RSA SecurID® SDK 2.5 for iOS Developer's Guide

RSA

Contents

Preface

About This Guide

This guide describes how to use the RSA SecurID SDK for iOS to integrate RSA SecurID one-time password (OTP) features directly into a custom iOS mobile app. The guide is intended for developers who have experience building iOS apps as well as a basic understanding of RSA SecurID two-factor authentication. Do not make this guide available to the general user population.

Product Documentation

For more information about RSA SecurID SDK for iOS, see the following documentation:

Release Notes. Describes what is new and changed in this release, as well as other pertinent information. The latest Release Notes are available at **https://community.rsa.com/community/products/securid/software-token-ios.**

Related Documentation

For more information related to the RSA SecurID or software tokens, see the following:

RSA Authentication Manager 8.x Administrator's Guide. Provides an overview of Authentication Manager and its features. Describes how to configure the system and perform a wide range of administration tasks, including managing users and security policies and provisioning RSA SecurID tokens.

For RSA Authentication Manager documentation on RSA Link, go to: **https://community.rsa.com/community/products/securid**

Security Console Help. Describes day-to-day administration tasks performed in the Security Console (RSA Authentication Manager user interface). To view Help, click the **Help** tab in the Security Console.

RSA SecurID Authentication Engine 3.0.0 for Java Developer's Guide. Provides a detailed description of the Authentication Engine API for Java. Go to **https://community.rsa.com/docs/DOC-104287**.

RSA SecurID Software Token Security Best Practices Guide. Identifies best practices designed to ensure secure operation of RSA SecurID software token applications. To access the *Best Practices Guide*, go to **https://community.rsa.com/docs/DOC-35128**.

RSA SecurID Software Token Deployment Planning Guide. Provides information on planning an enterprise-wide software token deployment, including transitioning from hardware tokens to software tokens, and examples of software token provisioning and delivery on desktop and mobile platforms. To download the *RSA SecurID Software Token Deployment Planning Guide*, go to **https://community.rsa.com/docs/DOC-35127**.

RSA SecurID Software Token Converter 3.1 Administrator's Guide. The Token Converter is a command line utility for converting individual RSASecurID software token files into alternative delivery formats, including custom compressed token format (CTF) URLs and QR Codes. To download the Token Converter, go to **https://community.rsa.com/docs/DOC-62014**.

Support and Service

RSA Link – RSA SecurID Space	**https://community.rsa.com/community/products/securid**
Customer Support	**https://community.rsa.com/community/rsa-customer-support**
RSA Ready Partner Program	**www.rsaready.com**

RSA Link contains a knowledgebase that answers common questions and provides solutions to known problems, product documentation, community discussions, and case management.

The RSA Ready Partner Program website provides information about third-party hardware and software products that have been certified to work with RSA products. The website includes Implementation Guides with step-by-step instructions and other information on how RSA products work with third-party products.

Before You Call Customer Support

Make sure you have access to the device running RSA SecurID SDK for iOS.

Please have the following information available when you call:

- ❑ Information about your development environment, including the version of Xcode and the version of the RSA SecurID SDK for iOS.
- ❑ Your RSA Customer/License ID.
- ❑ Product software version number.
- ❑ The model of the iOS device where the problem occurs.
- ❑ The iOS version running on the device where the problem occurs.

1 Overview and Setup

Intended Audience

This document is for iOS app developers who want to implement a mobile app that generates one-time passwords (OTPs) for RSA SecurID two-factor authentication. Developers must have professional experience using Objective-C and developing for the iOS platform.

This document also assumes a basic understanding of RSA SecurID two-factor authentication. For more information, go to **https://www.rsa.com/en-us/products-services/identity-access-management/securid**.

SDK Features

The RSA SecurID SDK 2.5 for iOS (RSA SecurID SDK) provides Objective-C classes and methods for building a mobile app that uses RSA SecurID software tokens to generate one-time passwords (OTPs). The RSA SecurID SDK also manages a token database on the device to store, retrieve, and update software token records. Tokens must be issued from a supported authentication server such as RSA Authentication Manager or using the RSA SecurID Authentication Engine (SAE) API. For more information, see "Authentication Server Requirements" on page 11.

The RSA SecurID SDK allows developers to:

- Get the unique binding ID to be used for binding a software token to a specific device. A device ID is engineered to allow installation only on a device with a matching ID.
- Import up to 10 software tokens. For an overview of token import methods, see "Software Token Provisioning Overview" on page 17.
- Get the current OTP and the next OTP. The OTP is the code generated by the software token symmetric key (the "seed") and algorithm, or by the generated code combined with a PIN.
- Set a user-friendly nickname for a token.
- Delete a token installed on the device, identified by serial number.
- Get token metadata.
- Get RSA SecurID library information.

For details on the methods used to implement these features, see Chapter 3, "Using the APIs."

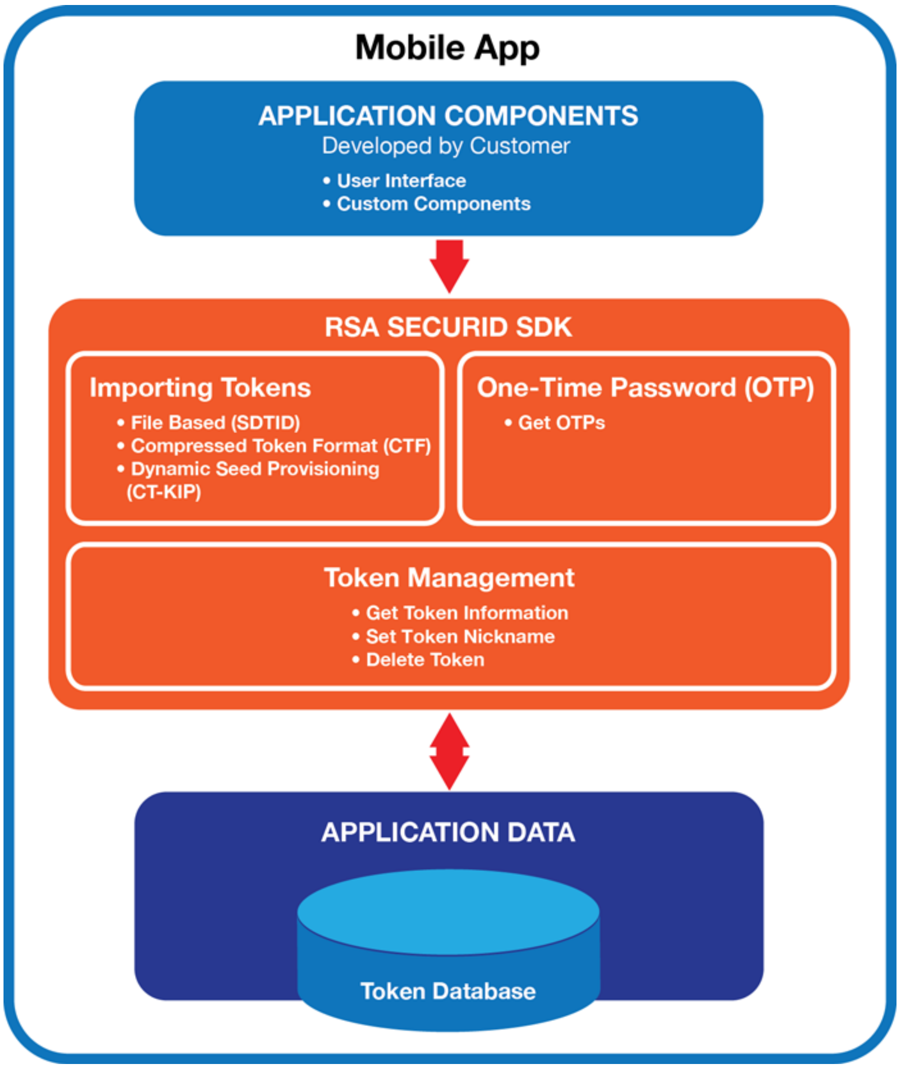
Supported Token Converter Versions

This release supports importing compressed token format (CTF) strings generated using RSA SecurID Software Token Converter 3.1. Additionally, you can import CTF strings generated using RSA Authentication Manager 8.1 Service Pack 1 (SP1) Patch 13 or later with the TLS 1.2 Mode update applied. For more information, see "Compressed Token Format" on page 18 and "Import Token Data Using Compressed Token Format (CTF)" on page 23.

Mobile Application and SDK Components

As shown in the following figure, the RSA SecurID SDK 2.5 for iOS allows you to integrate one-time password (OTP) features directly into your iOS mobile app. The customer (developer) is responsible for designing the app user interface to support OTP generation, as well as providing any custom components, such as communication interfaces. The RSA SecurID SDK provides methods for importing software tokens into the SDK, generating OTPs, and managing tokens.

Mobile application data is stored in the token database on the device. The RSA SecurID SDK automatically copy protects the application data so that it cannot be used on another device.



Changes in This Release

The RSA SecurID SDK 2.5 is now a dynamic framework that uses RSA BSAFE Micro Edition Suite (MES) 4.4. For more information on MES, see **https://rsa.jiveon.com/community/products/bsafe/mes/overview**.

