Ezio Mobile SDK V4.9

Security Guidelines

gemalto
security to be free

January 9, 2019

Contents

Preface

About This Guide

This document provides several security best practices when securing your applications that based on Ezio Mobile SDK. The security rules are classified into three sections:

- General rules that are platform independent, itemized as GENxx.
- Specific rules for the Android platform, itemized as ANDxx.
- Specific rules for the iOS platform, itemized as IOSxx.

Who Should Read this Guide

The document is intended for application developers who are developing on and integrating applications with Ezio Mobile SDK. This document uses the term "developer" and "application developer" to refer to the people implementing or customizing Ezio Mobile SDK. "User" refers to the end-user of the Ezio Mobile SDK solution on their mobile devices.

For More Information

The following table contains the complete list of documents for Ezio Mobile SDK V4.9.

Document	Description
Ezio Mobile SDK V4.9 Overview	This serves as an introduction to the Ezio Mobile SDK product. It includes information on the package contents, an overview of the system, and a description of the features and services provided by the SDK.
Ezio Mobile SDK V4.9 Release Notes	This contains detailed release information such as new features, issues fixed, known issues, devices and OS versions tested.
Ezio Mobile SDK V4.9 Migration Guides	This set of documents contains the necessary steps when migrating from an earlier versions to *Ezio Mobile SDK V4.9*.
Ezio Mobile SDK V4.9 Security Guidelines	This contains best and recommended security practices that an application developer should follow.
Ezio Mobile SDK V4.9 API Documentation	This document details the interfaces provided by the Ezio Mobile SDK libraries on Android and iOS platforms
Ezio Mobile SDK V4.9 Programmer's Guide	This guide details the usage of the Ezio Mobile SDK when extending the features, functionalities, and interfaces of Ezio Mobile solution.

You may also refer to the following list of links and documents:

- *Chip Authentication Program - Functional Architecture 2007*(MasterCard)
- Android PRNG Fix
- HOTP RFC

- TOTP RFC
- OCRA RFC
- *Gemalto Dynamic Signature (DS) Specification 1.8*
- *Gemalto Generic SWYS Specification 2.2*
- Android, Invalid Key Exception

Contact Us

For contractual customers, further help is provided in the Gemalto Self Support portal at http://support.gemalto.com or you can contact your Gemalto representative.
Gemalto makes every effort to prevent errors in its documentation. However, if you discover any errors or inaccuracies in this document, please inform your Gemalto representative.

About Ezio Mobile SDK

Ezio Mobile is an enterprise authentication solution for e-banking and e-commerce functions such as one-time password (OTP) management, challenge-responses, transaction data signing, data protection, and out-of-band (OOB) communication. It utilizes the user's mobile device as the security platform. The solution consists of mobile applications based on the Ezio Mobile SDK (software development kit).
For the detailed product overview, refer to *Ezio Mobile SDK V4.9 Overview*.

Security Best Practices

General Rules

GEN01. Code Obfuscation

The application and all its components (Ezio Mobile SDK included) must be obfuscated. This increases the complexity of reverse engineering the application and working out the algorithms or identifying its secret data. The obfuscation process of the application must include the Ezio Mobile SDK's library because its public API is not obfuscated.

Warning:

Ezio Mobile SDK requires specific obfuscation settings to be turned on in order to prevent unwanted side-effects at runtime. For a complete list, see *Ezio Mobile SDK Programmer's Guide*.

If the obfuscation tool (or any other tool) offers string encryption, it is recommended to use this feature in addition to code obfuscation. It effectively obfuscates plain text strings in the code so that they cannot easily be extracted and analyzed. Furthermore, this increases the complexity of analyzing the application code for potential weak points based on the use of these strings. Notice that this rule applies only to languages that compile to an intermediate byte code such as Java and C#. For languages that compile to machine code, this rule is not a requirement. However, there are tools (such as Arxan) that obfuscate code as it is compiled to machine code.

