NOPASS SDK FOR ANDROID/IOS

NoPass Passwordless Registration and Authentication

Version: **2.0.2**
Dated: **15 April 2021**

NoPass Passwordless Registration and Authentication
NoPass SDK for Android/iOS
Version 2.0.2
NoPass version 1.1.0
15 April 2021

Identité™, Inc.
3035, Turtle Brooke, Clearwater, Florida, 33761, USA
Website: www.Identite.us

CONTENTS

OVERVIEW

NoPass™ SDK is a software developer kit that allows you to build the NoPass™ 3-factor authentication into your existing mobile applications. With our SDK you keep the user from having to install an additional app on their smartphone that performs authentication.

NoPass™ SDK main objective is to provide other developers with a convenient method for registration, authentication, and restoration. The SDK helps developers utilize the binaries and plug them into their existing code. This will accelerate the development of a new application or your company's existing application with NoPass™ Passwordless Authentication built in. The SDK gives you access to a host of NoPass™ authentication, security and user management features.

To learn more about our product, visit us at **https://www.identite.us/**.

If you need additional support, email Identitè at *support@identite.us.*

Document Conventions

The following guidelines present some specific conventions used in this manual.

ELEMENT	DESCRIPTION
	Note—Additional information about a subject.
	Warning—Indicates a potential obstacle or condition requiring special attention.
	Tip—A type of note that suggests alternative methods that may not be obvious.
\	Used as a line break. Do not type.
<…>	Used to denote placeholders.
Save	Names of buttons, windows, menu items and other program interface elements.
`sudo`	Code samples, including keywords and variables within text.
Prerequisites	Cross-references to the document chapters or internal hyperlinks.
https://dev.mysql.com/	Cross-references to external hyperlinks to web pages.

1. NoPass SDK for Android

The main components are as follows:

- **NoPassRegistrationManager**
- **NoPassUserManager**
- **NoPassUtils**

1.1 Initial Setup

1) Connect the NoPass maven repository to our SDK:

```
maven {
      url "https://repository.identite.us/repository/maven-public/"
}
```

2) Copy the file into the **build.gradle (Project: My_Application)** folder.

3) Add the dependencies to your module level gradle build file:

```
allprojects {
    repositories {
            google()
            jcenter()
            maven {
                url "https://repository.identite.us/repository/maven-
public/"
            }
        }
    }
```

```
implementation "com.android.support:multidex:1.0.2"
implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.0"
implementation 'androidx.appcompat:appcompat:1.1.0'
implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
implementation "us.identite.sdk:core:1.0.0@aar"{
   exclude group: 'org.jetbrains.kotlin', module: 'kotlin-reflect'
}
implementation 'io.reactivex.rxjava2:rxjava:2.2.0'
implementation "io.reactivex.rxjava2:rxkotlin:2.2.0"
implementation "io.reactivex.rxjava2:rxandroid:2.0.0"
implementation "com.squareup.retrofit2:retrofit:2.3.0"
implementation "com.squareup.okhttp3:logging-interceptor:3.9.0"
implementation "com.squareup.retrofit2:converter-moshi:2.3.0"
implementation "com.squareup.retrofit2:adapter-rxjava2:2.3.0"
implementation("com.squareup.moshi:moshi-kotlin:1.5.0")
implementation "com.serjltt.moshi:moshi-lazy-adapters:2.1"
implementation "com.android.support:multidex:1.0.2"
implementation 'com.google.code.gson:gson:2.8.5'
implementation "androidx.room:room-runtime:2.1.0-alpha06"
implementation "androidx.room:room-rxjava2:2.1.0-alpha06"
implementation "com.scottyab:rootbeer-lib:0.0.7"
implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.0"
implementation 'com.google.http-client:google-http-client-gson:1.26.0'
implementation 'com.google.api-client:google-api-client-android:1.30.8'
implementation 'com.google.apis:google-api-services-drive:v3-rev194-
1.25.0'
implementation 'com.google.firebase:firebase-auth:19.2.0'
kapt "androidx.room:room-compiler:2.1.0-alpha06"
implementation 'com.google.firebase:firebase-messaging:20.2.3'
implementation 'com.google.firebase:firebase-analytics:17.4.4'
```

4) To make the core functionality work, hook the NoPass service to the firebase callbacks and provide the application instance.