Supported Platforms

RSA SecurID SDK 2.5 supports any iOS device running iOS 11 or later.

Development Environment

Your development environment must include the following:

- Xcode 10.1 or later. (Confirm the latest acceptable version with Apple before submitting to the Apple App Store.)
- RSA SecurID SDK 2.5 for iOS supports the ARM64 architecture for a device and the x86_64 architecture for a simulator.

The minimum iOS deployment version supported for building an application is iOS 11.

Package Contents

RSA SecurID SDK 2.5 for iOS is distributed as a framework bundle, which embeds the public headers. Developers must embed the Software Token SDK framework and BSAFE MES dynamic libraries.

The iOS Token package is provided in **iOS_250_sdk.zip**, which contains the following files.

Filename	Contents
\SecurIDLibFramework\doc	API documentation (automatically generated with Doxygen). Open the **index.html** file to view the API documentation.
\SecurIDLibFramework\Release-iphoneos	Token SDK framework for ARM architectures. Build with this framework when submitting apps to the Apple App Store.
SecurIDLibFramework\Release-merged	Token SDK framework for ARM and simulator architectures. Use for development to test your app with XCode simulator or deploy to the device for testing.
\SampleApp\SampleApp	Sample iOS app. To build and run the sample app, see Chapter 4, "Using the Sample App."

The following RSA SecurID SDK 2.5 for iOS documentation is available at **https://community.rsa.com/community/products/securid/software-token-ios**.

Filename	Contents
RSASecurIDiOSSDK250_devguide.pdf	This document
RSASecurIDiOSSDK250_release_notes.pdf	Release Notes
licenses_iOSSDK_250.pdf	Third-party licenses file

Authentication Server Requirements

The development infrastructure for your mobile app must include an authentication server for provisioning software tokens and providing back-end authentication services. The following table lists the supported authentication servers and the token attributes and provisioning methods supported by each authentication server. For information on token types, see "PIN Handling" on page 13. For information on token attributes and token provisioning, see Chapter 2, "RSA SecurID Software Tokens."

Authentication Server	Supported Token Attributes and Provisioning Methods
RSA SecurID Authentication Engine 3.0.0 for Java (API)	• Token types: PINPad-style, fob-style, and PINless • 60-second and 30-second tokencode interval • 6-digit or 8-digit OTP • Binding ID • Nickname • Token file provisioning (SDTID)
RSA Authentication Manager 8.2 Service Pack 1 (SP1) Patch 7 or later with the TLS 1.2 Mode update applied	• Token types: PINPAD-style, fob-style, and PINless • 60-second and 30-second tokencode interval • 6-digit or 8-digit OTP • Binding ID • Nickname • Token file provisioning (SDTID) • Dynamic seed provisioning (CT-KIP) • CTF provisioning

Migration of Mobile Application Data

The RSA SecurID SDK 2.5 for iOS supports automatic migration of token data from mobile apps that integrate versions 2.4 or higher of RSA SecurID SDK for iOS and execute on devices running iOS 11 and later. If your updated mobile app and users’ iOS devices meet this requirement, the software tokens used with the previous version of your app will work with your new app running on the same device. Tokens installed on one device cannot be migrated to another device.

Important: The SDK stores data in the iOS keychain and in the application **Documents** folder. Security settings for this keychain and application data allow access when the device is unlocked or after the device has been unlocked once after a restart, depending on your data access mode configuration. To ensure that token data is preserved during application upgrades, see Apple guidelines for specification of the AppID and AppID Prefix and their roles in controlling keychain data access.

2 RSA SecurID Software Tokens

About Software Tokens

RSA SecurID SDK for iOS supports RSA SecurID 820 software tokens, which are 128-bit (AES) tokens. When imported into a mobile app, RSA SecurID 820 software tokens generate one-time password (OTP) values for authenticating to environments protected by RSA SecurID technology. To generate OTP values, the RSA SecurID algorithm must be provided with the current time, token metadata, and the token’s symmetric key, or “seed.” The metadata and token seed are held in a “token record.” The authentication server administrator (for example, an RSA Authentication Manager administrator) distributes a token to a user, who imports the token into the mobile app on the user’s device. The token is securely stored in a token database on the device.

In mobile apps developed by RSA, an OTP value is called a “tokencode” when it has not been combined in any way with a user PIN. When combined with a PIN, the OTP value is called a “passcode.”

A mobile app built with the RSA SecurID SDK for iOS can import up to 10 tokens.

PIN Handling

By default, RSA SecurID software tokens require a PIN. RSA delivers software tokens with the token type of PINPad. However, an RSA Authentication Manager administrator or an SAE developer can optionally reconfigure software tokens as fob-style or PINless before exporting a token for delivery to the mobile app. The RSA SecurID SDK for iOS can import any of these token types, but the SDK does not provide methods to change the token type.

Token types determine how the user performs SecurID authentication, as well as how the RSA SecurID SDK handles PINs passed in to the OTP generation method.

- Fob-style use mimics the operation of RSA key fobs, such as the RSA SecurID 700. The user enters the PIN, followed by the current tokencode, in the protected resource.
- For PINPad-style, the user enters the PIN in the mobile app to generate a passcode (OTP), and then enters the OTP in the protected resource.

- PINless use requires entering the current tokencode in the protected resource.

Important: If you use PINless tokens for RSA SecurID authentication, you must also require a second authentication factor, such as a Windows password, to authenticate to protected systems. For complete information about the proper use of PINless tokens, see the *RSA SecurID Software Token Security Best Practices Guide*. To access the *Best Practices Guide*, go to **https://community.rsa.com/docs/DOC-35128**.

If the mobile app passes in a PIN to the OTP generation method, the RSA SecurID SDK uses the token PIN type as input for generating the OTP.

PIN Type	PIN Usage by the SDK
PINPAD	• The PIN is used as input into the SDK. • The tokencode is integrated with the PIN and returned as the OTP (passcode).
KEYFOB	• The PIN is used as input into the SDK. • The tokencode is appended to the PIN and returned as the OTP (passcode).
PINLESS	• The PIN is not used as input into the SDK. • The SDK returns an OTP (tokencode).

PIN Requirements

If you use tokens that require a PIN, you must build appropriate functionality into your mobile app user interface (UI).

- For fob-style or PINless, the UI should display the current OTP (tokencode).
- For PINPad-style, the UI should prompt the user for the PIN. After PIN entry, the UI should display the OTP that has been integrated with the PIN (passcode).

PINs used with software tokens must meet the following requirements.

- The minimum PIN length is 4 characters, and the maximum is 8 characters.
- PINPad-style software tokens require numeric PINs.
- Fob-style software tokens can have either numeric or alphanumeric PINs.
- The PIN cannot contain spaces.

Token Metadata

Token metadata is the information contained in a token record, such as the token serial number and expiration date. In order to perform certain operations, the mobile app must first obtain token metadata. The SecurIDTokenProtocol class defines the interface for metadata associated with a token.

Software Token Metadata

Token Type

The token type refers to whether the token is PINPAD, KEYFOB, or PINLESS. You cannot change the token type through the RSA SecurID SDK. The PIN is accepted as a parameter of the OTP generation methods. The (IPS_TOKEN_TYPE) tokenType method returns one of the following token type enumerators:

- IPS_TOKEN_TYPE_PINLESS. PINless token.
- IPS_TOKEN_TYPE_KEYFOB. Fob-style token.
- IPS_TOKEN_TYPE_PINPAD. PINPad-style token.

OTP Length

The OTP (tokencode) length can be either 8 digits (default) or 6 digits. An 8-digit token can be reconfigured as a 6-digit token, and vice versa. The OTP length can be reconfigured in Authentication Manager 8.1 or later or in SAE. You cannot change the OTP length through the RSA SecurID SDK.

The otpDigits method returns the OTP length, either 6 or 8 digits. The returned OTP length does not include the PIN.

OTP Interval

The OTP interval is the duration of the current OTP (tokencode), either 60 seconds (default) or 30 seconds. A token with a 60-second interval can be reconfigured as a 30-second token, and vice versa. The OTP interval can be reconfigured in Authentication Manager 8.1 or later or in SAE. You cannot use RSA SecurID SDK to change the OTP interval.

The otpInterval method returns the OTP interval.

Nickname

A nickname is a user-friendly name that can be assigned to a token, using the setNickname:nickname method. If there is no nickname, the SDK returns an empty string. The maximum nickname length is 32 characters. To set a token nickname, see "Nickname a Token" on page 33.

The tokenNickname method returns the token nickname.

Serial Number

Each token has a unique serial number. To perform any operation using a token installed in the token database, your code must pass in the serial number for that specific token. The maximum serial number length is 12 digits.

The tokenSerialNumber method returns the token serial number.

Expiration Date

The expiration date is the date when the token can no longer generate OTPs. The RSA SecurID SDK stops generating OTPs when a token expires. Tokens expire on the first second of the token expiration date (00:00:00 GMT).

When you import a token, the RSA SecurID SDK checks the expiration date. If the token has expired, the RSA SecurID SDK does not continue the import process.

The tokenExpirationDate method returns the token expiration date in the number of seconds since January 1, 1970 UTC.