GEN02. Communication over HTTP is Forbidden

During production use, HTTP must not be used for any communication between the application and the EPS or Ezio Server. The application must prevent any URL using the HTTP protocol from being used or it must convert the HTTP protocol to HTTPS. Additionally, the EPS or Ezio Server must enforce the use of HTTPS to prevent insecure connections from being established unintentionally. If this is not done, it may be possible for an attacker to intercept communications between the server and the application, allowing the attacker to potentially obtain sensitive information that could be used to generate valid OTPs.

During development phase, HTTP could be used for any communication between the application and the EPS or Ezio server.

Do not use SSLv2 and v3 protocol because these have serious vulnerabilities (see Transport Layer Protection Cheat Sheet). TLS v1.2 protocol is recommended with strong algorithms. You may also refer to the *TLS Deployment Best Practices* from Qualsys SSLabs.

GEN03. Use POST, not GET

Data submitted over HTTPS must use the POST request method and never GET. This prevents a user from being tricked into following a URL that embeds data in its link. Furthermore, if the application uses a framework that logs or caches the URLs, then using a GET method will expose the data. In addition, a user account might get locked if a cached URL with an old OTP value is used multiple times.

GEN04. Reject Invalid SSL Certificates

HTTPS connections must reject an SSL certificate that is invalid for any reason. The certificate's revocation status must be checked against revocation list (CRL) retrieval or Online Certificate Status Protocol (OCSP). Check for certificate issued by a valid CA. SSL chain verification must be done. Do not use self-signed certificates. Also, check for expiration of certificates.

GEN05. No Debug Symbols
The application must not contain any debug symbol. This increases the complexity of reverse engineering efforts and prevents trivial identification of sensitive variables, structures and logic.
See also platform specific rule IOS06. Remove Symbols from Xcode Output.

GEN08. PIN Sanitization
The application must systematically erase or encrypt the PIN in memory immediately after usage and must not create additional references to the PIN. The PIN is the primary means of protecting the user's secret key. The PIN is not cached by the Ezio Mobile SDK. There should not be any PIN verification or such stop condition in the application. An OTP is always calculated by the SDK and verified on the server side. This has the drawback where the user may call the bank for support because he cannot distinguish if it was a user error or a bank error.

GEN09. Prevent Sensitive Data Leaks
The application developer must actively manage the life cycle of its application to prevent sensitive data from being leaked while in the background. It is recommended that any sensitive data displayed in the UI be removed before running the background and all sensitive data be wiped or encrypted until the application is restored to the foreground. Also, do not log any sensitive data.
See also platform specific rules AND01. Prevent Sensitive Data Leaks, and IOS01. Prevent Sensitive Data Leaks.

GEN10. Anti-cloning of Data
The application must use device and service (service fingerprint is not available on iOS) as sources of fingerprint data. This prevents the credentials from being used on another device or by another user on the same device.
The device fingerprint became compulsory starting from SDK version 3.0. All tokens created in 3.0 and later versions will be assumed to be bringing device fingerprint data by default. Refer to the *Ezio Mobile SDK Programmer's Guide* for more details.
In addition to the previous fingerprint sources, it is highly recommended to use custom fingerprint data when creating tokens. By using this extra data, the application provides yet another layer of security over the stored credentials. It is up to the application to determine the type of data to be used.

GEN12. Wiping of Assets
The application must wipe the following list of assets as soon as possible because this can be used by hackers to attack the device key:

- OTP, as soon as it has been accepted (or rejected) in the Ezio Server
- Registration code (RC)

For more information about wiping, see IOS05. Clearing of Assets and AND07. Clearing Assets Before a TLS Session.

GEN13. Verify the Integrity of Public Keys
Public keys, by design, do not need privacy protection and therefore are not usually encrypted. However, maintaining and examining their integrity and authenticity is of great importance. It is up to the application to verify the data integrity of the RSA public key. Hash functions provide a way to verify data integrity on a key.

GEN14. Use a Trusted Keyboard
Ezio Mobile SDK provides a secure PIN pad for the application as a UI component to be integrated into the application. It is recommended that the application uses the secure PIN pad for PIN input. This prevents a third-party keyboard from logging the user's PIN and other sensitive input data.

