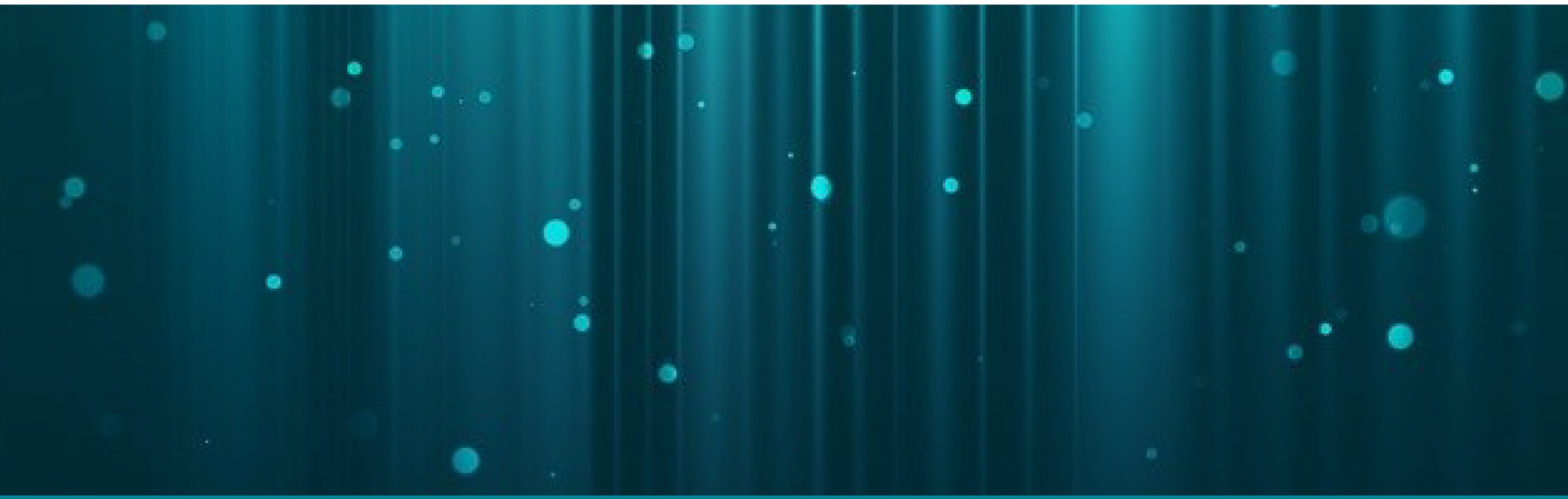
OneSpan



Orchestration SDK

Integration Guide

Version: 5.10

Contact us

Visit our website: **https://www.onespan.com**
Resource center: **https://www.onespan.com/resource-center**
Technical support and knowledge base: **https://www.onespan.com/support**

If there is no solution in the knowledge base, contact the company that supplied you with the OneSpan product.

Date: 2024-11-28

Contents

Figures

Tables

Procedures

Introduction

1

Welcome to the Orchestration SDK Integration Guide! This document describes how to integrate the Orchestration SDK.

This document provides information about:

- Features of the Orchestration SDK
- Exposed APIs
- Integration steps

What's new in the Orchestration SDK 2

This section provides an overview of changes introduced in the Orchestration SDK to facilitate the integration of the SDK and provide information on backward compatibility.

2.1 Version 5.10.1

2.1.1 Android

[INC0013466]-[MSS-9937] Issue with `StrongBoxUnavailableException` in Device Binding SDK

Recently, an issue with fingerprint generation arose in the Device Binding SDK, that is related to `StrongBoxUnavailableException`.

> **NOTE:** Please note that this is not an issue of our SDKs but is caused on the manufacturer's side and outside the control of OneSpan. There is the possibility that this may be fixed automatically in the future.

To avoid the `StrongBoxUnavailableException`, the Orchestration SDK internally checks for this exception from the Device Binding SDK and falls back to generating the fingerprint using *TEE*.

For more information, see the *[INC0013679]-[MSS-10210] Fallback mechanism for StrongBoxUnavailableException* entry in the What's New section of the *Device Binding SDK Integration Guide* (also available online at **Fallback mechanism for StrongBoxUnavailableException**).

2.2 Previous versions

2.2.1 Version 5.10.0

2.2.2 Android

Minimum supported version increased to Android 7 (API 24)

The minimum supported version of the SDK has been increased to Android 7 with API level 24.

2.2.3 iOS

Minimum supported version increased to iOS 14

The minimum supported version of the SDK has been increased to iOS 14.

2.2.4 Version 5.9.0

Android

Target API level increased to Android 14 (API 34)

The target version of the SDK has been increased to Android 14 with API level 34.

2.2.5 Version 5.8.1

iOS

Retrievable property removed

For the OneSpan Client Device Data Collector (CDDC) SDK, the *Keyboard identifier* property has been removed from the retrievable device and event properties for iOS due to a new limitation introduced by Apple.

Privacy manifest updated

The privacy manifest for the Orchestration SDK has been updated. The SDK can use geolocalization to track the user for analytic and security-related purposes. This applies to both coarse and precise geolocalization.

2.2.6 Version 5.8.0

All platforms

API improvement for crypto app indexes

The name and value names of the API crypto app indexes have been unified and now have the same names as used by Digipass:

- Name: `CryptoAppIndex` (instead of `CryptoApplicationIndex`)
- Value:
  - **Android**: `app<number>` or, on Java, APP_1, APP_2 etc. (instead of `FIRST, SECOND, THIRD etc.`)
  - **iOS** (Swift): `app<number>` (app1, app2 etc.) (instead of `index<number>`)

Android

CDDC keys deprecated (MSS-8259)

For the CDDC SDK, a number of properties and their corresponding setters have been deprecated and will be removed in a future version. With this, the following properties and setters will also be no longer available for the Orchestration SDK:

- Native code hooks
- Application repackaged
- Debugger attached

iOS

CDDC keys deprecated (MSS-8248)

For the CDDC SDK, a number of properties and their corresponding setters have been deprecated and will be removed in a future version. With this, the following properties and setters will also be no longer available for the Orchestration SDK:

- Rooting probability
- Root status
- Untrusted keyboard

- Untrusted screen-readers
- Native Code hooks

> **NOTE:** The `Root status (Key 15)` property and its corresponding setter has been renamed for iOS to *DeviceJailbroken*.

2.2.7 Version 5.7.0

Android

Updated `readme` file

The `readme` file in the product package has been improved to include more comprehensive steps and configuration options.

iOS

Bitcode support has been eliminated

Following the deprecation of Bitcode by Apple, we no longer support and have removed all Bitcode from the SDK framework.

2.2.8 Version 5.6.3

Android

Target API increased to 33

To meet new requirements for apps published on the Google Play Store, the target version of the SDK has been increased to Android 13 with API level 33. This change is needed to avoid APK rejection caused by security issues found with older API versions.

2.2.9 Version 5.6.2 (hotfix)

Access Group required for orchestration

On iOS, to avoid an issue reported by a customer related to the Device Binding Access Group, we introduced a change in the **implementation procedure**. As of this release, it is mandatory to provide an access group when creating an orchestration. For existing integrations, the `AppPrivate` access group must be used for activated users when creating an orchestration.

2.2.10 Version 5.6.0

Android

Target API increased to Android 12 (API 31)

To meet new requirements for apps published on the Google Play Store, the target version of the SDK has been increased to Android 12 with API 31 or higher. This change is needed to avoid APK rejection caused by security issues found with the older API versions.

Minimum supported version increased to Android 6 (API 23)

The minimum supported version has been increased to fully support new features and devices. Deprecated code has been replaced and simplified to be compatible with API 23 or higher.

iOS

Increased the minimum supported version to iOS 13

The minimum supported version has been increased to fully support all ARM64 devices and SwiftUI features. Deprecated code has been removed and replaced with code fully supporting iOS 13 or higher.

Fixed internal error conversion issues

Error codes converted between Objective-C and Swift need to have the `NSCustomError` setting implemented to display the correct value. Without this setting, the wrong codes could be displayed from the enum. The setting has been implemented in all the remaining error handling objects.

2.2.11 Version 5.5.1

Android

Maven repository

The Orchestration SDK and its Mobile Security Suite dependencies are now delivered exclusively through a local Maven repository. Due to this, you need to perform the following steps:

- Indicate the URL to the `maven` folder in the gradle as an available repository.
- Indicate the Orchestration SDK as a dependency with the following line:

  ```
  api 'com.onespan:sdk_orchestration_android:<version>'
  ```

Refer to the Android sample included in the product package for full code examples.

iOS

Change of framework name to *MSSOrchestration*

The framework name has been changed from *OrchestrationSDK* to *MSSOrchestration*. To use the new Objective-C API, replace all previous `OrchestrationSDK` imports with `#import <MSSOrchestration/MSSOrchestration.h>`.

API updates – NSErrors API support for Objective-C added

The Objective-C API points no longer throw an `NSException`. All API points that previously would throw such an `NSException` now require an `NSError` pointer. If an error occurs, it will be attached to the pointer that is provided as a parameter. Refer to the Objective-C sample included in the product package for full code examples.

Using the Orchestration SDK 3

The Orchestration SDK enables mobile developers to integrate the main features provided by OneSpan Mobile Security Suite.

3.1 Overview of the Orchestration SDK

The SDK consists of a client component. This component interprets the orchestration commands and executes them in a mobile application, fully hiding the complexity of the secure features' integration. The encoding and decoding of the orchestration commands is transparent to the mobile application.

Figure 1 illustrates an example of orchestration. It provides an overview of the **Remote Authentication** feature. If there is a risk for a given login request (e.g. unknown computer), the OneSpan Trusted Identity platform (TID) can dynamically request a step-up authentication on the mobile application using an authentication method (e.g. biometric recognition) that has been previously defined for that type of risk.

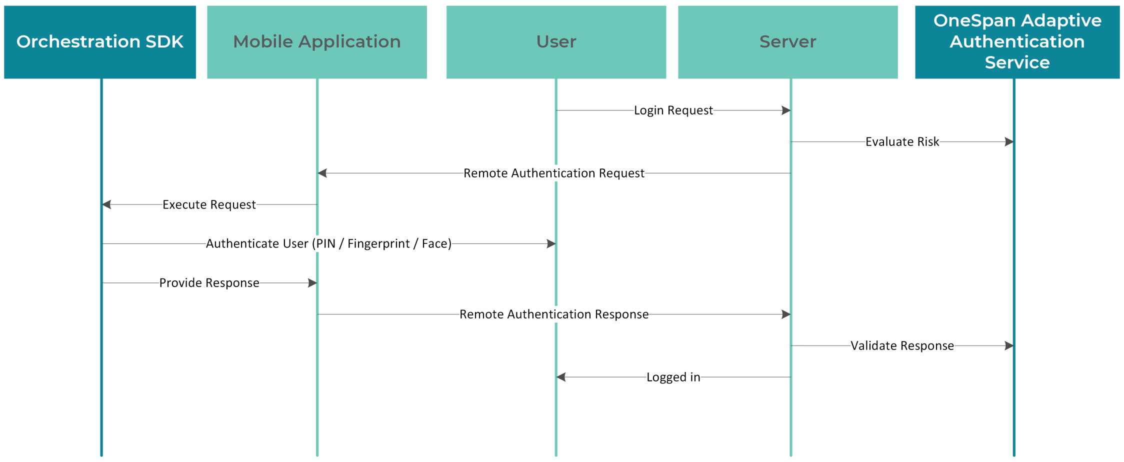

Orchestration SDK
Mobile Application
User
Server
OneSpan Adaptive Authentication Service
Login Request
Evaluate Risk
Remote Authentication Request
Execute Request
Authenticate User (PIN / Fingerprint / Face)
Provide Response
Remote Authentication Response
Validate Response
Logged in

Figure 1: Orchestration example – remote authentication overview

3.1.1 Supported platforms and requirements

The Orchestration SDK supports the following platforms for an integration in a mobile application:

Android devices:

- Minimum Android 7 (API level 24)
- Target Android 14 (API level 34)

iOS devices:

- iOS 14 or higher
- Swift 5.0 or higher
- Xcode 15 or higher

For more information about the Orchestration SDK features and functionalities, refer to **3.2 Features of the Orchestration SDK**.