Software Token Configuration

Software tokens provisioned as token files (SDTID files) or compressed token format (CTF) strings should use token file passwords and device binding. For more information on SDTID files and CTF strings, see "File-Based Provisioning (SDTID)" on page 18 and "Compressed Token Format" on page 18.

Software tokens provisioned using dynamic seed provisioning (CT-KIP protocol) use a one-time activation code and should use device binding. For more information on activation codes, see "Dynamic Seed Provisioning" on page 17.

Device Binding

Device binding associates a software token with an authorized iOS device. Binding is engineered to allow installation only on a device or class of devices with a matching device ID.

When the mobile app is installed on a device, the RSA SecurID SDK generates a binding ID for that app (a 24-character hexadecimal string). If more than one mobile app built with the SDK is installed on the same device, the SDK generates a different binding ID for each app.

Device binding must be performed in RSA Authentication Manager 8.1 or later or SAE before a token is exported for delivery to the mobile app. You must provide the binding ID to the Authentication Manager administrator or SAE developer. Use the getBindingId method to get the binding ID, as described in "Get the Binding ID" on page 23.

When a user attempts to import a token bound to a device, the RSA SecurID SDK gets the binding ID from the token and checks whether the data matches the 24-character hexadecimal string returned by getBindingId. If a match is found, and the user correctly enters other required information (for example, a token file password), the token is successfully imported.

Important: Instruct users to treat the binding ID as sensitive information and to use a secure channel to deliver it to the administrator.

Token Passwords

In addition to using device binding, RSA strongly recommends using token passwords with software tokens provisioned using compressed token format (CTF) strings or token files (SDTID files). The Authentication Manager administrator or SAE developer sets the token password. RSA recommends that the CTF NOT be communicated together with the binding ID.

When importing a token into the RSA SecurID SDK, you must pass in the password to one of the following:

- importTokenFromCtf:ctfPassword:serialNumber: For more information, see "Import Token Data Using Compressed Token Format (CTF)" on page 23.
- importTokenFromFile:filePassword:serialNumber: For more information, see "Import a Token from a Token File" on page 26.

Token Security on the Device

RSA SecurID SDK helps ensure token security after a token has been distributed to the intended device. The SDK automatically protects the app data stored in the token database on the iOS device. If the token data is copied to another device, it will not be usable on the second device.

Software Token Provisioning Overview

Software token provisioning is the process of configuring a software token in RSA Authentication Manager or SAE and delivering the token to the mobile app running on the user's device. The following sections provide an overview of the available provisioning options and the methods the mobile app can call to import tokens.

Dynamic Seed Provisioning

Dynamic seed provisioning is a client-server protocol that enables secure, rapid setup of software tokens. A feature of RSA Authentication Manager 8.2 Service Pack 1 Patch 7 or later, dynamic seed provisioning uses the industry-standard Cryptographic Token Key Initialization Protocol (CT-KIP). For increased security, use dynamic seed provisioning if you provision software tokens with RSA Authentication Manager because the CT-KIP process is engineered to prevent the potential interception of the token's seed.

Dynamic seed provisioning eliminates the need for a token distribution file. The Token app running on a mobile device (the client) and Authentication Manager (the CT-KIP server) use a four-pass CT-KIP protocol to exchange information that is used to dynamically generate a unique shared seed. Information critical to seed generation is encrypted during transmission using a public-private key pair. The generated token seed value is never transmitted across the network.

With dynamic seed provisioning, the mobile app must send a one-time provisioning activation code when it initially contacts the CT-KIP server. The Authentication Manager administrator specifies the activation code for a specific token when configuring the token. The CT-KIP server evaluates the activation code to verify that the token is approved for dynamic seed provisioning.

RSA Authentication Manager generates CT-KIP data within a custom CT-KIP URL that can be delivered to the mobile app as a QR code or a hyperlink in an email. You can use this delivery method or devise your own. Your mobile app would then call the importTokenFromCtkip:ctkipAuthCode:validateCert:delegate: method to pass in the underlying CT-KIP data. For more information, see "Import a Token Using CT-KIP" on page 26.

RSA Authentication Manager 8.2 Service Pack 1 Patch 7 or later allows an administrator to deliver CT-KIP token data as a QR Code in the Self Service Console. For instructions, see the *RSA Authentication Manager Administrator’s Guide* on RSA Link: **https://community.rsa.com/community/products/securid**.

The RSA SecurID SDK does not provide a method for importing QR Codes directly. Your mobile app will have to handle the QR Code and then pass the underlying CT-KIP data to the SDK.

To support dynamic seed provisioning (CT-KIP) with iOS 11 or later, you must make sure that the RSA Authentication Manager server meets the App Transport Security (ATS) requirements.

File-Based Provisioning (SDTID)

If RSA Authentication Manager is used for file-based provisioning, token data is generated within an SDTID file (an XML-format file with the extension .sdtid), for example, jjones_000056789012.sdtid. If RSA SecurID Authentication Engine (SAE) is used, the token data is generated as a file stream. The SDTID file can be delivered to the mobile app as an email attachment or by another method that you devise.

The importTokenFromFile:filePassword:serialNumber: method imports a token from an SDTID file. For more information, see “Import a Token from a Token File” on page 26.

RSA strongly recommends protecting SDTID files with a token file password as part of the provisioning process.

To deliver a token, you send an email with an SDTID file attachment to the email client on the device. If you password-protect the file, RSA recommends sending the password separately, using a secure channel and best practices for communicating sensitive data.

Compressed Token Format

Compressed token format (CTF) is an alternative format to SDTID files. CTF-format tokens consist of token data contained within a custom CTF URL. To obtain a custom CTF URL, you use an SDTID file as input to the RSA SecurID Software Token Converter 3.1 (Token Converter) command line utility. Token Converter 3.1 generates custom CTF URLs in an alphanumeric format. For backward compatibility with Token Converter 2.6.1, Token Converter 3.1 provides an option to generate numeric CTF strings.

RSA Authentication Manager 8.1 or later can generate legacy-format custom CTF URLs. To obtain new-format custom CTF URLs, however, the Authentication Manager administrator must provision SDTID files and convert them using Token Converter 3.1. RSA strongly recommends protecting CTF strings with a strong password.

The Token Converter also provides an option to convert an SDTID file into a QR Code. It does this by converting the SDTID file into a custom CTF URL and then embedding the CTF data in a QR Code.

The RSA SecurID SDK does not provide a method for importing QR Codes directly. Your mobile app will have to handle the QR Code and then pass the underlying CTF data to the importTokenFromCtf:ctfPassword:serialNumber: method.

To download the Token Converter and documentation, go to **https://community.rsa.com/community/products/securid/software-token-converter**.

For information on importing CTF data into the RSA SecurID SDK, see “Import Token Data Using Compressed Token Format (CTF)” on page 23.

3 Using the APIs

RSA SecurID SDK APIs

The following tables list the APIs provided in the RSA SecurID SDK 2.5 for iOS. For details, see the API documentation.

The remainder of the chapter describes how to use the APIs.

Class Methods

The RSA SecurID SDK provides the following Class methods.

Class Method	Description
+ getLibraryInfo:minor:patch:	Retrieves the library version information.
+ getBindingId	Retrieves a unique device identifier (binding ID) for device binding
+ isCtfFormat:scheme:	Determines whether the specified string is a compressed token format (CTF) formatted string.
+ isCtkipFormat:scheme:	Determines whether the specified string is a Cryptographic Token Key Initialization Protocol (CT-KIP) formatted string.
+ getInstance	Gets a shared instance of SecurIDLib.

Instance Methods

The RSA SecurID SDK provides the following Instance methods.

Important: When using the Instance methods, you must call the getInstance Class method before any other operations can be performed.

Instance Method	Description
- importTokenFromFile:filePassword:serialNumber:	Imports a token from a file.
- importTokenFromCtf:ctfPassword:serialNumber:	Imports a token using compressed token format (CTF).
- importTokenFromCtkip:ctkipAuthCode: validateCert:delegate:	Imports a token using the Cryptographic Token Key Initialization Protocol (CT-KIP).
- getTokenList:	Retrieves a list of tokens installed on the device.
- getOtp:pin:otp:secs:	Generates an OTP for a token.
- getNextOtp:pin:otp:secs:	Generates the next OTP for a token.
- reset	Removes all installed tokens. The database key is also regenerated.
- deleteToken:	Deletes the token with the specified serial number.
- setNickname:nickname:	Sets the nickname for the token with the specified serial number.
- changeDataAccessMode	Changes the data access mode to one of two values: Token data can only be accessed when device is unlocked or after the device has been unlocked once after a restart.
- getDataAccessMode	Retrieves the current data access mode.

Required App Updates

RSA SecurID SDK 2.5 for iOS is a dynamic framework. If you are not familiar with using frameworks, see the relevant Apple documentation on using frameworks.

Perform these updates in your app:

- Remove references to the previous SDK from the include and library search paths.
- Remove links to the old SDK static library from the linker flags.
- In Xcode for your application target, add the new framework using embedded binaries. The **Release-iphoneos** folder contains the framework that supports ARM64 architecture for devices. Use this for the app when publishing to the Apple App Store. The **Release-merged** folder contains the framework that supports both ARM64 and 64-bit simulator. These architectures are merged as a convenience for development.

