Payment Device SDK for Android

Tap To Mobile Supplement for Release 5.4.2

NMI

© NMI. All rights reserved.

Table of contents

1. Introduction 4
2. Preparing for Development 5
   - Pre-requisites 5
     - Android Minimum Version 5
     - Restrictions 5
     - Geofencing Restrictions 5
   - Project Setup 6
     - Dependencies 6
     - Additional Permission Requirements 9
     - Extending ChipDnaMobileSDK Application Class 9
3. App Onboarding 11
   - Pre-requisites 11
   - Play Integrity Keys 12
   - Key Store Hash 16
     - Google Play Signed Apps 17
     - Self Managed Signing 17
4. Modes Of Operation 18
   - Configuration 18
     - Tap To Mobile-Only Environment 18
     - Tap To Mobile and Payment Device Environment 18
     - Default Payment Device Behavior 18
   - Transaction processing 19
     - Accepted Parameter Values 19
5. Tap To Mobile API Updates 19
6. Set Properties 19
7. Method Calls 20
   - Get Status 20
     - Behavior 20
     - New Response Parameters 20
     - Recommendation 20
   - Method API Doc 21
   - Connect and Configure 21
     - Required Parameters for Tap To Mobile 22
     - Parameters for Payment Device Configuration 22
     - Behavior 22

Method API Doc 23

Start Transaction 23

New Parameter Key 23

Accepted Parameter Values 23

Behavior 23

Method API Doc 24

Starting A Transaction 24

Card Tap Activity Requirements 25

Parameters Required for a Tap To Mobile Transaction 26

Parameters That Will Cause Errors for a Tap To Mobile Transaction 26

Terminate Transaction 27

8. Events 27

Connect And Configure Events 27

ConfigurationUpdate 27

Transaction Events 28

TransactionFinished 28

New Error Codes 29

Connect And Configure 29

Connect And Configure Finished Event 30

Start Transaction 30

Requested Activity 31

Transaction Finished Event 31

Appendix 1 - Contactless Symbol Reproduction Requirements 31

Reproduction Requirements 31

Display 32

Placement 33

Use with other brands 33

Additional Requirements 34

EMVCo Right to Review and Decline Use Cases 35

Appendix 2 - Sequence Diagrams 36

Running a Tap To Mobile Transaction 36

Configured with just Tap To Mobile 36

Configured with TTM & Payment Device. 37

Tap To Mobile Supplement

Android supplement for Payment Device SDK Release 5.4.2 (Cassiopeia).

Document History

Document Version	Software Version	Date (yyyy-mm-dd)	Summary of Changes
1.0.0	4.0.2	2025-06-23	Document Created
1.0.1	4.0.3	2025-07-18	Removed restrictions around use of bluetoothAdded additional fields now returned during a transaction finished event
1.0.2	4.0.5	2025-09-12	Updated document version
1.0.3	4.0.6	2025-11-28	Updated document version.Corrected SQLite Version.Updated SQLCipher Version.
1.0.4	5.1.0	2026-02-11	Updated document version.
1.0.5	5.2.0	2026-03-25	Updated document version.
1.0.6	5.3.1	2026-06-19	Updated document version.
1.0.7	5.4.1	2026-08-14	Updated Cloud Commerce SDK to 6.9.0. Added the getEnvironment() override. Deprecated CertificateFingerprint.
1.0.8	5.4.2	2026-10-30	Updated documentation version. Removed leftover CertificateFingerprint guidance. Updated the MerchantTippingNotSupported description. Added Customer Vault support, and Credential on File is now applied to Tap To Mobile transactions.

1. Introduction

This supplement document contains additional information about how to set up Tap To Mobile functionality in the Payment Device SDK.

This document details any pre-requisites required, configuring your development environment, and how to prepare your application for release. It also goes through the Tap To Mobile specific API changes that integrations will need to be aware of.

2. Preparing for Development

Pre-requisites

Android Minimum Version

In order to support all required Card Schemes, the minimum supported Android version for Payment Device SDK for Android has been updated to 12 and is now enforced to at least 11.

The following minimum versions are also required:

- Gradle: 8.6
- Java 17

Restrictions

Tap To Mobile is an always online feature and cannot function without an internet connection at the point of interaction.

Offline transactions are not supported as they are restricted by payment industry's security standards for Tap To Mobile transactions.

When using Tap To Mobile in production developer options must be disabled and the device cannot be connected to a USB device due to MPoC security restrictions.

In cases where these restrictions are not met, attestation will fail and the appropriate errors are returned to the integrating app, these are detailed later in the document.

Geofencing Restrictions

Tap to Mobile employs geofencing as a fraud prevention measure. To ensure compliance during development, the API key used for testing must be associated with a merchant account registered in the same location where you are developing your application.

Additionally, transaction processing is currently restricted to the following countries:

- United States (US)
- United Kingdom (UK)
- All EEA countries (including Romania, Ukraine, and Poland)
- Canada
- China
- Indonesia
- Brazil
- South Africa
- India
- Vietnam
- Philippines
- Mexico
- Argentina
- Israel

If you require support for additional countries, please contact your account manager.

Project Setup

Dependencies

Implementing Tap To Pay requires adding the Cloud Commerce SDK to your Android project. There are two Cloud Commerce binaries provided due to MPoC security requirements, it ensures your production build remains lean, secure, and compliant. A test aar allows for debugging and testing of integration during development.

We include two AARs in our release package:

- Test AAR (cloud-commerce-sdk-debug): built as debuggable and preconfigured to use only sandbox endpoints. You get verbose logs and can't accidentally hit a live endpoint.
- Production AAR (cloud-commerce-sdk-release): non-debuggable and minimal logging and locked to live endpoints under full PCI-compliant security controls.

In the Android release these can be found at:

- Test SDK libs/cloud-commerce-sdk-debug-6.9.0.aar
- Production SDK libs/cloud-commerce-sdk-release-6.9.0.aar



To switch between the Production and Test SDKs, it is required to uninstall and reinstall the app - or clear its storage, since each environment requires its own storage.

To implement the Cloud Commerce SDK will require creating a new module for it. This module needs to contain the cloud commerce aar files and a build.gradle file. The gradle file wants to contain the following (depending on the aar you wish to implement):

configurations.maybeCreate("default")

artifacts.add("default",file("./cloud-commerce-sdk-debug-.aar"))

The settings.gradle file of the integrating application will also want to contain the lines, replacing {ModuleName} and {PathFromRepositoryRoot} with values of the new module containing the cloud-commerce-sdk :

```
include ':{ModuleName}'
project(':{ModuleName}).projectDir=new File('{PathFromRepositoryRoot}')
```

In addition the following projects must be implemented in the applications app level build.gradle file:

- implementation project (":CloudCommerceSDK")

In your application should also implement the following dependencies:

```
implementation 'com.google.android.play:integrity:1.1.0'
implementation 'com.squareup.retrofit2:adapter-rxjava3:2.9.0'
implementation 'com.google.android.gms:play-services-location:21.0.1'
implementation 'com.squareup.okhttp3:logging-interceptor:4.12.0'
implementation 'io.reactivex.rxjava3:rxjava:3.0.0'
implementation 'io.reactivex.rxjava3:rxandroid:3.0.0'
implementation 'com.squareup.retrofit2:converter-scalars:2.9.0'
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.google.code.gson:gson:2.13.2'
implementation 'commons-codec:commons-codec:1.11'
implementation 'com.squareup.okhttp3:okhttp-urlconnection:4.12.0'
implementation 'androidx.security:security-crypto:1.1.0-alpha06'
implementation 'com.google.android.gms:play-services-safetynet:17.0.0'
implementation 'org.slf4j:slf4j-api:2.0.18'
implementation 'ch.qos.logback:logback-classic:1.5.32'
implementation fileTree(include: ['*.jar'], dir: 'libs')
implementation 'androidx.core:core-ktx:1.8.0'
implementation 'org.jetbrains.kotlin:kotlin-stdlib:1.6.0'
implementation 'net.zetetic:sqlcipher-android:4.7.2@aar'
implementation 'androidx.sqlite:sqlite:2.1.0'
implementation 'com.jakewharton.timber:timber:4.7.1'
```

Logback is required at runtime by the Cloud Commerce SDK's remote logging. Both logback-classic and logback-core provide META-INF/INDEX.LIST, which collides during resource merging, so any application that packages the full build flavour must add the following to the android block of its app level build.gradle:

```
packaging { resources { excludes += 'META-INF/INDEX.LIST' } }
```

Without the Logback and SLF4J dependencies the Cloud Commerce SDK cannot initialise and connect and configure returns FailedToInitializeTapToMobile.

