error.

	by entering the account recovery information – further technical details in section (1). • If the user does not have access to account recovery information, then the SAA activation process detailed in section 3.7.1 should be followed. It should not be possible for the user to use a previous association code in this case. • As all the SP linkages are maintained with MSISDN, it will not be necessary to re-establish these linkages in case of reinstallation of the SAA Client. • All previous SAA Clients (MSISDN <-> SAA Client ID) related to a particular Mobile Connect account should be deleted or permanently disabled.
Lifecycle: Device change	• The user must be able to manually update their Mobile Connect account from the Operator's Mobile Connect website (or similar business process) when switching to a new device OR the Operator BSS systems must advise the ID GW of a device change. • In doing so, the existing SAA Client account should be blocked and the user prompted to download, install and activate an SAA Client instance on their new device. • The ID GW must notify the SAA Server of device change status for the SAA Server to act accordingly. • See section 5.5.2 for more details.
Lifecycle: Account status notifications	• The Operator must have the ability to modify the status of the Mobile Connect account from the Operator's BSS (e.g., the CRM portal). • The ID GW must check for lifecycle events and ensure the mobile account is in good standing before issuing an authentication/authorisation/consent request to the SAA Server. • The ID GW must notify the SAA Server of any account status change for the SAA Server to act accordingly. • The User must be able to delete their Mobile Connect account via the Operator's self-care portal. • See section 5.5.3 for more details.
Lifecycle: User churn	• User de-registers their Mobile Connect account with the old Operator resulting in deactivation of old SAA Client. • User activates the SAA Client with the SAA Server of the new Operator resulting in a new SAA Client ID.

Table 6: Base SAA functional requirements

4.0 Enhanced SAA functional requirements

One of the key principles of Mobile Connect is to use/re-use Operator assets, business processes and security mechanisms to provide a better overall authentication solution.

In many of the other Mobile Connect authenticators, network assets are implicitly used to deliver the authentication mechanism (e.g. USSD, Class 2 SMS etc.) hence providing an implicit Operator value-add. In the case of the SAA, an IP-channel is more suitable for connectivity but there are other ways in which network assets can be used to enhance the solution. In particular, the SAA can be bound to Operator identifiers thereby reusing Operator assets such as network authentication, SIM and Device identity association and knowledge.

This section describes an Enhanced SAA solution that leverages Operator APIs and network assets as well as introducing additional functionality to deliver a more robust/feature rich proposition. Note that the Enhanced SAA solution places a number of dependencies on both GSMA platforms (e.g., API Exchange) and Operator systems hence is not expected to be ready for deployment until a later Mobile Connect Release.

4.1 SAA Client User Interface features

The following table outlines user interface features that should be supported by the Enhanced SAA Client in addition to those required for the Base SAA as defined in section 3.1:

UI Category	UI Feature
Administration Settings	The SAA Client should allow a user to retrieve a list of SP bindings, and be able to request revoking or removing them from the ID GW
Administration Settings	The SAA Client should provide an inbox capability allowing a user to view a list of messages, read a specific message and delete a message

Table 7: Enhanced SAA Client UI features (delta to Base SAA)

4.2 Discovery of SP logo

The ID GW could retrieve additional SP details (SP logo URL, short name, SP background image URL) by calling the Request Validation API of the API Exchange using SP client_id as the parameter. This call could be made during the authorisation code request call from the SP client. To support variants of SP logo resources (mdpi, hdpi, xxhdpi etc.,), it is recommended that the Request Validation API return SP logo metadata as JSON document.

Note however that this functionality is not yet supported within the API Exchange so would require additional development hence is for further study.

4.3 User prompts

In addition to the user prompts requirements described in Base SAA (section 3.5), the additional requirements are:

- SAA Client should support display of SP logo.
- SAA Client should support display of SP background logo.
- The ID GW will communicate various context specific dynamic values (client_name, binding_message, context, transaction attribute values, SP logo URL, SP background image URL etc.,) via the SAA Adapter to the SAA Server. The SAA Server in turn will return these values to SAA Client for display purpose.

4.4 Mobile Connect lifecycle events

4.4.1 User churn

As part of the activation phase, the SAA Server assigns an identifier for the user (SAA Client ID) and this identifier is provided to the ID GW. If the user moves to another Operator, this SAA Client ID and device related information (SAA Client Device ID, Device Push Token) may need to be passed to the new Operator through the Lifecycle Account Migration API (e.g., in addition to the passing of the PCR mappings) to ensure that any authentication requests for that MSISDN are routed to the correct instance of the SAA Client (and not an SAA Client remaining on the user's previous handset).

However, it depends on the individual circumstances of the user and is applicable only if the user keeps their existing device. This is discussed further in the following subsection.

Note however that this functionality is not yet supported within the Lifecycle Account Migration API so would require additional development hence is for further study.

4.4.1.1 User keeps their existing device

Assuming that the user retains their existing SAA Client on their device, the process is likely to work as follows:

- If the Operators (ported from and ported to) are using an SAA subsystem from the same vendor[37], the user can keep their existing SAA Client instance and associate this with the Mobile Connect account as part of the Account Migration process. The SAA Client ID, Device Push Token, Device OS type and SAA Client Device ID are ported along with the PCRs as part of the Account Migration process. The Operator stores the SAA Client ID along with the MSISDN in the SAA Adapter and the SAA Client Device ID, Device Push Token and Device OS type within the SAA Server along with SAA Client ID. This is the recommended option for operator's supporting account migration process.

  Note that the Device Push Token is tied to the SAA Client and will need to be refreshed even if the SAA Client is reinstalled. The SAA Server should replace the existing Device Push Token with a new Device Push Token when supplied by the SAA Client.

Risks:

[37] Or if the interface between SAA Client and SAA Server (i.e., INT2) is standardised to allow for any SAA Client to interwork with any SAA Server; for more information on interface approaches see section 5.1

- Service Providers may still use an old MSISDN in requests[38]. Hence there is a danger that the SP will continue to use the user's old MSISDN allowing whoever receives this recycled number to authenticate to the user's account with the SP (although MSISDNs typically are quarantined for 90 days[39] hence this situation is unlikely). This risk is mitigated by the previous Operator deleting the user's old Mobile Connect account (which should happen when the user ports across their old Mobile Connect account to the new Operator - see section 5.5.3 for further details). So any request from an SP to the ID GW for an old MSISDN will be rejected with an appropriate error message. However there is an edge case where a user changes Operator and doesn't bother setting up a new account with their new Operator. Hence the recipient of the old device will be able to authenticate the previous owner's SP accounts. This risk is mitigated by use of PIN/biometric gestures for LoA3 transactions.
- SP uses cached endpoints of the old Operator; however, in such a scenario the PCR used by the SP in the OIDC request will no longer be valid, hence the old Operator will reject the request with an error indicating that the SP needs to call Discovery to determine the new Operator endpoints for the target user.

4.4.1.2 User upgrades to a new device

User churns and sets up a new Mobile Connect account with the new Operator. Similarly, as per the previous scenario, the SAA Client ID could be ported across with the PCRs or a new SAA Client ID generated. The SAA Client Device ID and Device Push Token though will change as in this scenario the user has upgraded to a new device.

Risks:

- Service Providers may still use an old MSISDN in service requests - mitigation is as described in the previous section
- SP uses cached endpoints of the old Operator; mitigation as per previous section

4.5 SAA Client local invocation (App deep-linking using custom URI scheme)

If the user is accessing the SP's service via an SP App on the mobile phone, the authentication flow can be optimised by the SP App invoking the SAA Client locally. The high level flow will be as follows:

1. SP App initiates OIDC authorization code request with Operator's ID GW passing relevant parameters (SP client_id, state, context, prompt, login_hint etc.,).
2. In parallel, SP App invokes the custom URI of SAA Client passing SP App's call-back URL, SP client_id and state parameters. This action activates SAA Client and pushes SP App into background.
3. ID GW processes the OIDC request, creates a new transaction record for the combination of user's MSISDN, SP client_id and state parameters and suppresses authentication challenge depending on the new value passed in **prompt** parameter.

[38] Trusted SPs are able to stipulate the target MSISDN as a login_hint in their service requests

[39] The period for which numbers are quarantined differs from jurisdiction to jurisdiction

*Note: A change will be required in the OIDC Mobile Connect profile to pass a new value in **prompt** parameter that will inform ID GW to suspend network push for initiating user's authentication process. The actual value to be passed will be finalised after further study.*

4. SAA Client initiates AuthN session with SAA Server passing SP client_id, state and other relevant parameters.
5. SAA Server retrieves transaction details from ID GW corresponding to user's MSISDN, SP client_id and state parameters created in step 3.
6. SAA Client prompts the user based on AuthN request response received from SAA Server.
7. User authenticates via SAA Client depending on level of assurance requested by SP and Operator's policy.
8. SAA Client returns control back to the SP App by invoking the callback URL of the SP App.
9. In the background, ID GW responds to original OIDC request with authorization code in the redirect URL as a query parameter.

The SP App can discover the SAA Client custom URI either:

1. Dynamically, reading Operator's MCC + MNC using platform specific functions and calling the API in the API Exchange to retrieve Operator specific provider metadata containing custom URL of SAA Client. This is the recommended approach.

   Note: A change would be required in the API Exchange to support retrieval of Operator specific provider metadata.
2. Static links published in mobileconnect.io (for each Operator's SAA Client) or similar GSMA public facing portal. See section 5.4.3 for technical details on SAA Client local invocation.

For Custom URI Scheme Namespace Considerations, please see [2] (section 4.1.2).

4.6 Extensible support for new authentication methods

The SAA solution may support a range of authentication approaches including knowledge-based (e.g., PIN) and biometrics. Going forward, the SAA solution should be extensible to support new authentication methods as they become available on the smartphone platform – this may include new capabilities intrinsic to the platform itself or new software-based authenticators that can be incorporated into the SAA implementation (e.g., PixelPIN[40], face recognition, iris scanning etc.).

In doing so, the SAA Client should enable the user to choose which authentication method they prefer for a given LoA – for instance, choosing between PIN vs fingerprint. This choice should be stored locally but communicated to the ID GW (through SAA Server->SAA Adapter) such that the authentication context parameter (amr) can be correctly specified in the OIDC response to the SP.

[40] http://pixelpin.co.uk

Note that in future phases it may be necessary for the SAA to adapt based on SP requirements and override a user preference – for instance, if the SP demands a biometric authentication for LoA3 rather than a PIN. In such a scenario, the SAA Server and SAA Client can do a handshake to exchange the allowed list of authentication modes based on the LoA requested and configured policies, which can then be presented to the user for selecting their preference.

Note that in any case, ID GW should retain control over authentication methods based on Operator's policy.

4.7 SP binding management

The SAA Client should provide an interface through which the user can:

- Review the list of Service Providers that the user has linked their Mobile Connect account to
- Remove one or more of the SPs from their Mobile Connect account
- Request for revocation of SP specific Access and ID tokens

Please note that this is a secured interface and should only be accessible after user has authenticated locally with SAA Client using the PIN.

There are two scenarios:

- **SP binding removal:** User wishes to remove an SP so that their Mobile Connect account can no longer be used for authenticating to that SP, results in:
  - ID Gateway shall no longer allow the specific SP to use the Mobile Connect service to authenticate the user.
  - At an ID GW level, this will lead to the PCR for that particular *user:SP* pairing being removed from the user's Mobile Client account.
  - User wishing to reinstate the Mobile Connect <-> SP binding will be required to re-authenticate with that particular SP.
- **SP binding revocation:** User wishes to revoke an SP to, in effect, 'log out' results in:
  - Revocation of the Access and ID Tokens for that *user:SP* pairing.
  - The SP will not know this until either a refresh token call is made (or) a new Authorisation call is made.

If an SP is removed or disabled by the user via the SAA Client, the ID Gateway shall no longer allow the specific SP to use the Mobile Connect service to authenticate the user.

The Identity GW should provide APIs through which the SAA Client can:

- Request a list of SP bindings
- Request that an SP binding is removed
- Request that an SP binding is revoked

4.8 Secured messaging feature

Secured messaging is a feature to deliver notifications (information, offers, reminders etc.,) from the SPs or Operators to Mobile Connect registered users on SAA Client. With secured messaging, it is possible to further engage with users while avoiding the spam and phishing problems of email.

The SAA Client should provide a message inbox interface through which the user can:

- Review the list of messages in inbox.
- Read the details of a message. The message must be marked as **'READ'** in SAA Server**.**
- Delete a message from inbox that in-turn will either mark the message as **'DELETED'** or physically delete the message from SAA Server.

If an SP is removed or disabled, the ID GW shall no longer allow the specific SP to send messages to users.

Please note that message inbox is a secured interface and should only be accessible after user has authenticated locally with SAA Client using the PIN.

4.8.1 Message template

A message template defines the structure of a message to be delivered to end users on the SAA Client. It should support keyword placeholders surrounded by double curly braces {{}} that can be replaced by actual text at runtime in the message.

For example: If the message template is:

```
This is a {{message}}
```

At runtime, SP provides parameter:

```
Message = simple text message
```

The actual message delivered to the user will be:

```
This is a simple text message
```

Additionally, a message template can also include title and URL of icon to be displayed in the message. The URL should be a publicly resolvable URL that will load an image resource at runtime when the message is rendered on the SAA Client.

SP will be responsible for managing their message templates using the portal provided by the Operator or through private APIs exposed by the ID GW. Please see section 4.8.2 for secured messaging API support.

Please note that the mechanisms of message template management is beyond the scope of this document.

4.8.2 Secured messaging API support

It should be possible for SPs to manage message templates and send messages to multiple recipients (Mobile Connect registered users) using the private APIs exposed by ID GW. The ID GW will be responsible for delegating the API request from SP to SAA Server for fulfilment.

Please note that the actual API specification in ID GW for providing secured messaging feature is beyond the scope of this document and is left to ID GW vendors to define.

Likewise, the SAA Server will expose messaging APIs to allow SPs to manage message templates and send messages to users inbox on SAA Client. Please see later for the secured messaging API definition in the SAA Server.

The interaction between SP and ID GW components for secured messaging support will be as follows:

SP ---> ID GW ---> SAA Server ---> SAA Client

4.9 Security enhancements

4.9.1 Network binding

In the case of the Base SAA, mitigation against app cloning risks is achieved by comparing the device token (SAA Client Device ID) stored in the device secured key-store with the device/SIM identifiers read from the device. This check can be further strengthened by utilising an Operator network API to fetch the registered device/SIM identifiers from the BSS/OSS systems for the MSISDN and compare with the values read from the device. The additional network check ensures that the device token is bound to network validated device/SIM identifiers and provides a strong mitigation strategy against device tampering and app cloning risks.

The potential technical solution for SAA activation using network binding concepts is covered in more detail in section 5.3.2.

4.9.2 Confidence Score

Creating a Confidence Score based on attributes such as location, device change time, SIM change time and frequency, length of contract, network presence, type of tariff etc. can provide an additional factor “Something the Network knows” to enhance the authentication service.

The Confidence Score would be calculated by the ID GW and passed back to the SP within the OIDC response using an optional claim in the ID Token:

Parameter	Description
auth_conf_score	The Confidence Score for the authentication based on additional context known to the Operator The auth_conf_score value MUST be in the format – X:Y, where X is the score out of Y

Table 8: Confidence Score claim (ID Token)

Note: where the MSISDN verification check provides explicit information that the user account and/or device are unreliable, the authentication could be immediately declined without the user being asked to authenticate (dependent on Operator/SP policy).

In the approach being described above, the user authenticates but the response is accompanied with a Confidence Score – in this situation, the Operator has no explicit knowledge that the authentication should be declined, but can provide input parameters (amr parameter in OIDC response) to an SP's risk engine. In both cases, it is ultimately the SP's decision on how to act on the authentication response.

Note that this approach of providing a confidence score is not exclusive to the SAA authenticator but could be used to enhance the Mobile Connect authentication product in general; however, in the case of the SAA authenticator where there may not be a direct value-add from the mobile network, being able to combine the rich UI and flexibility of an SAA authenticator with a Confidence Score could be important for delivering a differentiated product.

Note however, that introduction of a Confidence Score would require an extension to the OIDC Mobile Connect profile to accommodate the additional claim value within the ID Token hence is for further study.

4.10 Enhanced SAA functional requirements summary

The following table summarises the incremental functional requirements for the SAA subsystem and ID GW for the Enhanced SAA:

Item	Requirement
App instance	Recommended: • Single app that is branded Mobile Connect and used by all MNOs but using an MCC+MNC lookup and logo discovery mechanisms (see section 3.4) to incorporate the Operator logo • [optionally] SAA vendor SDK for Operator app • Section 6.1 provides more analysis on the different SAA subsystem and SAA Client distribution options
App discovery/ download	• Same as for the Base SAA solution (see section 3.9)
Identifiers	• Same as for the Base SAA solution (see section 3.9)
SAA Client invocation	• Same as for the Base SAA solution (see section 3.9) • [optional] SAA Client local invocation (app deep-linking method); see section 4.2
Authentication modes	• Same as for the Base SAA solution (see section 3.9)
User prompt composition & format	• SAA Client should support prompt text on a per transactional basis. • SAA Client should support display of SP short name (client_name) to provide context to the user on which service they are authenticating/authorising to[41]. • In case of Authentication and Authorisation prompts, SAA Client should support display of context value passed by SP to provide context to the user on which service they are authenticating/authorising to.

[41] The Service Name will be determined from the SP client_id in the ID GW and passed to the SAA Client.

	• In case of Attributes prompt, SAA Client should support display of transaction attribute values in the transaction text for seeking user's consent. • Optionally, SAA Client should support display of binding message (binding_message), a reference number on consumption device and authorisation device for interlocking purposes. • SAA Client should support display of SP logo. • SAA Client should support display of SP background logo. • The ID GW will communicate various context specific dynamic values (client_name, binding_message, context, transaction attribute values, SP logo URL, SP background image URL etc.,) via the SAA Adapter to the SAA Server. The SAA Server in turn will return these values to SAA Client for display purpose. • The authentication mode (Click OK, PIN[42] etc.) will be communicated via the SAA Server; the SAA Client will present the appropriate instruction to the user (e.g., a 'PIN' authentication mode being displayed as 'Enter PIN' on the SAA Client). • UTF-8 character set must be supported[43]. • SAA Client should provide a user option to cancel the transaction.
SAA Interfaces	• Common set of principles and API specifications for INT1 • See section 5.1 for more details

Security	• Same as for Base SAA solution plus:
	  • Network signature binding (binding the device signature with some server side verifiable information) as per section 4.9.1
	  • [optional] Confidence score calculated at the ID GW and passed back to the SP via the OIDC response[44]

Lifecycle: SAA Client activation	• Same as for the Base SAA solution (see section 3.9) • User instigates SAA Client activation by entering MSISDN on SAA Client (see section 3.7.1.1)
Lifecycle: Account recovery	• Same as for the Base SAA solution (see section 3.9)
Lifecycle: SAA Client deletion/reinstallation	• Same as for the Base SAA solution (see section 3.9)
Lifecycle: Device change	• Same as for the Base SAA solution (see section 3.9)
Lifecycle: Account status	• Operator must implement mechanisms to either check (poll) or push relevant lifecycle events to the ID GW • ID GW must check for lifecycle events and ensure the mobile account is in good standing before issuing authentication request to SAA Server

[42] SAA Clients are likely to support communication of Authentication mode as follows: Any; Time based One Time Passcode; Tap; Swipe; PIN; Fingerprint scan; Face recognition; Voice recognition

[43] Note that the OIDC AuthZ Request has an optional parameter, ui_locales, which is a space separated list of preferred languages as per RFC5646 but that this parameter is for guidance only and the ID GW can override this without prompting any error.

[44] Note that this capability could be introduced in general for the Mobile Connect Authenticate products and is not unique to the SAA solution

Lifecycle: Account suspension/ reactivation/deletion	• Same as for the Base SAA solution (see section 3.9)
Lifecycle: SP binding management	• Review the list of Service Providers that the user has linked their Mobile Connect account to • Remove one or more of the SPs from their Mobile Connect account • Request that an authentication is revoked
Lifecycle: User churn	• Same as for the Base SAA solution (see section 3.9) • The SAA Client ID may need to be passed to the new Operator through the Lifecycle Account Migration API (e.g., in addition to the passing of the PCR mappings) (see section 4.4.1)
Secured messaging feature	• Allow the SP to define and manage message templates (see section 4.8.1) • Allow the SP to send message to multiple end users (see section 4.8.2) • Allow the user to review the list of messages (see section 4.8) • Allow the user to read details of a message (see section 4.8) • Allow the user to delete a message (see section 4.8)

Table 9: Enhanced SAA functional requirements

5.0 Technical solution and implementation guidelines

This section provides technical solutions and implementation guidelines for realisation of the various functional requirements outlined in previous sections for the Base and Enhanced SAA solutions. Please note that these technical flows are provided for guideline purposes only and the actual implementation may differ from the proposed solutions.

