credorax

Source 3D Secure

SDK User Guide

API Version 1.1 Rev 1

June 2020

Table of Contents

Introduction

The Credorax in-App kit (SDK) for 3D secure 2.0 provides strong customer authentication in in-app purchases, while increasing the cardholder's seamless experience, as follows:

- The SDK enables gathering unique device information that assists the issuer in their TRA (Transaction Risk Analysis)
- When a challenge is needed, the SDK is responsible for communication with the issuer side

Useful Documents / References

The following documents may be useful in understanding the 3D secure SDK:

Source Payment Gateway API	The Source Payment API specification provides detailed information on processing card-not-present transactions.
Credorax SDK Package	The SDK package provided by the Solutions team
Credorax API Request samples, opens text files with examples	Initiation Request
	Initiation Response
	Areq Request

Intended Audience

This document is intended for merchants wishing to implement the 3D secure in-App Kit (SDK) functionality as part of their complete implementation of the Source Gateway 3D secure services.

Certification

All new implementations must undergo certification to ensure the quality of integrations and integrity of merchant data. Please note that only test-card data should be used for testing.

Additional certification may be required if new services or features are to be used.

Publication Identification

Overview

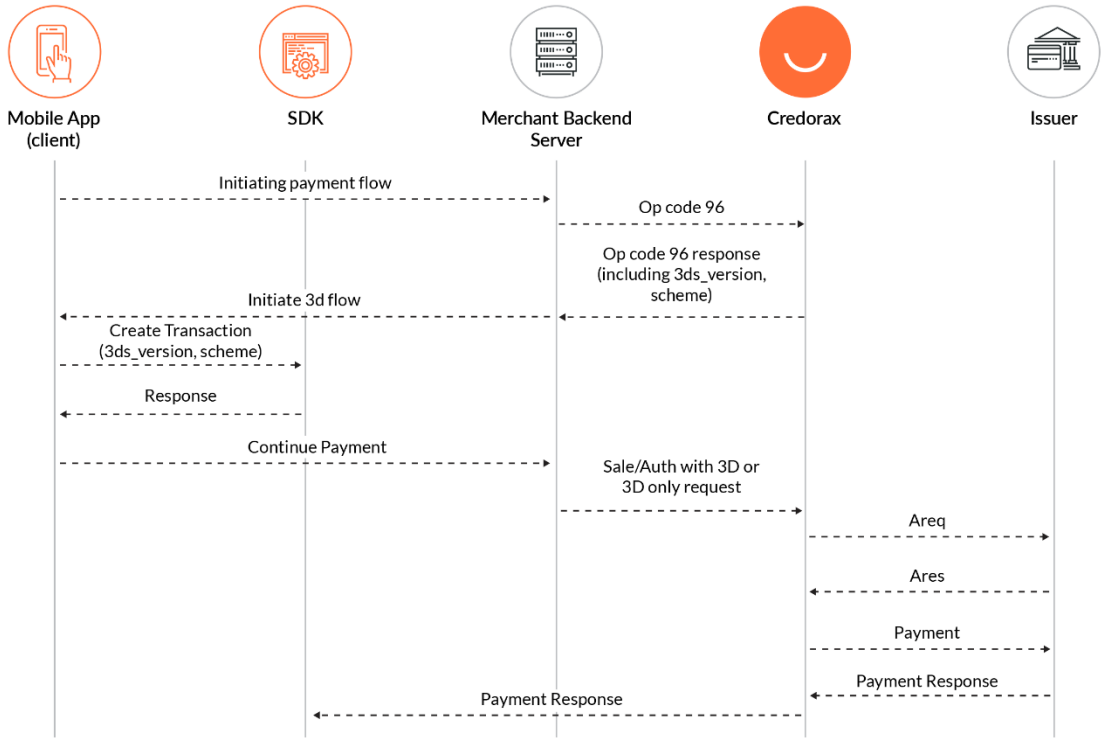
Introduction

The 3D secure in-App kit is a ready-made SDK that should be implemented as part of the native mobile application. The SDK is relevant only when the issuer supports 3D Secure version 2.1.0 and above.

Transaction Flow

When the SDK is implemented, the transaction flow is very similar to the flow that occurs when the payment is generated though a browser, except for some minor changes.