Please look at the Payment Device SDK for Android - API for Release Documentation supplied within the release bundle for a more detailed breakdown of required dependencies for running the Payment Device SDK.



There must be no reference to org.apache.http.legacy in any build.gradle file.

Additional Permission Requirements

During configuration the Payment Device SDK will perform a suite of security checks, including requiring the integrating application to be granted location permissions. The recommended level of permission is:

- android.permission.READ_PHONE_STATE - allows read only access to the phone state and is used to obtain a unique identifier for the mobile device required for a configuration update and to read the IMSI from the SIM to verify the Home Network Identity.
- android.permission.BLUETOOTH and android.permission.BLUETOOTH_ADMIN - allows applications to connect to Bluetooth devices.
- android.permission.BLUETOOTH_SCAN and BLUETOOTH_CONNECT (SDK versions 31 and above) - allows applications to connect to Bluetooth devices.
- android.permission.ACCESS_COARSE_LOCATION (SDK versions 23 and above) and android.permission.ACCESS_FINE_LOCATION (SDK versions 30 and above) - allows applications to scan for and connect to Bluetooth Low Energy devices.
- android.permission.INTERNET - allows applications to open network sockets.
- android.permission.android.hardware.telephony feature - should be included and set to false, prevents the devices without telephony from being filtered out on Google Play.
- android.permission.android.hardware.usb feature - should be set in Applications planning to use a USB connection to prevent the application being shown to devices without USB on the Google Play store.
- android.permission.RECORD_AUDIO - for verifying if any recording is ongoing when entering a PIN
- android.permission.HIDE_OVERLAY_WINDOWS - Allows an SDK to prevent and verify if any non-system overlay windows are being drawn on top of the app
- android.permission.NFC - Allows an SDK to perform I/O operations over NFC.
- android.permission.VIBRATE - Required for vibrating the application on card detection.

Extending ChipDnaMobileSDK Application Class

To initialize the Payment Device SDK to allow TapToPay, it is required that the integrating application must contain an application class that extends ChipDnaApplication. This is due to MPoC requirements for the TapToPay solution. ChipDnaApplication manages initialisation of the Cloud Commerce SDK at application startup, enabling API consumption by the app module. In addition to this, it also performs security checks in the background and disables access to API further in case of any vulnerability detected.

The success of the SDK initialisation can be observed by the integrating app through the following override methods:

```
void onSDKInitializationSuccess()

void onSDKInitializationFailed(
errorMessage: String?
)
```

```
String getEnvironment()
```

It is required when implementing them to call the super of the methods.

getEnvironment() is optional and, unlike the initialisation callbacks, must not call super. Return ParameterValues.LiveEnvironment for production or ParameterValues.TestEnvironment for testing; when it is not overridden ParameterValues.LiveEnvironment is used. The value should agree with the ParameterKeys.Environment value supplied to setProperties().

The following is an example of a class extending the ChipDnaApplication.class.

open class DevApplication : ChipDnaApplication() {

```
// Optional. Defaults to ParameterValues.LiveEnvironment when not overridden.

override fun getEnvironment(): String = ParameterValues.TestEnvironment

override fun onSDKInitializationSuccess() {

    // The following line is required when implementing this method
```

super.onSDKInitializationSuccess()

```
    Log.d("ChipDnaApplication", "SDK Initialization Success")

}

override fun onSDKInitializationFailed(errorMessage: String?) {

    // The following line is required when implementing this method

    super.onSDKInitializationFailed(errorMessage)

    Log.e("ChipDnaApplication", "SDK Initialization Failed: $errorMessage")

    Toast.makeText(this, "Fail: $errorMessage", Toast.LENGTH_SHORT).show()

}
```

}

The application class that extends ChipDnaApplication must also be included in the AndroidManifest.xml. The following example shows a section of an AndroidManifest.xml with the required change highlighted, the name of the class matching that of the example above.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.BLUETOOTH_SCAN"/>
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
android:maxSdkVersion="29"/>
<uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE"/>