5.1 SAA Interface options

Section 3.8 identified the core requirements for the interfaces within the SAA Subsystem:

- SAA Server <-> ID GW (INT1)
- SAA Server <-> SAA Client (INT2)

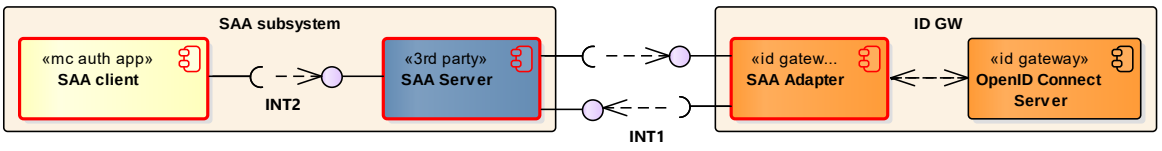

Figure 7: SAA interface options

Ideally it should be possible to easily swap out all or part of the SAA subsystem to accommodate new authentication capabilities as smartphones evolve. This could either be achieved by standardising INT1 (hence enabling the whole SAA subsystem to be easily swapped out) or by standardising INT2 (hence enabling different SAA Clients to be used with a common SAA Server) or by standardising both INT1 and INT2.

5.1.1 Option 1: INT1 and INT2 vendor proprietary

For the Base SAA, the aim will be to use existing COTS products for speed to market hence INT1 and INT2 are likely to be proprietary to each SAA vendor.

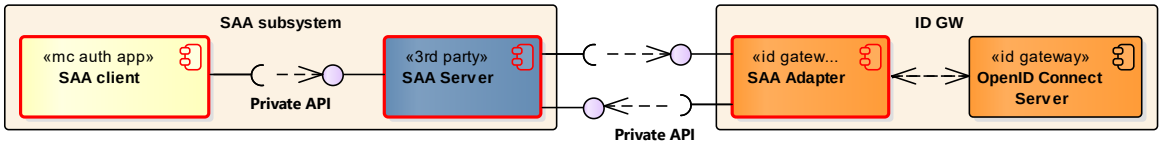

Figure 8: INT1 and INT2 are both proprietary interfaces

5.1.2 Option 2: INT1 standardised within Mobile Connect (Preferred approach)

This is effectively the approach that has been taken for the SIM applet by reusing the ETSI Mobile Signature Service API specification (TS 102 204) for the northbound interface from MSSP (Authentication Server) to the Identity GW – the Identity GW (in theory) only needs to develop an adapter to ETSI TS 102 204 and can then interwork with any MSSP vendor.

As shown below, a possible future improvement could be to standardise INT1 so that there would only need to be one integration carried out at the Identity GW and hence any SAA subsystem could then be used.

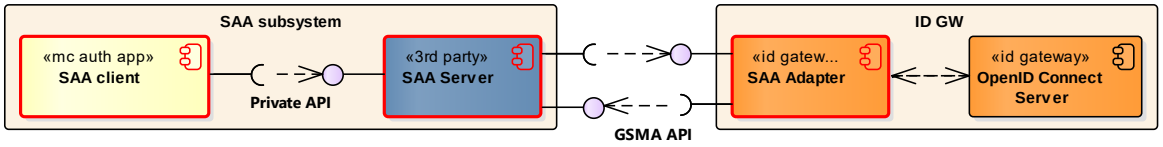

Figure 9: Standardising INT1

5.1.3 Option 3: INT2 standardised via FIDO UAF

Another approach would be to allow INT1 to be proprietary (per SAA vendor) but fix INT2 hence ensuring that different SAA Clients can be used without any further work required at the Identity GW – in effect, taking this approach moves the 'pluggability' from the Identity GW down to the Authentication Subsystem and to the device itself.

This approach of enabling support for multiple device-level authenticators irrespective of the back-end Authentication server and associated infrastructure is effectively what's been defined by the FIDO Alliance. By taking such an approach, INT2 could therefore utilise the FIDO UAF[45] protocol and use adapters on the smartphone for interfacing between the FIDO client and specific authenticators on the device[46].

In the FIDO approach, the Operator would need to procure the Identity GW, a FIDO-compliant Authentication Server (along with the FIDO Metadata Server) and an associated

[45] Universal Authentication Framework

[46] these adapters are known within the FIDO framework as Authenticator Specific Modules, ASMs

adapter to integrate the two, but once this has been done the Operator is free to try different authenticators without needing to change any of their back-end infrastructure.

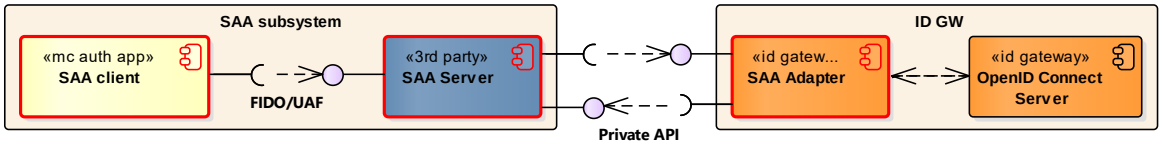

Figure 10: Standardising INT2 (e.g., FIDO UAF)

One important consideration of using FIDO is that a helper app would be needed on the device in order to invoke the FIDO authentication process based on a network-initiated push. In effect this becomes the 'Smartphone App' on the device but there may be some deployment dependencies/requirements that will require further consideration. There may also be commercial implications (licensing, IPR etc.) of Operators taking this approach and utilising FIDO hence this approach is still for further study. In addition, with the transition from FIDO 1.0 (UAF) to FIDO 2.0, the FIDO architecture changes hence more due diligence will be needed.

Summary

In summary, two approaches have been identified:

1. Standardise on INT1; SAA vendor subsystems remain proprietary.
2. INT1 remains proprietary per FIDO Server vendor; utilise UAF for INT2 hence enabling pluggability of Smartphone App Authenticators on the device itself.

5.1.4 Option 4: INT1 standardised within Mobile Connect, INT2 standardised via FIDO UAF

Of course the two approaches (Option 2 and Option 3) could also be combined to create a fourth option:

- INT1 being defined by Mobile Connect to simplify integration for Operators/Identity GW vendors.
- INT2 being defined by FIDO (UAF) to enable pluggability at the device.

Such an option would provide a complete "pluggable authenticator" approach, enabling the Operator to 'plugin' different SAA vendors as well as extending the pluggability to the device using UAF.

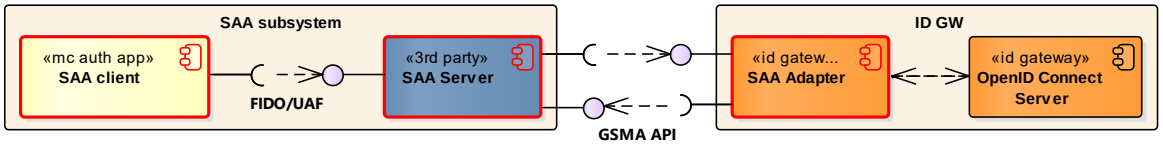

Figure 11: Standardising both INT1 and INT2

5.1.5 Pros and cons of the different SAA interface options

Interface options	Pros	Cons
Option 1: INT1 = Vendor proprietary INT2 = Vendor proprietary	• Enables existing SAA vendors to participate in MC ecosystem by providing MC certified SAA authenticator	• Requires Operator and/or Identity GW vendor to develop a bespoke adapter for the SAA vendor solution (however, this only needs to be done once) • Introduces SAA vendor lock in and makes it difficult for the Operators to migrate to another SAA vendor solution
Option 2: INT1 = Standardised API spec INT2 = Vendor proprietary	• Enables Operators and ID GW vendors to develop a single adapter to integrate with any SAA vendor solution • Enables pluggability of SAA sub-system allowing Operators to replace a vendor's SAA sub-system without impacting ID GW	• Places burden on SAA vendors to develop solutions in compliance with a standardised INT1 – many may choose not to bother and compete directly with Mobile Connect. • May introduce significant latency in getting agreement across Operators, Identity GW vendors and SAA vendors on how INT1 should be specified. • Implementing a new SAA may require implementation of a complete new SAA subsystem (e.g., from a new vendor) hence potentially introducing high cost for the Operator
Option 3: INT1 = Vendor proprietary INT2 = UAF (FIDO)	• Different authenticators can be implemented on the smartphone	• Requires the Operator and/or SAA vendor to develop ASMs for each new authenticator (e.g., fingerprint sensor; iris scanner etc.) • Requires Operator and/or Identity GW vendor to develop a bespoke adapter for the SAA vendor solution (however, this only needs to be done once) • Adopting FIDO may introduce commercial/strategic risks (TBD); but equally, not embracing FIDO could lead to FIDO solutions disintermediating Mobile Connect if SPs deploy their own multi-factor authentication solution
Option 4: INT1 = Standardised API Spec INT2 = UAF (FIDO)	• Enables Operators and ID GW vendors to develop a single adapter to integrate with any SAA vendor solution • Different authenticators can be implemented on the smartphone • Enables pluggability of SAA sub-system allowing Operators to replace a vendor's SAA sub-system without impacting ID GW	• Requires the Operator and/or SAA vendor to develop ASMs for each new authenticator (e.g., fingerprint sensor; iris scanner etc.) • Adopting FIDO may introduce commercial/strategic risks (TBD); but equally, might be needed to mitigate against disintermediation

Table 10: Comparison of SAA Interface options

For the Base SAA, the proposed approach is to go with whatever APIs the SAA vendors already have defined and deployed. For the Enhanced SAA, the possibility for Mobile Connect to define its own set of SAA subsystem APIs from the SAA Server -> SAA Adapter

(i.e. INT1) but allow for the INT2 interface (from SAA Server <-> SAA Client) to remain proprietary to the SAA vendor requires further investigation.

5.2 Security requirements & guidelines

This section provides an overview of the security measures that should be followed in securing the SAA solution end to end as illustrated in the following diagram:

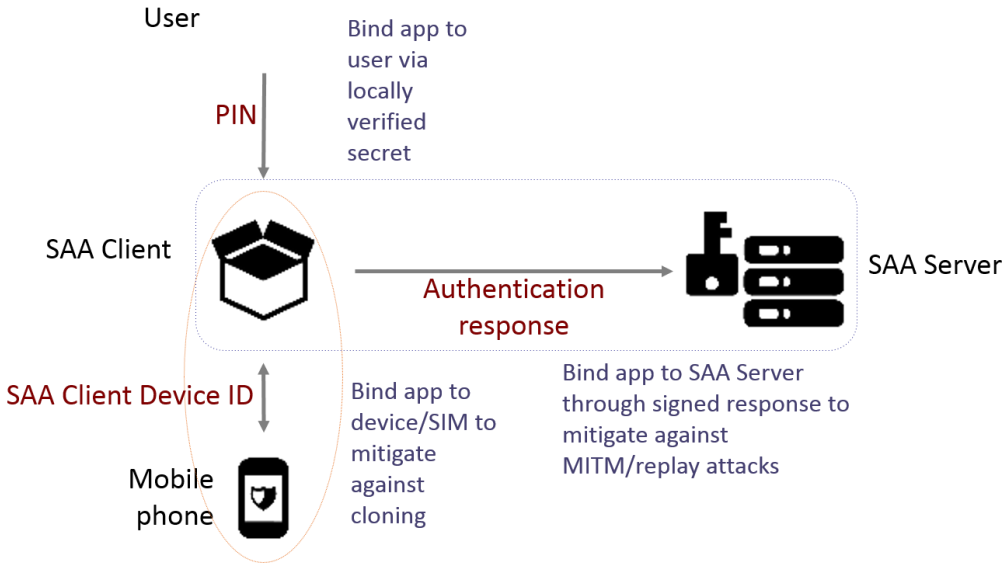

Figure 12: SAA Security Measures

The following subsections provide details of various security measures to provide a robust and secure SAA solution.

5.2.1 Device security checks

The SAA Client should perform device security checks on application start-up or activation. Some of these checks are performed locally by SAA Client and some are performed remotely by SAA Server as result of device check API invocation by SAA Client. This is the first API that should be called by SAA Client before invocation of any other APIs. These checks may include:

- **Detect jail broken/rooted device:** If the device is jail broken/rooted, notify the user accordingly and do not allow further use of the SAA Client.
- **Detect device/SIM changes**: Read device/SIM identifiers[11] from the Device secure key-store, generate a unique device token (SAA Client Device ID) as described in section 2.3 and compare with the existing device token (if present). Optionally, utilise Operator's network API to compare the network-registered device/SIM identifiers for the MSISDN with the device/SIM identifiers read from the device. If a change is found, notify the user accordingly and prompt the user to recover their existing account using account recovery information.
- **Verification of client signature**: The SAA Client creates a signature by signing the SAA Client Device ID with private key. The SAA Server verifies the signature by decrypting with corresponding public key. The decrypted SAA Client Device ID is then validated (e.g., whether or not it has been revoked due to Lifecycle events. See section 5.5.1 for lifecycle events integration with Mobile Connect). If verification of client signature fails, notify the user accordingly and do not

allow further use of the SAA Client. Further information on generation and utilisation of the SAA Client signature can be found in section 5.2.3

- **Code integrity check**: SAA Client must be able to detect at runtime that code has been added or changed from what it knows about its integrity at compile time. The app must be able to react appropriately at runtime to a code integrity violation by notifying the user accordingly and not allowing further use of the SAA Client.
- **Detection of Emulator/Debugging tools:** SAA Client must detect that the code is not running in an emulated environment/debugger and stop immediately.

On invocation of device check API call, SAA Server performs some of the above checks. On successful verification of these checks, SAA Server must generate a device check token proof and return it to SAA Client. The token proof generation algorithm is dependent on various security checks performed by SAA Server. For example: Detecting device/SIM changes or code integrity check or verification of client signature. It should be possible to deduce various security checks performed by SAA Server from device check token proof. SAA Client will be responsible for passing the token proof in subsequent API calls to SAA Server and SAA Server will be responsible for inspecting the token proof (deduce various security checks already performed) before completing the API request.

5.2.2 Device secure key-store

The SAA client must store data and tokens using whatever secure key-store is provided by the platform on which it resides.

5.2.2.1 Android

The most common security concern for an application on Android is whether the data that is saved on the device is accessible to other apps. There are three fundamental ways to store data on the device:

Using internal storage (Recommended approach)

By default, files created on internal storage are only accessible to the app. This protection is implemented by Android and sufficient for most applications.

To provide additional protection for sensitive data, data should be encrypted using a key that is not directly accessible to the application. For example, a key can be placed in a KeyStore[47] and can be unlocked seamlessly on the device by user's action. The local KeyStore[48]unlock is accomplished by a secure action such as swiping a finger, entering a PIN, or using biometrics depending on the LoA requested by the SP and Operator's policy.

Using external storage (Not Recommended)

[47] KeyStore - http://developer.android.com/reference/java/security/KeyStore.html

[48] KeyStore - http://developer.android.com/reference/java/security/KeyStore.html

Files created on external storage[49], such as SD Cards, are globally readable and writable. Because external storage can be removed by the user and also modified by any application, sensitive information should not be stored on external storage.

Using content providers

Content providers[50] offer a structured storage mechanism that can be limited to own application or exported to allow access by other applications. The ContentProvider[51] used in the SAA Client should not be accessible by other applications. This can be achieved by setting the value of 'android:exported' to 'false' for the SAA Client ContentProvider in the application manifest.

Please see Android Security Tips[52] for further reading.

5.2.2.2 iOS

Keychain Services[53] provides secure storage of passwords, keys, certificates, and notes for one or more users. A user can unlock a keychain with a single password, and any Keychain Services aware application can then use that keychain to store and retrieve secured data. There are a number of conditions that impact the security of the data an app stores in the Keychain:

- the presence and strength of a passcode on the device
- the access restrictions assigned to the app's Keychain items
- the chipset used in the device (recommended)

Presence of device passcode

Prior to iOS 8, there was no Apple-sanctioned way to determine if a passcode had been set on the device, so apps had no way of knowing if the data they entrusted to the Keychain was actually secure. Starting with iOS 8, Apple provided a new '*kSecAttrAccessible*' value that allows apps to store items in the Keychain only when the device has a '*passcode:kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly*'. Additionally, if the user removes their passcode, all the Keychain items with this access restriction will be removed as well, preventing the items from being exposed on the unprotected device.

Keychain access restrictions

Starting with iOS 4, each item in the Keychain can have its own access restriction defining when the item can be accessed; the item is securely encrypted the rest of the time. The recommended options are detailed below. These options have variants ending

[49] http://developer.android.com/guide/topics/data/data-storage.html#filesExternal

[50] Content providers - http://developer.android.com/guide/topics/providers/content-providers.html

[51] ContentProvider - http://developer.android.com/reference/android/content/ContentProvider.html

[52] http://developer.android.com/training/articles/security-tips.html

[53] Keychain services - https://developer.apple.com/library/mac/documentation/Security/Conceptual/keychainServConcepts/01introduction/introduction.html#//apple_ref/doc/uid/TP30000897-CH203-TP1

with '*ThisDeviceOnly'* that prevent the Keychain item from being included in iTunes backups, iCloud backups, or iCloud Keychain.

kSecAttrAccessibleWhenUnlocked

The Keychain item is secure when the device is off and when it is locked. This is recommended for items that need to be accessible only while the application is in the foreground. This is the default value for modern versions of iOS.

kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly

Similar to kSecAttrAccessibleWhenUnlocked, but only available if a passcode is set on the device. If the device lacks a passcode, the item will not be stored. Disabling the passcode after the item has been created with this restriction causes the item to be deleted. Items with this attribute never migrate to a new device through backups or iCloud. This access restriction is only available in iOS 8.0 and later.

Modern device hardware (recommended)

Devices running iOS 7 or iOS 8 on the Apple A7 and later A-series processors leverage Apple's new "Secure Enclave" technology. The Secure Enclave is a coprocessor that performs security-sensitive tasks, such as verifying the user's passcode and encrypting/decrypting keychain content without interference from malicious programs.

In practical terms, devices with a Secure Enclave cannot be jailbroken without the user's consent; i.e., while the device is locked or powered off. This significantly impedes attackers attempting to gain access to sensitive data by jailbreaking such a device.

Additionally, the Secure Enclave enforces a 5-second delay between failed attempts to unlock the device. This provides a governor against brute-force attacks in addition to safeguards enforced by iOS.

5.2.3 Public key cryptography for signing a challenge

It is highly recommended to use standard public key cryptography techniques to provide stronger authentication, protect against MITM attacks and enhance security as well as providing value-add through the business processes that PKI provides[54]. This model is similar to FIDO UAF. The use of public key cryptography supports non-repudiation requirements as well. The two-step process of keys creation and usage is defined as follows:

1. **During SAA setup/enrolment phase (Keys creation phase)**: the SAA Client creates a new key pair using either RSA-2048 or ECC-256 asymmetric cryptographic algorithms. It retains the private key and registers the public key with the SAA Server.
2. **During authentication phase (Keys usage phase):** the SAA Client proves possession of the private key to the SAA Server by signing a server generated challenge. The Client's private key can be used only after they are unlocked locally on the device by the user. The local unlock is accomplished by a secure action such

[54] Can be extended for non-repudiation use cases with proper ID proofing when issuing the certificates

as swiping a finger, entering a PIN, or using biometrics depending on the LoA requested by the SP and Operator's policy.

5.2.4 Summary of SAA security requirements

The SAA should implement mechanisms to mitigate against the following fraud and security issues:

- On start-up or activation, the SAA Client should perform device security checks as defined in section 5.2.1.
- Ensure the SAA Client has not been cloned:
  - Generate a digital fingerprint of the device and SIM card (SAA Client Device ID) in order to detect device/SIM card changes and protect from app cloning risks.
  - Utilisation of app cloning detection mechanisms.
- Protection against phishing attacks:
  - The SAA Client must show SP short name and transaction details (where applicable) on a per transaction basis as defined in section 3.5.
  - Optionally, the SAA Client must show SP logo and SP background image on a per transaction basis as defined in section 4.2. This is recommended for the Enhanced SAA.
  - In the case of a common SAA Client deployed across Operators (e.g., within a given market), the SAA Client should show the Operator name and logo as defined in section 3.4.
- Ensure the authentication response cannot be tampered with:
  - Use of transport layer security (TLS) for communication between SAA Client and SAA Server.
  - Using certificate checking to detect MITM attack vectors and ensuring that SAA Server only support secure crypto algorithms.
  - Inclusion of public key cryptography with minimum key length and avoiding weak crypto algorithms for signing SAA Server generated challenge as described in section 5.2.3.
- Ensure the SAA Client has not been compromised:
  - Code obfuscation (white-box cryptography) and black-boxing to make the app as tamperproof as possible.
  - Utilisation of rooting/jailbroken detection.
  - The SAA Client should store SAA Client Device ID and crypto keys securely on device secure key-store as defined in section 5.2.2.
- Message security, trust and non-repudiability
  - Digitally signed authentication challenge using the SAA Client's private key.
  - All API interaction must be done over a secure channel (HTTPS).
  - Auditing transaction (no sensitive data).
  - Use of Nonce to ensure replay attacks are minimised.
- Protection from OWASP top 10 mobile risks:

- Weak server side controls
- Insecure data storage
- Insufficient transport layer protection
- Unintended data leakage
- Poor authorisation and authentication
- Broken cryptography
- Client side injection
- Security decisions via untrusted inputs
- Improper session handling
- Lack of binary protections

Please see Annex A.1 for further information on security threats and prevention techniques.