3.2 Features of the Orchestration SDK

This chapter describes the main features and functionalities of the Orchestration SDK. The features are illustrated using workflow diagrams, and involve the following components:

- **Orchestration SDK**

> **NOTE:** For more information on the `users` and `device` commands, refer to the **TID Platform API Sandbox**.

- **Customer Mobile Application**. Your mobile application which integrates the Orchestration SDK.
- **User**. The individual using the Customer Mobile Application on their mobile device.
- **Customer Website**. Your website which users can access to perform sensitive actions (e.g. initiating an authentication request).
- **Customer Application Server**. Your server component which exposes Web services accessed by the Customer Mobile Application and the Customer Website.

> **NOTE:** Some server infrastructure considerations have been omitted on purpose in this document to ease the understanding of the features (e.g. front end/back end, database etc.).
>
> In the next sections, these points will be considered as properly integrated in the Customer Application Server.

- **OneSpan Trusted Identity platform**. A server component provided and hosted by OneSpan, which handles the server features of the Orchestration SDK.
- **Push Notification Service**. Notification service provided by Apple and Google, which can send push notification messages to Android and iOS devices.

3.2.1 Activation

With the activation feature of the Orchestration SDK you can provision authenticator instances to the Customer Mobile Application by securely exchanging activation data with OneSpan Trusted Identity platform. The activation process is initiated on the Customer Website, and then continued and completed using the Customer Mobile Application. Four network requests are required between the Customer Mobile Application and the Customer Application Server to complete the activation process.

The activation process will provision two authenticator instances in the Customer Mobile Application. For security reasons, each authenticator instance will be linked to a given authentication method: one instance for the password authentication method and another for the other authentication methods (No password and biometric recognition).

For more details concerning the authentication methods, see **3.2.7 Authentication methods**.

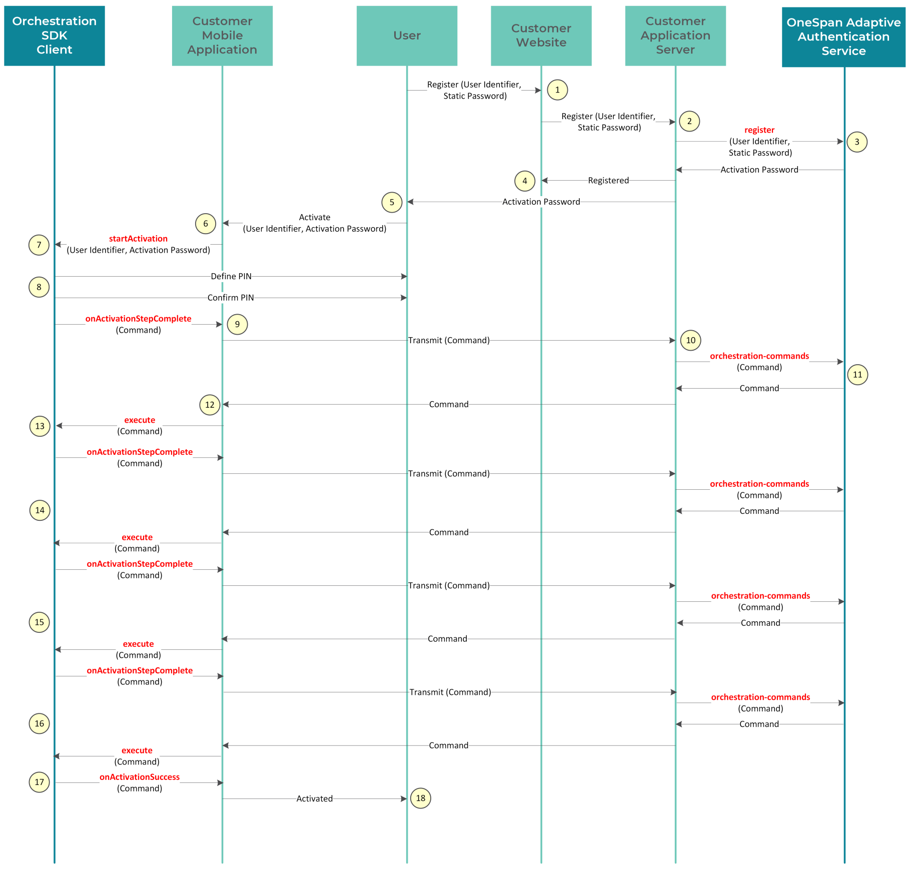

Orchestration SDK Client
Customer Mobile Application
User
Customer Website
Customer Application Server
OneSpan Adaptive Authentication Service
Register (User Identifier, Static Password)
1
Register (User Identifier, Static Password)
2
register (User Identifier, Static Password)
3
Activation Password
4
Registered
5
Activation Password
6
Activate (User Identifier, Activation Password)
7
startActivation (User Identifier, Activation Password)
Define PIN
8
Confirm PIN
onActivationStepComplete (Command)
9
Transmit (Command)
10
orchestration-commands (Command)
11
Command
12
Command
13
execute (Command)
onActivationStepComplete (Command)
Transmit (Command)
orchestration-commands (Command)
14
Command
Command
execute (Command)
onActivationStepComplete (Command)
Transmit (Command)
orchestration-commands (Command)
15
Command
Command
execute (Command)
onActivationStepComplete (Command)
Transmit (Command)
orchestration-commands (Command)
16
Command
Command
execute (Command)
17
onActivationSuccess (Command)
Activated
18

Figure 2: Activation workflow

▸ Activation workflow of OneSpan Orchestration SDK

1. The user initializes an activation session via the Customer Website, by providing their user identifier and static password.
2. The Customer Website transmits the user identifier and the static password to the Customer Application Server.
3. The Customer Application Server calls the **https://{tenant}.{environment}.tid.onespan.cloud/v1/users/register** endpoint from the OneSpan Trusted Identity platform API by providing their user identifier and the static password. In case of success, the web service returns the activation password.
4. The Customer Application Server notifies the Customer Website that registration has been completed.
5. The Customer Application Server transmits the activation password to the user via a channel other than the network (e.g. by sending an SMS message or an e-mail). The activation password remains valid for 10 minutes.
6. The user enters their user identifier and the activation password in the Customer Mobile Application to start the activation process.
7. The Customer Mobile Application calls the `startActivation` method of the Orchestration SDK to start the activation process.
8. The Orchestration SDK displays a virtual keypad or calls the password user authentication flow, where the user can define their password, and confirm it.
9. The Orchestration SDK builds the first orchestration command required for the activation process and transmits it to the Customer Mobile Application using the `onActivationStepComplete` method.
10. The Customer Mobile Application transmits the orchestration command to the Customer Application Server.
11. The Customer Application Server calls the `orchestration-commands` Web service of the OneSpan Trusted Identity platform by providing the orchestration command. A new orchestration command is returned as a result.
12. The Customer Application Server transmits the orchestration command to the Customer Mobile Application as a response to the previous request.

13. The Customer Mobile Application calls the `execute` method of the Orchestration SDK to continue the activation process.

14. A second activation step repeats *steps 9 to 13*.

15. A third activation step repeats *steps 9 to 13*.

16. A fourth activation step repeats *steps 9 to 13*.

17. The Orchestration SDK finalises the activation process and transmits the status to the Customer Mobile Application using the `onActivationSuccess` method.

18. The Customer Mobile Application notifies the user that the activation has been successfully completed.

> **NOTE:** **3.2.2 Notification registration** is required to finalize activation.

After a successful activation process, the Customer Mobile Application can use the other features of the Orchestration SDK (e.g. authentication, notification registration etc.).

See **5.2 Activation**for more information how to integrate this feature.

3.2.2 Notification registration

To be able to use push notification messages for remote authentication and remote transactions, the Customer Mobile Application must complete a notification registration process.

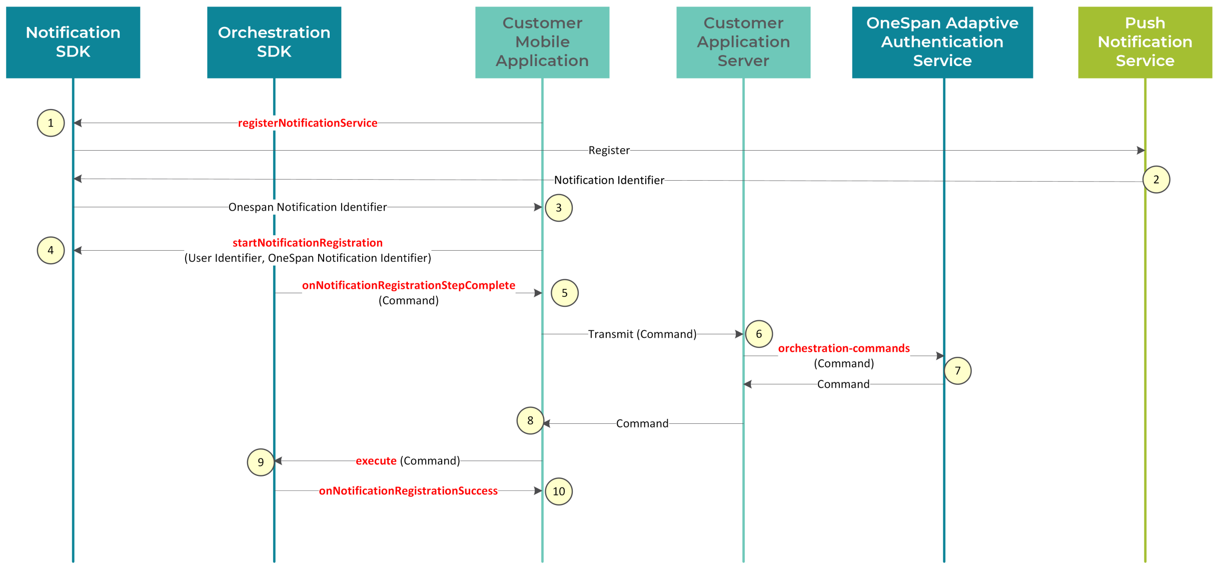
Notification registration consists in obtaining a notification identifier from the Push Notification Service, and in transmitting it to the OneSpan Trusted Identity platform via a dedicated orchestration command. The notification identifier is unique per device and per app.

For more information, refer to the *Notification SDK Integration Guide*.

The notification registration process can only be completed if the Customer Mobile Application has been activated (see **3.2.1 Activation**); this process must take place after a successful activation (mandatory for finalizing the activation flow on the server-side) and at application startup (only if the notification identifier has changed).

CAUTION: Notification registration is mandatory for finalizing the activation process on the server-side.

Figure 3 illustrates the notification registration workflow.



Notification SDK
Orchestration SDK
Customer Mobile Application
Customer Application Server
OneSpan Adaptive Authentication Service
Push Notification Service
1 registerNotificationService
Register
2 Notification Identifier
3 Onespan Notification Identifier
4 startNotificationRegistration (User Identifier, OneSpan Notification Identifier)
5 onNotificationRegistrationStepComplete (Command)
6 Transmit (Command)
7 orchestration-commands (Command)
Command
8 Command
9 execute (Command)
10 onNotificationRegistrationSuccess

Figure 3: Notification registration workflow

▸ Notification registration workflow

> **NOTE:** New APIs are created for Swift users of the iOS SDK. For more information, refer to the Xcode API documentation on `NotificationRegistrationDelegate` for this workflow.

1. The Customer Mobile Application calls the `registerNotificationService` method of the Notification SDK to obtain a notification identifier.

2. The Notification SDK contacts the Push Notification Service to obtain a notification identifier.

3. The Notification SDK returns to the Customer Mobile Application a **OneSpan Notification Identifier**, which is an abstraction of the previously retrieved notification identifier. For more information, refer to the *OneSpan Notification SDK Integration Guide*.

4. The Customer Mobile Application calls the `startNotificationRegistration` method of the Orchestration SDK to start the notification registration process.

5. The Orchestration SDK builds an orchestration command which includes the OneSpan Notification Identifier and transmits it to the Mobile Application Server using the `onNotificationRegistrationComplete` method.

6. The Customer Mobile Application transmits the orchestration command to the Customer Application Server.

7. The Customer Application Server calls the `orchestration-commands` Web service of the OneSpan Trusted Identity platform by providing the orchestration command. A new orchestration command is returned as a result.

8. The Customer Application Server transmits the orchestration command to the Customer Mobile Application as a response to the previous request.

9. The Customer Mobile Application calls the `execute` method of the Orchestration SDK to finalize the notification registration process.

10. The Orchestration SDK calls the `onNotificationRegistrationSuccess` method to notify the Customer Mobile Application.