Note: NoPass uses the Firebase cloud messaging system in its registration and authorization flows. So, in order to complete them successfully, you will have to set up the so-called *interceptors* inside your Firebase `onMessageReceived()` callback.

Simply provide the data for further processing. Any non-NoPass data will be ignored:

```
override fun onMessageReceived(remoteMessage: RemoteMessage) {
   interceptFirebaseMessage(remoteMessage.data)
}
```

Make sure the firebase token is provided correctly and will be available during registration and authorization flows:

```
interceptFirebaseToken(token)
```

5) And, finally, give us your application instance (inside your Application class `onCreate()` method:

```
provideApplication(this)
```

1.2 Registration

To create a secure user token, serving to authenticate a user, you will need to use the registration manager. Its only responsibility is to register a user account, containing all the necessary information. Once you have properly configured its instance singleton, you will be able to register new users.

Procedure

To setup the registration manager, do the following:

1) Instantiate the manager:

```
val manager = NoPassRegistrationManager.configure()
        .setOnConfirmationCodeRequiredListener {code ->
            //show it to a user
        }
        .setOnRegistrationResultListener { success, error/*null if no error
occurred*/ ->
           //react to the registration result.
         }
        .build()
```

2) You will need the parameters string from a QR-code, deep link or whatever way of delivering this data to the app you choose. Once you get it, you can trigger the registration process:

```
manager.register(paramsString)
```

Note: NoPass does not deliver any of the user data to outer networks since it is connected to your organization's on-premises backend system.

1.3 Authentication

Once you have registered the user, your NoPass backend server gets able to send authorization requests. To receive them, perform the following setup.

The **`NoPassUserManager`** class serves to receive authorization requests and send authorization responses.

To be able to receive and process requests do the following:

```
val userManager = NoPassUserManager.configure
       .setOnAuthRequestReceivedListener { noPassUser, keyphraseData, initialProgress ->
        //noPassUser contains the information about the user
        //keyphraseData contains picture and digits for showing to the user
        //initialProgress approximately shows how much time has passed since the request
//was sent
        //TODO: parse the user data to notify the user
}
       .setOnOTPChangedListener { keyphraseData ->
    //TODO: use the following data to show to the user inside the authentication message
    //the timeToLive property stands for the time in seconds the OTP and image values are
    //valid. After that time the callback will be invoked once again
}
       .setOnTickListener{ tick ->
    //current timer tick in seconds
}
       .setOnAuthenticationResultListener{success, error ->
    //TODO: process the server's response on auth request
}
       .setOnSessionTimeoutListener{user ->
    //the session is timed out. You can no longer authenticate during this session
}
    .build()
```

When the setup is ready, you can authorize your user during the authentication session by calling the following function:

```
userManager.authenticate()
```

or decline the authentication if needed:

```
userManager.decline(reason)
```

1.4 Other Operations

There are other operations that can be performed with the user manager.

1) Get all the users list:

```
userManager.getAllUsers { users, error ->

}
```

2) Get the authentication history:

```
userManager.getAuthenticationHistory { records, error ->

}
```

3) You can also delete the user from both backend and the mobile device:

```
userManager.deleteUser(params.id) {user, success, error ->
        //in case it is triggered from the app the result may be
        //processed in this callback
}

.setOnUserDeleteResultListener { noPassUser, success, throwable ->
    //react to the result. May be useful for "silent" deletions
}
```

4) User updating. This can occur only when initiated from the backend.

```
.setOnUserUpdateResultListener { noPassUser, success, throwable ->
    //react to the result
}
```

5) Backup. It is also possible to create a backup data string encrypted by a 6-digits pin-code. This data may be used to restore accounts later. To initiate backup, do the following:

```
userManager.backupAccounts(pin)

.setOnUsersBackupResultListener { backup,
                         usersList,
                         errors,
                         generalError ->
    //backup string is to be saved somewhere to preserve
}
```