GEN15. Jailbreak and Root Detection

It is recommended that the application use the jailbreak and root detection services provided by the Ezio Mobile SDK. A device that is jailbroken or rooted presents increased opportunities for other applications to access private files without authorization. Therefore, if the application detects that the device is jailbroken or rooted, the application must limit its capabilities or silently notify the bank to assess the risk of generating OTPs on such a device.

It is also recommended to not only rely on the root detection service when the device is jailbroken or rooted but also consider the following options:

- The application developer must be encouraged to follow a uniform set of rules and guidelines for design and coding in order to create a secure application.
- Provide to the end users the awareness security information on the importance to keep their device up to date and using it in a safe manner.

GEN16. Use Separate Channels for Sensitive Data

It is recommended to separate communication channels for sensitive data. For example, use network access to get the RC but letter for PIN.

Notice that SMS is not out of band, which is very important because otherwise the security only relies on one message and not by separated channels. For instance, a tablet can be used for enrollment and the same device receives the SMS. SMS is relatively easy to intercept by using a malware.

GEN18. Ensure Application Origin

Provisioning is a sensitive sequence as an attacker can try to register a user account on a non-legitimate device. Application developers or solution architects must take this in account and offer a way to minimize risk of application spoofing. We recommend linking the reception of the RC with the download of the application. The objective is to avoid a rogue application which fakes the real bank application and ask for the user's RC and PIN. For example, it is possible to link the RC and application URL by sending both in one SMS, or display the RC on the bank web site and send the application URL by SMS. It should warn users that they need to download the application only via this link.

GEN21. Secure Coding Practices

The application developer must be aware of and follow known secure coding practices. This includes performing input validation, being careful with memory management, not using insecure C functions, avoiding the use of immutable containers when storing sensitive data, and so on.

For reference, see OWASP Secure Coding Practices Quick Reference Guide.

GEN22. Audits and Penetration Testing

Architecture and source code audits are highly recommended to evaluate the security of the mobile application.

Moreover, having a penetration test is recommended because it simulates an attack primarily on the application and also on the device, the network and other layers of the system. This helps to determine the feasibility of an attack, identify vulnerabilities, and provide other security related benefits for the application.

For reference, see OWASP Web Application Penetration Testing.

GEN23. Application Skin Modification

Applications that support skin modifications must offer simple customizations that may fool users with fake screens.

GEN24. EPS TLS Configuration

The TLS configuration, default TLS configuration is to be used. Customized configuration is only available for debug versions of the SDK and it is strictly for development purpose *only*. Use the release version of the SDK

for production. Otherwise, the security of the TLS connection can easily be compromised with any of the following settings:

- Hostname Mismatch - May expose the connection to man-in-the-middle attacks.
- Self-signed Certificates - Removes verification of the certificate by trusted authorities and may expose the connection to man-in-the-middle attacks.
- Insecure Connections - Disables TLS altogether thereby removing the security of TLS connection.

GEN25. PIN Rules

The application must configure the Ezio Mobile SDK to use the same type of PIN rules as the EPS uses. Additionally, the Ezio Mobile SDK must be configured to use any other PIN rules enforced by the bank policies (for example, the old and new PIN must not be identical).
Refer to the *Ezio Mobile SDK Programmer's Guide* for further information on setting the PIN rules.

GEN26. HTTP Header Customization

The HTTP header customization during the provisioning was supported for the purpose of authentication of HTTP requests. This helps our customers to filter the HTTP provisioning requests before they arrive on the EPS server, and thus increases both security and reliability at the same time.
The HTTP headers could be freely customized except for some commonly used ones such as "Host" and "Content-Length" due to the security consideration. For the detailed guidelines on HTTP headers customization and the whole list of prohibited headers, refer to the *Ezio Mobile SDK Programmer's Guide*.