After a successful notification registration process, the OneSpan Trusted Identity platform is able to send push notification messages to the Customer Mobile Application.

The Customer Mobile Application must integrate the other methods of the Notification SDK to properly receive and process the push notification messages sent by the OneSpan Trusted Identity platform.

For more information about integrating this feature, see **5.3 Notification registration**.

3.2.3 Remote authentication

With the remote authentication feature of the Orchestration SDK, the user can authenticate to the Customer Website using the Customer Mobile Application, via an authentication request initiated on the OneSpan Trusted Identity platform, and based on the corresponding risk evaluation.

The remote authentication process is initiated using the Customer Website and evaluated for risk by OneSpan Trusted Identity platform before continuing with the Customer Mobile Application.

The authentication request is embedded in an orchestration command and can be transmitted via a push notification message initiated by OneSpan Trusted Identity platform or by another communication channel handled by the Customer Website (e.g. image scanning).

The authentication request contains the following parameters:

- A *session identifier* created by the Customer Application Server, which uniquely identifies the authentication session.
- A *request identifier* created by the Customer Application Server, which uniquely identifies the authentication request.
- An *authentication method*, which defines how the user must authenticate to sign the authentication request (see **3.2.7 Authentication methods**).
- (OPTIONAL) *Data to display* on the Customer Mobile Application to provide authentication request information to the user; the user can choose to approve or reject it.

  The data to display is a string defined by the Customer Application Server, which must be interpreted by the Customer Mobile Application.

Figure 4 illustrates the remote authentication workflow with the transmission of the authentication request via a push notification message.

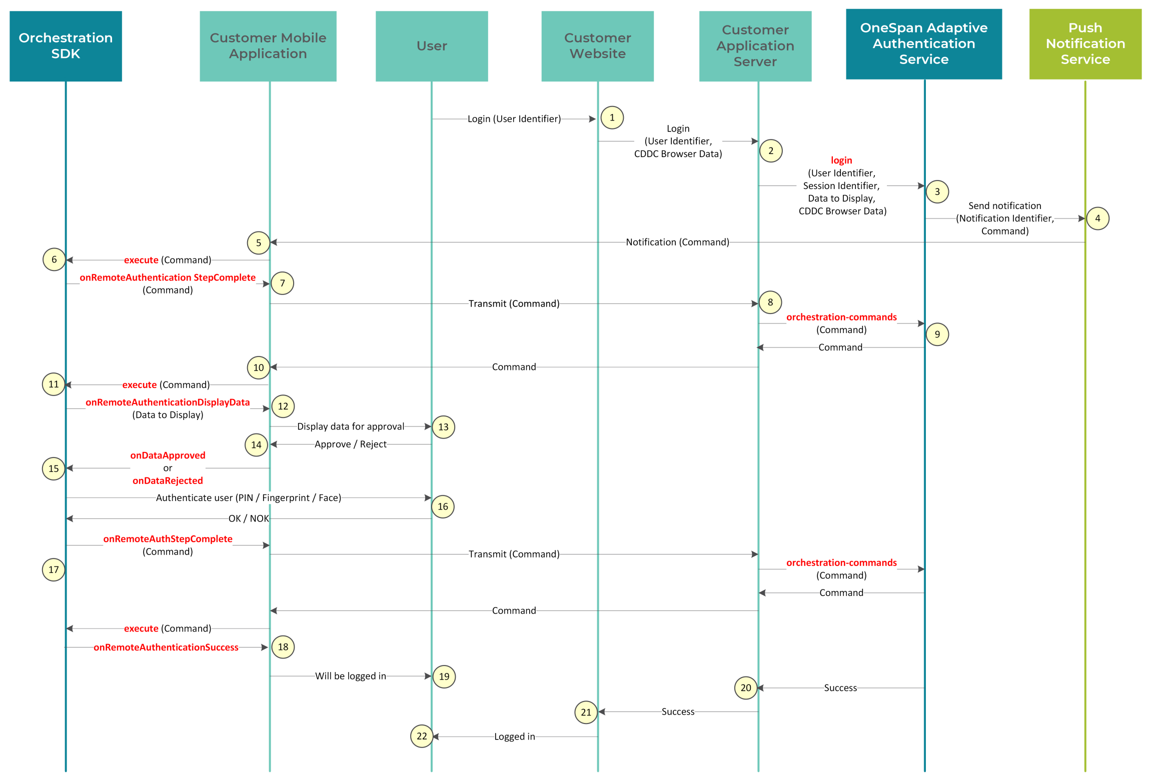

Orchestration SDK
Customer Mobile Application
User
Customer Website
Customer Application Server
OneSpan Adaptive Authentication Service
Push Notification Service
Login (User Identifier)
Login (User Identifier, CDDC Browser Data)
login (User Identifier, Session Identifier, Data to Display, CDDC Browser Data)
Send notification (Notification Identifier, Command)
Notification (Command)
execute (Command)
onRemoteAuthentication StepComplete (Command)
Transmit (Command)
orchestration-commands (Command)
Command
Command
execute (Command)
onRemoteAuthenticationDisplayData (Data to Display)
Display data for approval
Approve / Reject
onDataApproved or onDataRejected
Authenticate user (PIN / Fingerprint / Face)
OK / NOK
onRemoteAuthStepComplete (Command)
Transmit (Command)
orchestration-commands (Command)
Command
Command
execute (Command)
onRemoteAuthenticationSuccess
Will be logged in
Success
Success
Logged in

Figure 4: Remote authentication workflow

▸ Remote authentication workflow

1. The user initializes an authentication request via the Customer Website (e.g. for login purposes), providing their user identifier.
2. The Customer Website transmits the user identifier and the Client Device Data Collector (CDDC) browser data to the Customer Application Server.
3. The Customer Application Server calls the **https://{tenant}.{environment}.tid.onespan.cloud/v1/users/{userID@domain}/login** endpoint from the OneSpan Trusted Identity platform API by providing their user identifier, a session identifier (dynamically generated and uniquely identifying the authentication request), the received CDDC browser data, and, optionally, the data to display on the Customer Mobile Application.
4. The OneSpan Trusted Identity platform evaluates the risk related to the Web browser used for the authentication request (based on multiple parameters

provided in the previous step and on existing parameters related to the user, e.g. unknown country) and, in case of risk detection, initiates a step-up authentication request on the Customer Mobile Application with a given authentication method (see **3.2.7 Authentication methods**).

Depending on the configuration defined in the OneSpan Trusted Identity platform, multiple scenarios are possible:

- The authentication request can be transmitted via a push notification message initiated by the OneSpan Trusted Identity platform and sent by the Push Notification Service to the Customer Mobile Application. In this case, a push notification message is sent to all mobile devices of the user where the Customer Mobile Application is installed and activated.

- The authentication request can be transmitted by the Customer Application Server via a different channel (e.g. display a Cronto image containing the orchestration command related to the authentication request and scan it with the Customer Mobile Application).

- The authentication request can be blocking or non-blocking.

The following steps describe a blocking scenario with the authentication request transmitted via push notification.

5. The Push Notification Service sends a push notification message containing the orchestration command related to the authentication request to the Customer Mobile Application.

6. The Customer Mobile Application obtains the orchestration command contained in the push notification message and calls the `execute` method of the Orchestration SDK to perform the remote authentication.

7. The Orchestration SDK builds an orchestration command and transmits it to the Customer Mobile Application using the `onRemoteAuthenticationStepComplete` method.

8. The Customer Mobile Application transmits the orchestration command to the Customer Application Server.

9. The Customer Application Server calls the `orchestration-commands` Web service of the OneSpan Trusted Identity platform by providing the orchestration command. A new orchestration command is returned as a result.

10. The Customer Application Server transmits the orchestration command to the Customer Mobile Application as a response to the previous request.
11. The Customer Mobile Application calls the `execute` method of the Orchestration SDK to continue the remote authentication process (only if the authentication request is still pending).
12. The Orchestration SDK calls the `onRemoteAuthenticationDisplayData` method to transmit the data to display to the Customer Mobile Application (if data has been defined in *step 3*).
13. The Customer Mobile Application displays a screen to the user containing the data to display and two buttons to approve or reject the authentication request.
14. The user must approve or reject the authentication request, according to the displayed data.
15. Based on the user's decision in the previous step, the Customer Mobile Application calls the `onDataApproved` or `onDataRejected` method of the Orchestration SDK.
16. In both cases, the Orchestration SDK prompts the user to authenticate by using an authentication method defined by the OneSpan Trusted Identity platform, based on the evaluated risk (see *step 4*).
17. The Orchestration SDK signs the authentication request, builds an orchestration command, and transmits it to the Customer Mobile Application using the `onRemoteAuthenticationStepComplete` method.
18. Repeat *steps 8 to 11*.
19. In case of validation by the OneSpan Trusted Identity platform, the Orchestration SDK calls the `onRemoteAuthenticationSuccess` method to notify the Customer Mobile Application.
20. The Customer Mobile Application notifies the user that the authentication request has been successful and that they are about to be logged in to the Customer Website.
21. The OneSpan Trusted Identity platform provides a response to the call to the `login` Web service from *step 3*, indicating the success of the authentication request.
22. The Customer Application Server transmits the success status to the Customer Website. The user is now logged in to the Customer Website.

For more information about integrating this feature, see **5.4 Remote authentication**.

3.2.4 Remote transaction

With the remote transaction feature of the Orchestration SDK, the user can perform a transaction to the Customer Website using the Customer Mobile Application, via a transaction request initiated on OneSpan Trusted Identity platform, and based on the corresponding risk evaluation.

The remote transaction process is initiated using the Customer Website and evaluated for risk by the OneSpan Trusted Identity platform before continuing with the Customer Mobile Application.

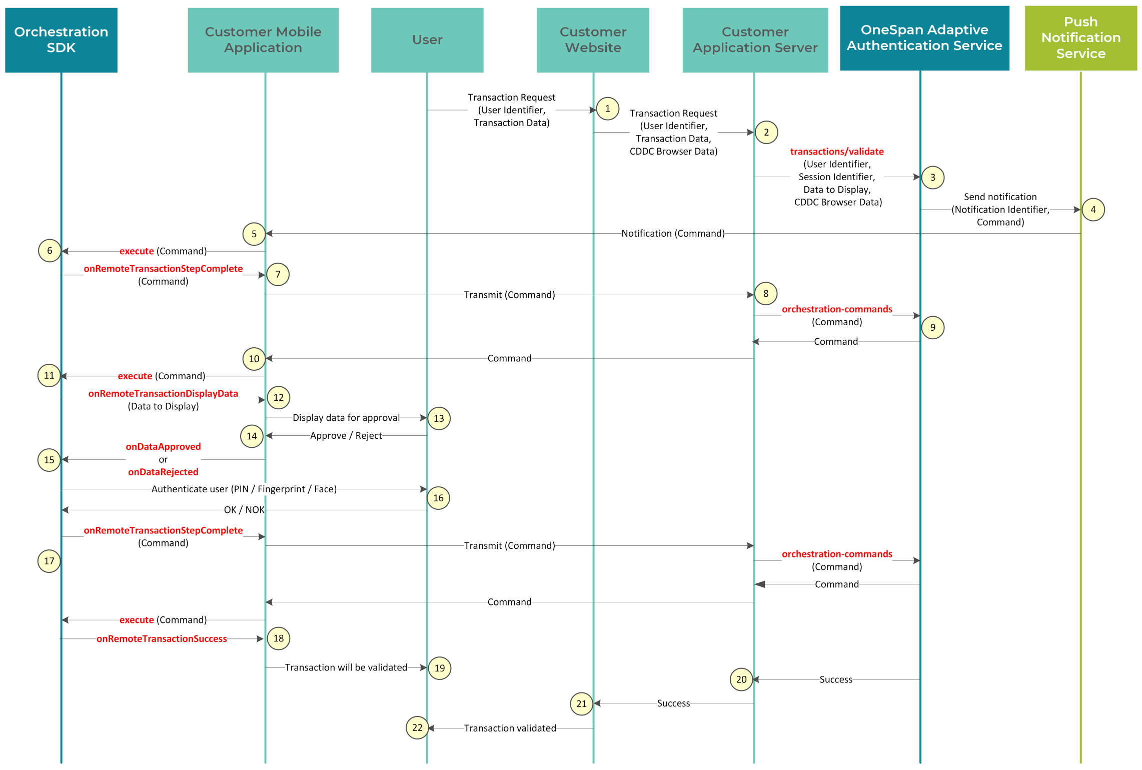
The transaction request is embedded in an orchestration command and can be transmitted via a push notification message initiated by the OneSpan Trusted Identity platform or by another communication channel handled by the Customer Website (e.g. image scanning).