5.3 SAA Client activation and association

5.3.1 MSISDN discovery in SAA Client

SAA Client discovers MSISDN of the user during activation phase. The MSISDN can be discovered in one of two ways:

- **If on-net (or the user is asked to switch off WLAN to force on-net):**
  - **Option 1:** the MSISDN can be retrieved through header enrichment. The SAA client would typically follow these steps:
    1. Make a simple API call to SAA Adapter through SAA Server. The assumption here is that ID GW and SAA Adapter will be deployed inside Operator's network. The Operator should white list the network domain of SAA Adapter for header enrichment.
    2. On receiving the API request, Operator's network component will insert the MSISDN in the header of the API request.
    3. SAA Adapter retrieves the MSISDN from the header and encrypts it using a symmetric key.
    4. Encrypted MSISDN is returned back to SAA Client through SAA Server.
    5. Sends the encrypted MSISDN to ID GW (through SAA Server->SAA Adapter) at the time of linking.
    6. ID GW decrypts the token using the symmetric key.

  - **Option 2:** the MSISDN can be retrieved through Operator API**.** The SAA client would typically follow these steps:
    1. Make a call to the appropriate Operator API to request user's MSISDN.
    2. Receive in return a unique token representing user's MSISDN.
    3. Provide such token to the ID GW (through SAA Server->SAA Adapter) at the time of linking.
    4. Using the token, the ID GW can then retrieve the actual MSISDN by invoking the Operator API.

- **If off-net (or where Header Enrichment is not available):**
  - **Option 1: SMS with one-time verification code:**

1. The SAA Client requests the user to enter their MSISDN which is sent to the ID GW for verification (through SAA Server->SAA Adapter).
2. To verify that the current device MSISDN matches, the ID GW pushes an SMS with a one-time verification code to the user's device.
3. The SAA Client either monitors the incoming SMS queue to retrieve the one-time verification code (this should be within 2 seconds) or prompts the user to enter the one-time verification code manually.
4. The SAA Client sends the one-time verification code to the ID GW for verification (through SAA Server->SAA Adapter) at the time of linking. If the one-time verification code matches, then the MSISDN is confirmed in the ID GW.

- **Option 2: Mobile-Originated (MO-SMS) and Mobile-Terminated (MT-SMS) SMS**
  1. The SAA Client requests the user to enter their MSISDN and generates a MO-SMS to an Operator registered short code.
  2. The SAA Client receives an MT-SMS in response, in turn validating the MSISDN.

     Note Both the mobile phone and the Operator's network must support this feature for the above option to work.

5.3.2 Technical flow: SAA Client activation and Mobile Connect registration instigated via SAA Client (MSISDN based pairing)

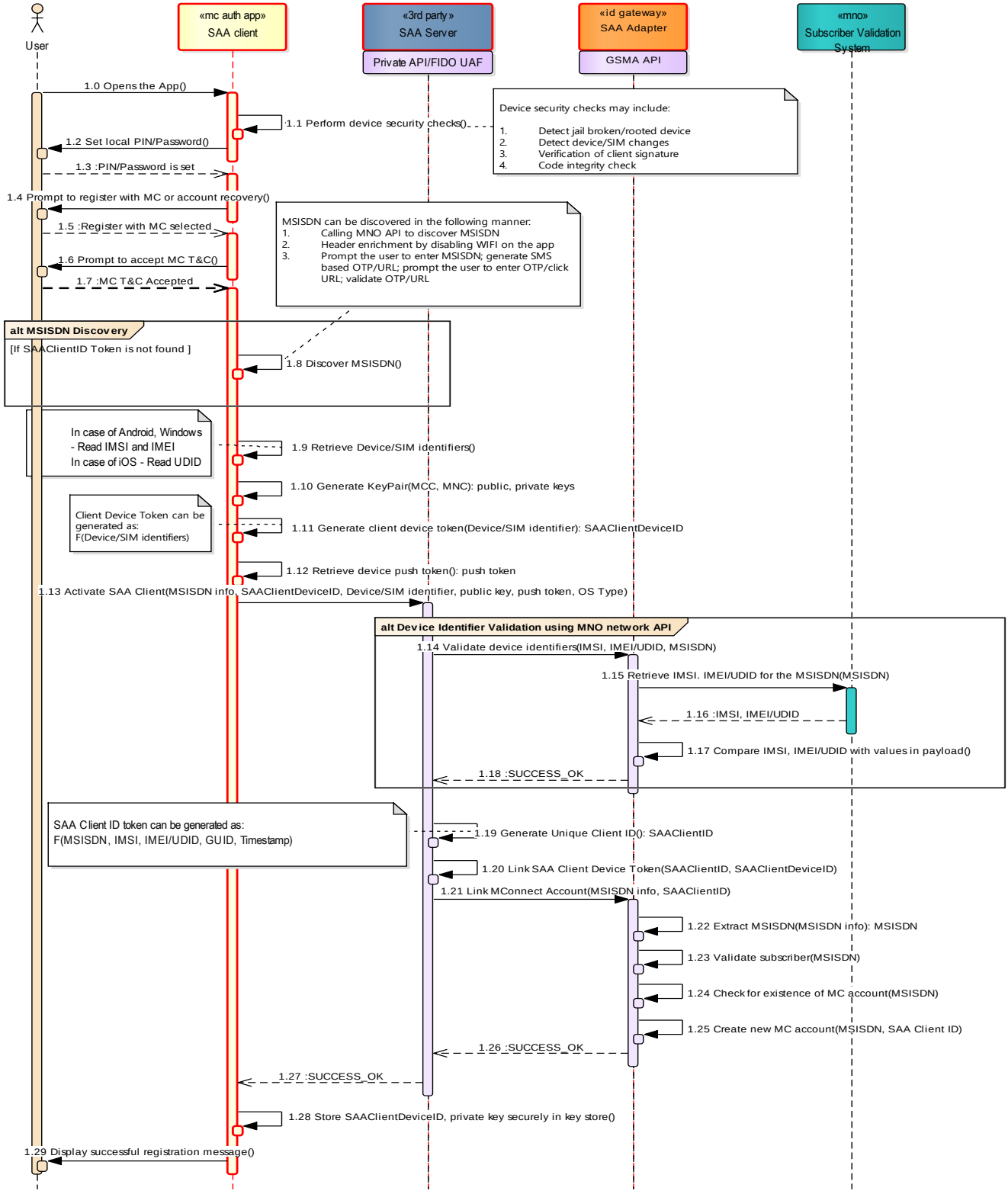

Figure 13: SAA Client activation and Mobile Connect registration instigated via the SAA Client (MSISDN based pairing)

1. User opens SAA Client.
2. SAA Client performs device security checks as described in section 5.2.1.
3. User is prompted to set local PIN. The PIN is seeded and hashed using SHA-256 hashing algorithm and stored securely on the device secure key-store as described in section 5.2.2.
4. User is prompted to register with Mobile Connect or recover an existing Mobile Connect account.
5. User selects the option to register for Mobile Connect account.
6. SAA Client prompts the user to accept Mobile Connect terms and conditions.
7. SAA Client discovers the MSISDN of the user as described in section 'MSISDN discovery'.
8. SAA Client reads device/SIM identifiers.
9. SAA Client generates cryptographic keys (public-private key pair) as described in section 5.2.3. This key pair will be used for signing and confirming the challenge for client verification during the authentication process.
10. SAA Client generates unique client device token (SAA Client Device ID) using the device/SIM identifiers obtained in step 8.
11. SAA Client reads device push token for registering with SAA Server.
12. Mobile Connect registration and SAA Client activation **using MSISDN**:
    (1) SAA Client invokes SAA vendor's setup API by passing MSISDN information (encrypted MSISDN or MSISDN and one-time verification code or token and MSISDN retrieval URL), public key, generated client device token (SAA Client Device ID), device/SIM identifiers, device push token and device platform type.
    (2) [Optional] SAA Server invokes device API of Operator through SAA Adapter to read the device/SIM identifiers for the authenticated MSISDN. The two sets of identifiers are compared (from the device and from the network) and an appropriate response is returned back to SAA Server.
    (3) On successful validation, SAA Server generates a unique client identifier (SAA Client ID). One approach for generation of this identifier can be a combination of device/SIM identifiers, MSISDN, GUID and timestamp.
    (4) SAA Server creates new SAA user account and associates newly generated SAA Client ID with SAA Client Device ID along with all the parameters passed in the setup API.
    (5) SAA Server invokes an API in the SAA Adapter to link the SAA Client ID with MSISDN passing SAA Client ID and MSISDN information (encrypted MSISDN or MSISDN and one-time verification code or token and MSISDN retrieval URL).
    (6) [where appropriate] SAA Adapter retrieves MSISDN using the information passed in previous step and validates the same.
    (7) [alternate exception flow] If a Mobile Connect account for the MSISDN already exists, appropriate error response is returned back to SAA Client through SAA Server, prompting the user to either recover existing Mobile Connect account or provide verification details to continue.
    (8) SAA Adapter creates a new Mobile Connect account associating MSISDN and SAA Client ID. The adapter adds the flag that the user has accepted T&Cs, and responds to the SAA Server with a successful response.
    (9) SAA Server returns successful response to SAA Client.

(10) SAA Client will be responsible for securely storing SAA Client Device ID and crypto keys on the device secure key-store as described in section 5.2.2.

(11) SAA Client should indicate to the user that the process is now complete (display some user friendly message).

13. Confirmation: Upon completion of the above step:

(1) User's MSISDN and SAA Client ID are now fully enrolled with Mobile Connect. Optionally, user may be prompted to enter personal information to validate their account details with the Operator.

(2) SAA Client displays Mobile Connect account recovery text to the user (recommendation is to use a random text of 16 alpha-numeric characters). Optionally, user can provide their email address for the account recovery text to be emailed.

5.3.3 Technical flow: SAA Client activation instigated via an Operator self-care portal (Association code based pairing)

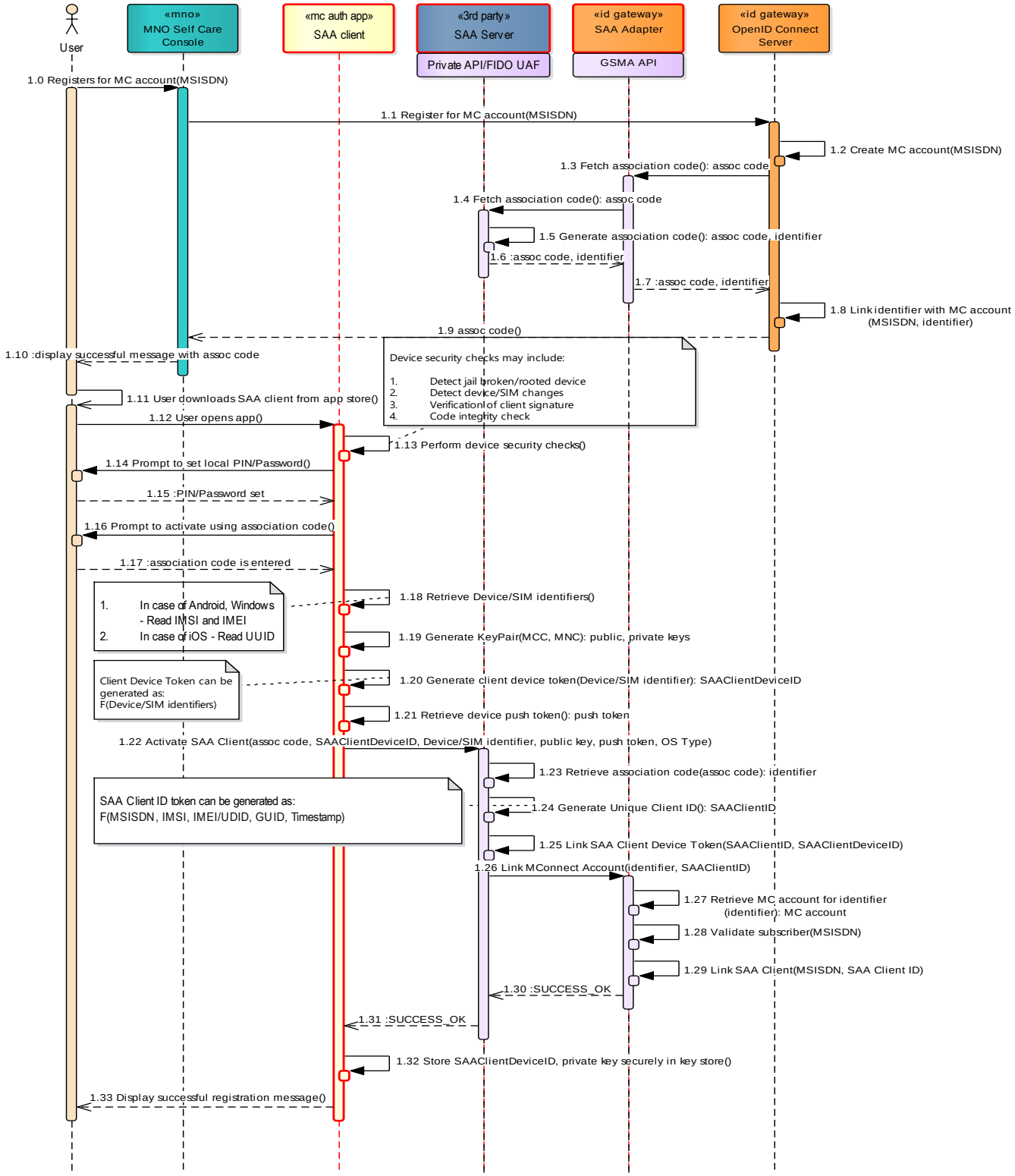

Figure 14: SAA Client activation instigated via Operator self-care portal (Association code based pairing)

1. User accesses the Operator self-care portal to register for Mobile Connect by providing their MSISDN and (optionally) display the T&C's online:
   (1) Operator's self-care console sends an API request to ID GW to register the user for MC account passing the MSISDN.
   (2) ID GW registers the user for a new MC account, if not already registered.
   (3) ID GW requests a new association code from SAA server through SAA adapter.
   (4) SAA server generates a new unique short-lived association code and identifier; returns it to ID GW through SAA adapter.
   (5) ID GW associates the user's MC account with the unique identifier; returns the association code back to MNO self-care console for display purpose.
2. Operator self-care portal displays the association code to the user with a successful registration message. The portal presents links for the user to download the SAA Client from appropriate App Store if not already pre-embedded/downloaded, and request that the user open the app.
3. User downloads SAA Client from appropriate App Store.
4. User opens SAA Client.
5. SAA Client performs device security checks as described in section 5.2.1.
6. User is prompted to set local PIN. The PIN is seeded and hashed using SHA-256 hashing algorithm and stored securely on the device secure key-store as described in section 5.2.2.
7. SAA Client prompts the user to enter association code.
8. User enters the association code and initiates the activation process.
9. SAA Client reads device/SIM identifiers.
10. SAA Client generates cryptographic keys (public-private key pair) as described in section 5.2.3. This key pair will be used for signing and confirming the challenge for client verification during authentication process.
11. SAA Client generates unique client device token (SAA Client Device ID) using device/SIM identifiers read in Step 9.
12. SAA Client reads device push token for registering with SAA server.
13. SAA Client enrolment process **using association code**:
   (1) SAA Client invokes SAA vendor's setup API by passing association code, public key, generated client device token (SAA Client Device ID), device/SIM identifiers, device push token and device OS type.
   (2) SAA Server validates the association code and retrieves the unique identifier.
   (3) On successful validation, SAA server generates a unique client identifier (SAA Client ID). One approach for generation of this identifier can be a combination of device/SIM identifiers, MSISDN, GUID, timestamp.
   (4) SAA server creates new SAA Client account and associates newly generated SAA Client ID with SAA Client Device ID along with all the parameters passed in the setup API.
   (5) SAA server invokes an API in SAA adapter to link SAA Client ID with user's Mobile Connect account passing SAA Client ID and unique identifier.
   (6) SAA adapter retrieves the Mobile Connect account for the unique identifier.
   (7) [alternate exception flow] If a Mobile Connect account is not found, appropriate error response is returned back to SAA Client through SAA Server, prompting the user to register for Mobile Connect through Operator self-care portal.

(8) SAA Adapter links SAA Client ID with the existing account and responds with a successful response to SAA server.

(9) SAA Server returns successful response to SAA Client.

(10) SAA Client will be responsible for storing SAA Client Device ID and crypto keys securely on the device secure key-store as described in section 5.2.2.

(11) SAA Client should indicate to the user that the process is now complete (display some user friendly message).

14. Confirmation: Upon completion of above step:

(1) User's MSISDN and SAA Client ID are now fully enrolled with Mobile Connect. Optionally, user may be prompted to enter personal information to validate their account details with the Operator.

(2) SAA Client displays Mobile Connect account recovery text to the user (recommendation is to use a random text of 16 alpha-numeric characters). Optionally, user can provide their email address for the account recovery text to be emailed.

5.3.4 Technical flow: SAA Client activation instigated via an Operator self-care portal (MSISDN based pairing)

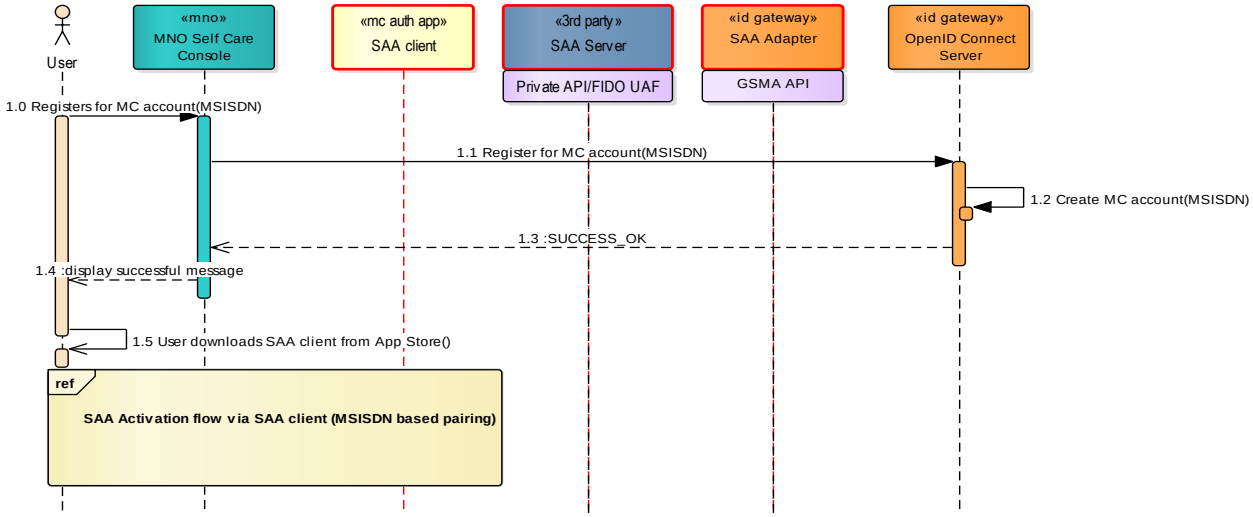

Figure 15: SAA Client activation instigated via an Operator self-care portal (MSISDN based pairing)

1. User accesses the Operator self-care portal to register for Mobile Connect by providing their MSISDN and (optionally) display the T&C's online:

(1) Operator's self-care console sends an API request to ID GW to register the user for MC account passing the MSISDN.

(2) ID GW registers the user for a new MC account, if not already registered.

2. MNO self-care console displays a successful registration message to the user. The console presents links for the user to download the SAA Client from the appropriate App Store if not already pre-embedded/downloaded, and request that the user open the app.
3. User downloads SAA Client from appropriate App Store.
4. Same as Steps (1)–(13) in section 5.3.2.