GEN27. Secure Storage Usage

Ezio Mobile SDK provides Secure Storage for the end application to store data securely. The data is protected with a password and the device fingerprint. The password is managed by the Password Manager module. The use of default password is supported by the SDK, but it is strongly recommended to use a customized password for Secure Storage to minimize the risk of "Code-Lifting" attack.

Warning:

Ezio Mobile SDK *does not* recommend to store user PIN or other data which are used to provide new factor of authentication.

GEN28. Authentication Input Caching

Ezio Mobile SDK 4.0 removes the use of SecurePin for OTP computation. AuthInput is introduced and different Authentication modes provides different types of AuthInput, For example, PinAuthInput is created from user PIN, BioFingerprintAuthInput is created after user authenticates with his Biometric fingerprint and FaceAuthInput is created after user authenticates with the face data enrolled in the to system. It is recommended to wipe it as soon as it is not needed. AuthInput can be re-used for consecutive OTP generation until 'wipe' is called on the AuthInput object. To avoid auto generation of OTP for unlimited duration, it is recommended to enforce the wiping on the exit point of the application/screen, for example onPause()for android and viewDidDisapper for iOS or to check the device state (if idle for a preset duration). It is also recommended to limit the number of OTP generations without inputting a PIN.

GEN29. Matching Threshold and FAR/FRR for FaceID Authentication

FaceID authentication mode is introduced in Ezio Mobile SDK 4.2. Biometric features matching algorithm provides similarity score as a result. The higher is score, the higher is probability that features collections are obtained from the same person.The higher the threshold value, the more similar feature collections will have to be to yield positive result during matching.
Matching threshold is the minimum score that verification and identification functions accept to assume that the compared face belongs to the same person. Matching threshold is linked to false acceptance rate (FAR, different subjects erroneously accepted as of the same) of matching algorithm. The higher is threshold, the

lower is FAR and higher FRR (false rejection rate, same subjects erroneously accepted as different) and vice a versa. Matching should be determined from this table:

FAR (false acceptance rate)	Matching threshold (score)
100 %	0
10 %	12
1 %	24
0.1 %	36
0.01 %	48
0.001 %	60
0.0001 %	72
0.00001 %	84
0.000001 %	96

Matching threshold/FAR should be selected according to the system's development requirements and taking into account mentioned identification false acceptance accumulation.

Note:

This rule is only applicable for Ezio Mobile SDK Full release package.

Rules for Android

AND01. Prevent Sensitive Data Leaks

For background information, see GEN09. Prevent Sensitive Data Leaks.

The application developer must prevent screenshots and override the onPause() method (see this Android developer reverence) as specified in GEN09 to prevent sensitive data from being leaked.

An example of sensitive data being leaked in the UI is when an OTP is displayed to the user. Starting with Android 4.0, the task switcher can take a screenshot of applications that are running in the background. If the application is actively displaying an OTP, then it is possible to retrieve the OTP from a screenshot taken by the task switcher service (or screenshot taken by the user). This sensitive data leak can be prevented by the app's Activity.onCreate method with the following code:

```
getWindow().setFlags(LayoutParams.FLAG_SECURE, LayoutParams.FLAG_SECURE);
```

In Android 4.4 there is a feature that allows a user to record the screen activity. To prevent this, add `SurfaceView.setSecure` (true) into the `Activity.onCreate()` method.

AND03. Password-Protect Secondary Assets

It is recommended that a password is used to protect secondary assets (see *Ezio Mobile SDK V4.8 Overview*). This significantly increases the security of secondary assets over the default protection scheme. However, this requires the user to enter their password in order to perform actions on the Token application.

AND04. Preventing Excessive Permissions

It is recommended that an application does not enable Android permissions that are not required to prevent an attacker from retrieving certain information.

AND05. Preventing Logging Services

It is recommended that an application keeps Android logging services at the minimum level and avoid logging any sensitive information to prevent the enabling of excessive permissions if it does not require them.

AND06. Android Security Tips

The application developer must be aware of and follow security features and technologies provided by Android. See Android Security Tips.