6) Restore. If there are properly backed up accounts, they can be restored by a backup string and the pin code. To begin restoration, run the command:

```
userManager.restoreAccounts(backupData, pin)

.setOnUsersRestoreResultListener { allSuccess,
                          usersList,
                          errors,
                          generalError ->
}
```

Restore/backup implies a series of interactions for each user account and does not break the whole procedure on particular user errors. Hence, the errors parameter is represented by a map of users and errors associated with them to provide more detailed information about what could go wrong.
The `generalError` is returned when an error that is not related to a certain user occurs.

2. NoPass SDK for iOS

2.1 Initial setup

Before you begin

1) In your **Xcode project** directory, create a Podfile, and add your dependencies:

```
use_frameworks!
target 'YourApp' do

  pod 'NoPass-iOS-SDK', :git =>
'https://github.com/identite/nopass.sdk.ios.git', :tag => '1.8.1'

end
```

Tip: CocoaPods provides a **`podinit`** command to create a Podfile with smart defaults. You should use it.

2) Now, you can install dependencies in your project: **`$pod install`**.

3) Make sure to always open the Xcode workspace instead of the project file building your project:

```
$ open YourApp.xcworkspace
```

4) Now you can import your dependencies:

```
import NoPassSDK
```

Migrate logic

Migrate business logic (one-time password generation, encryption/decryption, store, network logic, etc.) from the NoPass application to the NoPass framework.

Remove 3rd party dependency

To achieve better support SDK, replace the 3rd party libraries with native realization.

`Realm` ➔ `CoreData`

`Alamofire` ➔ `NSURlSession`

etc.

Some libraries should be added like the source code.

SDK settings logic

Add logic to basic settings SDK like `portalURL, apiKEY, apiVersion`, etc.

To configure SDK, call the setup method:

```
NoPassSDK.setSecretKey(AppConfig.API_KEY)
```

2.2 Registration

To start the registration flow, pass the encrypted text from QR or DeepLink to NoPassSDK. You must use the **`NoPassRegistrationService`** for it.

```
public func startRegistration(result: String, enabled2FaMethod:
NoPassSDK.BiometricType, isScreenLock: Bool)
```

Parameters:
result—text from QR or DeepLink
enabled2FaMetho—2Fa method that is used on iPhone
isScreenLock—a bool value that determines if biometry or device passcode is switched on

To handle the registration flow, implement `NoPassRegistrationDelegate`.

```
protocol NoPassRegistrationDelegate {
      // Return registration code (if needed)
      func registration(code:String )
      // Handel result of registration flow
      func finishRegistration(result: String?, error: Error?)
}
```

2.3 Authentication

To start the authentication flow, pass the encrypted text from the push notification to NoPassSDK. You must use **`NoPassAuthService`** for it.

```
public func startAuthFlow(data: [AnyHashable : Any], enabled2FaMethod:
NoPassSDK.BiometricType, isScreenLock: Bool) -> NoPassSDK.NoPassAuthModel?
```

To confirm authentication, call:

```
public func authorize(enabled2FaMethod: NoPassSDK.BiometricType = .null,
isScreenLock: Bool)
```

To decline authentication, call:

```
public func decline(type: NoPassSDK.DeclineType, enabled2FaMethod:
NoPassSDK.BiometricType, isScreenLock: Bool)
```

To handle the authentication flow, implement `NoPassAuthServiceDelegate`:

```
public protocol NoPassAuthServiceDelegate : AnyObject {

    func onAuthDataChange(comparisonContent:
NoPassSDK.NoPassAuthComparisonContent, authExparedDate: Date, nextUpdate:
TimeInterval)

    func onRadiusAuthStart(clientName: String, account: NoPassSDK.NoPassAccount,
authExparedDate: Date)

    func onAuthFinish(error: NoPassSDK.NopassError?, authStatus:
NoPassSDK.AuthStatus)
}
```

Other methods

To check an active authentication session:

```
public func isHaveAuthSessionNow() -> Bool
```

To get authentication comparison content:

```
public func getAuthComparisonContent(data: [String : Any], userSeed: String) ->
NoPassSDK.NoPassAuthComparisonContent?
```

2.4 Push notifications

NoPass uses the Firebase cloud messaging system in its registration and authorization flows.