5.3.5 Technical flow: Recovery of existing account using recovery information

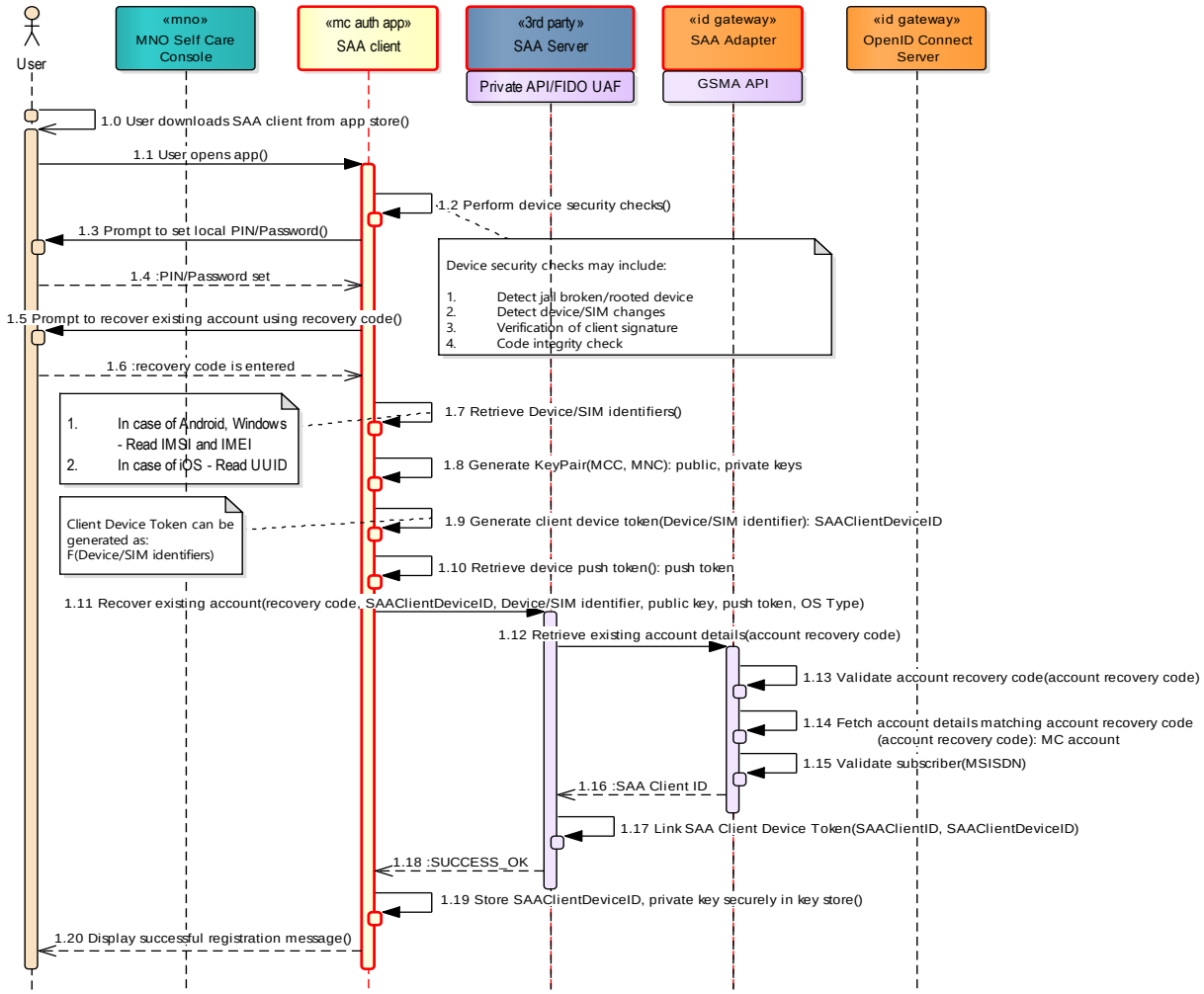

Figure 16: Recovery of existing account using account recovery code

1. User downloads SAA Client from appropriate App Store.
2. SAA Client performs device security checks as described in section 5.2.1.
3. User is prompted to set local PIN. The PIN is hashed and seeded using SHA-256 hashing algorithm and stored securely on the device secure key-store as described in section 5.2.2.
4. User is prompted to register with Mobile Connect or recover an existing Mobile Connect account.
5. User selects the option to recover an existing account by entering the recovery information. User can find the recovery code in the account recovery email that was sent as part of the original activation process described in previous technical flow sections. Optionally, user's MSISDN will be discovered as described in section 5.3.1. This guarantees that the MSISDN/SIM is in possession of a registered Mobile Connect user.
6. SAA Client reads device/SIM identifiers.

7. SAA Client generates cryptographic keys (public-private key pair) as described in section 5.2.3. This key pair will be used for signing and confirming the challenge for client verification during authentication process.
8. SAA Client enrolment process **using account recovery code**:
   (1) SAA Client invokes SAA vendor's recovery API by passing account recovery information, public key, generated client device token (SAA Client Device ID), device/SIM identifiers, device push token and device OS type.
   (2) SAA Server calls an API in ID GW through SAA Adapter to retrieve existing client details passing the account recovery information.
   (3) SAA Adapter validates the account recovery information, retrieves corresponding MC account details and returns previous SAA Client ID.
   (4) SAA Server associates SAA Client ID with SAA Client Device ID along with all the parameters passed in the setup API.
   (5) SAA Server returns successful response to SAA Client.
   (6) SAA Client will be responsible for storing SAA Client Device ID and crypto keys securely on the device secure key-store as described in section 5.2.2.
   (7) SAA Client should indicate to the user that the process is now complete (display some user friendly message).
9. Confirmation: Upon completion of above step:
   (1) User's MSISDN and new SAA Client ID are now fully enrolled with Mobile Connect.

5.4 Authentication flows

5.4.1 Remote invocation (network push; separate consumption device)

5.4.1.1 User flow (iOS and Android)

The various flows captured below do not include any server side interactions including request/response messages. Please see corresponding technical flows for detailed interaction between various actors.

Scenario 1 – iOS based SAA Client; device is locked; SAA Client is not active

1. User initiates Mobile Connect AuthN request in the SP's website on the consumption device browser.
2. Consumption device browser's user agent initiates OIDC call with Operator's ID GW.
3. ID GW initiates AuthN request with SAA Server through SAA Adapter, resulting in a push notification trigger to the authentication device.
4. Push notification prompt is displayed in the notification area of the authentication device (containing the SAA Client name and prompt).
5. User unlocks the authentication device.
6. User can either click on the notification prompt in the notification bar or SAA Client icon to open the app.
7. SAA Client prompts the user to authenticate (based on LoA requested by the SP and Operator's policy) on the authentication device.
8. User authenticates via the SAA Client.

9. OIDC AuthN response is sent back to the consumption device's user agent from the ID GW.
10. User continues with the journey on the consumption device, the result of the authentication being displayed to the user.

Scenario 2 – iOS based SAA Client; device is locked; SAA Client is already open and active

1. User initiates Mobile Connect AuthN request in the SP's website on the consumption device browser.
2. Consumption device browser's user agent initiates OIDC call with Operator's ID GW.
3. ID GW initiates AuthN request with SAA Server through SAA Adapter, resulting in a push notification trigger to the authentication device.
4. Push notification prompt is displayed in the notification area of the authentication device (containing the SAA Client name and prompt).
5. User unlocks the authentication device.
6. SAA Client prompts the user to authenticate (based on LoA requested by SP and Operator's policy) on the authentication device.
7. User authenticates via the SAA Client.
8. OIDC AuthN response is sent back to the consumption device's user agent from the ID GW.
9. User continues with the journey on the consumption device, the result of the authentication being displayed to the user.

Scenario 3 – iOS based SAA Client; device is unlocked; SAA Client is not active

1. User initiates Mobile Connect AuthN request in SP's website on consumption device browser.
2. Consumption device browser's user agent initiates OIDC call with Operator's ID GW.
3. ID GW initiates AuthN request with SAA Server through SAA Adapter, resulting in a push notification trigger to authentication device.
4. Push notification prompt is displayed in the notification area of the authentication device (containing SAA Client name and prompt).
5. User clicks on the push notification prompt, iOS activates SAA Client.
6. SAA Client prompts the user to authenticate (based on LoA requested by SP and Operator's policy) on authentication device.
7. User authenticates on SAA Client.
8. OIDC AuthN response is sent back to the consumption device's user agent from ID GW.
9. User continues with the journey on consumption device, the result of the authentication being displayed to the user.

Scenario 4 – iOS based SAA Client; device is unlocked; SAA Client is already open and active

1. User initiates Mobile Connect AuthN request in SP's website on consumption device browser.
2. Consumption device browser's user agent initiates OIDC call with Operator's ID GW.

3. ID GW initiates AuthN request with SAA Server through SAA Adapter, resulting in a push notification trigger to authentication device.
4. SAA Client prompts the user to authenticate (based on LoA requested by SP and operator's policy) on authentication device.
5. User authenticates on SAA Client.
6. OIDC AuthN response is sent back to the consumption device's user agent from ID GW.
7. User continues with the journey on consumption device, the result of the authentication being displayed to the user.

Scenario 5 – Android based SAA Client; device is locked; SAA Client is not active

1. User initiates Mobile Connect AuthN request in SP's website on consumption device browser.
2. Consumption device browser's user agent initiates OIDC call with Operator's ID GW.
3. ID GW initiates AuthN request with SAA Server through SAA Adapter, resulting in a push notification trigger to authentication device.
4. Push notification prompt is displayed in the notification area of the authentication device (containing SAA Client icon).
5. User unlocks the authentication device.
6. User can either click on the notification prompt in the notification bar or SAA Client icon to open the app.
7. SAA Client prompts the user to authenticate (based on LoA requested by SP and Operator's policy) on authentication device.
8. User authenticates on SAA Client.
9. OIDC AuthN response is sent back to the consumption device's user agent from ID GW.
10. User continues with the journey on consumption device, the result of the authentication being displayed to the user.

Scenario 6 – Android based SAA Client; device is locked; SAA Client is already open and active

1. User initiates Mobile Connect AuthN request in SP's website on consumption device browser.
2. Consumption device browser's user agent initiates OIDC call with Operator's ID GW.
3. ID GW initiates AuthN request with SAA Server through SAA Adapter, resulting in a push notification trigger to authentication device.
4. Push notification prompt is displayed in the notification area of the authentication device (containing SAA Client icon).
5. User unlocks the authentication device.
6. SAA Client prompts the user to authenticate (based on LoA requested by SP and Operator's policy) on authentication device.
7. User authenticates on SAA Client.
8. OIDC AuthN response is sent back to the consumption device's user agent from ID GW.

9. User continues with the journey on consumption device, the result of the authentication being displayed to the user.

Scenario 7 – Android based SAA Client; device is unlocked; SAA Client is not active

1. User initiates Mobile Connect AuthN request in SP's website on consumption device browser.
2. Consumption device browser's user agent initiates OIDC call with Operator's ID GW.
3. ID GW initiates AuthN request with SAA Server through SAA Adapter, resulting in a push notification trigger to authentication device.
4. Push notification prompt is displayed in the notification area of the authentication device (containing SAA Client name and prompt).
5. User clicks on the push notification prompt, Android activates the SAA Client.
6. SAA Client prompts the user to authenticate (based on LoA requested by SP and Operator's policy) on authentication device.
7. User authenticates on SAA Client.
8. OIDC AuthN response is sent back to the consumption device's user agent from ID GW.
9. User continues with the journey on consumption device, the result of the authentication being displayed to the user.

Scenario 8 – Android based SAA Client; device is unlocked; SAA Client is already open and active

1. User initiates Mobile Connect AuthN request in SP's website on consumption device browser.
2. Consumption device browser's user agent initiates OIDC call with Operator's ID GW.
3. ID GW initiates AuthN request with SAA Server through SAA Adapter, resulting in a push notification trigger to authentication device.
4. SAA Client prompts the user to authenticate (based on LoA requested by SP and Operator's policy) on authentication device.
5. User authenticates on SAA Client.
6. OIDC AuthN response is sent back to the consumption device's user agent from ID GW.
7. User continues with journey on consumption device, the result of the authentication being displayed to the user.

5.4.1.2 Technical flow

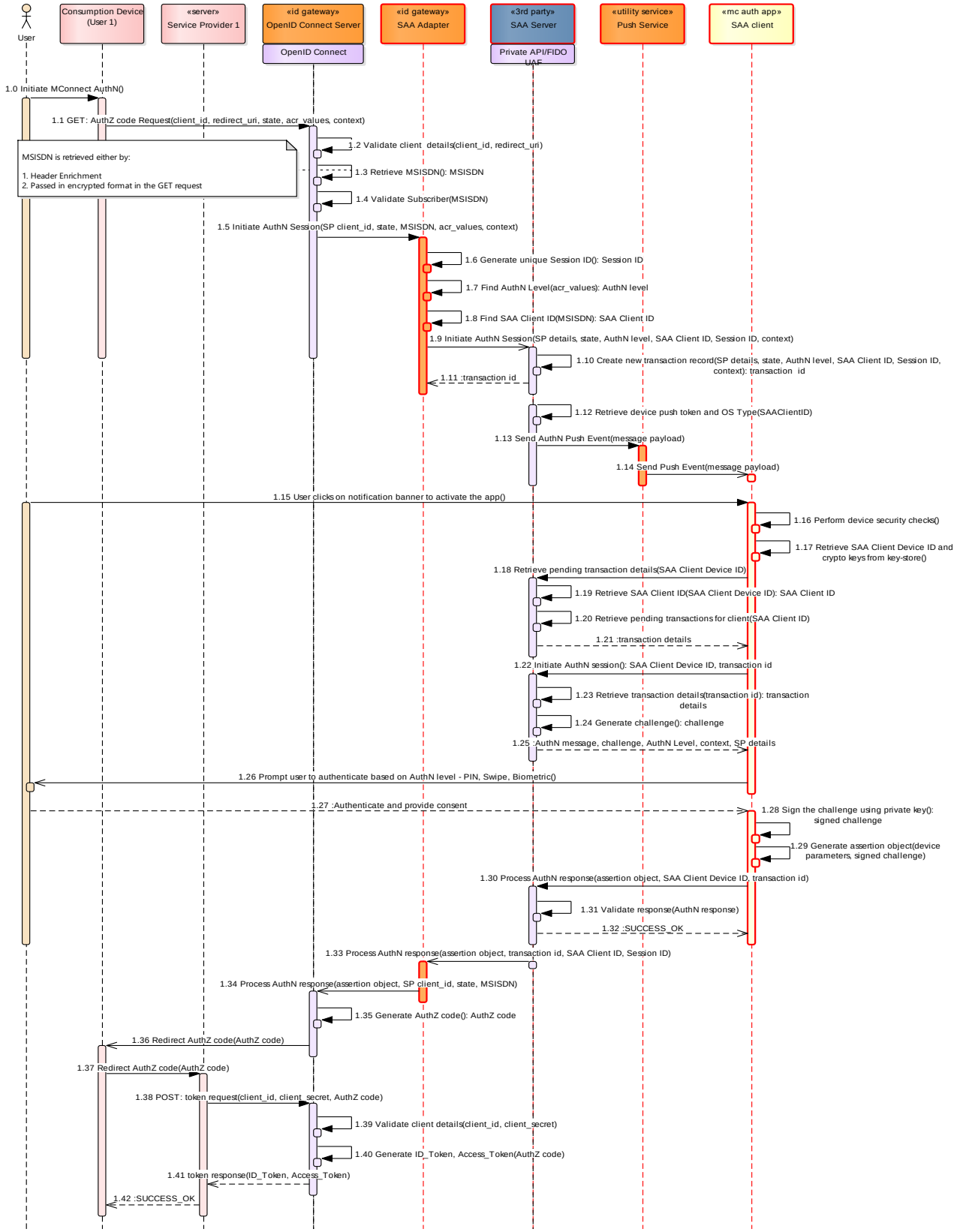

Figure 17: SAA Client remote invocation technical flow (MSISDN prompt in SP App)

1. User accesses SP's web page on the consumption device and needs to authenticate:
   - The SP implementation uses a Discovery mechanism such as the API Exchange to identify the Operator and needed parameters.
2. The user agent of the consumption device sends an OIDC AuthZ code request to the ID GW of the operator passing SP client_id, state, context and other relevant parameters.
3. The ID GW validates the parameters passed in the request.
4. The ID GW retrieves the MSISDN/PCR of the user. The MSISDN/PCR is retrieved either by:
   - Header enrichment
   - Encrypted MSISDN passed in AuthZ code request's login_hint parameter
   - Plain text MSISDN passed in AuthZ code request's login_hint parameter, in case of trusted SP
   - PCR passed in AuthZ code request's login_hint parameter
5. Optionally, If the ID GW is unable to retrieve MSISDN/PCR in the OIDC request, it presents a MSISDN discovery page in SP app's external user agent to capture user's registered MSISDN.
6. The ID GW checks the standing of the MSISDN/Mobile Connect account being authenticated; If the mobile account is inactive/suspended, the ID GW rejects the SP authentication request (passing back the requisite error codes).
7. The ID GW initiates an AuthN request call with the SAA Adapter passing the parameters received in original request along with MSISDN.
8. SAA Adapter generates a unique session id for the combination of MSISDN, SP client_id and state.
9. SAA Adapter determines the required AuthN level based on the acr_value passed in the SP request.
10. SAA Adapter retrieves the SAA Client ID corresponding to the MSISDN.
11. SAA Adapter utilises the SAA Server API to initiate an AuthN session with the SAA Client passing the relevant parameters (AuthN level, session id, details of SP [short name, SP logo URL, SP background image URL etc.,], context and SAA Client ID).
    (1) SAA Server checks the validity of the SAA Client ID (e.g., whether or not it has been revoked due to Lifecycle events. See section 5.5.1 for lifecycle events integration with Mobile Connect).
    (2) If valid, SAA Server creates a new transaction record with an unique transaction id for SAA Client ID; transaction id is returned to SAA Adapter.
    (3) SAA Adapter records the transaction id against the session id and marks the transaction in "Pending" state.
12. SAA Server retrieves the device push token and Platform type[55] for the SAA Client ID.
13. SAA Server triggers platform-specific push notification message to initiate AuthN session on SAA Client.
14. SAA Client is activated when the user clicks on the notification prompt on the authentication device.

[55] i.e., iOS or Android

15. SAA Client performs device security checks as described in section 5.2.1.
16. SAA Client retrieves private key and SAA Client Device ID tokens from device secure key-store as described in section 5.2.2.
17. SAA Client retrieves the pending transaction details (including transaction id) from SAA Server passing SAA Client Device ID.
18. SAA Client invokes an API in SAA Server to initiate AuthN session passing SAA Client Device ID and transaction id.
19. SAA Server validates incoming parameters and retrieves transaction record matching the transaction id.
20. SAA Server generates a unique challenge and returns an AuthN request message containing the challenge, transaction id, AuthN level, context and details of SP.
21. SAA Client prompts the user for a 'Click OK' or to 'Enter their PIN' or 'Fingerprint swipe' or 'Facial/Iris Scan' depending on AuthN level.
22. On successful authentication, SAA Client generates a new assertion object containing device characteristics (locale, device locked/unlocked etc.) and signed challenge (using the private key).
23. SAA Client invokes an API in SAA Server to process AuthN response message passing the assertion object, SAA Client Device ID and transaction id.
24. SAA Server validates the AuthN response message and responds appropriately to SAA Client.
25. SAA Server invokes a callback API in SAA Adapter to process AuthN response message passing the transaction id and assertion object.
26. SAA Adapter records the assertion object and marks the transaction as "Complete"; passes AuthN response back to ID GW containing the transaction id and session id.
27. The ID GW generates an OIDC response back to the SP:
    (1) The AuthZ code is sent as a redirect (through consumption device's user agent) to the registered redirect_uri of SP.
    (2) The SP backend server implementing the redirect_uri retrieves the AuthZ code from the URI.
    (3) The SP backend server makes the token call passing the AuthZ code to get the access token and the ID Token which contains the PCR.

5.4.2 Remote invocation (network push; mobile browser)

5.4.2.1 User flow

In this scenario, the user has invoked Mobile Connect via the browser on their mobile phone (e.g., the user has navigated to an SP login page and chosen to log in using Mobile Connect). The mobile browser will invoke Mobile Connect in the normal way, and the technical flow will be as per the default remote invocation case (see section 5.4.1.2); however on the same consumption device, the user will be navigated away from their mobile browser in order to authenticate via the SAA Client hence there is the added consideration of how to return the user to the mobile browser so that they can continue with the SP service.

The various flows captured below do not include any server side interactions including request/response messages. Please see corresponding technical flows for detailed interaction between various actors.

Scenario 1 – Same consumption and authentication device (iOS based)

1. User initiates Mobile Connect AuthN request in SP's website on mobile browser.
2. Mobile browser's user agent initiates OIDC call with Operator's ID GW.
3. ID GW initiates AuthN request with SAA Server through SAA Adapter, resulting in a push notification trigger to authentication device.
4. Push notification prompt is displayed in the notification area of the authentication device (containing SAA Client name and prompt).
5. User clicks on push notification prompt, iOS activates SAA Client and mobile browser app goes into the background.
6. SAA Client prompts the user to authenticate (based on LoA requested by SP and Operator's policy) on authentication device.
7. User authenticates on SAA Client and deactivates it by clicking on the home button.
8. OIDC AuthN response sent back to the mobile browser's user agent from ID GW.
9. User clicks on the mobile browser icon to restart the app and continue with the journey, the result of the authentication being displayed to the user.