<application
android:name=".DevApplication"
```

3. App Onboarding

In order to ensure security and MPoC compliance, we require the following information to form our attestation:

- ***App Package name*** **- Package name of the integrating application.**
- **Key Store Hash** - This is a SHA-256 hash value from the keystore certificate.
- **Play Integrity Keys** - These are response encryption keys used to secure the communication between your Android application and the Google Play services.

The following sections will guide you through the process of onboarding your app and obtaining these values.

Please contact NMI at taptopay-app-onboarding@nmi.com, to share this information.

Pre-requisites

To manage your app's security, keys, and certificates, you will need to use the **Google Play Console**. This is the tool used to manage, release, and track your apps on the Google Play Store.

Additionally, you will need to create a **Google Play Project**, which integrates your app with backend services and APIs like the Google Play Developer API, and other Google services.

If you are not set up with a Google Play Console, you will need to register for a Console Developer account, refer to the guide provided by Google.

To create a Google Play Project through your Google Play Console:

- Navigate to **Menu menu** > **IAM & Admin** > **Create a Project**.
- Follow the on-screen instructions. More information about these steps can be found in creating-managing-projects in the google cloud documentation.

Play Integrity Keys

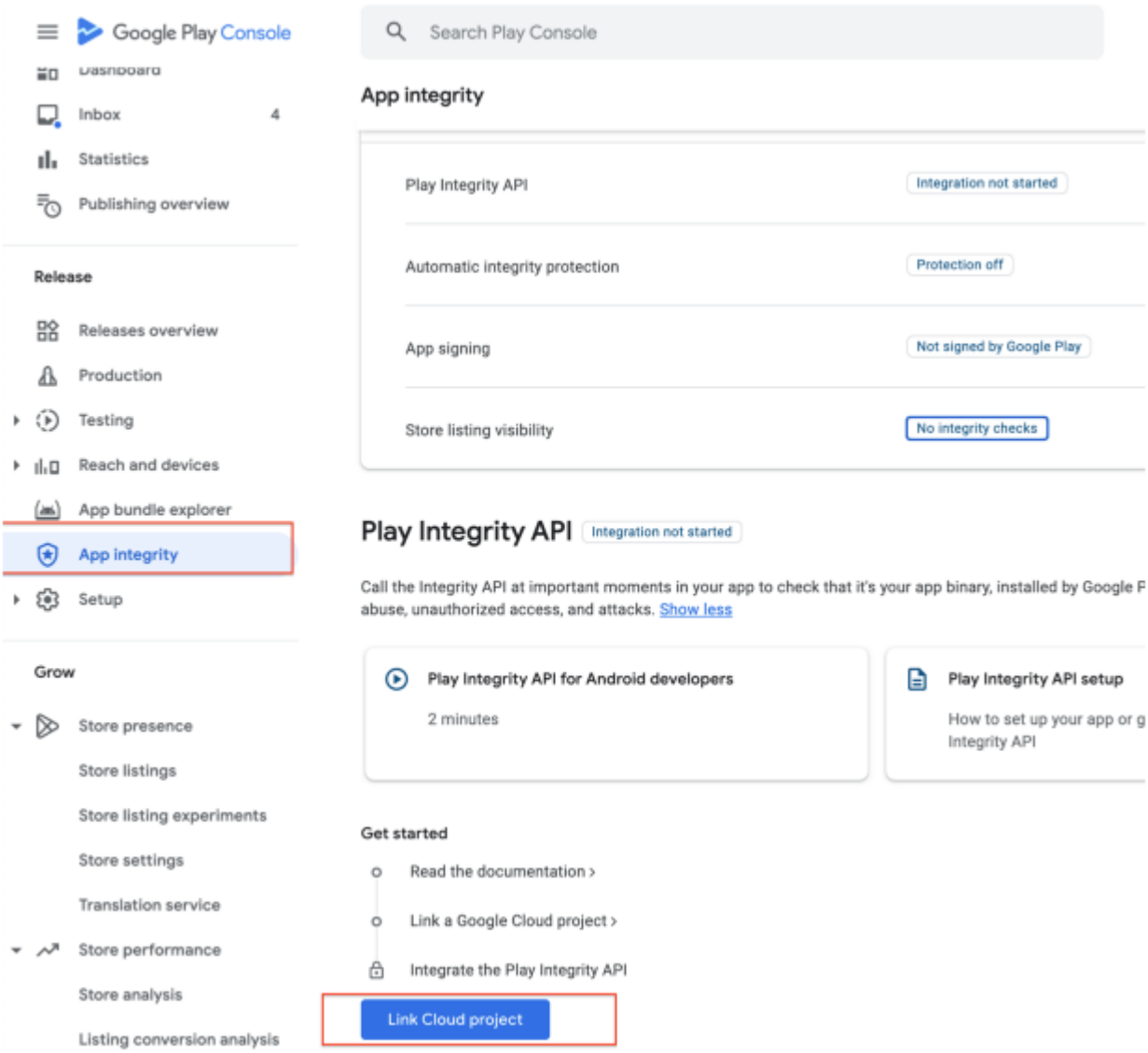
To obtain your Play Integrity keys from the Google Play Console, you must link your Google Cloud Project to the app.

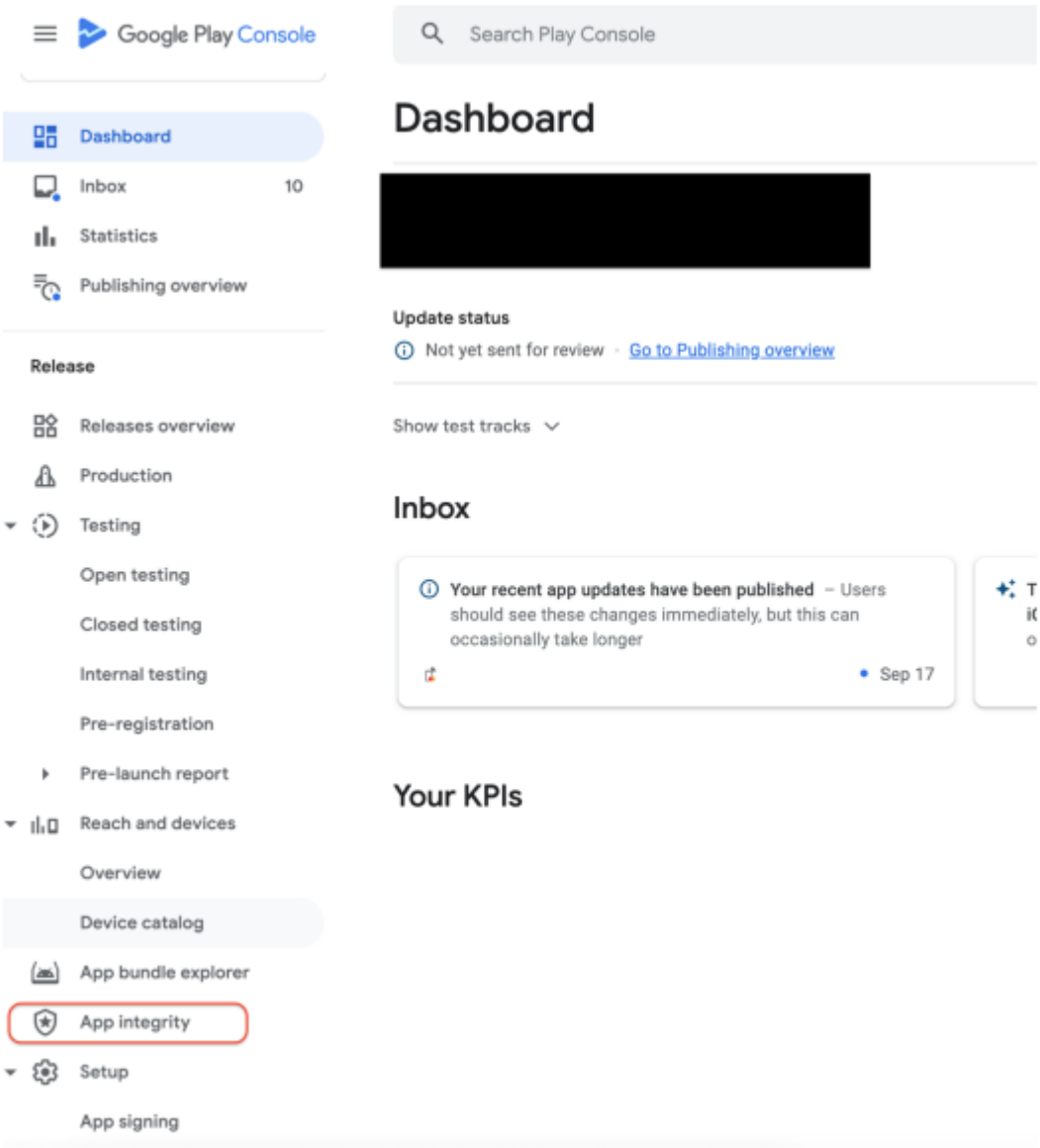
Ensure that you are the owner of the app for the next steps, as the app owner must match the owner of the Google Cloud Project.

Before starting the following steps, please ensure that you have requested a **PEM file** from us, this is required to obtain your Play Integrity keys.

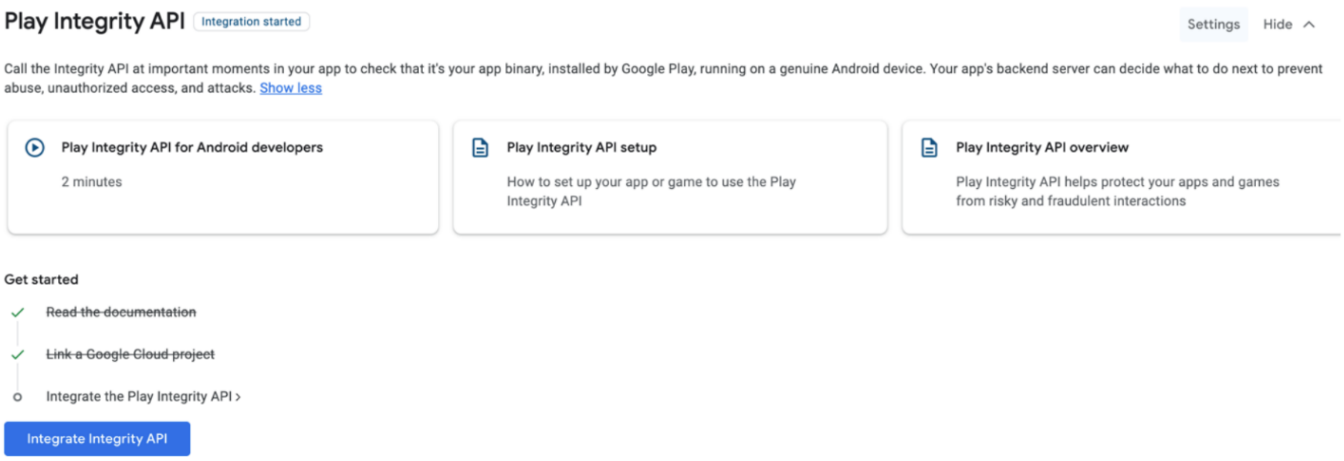
To link your Google Play Project you need to enable Play Integrity API via your Google Play Console.

- Navigate to **Release** > **App integrity**.
- Under **Play Integrity API** select **Link a Cloud project**.


Google Play Console
Search Play Console
Dashboard
Inbox 4
Statistics
Publishing overview
Release
Releases overview
Production
Testing
Reach and devices
App bundle explorer
App integrity
Setup
Grow
Store presence
Store listings
Store listing experiments
Store settings
Translation service
Store performance
Store analysis
Listing conversion analysis
App integrity
Play Integrity API
Integration not started
Automatic integrity protection
Protection off
App signing
Not signed by Google Play
Store listing visibility
No integrity checks
Play Integrity API
Integration not started
Call the Integrity API at important moments in your app to check that it's your app binary, installed by Google P
abuse, unauthorized access, and attacks. Show less
Play Integrity API for Android developers
2 minutes
Play Integrity API setup
How to set up your app or g