Frictionless Flow – Server to Server scenario



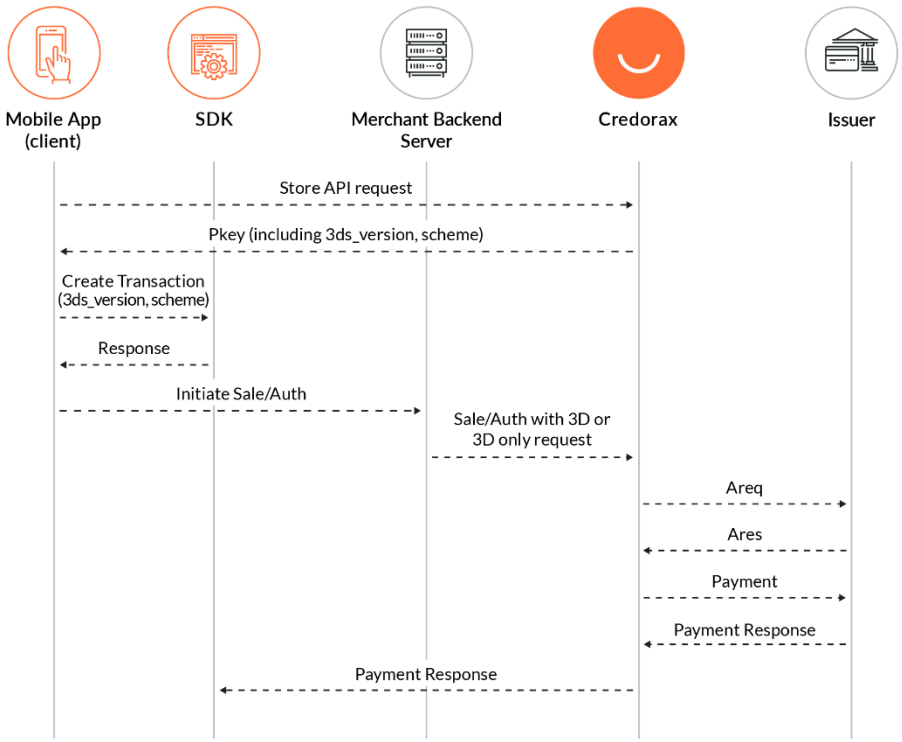
Explanation

1. Once the application identifies that a payment should be generated, the Merchant Backend Server triggers sending *operation code 96* to Credorax.

2. Credorax checks the 3D secure version supported by the issuer. If the version is lower than 2.1.0 or the issuer does not support 3D secure 2.0, the SDK is not relevant; in that case, refer to *What to do when 3ds_version is not 2.1.0 or above*.
3. If the issuer supports 3D secure version 2.0, the application triggers a "Create Transaction" call to the SDK. This request must include the supported 3D secure version as received in the operation code 96 response.
4. The SDK responds with information it gathered from the device itself.
5. The application passes the information to the merchant backend server as described in *Table 1 – Mapping of retrieved information*, and the merchant backend server uses this information in a regular payment request with 3D secure.

   Keep in mind that the payment request must include a *3ds_initiate* parameter with a value of 01 or 03, and a *3ds_channel* parameter with a value of 01.
6. From this point on, the flow is as described in Appendix I in the *Credorax Payment API* guide.

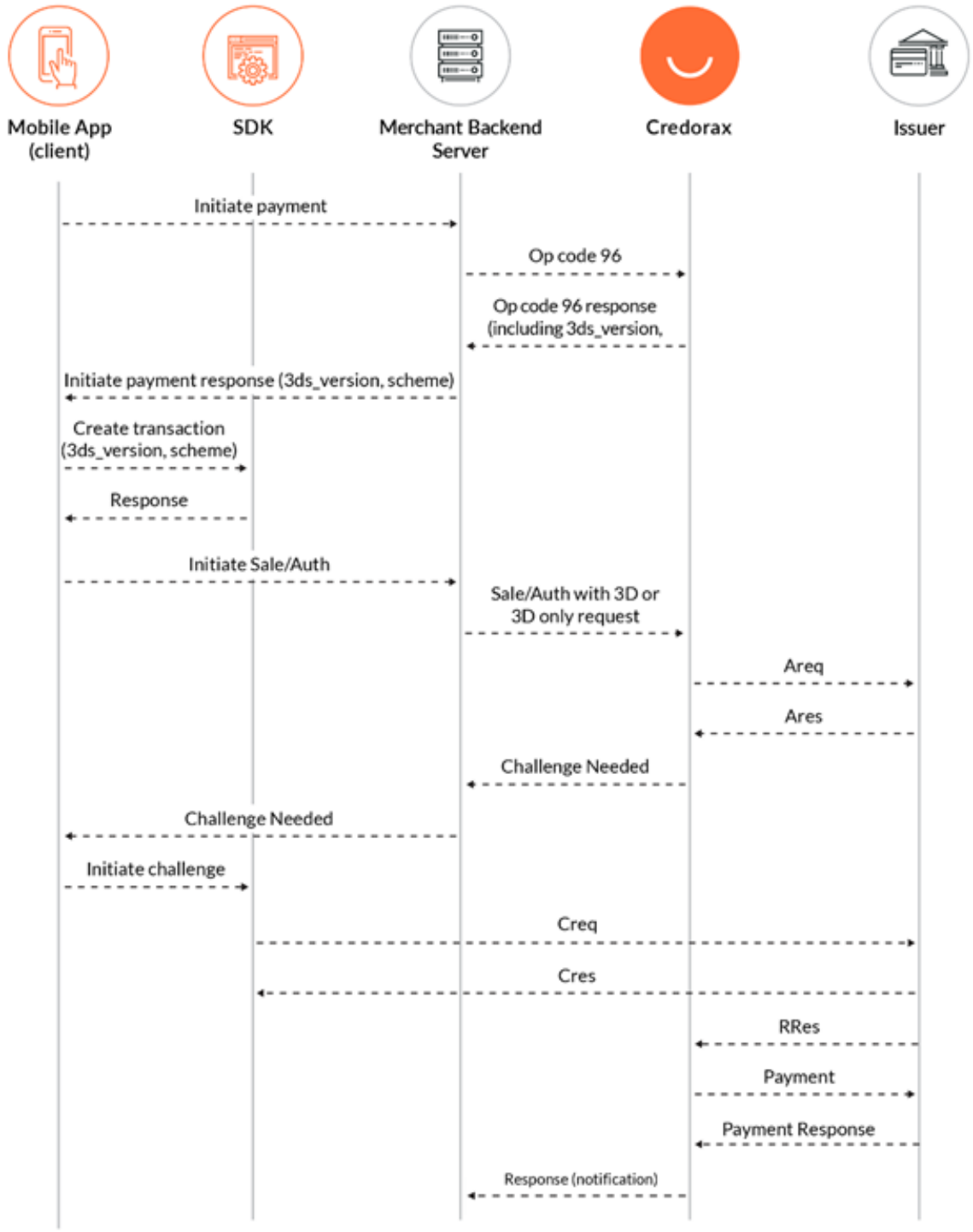
Frictionless Flow – Hosted Payment Page (HPP) Code