The authentication request contains the following parameters:

- A *session identifier* created by the Customer Application Server, which uniquely identifies the transaction session.
- A *request identifier* created by the Customer Application Server, which uniquely identifies the transaction request.
- An *authentication method*, which defines how the user must authenticate to be able to sign the transaction request (see **3.2.7 Authentication methods**).
- *Data to display* on the Customer Mobile Application to provide transaction request information to the user; the user can choose to approve or reject it.

  The data to display is a string defined by the Customer Application Server, which must be interpreted by the Customer Mobile Application.

Figure 5 illustrates the remote transaction workflow with the transmission of the transaction request via a push notification message.



Orchestration SDK
Customer Mobile Application
User
Customer Website
Customer Application Server
OneSpan Adaptive Authentication Service
Push Notification Service
Transaction Request (User Identifier, Transaction Data)
1
Transaction Request (User Identifier, Transaction Data, CDDC Browser Data)
2
transactions/validate (User Identifier, Session Identifier, Data to Display, CDDC Browser Data)
3
Send notification (Notification Identifier, Command)
4
Notification (Command)
5
execute (Command)
6
onRemoteTransactionStepComplete (Command)
7
Transmit (Command)
8
orchestration-commands (Command)
9
Command
Command
10
execute (Command)
11
onRemoteTransactionDisplayData (Data to Display)
12
Display data for approval
13
Approve / Reject
14
onDataApproved or onDataRejected
15
Authenticate user (PIN / Fingerprint / Face)
16
OK / NOK
onRemoteTransactionStepComplete (Command)
17
Transmit (Command)
orchestration-commands (Command)
Command
Command
execute (Command)
onRemoteTransactionSuccess
18
Transaction will be validated
19
Success
20
Success
21
Transaction validated
22

Figure 5: Remote transaction workflow

▸ Remote transaction workflow

1. The user initializes a transaction request via the Customer Website (e.g. for money transfer purposes), providing their user identifier and the transaction data.

2. The Customer Website transmits the user identifier and the Client Device Data Collector (CDDC) browser data to the Customer Application Server.

3. The Customer Application Server calls the **https://{tenant}.{environment}.tid.onespan.cloud/v1/users/{userID@domain}/transactions/validate** endpoint from the OneSpan Trusted Identity platform API by providing their user identifier, a session identifier (dynamically generated and uniquely identifying the authentication request), the received CDDC browser data, and the data to display on the Customer Mobile Application.

4. The OneSpan Trusted Identity platform evaluates the risk related to the Web browser used for the transaction request (based on multiple parameters

provided in the previous step and on existing parameters related to the user, e.g. amount too high) and, in case of risk detection, initiates a step-up transaction request on the Customer Mobile Application with a given authentication method (see **3.2.7 Authentication methods**).

Depending on the configuration defined in the OneSpan Trusted Identity platform, multiple scenarios are possible:

- The transaction request can be transmitted via a push notification message initiated by the OneSpan Trusted Identity platform and sent by the Push Notification Service to the Customer Mobile Application. In this case, a push notification message is sent to all mobile devices of the user where the Customer Mobile Application is installed and activated.

- The transaction request can be transmitted by the Customer Application Server via a different channel (e.g. display a Cronto image containing the orchestration command related to the transaction request and scan it with the Customer Mobile Application).

- The transaction request can be blocking or non-blocking.

The following steps describe a blocking scenario with the transaction request transmitted via push notification.

5. The Push Notification Service sends a push notification message containing the orchestration command related to the transaction request to the Customer Mobile Application.

6. The Customer Mobile Application obtains the orchestration command contained in the push notification message and calls the `execute` method of the Orchestration SDK to perform the remote authentication.

7. The Orchestration SDK builds an orchestration command and transmits it to the Customer Mobile Application using the `onRemoteTransactionStepComplete` method.

8. The Customer Mobile Application transmits the orchestration command to the Customer Application Server.

9. The Customer Application Server calls the `orchestration-commands` Web service of the OneSpan Trusted Identity platform by providing the orchestration command. A new orchestration command is returned as a result.

10. The Customer Application Server transmits the orchestration command to the Customer Mobile Application as a response to the previous request.
11. The Customer Mobile Application calls the `execute` method of the Orchestration SDK to continue the remote transaction process (only if the transaction request is still pending).
12. The Orchestration SDK calls the `onRemoteTransactionDisplayData` method to transmit the data to display to the Customer Mobile Application.
13. The Customer Mobile Application displays a screen to the user containing the data to display and two buttons to approve or reject the transaction request.
14. The user must approve or reject the transaction request, according to the displayed data.
15. Based on the user's decision in the previous step, the Customer Mobile Application calls the `onDataApproved` or `onDataRejected` method of the Orchestration SDK.
16. In both cases, the Orchestration SDK prompts the user to authenticate by using an authentication method defined by the OneSpan Trusted Identity platform, based on the evaluated risk (see *step 4*).
17. The Orchestration SDK signs the transaction request, builds an orchestration command, and transmits it to the Customer Mobile Application using the `onRemoteTransactionStepComplete` method.
18. Repeat *steps 8 to 11*.
19. In case of validation by the OneSpan Trusted Identity platform, the Orchestration SDK calls the `onRemoteTransactionSuccess` method to notify the Customer Mobile Application.
20. The Customer Mobile Application notifies the user that the transaction request has been validated.
21. The OneSpan Trusted Identity platform provides a response to the call to the `transactions/validate` Web service from *step 3*, indicating the success of the transaction request.
22. The Customer Application Server transmits the success status to the Customer Website. The user is informed that the transaction has been validated on the Customer Website.

For more information about integrating this feature, see **5.5 Remote transaction**.

3.2.5 Local authentication

With the local authentication feature of the OneSpan Orchestration SDK, the user can authenticate to the Customer Website using a one-time password (OTP) generated via the Customer Mobile Application. The OTP can be transmitted manually by the user, or remotely by the Customer Mobile Application. An authentication method must be defined to authenticate the user before the OTP is generated. See **3.2.7 Authentication methods** for more information.

Figure 6 illustrates the local authentication workflow with a manual transmission of the OTP.

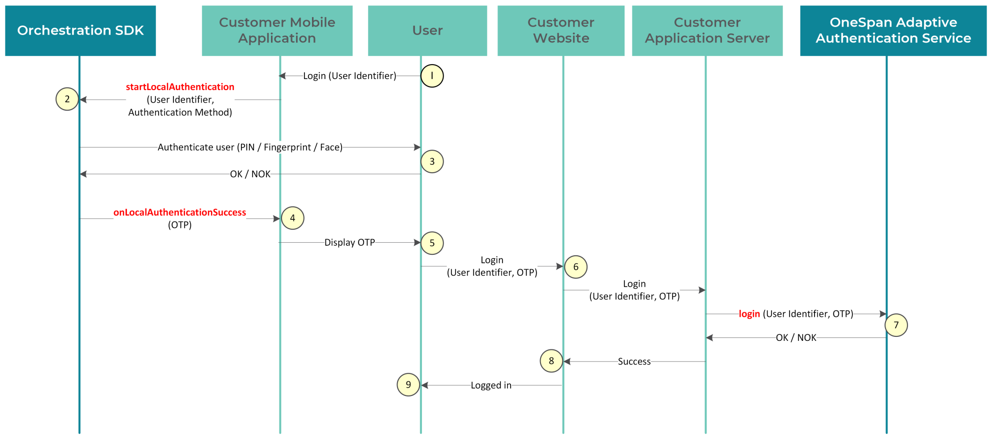

Orchestration SDK
Customer Mobile Application
User
Customer Website
Customer Application Server
OneSpan Adaptive Authentication Service
1
Login (User Identifier)
2
startLocalAuthentication (User Identifier, Authentication Method)
Authenticate user (PIN / Fingerprint / Face)
3
OK / NOK
onLocalAuthenticationSuccess (OTP)
4
Display OTP
5
Login (User Identifier, OTP)
6
Login (User Identifier, OTP)
login (User Identifier, OTP)
7
OK / NOK
Success
8
Logged in
9

Figure 6: Local authentication workflow

▸ Local authentication workflow

> **NOTE:** New APIs are created for Swift users of the iOS SDK. For more information, refer to the Xcode API documentation on `LocalAuthenticationDelegate` for this work-flow.

1. The user initializes an authentication request via the Customer Mobile Applic-ation (e.g. for login purposes), providing their user identifier.

2. The Customer Mobile Application calls the `startLocalAuthentication` method of the Orchestration SDK to perform the local authentication with a given

authentication method (see **3.2.7 Authentication methods** for more information).

3. The Orchestration SDK prompts the user to authenticate by using an authentication method defined by the Customer Mobile Application.
4. In case of successful user authentication, the Orchestration SDK generates a one-time password (OTP), and transmits it to the Customer Mobile Application using the `onLocalAuthenticationSuccess` method.
5. The Customer Mobile Application displays the OTP to the user.
6. The user initializes an authentication request via the Customer Website (e.g. for login purposes), by providing their user identifier and the generated OTP. This request is transmitted to the Customer Application Server.
7. The Customer Application Server calls the `login` method of the OneSpan Trusted Identity platform to verify the OTP.
8. The Customer Application Server provides a response to the Customer Website by indicating the success of the authentication request.
9. The user is logged in to the Customer Website.

For more information about integrating this feature, see **5.6 Local authentication**.

3.2.6 Local transaction

With the local transaction feature of the OneSpan Orchestration SDK, the user can do a transaction to the Customer Website using a signature generated via the Customer Mobile Application. The signature can be transmitted manually by the user or remotely by the Customer Mobile Application. An authentication method must be defined to authenticate the user before the signature is generated. See **3.2.7 Authentication methods** for more information.

Figure 7 illustrates the local transaction workflow with a manual transmission of the signature.

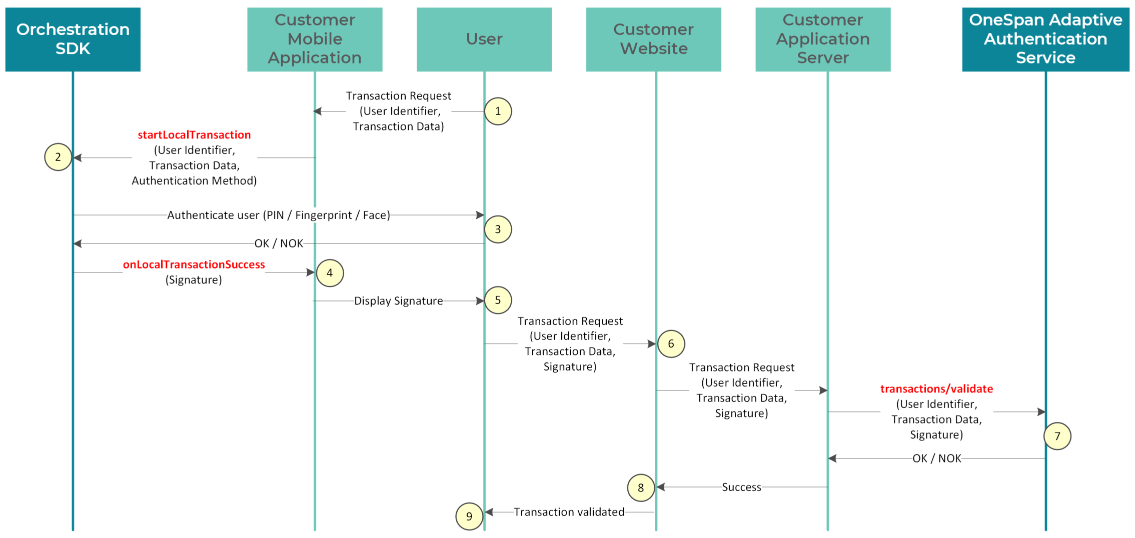

Orchestration SDK
Customer Mobile Application
User
Customer Website
Customer Application Server
OneSpan Adaptive Authentication Service
1 Transaction Request (User Identifier, Transaction Data)
2 startLocalTransaction (User Identifier, Transaction Data, Authentication Method)
3 Authenticate user (PIN / Fingerprint / Face)
OK / NOK
4 onLocalTransactionSuccess (Signature)
5 Display Signature
6 Transaction Request (User Identifier, Transaction Data, Signature)
Transaction Request (User Identifier, Transaction Data, Signature)
7 transactions/validate (User Identifier, Transaction Data, Signature)
OK / NOK
8 Success
9 Transaction validated

Figure 7: Local transaction workflow

▸ Local transaction workflow

NOTE: New APIs are created for Swift users of the iOS SDK. For more information, refer to the Xcode API documentation on `LocalTransactionDelegate` for this workflow.