Scenario 2 – Same consumption and authentication device (Android based)

1. User initiates Mobile Connect AuthN request in SP's website on mobile browser.
2. Mobile browser's user agent initiates OIDC call with Operator's ID GW.
3. ID GW initiates AuthN request with SAA Server through SAA Adapter, resulting in a push notification trigger to authentication device.
4. Push notification prompt is displayed in the notification area of the authentication device (containing SAA Client name and prompt).
5. User clicks on push notification prompt, Android activates SAA Client
6. SAA Client prompts the user to authenticate (based on LoA requested by SP and operator's policy) on authentication device.
7. User authenticates on SAA Client and deactivates it by clicking the home button.
8. OIDC AuthN response sent back to the mobile browser's user agent from ID GW.
9. User clicks on the mobile browser icon or goes back to the mobile browser app using the back button, continues with the journey, the result of the authentication being displayed to the user.

5.4.3 Local invocation from SP app (deep-linking using custom URI scheme)

5.4.3.1 Logical architecture – Component View

The diagram below depicts the logical architecture and conceptual flow of messages between various components for completing the authentication process in the particular scenario of local invocation of the SAA Client from an SP app using the app deep-linking method:

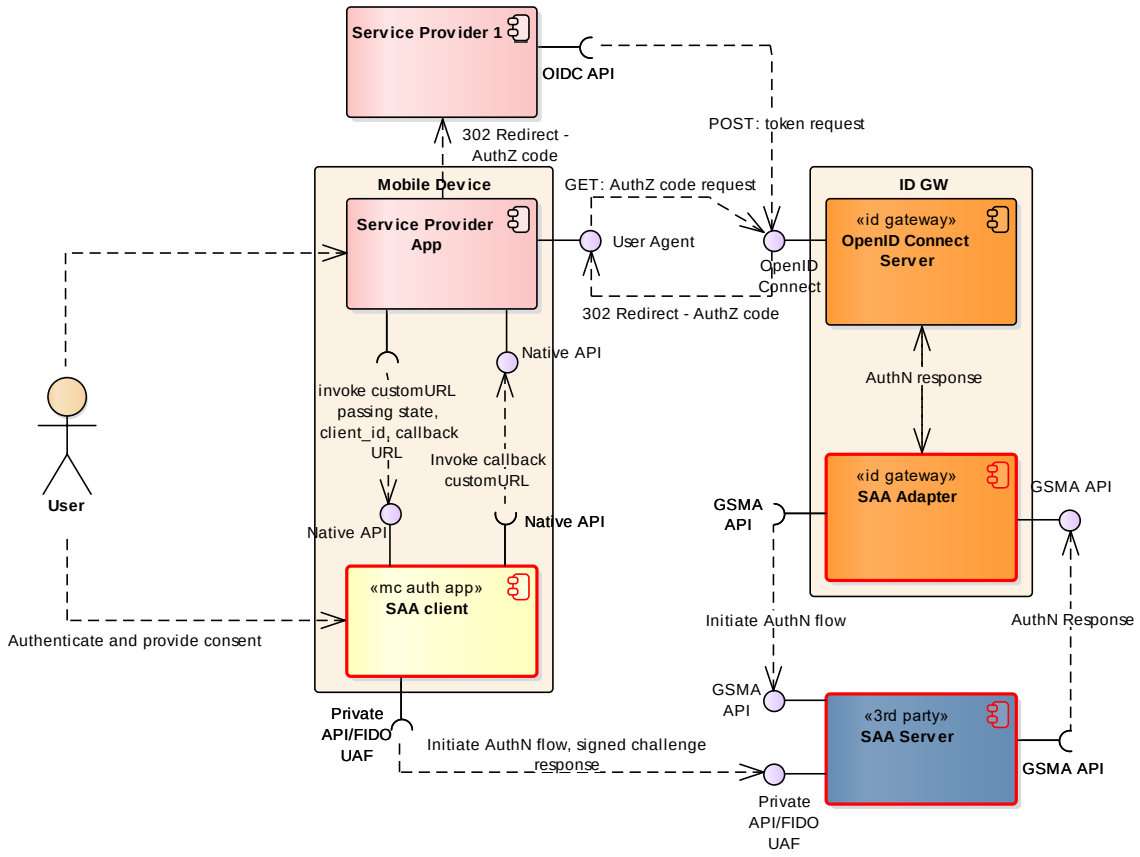

Figure 18: SAA Client local invocation component view

5.4.3.2 App deep-linking implementation guidelines

Deep-linking is a methodology for launching a native mobile application via a link (URL) by connecting a unique URL to a defined action in a mobile app thereby seamlessly linking users to relevant content.

Deep-link structure

Deep-linking functions much like a traditional hyperlink on a webpage. It is composed of separate elements that make up what is referred to as a Uniform Resource Identifier (URI). The URI contains all the information that, when invoked, launches a mobile application with a specific screen.

The best practice is to implement a URL with a unique scheme name and routing parameters (path and query strings) that represent custom actions to take in the app (SP App and SAA Client). The recommendation is to use a simple URL structure as shown in the example below:

```
scheme_name://path?query_string
```

For URI Scheme Namespace Considerations, please see section 4.1.2 of the OAuth 2.0 for Native Apps spec.

Routing parameters are optional, but are highly recommended. They can be used for routing the user to specific screens of the application or passing in additional parameters. The query string is optional, and might be used to pass specific named parameters.

Developer introduction

- Select the URI scheme and declare it in the app's manifest. As discussed in the previous section, the scheme name must be unique to the SP App and SAA Client otherwise conflicts with other apps may occur.
- Define the actions that must be launched by using a deep-link. Make sure these actions are in accordance with the URI syntax that is chosen. As discussed in the previous section, the use of URL syntax is highly recommended; e.g., `scheme_name://path?query_string`

Deep-link implementation – iOS

The high level process is as follows:

- iOS apps are self-contained entities. There is only one point of entry in the app: the AppDelegate. When a deep-link to the SAA Client is initiated, it will call the AppDelegate with the deep-linking metadata.
- It is important to maintain a consistent state in the SP App and SAA Client while providing the desired experience.
- For example, this can mean allowing the user to return to the main screen of the app. To accomplish this, the appropriate view controllers should be used to send the user to the desired part of the app while still maintaining the correct view hierarchy.
- When the app is opened, it is possible to recover the URL that was used to launch it and process it as needed.

Further details can be found in reference documentation[56] of AppDelegate[57]

Deep-link implementation – Android

The high level process is as follows:

- Android apps are composed of a set of Activities. Each of these Activities can be called by other apps if configured as such. Depending on how the app and deep-links are structured, you can choose to use one central endpoint or many.
- It is important to maintain a consistent state in the SP App and SAA Client while providing the desired experience. An Android Activity will launch on top of the current context and it is important to ensure that the appropriate view hierarchy is maintained. Additionally, it is important to have the necessary data ready for the Activity when it is loaded for the user.
- When the app is opened, it is possible to recover the URL that was used to launch it and process it as needed.

56 https://developer.apple.com/library/IOs/documentation/UIKit/Reference/UIApplicationDelegate_Protocol/Reference/Reference.html

57 https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplicationDelegate_Protocol/index.html

Further details can be found in reference documentation[58] of Android deep-linking[59]

5.4.3.3 User flow

The flows captured below do not include any server side interactions including request/response messages. Please see corresponding technical flows for detailed interaction between various actors.

1. User initiates Mobile Connect AuthN request in SP App via an external user agent.
2. SP App's user agent initiates OIDC call with Operator's ID GW; SP App invokes the custom URL of the SAA Client to launch it.
3. SAA Client prompts the user to authenticate (based on LoA requested by SP and Operator's policy).
4. User authenticates via the SAA Client; SAA Client deactivates and returns control back to the SP App by invoking the custom URL of the SP App; Device's OS activates the SP App.
5. OIDC AuthN response sent back to the SP App's user agent from ID GW.
6. User continues with the journey on SP app; the result of the authentication being displayed to the user.

[58] http://developer.android.com/training/app-indexing/deep-linking.html

[59] Android deep-linking - http://developer.android.com/training/app-indexing/deep-linking.html

5.4.3.4 Technical flow – User is prompted for MSISDN in SP App

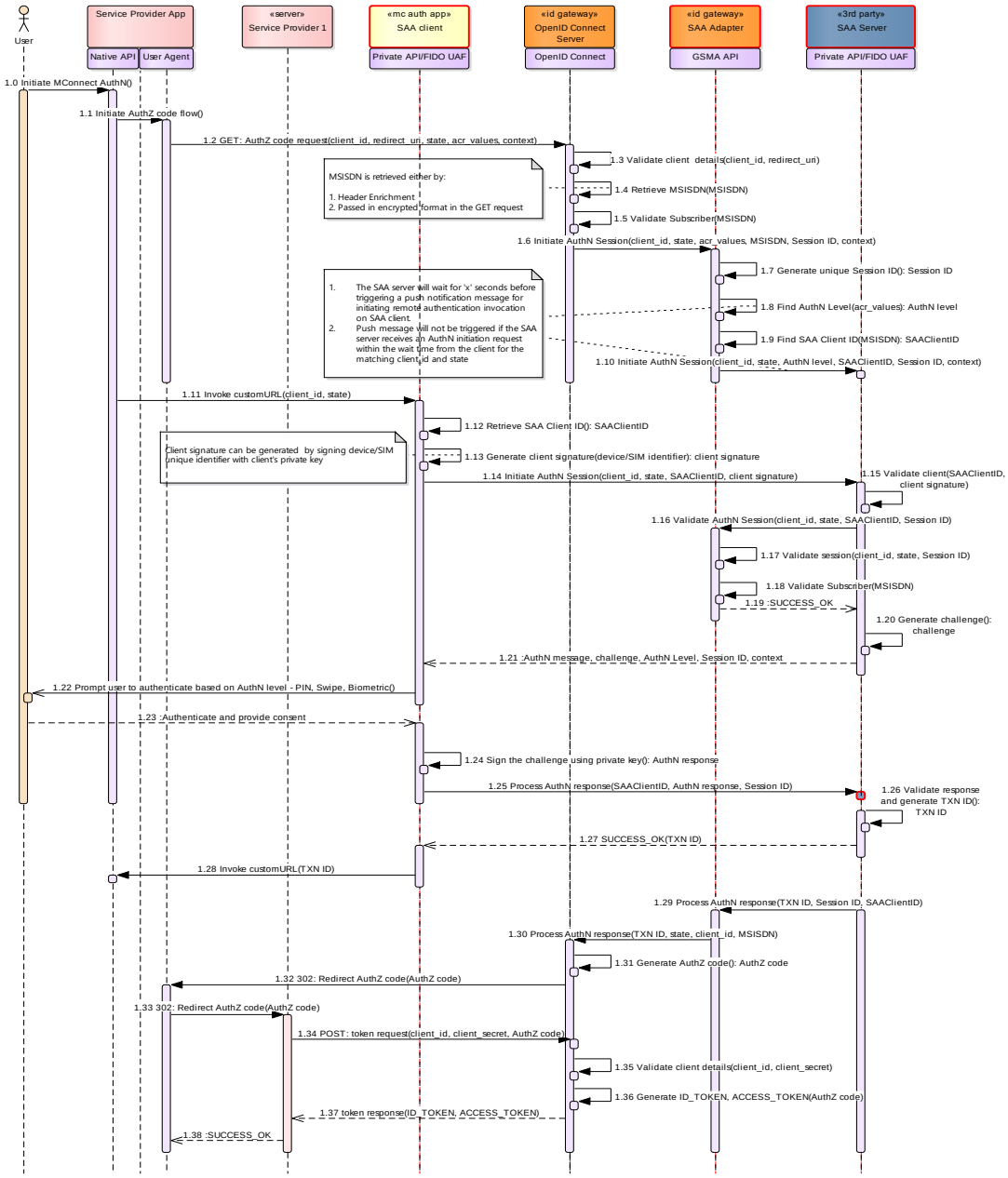

Figure 19: SAA Client local invocation technical flow (MSISDN prompt in SP App)

1. User accesses SP app on the mobile device and needs to authenticate
   - The SP implementation uses a Discovery mechanism such as the API Exchange to identify the Operator and needed parameters.
2. The SP app sends an OIDC AuthZ code request to the ID GW of the Operator through an external user agent passing SP client_id, state, context, login_hint and other relevant parameters. In parallel, the SP app invokes the custom URL of the SAA Client.
3. The ID GW validates the parameters passed in the request.
4. The ID GW retrieves the MSISDN/PCR of the user. The MSISDN/PCR is retrieved either by:
   - Header enrichment
   - Encrypted MSISDN passed in AuthZ code request's login_hint parameter
   - Plain text MSISDN passed in AuthZ code request's login_hint parameter, in case of trusted SP
   - PCR passed in AuthZ code request's login_hint parameter
5. Optionally, If the ID GW is unable to retrieve MSISDN/PCR in the OIDC request, it presents a MSISDN discovery page in SP app's external user agent to capture user's registered MSISDN.
6. The ID GW checks the standing of the MSISDN/Mobile Connect account being authenticated; if the mobile account is inactive/suspended, the ID GW rejects the SP authentication request (passing back the requisite error codes).
7. The ID GW initiates an AuthN request call via the SAA Adapter passing the parameters received in the original request along with MSISDN.
8. SAA Adapter generates a unique session id for the combination of SP client_id and state.
9. SAA Adapter determines the required AuthN level based on the acr_value passed in the SP request.
10. SAA Adapter retrieves the SAA Client ID corresponding to the MSISDN.
11. SAA Adapter utilises the SAA Server API to initiate an AuthN session with the SAA Client passing the relevant parameters (AuthN level, session id, details of SP [short name, SP logo URL, SP background image URL etc.,], context and SAA Client ID).
    (1) The SAA Server checks the validity of the SAA Client ID (e.g., whether or not it has been revoked due to Lifecycle events. See section 5.5.1 for lifecycle events integration with Mobile Connect).
    (2) If valid, SAA Server creates a new transaction record with unique transaction id for SAA Client ID; transaction id is returned to SAA Adapter.
    (3) SAA Adapter records the transaction id against the session id and marks the transaction in "Pending" state.
12. SAA Server waits for 'x' milliseconds before triggering a push message via the appropriate PNS:
    (1) The SAA Server must support invocation of the SAA Client using the network push model and also support local invocation of the SAA Client. Ideally, a special value in **prompt** parameter in the OIDC AuthZ code request will inform the ID GW to suppress network push for initiating user's AuthN challenge. At present, the Mobile Connect OIDC spec does this special value to be passed in the **prompt** parameter.

(2) As an alternative, the SAA Server waits for ‘x’ milliseconds before triggering a push message. During the wait time, if it receives an AuthN initiation request from SAA Client directly (in case of local invocation), then triggering of push message is suppressed. An edge case can be a delay in SAA Client initiating AuthN request and SAA Server meanwhile triggering a network push. This will not have any adverse impact on SAA Client, as SAA Client is already active and no user’s action is required to activate SAA Client.

13. On activation, SAA Client performs device security checks as described in section 5.2.1.
14. SAA Client retrieves private key, SAA Client Device ID from device secure key-store as described in section 5.2.2.
15. SAA Client retrieves pending transaction details (including transaction id) from SAA Server passing SAA Client Device ID.
16. SAA Client utilises the SAA Server API to initiate an AuthN session passing SAA Client Device ID and transaction id.
17. SAA Server validates incoming parameters and retrieves transaction record matching the transaction id.
18. SAA Server generates a unique challenge and returns an AuthN request message containing the challenge, transaction id, AuthN level, context and details of SP.
19. The SAA Client prompts the user for a ‘Click OK’ or to ‘Enter their PIN’ or ‘Fingerprint swipe’ or ‘Facial/Iris Scan’ depending on AuthN level.
20. On successful authentication, SAA Client generates a new assertion object containing device characteristics (locale, device locked/unlocked etc.) and signed challenge (using the private key).
21. SAA Client invokes an API in SAA Server to process AuthN response message passing the assertion object, SAA Client Device ID and transaction id.
22. SAA Server validates the AuthN response message and responds appropriately to SAA Client.
23. SAA Server returns the transaction id back to SAA Client.
24. SAA Client returns control back to the SP app’s user agent by invoking the custom URL of the SP app and passing the transaction id.
25. SAA Server invokes a call-back API in SAA Adapter to process AuthN response message passing the transaction id and assertion object.
26. SAA Adapter records the assertion object and marks the transaction as “Complete”; passes AuthN response back to ID GW containing the transaction id and session id.
27. The ID GW generates an OIDC response back to the SP:
    (1) The AuthZ code and state parameters are sent as query parameters in the OIDC response (through consumption device’s user agent) to the registered redirect_uri of SP.
    (2) The SP backend server implementing the redirect_uri retrieves the AuthZ code from the URI.
    (3) The SP backend server makes the token call passing the AuthZ code to get the access token and the ID Token which contains the PCR.
    (4) The SP backend server returns control back to the SP app’s external user agent.

5.4.3.5 Technical flow – User is not prompted to enter MSISDN in SP App

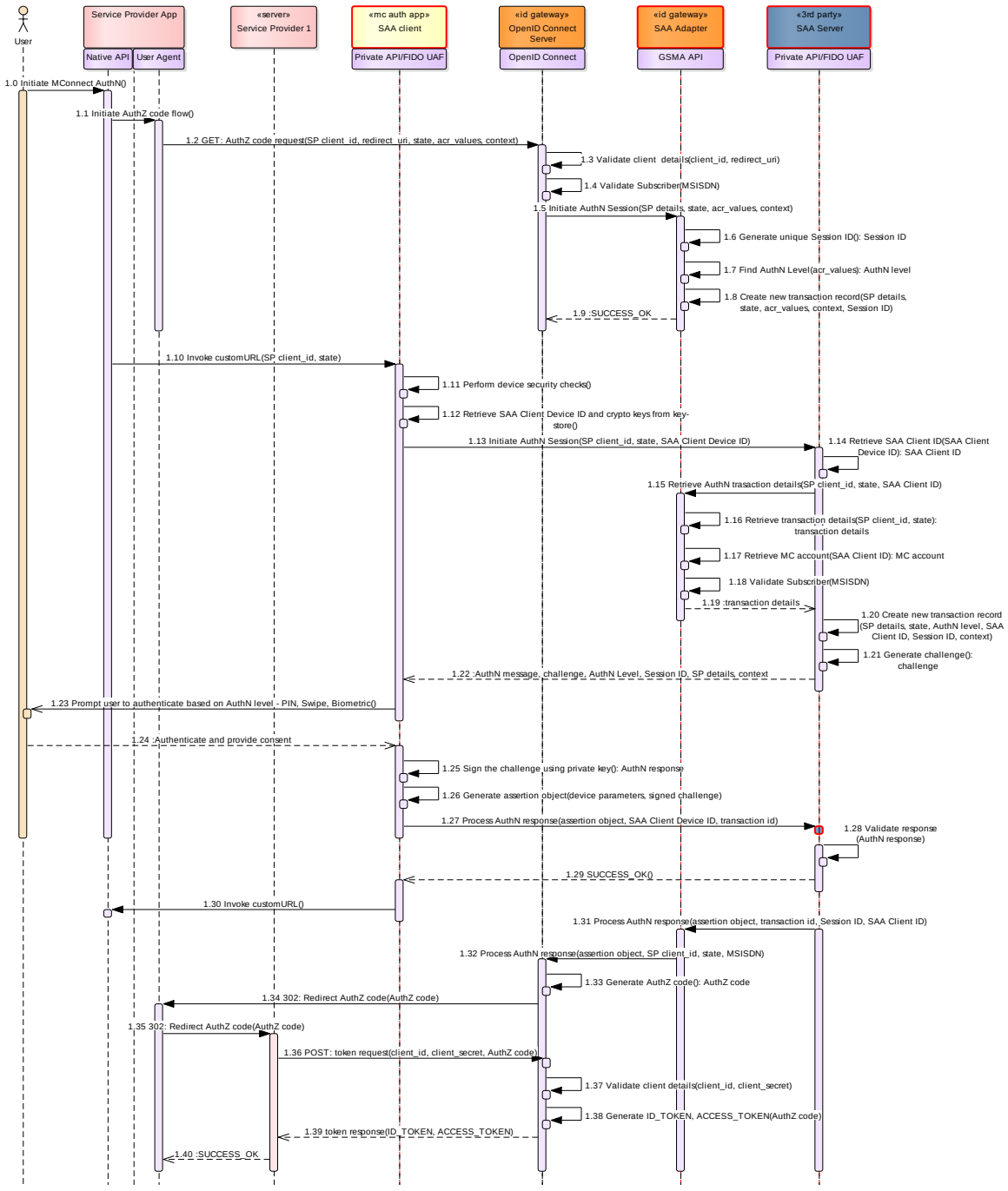

Figure 20: SAA Client local invocation technical flow (without MSISDN prompt in SP App)

1. User accesses SP app on the mobile device and needs to authenticate
   - The SP implementation uses a Discovery mechanism such as the API Exchange to identify the Operator and needed parameters
2. The SP app sends an OIDC AuthZ code request to the ID GW of the Operator through an external user agent passing SP client_id, state, context and other relevant

parameters. In parallel, the SP app invokes the custom URL of the SAA Client passing SP client_id and state parameters.