1. Once the application identifies that a payment should be generated, the Application triggers sending a *Store* request to Credorax.
2. Credorax checks the version supported by the issuer and returns the 3ds_version alongside the Pkey parameter. If the version is lower than 2.1.0 or the issuer does not support 3D secure 2.0, the SDK is not relevant; in that case, refer to *What to do when 3ds_version is not 2.1.0 or above*.
3. If the issuer supports 3D secure version 2.0, the application triggers a "Create Transaction" call to the SDK. This request must include the supported 3D secure version as received in the *Store* response.
4. The SDK responds with information it gathered from the device itself.
5. The application passes this information to the Merchant Backend Server as described in *Table 1 – Mapping of retrieved information*, and the merchant backend server uses this information in a regular payment request with 3D secure.

   Keep in mind that the payment request must include a *3ds_initiate* parameter with a value of 01 or 03, and a *3ds_channel* parameter with a value of 01.
6. From this point, the flow is as described in Appendix I in the *Source Payment Gateway API* guide.

Challenge Flow – Server to Server scenario



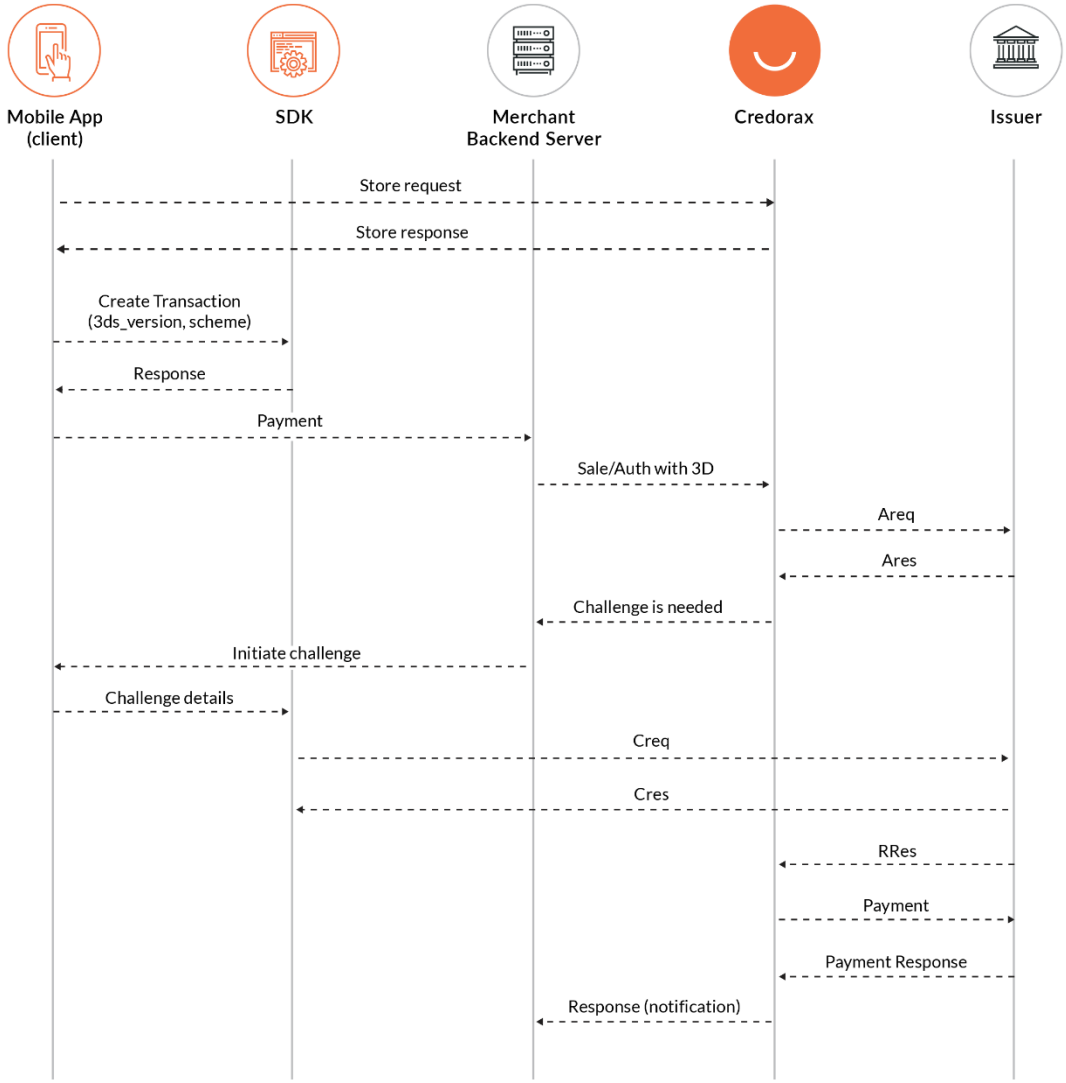
1. Once the application identifies that a payment should be generated, the Merchant Backend Server triggers sending *operation code 96* to Credorax.