- Update your source code to include or import the framework. The public header files from the previous SDK versions are now bundled in the 2.5 framework. Replace any references to importing the old SDK header files with this:

  #import <SecurIDLibFramework/SecurIDLibFramework.h>
- Usage of the SDK remains unchanged.

Sample App

The RSA SecurID SDK 2.5 for iOS includes a complete Xcode project that can be deployed to a development device and can also run in Xcode simulators. The procedures in this chapter reference the sample app files. The sample app is located in the **SampleApp\SampleApp** folder of the RSA SecurID SDK 2.5 for iOS kit, **ios_250_sdk.zip**. For instructions on building and using the sample app, see Chapter 4, "Using the Sample App."

Important: The sample app files should not be used in production environments.

Get the Binding ID

The RSA SecurID SDK 2.5 for iOS generates a binding ID for each installed mobile app that integrates the SDK. The binding ID is used to bind a token to a specific authorized device. The device binding attribute (DeviceSerialNumber) must be set in the authentication server before tokens are distributed to users. You must obtain the binding ID from the installed mobile app and provide it to the RSA Authentication Manager administrator or SAE developer. For an overview of device binding, see "Device Binding" on page 16.

Note: If you upgrade the app, the binding ID is preserved.

Procedure

Call getBindingId to return a 24-character hexadecimal string for device binding.

Example

```
// Get the binding ID for device binding
NSString *deviceBinding = [SecurIDLib getBindingId];
```

Import Token Data Using Compressed Token Format (CTF)

Compressed token format (CTF) strings provide an alternative to using token files to provision software tokens to a device running the mobile app. For an overview of compressed token format, see "Compressed Token Format" on page 18.

This section describes how to import CTF strings into the RSA SecurID SDK. The importTokenFromCtf:ctfPassword:serialNumber: method is used to import CTF strings.

Before You Begin

An administrator can provision CTF strings directly from RSA Authentication Manager 8.1 or later. Administrators can password protect CTF strings during provisioning.

Create a Custom CTF URL

RSA uses custom CTF URLs containing token data as a convenient format for delivering CTF strings to its proprietary iOS app (RSA SecurID Software Token for iOS). Use of custom CTF URLs is optional. You can devise any mechanism for delivering CTF strings that meets your requirements.

If you choose to create custom CTF URLs, you must build support for them into your mobile app. The RSA SecurID SDK expects the following syntax for custom CTF URLs: <your_scheme>://ctf?ctfData=<CTF Data>.

Important: Do not use the scheme com.rsa.securid. This scheme is reserved for use by RSA.

To generate a custom CTF URL using the Token Converter, you specify your custom CTF URL prefix as the argument to the Token Converter -prefix option. For example, the command

```
java -jar TokenConverter.jar c:\tmp\token1.sdtid -ios
-p t0kenpw1 -prefix com.yourco.securid://ctf?ctfData= -o c:\tmp\tokenfile.txt
```

generates your custom CTF URL containing a CTF string:

```
com.yourco.securid://ctf?ctfData=AwAAGKCsXaTYtDJ4iQGFH84mt0nCUCX4fNSuE7bh7mPzjsYYo
KxdpNi0MniJAYUfzia3ScJQJfh81K4TtuHuY%2FOOxk8b9EaDHyuMxtHqOnB39TU1JA8OkTOhh%
2B%2BCqKl16qK%2FS%2BzQ1FVuilNzLCQ9RaJz4Uv1N1CUKae0UNQJakYqH4UQLUnspbzsFQE
H6JXSFUroB4DnuYPidwerG22fOQj6%2FxWiVrz3kh0qs1Mhapkp5o0pb5Wsd0kWYi%2F3S9AlHyc9
u0I%2FCW3NdZPsz2Py832dJOkn9fccvE8%2B2HDBfyh6uDStuIEKXJgNmh5Ji3r3wuVMgGTvGteN
NcBobFRz%2FFFtnKNtc0QLVogEuepDSO5T8uP8XpCvOFgWbUsfsRZ%2BENTt
```

To download the Token Converter and documentation, go to **https://community.rsa.com/docs/DOC-62014**.

You are responsible for parsing the custom CTF URL and passing the CTF string to the importTokenFromCtf:ctfPassword:serialNumber: method.

Check String Format

For customers using custom CTF URLs, the RSA SecurID SDK provides the isCtfFormat:scheme: method, which determines whether the type of string passed into the RSA SecurID SDK is a CTF string. This method checks to see if the scheme matches the scheme passed in to the scheme parameter, the custom CTF URL provided to the tokenUrl parameter is the correct format, and there is CTF data available. Call this method before calling importTokenFromCtf:ctfPassword:serialNumber:.

Example

```
// Check if this a CTF formatted string
BOOL isCtf = [SecurIDLib isCtfFormat:tokenUrl scheme:@"com.yourco.securid"];
```

Import a CTF String

Import Procedure

1. Using a shared instance of SecurIDLib, call importTokenFromCtf:ctfPassword:serialNumber: and pass in the CTF string to the ctfData input parameter. This parameter must not be nil.

   Strip off the custom CTF URL prefix before passing in the CTF string. For example, you would strip off the portion shown in red:

   com.yourco.securid://ctf?ctfData=AwAAGKCsXaTYtDJ4iQGFH84mt0nCUCX4fNSuE7bh7mPz
jsYYoKxdpNi0MniJAYUfzia3ScJQJfh81K4TtuHuY%2FOOxk8b9EaDHyuMxtHqOnB39TU1JA8
OkTOhh%2B%2BCqKl16qK%2FS%2BzQ1FVuilNzLCQ9RaJz4Uv1N1CUKae0UNQJakYqH4U
QLUnspbzsFQEH6JXSFUroB4DnuYPidwerG22fOQj6%2FxWiVrz3kh0qs1Mhapkp5o0pb5Wsd0k
WYi%2F3S9AlHyc9u0I%2FCW3NdZPsz2Py832dJOkn9fccvE8%2B2HDBfyh6uDStuIEKXJgNm
h5Ji3r3wuVMgGTvGteNNcBobFRz%2FFFtnKNtc0QLVogEuepDSO5T8uP8XpCvOFgWbUsfsRZ
%2BENTt

2. If the token file is password protected, pass in the password to the ctfPassword input parameter. The maximum token file password length is 24 characters. The input parameter must be convertible to a UTF-8 encoded string. You can either pass in nil or an empty string if there is no password.

3. Create and pass in an NSMutableString object to the serialNumber output parameter to receive the token serial number. This parameter must not be nil.

   The SDK populates the NSMutableString object with the token serial number if the token import succeeds or if the SDK encounters a duplicate token error.

   For specific error codes that may be returned if the token is not successfully imported, see the API documentation.

Example

```
// import a password-protected token using a CTF string
SECURIDLIB_STATUS status =
        [sharedSecurIDInstance importTokenFromCtf:ctfData
                                              ctfPassword:ctfPassword
                                            serialNumber:tokenSerialNumber];
```

Delivery of a Custom CTF URL

If you use custom CTF URLs, it is the responsibility of your mobile app and server application to determine how to deliver the custom CTF URL to the user's device. A typical method is to send an email containing a CTF URL hyperlink to the device's email client.

Import a Token from a Token File

The importTokenFromFile:filePassword:serialNumber: method imports a software token from an SDTID file.

To help ensure the security of the SDTID file in transit, the RSA Authentication Manager administrator should password protect the file when provisioning the token.

Import Procedure

1. Using a shared instance of SecurIDLib, call importTokenFromFile:filePassword:serialNumber: and pass in an NSData object containing the token file data. This parameter must not be nil.
2. If the token file is password protected, pass in the password as a UTF-8 encoded string to the filePassword parameter. The maximum token file password length is 24 characters. You can either pass in nil or an empty string if there is no password.
3. Create and pass in an NSMutableString object to the serialNumber output parameter to receive the token serial number. This parameter must not be nil.

   The SDK populates the NSMutableString object with the token serial number if the token import succeeds or if the SDK encounters a duplicate token error.

   For specific error codes that may be returned if the token is not successfully imported, see the API documentation.

Example

```
// import token from SDTID file data
SECURIDLIB_STATUS status =
      [sharedSecurIDInstance importTokenFromFile:fileData
                                               filePassword:filePassword
                                               serialNumber:tokenSerialNumber];
```

Delivery of an SDTID File

It is the responsibility of the mobile app and the server application to determine how to deliver the SDTID file to the iOS device. A typical method is to send an SDTID file attachment to the device's email client.

Import a Token Using CT-KIP

The Cryptographic Token Key Initialization Protocol (CT-KIP) is used to dynamically provision software tokens to a device running the mobile app. For an overview of Dynamic Seed Provisioning, see "Dynamic Seed Provisioning" on page 17.

This section describes the steps for importing CT-KIP strings into the RSA SecurID SDK. The importTokenFromCtkip:ctkipAuthCode:validateCert:delegate:method is used to import CT-KIP data. The mobile app must specify the delegate to be notified when the import completes. The delegate is notified of all import attempts, whether successful or unsuccessful.