1. The user initializes a transaction request via the Customer Mobile Application (e.g. for login purposes), providing their user identifier and the transaction data.
2. The Customer Mobile Application calls the `startLocalTransaction` method of the Orchestration SDK to perform the local authentication with a given authentication method (see **3.2.7 Authentication methods** for more information).
3. The Orchestration SDK prompts the user to authenticate, using an authentication method defined by the Customer Mobile Application.
4. In case of successful user authentication, the Orchestration SDK generates a signature and transmits it to the Customer Mobile Application using the `onLocalTransactionSuccess` method.
5. The Customer Mobile Application displays the signature to the user.
6. The user initializes a transaction request via the Customer Website (e.g. for money transfer purposes) by providing their user identifier, the transaction data,

and the generated signature. This request is transmitted to the Customer Application Server.

7. The Customer Application Server calls the `transactions/validate` method of the OneSpan Trusted Identity platform to verify the signature of the transaction request.
8. The Customer Application Server provides a response to the Customer Website, indicating the success of the transaction request.
9. The user is informed that the transaction has been validated on the Customer Website.

For more information about integrating this feature, see **5.7 Local transaction**.

3.2.7 Authentication methods

User authentication requested by the Orchestration SDK can take place using one of the following methods:

- No password
- Password
- Biometric recognition

In case of remote authentication or remote transaction the authentication method is dynamically defined by the OneSpan Trusted Identity platform after risk evaluation. In case of local authentication or transaction, the Customer Mobile Application must define the authentication method.

If the requested authentication method is not supported by the mobile device (e.g. biometric recognition requested but no biometric sensor on the device), the Orchestration SDK will fall back to authentication with password.

No password

If the authentication method is *No password*, no user action is required for the completion of the process.

Password

If the authentication method is *password*, the Orchestration SDK displays a virtual keypad or call the password user authentication flow, and the user needs to enter their password.

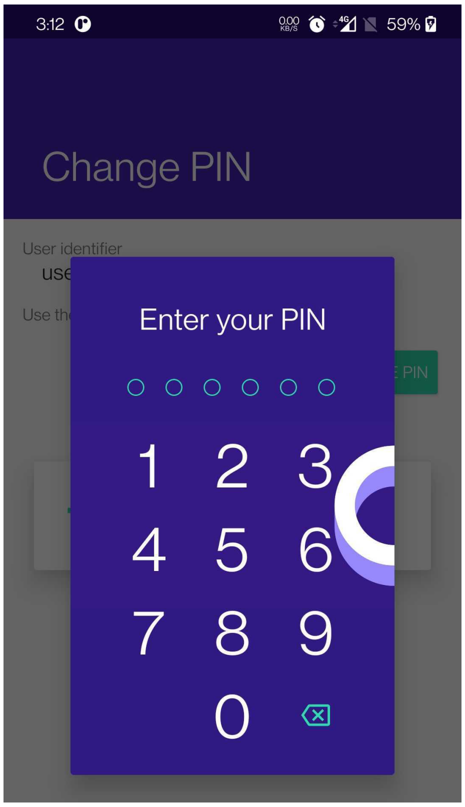


Figure 8: Authentication using a virtual keypad

For more information about this authentication method, refer to the *Digipass SDK Integration Guide*.

NOTE: The texts displayed in the dialog and other graphical elements (i.e. text, colors, icons) are customizable. For more information, see **4 Integrate the Orchestration SDK**.

Biometric

If the authentication method is *biometric recognition*, the Orchestration SDK displays a dialog prompting the biometric scanner of the device. The **Fallback** button provided by the Biometric Sensor SDK is not available in the biometric dialog.

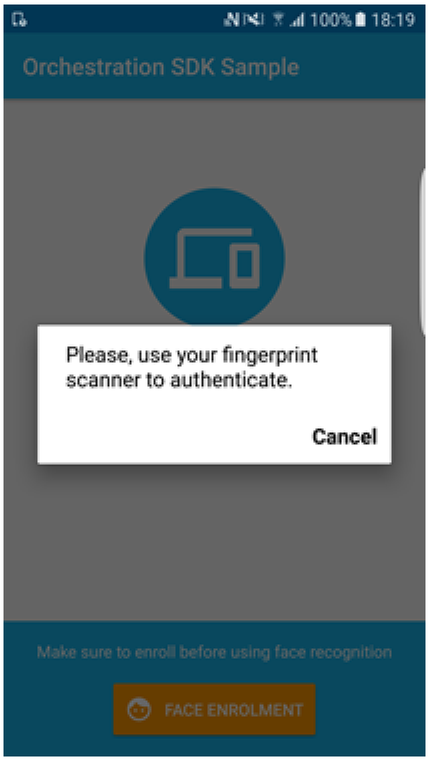

100% 18:19
Orchestration SDK Sample
Please, use your fingerprint scanner to authenticate.
Cancel
Make sure to enroll before using face recognition
FACE ENROLMENT

Figure 9: Authentication using biometric recognition

For more information about this authentication method, refer to the *`OneSpan Biometric Sensor SDK Integration Guide`*.

> **NOTE:** The texts displayed in the dialog are customizable. All other graphical elements (i.e. text, colors, icons) cannot be customized. For more information, see **4 Integrate the Orchestration SDK**.

External user authentication

Introduction

It is possible to override the user authentication. Instead of displaying the integrated Orchestration user authentication, the Orchestration SDK will call a specific callback to inform the Customer Mobile Application that such an authentication is required. In this call, the type of authentication will be indicated. For now only password is available. Below you will find some best practices for this kind of authentication.

NOTE: If an activation was done using an external password, it will have to be provided to change the password.

NOTE: The input for the PASSWORD authentication is not stored by the Orchestration SDK. This should never be store in the Customer Mobile Application.

Usage

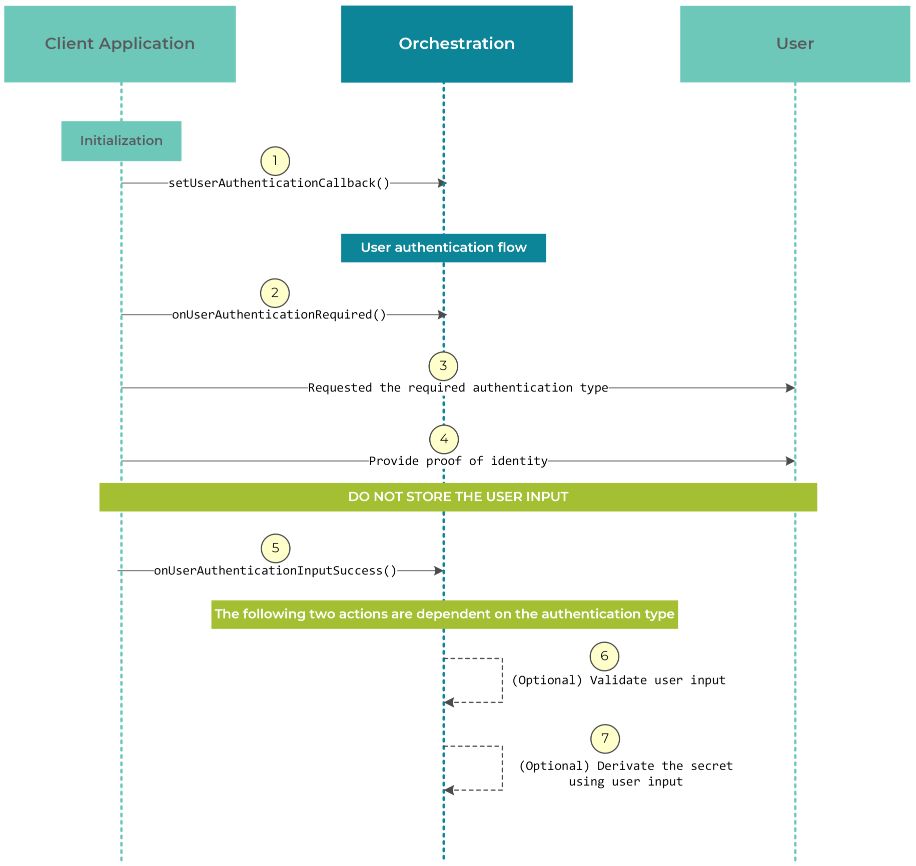

Client Application
Orchestration
User
Initialization
1
setUserAuthenticationCallback()
User authentication flow
2
onUserAuthenticationRequired()
3
Requested the required authentication type
4
Provide proof of identity
DO NOT STORE THE USER INPUT
5
onUserAuthenticationInputSuccess()
The following two actions are dependent on the authentication type
6
(Optional) Validate user input
7
(Optional) Derivate the secret using user input

Figure 10: External user authentication workflow

▸ User authentication flow

NOTE: New APIs are created for Swift users of the iOS SDK. For more information, refer to the Xcode API documentation on `UserAuthenticationDelegate` for this workflow.

1. The Customer Mobile Application registers a `UserAuthenticationCallback` and the different user authentication methods that will be overridden.
2. The Orchestration SDK informs the Customer Mobile Application, that a user authentication is required. The following information will be provided:
   - Type of user authentication.
   - A Boolean that indicates if this is for enrollment purposes.
   - A callback to send back the user input.
3. The Customer Mobile Application presents the requested authentication to the user.
4. The user identifies themselves via the Customer Mobile Application.
5. The Customer Mobile Application sends the user input back to the Orchestration SDK.
6. (OPTIONAL) The Orchestration SDK validates the provided input. For example the password should respect the weak password rules of Digipass SDK.
7. (OPTIONAL) The user input can be used to derive the secret stored in the Digipass SDK. , This way, it can only be used when the user provides the same input.

Password input best practices

Mobile apps often handle sensitive information in the form of passwords. The following recommendations outline the security precautions that should be taken when entering passwords into mobile apps.

Build an in-house keypad (do not use one of the inbuilt keyboards of the device)

Using a custom keypad that is part of the app, prevents attacks using malicious keyboards or generic keyboards sniffers. Third-party keyboards may hide malicious functionalities and can be used to steal passwords. For this reason, they should not be used for password input.

When using an in-house keypad, consider randomizing the position of the keys in the keypad

This security measure will make event grabbing attacks less likely.

Do not allow copying of the password from its input field

If the password is entered in an input field, make sure it cannot be copied to the clipboard.

Use password text fields that hide the value of the password

If the password is entered in an input field, ensure that the password is masked (e.g., screen displays this format "****"). This security measure prevents attackers from reading the password over the shoulder or from capturing the password via screenshot.

Avoid the use of WebViews for password input

WebViews present additional security risks and should not be used for the input of sensitive information.

Clear passwords in memory as soon as they are no longer needed

Passwords should reside in memory as short as possible.

This means that the object used for storing the password in memory should be clearable (e.g., in Java passwords should be stored in Byte arrays instead of Strings). Once the password is no longer required, the object should be actively cleared (e.g., by zeroing the bytes in the array).

Validate the input before storing it in the password object in memory

In case one of the inbuilt device keyboards is used (and not an in-house keypad), the input from the keyboards should be sanitized before accepting it. Note that this is not the same as checking against a password policy. This is just to ensure that no invalid characters are entered in the input field.

Validate the password output before sending it to a downstream SDK or API call

Although it is the responsibility of the downstream SDK or API to sanitize its inputs, it is a good practice to also sanitize the output before using it in and SDK or API call. For example, the password could be validated against a list of acceptable characters and a maximum length.

3.2.8 Change password

The Orchestration SDK provides facilities for the user to change their password. The password was defined during the activation process.

The Orchestration SDK displays a virtual keypad or calls the password user authentication flow, where the user can enter the old password, and then define and confirm the new password. If an external user authentication by password has been configured the "User Authentication Flow" will be called.

A network request is required to check whether the entered old password is valid: an OTP is generated with the old password and validated on the Customer Application Server.

Figure 11 illustrates the change password workflow.