2. Credorax checks the 3D secure version supported by the issuer. If the version is lower than 2.1.0 or the issuer does not support 3D secure 2.0, the SDK is not relevant; in that case, refer to *What to do when 3ds_version is not 2.1.0 or above*.
3. If the issuer supports 3D secure version 2.0, the application triggers a "Create Transaction" call to the SDK. This request must include the supported 3D secure version as received in the operation code 96 response.
4. The SDK responds with information it gathered from the device itself.
5. The application passes this information to the merchant backend server as described in *Table 1 – Mapping of retrieved information*, and the merchant backend server uses this information in a regular payment request with 3D secure.

   Keep in mind that the payment request must include a *3ds_initiate* parameter with a value of 01 or 03, and a *3ds_channel* parameter with a value of 01.
6. In its ARes response, the issuer decides that a challenge is needed (*3ds_status* = "C"). Source returns the response to the merchant backend server in the *3ds_acssignedcontent* parameter.
7. The mobile application processes with a challenge flow by calling a "doChallenge()" method on a previously initialised Transaction object.

   In the series of CReq and CRes that follows, the mobile application is idle since the entire interaction with the Access Control Server (ACS) is handled by the SDK.
8. After the challenge flow is completed, the application receives back the session with the authentication results from the issuer. You should ignore this and wait for the notification from Source.
9. Source initiates the payment (only if it was requested by the merchant) according to the authentication results, and returns the payment results in a notification to the merchant backend server.

Challenge Flow – Hosted Payment Page (HPP) Code



1. Once the application identifies that a payment should be generated, the Application triggers sending a *Store* request to Credorax.
2. Credorax checks the version supported by the issuer and returns the 3ds_version alongside the Pkey parameter. If the version is lower than 2.1.0 or the issuer does not support 3D secure 2.0, the SDK is not relevant; in that case, refer to *What to do when 3ds_version is not 2.1.0 or above*.
3. If the issuer supports 3D secure version 2.0, the application triggers a "Create Transaction" call to the SDK. This request must include the supported 3D secure version as received in the *Store* response.
4. The SDK responds with information it gathered from the device itself.

5. The application passes this information to the Merchant Backend Server as described in *Table 1 – Mapping of retrieved information*, and the merchant backend server uses this information in a regular payment request with 3D secure.

   Keep in mind that the payment request must include a *3ds_initiate* parameter with a value of 01 or 03, and a *3ds_channel* parameter with a value of 01.

6. In its ARes response, the issuer decides that a challenge is needed (*3ds_status* = "C"). Source returns the response to the merchant backend server in the *3ds_acssignedcontent* parameter.

7. The mobile application processes with a challenge flow by calling a "doChallenge()" method on a previously initialised Transaction object.

   In the series of CReq and CRes that follows, the mobile application is idle since the entire interaction with the Access Control Server (ACS) is handled by the SDK.

8. After the challenge flow is completed, the application receives back the session with the authentication results from the issuer. You should ignore this and wait for the notification from Source.

9. Source initiates the payment (only if it was requested by the merchant) according to the authentication results, and returns the payment results in a notification to the merchant backend server.

What to do when 3ds_version is not 2.1.0 or above

If the *store* API or *operation code 96* return a 3D secure version number that is lower than 2.1.0 (meaning version 1.0), or the issuer does not support 3D secure, the SDK is not relevant. In this case, if you wish to initiate a 3D secure process, do not activate the SDK but send the request as described in Appendix I of the *Source Payment Gateway API* guide, with *3ds_channel*=02 (Browser). If the *3ds_acsurl* parameter is received, open a web-view within the app to initiate the authentication flow.