AND07. Clearing Assets Before a TLS Session

For background information, see GEN12. Wiping of Assets.

To prevent attacks that exploit the Heartbleed OpenSSL vulnerability, it is recommended to:

- Wipe the registration code immediately after the token builder creation. For example:

```
ProvisioningConfiguraiton epsCofig = new
EpsConfigurationBuilder(registrationCode,
…).build();
registrationCode.wipe();
```

- Wipe assets not required for a TLS session prior to establishing the session. This includes TLS sessions established during the provisioning phase and any TLS sessions the application establishes. The following are specific examples but are not exhaustive:
  - Never ask the user for the PIN until the provisioning has completed. Therefore, even though it may be more convenient to the user, do not ask for the registration code and the PIN at the same time.
  - Wipe all assets for existing tokens prior to provisioning a new token. This only applies to applications that manage several tokens.
  - The application must wipe the PIN or any other unnecessary asset prior to establishing a TLS session that verifies an OTP.

AND08. Using Secure PIN Pad

Ezio Mobile SDK provides a secure PIN Pad for the application as a UI component to be integrated into the application.

- Each secure PIN Pad view component instance is designed to be used *only once*.
- The application must wipe any data resided in the secure PIN Pad by calling the cleanup method provided by the SDK after receiving the user PIN.
- The application must pass the PIN directly into the SDK for further usage *without* any manipulation on the returned PIN object.
- In the event of exiting the application while the secure pin pad is still visible (for example, the user presses the 'Home' button), it is highly recommended that the application uses the provided clean up method to wipe the PIN Pad and destroy/remove the secure PIN Pad from the view.

Refer to the *Ezio Mobile SDK Programmer's Guide* for further information regarding the use of Secure PIN Pad.

AND09. Preventing Text Cut/Copy/Paste

Starting from Android API 18 (also available for previous versions of Android through Android Support Library), the new APIs provided by `AccessibilityNodeInfo` allow `AccessibilityService` to select, cut, copy and paste text in a node. It is recommended to *not* use these APIs on the screens with sensitive data in order to prevent an attacker from retrieving certain information.

AND10: Preventing the Tampering of the Application

To prevent malicious hacking into the application code, it is recommended to verify the application's signing certificate signature hash at run time. This signature cannot be replicated by attackers when they sign the application after tampering the code or manifest file. However, a strong obfuscation is required to protect the

signing certificate hash from modification. String encryption can be applied to the hash through tools such as Dexguard or Arxan.

AND11: Verify the Installer
If you are distributing the app via Google Play Store, it is recommended to check if the installer of the application corresponds to the Play Store package (`com.android.vending`).

AND11: Preventing Execution on an Emulator
In a normal scenario, your app will be running on the device except during development environment. It is recommended to check if the app is running on a device or emulator.

AND12: Enabling touch filtering on View
Sometimes it is essential that an application be able to verify that an action is being performed with the full knowledge and consent of the user, such as granting a permission request, making a purchase or clicking on an advertisement. Unfortunately, a malicious application could try to spoof the user into performing these actions, unaware, by concealing the intended purpose of the view. As a remedy, the Android framework offers a touch filtering mechanism that can be used to improve the security of views that provide access to sensitive functionality.
APIs for enabling touch filtering are below:

```
setFilterTouchesWhenObscured(boolean);
onFilterTouchEventForSecurity(MotionEvent);
```

AND13: Binary obfuscation guidelines
Ezio Mobile SDK is delivered with all internal implementation obfuscated. However, public APIs are preserved for SDK integrator usage. It is required to perform obfuscation on the final application eg. Proguard/Dexguard or other obfuscation tools. Due to the usage advanced obfuscation features and Android native code implemnetation, there are packages that has to be preserved during application obfuscation. It is extremely important to follow the obfuscation recommendations provided.
Refer to Programmers Guide on the details and pay special attention to:

```
-flattenpackagehierachy util
```

Using package name '**util**' is recommended to ensure obfuscation consistency. The preverved package from Ezio Mobile SDK are sensitive classes which should not 'stand out' in the final binary.