3. The ID GW validates the parameters passed in the request.
4. The ID GW is unable to retrieve the MSISDN from the incoming request. It initiates an AuthN request call via the SAA Adapter passing the parameters received in the original request **without the** MSISDN.

   Note: A change will be required in OIDC Mobile Connect profile to pass a new value in login_hint that will inform ID GW to suppress display of MSISDN page if unable to extract from login_hint and stop further MSISDN validation.
5. SAA Adapter generates a unique session id for the combination of SP client_id and state.
6. SAA Adapter records new transaction in “Pending” state along with incoming parameters and session id.
7. On activation, SAA Client performs device security checks as described in section 5.2.1.
8. SAA Client retrieves private key, and SAA Client Device ID from device secure key-store as described in section 5.2.2.
9. SAA Client utilises the SAA Server API to initiate an AuthN session passing SAA Client Device ID, SP client_id and state parameters.
10. SAA Server retrieves SAA Client ID matching SAA Client Device ID and validates it (e.g., whether or not it has been revoked due to Lifecycle events. See section 5.5.1 for lifecycle events integration with Mobile Connect).
11. On successful validation, SAA Server creates a new transaction record with unique transaction id for SAA Client ID; the SAA Server invokes an API in SAA Adapter to retrieve AuthN transaction details passing SAA Client ID, SP client_id, state and transaction id.
12. SAA Adapter retrieves the MSISDN matching SAA Client ID. It checks the standing of the MSISDN/Mobile Connect account being authenticated; if the mobile account is inactive/suspended, the SP authentication request is rejected (passing back the requisite error codes).
13. SAA Adapter retrieves the transaction details matching SP client_id and state (created in step 6); records transaction id and returns the transaction details to SAA Server (AuthN level, session id, details of SP [short name, SP logo URL, SP background image URL etc.,], context and SAA Client ID).
14. SAA Server generates a unique challenge and returns an AuthN request message containing the challenge, transaction id, AuthN level, context and details of SP.
15. SAA Client prompts the user to authenticate (using the appropriate user I/O based on the authenticator type that was selected; e.g., ‘Click OK’, ‘Enter PIN’, ‘Fingerprint swipe’, ‘Facial/Iris Scan’ etc.) dependent on AuthN level.
16. On successful authentication, SAA Client generates a new assertion object containing device characteristics (locale, device locked/unlocked etc.) and signed challenge (using the private key).
17. SAA Client invokes an API in SAA Server to process AuthN response message passing the assertion object, SAA Client Device ID and transaction id.

18. SAA Server validates the AuthN response message and responds appropriately to SAA Client passing transaction id for successful transaction.
19. SAA Client returns control back to the SP app's user agent by invoking the custom URL of the SP app and passing the transaction id.
20. In parallel, SAA Server invokes a callback API in SAA Adapter to process AuthN response message passing the transaction id and assertion object.
21. SAA Adapter retrieves transaction details matching the transaction id; records the assertion object and marks the transaction as "Complete"; passes AuthN response back to ID GW containing the transaction id and session id.
22. The ID GW generates an OIDC response back to the SP:
    (1) The AuthZ code and state parameters are sent as query parameters in the OIDC response (through consumption device's user agent) to the registered redirect_uri of SP.
    (2) The SP backend server implementing the redirect_uri retrieves the AuthZ code from the URI.
    (3) The SP backend server makes the token call passing the AuthZ code to get the access token and the ID Token which contains the PCR.
    (4) The SP backend server returns control back to the SP app's external user agent.

5.5 Mobile Connect lifecycle events

5.5.1 Lifecycle events integration with Mobile Connect

The Operator Systems (e.g. Provisioning Systems, CRM/CVM systems, HLR etc.) involved in any Lifecycle events need to notify user's account status to the Mobile Connect System (ID GW) so that an appropriate action (progressing or rejecting a request) can be taken when an AuthN request arrives at the ID GW for that user. Also, ID GW in turn will have to notify SAA Server through SAA Adapter to allow SAA Server to change the status of corresponding SAA Client ID accordingly.

The proposed integration options are:

1) The Operator's BSS sends a push notification of the event to the ID GW and a specific state is added in the Mobile Connect account database at the ID GW **(Base requirement)**.
   1. This can be done using a call-back/notification API implemented by the ID GW and called by the Operator systems.

2) The Operator systems expose an MSISDN validation API which is called by the ID GW:
   1. When an Authentication request arrives.
   2. Periodically polled and the Mobile Connect account database updated with the state.

3) Enabling the user to manually update their Mobile Connect account from the Operator's Mobile Connect website (or similar business process).

The following diagram shows an indicative architecture for the notification mechanism:

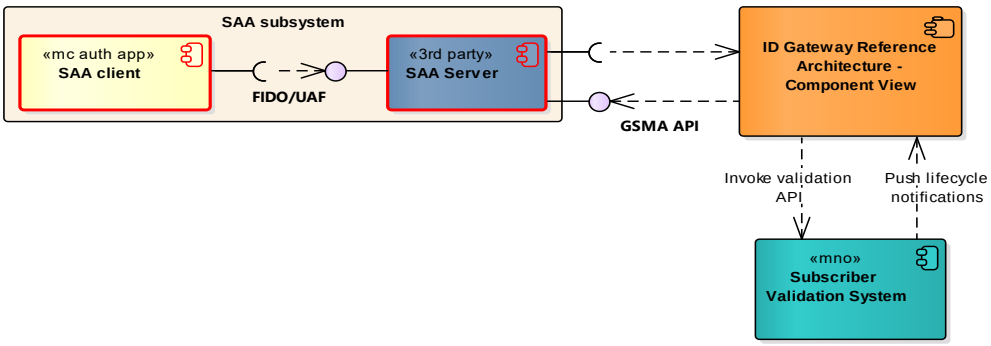

Figure 21: Lifecycle events integration with Mobile Connect

5.5.2 Handling of device change notification

The device change scenario should be managed as follows:

1. The ID GW marks the state of the Mobile Connect account for the MSISDN to "**device changed**".
2. The ID GW -> SAA Adapter removes the association of the SAA Client ID.
3. The ID GW-> SAA Adapter notifies the SAA Server to block the SAA Client.
4. The user is either sent an SMS with a link to download the SAA Client OR the user initiates the process using the Operator self-care portal.
5. The SAA Client activation process is performed using account recovery information (see section (1)).

5.5.3 Handling of Mobile account status notification

5.5.3.1 Mobile account suspended

- The SP should receive an error message indicating an error in accessing account details.
- Both the Mobile Connect account and SAA Client ID should be marked as '**Suspended**' in the ID GW and SAA Server respectively.
- The SP can then flag that user account as being unavailable via Mobile Connect and use an alternate method for authentication.
- The SAA Server needs to provide an API that the ID GW can invoke to notify the SAA Server of the account's suspended status.

5.5.3.2 Mobile account reactivation

- This should also reactivate the Mobile Connect account and SAA Client ID from the ID GW and SAA Server respectively.
- Both the Mobile Connect account and SAA Client ID should be marked as '**Active**' in the ID GW and SAA Server respectively.
- The current assumption is that the SP will not be notified explicitly when the account is reactivated so will need to discover this through issuing authentication requests.
- The SAA Server needs to provide an API that the ID GW can invoke to notify the account's reactivated status.

5.5.3.3 Mobile account deletion

- The SP should receive an error message indicating an error in accessing account details.
- Both the Mobile Connect account and SAA Client ID should be marked as '**Deleted**' in the ID GW and SAA Server respectively. The duration to keep the account in a dormant state before deleting is based on Operator policy.
- The SAA Server needs to provide an API that the ID GW can invoke to notify the account deletion status.

5.5.3.4 Technical flow

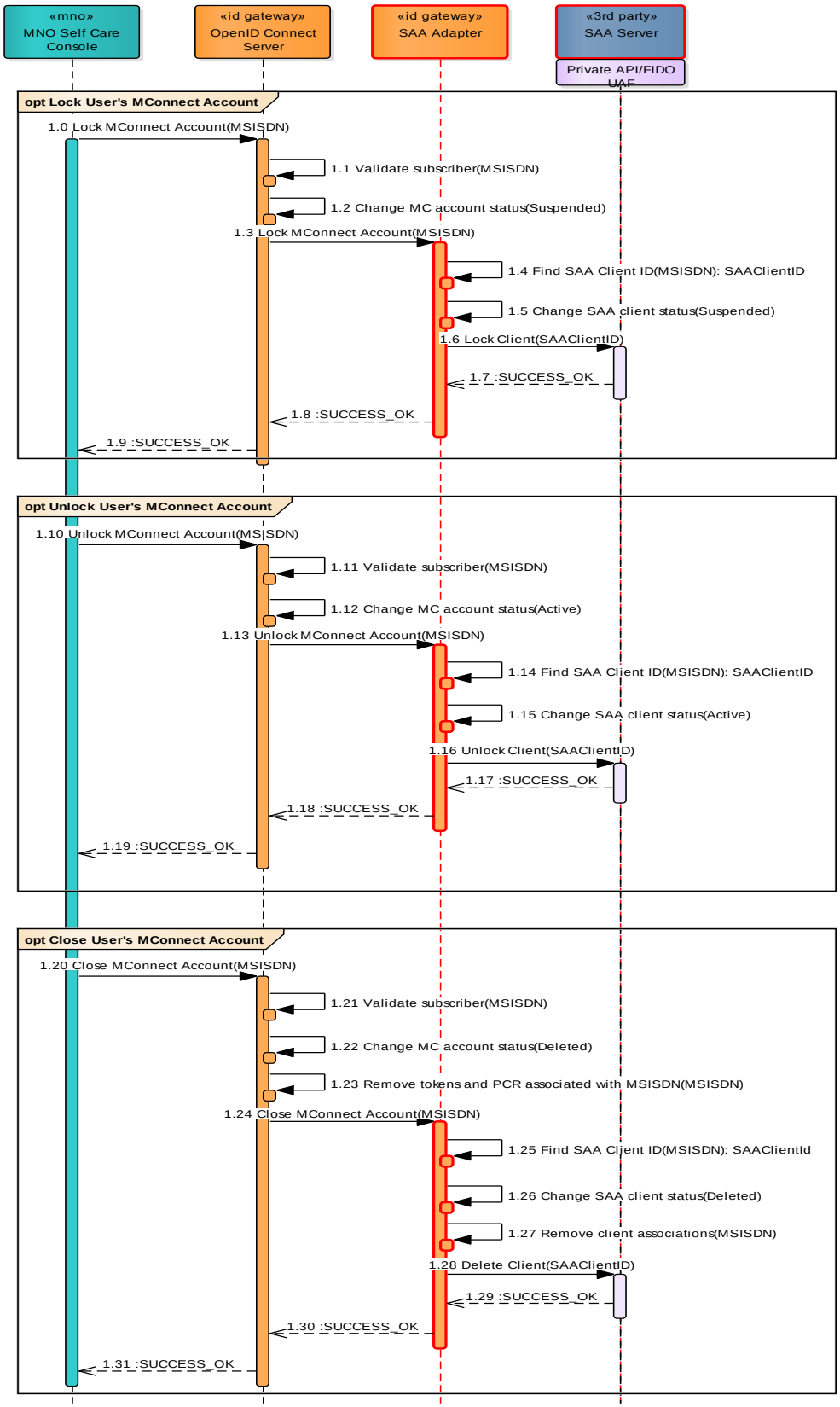

Figure 22: User's account suspension/reactivation/deletion technical flow

1. MNO self-care console notifies ID GW of mobile account status change (Blocked/Deleted/Reactivated).
2. ID GW validates the status of corresponding MC account.
3. ID GW changes the status of MC account accordingly (Blocked, Active, Deleted).
4. ID GW notifies SAA adapter to handle account status change.
5. SAA adapter modifies the status of SAA Client account accordingly (Blocked, Active, Deleted).
6. SAA adapter notifies SAA server through notification API to handle account status change for the SAA Client.
7. SAA server modifies the status of SAA Client account accordingly.

5.6 SP binding management

5.6.1 Technical flow

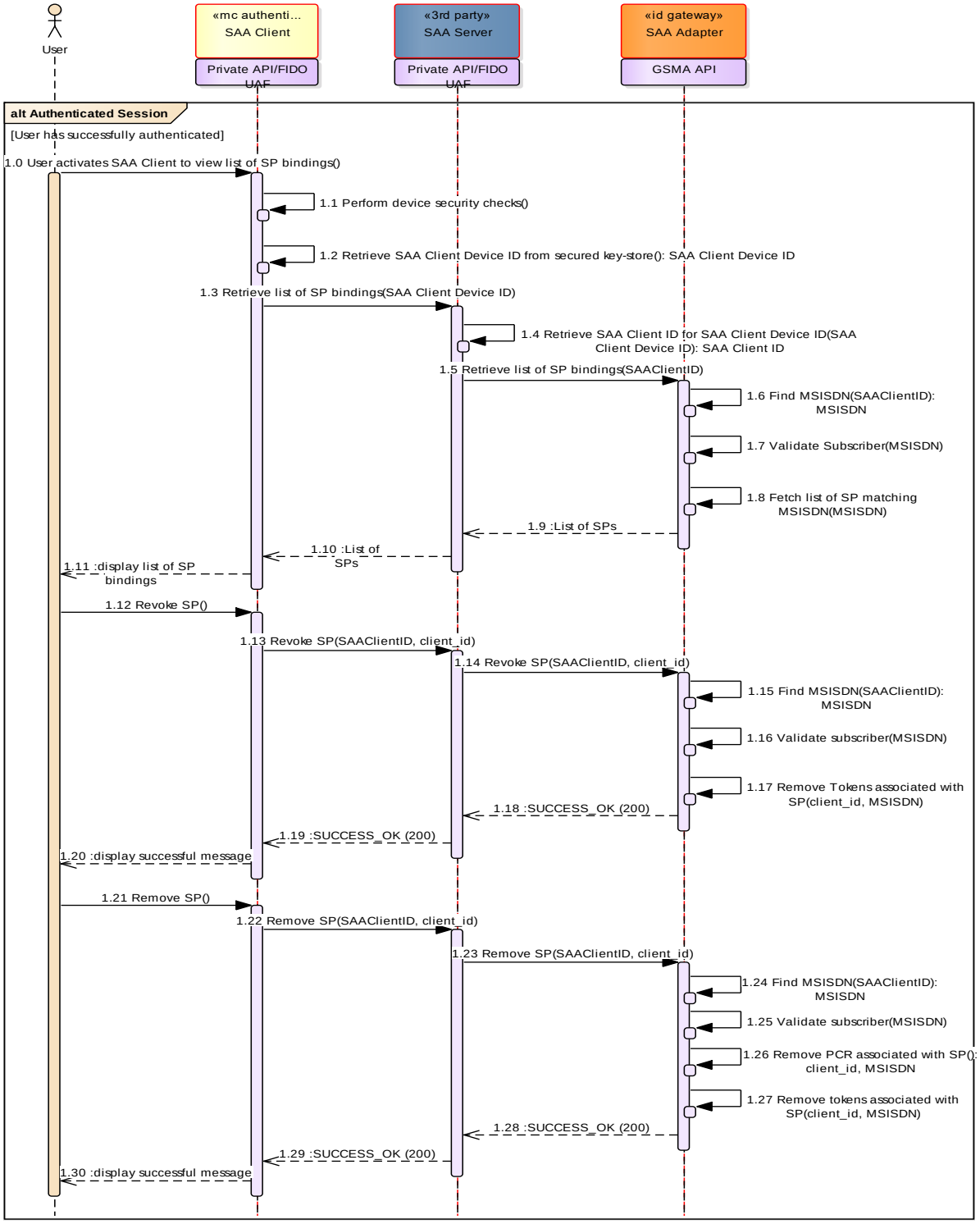

Figure 23: SP binding management technical flow

1. On activation, SAA Client performs device security checks as described in section 5.2.1.
2. User decides to review SP bindings in SAA Client.
3. SAA Client authenticates the user locally using the PIN.
4. SAA Client retrieves SAA Client Device ID from device secure key-store as described in section 5.2.2.
5. On successful authentication and validation, SAA Client invokes an API provided by the SAA Server to retrieve the list of SP bindings passing SAA Client Device ID and SAA Client ID.
6. SAA Server retrieves SAA Client ID matching SAA Client Device ID and validates it (e.g., whether or not it has been revoked due to Lifecycle events. See section 5.5.1 for lifecycle events integration with Mobile Connect).
7. On successful validation, SAA Server invokes corresponding API in SAA Adapter passing SAA Client ID.
8. SAA Adapter retrieves the MSISDN for SAA Client ID and validates it.
9. It returns a list of SP bindings back to SAA Client through SAA Server.
10. If the **user decides to revoke an SP**, the above validation steps are carried out again before SAA adapter removes user's OIDC tokens associated with the SP from ID GW database.
11. If the **user decides to remove an SP binding**, the above validation steps are carried out again before SAA Adapter removes user's OIDC tokens associated with the SP, removes user's PCR associated with the SP and deletes the SP binding record from ID GW database.

5.7 Secured messaging flows

5.7.1 Message template management

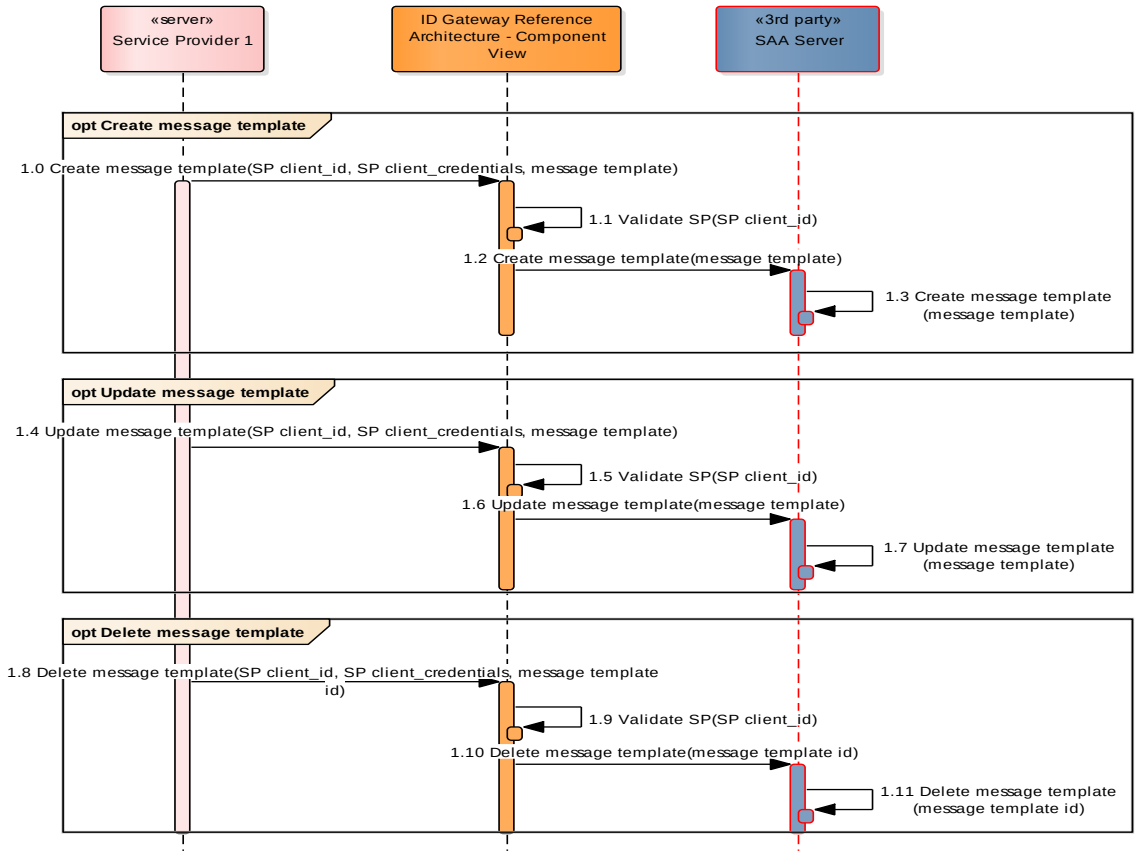

Figure 24: Message template management technical flow

1. SP invokes a private API in ID GW to either create or update or delete a message template.
2. ID GW checks the standing of SP status in ID GW (Blocked or Deleted) to ensure validity of SP.
3. ID GW delegates the API request to SAA Server for processing.
4. SAA Server either creates or updates or deletes the message template and responds accordingly to the calling client.