Create a Custom CT-KIP URL

RSA uses custom CT-KIP URLs as a convenient format for delivering CT-KIP data to its proprietary iOS app (RSA SecurID Software Token for iOS). Use of custom CT-KIP URLs is optional. You can devise any mechanism for delivering CT-KIP data that meets your requirements.

If you choose to create custom CT-KIP URLs, you must build support for them into your mobile app. The RSA SecurID SDK expects the following syntax for custom CTF URLs: <your_scheme>://ctkip?url=<CT-KIP Data>. The CT-KIP data includes the CT-KIP server URL and, optionally, the one-time activation code. For example:

```
com.yourco.securid://ctkip?url=https://ctk-server123.yourco.com/ctkip/services/CtkipService
&activationCode=123456789012
```

Important: Do not use the scheme com.rsa.securid. This scheme is reserved for use by RSA.

Important: For increased security, RSA strongly recommends delivering the activation code using a separate, secure channel. If you deliver the activation code separately, leave out the &activationCode= syntax.

You are responsible for parsing the URL and passing the CT-KIP data and the activation code to the importTokenFromCtkip:ctkipAuthCode:validateCert:delegate: method.

Check That Custom CT-KIP URL Data Is Available

For customers using custom CT-KIP URLs, the RSA SecurID SDK provides the isCtkipFormat:scheme: method, which determines whether the type of string passed into the RSA SecurID SDK is a CTF string. This method checks to see if the scheme matches the scheme passed in to the scheme parameter, the custom CT-KIP URL provided to the tokenUrl parameter is the correct format, and there is CT-KIP data available. Call this method before calling importTokenFromCtkip:ctkipAuthCode:validateCert:delegate:.

Example

```
// Check if this a CT-KIP formatted string
BOOL isCtkip = [SecurIDLib isCtkipFormat:tokenUrl scheme:@"com.yourco.securid"];
```

Import a CT-KIP String

Import Procedure

1. Using a shared instance of SecurIDLib, call importTokenFromCtkip:ctkipAuthCode:validateCert:delegate:.
2. Pass in the URL of the CT-KIP server to the ctkipUrl input parameter, and pass in the one-time activation code (authorization code) for the token to the ctkipAuthCode input parameter to establish an HTTPS connection between the mobile app and the CT-KIP server.

Note: The SDK restricts the length of the URL string to 1024 characters. The SDK restricts the activation code to a maximum of 40 alphanumeric characters.

3. If you are sure your CT-KIP server certificate is trusted, pass in YES to enable SSL certificate validation. If the certificate validation fails, SECURIDLIB_ERR_CTKIP_SSL_CERTIFICATE is returned. In that case, have your mobile app prompt the user with a message that the certificate is not trusted and provide the option to accept or reject the certificate. If the user accepts the certificate, your mobile app can retry the CT-KIP operation, passing in NO for the validation.
4. Create and pass in an NSObject to the delegateToNotify parameter to receive notification of success or failure upon completion of the CT-KIP operation.

   The NSObject conforms to SecurIDLibDelegate protocol and must not be nil.

   For specific error codes that may be returned, see the API documentation.

Example

Header File:

```
#import <Foundation/Foundation.h>
#import <SecurIDLibFramework/SecurIDLibFramework.h>

// implement SecurIDLibDelegate protocol methods
@interface ImportCtkipClass : NSObject <SecurIDLibDelegate>
{
   SecurIDLib* sharedSecurIDInstance;
}

// initiate import of token using CT-KIP
-(void)initiateImport:(NSString*)ctkipServerUrl
              ac:(NSString*)authorizationCode;

@end
```

Implementation File:

```
#import "ImportCtkipClass.h"

@implementation ImportCtkipClass

- (id) init {
    if ((self = [super init]))
    {
        // perform initialization
        sharedSecurIDInstance = [SecurIDLib getInstance];
    }
    return self;
}

-(void)initiateImport:(NSString*)ctkipUrl
                   ac:(NSString*)authorizationCode
{
        // initiate import via CT-KIP
    SECURIDLIB_STATUS status =
        [sharedSecurIDInstance importTokenFromCtkip:ctkipUrl
                                           ctkipAuthCode:authorizationCode
                              validateCert:YES
                                  delegate:self];

    // handle status
}

#pragma mark - SecurIDLibDelegate Protocol

// CT-KIP import completed
- (void)ctkipImportFinished:(NSString*)serialNumber
{
    // token successfully imported
    // serialNumber identifies the token that was imported

    // handle import success
}

// CT-KIP import failed
- (void)ctkipImportFailed:(NSError*)error
{
    // handle import failure
}

@end
```

Delivery of a Custom CT-KIP URL

If you use custom CT-KIP URLs, it is the responsibility of your mobile app and your server application to determine how to deliver your custom CT-KIP URL and the one-time activation code to the user's device. A typical method is to send an email containing a CT-KIP URL hyperlink to the device's email client.

Get Token Metadata

The getTokenList: method returns an array of object IDs. Objects in the array contain token metadata and conform to SecurIDTokenProtocol protocol. You can use the protocol methods to access the metadata, as shown in the following table.

Protocol Methods	Metadata Returned
(NSString*)tokenSerialNumber	Token serial number
(NSString*)tokenNickname	Token nickname
(NSInteger)otpInterval	OTP (tokencode) interval (30 or 60 seconds)
(NSInteger)otpDigits	OTP (tokencode) length (6 or 8 digits)
(IPS_TOKEN_TYPE)tokenType	One of the token types defined in the IPS_TOKEN_TYPE enumerations
(NSUInteger)tokenExpirationDate	Token expiration date (seconds since midnight Jan 1, 1970 UTC)

Procedure

Using a shared instance of SecurIDLib, call getTokenList:. The RSA SecurID SDK returns an array object ID. The objects in the array conform to the SecurIDTokenProtocol, which defines the interface for metadata associated with a token.

Example

```
//   Get list of token metadata from the token database

SecurIDLib* securid = [SecurIDLib getInstance];
NSMutableArray* tokenInfoList = [[NSMutableArray alloc] init];

SECURIDLIB_STATUS status = [securid getTokenList:tokenInfoList];
if ((status == SECURIDLIB_ERR_SUCCESS) && ([tokenInfoList count] > 0))
{

        for (id<SecurIDTokenProtocol> tokenInfo in tokenInfoList)
        {
              // token serial number
              NSString* serialNumber = [tokenInfo tokenSerialNumber];
              // token nickname
              NSString* nickName = [tokenInfo tokenNickname];
              // token interval
              NSInteger interval = [tokenInfo otpInterval];
              // token digits
              NSInteger digits = [tokenInfo otpDigits];
              // token type
              IPS_TOKEN_TYPE tokenType = [tokenInfo tokenType];
              // token expiration
              NSUInteger expiration = [tokenInfo tokenExpirationDate];
        }

}
```

Get a One-Time Password (OTP)

The getOtp:pin:otp:secs: method gets an OTP object for the current OTP (tokencode) interval.

Important: The getOtp:pin:otp:secs: method accesses the token database and decrypts the token seed. For best performance, RSA recommends calling this method only once per token interval. For example, for a 60-second token, call this method only when the OTP (tokencode) changes.

Procedure

1. Using a shared instance of SecurIDLib, call getOtp:pin:otp:secs: to generate the OTP for a token in the database.
2. Pass in an NSString object containing the serial number for the token for which you want to generate the OTP to the first parameter of this method. This parameter must not be nil.
3. If the token requires a PIN, pass in the PIN string to the pin parameter. This parameter is optional. For information on how the RSA SecurID SDK handles the PIN, see "PIN Handling" on page 13.

4. Create and pass in an NSMutableString object for the otp parameter to receive the current OTP.

   The SDK populates the NSMutableString object with the current OTP if the operation is successful. This parameter must not be nil.

5. Create and pass in the address of a NSUInteger object to the secs output parameter to receive the number of seconds remaining before the current OTP changes. This parameter must not be nil.

   For specific error codes that may be returned if the OTP operation fails, see the API documentation.

Example

```
// get OTP
SECURIDLIB_STATUS status =
    [sharedSecurIDInstance getOtp:tokenSerialNumber
                                 pin:userSecurIDPin
                                 otp:otp
                                secs:&secondsRemaining];
```

Get the Next One-Time Password (OTP)

The getNextOtp:pin:otp:secs: method gets an OTP object for the next OTP (tokencode) interval.

Important: The getNextOtp:pin:otp:secs: method accesses the token database and decrypts the token seed. For best performance, RSA recommends calling this method only once per token interval. For example, for a 60-second token, call this method only when the OTP (tokencode) changes.

Procedure

1. Using a shared instance of SecurIDLib, call getNextOtp:pin:otp:secs: to generate the next OTP for a token in the database.
2. Pass in an NSString object containing the serial number for the token for which you want to generate the next OTP to the first parameter of this method. This parameter must not be nil.
3. If the token requires a PIN, pass in the PIN string to the pin parameter. This parameter is optional. For information on how the RSA SecurID SDK handles the PIN, see "PIN Handling" on page 13.
4. Create and pass in an NSMutableString object for the otp parameter to receive the next OTP.