Rules for iOS

IOS01. Prevent Sensitive Data Leaks
For background information, see GEN09. Prevent Sensitive Data Leaks.
The application developer must use the following methods for managing sensitive data in the application lifecycle (see the Apple developer site):

- `applicationWillResignActive` - Prepare to clear the display of and encrypt any sensitive data. To clear the display, the following code can be used to hide the displayed content:

```
[UIApplication sharedApplication].keyWindow.hidden = YES;
```

- `applicationDidBecomeActive` - Prepare to display and decrypt sensitive data.

IOS05. Clearing of Assets

It is recommended that a memory sanitization routine be created that overwrites variables and safely releases them from the allocated process space back to the system pool (such as heap space) to prevent information leaks from occurring within memory. It is highly recommended that any variables that are handled for cryptographic purposes, such as the PIN, are sanitized in this way in order to prevent malware harvesting this information from compromised iOS devices.

Note:

For Objective-C strings, it may be impossible to properly wipe NSString objects. It is strongly recommend to use NSMutableString or, better yet, byte arrays for any sensitive data that will need to be wiped from memory after usage.

IOS06. Remove Symbols from Xcode Output

The application must remove all symbols from the final release binary. The following settings are recommended to be set in the Xcode project in order to remove debug information and other symbols:

- `DEPLOYMENT_POSTPROCESSING = YES`
- `GCC_GENERATE_DEBUGGING_SYMBOLS = NO`
- `STRIP_INSTALLED_PRODUCT = YES`
- `STRIP_STYLE = all`
- `COPY_PHASE_STRIP = YES`

IOS07. Xcode Compiler Scurity and Obfuscation Options

The application must use Xcode options that increase security and provide obfuscation by complicating disassembly. The following settings are recommended to be set in the Xcode project:

- `GCC_UNROLL_LOOPS = YES`
- `GCC_OPTIMIZATION_LEVEL = 3`
- `OTHER_CFLAGS = "-fstack-protector-all -finline-functions "`
- `CLANG_ENABLE_OBJC_ARC = YES`
- `GCC_DYNAMIC_NO_PIC = NO`
- `LD_NO_PIE = NO`
- `RUN_CLANG_STATIC_ANALYZER = YES`
- `CLANG_ANALYZER_SECURITY_FLOATLOOPCOUNTER = YES`
- `CLANG_ANALYZER_SECURITY_INSECUREAPI_RAND = YES`
- `CLANG_ANALYZER_SECURITY_ INSECUREAPI_STRCPY= YES`
- `CLANG_ANALYZER_SECURITY_ INSECUREAPI_GETS= YES`
- `CLANG_ANALYZER_SECURITY_ INSECUREAPI_GETPW= YES`
- `CLANG_ANALYZER_SECURITY_ INSECUREAPI_MKSTEMP= YES`
- `CLANG_ANALYZER_SECURITY_ INSECUREAPI_MKTEMP= YES`
- `CLANG_ANALYZER_SECURITY_ INSECUREAPI_VFORK= YES`

IOS08. Objective-C Instance Variable Vulnerability

The application must not store sensitive assets (such as pointers to encryption keys) in Objective-C instance variables because it is possible to reference these variables by using the Objective-C runtime [iOS_Hacking]. Automatic and dynamically allocated variables that are not Objective-C variables are safer.

IOS09. Disable Auto-correction Cache for Sensitive Input

The application must disable the auto-correction cache for inputs that request sensitive data (such as PIN). This prevents an attacker with access to the device from using the autocomplete suggested strings to view the sensitive text input data. The application may perform one of the following actions to disable the auto-correction cache:

- Set the text field's `secureTextEntry` to YES.
- Set the text field's `autoCorrectionType` to `UITextAutocorrectionTypeNo`.