System Requirements

Android

Criteria	Details
IDE	Any
Dev language	Any language that supports JAVA 7 and above
Android Version	4.4 and above

iOS

Criteria	Details
IDE	Xcode10
Dev language	Swift 4.2, Objective-C
Android Version	iOS 10 +and above

Operation code 96

Request format:

Parameter name	Description	Type	m/o
M	Source assigned gateway merchant ID	[A-Z0-9_]	m
K	Unique cipher used for authenticating requests. Refer to *Appendix A: Message Cipher* in the *Source Payment Gateway API Specifications* for further details on how to generate the cipher.	[0-9A-Za-z]	m
O	Operation Code. The operation code is used for determining the requested service. See the list in *Basic Operations* in the *Source Payment Gateway API Specifications*.	[0-9]	m
a1	Request ID. A unique transaction reference number. It should be unique to each transaction and to each MID. May be used when corresponding with the payment processor or reconciling transactions. Note: No plaintext cardholder data should be provided in this field.	[A-Za-z0-9-]	m
b1	PAN – Primary Account Number	[0-9]	m

Response format:

Parameter Name	Description	Type	m/o
M	Source assigned gateway merchant ID	[A-Z0-9_]	m
K	Unique cipher used for authenticating requests. Refer to *Appendix A: Message Cipher* in the *Source Payment Gateway API Specifications* for further details on how to generate the cipher.	[0-9A-Za-z]	m
O	Operation Code. The operation code is used for determining the requested service. See the list in *Basic Operations* in the *Source Payment Gateway API Specifications*.	[0-9]	m
3ds_trxid	The assigned 3D transaction ID	[a-zA-Z0-9, -]	m
3ds_version	The 3D protocol version supported by the issuer	[0-9]	m

Parameter Name	Description	Type	m/o
3ds_method	The issuer's URL that should be used to trigger the collection of the device fingerprint by the issuer	URL	o

SDK Configuration

This section explains how to configure the SDK before first use.

When a transaction is initiated, the SDK receives information about the Directory Server (DS) that will participate in the message flow. With that, the SDK determines which properties to use.

Directory Server configuration consists of configuring the following.

Schemes

Specify the schemes that can be used.

Android

To define the DS schemes, use `ConfigParameters.addParam(...)` with the following arguments:

Argument	Value
group	null
paramName	"schema_names"
paramValue	Comma separated values of schemes

Example:

```
configParameters.addParam(null, "schema_names", "mastercard, visa");
```

iOS

To define a DS scheme, simply create a dictionary from the root of the `DSsConfiguration.plist` where the name of the scheme is the dictionary key. Each DS scheme dictionary contains the following:

- **dsPublicKey** – a public key in PEM format, or a certificate in PEM or DER format
- **dsRootCertificate** – a string value representation of the encryption certificate filename
- **IDs** – an array containing string representations of the DS IDs
- **logoImageName** – a string value representation of the scheme logo image name.

Please note that the key names must be set exactly as stated above for the configuration to be valid.

DS IDs

Specify the DS IDs that belong to a certain scheme.

Android

To define the DS IDs that belong to a certain scheme, use `ConfigParameters.addParam(...)` with the following arguments:

Argument	Value
group	"schema_ds_ids"
paramName	The name of the scheme to which these DS IDs belong
paramValue	Comma separated list of DS IDs for the scheme

The merchant mobile application uses the Cardholder Account Number and optionally other cardholder information to identify the DS ID. A DS ID is the scheme's Card Brand.

The following table lists the code of each scheme.

Code	Card Scheme
1	Visa
2	Mastercard
3	American Express
4	Isracard
9	Maestro
10	JCB
12	Discover
13	Diners

Example:

```
configParameters.addParam("schema_ds_ids", "mastercard", 2)
configParameters.addParam("schema_ds_ids", "visa", 1)
```

iOS

To define the DS IDs that belong to a certain DS scheme, create a new array named **IDs** containing string values of the DS IDs. Repeat for each scheme DS.

DS ID	
parent	Scheme Dictionary
key	IDs
value	ID of the scheme, each in a new row

The merchant mobile application uses the Cardholder Account Number and optionally other cardholder information to identify the DS ID. A DS ID is the scheme's Card Brand.

Schemes public key

Specify the public key for each scheme that will be used for encryption of device data.

Android

To define the public key that will be used for a certain scheme, use `ConfigParameters.addParam(...)` with the following arguments:

Argument	Value
group	"schema_public_key"
paramName	The name of the scheme for which a public key is defined
paramValue	ASN.1 encoding of the public key or the certificate (DER/PEM) in Base64 encoded format

Example:

```
configParameters.addParam("schema_public_key", "mastercard",
loadPublicKey("certificates/mastercard_rsa.cer"));
configParameters.addParam("schema_public_key", "visa",
loadPublicKey("certificates/visa_ec.cer"));
```

iOS

To define the public key that will be used for encryption of Device Info, create a new element in the selected scheme name. The key should be *dsPublicKey* while the value should be the name of the DS certificate or the public key to be used.

DS Public Key	
parent	Scheme Dictionary
key	dsPublicKey
value	Name of the encryption certificate, or public key

Schemes root public key

For each scheme specify the public keys that will be used for verification of the ACS certificate chain.

Android

To define the root public key that will be used for a certain scheme, use `ConfigParameters.addParam(...)` with the following arguments:

Argument	Value
group	"schema_root_public_key"
paramName	The name of the scheme for which a root public key is defined
paramValue	ASN.1 encoding of the public key or the certificate (DER/PEM) in Base64 encoded format

Note that if the provided certificate is a chain of certificates, the first certificate in the list is used.