1) For correct work of the registration/authorization flow, pass data from incoming push notification to SDK:

```
NoPassNotificationService.shared.setRegistrationToken(token: fcmToken)
```

2) To handle the push data, you must call the **`passNotification`** method.

```
public func passNotification(data: [AnyHashable : Any]?, enabled2FaMethod:
NoPassSDK.BiometricType, isScreenLock: Bool)
```

3) To get the NoPass notification, type:

```
extension AppDelegate: MessagingDelegate {
    func messaging(_ messaging: Messaging, didReceiveRegistrationToken
fcmToken: String) {
NoPassNotificationService.shared.setRegistrationToken(token: fcmToken)
    }

    func messaging(_ messaging: Messaging, didReceive remoteMessage:
MessagingRemoteMessage) {
NoPassNotificationService.shared.passNotification(data:
data,
enabled2FaMethod: LocalAuthService.biometricType(),
isScreenLock: LocalAuthService.isSceenLock())
   }
}
```

2.5 Other operations

There are other operations that can be performed with the account service manager. You must use **`NoPassAccountService`**.

1) Get all the users list:

```
NoPassAccountService.shared.fetchAccounts()
```

2) Get the authentication history:

```
NoPassAccountService.shared.fetchHisory()
```

3) You can also delete the user from both backend and the mobile device:

```
public func removeAccount(account: NoPassSDK.NoPassAccount,
enabled2FaMethod: NoPassSDK.BiometricType, isScreenLock: Bool, completion:
((NSError?) -> Void)?)
```

4) To subscribe on accounts changing action, you must call:

```
NoPassAccountService.shared.subscribe()
```

And set up closure:

```
        NoPassAccountService.shared.onAccountsChange = { [weak self] in
            // some logic for updating UI
        }
```

5) Backup. It is also possible to create a backup data string encrypted by a 6-digits pin-code. This data may be used to restore accounts later. To initiate backup, do the following:

```
// encryptedBackupData - existed backup data
public func backupAccounts(pin: String, encryptedBackupData: String?,
enabled2FaMethod: NoPassSDK.BiometricType, isSreenLock: Bool, completion:
((NoPassSDK.NopassError?, String?) -> Void)?)
```

6) Restore. If there are properly backed up accounts, they can be restored by a backup string and the pin code. To begin restoration, run the following command:

```
// restoreDidStart closure returns a count of backup accounts and
NopassError
public func restoreAccounts(backupData: String, pin: String, delegate:
NoPassSDK.RestoreFlowDelegate?, enabled2FaMethod: NoPassSDK.BiometricType,
isScreenLock: Bool, restoreDidStart: ((Int, NoPassSDK.NopassError?) ->
Void)?)
```

7) To remove backup data:

```
public func clearBackupData()
```

8) To check if file can be decoded:

```
    public func isCanDecodeBackupFile(encodedString: String, pin: String) -
> Bool
```

9) To start the synchronization flow, pass the encrypted text from QR to NoPassSDK. You must use **`NoPassSynchronisationService`**

```
public func startSyncAccount(result: String, enabled2FaMethod:
NoPassSDK.BiometricType, isScreenLock: Bool)
```

10) To handle the synchronization flow, implement **`NoPassSynchronisationServiceDelegate`**:

```
public protocol NoPassSynchronisationServiceDelegate : AnyObject {

    func synchronisationDidFinish()

    func syncRegistrationCode(code: String, isNeedConfirmationCode: Bool)

    func accountWasSynchronised(account: NoPassSDK.NoPassAccount?, error:
NoPassSDK.NopassError?)

    func synchronisationDidFail(error: NoPassSDK.NopassError)
}
```