   The SDK populates the NSMutableString object with the next OTP if the operation is successful. This parameter must not be nil.

5. Create and pass in the address of a NSUInteger object to the secs output parameter to receive the number of seconds remaining before the next OTP changes. This parameter must not be nil.

   For specific error codes that may be returned if the next OTP operation fails, see the API documentation.

Example

```
// get next OTP
SECURIDLIB_STATUS status =
      [sharedSecurIDInstance getNextOtp:tokenSerialNumber
                                        pin:userSecurIDPin
                                        otp:otp
                                       secs:&secondsRemaining];
```

Manage the Token Database

You can perform the following actions on tokens stored in the token database on the user's device:

- Nickname a token
- Delete a token
- Remove all installed tokens

Nickname a Token

The setNickname:nickname: method sets a nickname for the token with the specified serial number.

Procedure

Using a shared instance of SecurIDLib, call setNickname:nickname:. Pass in the serial number string of the token to the serialNumber parameter, and pass in a nickname to the nickname parameter. The serialNumber and nickname parameters must not be nil.

Example

```
// setNickname
SECURIDLIB_STATUS status =
      [sharedSecurIDInstance setNickname:tokenSerialNumber nickname:nickname];
```

Delete a Token

The deleteToken: method deletes the token with the specified serial number from the token database.

Procedure

Using a shared instance of SecurIDLib, call deleteToken: and pass in the serial number string of the token to be deleted to the serialNumber parameter. This parameter must not be nil.

Example.

```
// delete token
SECURIDLIB_STATUS status =
      [sharedSecurIDInstance deleteToken:tokenSerialNumber];
```

Remove All Installed Tokens

Under certain circumstances, you may need to remove all installed tokens. For example, if the RSA SecurID SDK cannot decrypt your token data due to a copy-protection error, you cannot make use of the SecurIDLib object. The reset method removes all installed tokens, restores the SecurIDLib object to a functional state, and regenerates the key for encryption and decryption.

Procedure

Using a shared instance of SecurIDLib, call reset.

Example

```
// reset
SECURIDLIB_STATUS status =[sharedSecurIDInstance reset];
```

Get Library Information

The getLibraryInfo:minor:patch: method returns the major, minor, and patch version numbers of the RSA SecurID SDK, for example, 2.5.0.

Procedure

Call getLibraryInfo:minor:patch:

Example

```
// get library info
int majorVersion = 2;
int minorVersion = 5;
int patchVersion = 0;
[SecurIDLib getLibraryInfo:&majorVersion
                    minor:&minorVersion
                    patch:&patchVersion;
```

Manage Data Access Mode

You can perform the following actions to the data access mode:

- Change the data access mode setting.
- Get the data access mode setting.

Change the Data Access Mode Setting

The changeDataAccessMode method changes the token data access mode to one of two values:

- SECURIDLIB_DATA_ACCESS_ONLY_WHEN_DEVICE_UNLOCKED
- SECURIDLIB_DATA_ACCESS_AFTER_FIRST_DEVICE_UNLOCK

Procedure

Do one of the following:

- To access the token data only when device is unlocked, using a shared instance of SecurIDLib, call changeDataAccessMode with parameter SECURIDLIB_DATA_ACCESS_ONLY_WHEN_DEVICE_UNLOCKED
- To access the token data after the device has been unlocked once after a restart, using a shared instance of SecurIDLib, call changeDataAccessMode with parameter SECURIDLIB_DATA_ACCESS_AFTER_FIRST_DEVICE_UNLOCK

Examples

```
// changeDataAccessMode
SECURIDLIB_STATUS status =
      [sharedSecurIDInstance
changeDataAccessMode:SECURIDLIB_DATA_ACCESS_ONLY_WHEN_DEVICE_UNLOCKED];
```

```
// changeDataAccessMode
SECURIDLIB_STATUS status =
      [sharedSecurIDInstance
changeDataAccessMode:SECURIDLIB_DATA_ACCESS_AFTER_FIRST_DEVICE_UNLOCK];
```

Get the Data Access Mode Setting

The getDataAccessMode method retrieves the current data access mode.

Procedure

Using a shared instance of SecurIDLib, call getDataAccessMode

Example

```
// getDataAccessMode
SECURIDLIB_DATA_ACCESS_MODE mode =
      [sharedSecurIDInstance getDataAccessMode];
```

4 Using the Sample App

About the Sample App

The RSA SecurID SDK for iOS includes a complete Xcode project. Once you add the SDK 2.5 framework, you can immediately deploy the sample app to a development device as long as you have a development certificate and a provisioning profile. The sample app contains sample software token files and CTF strings.

You can use the sample app on a development device to:

- Get the SDK version and token binding ID
- Import SecurID 820 software tokens
- Generate one-time passwords (OTPs) (supports PIN entry)
- View token information
- Delete tokens

Integrating the Framework

If you are using an existing project and not the sample project provided by RSA, you must integrate the framework into your project. Make sure you do the following:

- For app target, add the framework to "Embedded Binaries."
- In the source code, import the framework's master header file.

The framework in **Release-merged** folder is for development use. It supports device and simulator architectures to allow your app to be deployed to devices and run in the Xcode simulator.

The framework in **Release-iphoneos** supports device only. Your app built with this framework can be deployed to device. Use this version for building your app for submission to the Apple App Store.

For more information, see **https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPFrameworks/Tasks/IncludingFrameworks.html**.

Secure Coding Practice

RSA strongly recommends that you strip all debug symbols when you build your final mobile app to help ensure the security of your code. For specific information, see the Apple developer secure coding documentation.

For the Apple *Secure Coding Guide* in HTML format, see **https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html**.

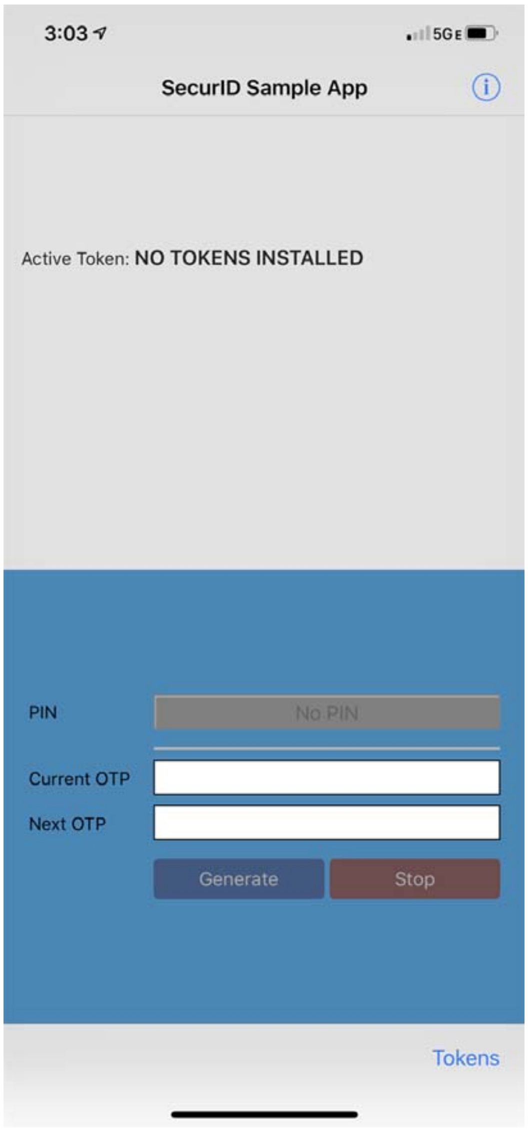
Using the Sample App

This section briefly describes how to use the sample app. The illustrations show how the sample app appears on an iOS 12.2 device.

Start the Sample App

Start the sample app on a device or simulator. To start the sample app on a device, tap the SampleApp icon.

The Active Token scroll view displays “Active Token: NO TOKENS INSTALLED.”

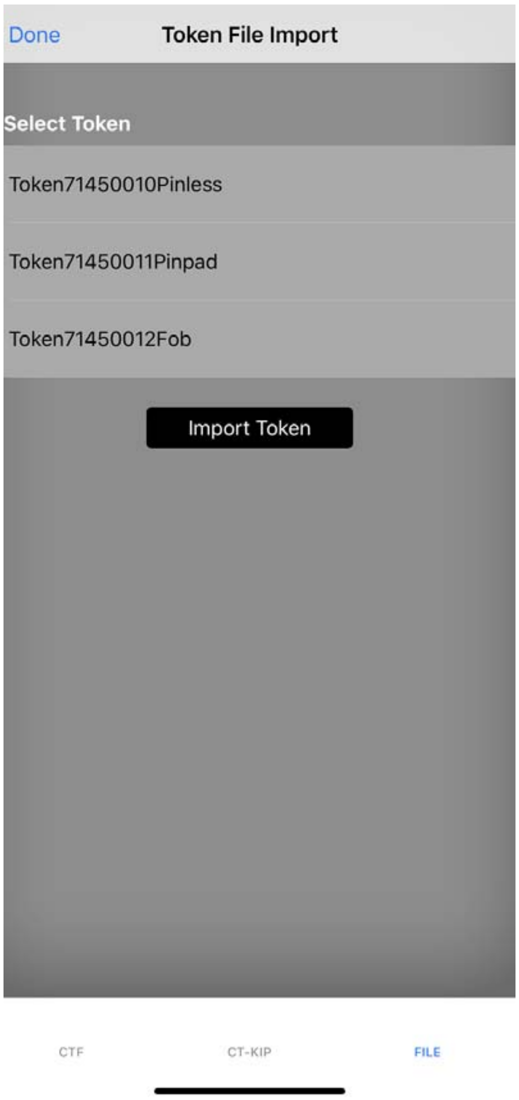


Import a Token from a File

Important: If you use the file import method, you can only import the software tokens provided in the sample app. You can import a PINless, PINPad, and fob-style token.