5.7.2 Send message

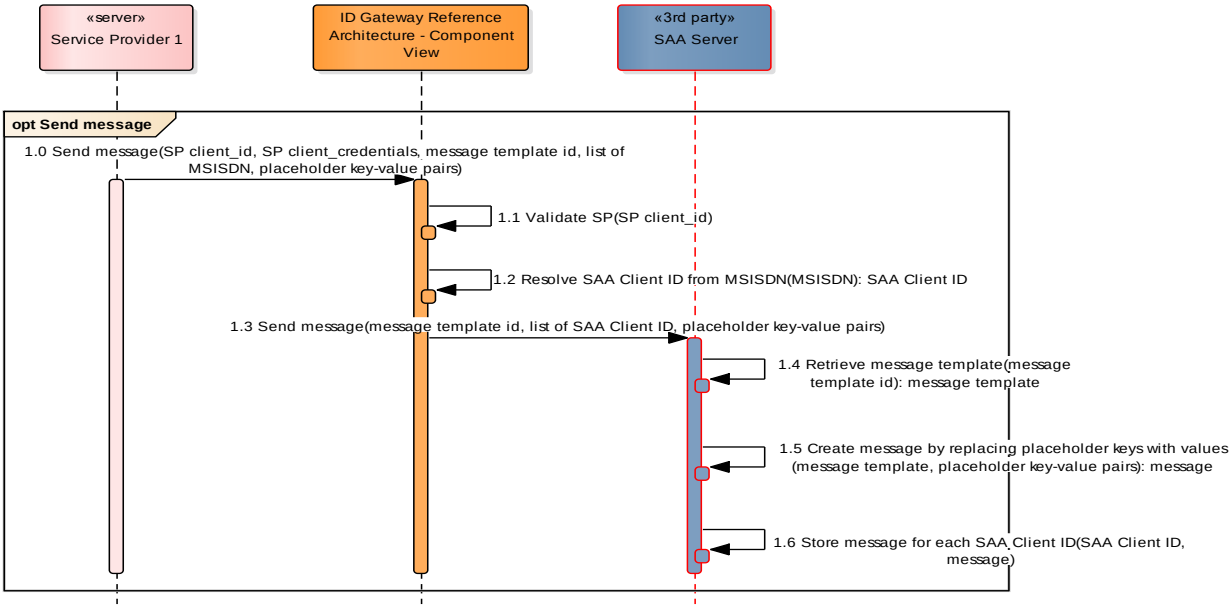

Figure 25: Send message technical flow

1. SP invokes a private API in ID GW to send a message to multiple recipients identified by MSISDN. The API request contains message template id, list of MSISDN and placeholder parameters (key-value pair).
2. ID GW checks the standing of SP status in ID GW (Blocked or Deleted) to ensure validity of SP.
3. ID GW resolves the list of MSISDN to retrieve corresponding SAA Client ID for each MSISDN.
4. ID GW invokes an API in SAA Server passing message template id, list of SAA Client IDs and placeholder parameters.
5. SAA Server retrieves the message template corresponding to message template id and replaces placeholder keywords with corresponding values.
6. SAA Server creates the message for each SAA Client ID in local database storage. The message will be retrieved when the user accesses message inbox interface on SAA Client.

5.7.3 Read message

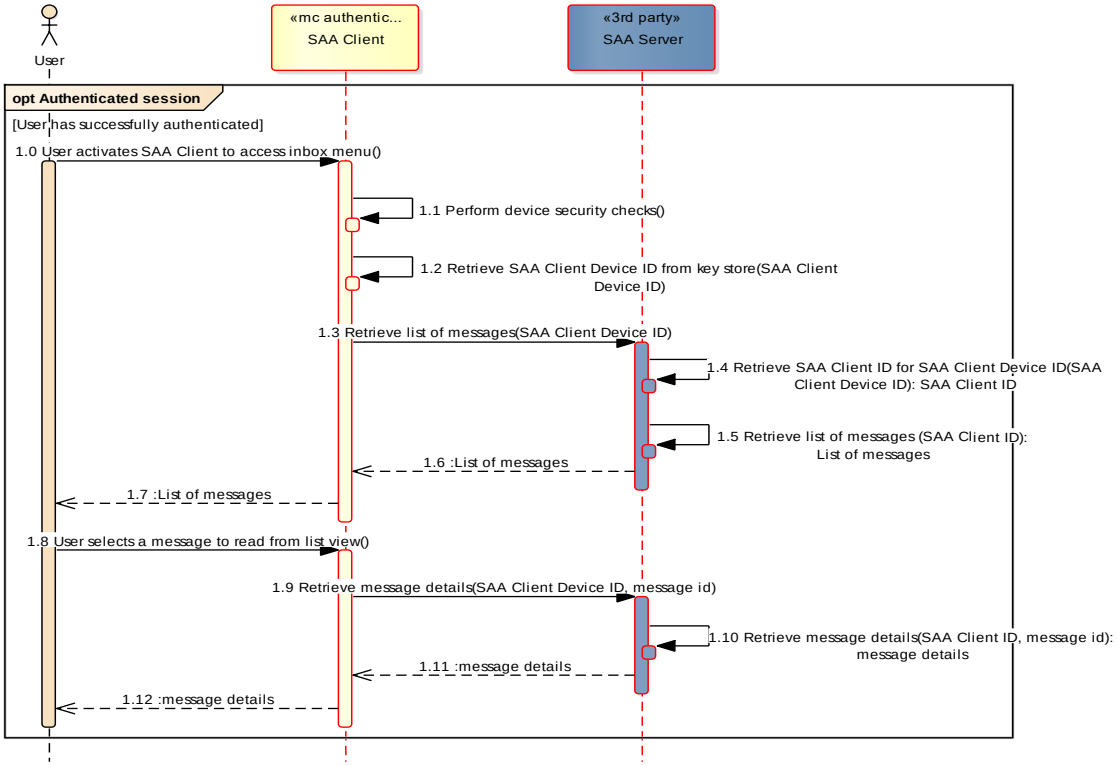

Figure 26: Read message technical flow

1. On activation, SAA Client performs device security checks as described in section 5.2.1.
2. User decides to review list of messages in SAA Client.
3. SAA Client authenticates the user locally using the PIN.
4. SAA Client retrieves SAA Client Device ID from device secure key-store as described in section 5.2.2.
5. On successful authentication and validation, SAA Client invokes an API provided by the SAA Server to retrieve the list of messages passing SAA Client Device ID.
6. SAA Server retrieves SAA Client ID matching SAA Client Device ID and validates it (e.g., whether or not it has been revoked due to Lifecycle events. See section 5.5.1 for lifecycle events integration with Mobile Connect).
7. On successful validation, SAA Server retrieves the list of messages for SAA Client ID from local database storage and returns the list to SAA Client.
8. User selects the message from the inbox to read.
9. SAA Client fetches the details of the message from SAA Server and renders it for display resolving the icon image from the public facing icon image URL.
10. On reading the message, SAA Client marks the message status as **“READ”** in SAA Server.

5.7.4 Delete message

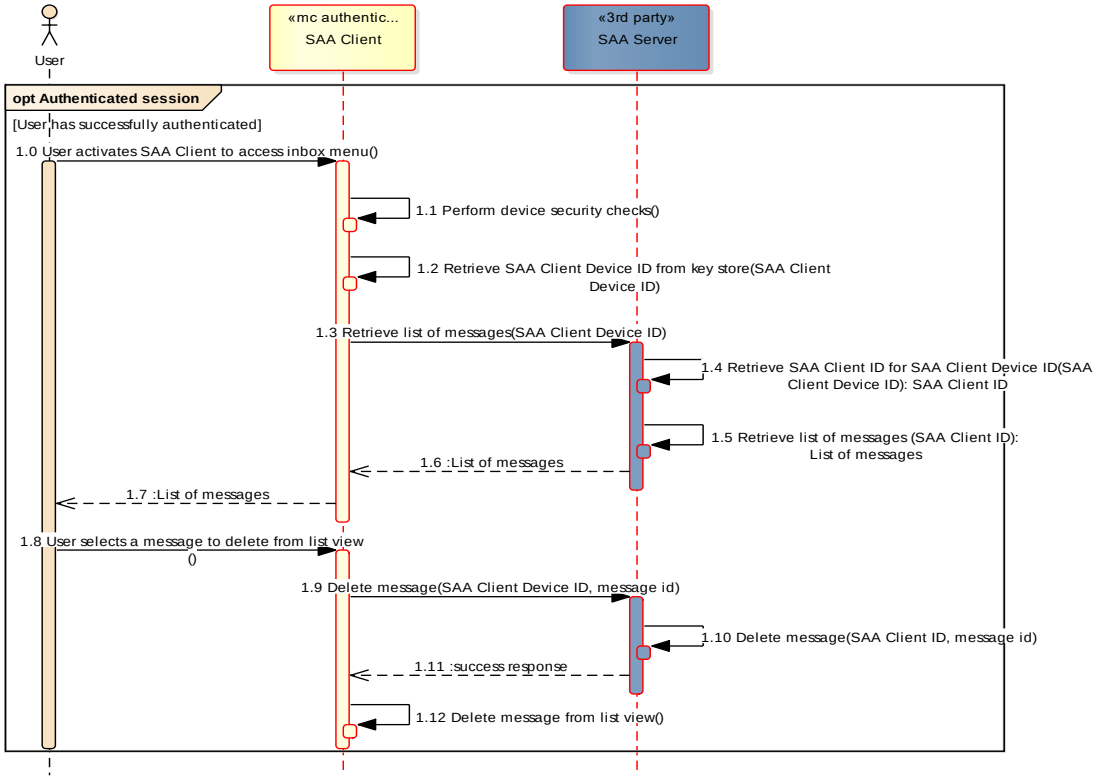

Figure 27: Delete message technical flow

1. On activation, SAA Client performs device security checks as described in section 5.2.1.
2. User decides to delete a messages from the inbox in SAA Client.
3. SAA Client authenticates the user locally using the PIN.
4. SAA Client retrieves SAA Client Device ID from device secure key-store as described in section 5.2.2.
5. On successful authentication and validation, SAA Client invokes an API provided by the SAA Server to delete the message passing SAA Client Device ID and message identifier.
6. SAA Server retrieves SAA Client ID matching SAA Client Device ID and validates it (e.g., whether or not it has been revoked due to Lifecycle events. See section 5.5.1 for lifecycle events integration with Mobile Connect).
7. On successful validation, SAA Server deletes the message for SAA Client ID identified by message identifier from local database storage and returns a successful response to SAA Client.
8. SAA Client removes the message from the list view.

5.8 API summary (Base and Enhanced SAAs)

The following table summarises all the APIs that have been outlined for the Base and Enhanced SAA solutions:

Calling Component	Provider Component	API Description
Base SAA		
SAA Adapter	SAA Server	Means for the ID GW to initiate an AuthN session with the SAA Server passing unique session id and other parameters: • Stipulate the preferred/required LoA (the SAA Client then selecting an authenticator based on locally-set user preferences) • Stipulate the authenticator type that the SAA must use in authenticating the user (e.g., PIN vs biometric) • Override any local user preference expressed on the device (in terms of which authenticator to use for a given LoA) This is the standard interface (INT1) between SAA Server and ID GW (SAA Adapter)
SAA Adapter	SAA Server	Means for the ID GW to notify the SAA Server of user's device change status to support Mobile Connect lifecycle events (see section 5.5.1)
SAA Adapter	SAA Server	Means for the ID GW to notify the SAA Server of user's account status change to support Mobile Connect lifecycle events (see section 5.5.1)
SAA Server	SAA Adapter	Means for the SAA Server to validate user's Mobile Connect account in ID GW
SAA Client	SAA Server	Means for the SAA Client to enrol/setup with the SAA Server using an **association code** for linking with the user's Mobile Connect account (see section 3.7.1.2)
SAA Client	SAA Server	Means for the SAA Client to recover an existing account using the **user's recovery information** (see section 3.7.2)
SAA Server	SAA Adapter	Means for the SAA Server to retrieve user's account details using account recovery information (see section 3.7.2)
SAA Server	SAA Adapter	Means for the SAA Server to retrieve user's account recovery information for display purpose (see section 3.7.2)
SAA Client	SAA Server	Means for the SAA Client to retrieve user's account recovery information for display purpose (see section 3.7.2)
SAA Client	SAA Server	Means for the SAA Client to retrieve pending transaction details (see section 5.4.1.2)
SAA Client	SAA Server	Means for the SAA Client to initiate AuthN session with SAA Server passing **server generated transaction id** (see section 5.4.1.2)
SAA Client	SAA Server	Means for the SAA Client to send AuthN challenge response after user's authentication on the device (see section 5.4.1.2)
SAA Server	SAA Adapter	Means for the SAA Server to send AuthN assertion to ID GW following user's AuthN challenge response (see section 5.4.1.2)
Enhanced SAA		
SAA Adapter	API Exchange	Update to the API Exchange Request Validator API for the ID GW to retrieve SP details (SP logo URL, SP background image URL, SP short name and SP name) by passing SP client_id. This information will be used by SAA Client for displaying transaction related context information (see section 4.2)
SP App	API Exchange	Update to the API Exchange's Discovery API to retrieve the Custom URL of Operator-specific SAA Client for invoking the SAA Client on the same device using app deep-linking method (see section 4.5)
SAA Client	Operator subsystem	Means for the SAA Client to retrieve one-time token representing user's MSISDN during SAA Client activation process (see section 5.3.1)
SAA Adapter	Operator subsystem	Means for the ID GW to retrieve MSISDN using one-time token during SAA Client activation process (see section 5.3.1)
SAA Adapter	Operator subsystem	Means for the SAA Adapter to retrieve network parameters (IMSI and IMEI) for a MSISDN during SAA Client activation process (see section 4.9.1)

SAA Client	ID GW/SAA Server	Means for the SAA Client to retrieve Operator's values including MVNO brand/sub-brand for a MSISDN for personalisation of SAA Client (see section 3.4)
SAA Client	ID GW/SAA Server	Means for the SAA Client to retrieve Operator's logo metadata JSON document for a MSISDN for personalisation of SAA Client (see section 3.4)
SAA Client	ID GW/SAA Server	Means for the SAA Client to retrieve Operator's logo for a public facing resource URL for personalisation of SAA Client (see section 3.4)
Old Operator ID GW	New Operator ID GW	Modifications to Mobile Connect lifecycle account migration API to support migration of SAA identifiers for user churn event (see section 4.4.1)
SAA Server	SAA Adapter	Means for the SAA Server to retrieve network parameters (IMSI and IMEI) for a MSISDN to compare against existing SAA Client Device ID token for detection of device/SIM changes and app clone characteristics (see section 5.2.1)
ID GW	SP	Means for the ID GW to return Confidence Score as an optional claim in ID token; requires an update to the OIDC Mobile Connect profile (see section 4.9.2)
SAA Client	SAA Server	Means for the SAA Client to setup/enrol with SAA Server using **MSISDN** for linking with user's Mobile Connect account
SAA Client	SAA Server	Means for the SAA Client to initiate AuthN session with SAA Server passing **SP client_id and state** (local invocation)
SAA Client	SAA Server	Means for the SAA Client to retrieve list of user's SP bindings to allow user to either remove or revoke a SP binding (see section 4.7)
SAA Client	SAA Server	Means for the SAA Client to revoke an user's SP binding (see section 4.7)
SAA Client	SAA Server	Means for the SAA Client to remove an user's SP binding (see section 4.7)
SAA Server	SAA Adapter	Means for the SAA Server to retrieve list of user's SP bindings from ID GW (see section 4.7)
SAA Server	SAA Adapter	Means for the SAA Server to revoke a user's SP binding in ID GW (see section 4.7)
SAA Server	SAA Adapter	Means for the SAA Server to remove a user's SP binding in ID GW (see section 4.7)
SAA Server	SAA Adapter	Means for the SAA Server to check if a user is already registered for Mobile Connect in ID GW (see section 5.3.1)
SP	ID GW	Means for the SP to create a new message template (see section 4.8.2)
SP	ID GW	Means for the SP to update a message template (see section 4.8.2)
SP	ID GW	Means for the SP to delete a message template (see section 4.8.2)
SP	ID GW	Means for the SP to send a message to multiple end users (see section 4.8.2)
ID GW	SAA Server	Means for the ID GW to create a new message template (see section 4.8.2)
ID GW	SAA Server	Means for the ID GW to update a message template (see section 4.8.2)
ID GW	SAA Server	Means for the ID GW to delete a message template (see section 4.8.2)
ID GW	SAA Server	Means for the ID GW to send a message to multiple end users (see section 4.8.2)
SAA Client	SAA Server	Means for the SAA Client to retrieve list of messages (see section 4.8)
SAA Client	SAA Server	Means for the SAA Client to retrieve details of a message (see section 4.8)
SAA Client	SAA Server	Means for the SAA Client to delete a message (see section 4.8)

Table 11: API summary (Base and Enhanced SAAs)

In summary, it can be seen that for the Base SAA there are no dependencies on either the GSMA platforms or Operator subsystems.

NOTE: for the Base SAA it is assumed that lifecycle events will be notified to the Mobile Connect system via a manual business process. These are vitally important to ensure security and mitigate risk.

For the Enhanced SAA, given its aims of enhancing the user experience and security of the SAA solution by more tightly integrating it with the Operator's network, there are inevitably many more dependencies on both GSMA platforms (e.g., API Exchange) and Operator

subsystems hence it is not expected to be ready for deployment until a later Mobile Connect Release.

6.0 Deployment considerations

6.1 SAA subsystem deployment options

The SAA vendor will typically provide the SAA Server and Client; the SAA Adapter will generally be developed by the Identity GW vendor in order to interface to the API exposed by the SAA Server.

Operators may choose to deploy their own SAA subsystems (SAA Client + SAA Server) or do so in collaboration with the other Operators in a given market to ensure homogeneity of the solution towards users and ease of porting. The options break down as follows:

1) Operators deploy different SAA subsystems (different vendors)
2) Operators deploy different SAA subsystem instances but from the same vendor
3) Operators deploy a single common SAA subsystem within a market (single vendor)

Each of these deployment options can then be broken down further to reflect different approaches for distributing the SAA Client:

- Operator integrated — Operator leverages SAA Client SDK to integrate functionality into one of their own apps (e.g., self-care); see section 6.2
- Operator own — Operator deploys own-branded SAA Client (resulting in multiple apps on the App Stores)
- Single — Single common SAA Client deployed by all Operators within a market (no Operator branding)
- Personalised — Single common SAA Client branded Mobile Connect and used by all Operators within a market but personalised upon activation to show Operator branding

The following table summarises the different combinations:

			SAA Client options			
	SAA Client	SAA Server	Single	Personalised	Operator own	Operator integrated
Option 1: Operators deploy different SAA subsystems	Different per Operator		No	No	Yes	Yes
Option 2: Operators deploy different SAA subsystem instances but from the same vendor	Either common or different per Operator	Different per Operator	Yes	Yes	Yes	Yes

Option 3: Operators deploy a single common SAA subsystem within a market	Common across Operators in a market	Yes	Yes	No	No

Table 12: SAA subsystem deployment options

The pros and cons of the various options are as follows:

Interface options	Pros	Cons
Option 1: Operators deploy different SAA subsystems	• Allows flexibility in selection of SAA subsystem from various vendors based on Operator choice and preference for a particular vendor • Any defects in SAA subsystem does not affect other Operators	• Requires Operator and/or Identity GW vendor to develop a bespoke adapter for integration with vendor's SAA Server (however, this only needs to be done once) • SAA Client cannot be re-used when user ports • Proliferation of Operator specific SAA Clients in the app stores thereby causing fragmentation and confusion for users • Operators have to negotiate separate contract with SAA subsystem vendors • It may not be possible to negotiate favourable pricing with SAA subsystem vendor (cannot leverage economies of scale across x-Operator footprint)
Option 2: Operators deploy different SAA subsystem instances but from the same vendor	• SAA Client can be re-used when user ports • Flexibility to deploy a common SAA Client x-Operator (optionally personalised by Operator) or separate Operator-specific SAA Clients • Common software upgrade and patches by SAA subsystem vendor for all the instances • Any advancement in supporting future authentication modes is available to all the Operators	• Requires Operator and/or Identity GW vendor to develop a bespoke adapter for integration with vendor's SAA Server (however, this only needs to be done once by the ID GW vendor and can then be re-used across Operator deployments) • Risk of proliferation of Operator specific SAA Clients in the App Stores thereby causing fragmentation and confusion for users • Operators have to negotiate separate contracts with the SAA subsystem vendor • Any defects in the SAA subsystem affects all the Operators
Option 3 (Recommended): Operators deploy a single common SAA subsystem within a market	• Single SAA Client for a given market with the option of Operator personalisation thereby avoiding confusion and fragmentation • SAA Client can be re-used seamlessly when user churns (no need to re-associate with the SAA Server)	• Requires Operator and/or Identity GW vendor to develop a bespoke adapter for integration with vendor's SAA Server (however, this only needs to be done once by the ID GW vendor and can then be re-used across Operator deployments) • Any defects in SAA subsystem affects all the Operators

	• Common software upgrade and patches by SAA subsystem vendor for single instance • Operators can negotiate favourable pricing with the SAA subsystem vendor (leveraging economies of scale) • Operators can negotiate a single contract with SAA subsystem vendor (or at least a common master agreement and individual terms as needed) • Any new authenticator supported in the SAA Client is available to all the Operators	

Table 13: Pros and Cons of SAA subsystem deployment options

Whilst all the options listed above are viable, it is strongly recommended that Operators within a given market utilise a common SAA Client. In doing so they will reduce confusion and fragmentation for the user (through providing a single app on the App Store) whilst delivering a consistent user experience x-Operator. Taking such an approach still allows the SAA Client to be personalised on activation to reflect the correct Operator (although such functionality will probably have a dependency on development at the ID GW/SAA Server hence will be limited to the Enhanced SAA and delivery in a later Mobile Connect Release).