Example:

```
configParameters.addParam("schema_root_public_key", "mastercard",
loadPublicKey("certificates/root_mastercard_rsa.cer"));
configParameters.addParam("schema_root_public_key", "visa",
loadPublicKey("certificates/root_visa_ec.cer"));
```

iOS

To define the root certificate that will be used for a certain DS scheme, create a new element in the selected scheme name. The key should be *dsRootCertificate* while the value should be the name of the DS Root certificate.

DS Root Certificate	
parent	Scheme Dictionary

DS Root Certificate	
key	dsRootCertificate
value	Name of the root certificate

Scheme logo Resource ID

For each scheme specify the Drawable Resource ID that will be used as the logo.

Android

To define the logo to be used for a certain scheme, use `ConfigParameters.addParam(...)` with the following arguments:

Argument	Value
group	"schema_logo"
paramName	The name of the scheme for which a logo is defined
paramValue	String value of the Drawable Resource ID

Example:

```
configParameters.addParam("schema_logo", "mastercard",
Integer.toString(R.drawable.schema_logo_mastercard_1));
configParameters.addParam("schema_logo", "visa",
Integer.toString(R.drawable.schema_logo_visa));
```

iOS

To define the logo that will be used for a certain DS scheme, create a new element in the scheme DS dictionary. The key should be *logoImageName*, while the value should be the name of the logo image for the DS scheme. Make sure that the logo image is in the application bundle. Repeat the process for each DS scheme.

DS Logo image	
parent	Scheme Dictionary
key	logoImageName
value	Name of image filename

Android Permissions

This section lists the mandatory and optional Android permissions.

Permission	Mandatory	Description
`android.permission.INTERNET`	Yes	Required for communication with the ACS during the Challenge Flow. This permission must be granted before *Starting the Challenge Flow*.
`android.permission.ACCESS_COARSE_LOCATION`	No	Required for collecting the device coarse location that will be provided as a device info parameter. This permission should be granted before *Initialisation*.
`android.permission.ACCESS_FINE_LOCATION`	No	Required for collecting the device fine location that will be provided as a device info parameter. This permission should be granted before *Initialisation*.
`android.permissions.ACCESS_NETWORK_STATE`	No	Required for collecting the device IP Address that will be provided as a device info parameter. This permission should be granted before *Initialisation*.
`android.permission.BLUETOOTH`	No	Required for collecting bluetooth hardware info that will be provided as a device info parameter. This permission should be granted before *Initialisation*.
`android.permission.READ_PHONE_STATE`	No	Required for collecting telephony information that will be provided as device info parameters. This permission should be granted before *Initialisation*.

Permission	Mandatory	Description
`android.permission.SEND_SMS`	No	Required for collecting SMS information that will be provided as device info parameters. This permission should be granted before *Initialisation*.
`android.permission.ACCESS_WIFI_STATE`	No	Required for collecting WiFi and network-related information that will be provided as device info parameters. This permission should be granted before *Initialisation*.

Implementation Guidelines

SDK integration – Android

The SDK can be integrated manually or using a Maven Repository.

Using a Maven repository:

1. Upload on a private Maven Repository the SDK, Javadoc and maven.pom artefacts.
2. Add the Android 3DS SDK as dependency:

```
dependencies {
 implementation 'com.netcetera.android-3dssdk:3ds-sdk:2.0.0'

 // other dependencies
}
```

Manual integration:

1. Download the SDK Package, as received from the Credorax Solutions team.
2. On Android Studio, navigate to **File** > **New** > **Module** > **AAR Package**. Follow the wizard to include the SDK .AAR artefact as a module.
3. Add the 3DS SDK module as dependency:

```
dependencies {
 implementation project(":3ds-sdk")

 // other dependencies
}
```

credorax source

SDK integration – iOS

The SDK can be integrated manually or using CocoaPods.

CocoaPods integration

CocoaPods is the most popular dependency manager for iOS applications.

1. Add the 3DS SDK in the list of pods in Podfile.

```
platform :ios, '10.0'


source 'https://github.com/CocoaPods/Specs.git'
source 'https://github.com/ios-3ds-sdk/Specs.git'


target 'ThreeDS-Requestor-App' do
 # Comment the next line if you're not using Swift and don't want to use
dynamic frameworks
 use_frameworks!
 pod 'ThreeDS_SDK', '2.0.0'
For development you can use the 'universal' subspec to be able to run the app
on simulator
 # pod 'ThreeDS_SDK/universal', '2.0.0'
end
```

2. Add the credentials provided by Netcetera to the ~/.netrc file:

```
machine merchant-plug-in.extranet.netcetera.biz
login <username>
password <password>
```

3. Run the installation to install the pods:

```
pod install
```

Manual integration

Manual integration can be used by any app with or without a dependency manager. Developers can perform manual integration using the provided download link, as follows:

1. Download NetceteraiOS 3DS SDK zip from the download link. It contains 2 folders:
   - `iphoneos` – for builds in productions
   - `universal` – can be used for development
2. Add the framework (iphoneos or universal) in **Build Phases** > **Embedded Binaries**.