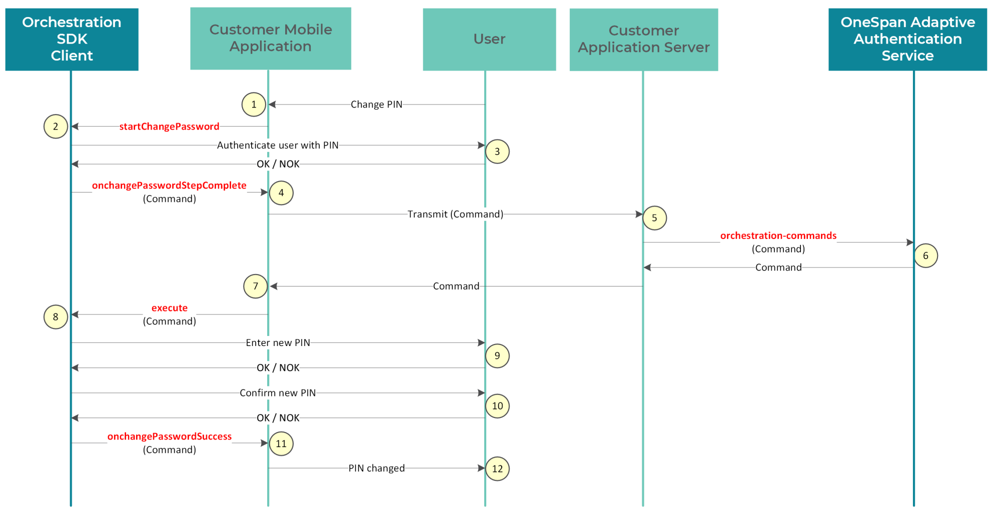

Orchestration SDK Client
Customer Mobile Application
User
Customer Application Server
OneSpan Adaptive Authentication Service
1
Change PIN
2
startChangePassword
Authenticate user with PIN
3
OK / NOK
onchangePasswordStepComplete (Command)
4
Transmit (Command)
5
orchestration-commands (Command)
6
Command
7
Command
8
execute (Command)
Enter new PIN
9
OK / NOK
Confirm new PIN
10
OK / NOK
onchangePasswordSuccess (Command)
11
PIN changed
12

Figure 11: Change password workflow

▸ Change password workflow

NOTE: New APIs are created for Swift users of the iOS SDK. For more information, refer to the Xcode API documentation on `ChangePasswordDelegate` for this workflow.

1. The user initiates the change password process using the Customer Mobile Application.
2. The Customer Mobile Application calls the `startChangePassword` method of the Orchestration SDK to change the user's password.
3. The Orchestration SDK displays the virtual keypad or calls the password user authentication flow, where the user can enter their old password.
4. The Orchestration SDK generates a one-time password (OTP) with the old password, builds an orchestration command, and transmits it to the Customer Mobile Application using the `onChangePasswordStepComplete` method.
5. The Customer Mobile Application transmits the orchestration command to the Customer Application Server.
6. The Customer Application Server calls the `orchestration-commands` Web service of the OneSpan Trusted Identity platform by providing the orchestration command. A new orchestration command is returned as a result.
7. The Customer Application Server transmits the orchestration command to the Customer Mobile Application as a response to the previous request.
8. The Customer Mobile Application calls the `execute` method of the Orchestration SDK to continue the change password process (only if the OTP validation succeeded).
9. In case of successful user authentication, the Orchestration SDK displays the virtual keypad or calls the password user authentication flow, where the user can enter their new password.
10. If the password is not weak, the Orchestration SDK displays the virtual keypad or calls the password user authentication flow, where the user can confirm their new password.
11. The Orchestration SDK calls the `onChangePasswordSuccess` method to notify the Customer Mobile Application of the changed password.
12. The Customer Mobile Application notifies the user that the password has been successfully changed.

For more information about integrating this feature, see **5.9 Change password**.

3.2.9 Device data collection

The orchestration commands sent by the Orchestration SDK contain device data collected periodically.

This data is used for risk evaluation in the OneSpan Trusted Identity platform and protected via the Secure Channel feature to ensure confidentiality, integrity, and non-repudiation. For more information about the collected data, refer to the `Client Device Data Collector SDKIntegration Guide`.

Collection of device data is started automatically when the Orchestration SDK is initialized, with the following default collection parameters:

- Refresh interval: 60 seconds
- Collection enabled for geolocation, Bluetooth, and Wi-Fi

Special permissions may be required (see **4.1 Integrating OneSpan Orchestration SDK with Android**).

Data that cannot be collected automatically must be populated during the integration. For more information about integrating this feature, see **5.10 Device data collection**.

3.2.10 Multi-user management

The Orchestration SDK provides facilities to activate multiple users in the Customer Mobile Application. To achieve this, each activation process must use its own user identifier.

> **NOTE:** Each user has their own password; the password is not shared by all users.
>
> Notification registration must be performed for each activated user.

The Orchestration SDK provides methods to check whether a user is activated, to obtain user information, and to delete a user.

For more information about integrating this feature, see **5.11 Multi-user management**.

Integrate the Orchestration SDK

4

This section provides information on exposed APIs and instructions to integrate the Orchestration SDK with the supported platforms.

4.1 Integrating OneSpan Orchestration SDK with Android

▸ To use OneSpan Orchestration SDK in your Android project

1. Configure your Maven repository using dependencies from the `Bin/maven` directory.

2. Add the following dependencies to the dependencies configuration in the `build.gradle` file :

```
dependencies {
...
  implementation 'androidx.biometric:biometric:1.1.0'
  implementation 'com.google.firebase:firebase-core:21.1.1'
  implementation 'com.google.firebase:firebase-messaging:23.1.2'
  implementation 'androidx.appcompat:appcompat:1.6.1'
  implementation 'androidx.legacy:legacy-support-v4:1.0.0'
  implementation 'androidx.lifecycle:lifecycle-runtime:2.6.0'
  implementation 'com.google.android.material:material:1.8.0'
  implementation 'com.google.android.gms:play-services-location:21.0.1'
  implementation 'com.esotericsoftware:kryo:5.3.0'
  implementation 'org.bouncycastle:bcprov-jdk15on:1.70
  implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.7.0"
...
}
```

3. Set the correct permissions for client device data collection in the `AndroidManifest.xml` file:

   - Geolocation/Bluetooth/Wi-Fi: *ACCESS_COARSE_LOCATION* or *ACCESS_FINE_LOCATION*
   - Wi-Fi: *ACCESS_WIFI_STATE*, *ACCESS_NETWORK_STATE*, and *CHANGE_WIFI_STATE*
   - Bluetooth: *BLUETOOTH* and *BLUETOOTH_ADMIN* for Android lower than 12

- Bluetooth: *BLUETOOTH_CONNECT* and *BLUETOOTH_SCAN* for Android 12 and later
- Sampling rate: `HIGH_SAMPLING_RATE_SENSORS` for Android 12 and later

The Orchestration SDK provides facilities for biometric recognition on Android 6.0 devices (and later) with a biometric scanner.

NOTE: If you use the Mobile Application Shielding obfuscation feature, you must add the following rule to the Mobile Application Shielding obfuscation configuration file:

```
untouchable com.esotericsoftware.kryo.*;
```

TIP: For more specific integration details, refer to the code sample that is provided with the Orchestration SDK product package.

NOTE: The Orchestration SDK requires the following permissions:

- *USE_BIOMETRIC* for biometric recognition on Android 6.0 devices (and later) with the available biometric authentication method.
- *VIBRATE* for password input.

These permissions are listed in the `AndroidManifest.xml` file contained in `OrchestrationSDK.aar`. This file is included in the maven repository, therefore the permissions do not need to be duplicated in the `AndroidManifest.xml` file of the Customer Mobile Application.

NOTE: As of Android 13, to receive notifications in the Orchestration SDK you have to manually handle the new runtime permission `POST_NOTIFICATIONS`. For more information, refer to **developer.android.com**.

NOTE: The Orchestration SDK library size has increased due to 64-bit support. To reduce the binary size, it is possible to generate one app per architecture. For more details refer to **Build Multiple APKs**. The orchestration sample `build.gradle` contains a commented-out section to generate split APKs.

CAUTION: At this time, it is not possible to use the Android *Allow Backup* feature. There is a known issue that excludes storage files from the backup which are critical for the Orchestration SDK to work properly. We strongly recommend disabling the *Allow Backup* option.

You are now ready to use the Orchestration SDK.

4.2 Integrating OneSpan Orchestration SDK with iOS

▸ Using OneSpan Orchestration SDK in your iOS project

1. Link `MSSOrchestration.xcframework` to your Xcode project.
2. Link `MSSDeviceBinding.xcframework` to your Xcode project.
3. Link `MSSNotificationClient.xcframework` to your Xcode project.
4. Link `MSSOrchestrationResources.bundle` to your Xcode project as standard resource file.
5. Add the following keys and their description to your `plist`:
   - `NSCameraUsageDescription`
   - `NSFaceIDUsageDescription`
   - `NSLocationWhenInUseUsageDescription`
   - `NSBluetoothPeripheralUsageDescription`
   - `NSBluetoothAlwaysUsageDescription`
   - `NSLocalNetworkUsageDescription` (if applicable)

CAUTION: As of release 5.6.2, it is mandatory to use the `AppPrivate` access group for activated users when creating an orchestration. The access group setup is similar to the one introduced in the Device Binding SDK. For more information, refer to the technical documentation included in the Orchestration SDK package. For a detailed tutorial on how to configure the project and use the proper access group, see the `MSSDeviceBinding.doccarchive` file from the `iOS/Documentation` folder of the Device Binding SDK package.

NOTE: The `NSLocalNetworkUsageDescription` description has to be set only if your application (or specific test target) actually accesses the server in a local network. When this applies, set `waitsForConnectivity` to **true**. With this `URLSessionConfiguration` used in the communication waits for user's interaction, displaying a permission pop-up.

TIP: For more specific integration details, refer to the code sample that is provided with the Orchestration SDK product package.

You are now ready to use the Orchestration SDK.

Exposed APIs in the Orchestration SDK 5

This section provides information on exposed APIs and instructions to integrate the Orchestration SDK with the supported platforms.

5.1 Core

NOTE: The Orchestration SDK contains a native interface for Swift. This interface provides native Swift methods and objects. It offers the same logical flow as the Objective-C interface but provides better signatures and more data on orchestrator instances. The Swift sample app delivered with the Orchestration SDK package uses the relevant Swift APIs, and the `README.md` file delivered with the sample app contains examples and code snippets.

5.1.1 Entry point

The `Orchestrator` class is the entry point of the Orchestration SDK. It manages the flows related to the features of the SDK by providing methods to start these flows (e.g. `startActivation`) and to interpret the orchestration commands received from the OneSpan Trusted Identity platform (via the `execute` method).

An `Orchestrator` object must be created by using the dedicated `Orchestrator.Builder` class.

The following parameters must be defined when building an `Orchestrator` object:

- Hardcoded salts, which will be used for weak diversification with certain security features (e.g. device binding, secure storage). These salts will be derived by the Orchestration SDK to complicate reverse-engineering attacks.
- Default domain, as configured in the OneSpan Trusted Identity platform. If the Customer Mobile Application must manage multiple domains, the default domain can be dynamically overwritten for each action (e.g. `startActivation`).
- Android only: A `Context` object is required for using the native features of the mobile device (e.g. data storage).
- Android only: An `ActivityProvider` object is required to properly display and handle the Biometric and PinPad dialog.
- A `CDDCParams` object, which will define the parameters for device data collection. These parameters are optional; default parameters are hardcoded in the Orchestration SDK. For more information, see **5.10 Device data collection**.

- An `OrchestrationErrorCallback` object, which will be used to throw errors from the Orchestration SDK to the Customer Mobile Application. For more information, see **5.1.2 Error and warning management**.

- An `OrchestrationWarningCallback` object, which will be used to throw warnings from the Orchestration SDK to the Customer Mobile Application. For more information, see **5.1.2 Error and warning management**.

> **NOTE:** New APIs are created for Swift users of the iOS SDK. For more information, refer to the Xcode API documentation on `CDDCParameters`, `OrchestrationErrorDelegate` and `OrchestrationWarningDelegate`.

5.1.2 Error and warning management

Error

Errors are thrown via the `onOrchestrationError` method of the `OrchestrationErrorCallback` object. The `OrchestrationError` object provides error codes and the cause of the error. The possible errors are listed in the `OrchestrationErrorCodes` class.