Procedure

1. Tap **Tokens**.
2. On the tab bar, tap **FILE**.
3. Select the token you want.
   The token field is highlighted.
4. Tap **Import Token** and then **OK**.



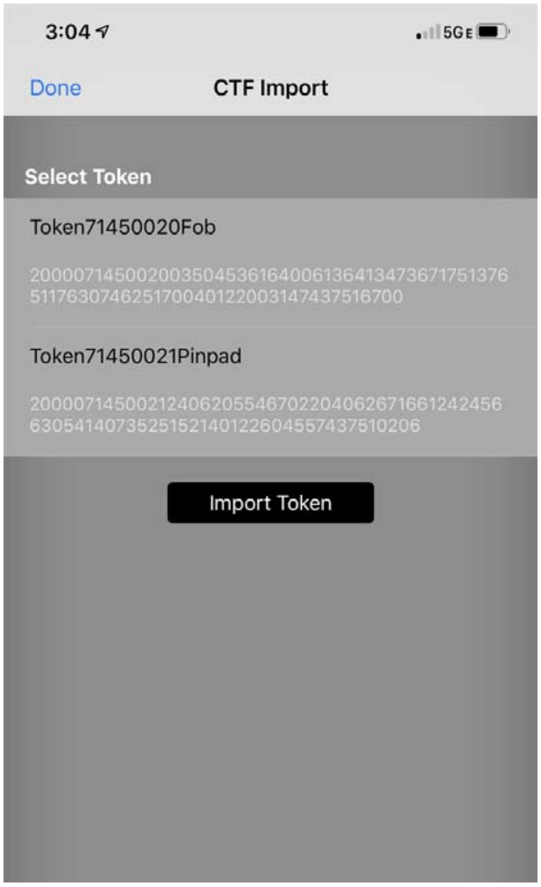
5. Select the next token you want to import, and follow the same process. When finished, tap **Done**.

Import a Token Using CTF

Important: If you use the CTF import method, you can only import the CTF software tokens provided in the sample app. You can use CTF to import a PINless token and a PINPad token.

Procedure

1. Tap **Tokens**.
2. On the tab bar, tap **CTF**.
3. Select the CTF data file you want to import.



Token71450020Fob

200007145002003504536164006136413473671751376
511763074625170040122003147437516700

Token71450021Pinpad

200007145002124062055467022040626716612424566
30541407352515214012260455743751020 6

The CTF string is displayed.

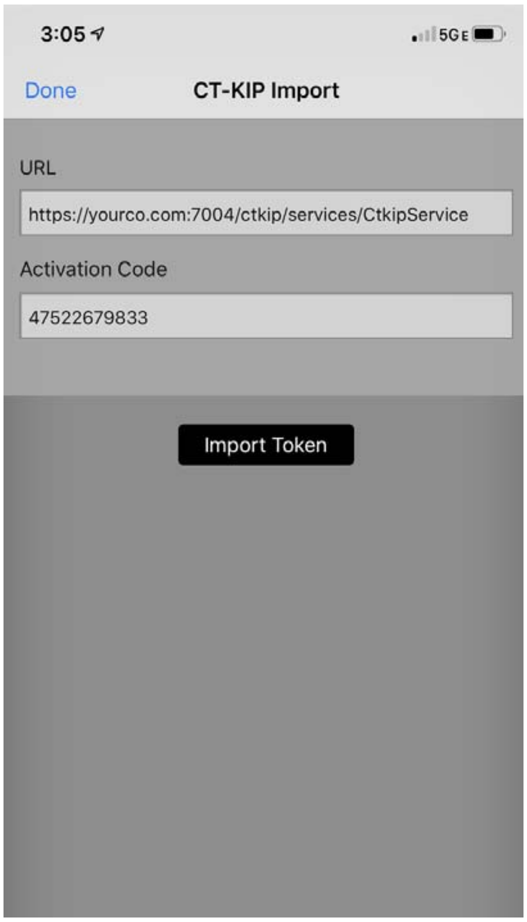
4. Tap **Import Token** and then **OK**.
5. Select the next CTF data file you want to import, and follow the same process. When finished, tap **Done**.

Import a Token Using CT-KIP

Important: If you use the CT-KIP import method, you can only import your own CT-KIP software tokens. You must have devised a means of delivering the tokens to your iOS device or simulator. Make sure you have the URL of the CT-KIP server and the activation code.

Procedure

1. Tap **Tokens**.
2. On the tab bar, tap **CT-KIP**.
3. In the **URL** field, enter the URL of your CT-KIP server. In the **Activation Code** field, enter the one-time activation code. Tap **return** to close the keyboard.



URL
https://yourco.com:7004/ctkip/services/CtkipService
Activation Code
47522679833

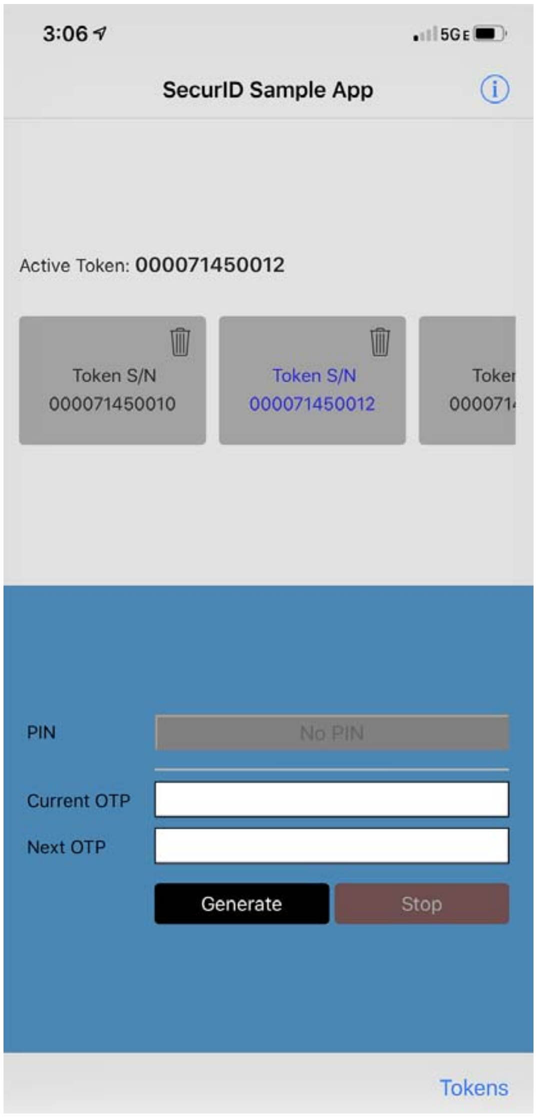
4. Tap **Import Token**.
   The SDK attempts to communicate with the CT-KIP server. If the communication is successful, the token is imported and a success message is displayed. If the token cannot be imported, an error message is displayed. Tap **OK**.
5. Repeat steps 1–5 to import additional tokens from CT-KIP. When finished, tap **Done**.

Select a Different Token

If you imported several tokens, you can change the active token.

Procedure

1. In the Active Token scroll view, swipe horizontally to view the tokens you have imported.
2. To change the active token, tap the token you want to use.
   The serial number in the Active Token scroll view changes to the serial number of the selected token, indicating that the selected token is active.