License Setup - Android

1. Open the license (.lic) from the delivery package with any text editor.
2. Insert the contents into **ConfigParameters** using "**license-key**" as the parameter name.

```
ConfigParameters.addParam(null, "license-key",
"eyJhbGciOiJSUzI1NiJ9.eyJ2ZXJzaW9uIjoyLCJ2YWxpZC11bnRpbCI6IjIwMzAtMTIiLCJuYW1lIjoiTmV0Y2V0ZXJhIi" +
 "wibW9kdWxlIjoiM0RTIn0.b6OUNt1OTUDzdGtJ8C6d5Bco2JgdX0hmNWab0hvhz_vfXhDQMtzjSpdpqLQciq6_dYFsrav2G-mbci1Ax" +
 "-t1KcRUQxurf11iTZ23DVF6quS-L3KqBHuEPeec1AIOdd1-U3QEUTglCp7NDuIzixI7mqAhwwNjKHrARkDIISxksYAsHjkX3LvqnpLF1" +
 "M1YAwgp5SiAlplaWSh_d4ahKJn7WMjLt8fGfgt7TjuJZA94_d3yNxQqJ6sM-n0v_Scel6QoAingrvn7XBB65QIR4v9S2kkCx1BwYb5X" +
 "Y6HONoUXssL6_iutiWqp1fta0z6uPsADMhlDtMQ");
```

3. Pass the configured **ConfigParameters** in the SDK Initialisation.

License Setup - iOS

1. Use the license key from the provided license file.
2. To define the license, use the method **ConfigParameters.addParam(...)** with arguments:

```
try configParameters.addParam(group:nil,
                              paramName: "license-key"
                              paramValue:"eyJhbGciOiJS…")
```

3. Pass the **ConfigParameters** in SDK initialisation.

UI Customisation API

The UI customisation API is initiated by the SDK and is responsible for UI setup during the challenge flow.

The SDK includes various functions that support UI customisation.

Android example:

```
UiCustomization uiCustomization = new UiCustomization();

LabelCustomization labelCustomization = new LabelCustomization();
labelCustomization.setHeadingTextFontSize(14);
uiCustomization.setLabelCustomization(labelCustomization);

ButtonCustomization buttonCustomization = new ButtonCustomization();
buttonCustomization.setCornerRadius(5);
uiCustomization.setButtonCustomization(button Customization,
UiCustomization.ButtonType.SUBMIT);
```

iOS example:

```
Public class UiCustomization {
Public func setButtonCustomization(buttonCustomization: ButtonCustomization,
buttonType: ButtonType)
public func setButtonCustomization(buttonCustomization: ButtonCustomization,
btnType: String)
public func setToolbarCustomization(toolbarCustomization:
ToolbarCustomization)
public func setLabelCustomization(labelCustomization: LabelCustomization)
public func setTextBoxCustomization(textBoxCustomization:
TextBoxCustomization)
…
}
```

After setting up the uiCustomization, you need to send it as part of the SDK initialisation. The SDK has a default UI, so if none of these parameters are passed, the SDK will present the challenge flow in its default configuration.

Instantiation

To use the SDK, a first instance of "ThreeDS2Service" must be created.

Instantiation example:

Android	`ThreeDS2Service = new ThreeDS2ServiceLogic();`
iOS	`let threeDS2Service: ThreeDS2Service = ThreeDS2ServiceSDK()`

Initialisation

In order to use the ThreeDS2Service, it must be initialised using ConfigParameters. During the initialisation step, the SDK:

- Performs security checks
- Collects device information

Initialisation example:

Android	`try {`
	`  ConfigParameters configParameters = …;`
	
	`  ThreeDS2Service.initialize(context, configParameters, locale, uicustomization);`
	`} catch (InvalidInputException | SDKRuntimeException | SDKAlreadyInitializedException e) {`
	`  //…`
	`}`

iOS	`do{`
	`  let configParameters = …`
	`  try threeDS2Service.initialize(configParameters,`
	`  locale; nil,`
	`  uiCustomization: nil)`
	`}`
	`Catch ThreeDS2Error.InvalidInput(let message, _) {//…}`
	`Catch ThreeDS2Error.SDKAlreadyInitialized(let message, _) {//…}`

Warnings

After the initialisation of the SDK, security checks have already been performed. Now the SDK can retrieve the outcome of the security checks as a list of Warnings. To obtain the result of these security checks, call **ThreeDS2Service.getWarnings**().

It's up to the merchant app to decide what action to perform when warnings are generated. These warnings are sent as part of the collected device info.

Android	`try {`
	`  List<Warning>warnings = threeDS2Service.getWarnings();`
	`  //Handle warnings`
	`} catch (SDKNotInitializedException e) {`
	`  //…`
	`}`

iOS	`do{`
	`  let sdkWarnings = try threeDS2Service.getWarnings()`
	`}`
	`Catch ThreeDS2Error.SDKNotInitialized(let message, _) {//…}`

Authentication

The 3DS Authentication flow starts with a request sent to the Source Gateway. The 3DS SDK generates authentication parameters that need to be used in this initial request.