Integrity API
Get started
Read the documentation ›
Link a Google Cloud project ›
Integrate the Play Integrity API
Link Cloud project

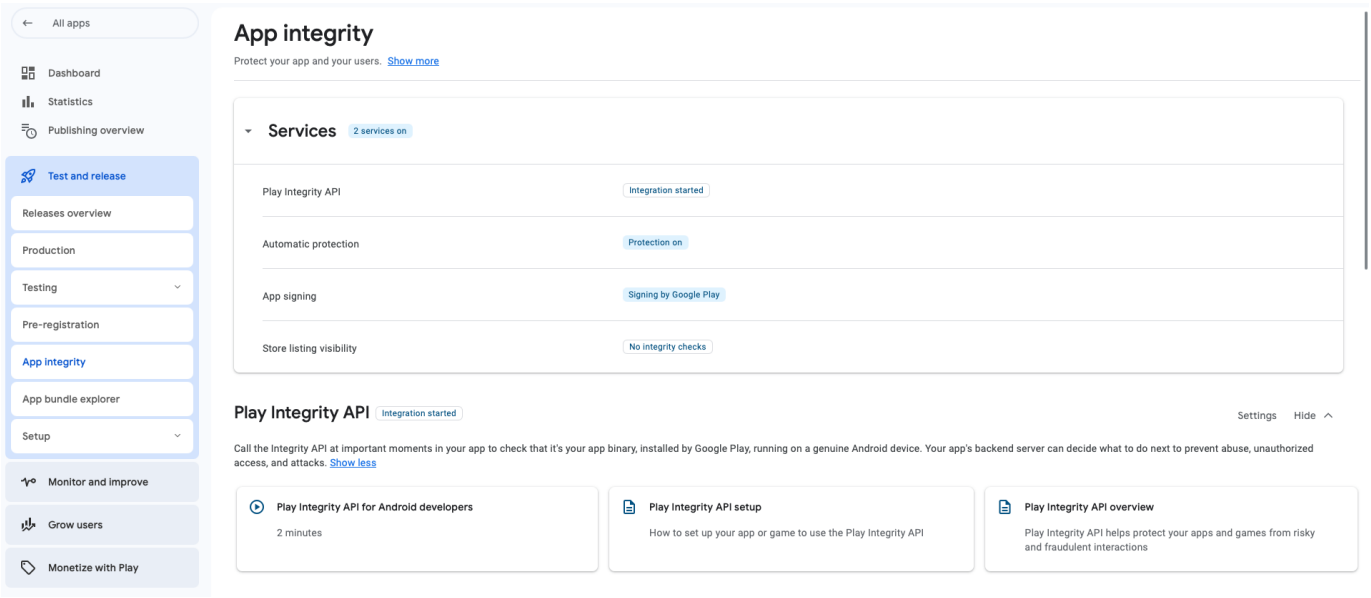


- Select a Cloud Project to link to your app and enable Play Integrity API for the project. For more details on this see Set up in Google Play Console in Androids documentation.
- Once this is successfully enabled you should see the button **Link Cloud Project** changes to Integrate the **Play Integrity API**.

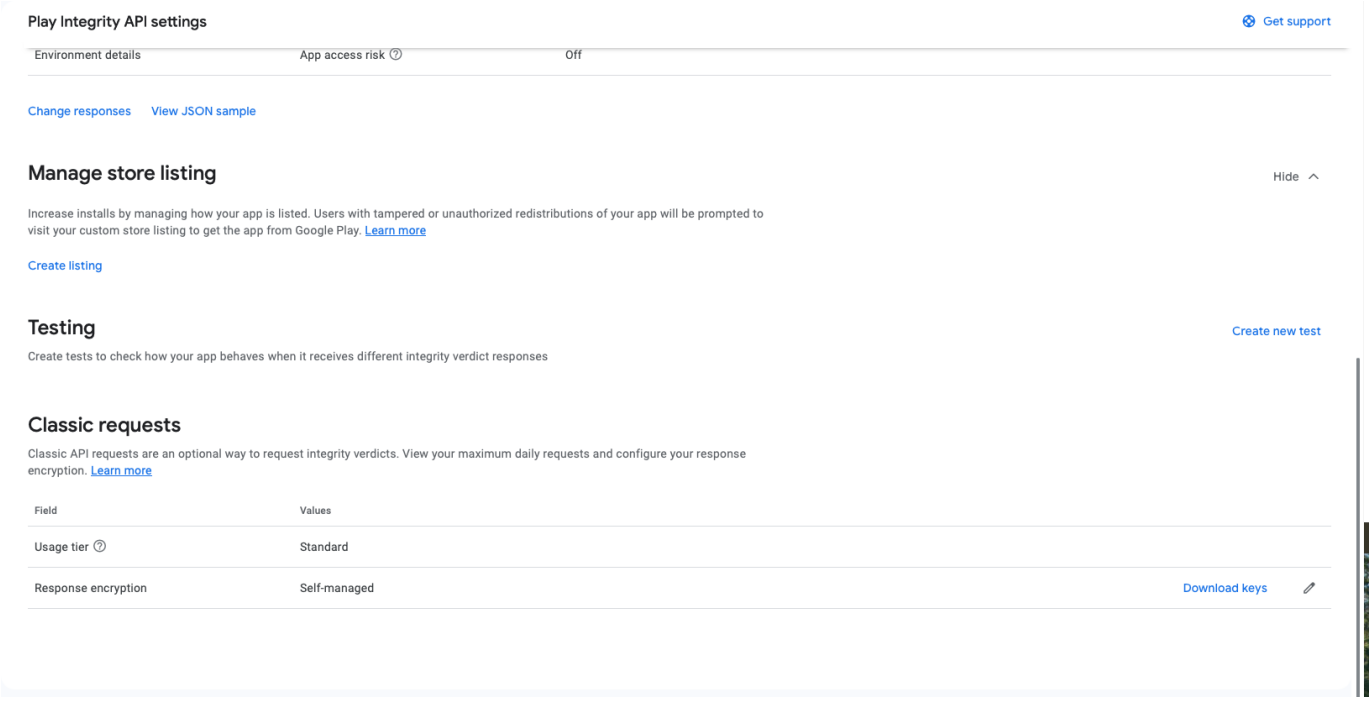


Once you have your PEM file, you can make a classic request to obtain your integrity keys. To do this your app must be on Google Play.

- Go to **Google Play Console** and select your app.
- Navigate to **Release** > **App integrity**.
- Select **Settings** next to **Play Integrity API**.



- Navigate to **Classic requests** where you can request and download your play integrity keys.
- Select **Edit** next to **Response encryption,** and a window should open.



- Select **Manage and download my response encryption keys**.
- Follow the instructions to upload the **PEM file** we provided you with.
- Once uploaded successfully click **Save** to download the encrypted keys.

For more information on requesting integrity keys see androids documentation.

When enabling play integrity for a new application, this will require a release that is prompted to at least Internal Testing.

Key Store Hash

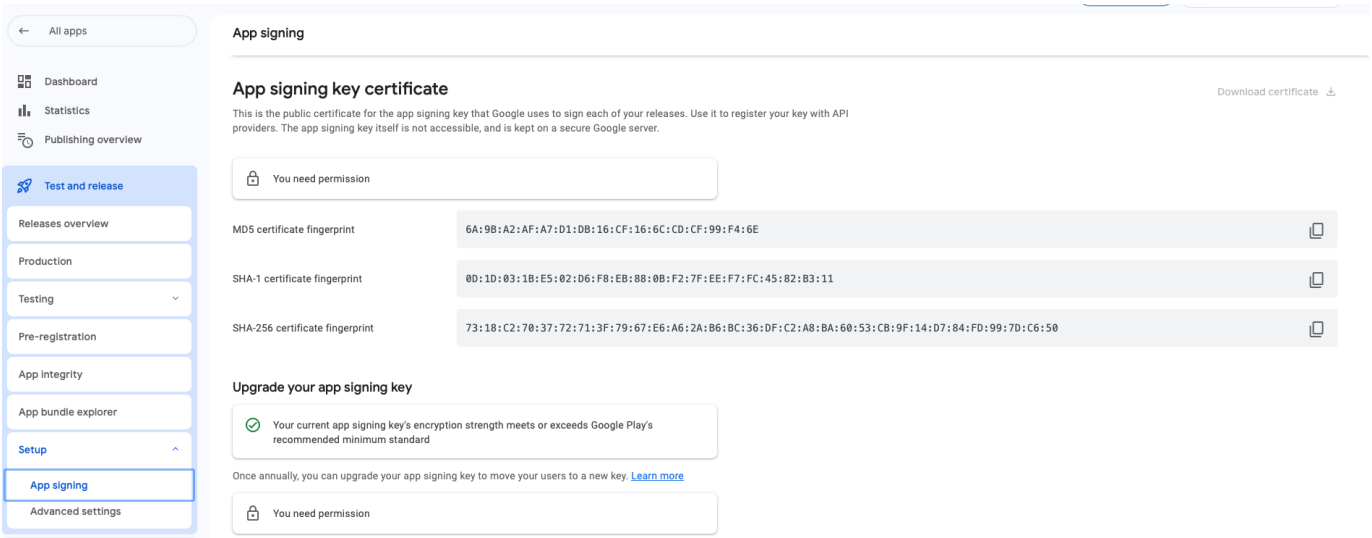
Ensure that the Base64 encoded SHA256 digest of the keystore certificate used for app signing is shared with NMI.

The keystore hash must come from the same signing key you use for your app—whether for production or testing. If you sign your debug APK with a different key than your release build, you'll need to supply two separate onboarding configurations (one for test, one for production). We use your onboarding data, including the keystore hash, to perform integrity checks. A mismatch will cause calls to connectAndConfigure to fail.

There are two methods of managing app signing that will mean obtaining the key store hash is different depending on the method you are implementing. The following sections will step through how this may be done for each method.

Google Play Signed Apps

If you're using google app signing the release key required can be found in the **Google Play Console**. The key found on **Google Play Console** is in hex format and so you will need to convert it to Base64 format before sharing it with us. The image below shows where you can find this on the **Google Play Console.**



Self Managed Signing

If you are managing your own keys and don't make use of Google Play Signing, you can use the following command to export the keystore certificate.

keytool -export -alias -keystore -rfc -file

The SHA-256 fingerprint can be fetched from keystore certificate using the following command:

openssl x509 -noout -fingerprint -sha256 -inform pem -in

This will provide the SHA-256 fingerprint in hex so will need to be converted to Base64.

To generate it in digest the following sample code can also be used:

```
public static String getSHA256Fingerprint(X509Certificate cert) throws Exception {
java.security.MessageDigest md = java.security.MessageDigest.getInstance("SHA-256");
byte[] der = cert.getEncoded();
md.update(der);
byte[] shaFingerprint = md.digest();
return java.util.Base64.getEncoder().encodeToString(shaFingerprint);
}
```

You will need to add this key to your **Google Play Console** if you are managing this yourself so please see their documentation to do.

For more information on how to sign your app, refer to the Android documentation on app signing.

4. Modes Of Operation

Payment Device SDK offers flexible transaction processing options. It can be configured to handle Tap to Pay transactions independently, process transactions through an external payment device, or operate in a combined mode. In the combined mode, transactions can be initiated simultaneously on both Tap to Mobile and an external payment device, allowing for versatile and seamless payment processing.

Configuration

During calls to connectAndConfigure you can request which mode the Payment Device SDK will operate in using the following parameters.

Tap To Mobile-Only Environment

- Include ParameterKeys.TapToMobilePOI with a value of ParameterValues.TRUE.
- Include ParameterKeys.PaymentDevicePOI with a value of ParameterValues.FALSE.

This configuration stops the Payment Device SDK from attempting to set up an external payment device.

Tap To Mobile and Payment Device Environment

- Include ParameterKeys.TapToMobilePOI with a value of ParameterValues.TRUE.
- Include ParameterKeys.PaymentDevicePOI with a value of ParameterValues.TRUE. This configuration enables the use of both Tap To Mobile and external payment devices.

Default Payment Device Behavior

If no POI Parmamer key is provided, the SDK will default to enabling the external payment device and Tap To Mobile transactions will not be offered.

Transaction processing

Once configured you can make use of the ParameterKeys.TransactionPOI parameter key to select which POI the transaction is started on.

Accepted Parameter Values

- ParameterValues.PaymentDevice: Specifies the use of a physical payment device for the transaction.

ParameterValues.TapToMobile: Specifies the use of the Tap To Mobile functionality for the transaction.

Note: when a single POI is configured, that POI will be used by default. When both have been configured the PaymentDevice will be preferred.

5. Tap To Mobile API Updates

6. Set Properties

There are no additional requirements around the setProperties method required to run Tap To Mobile transactions beyond the usual:

- API Key
- Application Identifier
- Environment
- External Device Info (if applicable)
- CertificateFingerprint - deprecated. This value is no longer used for Tap To Mobile and can be removed from your integration. It is no longer required by connect and configure.

The merchant account must have a valid Merchant Category Code (MCC) configured on the gateway. From Cloud Commerce SDK 6.9.0 the MCC is mandatory and is validated against an approved list; if it is missing or invalid, Tap To Mobile configuration fails. The MCC is taken from the merchant record and is not supplied by the integrating application.

7. Method Calls

Get Status

The getStatus method provides detailed information about the current status of the Payment Device SDK and the configured payment environment. It has been updated to include additional response parameters.

Behavior

The getStatus method now returns identifiers that distinguish between the configured payment methods. These identifiers are crucial for integration with NMI's payment gateway and for troubleshooting specific device-related issues.

New Response Parameters

- ParameterKeys.POSGUID
- An identifier representing the Point of Sale Device. This value is used to identify the Point of Sale with NMI's payment gateway.
- ParameterKeys.TapToMobilePOIIdentifier
- An identifier representing the Tap To Mobile Point of Interaction. This value is used to identify this instance of Tap To Mobile to NMI's payment gateway.

Recommendation

It's highly recommended to make this identifier accessible to application users to facilitate troubleshooting for specific devices.

Method API Doc

Parameters:
requestParameters - Parameters collection which can contain the following: ParameterKeys.PaymentPlatformTest (Optional) With a value of ParameterValues.TRUE, this will check the status of the payment platform, returning the result in a PaymentPlatformStatus object in the response. Returns:
Parameter collection containing the following: ParameterKeys.Result Defines if the call to getStatus was successful using the values ParameterValues.TRUE and ParameterValues.FALSE. ParameterKeys.VersionInformation An XML representation of a VersionInformation object. ChipDnaMobileSerializer.deserializeVersionInformation(String) can be used to retrieve an object. ParameterKeys.ChipDnaStatus A string representing the current status of ChipDNA Mobile. ParameterKeys.DeviceStatus An XML representation of a DeviceStatus object. ChipDnaMobileSerializer.deserializeDeviceStatus(String) can be used to retrieve an object. ParameterKeys.TmsStatus An XML representation of a TmsStatus object. ChipDnaMobileSerializer.deserializeTmsStatus(String) can be used to retrieve an object. ParameterKeys.TerminalStatus An XML representation of the live TerminalStatus. ChipDnaMobileSerializer.deserializeTerminalStatus(String) can be used to retrieve an object. ParameterKeys.Environment The current environment ChipDNA Mobile is running in. Will have a value of either ParameterValues.LiveEnvironment or ParameterValues.TestEnvironment. ParameterKeys.RequestQueueStatus An XML representation of a RequestQueueStatus object. ChipDnaMobileSerializer.deserializeRequestQueueStatus(String) can be used to retrieve an object. ParameterKeys.PaymentPlatformStatus An XML representation of a PaymentPlatformStatus object. ChipDnaMobileSerializer.deserializePaymentPlatformStatus(String) can be used to retrieve an object. ParameterKeys.LinkedRefundsSupported When present with the value ParameterValues.TRUE, this indicates that linked refunds are supported with the currently configured terminal. ParameterKeys.CashTransactionsSupported When present with the value ParameterValues.TRUE, this indicates that cash transactions are supported with the currently configured terminal. ParameterKeys.ChequeTransactionsSupported When present with the value ParameterValues.TRUE, this indicates that cheque transactions are supported with the currently configured terminal. ParameterKeys.TransactionHistorySupported When present with the value ParameterValues.TRUE, this indicates that transaction history is supported with the currently configured terminal. ParameterKeys.EmailReceiptSupported When present with the value ParameterValues.TRUE, this indicates that centralized email receipting is supported with the currently configured terminal. ParameterKeys.SmsReceiptSupported When present with the value ParameterValues.TRUE, this indicates that centralized SMS receipting is supported with the currently configured terminal. ParameterKeys.RefundOperatorPinSupported Indicates that an operator PIN must be supplied to perform standalone refunds with the currently configured terminal if present with the value ParameterValues.TRUE. ParameterKeys.OfflineProcessingSupported When present with the value ParameterValues.TRUE, this indicates that offline transaction processing is supported with the currently configured terminal. ParameterKeys.TippingSupported Indicates the type of tipping supported with the currently configured terminal. Values can be: - ParameterValues.EndOfDayTipping
- ParameterValues.OnDeviceTipping
- ParameterValues.BothTipping ParameterKeys.ApplicationIdentifier A string representing the current identifier for the application. ParameterKeys.MerchantDisplayName Indicates the currently configured merchant display name, if present. ParameterKeys.PinPadName Indicates the currently configured PINpad name, if present. ParameterKeys.PinPadConnectionType Indicates the currently configured PINpad connection type, if present. ParameterKeys.PinPadIpAddress Indicates the currently configured PINpad IP address, if present. ParameterKeys.PinPadPort Indicates the currently configured PINpad port, if present. ParameterKeys.FirmwareUpdateAvailable Indicates whether a firmware update is available for the currently configured PINpad. ParameterKeys.FirmwareUpdateStatus An XML representation of a FirmwareUpdateStatus object. ChipDnaMobileSerializer.deserializeFirmwareUpdateStatus(String) can be used to retrieve the object. ParameterKeys.POSGUID A string representing the current point of sale globally unique identifier. ParameterKeys.TapToMobilePOIIdentifier A string representing the current TTM point of interaction identifier.

Connect and Configure

The connectAndConfigure method enables the setup and configuration of both Tap To Mobile and external payment devices. The behavior of this method is determined by the parameters provided during the call.

Required Parameters for Tap To Mobile

- ParameterKeys.TapToMobilePOI
- **Value**: ParameterValues.TRUE This parameter must be included to trigger the configuration of Tap To Mobile.

Parameters for Payment Device Configuration