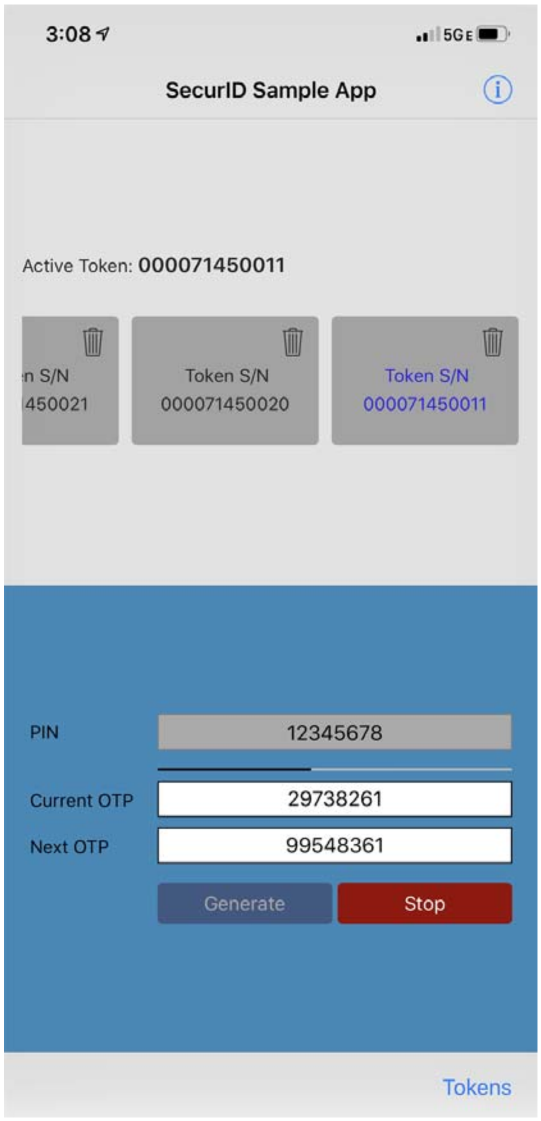
Generate OTP and Next OTP

You generate the current OTP and Next OTP from the Active Token scroll view. If the active token requires a PIN, the **PIN** field is enabled and displays **Enter PIN**. If the active token does not require a PIN, the **PIN** field is disabled and displays **No PIN**.

Procedure

1. Do one of the following:
    - If a PIN is required, tap the **PIN** field to open the keyboard, and enter the PIN. Tap **return** to close the keyboard.
    - If no PIN is required, go to step 2.

2. Tap **Generate**.



The current OTP and next OTP are displayed. The OTP changes at 60-second or 30-second intervals, depending on the tokencode interval of the active token. The countdown box displays the seconds remaining before the code changes.

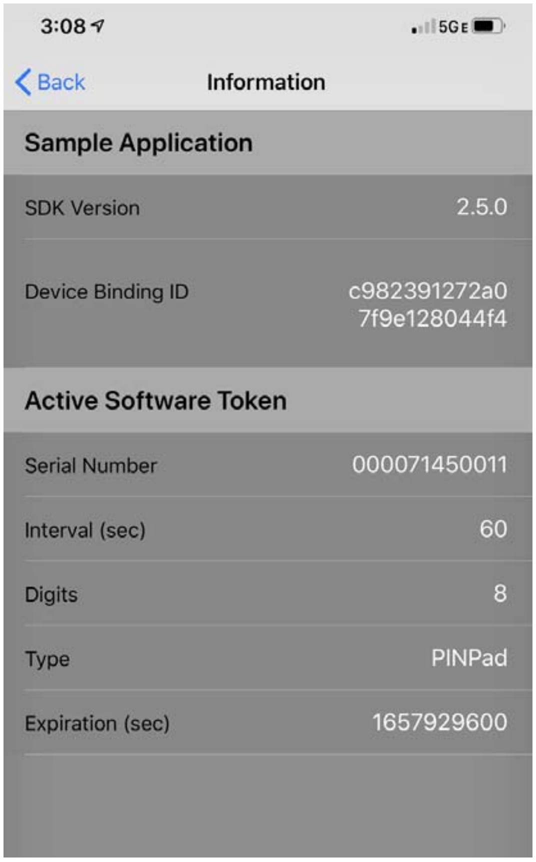
3. To stop generating OTPs, tap **STOP**.

View App Information and Token Information

Procedure

To display information about the sample application and the active token, tap ⓘ in the Active Token scroll view.

If no tokens are installed, only the sample application information is displayed.



3:08

Back Information

Sample Application

SDK Version	2.5.0
Device Binding ID	c982391272a0 7f9e128044f4

Active Software Token

Serial Number	000071450011
Interval (sec)	60
Digits	8
Type	PINPad
Expiration (sec)	1657929600

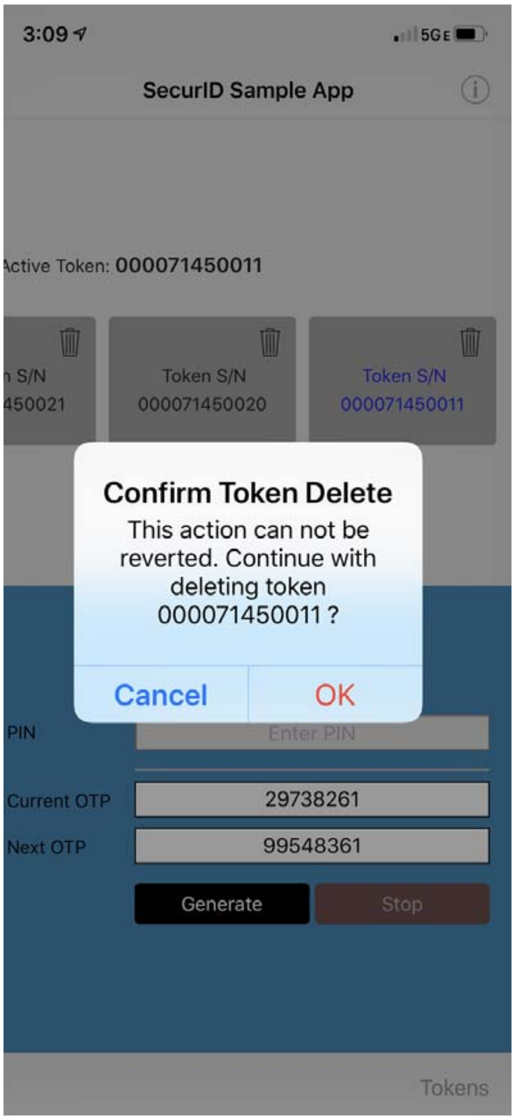
Delete a Token

Procedure

1. To delete any token, in the Active Token scroll view, tap the trash can on the token image.

   You are prompted to confirm the deletion.

2. Tap **OK** to delete the token, or **Cancel** to retain the token.



Confirm Token Delete

This action can not be reverted. Continue with deleting token 000071450011 ?

Cancel OK

Current OTP 29738261

Next OTP 99548361

Glossary

authentication

Reliably verifying the identity of a user or process.

binding ID

A 24-character hexadecimal string that is engineered for binding a software token to a specific mobile device. The RSA SecurID SDK generates a binding ID for the mobile app when the app is installed on an iOS device.

CTF

A human-readable string containing software token data. In most cases you obtain a CTF string by using the RSA SecurID Software Token Converter command line utility.

CT-KIP

Cryptographic Token Key Initialization Protocol, the four-pass protocol used in Dynamic Seed Provisioning. See also "dynamic seed provisioning."

device binding

Associating a software token with an app-specific identifier (binding ID), a value that is unique to one device. Binding a software token to a binding ID helps ensure that the token can be installed only on the device for which it is intended. See also "binding ID."

dynamic seed provisioning

A method for provisioning tokens without the use of token files. The mobile app and the authentication server (for example, RSA Authentication Manager), use a four-pass CT-KIP protocol to exchange information that is used to dynamically establish a shared seed on both sides.

file-based provisioning

Provisioning of a software token within a file. By convention, software token files have the extension **.sdtid**.

fob-style software token

A software token that functions like an RSA hardware key fob. The user enters the PIN, followed by the current tokencode, in the protected resource.

metadata

Information about the contents of a token record, such as the token serial number and token expiration date.

mobile app

An iOS app created using the RSA SecurID SDK. When provisioned with a software token, the mobile app generates OTPs. The mobile app can be part of a distributed app or a standalone app.

nickname

A user-friendly name assigned to a software token. Helps users who have more than one token identify their tokens.

one-time password (OTP)

A password that is valid for only one login session. In RSA SecurID authentication, the OTP is the code generated by the token seed and algorithm (tokencode) or the generated code coupled with a PIN (passcode).

passcode

An OTP value composed of an RSA SecurID PIN combined with a tokencode.

PIN

A user's secret personal identification number. One of the factors in RSA SecurID two-factor authentication.

PIN-enabled token

A software token that requires entering a PIN for OTP authentication.

PINless token

A software token that does not require entering a PIN as part of OTP authentication.

PINPad-style software token

A software token that functions like a hardware PINPad. The PIN is algorithmically combined with the tokencode to create the OTP.

SAE

RSA SecurID Authentication Engine. A library of code supporting one-time password (OTP) operations and token management. The customer integrates SAE into their server.

seed

A cryptographic key used in the generation of a one-time password (OTP). It represents the shared secret between SAE and the server. Each RSA SecurID token has a unique seed.

server

A server created by an RSA customer that integrates SAE functionality, including verification of OTPs and token management.

software token

A software-based security token that generates OTP values at short, regular intervals (every 60 seconds or 30 seconds).

SDTID file

An RSA SecurID software token file. The file has the extension **.sdtid**.

tokencode

An OTP value that has not been combined in any way with a user PIN.

token database

The database in the mobile app where token data is stored.

token file

An SDTID file. The file extension is **.sdtid**.

token file password

A password assigned to a token file to help prevent the token from being used if the file is intercepted in transit by an unauthorized user.

token record	A record stored in the token database that contains the token seed plus metadata.
two-factor authentication	A form of strong authentication using something you know, such as an RSA SecurID PIN, plus something you have, such as an RSA SecurID software token.