Warning

Warnings are thrown via the `onOrchestrationWarning` method of the `OrchestrationWarningCallback` object. The `OrchestrationWarning` object provides warning codes and the cause of the warning, as well as a list of all possible warning codes. The possible errors are listed in the `OrchestrationWarningCodes` class.

> **NOTE:** New APIs are created for Swift users of iOS SDK. Please consult Xcode API documentation provided on `OrchestrationErrorDelegate` and `OrchestrationWarningDelegate`. All possible errors are listed under `OrchestrationError` enum and all possible warnings are listed under `OrchestrationWarning` enum.

5.1.3 Callback mechanism

The Orchestration SDK uses asynchronous mechanisms to execute the flows related to the features of the SDK.

Callback methods are used to notify the Customer Mobile Application when an action related to a specific flow is finished (e.g. the activation has succeeded) or requires action of the Customer Mobile Application (e.g. perform a network request with the given orchestration command).

NOTE: The callback methods are called from the UI thread on Android, and from the main thread on iOS.

The Customer Mobile Application must provide an implementation of the dedicated callback methods to receive these notifications.

Callback registration is required for the following features:

- Activation
- Remote authentication
- Remote transaction
- Local authentication
- Local transaction
- Change password
- Notification registration

5.1.4 Application life cycle

On Android, to properly display UI components such as the virtual keypad, the `Orchestrator` object is life-cycle-aware.

For the `Orchestrator` object to work properly, an instance of the `ActivityProvidor` is required. This returns weak references to the current `Activity` object.

NOTE: This requirement does not apply to applications on iOS.

5.2 Activation

NOTE: New APIs are created for Swift users of the iOS SDK. For more details, refer to the Xcode API documentation on `OnlineActivationParameters` and `OnlineActivationDelegate`.

The activation process must be initiated by calling the `startActivation` method of the `Orchestrator` object.

An `OnlineActivationParams` object must be provided as an input parameter of the `startActivation` method, and this `OnlineActivationParams` object must be initiated with the following parameters:

- *User identifier*. A string which uniquely identifies the user on the OneSpan Trusted Identity platform.
- *Activation password*. Secret data shared between the Customer Application Server and the user.
- (OPTIONAL) *Cryptographic Application Index*. The index of the cryptographic application that must be used to sign the instance activation message. The default value is **1**.
- An object implementing the `OnlineActivationCallback` interface.

The `OnlineActivationCallback` interface is used by the Orchestration SDK to interact with the Customer Mobile Application during the activation process. It exposes the following methods:

- `onActivationStepComplete`. Called upon activation step success, the provided command must be sent to the server.
- `onActivationSuccess`. Called upon activation process success.
- `onActivationInputError`. Called upon activation process error due to incorrect user input. The possible errors are listed in the `OrchestrationErrorCodes` class.
- `onActivationAborted`. Called when the activation process is canceled.

For more information about this feature, see**3.2.1 Activation**

5.3 Notification registration

NOTE: New APIs are created for Swift users of the iOS SDK. For more information, refer to the Xcode API documentation on `NotificationRegistrationParameters` and `NotificationRegistrationDelegate`.

The Customer Mobile Application can initialize a notification registration process by calling the `startNotificationRegistration` method of the `Orchestrator` object.

A `NotificationRegistrationParams` object must be provided as an input parameter of the `startNotificationRegistration` method; this `NotificationRegistrationParams` object must be initiated with the following parameters:

- *User identifier*. A string which uniquely identifies the user on the OneSpan Trusted Identity platform.
- *Notification identifier*. Uniquely identifies an app on a given device in the context of push-notification-based authentication. Defined by the notification service (i.e. Apple or Google).
- An object implementing the `NotificationRegistrationCallback` interface.

The `NotificationRegistrationCallback` interface is used by the Orchestration SDK to interact with the Customer Mobile Application during the notification registration process. It exposes the following methods:

- `onNotificationRegistrationStepComplete`: Called when a step of the notification registration process is complete. The provided orchestration command must be sent to the server.
- `onNotificationRegistrationSuccess`: Called upon successful notification registration.

For more information about this feature, see **3.2.2 Notification registration**.

5.4 Remote authentication

NOTE: New APIs are created for Swift users of the iOS SDK. For more information, refer to the Xcode API documentation on `RemoteAuthenticationDelegate`.

The Customer Mobile Application must call the `setRemoteAuthenticationCallback` method and provide an implementation of the `RemoteAuthenticationCallback` interface to process a remote authentication flow.

The `RemoteAuthenticationCallback` interface is used by the Orchestration SDK to interact with the Customer Mobile Application during the remote authentication process. It exposes the following methods:

- `onRemoteAuthenticationDisplayData`. Called when the Orchestration SDK needs the `RemoteAuthenticationCallback` interface implementation to display data for approval by the user.
  - The Customer Mobile Application must call the `onDataApproved` method of the `DisplayDataCaller` object if the user approves the authentication request.
  - The Customer Mobile Application must call the `onDataRejected` method of the `DisplayDataCaller` object if the user rejects the authentication request.
- `onRemoteAuthenticationStepComplete`. Called when a step of the remote authentication process is complete. The provided orchestration command must be sent to the server.
- `onRemoteAuthenticationSuccess`. Called upon remote authentication success. It can also be called to notify the Customer Mobile Application that the rejected request has been taken into account by the Customer Application Server.
- `onRemoteAuthenticationSessionOutdated`. Called when the remote authentication session is outdated (expired, already approved, or already rejected).
- `onRemoteAuthenticationAborted`. Called when remote authentication is canceled.
- `onRemoteAuthenticationPasswordError`. Called upon remote authentication process error due to incorrect user input. The possible errors are listed in the `OrchestrationErrorCodes` class.

For more information about this feature, see **3.2.3 Remote authentication**.

5.5 Remote transaction

NOTE: New APIs are created for Swift users of the iOS SDK. For more information, refer to the Xcode API documentation on `RemoteTransactionDelegate`.

The Customer Mobile Application must call the `setRemoteTransactionCallback` method and provide an implementation of the `RemoteTransactionCallback` interface to process a remote transaction flow.

The `RemoteTransactionCallback` interface is used by the Orchestration SDK to interact with the Customer Mobile Application during the remote transaction process. It exposes the following methods:

- `onRemoteTransactionDisplayData`. Called when the Orchestration SDK needs the `RemoteTransactionCallback` interface implementation to display data for approval by the user.
    - The Customer Mobile Application must call the `onDataApproved` method of the `DisplayDataCaller` object if the user approves the transaction request.
    - The Customer Mobile Application must call the `onDataRejected` method of the `DisplayDataCaller` object if the user rejects the transaction request.
- `onRemoteTransactionStepComplete`. Called when a step of the remote transaction process is complete. The provided orchestration command must be sent to the server.
- `onRemoteTransactionSuccess`. Called upon remote transaction success. It can also be called to notify the Customer Mobile Application that the rejected request has been taken into account by the Customer Application Server.
- `onRemoteTransactionSessionOutdated`. Called when the remote transaction session is outdated (expired, already approved, or already rejected).
- `onRemoteTransactionAborted`. Called when remote transaction is canceled.
- `onRemoteTransactionPasswordError`. Called upon remote transaction process error due to incorrect user input. The possible errors are listed in the `OrchestrationErrorCodes` class.

For more information about this feature, see **3.2.4 Remote transaction**.

5.6 Local authentication

NOTE: New APIs are created for Swift users of the iOS SDK. For more information, refer to the Xcode API documentation on `LocalAuthenticationParameters` and `LocalAuthenticationDelegate`.

The Customer Mobile Application can initialize a local authentication process by calling the `startLocalAuthentication` method of the `Orchestrator` object.

A `LocalAuthenticationParams` object must be provided as an input parameter of the `startLocalAuthentication` method; this `LocalAuthenticationParams` object must be initiated with the following parameters:

- *User identifier*. A string which uniquely identifies the user on the OneSpan Trusted Identity platform.
- *Cryptographic Application Index*. The index of the cryptographic application which must be used to generate the OTP.
- (OPTIONAL) *Challenge*. A string which can be used to diversify the OTP generation. If the crypto-app index is related to a Challenge/Response application, a challenge parameter must be provided.
- *Protection Type*. The authentication method which must be used to authenticate the user before the OTP is generated (see **3.2.7 Authentication methods** for more information).
- An object implementing the `LocalAuthenticationCallback` interface.

The `LocalAuthenticationCallback` interface is used by the Orchestration SDK to interact with the Customer Mobile Application during the local authentication process. It exposes the following methods:

- `onLocalAuthenticationSuccess`: Called upon local authentication success. It returns the generated OTP and, depending on the Digipass configuration, a host code (used to authenticate the authentication server on which the OTP has been submitted).
- `onLocalAuthenticationAborted`: Called when local authentication is canceled.

- `onLocalAuthenticationPasswordError`: Called upon a local authentication process error due to incorrect user input. The possible errors are listed in the `OrchestrationErrorCodes` class.

For more information about this feature, see **3.2.5 Local authentication**.

5.7 Local transaction

NOTE: New APIs are created for Swift users of the iOS SDK. For more information, refer to the Xcode API documentation on `LocalTransactionParameters` and `LocalTransactionDelegate`.

The Customer Mobile Application can initialize a local transaction process by calling the `startLocalTransaction` method of the `Orchestrator` object.

A `LocalTransactionParams` object must be provided as an input parameter of the `startLocalTransaction` method; this `LocalTransactionParams` object must be initiated with the following parameters:

- *User identifier*. A string which uniquely identifies the user on the OneSpan Trusted Identity platform.
- *Cryptographic Application Index*. The index of the cryptographic application which must be used to generate the signature.
- *Data Fields*. The transaction data to sign. A list of max. 8 data fields; the length of the data fields is limited to 16 characters.
- *Protection Type*. The authentication method which must be used to authenticate the user before signing the transaction (see **3.2.7 Authentication methods** for more information).

The `LocalTransactionCallback` interface is used by the Orchestration SDK to interact with the Customer Mobile Application during the local transaction process. It exposes the following methods:

- `onLocalTransactionSuccess`. Called upon local transaction success. It returns the generated signature and, depending on the Digipass configuration, a host code (used to authenticate the authentication server on which the signature has been submitted).
- `onLocalTransactionAborted`. Called when local transaction is cancelled.

- `onLocalTransactionPasswordError`. Called upon a local transaction process error due to incorrect user input. The possible errors are listed in the `OrchestrationErrorCodes` class.

For more information about this feature, see **3.2.6 Local transaction**.

5.8 Authentication method

5.8.1 Customization of integrated authentication

The texts related to the authentication methods can be customized using the keys listed in **Table 1** in the configuration files of the Customer Mobile Application (i.e. `string.xml` for Android and `Localizable.strings` for iOS).

Table 1: Virtual keypad - Text customization keys

Key	Default value	Description
`orch_pinpad_text_registration`	Choose a password	Registration text
`orch_pinpad_text_registration_confirm`	Confirm your password	Confirmation text during registration
`orch_pinpad_text_authentication`	Enter your password	Text for authentication
`orch_pinpad_text_update`	Choose a new password	Text for updating the password
`orch_pinpad_text_update_confirmation`	Confirm new password	Text for confirming the updated password
`orch_pinpad_error_weak`	The password is too simple. Choose a more complex password.	Weak password error
`orch_pinpad_error_confirmation`	The password confirmation has failed. Make sure you entered the same password twice.	Password confirmation error

Table 2: Biometric recognition

Key	Default value
`orch_biometric_title`	Biometric Authentication
`orch_biometric_description`	Please, use your Biometric scanner to authenticate
`orch_biometric_failed`	Authentication failed
`orch_biometric_btn_cancel`	Cancel

The color can be customized using the keys listed in **Table 3** in the configuration files of the Customer Mobile Application (i.e. `color.xml` for Android and `Localizable.strings` for iOS).

Table 3: Virtual keypad - Color customization keys

Key	Default value	Description
`orch_pinpad_background_color`	#ffffffff	Background color
`orch_pinpad_arrow_color`	#ffffc107	**Delete** arrow color
`orch_pinpad_input_color`	#ffffc107	Secure input color
`orch_pinpad_text_color`	#ff000000	Title color
`orch_pinpad_text_error_color`	#ffff0000	Text error color
`orch_pinpad_digit_color`	#ff000000	Color of the Virtual keypad digits

The font size can be customized using the keys listed in **Table 4** in the configuration files of the Customer Mobile Application (i.e. `dimens.xml` for Android and `Localizable.strings` for iOS).