- ParameterKeys.PaymentDevicePOI
- **Value**: ParameterValues.TRUE This value enables the Payment Device SDK to configure and use an external payment device for transactions.
- **Value**: ParameterValues.FALSE This value prevents the Payment Device SDK from attempting to configure an external payment device, enabling a Tap To Mobile-only environment.

Behavior

1. Tap To Mobile-Only Environment

- Include ParameterKeys.TapToMobilePOI with a value of ParameterValues.TRUE.
- Include ParameterKeys.PaymentDevicePOI with a value of ParameterValues.FALSE. This configuration stops the Payment Device SDK from attempting to set up an external payment device.
- Tap To Mobile and Payment Device Environment
- Include ParameterKeys.TapToMobilePOI with a value of ParameterValues.TRUE.
- Include ParameterKeys.PaymentDevicePOI with a value of ParameterValues.TRUE. This configuration enables the use of both Tap To Mobile and external wifi connected payment devices.
- Default Payment Device Behavior
- If no POI Parmamer key is provided, the SDK will default to enabling the external payment device.

Method API Doc

Parameters:
requestParameters - Parameters collection which can contain the following:
ParameterKeys.ForceTmsUpdate Force connectAndConfigure to perform a TMS update whether one is required or not.
ParameterKeys.FullTmsUpdate Force a full TMS update whether one is required or not. All configuration will be downloaded regardless of whether it's required.
ParameterKeys.BLEScanTime (Optional) Set the length of time to scan for Bluetooth Low Energy (BLE) devices. The value is required to be a string valued number between 1 and 30. The default value of 5 seconds will be used if this value is not available.
ParameterKeys.ApplyFirmwareUpdate (Optional) When present with the value ParameterValues.FALSE, this stops a firmware update beginning, if one is available. ParameterKeys.PaymentDevicePOI (Optional) Sets if the Payment Device point of interaction is to be configured for payment processing. Values can be ParameterValues.TRUE or ParameterValues.FALSE. (Defaults to ParameterValues.TRUE if the parameter key is not present).
ParameterKeys.TapToMobilePOI (Optional) Sets if the Tap To Mobile point of interaction is to be configured for payment processing. Values can be ParameterValues.TRUE or ParameterValues.FALSE. (Defaults to ParameterValues.FALSE if the parameter key is not present).Returns:A parameter collection which can contain:ParameterKeys.Result ParameterKeys.Errors

Start Transaction

The startTransaction method allows you to specify the type of payment to be performed during a transaction. A new parameter key has been introduced to facilitate this:

New Parameter Key

- ParameterKeys.TransactionPOI: This parameter determines the transaction point of interaction to be used.

Accepted Parameter Values

- ParameterValues.PaymentDevice: Specifies the use of a physical payment device for the transaction.
- ParameterValues.TapToMobile: Specifies the use of the Tap To Mobile functionality for the transaction.

Behavior

1. **Single Transaction POI Configured** If only one transaction POI is configured (either Payment Device or Tap To Mobile), it will be automatically selected when starting a transaction.
2. **Both Transaction POI's Configured** If both Payment Device and Tap To Mobile are configured, the wifi connected payment device will be used by default unless the Transaction POI parameter explicitly specifies Tap To Mobile.

Method API Doc

Parameters:
requestParameters - Parameters collection which can contain the following:
ParameterKeys.Amount (Required) The amount to be used in the transaction.
ParameterKeys.UserReference (Required) A unique reference for this transaction.
ParameterKeys.TransactionType (Required) The transaction type for this transaction. Values can be ParameterValues.Sale or ParameterValues.Refund or ParameterValues.AccountVerification.
ParameterKeys.Currency Set the currency for this transaction. Only required when ChipDnaMobile.getAvailableCurrencies(Parameters)returns more than one currency. If only a single currency is supported, it will be used by default.
ParameterKeys.PANKeyEntry Requests a PAN Key Entry transaction is started for a card not present transaction. Values can be ParameterValues.TRUE or ParameterValues.FALSE.
ParameterKeys.DelayOnlineProcessing Requests that online processing is delayed. Values can be ParameterValues.TRUE or ParameterValues.FALSE. In this case the transaction will process to the point of going online and then return a EncodedRequest to the registered ITransactionFinishedListener.
ParameterKeys.PaymentMethod Specifies the payment method being used for this transaction. Values can be ParameterValues.Card, ParameterValues.Cash or ParameterValues.Cheque.
ParameterKeys.CredentialOnFileFirstStore (Optional) Flags the transaction as the first store for a Credential on File transaction when the value is ParameterValues.TRUE.
ParameterKeys.CredentialOnFileReason (Optional) Indicates the reason for a Credential on File transaction. Only considered if the value for ParameterKeys.CredentialOnFileFirstStore is ParameterValues.TRUE. Values can be: - ParameterValues.ReasonUnscheduled
- ParameterValues.ReasonInstallment
- ParameterValues.ReasonIncremental
- ParameterValues.ReasonResubmission
- ParameterValues.ReasonDelayedCharge
- ParameterValues.ReasonReAuth
- ParameterValues.ReasonNoShow
ParameterKeys.CustomerVaultCommand (Optional) Stores the card in the payment gateway's Customer Vault. Values can be add-customer or update-customer. Maximum 199 characters.
ParameterKeys.CustomerVaultId (Optional) The Customer Vault record to use with ParameterKeys.CustomerVaultCommand, for example the record to update with update-customer. Maximum 199 characters.
ParameterKeys.OnDeviceTippingPrompt (Optional) Sets a custom tipping prompt for on-device tipping. Only supported on Miura M020 and M021 PIN pads.
ParameterKeys.DynamicTippingAmounts (Optional) or ParameterKeys.DynamicTippingPercentages (Optional) Enables on-device dynamic tipping using the provided amounts or percentages. Only one of the amount types can be used in a request. Dynamic tipping is currently only supported on Miura M020 and M021 PIN pads.
ParameterKeys.DynamicTippingHeader (Optional) Sets the header for the dynamic tipping screen. This is only supported if dynamic tipping is enabled.
ParameterKeys.AutoConfirm (Optional) Sets if auto confirmation of a transaction should happen. Values can be ParameterValues.TRUE or ParameterValues.FALSE. The default value is ParameterValues.FALSE, unless processing via Tap To Mobile, where the default value is ParameterValues.TRUE and ParameterValues.FALSE will not be accepted.
ParameterKeys.TransactionPOI (Optional) Sets the transaction point of interaction to be used for this transaction. If one POI has been configured via ChipDnaMobile.connectAndConfigure, then this will be the default value, if one is not present. If multiple POI have been configured, then the default value will be ParameterValues.PaymentDevice, if the parameter key is not present. Can be one of the following values: - ParameterValues.PaymentDevice
- ParameterValues.TapToMobileReturns:A parameter collection which can contain:ParameterKeys.Result ParameterKeys.Errors

Starting A Transaction

This section describes the parameters and callbacks required for a Tap To Mobile transaction, the parameters that will cause errors if used, and the parameters returned in a transaction finished event.

Card Tap Activity Requirements

To perform a transaction using Tap To Pay, an activity is required to be passed into the Payment Device SDK. This is so that the SDK can access NFC permissions to prompt and perform a card read. It is not necessary to create a new Activity to parse in, the integrating application can use the current activity.

As the activity is being used to prompt for card Tap, the activity must adhere to the EMVCo Contactless Symbol and supported Card Brand images requirements. The requirements around how these are to be used and displayed are in Appendix 1 - Contactless Symbol Reproduction Requirements.

To enable contactless payments TapToMobile, the integrating application must:

- Register a RequestActivityListener before calling startTransaction. This is necessary because the SDK needs an Activity to handle the NFC prompt for card reads.
- When the SDK requires an Activity, it triggers RequestActivityListener.onRequestActivity. The application must respond by calling continueRequestedActivity with an appropriate Activity.
- Note: The provided Activity must remain in the foreground and active until a CardTapped transaction update is received. If the Activity is destroyed or moved to the background too soon, the transaction may fail.
- Once the transaction proceeds, the SDK emits transaction events, including CardEntryPrompted, which enables card reading.
- When the transaction completes, the application is notified via onTransactionFinishedListener.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addRequestActivityListener(
    IRequestActivityListener listener
)

void removeRequestActivityListener(
    IRequestActivityListener listener
)

void clearAllRequestActivityListener()

Parameters continueRequestedActivity(
    Activity activity
)
```