Android	`Transaction transaction = threeDS2Service.createTransaction(directoryServerID, messageVersion)` `AuthenticationRequestParameters authenticationRequestParameters = transaction.getAuthenticationRequestParameters();`

iOS	`let directoryServerId = //…`
	`let transaction = threeDS2Service.createTransaction:`
	`directoryServerId: directoryServerId,`
	`messageVersion: "2.1.0")`
	`let transactionParameters = transaction.getAuthenticationRequestParameters()`

The retrieved information should be mapped as follows in the call to Credorax (Source API).

Table 1 – Mapping of retrieved information

3DS SDK output	Source API	Description
SDK Reference Number	3ds_sdkreferencenumber	Identification number of the SDK
SDK Application ID	3ds_sdkappid	Generated ID that is used to identify the application that is performing the transaction
SDK Transaction ID	3ds_sdktransid	Generated ID that is used to identify the ongoing transaction
Message version	3ds_version	The 3DS protocol version that shall be used for the upcoming message exchanges
SDK Ephemeral public key	3ds_sdkephempubkey	The public key that shall be used for establishing secure communication in the Challenge Flow
Device Data	3ds_sdkencdata	Set of device information and security warnings that are used for risk assessment

Starting the Challenge Flow

If the ACS assesses the transaction as high-risk – meaning the risk is above a certain threshold or the transaction requires a higher level of authentication – it forces Challenge Flow communication. In case of a Challenge Flow, the application needs to call **Transaction.doChallenge**(...), and the SDK takes over the Challenge process.

```
try {
  Activity activity = ... // getActivity();
  ChallengeParameters challengeParameters = ... // createChallengeParameters(authenticationResponse);
  ChallengeStatusReceiver challengeStatusReceiver = this;
  int timeOut = 5;
  transaction.doChallenge(activity, challengeParameters, challengeStatusReceiver, timeOut);
} catch (InvalidInputException e) {
   // ...
}
```

Once a challenge has been started, invocation on **Transaction.doChallenge**(...) and **Transaction.close**() will result in an SDK Runtime Error, but the Challenge Flow won't be interrupted, nor will the transaction be put in an invalid state. Once any result comes through the **ChallengeStatusReceiver**, calling of **Transaction.doChallenge**(...) and **Transaction.close**() is allowed if required.

Parameter	Description
Activity	The Activity that is in the foreground of the application.
challengeParameters	Instance of ChallengeParameters created with values from the Authentication Response. Credorax Gateway will return those in the response alongside the indication that a challenge is needed.
challengeStatusReceiver	Callback object that implements ChallengeStatusReceiver. It will be notified about the challenge status.
timeOut	Timeout interval (in minutes) within which the challenge process must be completed. The minimum timeout interval is defined to be 5 minutes.

Challenge Flow Results

After invoking **Transaction.doChallenge**(...), Challenge Flow is started and the control of the UI is handed over to the 3DS SDK. The Application has the controls back when any of the callback methods from the **ChallengeStatusReceiver** are invoked. Implement a class that conforms to the **ChallengeStatusReceiver** protocol.

```
class MyChallengeManager implements ChallengeStatusReceiver {

  public void completed(CompletionEvent completionEvent) {
    // Handle successfully or unsuccessful completion of challenge flow
  }

  public void cancelled() {
    // Handle challenge canceled by the user
  }

  public void timedout() {
    // Handle challenge timeout
  }

  public void protocolError(ProtocolErrorEvent protocolErrorEvent) {
    // Handle protocol error that has been send by the ACS
  }

  public void runtimeError(RuntimeErrorEvent runtimeErrorEvent) {
    // Handle error that has occurred in the SDK at runtime
  }
}
}
```

When any of the callback methods of the **ChallengeStatusReceiver** are invoked, the challenge flow UI is dismissed by the 3DS SDK and the Application has control of the UI.

Closing the Transaction

After 3DS Authentication is completed, the application should close the transaction by calling **Transaction.close**() in order to clear references and avoid possible memory leaks.

Cleaning up the ThreeDS2Service

Similar to closing the transaction, in order to free up resources which are used by the ThreeDS2Service, the **ThreeDS2Service.cleanup**() should be used. Once an instance of ThreeDS2Service has freed up the

used resources, it is in the same state as a newly created ThreeDS2Service and can be used again, but should go through *Initialisation* again. After the **ThreeDS2Service.cleanup**() is performed, previously created transaction objects are in an invalid state and shall not be used anymore. It is recommended to always create a new transaction object after initialising the service.

Change History

Version	Subject/Date	Description
1.1 Rev 1	June 2020	Removed references to RID's Changed 3ds_sdkrefnum to 3ds_sdkreferencenumber
1.1	August 2019	Added ability to retrieve 3D Secure version via Op Code 96
1.0	August 2019	First release

Need Support?

Contact our 24/7 Client Relations Center for any additional information or technical issue:

US: +1.617.715.1977

UK: +44.20.3608.1288

EU: +356.2778.0876

Email: support@credorax.com