Table 4: Virtual keypad - Font size customization keys

Key	Android Default value	iOS Default value	Description
`orch_pinpad_input_empty_size`	N/A	15	Secure input inactive font size, only on iOS
`orch_pinpad_input_full_size`	N/A	20	Secure input active font size, only on iOS
`orch_pinpad_title_text_size`	20sp	25	Title font size
`orch_pinpad_text_error_size`	18sp	16	Text error font size
`orch_pinpad_digit_text_size`	40sp	50	Font size of the Virtual keypad digits

The other graphical elements are defined differently on iOS and Android.
For iOS, you can define these by customizing the keys listed in **Table 5** in the configuration files of the Customer Mobile Application (i.e. `Localizable.strings`).

Table 5: Virtual keypad - Graphical elements customization keys for iOS

Key	Default value	Description
`orch_pinpad_background_image`		Background image (can be empty)
`orch_pinpad_background_mode`	fit	Background image display mode. Possible values: • **center** • **fit** • **stretch**
`orch_pinpad_arrow_image`	backspace	**Delete** arrow image
`orch_pinpad_input_font_name`		Secure input font (**System** if empty)
`orch_pinpad_input_empty_character`	□	Secure input empty character
`orch_pinpad_input_full_character`	■	Secure input full character
`orch_pinpad_text_font_name`	HelveticaNeue-Light	Title font name
`orch_pinpad_text_error_font_name`	HelveticaNeue	Text error font name
`orch_pinpad_digit_font_name`	HelveticaNeue-UltraLight	Font name of the Virtual keypad digits

For Android, the font can be customized by overriding the styles listed in **Table 6** in the configuration files of the Customer Mobile Application (i.e. `styles.xml`).

Table 6: Virtual keypad - Font customization for Android

Style	Default font family name	Description
`PinpadTitleFont`	sans-serif-thin	Title font style
`PinpadErrorFont`	sans-serif-thin	Text error font style
`PinpadDigitFont`	sans-serif-thin	Font style of the Virtual keypad digits

For Android, the secure input and the backspace arrow can be customized by overriding the drawable elements listed in **Table 7** in the configuration files of the Customer Mobile Application (i.e. in the drawable folder).

Table 7: Virtual keypad - Drawable element customization for Android

Drawable	Description
`orch_pinpad_backspace.xml`	**Delete** arrow drawable
`orch_pinpad_clue_activated.xml`	Secure input full drawable
`orch_pinpad_clue_deactivated.xml`	Secure input empty drawable
`orch_pinpad_clue_highlighted.xml`	Secure input highlighted drawable
`orch_pinpad_background_image.xml`	Virtual keypad background drawable, can be overridden by an image.

For Android, the Virtual keypad background can be customized in multiple ways:

- by changing its background color, as indicated in the Virtual keypad color table
- by overriding its background image, as indicated in the Virtual keypad drawable table
- by overriding its layout in the configuration files of the Customer Mobile Application (i.e. in the layout folder)

Table 8: Virtual keypad - Background customization for Android

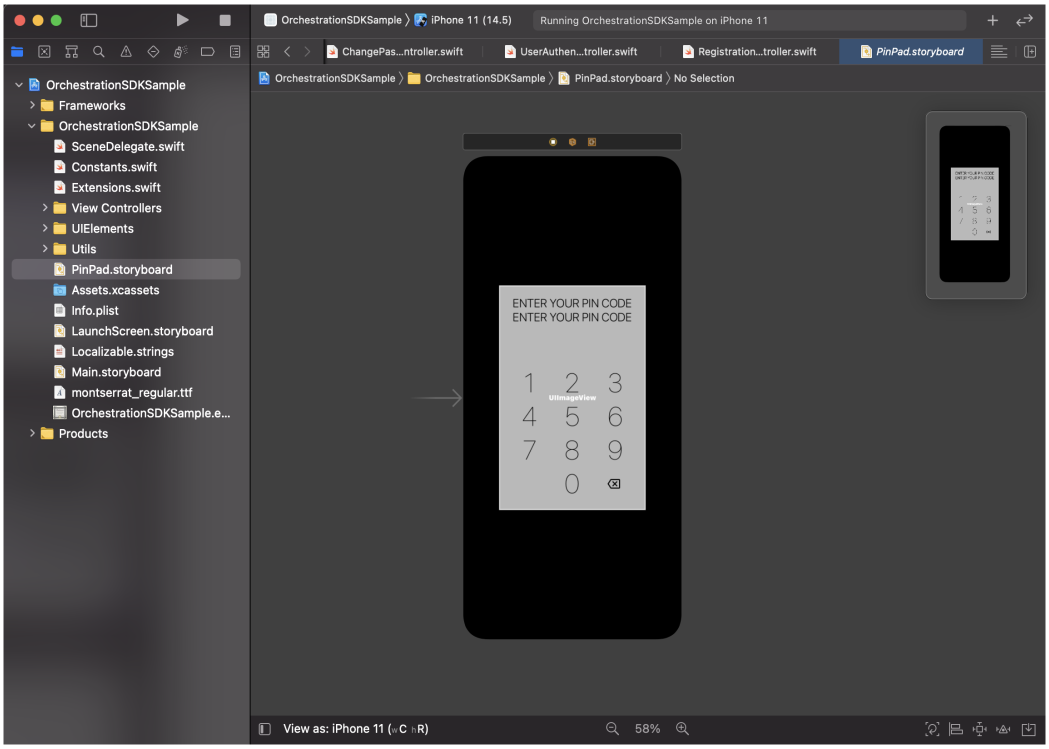
Layout	Description
`orch_pinpad_background_layout.xml`	Layout of the Virtual keypad background; the background image can be changed by using the `android:src` key.

Dark mode support for iOS

In the iOS sample, a file named `PinPad.storyboard` is available.

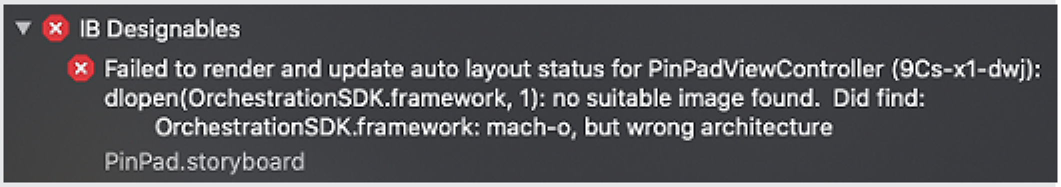
This file makes it possible to customize fonts, icon and colors. The Dark mode is managed by declaring named colors in the assets catalog (`Assets.xcassets`) or by using system colors.

The storyboard will take over the string customization as soon as it is added to the integrating project.



NOTE: This storyboard contains a lot of links to outlets defined inside the OrchestrationSDK. These links can't be restored if they are removed.

NOTE: The error you see in the storyboard is perfectly normal and doesn't make the compilation fail.



```
IB Designables
Failed to render and update auto layout status for PinPadViewController (9Cs-x1-dwj):
dlopen(OrchestrationSDK.framework, 1): no suitable image found.  Did find:
        OrchestrationSDK.framework: mach-o, but wrong architecture
PinPad.storyboard
```

5.8.2 External user authentication

We also provide an option to override the user authentication. For example instead of displaying the integrated Virtual keypad, you can display your own password authentication.

The Customer Mobile Application must call the `setUserAuthenticationCallback` method and provide:

- an implementation of the `UserAuthenticationCallback` interface that will be called when an overridden user authentication is required.
- a list of `UserAuthenticationType` containing all the user authentication you want to override. For now only the `UserAuthenticationType.PASSWORD` can be overridden.

The `UserAuthenticationCallback` interface is used by the Orchestration SDK Client to interact with the Customer Mobile Application during the process of overridding the user authentication. It exposes the following methods:

- `onUserAuthenticationRequired`. Called when the Orchestration SDK Client needs the `UserAuthenticationCallback` object to authenticate the end user.
    - The Customer Mobile Application must call the `onUserAuthenticationInputSuccess` method of the `UserAuthenticationInputCallback` object if the user is authenticated. An input from the user is expected, it may be used to derive the secret store in the device.
    - The Customer Mobile Application must call the `onUserAuthenticationInputAborted` method of the `UserAuthenticationInputCallback` object if the user aborts the user authentication.
- `onUserAuthenticationInputError`. This is called when the Orchestration SDK Client needs to inform the `UserAuthenticationCallback` method that there is an issue with the input.

  NOTE: For iOS, different names are used: `UserAuthenticationCallback` is called `UserAuthenticationDelegate`, and `UserAuthenticationInputCallback` is called `UserAuthenticationInputDelegate`.

NOTE: For more information about this feature, see **3.2.7 Authentication methods**.

5.9 Change password

NOTE: New APIs are created for Swift users of the iOS SDK. For more information, refer to the Xcode API documentation on `ChangePasswordParameters` and `ChangePasswordDelegate`.

The Customer Mobile Application can initialize a change password process by calling the `startChangePassword` method of the `Orchestrator` object.

A `ChangePasswordParams` object must be provided as an input parameter of the `startChangePassword` method.

The `ChangePasswordParams` object must be initiated with the following parameters:

- *User identifier*. A string which uniquely identifies the user on the OneSpan Trusted Identity platform.
- *Cryptographic Application Index*: The index of the cryptographic application which must be used to generate the OTP.
- An object implementing the `ChangePasswordCallback` interface.

The `ChangePasswordCallback` interface is used by the Orchestration SDK to interact with the Customer Mobile Application during the change password process. It exposes the following methods:

- `onChangePasswordStepComplete`. Called when a step of the change password process is complete. The provided orchestration command must be sent to the server.
- `onChangePasswordSuccess`. Called upon successful change of the password.
- `onChangePasswordAborted`. Called when changing the password is cancelled.
- `onChangePasswordInputError`. Called upon an error in the change password process due to incorrect user input. The possible errors are listed in the `OrchestrationErrorCodes` class.

For more information about this feature, see **3.2.8 Change password**.

5.10 Device data collection

> **NOTE:** New APIs are created for Swift users of the iOS SDK. For more information, refer to the Xcode API documentation on `CDDCDataFeeder` and `CDDCMessageParameters`.

Device data collection can be optionally configured with a `CDDCParams` object. The following parameters can be configured:

- *Refresh interval*. The interval (in seconds) at which device data is refreshed. Default: 60 seconds.
- *Optional device data*. The collection of certain device data types (i.e. geolocation, Bluetooth, and Wi-Fi) requires specific permissions. By default, these data types are not collected. For more information about required permissions on Android, see **4.1 Integrating OneSpan Orchestration SDK with Android**.

Most of the device data is collected automatically by the Orchestration SDK. Device data which cannot be collected automatically can be provided by the Customer Mobile Application, by using the `CDDCDataFeeder` object.

The `CDDCDataFeeder` object must be retrieved using the `getCDDCDataFeeder` method of the `Orchestrator` object.

The `Orchestrator` object provides the `getCDDCMessage` method to transmit the collected device data to the OneSpan Trusted Identity platform without using an orchestration command.

A `CDDCMessageParams` object must be provided as an input parameter of the `getCDDCMessage` method.

The `CDDCMessageParams` object must be initiated with the following parameters:

- *User identifier*. A string which uniquely identifies the user on the OneSpan Trusted Identity platform.
- *Encrypted*. Indicates whether the data must be encrypted.

- *Event Name*. A string representing an event related to the collected data. For more information, refer to the `Client Device Data Collector SDK Integration Guide` (part of the OneSpan Risk Analytics package).
- *Application Data*. A JSON string representing additional data to add in the CDDC message. For more information, refer to the `Client Device Data Collector SDK Integration Guide` (part of the OneSpan Risk Analytics package).

For more information about this feature, see **3.2.9 Device data collection**.

5.11 Multi-user management

The `UserManager` object provides the following methods:

- `getUsers`. Lists the activated users.
- `isUserActivated`. Checks whether the provided user identifier corresponds to an activated user.
- `getUserInformation`. Retrieves the information related to a specific user.
- `deleteUser`. Deactivates a specified user.

The `UserManager` object must be retrieved using the `getUserManager` method of the `Orchestrator` object.

For more information about this feature, see **3.2.10 Multi-user management**.

Index