Parameters Required for a Tap To Mobile Transaction

The following parameters are required to start a TTM transaction:

- ParameterKeys.Amount
- ParameterKeys.UserReference
- ParameterKeys.PaymentMethod
- Must be set to ParameterValues.Card.
- ParameterKeys.TransactionType
- Must be set to ParameterValues.Sale.

Parameters That Will Cause Errors for a Tap To Mobile Transaction

Tap To Mobile currently only supports auto confirmed sale transactions therefore the following parameters will return an error if included when starting a Tap To Mobile transaction:

- ParameterKeys.TransactionType
- ParameterValues.Refund
- ParameterValues.AccountVerification
- ParameterKeys.DelayOnlineProcessing
- ParameterValues.TRUE
- ParameterKeys.PANKeyEntry
- ParameterValues.TRUE
- ParameterKeys.TippingType
- ParameterValues.OnDeviceTipping
- ParameterValues.EndOfDayTipping
- ParameterValues.BothTipping
- ParameterKeys.AutoConfirm
- ParameterValues.FALSE

Terminate Transaction

```
Parameters terminateTransaction(Parameters)
```

To cancel a transaction in progress call terminateTransaction().

During a Tap To Pay transaction, terminateTransaction() can be called between the transaction updates CardEntryPrompted and CardTapped.

If the transaction update CardTapped has been received before terminateTransaction() is called and the result of the transaction when it has finished is Approved then voidTransaction() can be used to cancel the transaction.

8. Events

Connect And Configure Events

This section describes the events returned by the SDK during operation of Tap To Mobile, including those for the connectAndConfigure process, error codes, and transaction events.

ConfigurationUpdate

The following new events are emitted during the connectAndConfigure process:

CheckingTapToMobileConfig indicates that the Payment Device SDK is checking the current configuration of Tap To Mobile.

UpdatingTapToMobileConfig Indicates that the Payment Device SDK is updating its Tap To Mobile Configuration.

Transaction Events

TransactionFinished

The following parameters, if available, will be returned in a transaction finished event for a Tap To Mobile transaction:

- ParameterKeys.UserReference
- ParameterKeys.TransactionResult
- ParameterKeys.Amount
- ParameterKeys.AuthCode
- ParameterKeys.TransactionDateTime
- ParameterKeys.DateTimeFormat
- ParameterKeys.MaskedPan
- ParameterKeys.TransactionType
- ParameterKeys.Currency
- ParameterKeys.ReceiptData
- ParameterKeys.PreformattedMerchantReceipt
- ParameterKeys.PreformattedCustomerReceipt
- ParameterKeys.Errors
- ParameterKeys.Par
- ParameterKeys.CardHashCollection
- ParameterKeys.AcquirerResponseCode
- ParameterKeys.CardSchemeId
- ParameterKeys.CardReference
- ParameterKeys.TransactionState
- ParameterKeys.CardReference
- ParameterKeys.TransactionId
- ParameterKeys.CustomerVaultId

New Error Codes

This section details the new error codes added as part of Tap To Mobile. A full list of error codes can be viewed within the generated web documentation supplied within the release bundles docs directory. Navigate to index.html then under Modules > ChipDnaMobileErrorCodes.

Connect And Configure

The following new error codes may be returned when calling connectAndConfigure.

- TapToMobileNotSupported: Indicates that the current hardware does not support Tap To Mobile payment processing.
- NoPoiSelected: Indicates that no point of interaction was selected for configuration.
- LocationPermissionsNotGranted: Indicates that the application does not have location permissions granted, which are required for Tap To Mobile payment processing.
- ApplicationUpdateRequired: Indicates that the CloudCommerce SDK being used is now too old and an update is required. This can be for the following reasons:
- Pinned certificate rotation: The SDK embeds a certificate for communication with the Mastercard server. If this certificate is rotated, then an update of the SDK would be done, and an updated version of the Cloud Commerce Android SDK would be released. In such cases this will be a forced upgrade as the older SDK would cease to work due to expiration of the said certificate.
- Support For New Kernel: The current version of the CloudCommerce Android SDK supports 2 Kernels Mastercard and Visa. In future, if the support for a new kernel (e.g. AMEX) is added, then a new version of the SDK would be released. This would not be a forced upgrade as the older version of the SDK would still work without this upgrade.
- Changes in attestation: If any change to the attestation services or updates occur, then an updated version of the CloudCommerce Android SDK would be released.

Connect And Configure Finished Event

The following new error codes may be returned in the connectAndConfigureFinished event.

- CountryCodeInvalid: Indicates an error with the country code being used for Tap To Mobile.
- AttestationFailed: Indicates that the attestation process for Tap To Mobile has failed.
- CurrentCountryNotAllowed:The country of your merchant account doesn't match the location you're attempting to perform a transaction in.
- NoLocationFound: Tap To Mobile was unable to determine your location. Ensure your location permissions are correctly set.
- DeveloperOptionsEnabled: In production TapToMobile cannot be initialized when Developer options are enabled.
- USBCableConnectedOrBluetoothEnabled: Due to TapToMobile MPoC restrictions USB cannot be connected or Bluetooth Enabled in production.
- CustomROMDetected: Indicates device has custom ROM so will cause Attestation to fail.
- EmulatorFound: An emulator cannot be used with TapToMobile in a Production environment.
- ShowTouchesEnabled: Indicates that Show touches is enabled in Developer options.
- TerminalConfigurationError: Indicates that Tap To Mobile could not be configured because the terminal profile could not be found or the product type is not valid for this merchant. Check that the merchant account is fully provisioned for Tap To Mobile, including its Merchant Category Code.

Start Transaction

The following new error codes may be returned during calls to startTransaction.

- TransactionPOINotConnected: Indicates that the transaction point of interaction attempting to be used is not connected.
- TransactionPOIInvalid: Indicates that the transaction point of interaction being used is invalid.
- AutoConfirmRequired: Indicates that the transaction requires auto-confirmation.
- TipAmountInvalid:The tip amount supplied to start transaction isn't in a valid format
- TipAmountNotAllowed:Merchant Tipping is not supported for this TransactionPOI
- MerchantTippingNotSupported: Merchant Tipping is not enabled in TMS.
- RequestActivityListenerRequired: IRequestActivityListener must be observed and implemented to start a transaction with TapToMobile.

Requested Activity

The following new error codes may be returned when calling continueRequestedActivity.

- InvalidActivity: Indicates the activity required to prompt for card read is either null, finishing or destroyed.

Transaction Finished Event

The following new error codes may be returned in the transactionFinished event.