An alternative is for each Operator to integrate the SAA Client functionality into their own [self-care] app – whilst this addresses the App Store fragmentation issue, it introduces a dependency on integration within one the Operator's existing apps which is likely to cause delay in bringing an SAA solution to market and could result in inconsistent user experiences x-Operator. More details on the SDK approach are included in the following section.

In terms of subsystem deployment, Operators can either deploy their own instances of the SAA Server (Option 2) or share a single deployment (Option 3). Taking the latter approach has the advantage of removing the need for the user to re-associate their SAA Client when churning to a new Operator.

6.2 SAA Client SDK

Operators can enhance their existing apps with SAA capabilities by integrating an SAA vendor SDK.

The SAA vendor SDK may not provide UI flows, but it will manage security and communication with the vendor's SAA Server. The SDK should also manage app integrity in terms of secured storage of crypto keys (as described in section section 5.2.2) and device security checks (as described in 5.2.1).

UI Considerations for Operator App to support SAA

1. The UI flows of the SAA (as described in section 3.1) will need to be incorporated into the Operator's existing app.

2. In case of remote invocation, the Operator app will be activated when the user clicks on the notification prompt. This behaviour is identical to a standalone SAA Client. The Operator app will need to register an authentication intent (depending on OS) for a specific push message and act accordingly. The app will have to handle consistent state while providing the desired experience; for example: restoring the state (previous screen) after the user completes the authentication action.
3. In case of local invocation, the Operator app will need to provide a custom URL for the calling SP app to invoke. Again the URL will be tied to a specific action (authentication action) of the Operator app. Under this scenario, the Operator app will have to handle consistent state while providing the desired experience; for example: restoring the state (previous screen) after the user completes the authentication action and returning control back to the SP app by invoking the custom URL of the SP app. Please see 5.4.3.2 for further implementation guidelines for Android and iOS OS.
4. The Operator app's SAA Client SDK will have to communicate with the SAA Server for SAA functionality. It is recommended that the SAA Server is deployed inside the same internal network as the Operator's server used by the Operator app to optimise QoS and reduce any latency.

6.3 Cost-benefit analysis for Base and Enhanced SAA solutions

Feature	Cost	Benefit
Base SAA		
Common help contents	Participating Operators in a given region will have to agree on common help information to be displayed in the 'Help' menu of SAA Client	It becomes easier for users to follow common help information (FAQ, Privacy, Help Contents etc.,). Also, common issues can be much easily identified in the FAQ page
App instance	Participating Operators within a market/region have to agree to deploy a single SAA Client app	By deploying a common SAA Client for a given market reduces fragmentation, provides better user experience when user churns and still allows for Operator personalisation
App discovery/ download	Operator can upgrade their existing device provisioning process and pre-embed SAA Client on the device	Allows easier distribution and quick adoption of SAA Client
Support Mobile Connect registration process	Operator will have to modify their website to allow user's Mobile Connect registration process	Allows user to register for a Mobile Connect account with confidence from a trusted and well-known Operator's portal
Support Mobile Connect lifecycle events	The Operator Systems (e.g. Provisioning Systems, CRM/CVM systems, HLR etc.) involved in any Lifecycle events need to make this information available to the Mobile Connect System (ID GW)	Allows the ID GW to check for lifecycle events and ensure the mobile account is in good standing before issuing an authentication/authorisation/consent request to the SAA Server
Enhanced SAA		

User Interface Features	Operators exposing an API to retrieve MCC + MNC values including participating MVNO's specific values for a MSISDN	Allows personalisation of MVNO brands/sub-brands as well
App instance	Operator can optionally leverage SAA Client SDK to integrate functionality into one of their own apps (e.g., self-care)	Allows easier distribution and quick adoption of SAA Client by wider Operator's user base
Security enhancements	Operator exposing network API to allow retrieval of device/SIM identifiers	Allows SAA Client device token to be bound to Operator's network validated device/SIM identifiers, thereby enhancing checks against device tampering and app cloning risks
Support MSISDN based SAA Client activation process	Operators exposing MSISDN discovery API	Allows seamless registration of Mobile Connect account from SAA Client, thereby reducing friction
Support User churn lifecycle event	Operators will have to support Mobile Connect Lifecycle Account Migration API	Allows users to seamlessly continue using Mobile Connect account and SAA when churning to a different participating Operator

Table 14: Cost-benefit analysis for Base and Enhanced SAA solutions

Annex A Further information

A.1 Security threats and prevention techniques

Please see the [3][60] for further reading and reference. The table below describes various security threats and possible prevention techniques:

Security Risk	Threat Agents	Security Weakness	Prevention Techniques
Weak server side controls (SAA Server)	Any entity that acts as a source of untrustworthy input to SAA Server API service. Examples of such entities include: a user, malware, or a vulnerable app on the mobile device	The exposed service or API call is implemented using insecure coding techniques that produce an OWASP Top Ten vulnerability within the server. Through the mobile interface, an adversary is able to feed malicious inputs or unexpected sequences of events to the vulnerable endpoint	Secure coding and configuration practices must be used on SAA Server
Insecure data storage	Agents that may exploit this vulnerability include the following: an adversary that has attained a lost/stolen mobile device; malware or other repackaged app acting on the adversary's behalf that executes on the mobile device	Insecure data storage vulnerabilities occur when development teams assume that users or malware will not have access to a mobile device's filesystem and subsequent sensitive information in data-stores on the device. Filesystems are easily accessible. Organizations should expect a malicious user or malware to inspect sensitive data stores. Rooting or jailbreaking a mobile device circumvents any encryption protections. When data is not protected properly, specialized tools are all that is needed to view application data	• Detection of rooted/jail broken device and disallow any further access during app start-up • Secured storage of crypto keys and secrets on mobile devices (as defined in section A.4.2) • Follow iOS and Android best practices for storing sensitive information on mobile devices
Insufficient transport layer protection	Data is commonly exchanged in a client-server fashion. When the solution transmits its data, it must traverse the mobile device's carrier network and the internet. Threat agents might exploit vulnerabilities to intercept sensitive data while it's traveling across the wire. The following threat agents exist: • An adversary that shares your local	Mobile applications frequently do not protect network traffic. They may use SSL/TLS during authentication but not elsewhere. This inconsistency leads to the risk of exposing data and session IDs to interception. The use of transport security does not mean the app has implemented it correctly. To detect basic flaws, observe the phone's network traffic. More subtle flaws require inspecting the design of the application and the applications configuration	• Assume that the network layer is not secure and is susceptible to eavesdropping • Apply SSL/TLS to transport channels that the mobile app will use to transmit sensitive information, session tokens, or other sensitive data to SAA Server • Avoid mixed SSL sessions as they may expose the user's session ID • Use strong, industry standard cipher suites with appropriate key lengths • Use certificates signed by a trusted CA provider

[60] https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Top_10_Mobile_Risks

	network (compromised or monitored Wi-Fi); • Carrier or network devices (routers, cell towers, proxy's, etc); or • Malware on your mobile device		• Never allow self-signed certificates, and consider certificate pinning for security conscious applications • Always require SSL chain verification • Only establish a secure connection after verifying the identity of the endpoint server using trusted certificates in the key chain • Alert users through the UI if SAA Client detects an invalid certificate • Apply a separate layer of encryption to any sensitive data before it is given to the SSL channel. In the event that future vulnerabilities are discovered in the SSL implementation, the encrypted data will provide a secondary defence against confidentiality violation • Follow iOS and Android best practices for establishing securing the transport
Unintended data leakage	Agents that may exploit this vulnerability include the following: mobile malware, modified versions of legitimate apps, or an adversary that has physical access to the victim's mobile device	Unintended data leakage occurs when a developer inadvertently places sensitive information or data in a location on the mobile device that is easily accessible by other apps on the device. Typically, these side-effects originate from the underlying mobile device's operating system	It is important to threat model operating systems, platforms, and frameworks, to see how they handle the following types of features: • URL Caching (Both request and response) • Keyboard Press Caching • Copy/Paste buffer Caching • Application backgrounding • Logging • HTML5 data storage • Browser cookie objects • Analytics data sent to 3rd parties
Broken cryptography	Agents that may exploit this vulnerability include the following: anyone with physical access to data that has been encrypted improperly, or mobile malware acting on an adversary's behalf	In order to exploit this weakness, an adversary must successfully return encrypted code or sensitive data to its original unencrypted form due to weak encryption algorithms or flaws within the encryption process	• Code obfuscation (white-box cryptography) and black-boxing to make the app as tamperproof as possible • Do not include the keys in the same attacker-readable directory as the encrypted content • Avoid the use of hardcoded keys within the binary • Avoid use of custom encryption protocols • Always use modern algorithms that are accepted as strong by the security community, and whenever possible leverage the state of the art encryption APIs within the mobile platform • Avoid use of insecure and/or deprecated algorithms

Client side injection	Consider anyone who can send untrusted data to the mobile app, including external users, internal users, the application itself or other malicious apps on the mobile device	Client-side injection results in the execution of malicious code on the mobile device via the mobile app	Follow iOS and Android best practices (secure coding techniques) to prevent client side injection
Security decisions via untrusted inputs (applicable to SAA Client local invocation method)	Threat agents include entities that can pass untrusted inputs to the sensitive method calls. Examples of such entities include, but are not limited to, users, malware and vulnerable apps	An attacker can intercept the calls (IPC or web service calls) and temper with such sensitive parameters. Weak implementation of such functionalities leads to improper behaviour of an app	In general try and adhere to the following IPC design patterns: • SAA Client should restrict access to a white-list of trusted SP applications • All input received from IPC entry points must undergo stringent input validation in order to prevent input driven attacks • Do not pass any sensitive information through IPC mechanisms, as it may be susceptible to being read by third party applications under certain scenarios
Improper session handling	Any mobile app with access to HTTP/S traffic, cookie data	Improper session handling occurs when the session token is unintentionally shared with the adversary during a subsequent transaction between the SAA Client and the SAA Server	• Always invalidate sessions on SAA Client and SAA Server • Allow adequate session timeout protection • Always reset session cookies during authentication state changes • To handle sessions properly, ensure that SAA Client code creates, maintains, and destroys session tokens properly over the life-cycle of a user's mobile app session
Lack of binary protections	Typically, an adversary will analyse and reverse engineer a mobile app's code, then modify it to perform some hidden functionality	A lack of binary protections results in a mobile app that can be analysed, reverse-engineered, and modified by an adversary in rapid fashion	The application must follow secure coding techniques for the following security components within SAA Client: • Jailbreak Detection Controls; • Checksum Controls; • Certificate Pinning Controls; • Debugger Detection Controls The app must adequately mitigate two different technical risks that the above controls are exposed to: • The organization building the SAA Client must adequately prevent an adversary from analysing and reverse engineering the app using static or dynamic analysis techniques • The SAA Client must be able to detect at runtime that code has been added or changed from what it knows about its integrity at compile

			time. The app must be able to react appropriately at runtime to a code integrity violation
Identity Fraud	Typically, an adversary will use victim's mobile number during SAA setup/enrolment process	Lack of subscriber (MSISDN) verification checks will lead to adversary performing authentication requests on behalf of the victim resulting in: • Privacy related and confidential data theft; • Unauthorized access and fraud; • Brand and trust damage; • Revenue loss and piracy;	SAA Server must use these techniques to mitigate the risk: • Use Operator API (if available) to validate MSISDN and device/SIM identifiers during SAA enrolment and authentication process • MSISDN verification using SMS based OTP/URL technique, if Operator API is not available • SAA Server must expose notification APIs to support Mobile Connect lifecycle events
Session hijack	Agents that may exploit this vulnerability include the following: mobile malware, malicious app (modified versions of legitimate SAA Client)	Session hijack can be in the form of: • Invocation of malicious app on receiving push notification message for initiating authentication process • Malicious app responding to SMS based MSISDN verification during SAA setup/enrolment process	• Use of prevention techniques identified under 'Lack of binary protections' risk
App cloning	Typically, an adversary will clone SAA Client from victim's device to own device	Cloned SAA Client can be malicious and can cause severe brand and trust damage	• Use a digital fingerprint of the device and SIM card, or other such mechanism, in order to detect device / SIM card changes • Use techniques identified under 'Network binding' section

Table 15: Security threats and prevention techniques

A.2 Comparison of SAA vs SIM applet from a Security and Fraud perspective

Extract from an analysis report of Mobile Connect authenticators (based on CPAS 4); Study conducted by: Validsoft, TeleSign, Dialog, GSMA + other members of SFRA

Authenticator	Transport Mechanism	LoA	Security Pros	Security Cons	Mitigations	Authenticator Rating (High/Medium/Low)
SIM Applet (3DES)	Mobile network	2/3	* Well understood dynamic PIN/password approach * Uses SIM as a secure element and a secure execution environment and builds on proven security model for telcos	Potentially standard imposter/DOS attack (imposter uses own phone) if the initial ID and entered MSISDN are not coupled within the system.	Augment MSISDN as user ID by another element requested from the user, which is captured during user registration (e.g. a "spam code", DOB etc.) or based on the context (e.g. make/model of the phone, location etc.) depending on the implementation.	Medium/High (assuming second element such as “spam code” or similar used - recommend excluding public data such as DOB) and High using PIN (LoA3)
				Account Takeover	ID proofing as part of robust business processes	
SIM Applet (AES or OATH OCRA)	Data / Mobile network	2/3	As above plus: * The Authenticator interactions and messages happen over an encrypted channel - both at the transport level and also at the application/messaging level- making MitM/MitB unlikely during authentication.	As above	As above	High (spam-code or equivalent and PIN used)
Smartphone app (PKI)	Data / Mobile network	2/3	* Builds on well understood PKI security model familiar to potential customers	Provisioning and enrolment	Robust process for PKI and using TEE (Trusted Execution Environment)	Medium (if PIN is used - LoA3)
				Reliance of security on the device	Usage of TEE	
				Device takeover	Require PIN / Robust business processes	

Table 16: Comparison of SAA vs SIM applet from a security and fraud perspective

A.3 FIDO-enabled SAA

In order for the SAA solution to support FIDO v1.0 UAF, the following options are applicable:

1. SAA Server is also a FIDO server and is FIDO compliant; SAA Client integrates a FIDO UAF client
2. SAA Server is also a FIDO server and is FIDO compliant; SAA Client interworks with FIDO UAF client resident on the target smartphone[61]
3. SAA Server is not FIDO compliant, but can still support FIDO client by integrating UAF library and integrating with a separate FIDO server

Note that the FIDO approach has the benefit of introducing a pluggable authenticator framework capable of supporting a variety of authenticators within the smartphone although the downside is the added complexity it introduces given that FIDO UAF is typically invoked client-side hence differs from the network-initiated norm of Mobile Connect.

Further consideration of how FIDO UAF may be integrated into the Mobile Connect infrastructure and the associated impacts on the role and function of an SAA Client and subsystem will be handled separately.

A.4 Future options

This annex identifies a number of areas where the SAA solution could be enhanced in the future but still require further study and consideration.

A.4.1 SP lifecycle notifications

As has been discussed throughout the document, there are a number of lifecycle events that will impact on the delivery of the Mobile Connect service and might therefore need to be communicated back to the SP within the OIDC response.

A list of potential lifecycle events are as follows (non-exhaustive):

- User changes MSISDN (same Operator)
  - Mandatory: Applicable to Service Providers who use an MSISDN in their OIDC Authorization requests; if the SP is not informed there is a risk that they will ask the wrong user to be authenticated
- Mobile account is suspended (e.g., unpaid; lost/stolen)
  - Optional: SP requests will be rejected with error code hence notification to the SP may not be that important (TBC)
- Mobile account is reactivated
  - Optional: an SP receiving an OIDC ‘reject’ response due to account suspension may black flag Mobile Connect for that user hence will need to be informed when the Mobile Connect account is live again; alternatively, the SP may just continue to issue Mobile Connect authentication requests ‘in case’ the Mobile Connect account has been reactivated
- Mobile account is deleted (e.g., unpaid; inactive)

[61] e.g., the Samsung Galaxy 6 onwards provide native FIDO client support embedded in the platform

    - Optional: SP requests will be rejected with error code indicating that Mobile Connect authentication is no longer available for that user
- User churns
    - Optional: SP will request authentication to the old Operator endpoints but the old mobile account (and Mobile Connect account) should be deactivated once the user churns; the error code returned will direct the SP to call Discovery to determine the endpoints for the new Operator

Note that many of these lifecycle events are generic to Mobile Connect and are discussed in the Mobile Connect Product Manager's Lifecycle Handbook [10].

A.4.2 Utilising the SIM as a secure key-store

Keys and the binding tokens (SAA Client ID, SAA Client Device ID) should be placed in secure storage (TEE or Secure Element) and cryptographic processes should be performed in secure environments.

Within this, depending on the SIM stock that has been deployed in-market, there may be an opportunity to leverage the SIM for supporting secure cryptographic processes. Doing so, exploits the best of both approaches:

- Smartphone app for delivering a rich user experience
- A SIM applet for secure storage of secrets and crypto execution

The OpenMobile API specified by the SIM Alliance provides a standardised interface between app and SIM but is yet to reach mass market.

The options open to SAA implementations are for further study.

Annex B Document Management

Version	Date	Brief Description of Change	Approval	Editor / Company
0.1	15/8/15	First draft created	David Pollington	Gautam Hazari/GSMA
0.2	31/8/15	Additional edits to incorporate use of UAF and FIDO as an implementation option	David Pollington	David Pollington GSMA
0.3	01/09/15	Extended the Interface options	David Pollington	Gautam Hazari/GSMA
0.4	22/10/15	Updated	David Pollington	Jim & David
0.5	17/11/15	Updated the setup and authentication mechanism	David Pollington	Gautam Hazari
0.6	20/11/15	Added requirements from SAA vendor session 30/10	David Pollington	Jim Small
0.63	10/12/15	Editorial changes and expansion of functional requirements	David Pollington	David Pollington
0.65	17/12/15	Further edit / expansion of functional requirements	David Pollington	Robert Blumenthal / Jim Small
0.66	29/03/16	New section on Security Guidelines, Technical Implementation Guidelines and rearranging the sections. Standardised terminology used in the document	David Pollington	Kamal Shah
0.67	8/4/16	Updated throughout plus a series of comments added identifying areas for further work	David Pollington	David Pollington GSMA
0.68	10/4/16	Updated some sections based on comments in v0.67	David Pollington	Kamal Shah, David Pollington
0.69		New user and technical flows Updated some sections based on comments in v0.68 New section on Device secure key-store for iOS and Android New section on App deep-linking implementation details New section on Public key cryptography Changes to MSISDN discovery logic Finalisation of User Prompts Changes to User Churn section	David Pollington	Kamal Shah, David Pollington
0.70	18/4/16	New section on SAA Client UI Features Completed API summary for Base/Enhanced SAA solutions Modified User Flow sections New section on Device security checks Modified sections after technical workshop with MeonTrust	David Pollington	Kamal Shah
0.71	25/4/16	Modified sections as per previous comments from David	David Pollington	Kamal Shah
0.73	27/04/16	Modifications to technical flows, sequence diagrams, rearranged some sections, updated API summary table	David Pollington	Kamal Shah, David Pollington
0.76	2/5/16	Additional edits for finalising first release	David Pollington	David Pollington
0.77	23/5/16	Modifications as per comments from various stakeholders	David Pollington	Kamal Shah
1.0	27/5/16	Final edits	David Pollington	David Pollington

1.1	8/8/16	Modifications as per feedback from Operators and vendors Inclusion of secure messaging support Reference to SAA API Spec	David Pollington	Kamal Shah
1.2	12/9/16	Modifications as per feedback from TEF (Cristina)	David Pollington	Kamal Shah
1.2	24/9/16	Replaced Mobile Connect Reference Architecture component view diagram to remove reference to Apigee	David Pollington	Kamal Shah
1.2.1	11/07/2019	Updated references as part of document refresh	David Pollington	Nick Spencer
1.2.1	06/12/2022	Go through TG approval	TG	Yolanda Sanz/GSMA

B.1 Other Information

Type	Description
Document Owner	IDG
Editor / Company	Yolanda Sanz / GSMA

It is our intention to provide a quality product for your use. If you find any errors or omissions, please contact us with your comments. You MAY notify us at prd@gsma.com

Your comments or suggestions & questions are always welcome.