- MerchantTerminatedTransaction: Indicates that the transaction was terminated.
- TapToMobileSessionClosed: Indicates that the current Tap To Mobile Session is no longer available.
- USBCableConnectedOrBluetoothEnabled: Due to TapToMobile MPoC restrictions USB cannot be connected or Bluetooth Enabled in production.
- NfcDisabled: NFC must be enabled to perform a transaction.
- NoPatternOrPinSet: Indicates that the device doesn't have a pattern or a pin set.
- ShowTouchesEnabled: Indicates that Show touches is enabled in Developer options.
- CameraUsed: Indicates that the camera is being used by another application.
- MicrophoneUsed: Indicates that the microphone is being used by another application.

Appendix 1 - Contactless Symbol Reproduction Requirements

Due to the Card Tap Screen being supplied in the integrating application, there are some requirements as to the EMVCo Contactless Symbol that is used. The Contactless Symbol is a trademark owned by and used with permission of EMVCo, LLC.

Reproduction Requirements

These standards govern the use of the EMV® Contactless Symbol, a trademark of EMVCo, LLC., both at the point of sale terminals and within supporting marketing collateral. Following these guidelines will ensure that the mark is clearly displayed, understood and acted upon consistently, helping to ensure an optimal consumer experience.

A Trademark License Agreement must be in place with EMVCo, LLC for the right to use the EMV Contactless Symbol.

Display



- To ensure visual consistency, never alter the drawing, arrangement or proportion of the individual elements of the image.
- A minimum clear space of ¼ the height of the Contactless Symbol should be maintained on all four sides of the image, wherever it is used.
- It should always be unobstructed.
- The Contactless Symbol image appears optimally when shown no smaller than 6.5mm in height. The Contactless Symbol must appear at a size equal to the other brand imagery displayed.
- The Contactless Symbol must appear as a solid color that provides the best color contrast and legibility against the selected background. The background must be a solid color, it should be neutral and must provide sufficient contrast for the Symbol to be clear and visible.
- The center of the Contactless Symbol is located by bisecting the overall width (including hand/device) and height (oval shape).
- The Contactless Symbol must always appear in the visual orientation shown to the above; it must never be rotated or flipped.

Placement

- Contactless Symbol must be placed over the part of the antenna read area ("landing zone") where the signal is strongest to ensure the signal is activated when a consumer taps or holds a contactless enabled device near the Symbol. If the antenna read area is located behind the display screen of the terminal, then the Symbol must be displayed digitally in the appropriate area of the screen.
- If the Contactless Symbol is placed on the terminal display (vs. terminal “surround”), it must be visible before and during the transaction process to ensure a consumer is both aware of the terminal’s contactless capability (while making the payment decision) and knows where to tap or hold the payment device. Once a terminal is activated for payment, the Symbol must remain visible for a minimum of 15 seconds or for the duration of the polling period.
- The Symbol must only appear once on each terminal.
- The Symbol is intended only for payment-related use on EMV compliant point-of-sale terminals and should never appear on payment cards, other payment form factors, or for contactless transactions that are not payment-related.

Use with other brands

- When the Symbol and the payment brands are combined on the same panel,screen, or surface area of the terminal, it is recommended that the Contactless Symbol be separated from the payment brands with a 1 point key line (see graphic below).
- The Symbol cannot be given a proprietary name by any payment brand.



- The Symbol must be displayed on the contactless POS “read area” of any
- contactless enabled terminal during the time of payment transaction.
- The relevant Payment System Brand Marks** should be referenced on or near the contactless “read area” of any payment terminal that has been enabled for contactless payments during the time of payment transaction to indicate acceptance.
- The above standards are not intended to prevent any other logos from being displayed on the terminal.

**Payment System Brand Mark are those used to indicate which card brands are accepted by the POS terminal at the time of payment.

Additional Requirements

- Optional text may accompany the EMV Contactless Symbol when the point of sale terminal accepts only contactless payments. Contactless-only readers may only be deployed with the permission of the relevant network brands. Acceptable text is “Contactless Payment Only” or the local language equivalent. The text should be set in a sans serif font no larger than ¼ the height of the Symbol, and must be placed outside the minimum clear space of the Symbol.

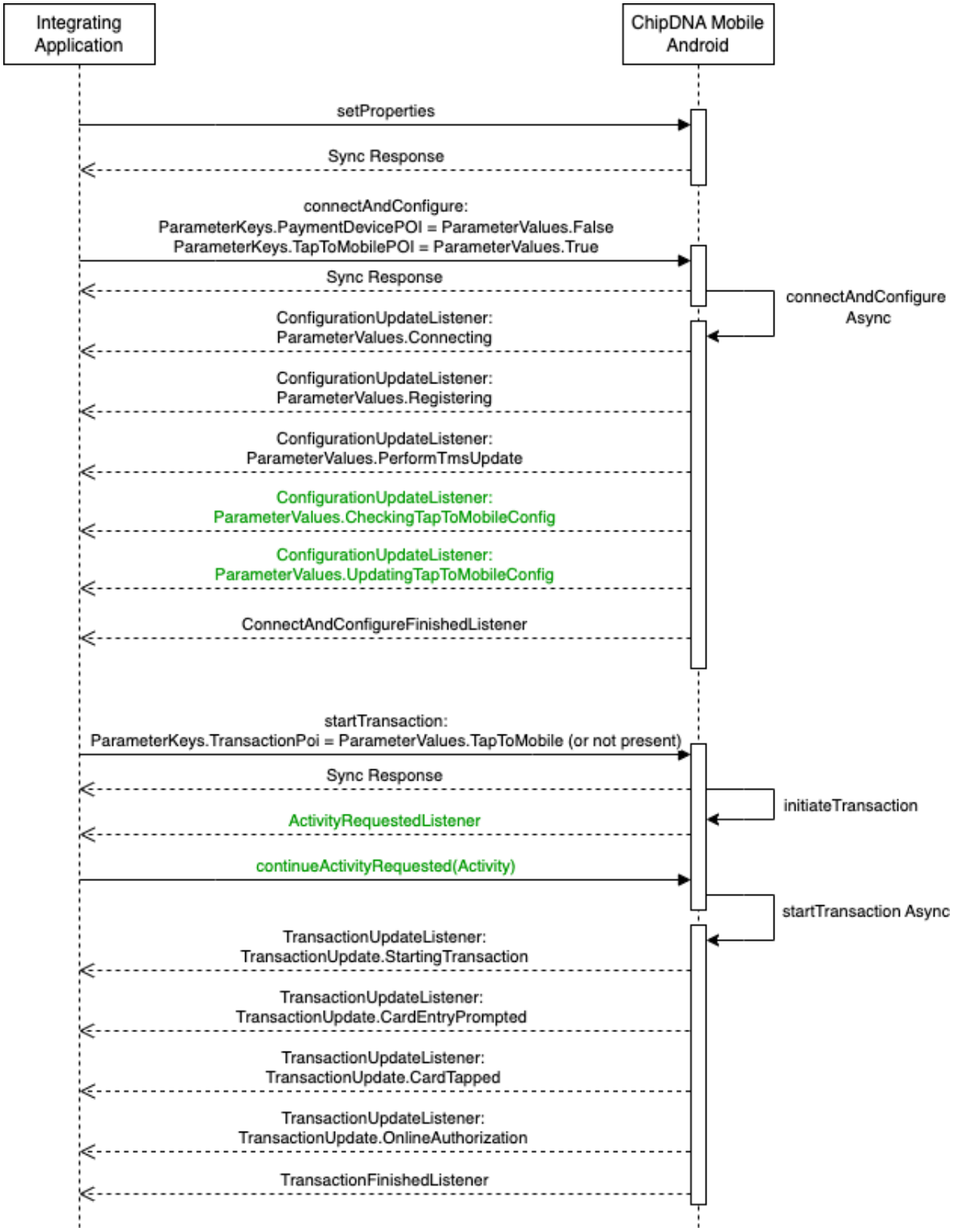
EMVCo Right to Review and Decline Use Cases

Conditions of the Trademark License Agreement require Licensees to inform EMVCo of any new use case that may fall outside of the Contactless Symbol scenarios outlined above. In the event that a proposed use case would cause market confusion, EMVCo reserves the right to decline a particular use case. EMVCo reserves the right to withdraw permission of use in instances of non-compliance with these guidelines. Licensees must inform EMVCo of new use cases via email at secretariat@emvco.com. In all cases it is the responsibility of the licensee or service provider to ensure quality assurance testing occurs to confirm the contactless solution deployed meets expected levels of performance and interoperability.

Appendix 2 - Sequence Diagrams

Running a Tap To Mobile Transaction

Configured with just Tap To Mobile



Configured with TTM & Payment Device.