IOS10. Disable Data Copy/Paste for Sensitive Data

The application must disable the copy/paste menu for sensitive data. This prevents an attacker with access to the device from pasting and viewing the copied data. The following sample code disables the copy/paste menu:

```
-(BOOL)canPerformAction:(SEL)action withSender:(id)sender {
      UIMenuController *menuController = [UIMenuController sharedMenuController];
      if (menuController) {
          [UIMenuController sharedMenuController].menuVisible = NO;
      }
     return NO;
}
```

IOS11. Implement Inline Jailbreak Detection

The jailbreak detection API is implemented in C. C function names can be stripped from the final binary, while Objective-C method names will still be present. It is therefore easier to identify and hook Objective-C methods compared to C functions. For more information about symbol stripping, see IOS06. Remove Symbols from Xcode Output.

These properties of Objective-C must be kept in mind when implementing jailbreak detection in the application. Instead of performing the checks in a separate Objective-C method, they should be performed inline prior to critical security operations.

```
-(void)generateOtp
{
     if (EMJailbreakDetectorGetJailbreakStatus() == EMJailbreakStatusJailbroken){
         //Abort
         return;
     }
     //No jailbreak detected. Complete the otp generation…
}
```

IOS12. Secure PIN Pad

Ezio Mobile SDK provides a secure PIN Pad for the application as a UI component to be integrated into the application.

- Each secure PIN Pad view component instance is designed to be used *only once*.
- The application must wipe any data resided in the secure PIN Pad by calling the cleanup method provided by the SDK after receiving the user PIN.
- The application must pass the PIN directly into the SDK for further usage *without* any manipulation on the returned PIN object.
- In the event of exiting the application while the secure pin pad is still visible (for example, the user presses the 'Home' button), it is highly recommended that the application uses the provided clean up method to wipe the PIN Pad and destroy/remove the secure PIN Pad from the view.

Refer to the *Ezio Mobile SDK Programmer's Guide* for further information regarding the use of Secure PIN Pad.

IOS13. Managing App Versions in the App Store

Prior to iOS7, users were only able to download the latest version of an application available in the App Store. Along with the release of iOS7, it is now possible for users to re-download previous versions that were already purchased or installed, allowing customers to use apps with older devices that may no longer be supported by the current version of your application. If you do not wish to make these versions available, you can manage the availability of your apps' previous versions in the Rights and Pricing section of the Manage Your Apps module in iTunes Connect. For details, refer to iTunes Developer Guide.

From a security standpoint, it is recommended to keep attack surfaces as small as possible. While keeping older versions provides a much wider audience, it can also expose the solution to old firmware whose security may have already been compromised. It is best to use one version of your application (the latest one) available at one time, and configure your to build settings to support wider iOS versions, instead.

IOS14. About Touch ID Accuracy

According to Apple's security document, the Touch ID probability of a false positive is 1/50000 and enforces a maximum of five tries before blocking this authentication method (the device passcode is then needed to unblock thereafter). The probability that at least one try successfully authenticates a person not enrolled is: $p=1-(1-1/50000)^5$, which is equivalent to 9.99×10^{-5}

This result is better than a PIN of four digits with three tries.

Terminology

This section contains abbreviations and terms found in this document or related documents.

Abbreviations

EPS	Enrollment Provisioning Server
ESN	Electronic Serial Number
HOTP	HMAC based One Time Password
HSM	Hardware Security Module
IMEI	International Mobile Equipment Identifier
IMSI	International Mobile Subscriber Identifier
OATH	Open Authentication
OOB	Out Of Band
OCRA	OATH Challenge-Response Algorithm
OTP	One Time Password
PM	Password Manager
PIN	Personal Identification Number
PTC	PIN Try Counter
RC	Registration Code
SM	Security Module (see also HSM and SSM)
SMS	Short Message Service
SSM	Software Security Module
TLS	Transport Layer Security
TOTP	Time-based One Time Password
URL	Universal Resource Locator
VIC	Verify Issuer Code
VICATC	VIC Application Transaction Counter
VICTC	VIC Try Counter
DSKPP	Dynamic Symmetric Key Provisioning Protocol

Glossary of Terms

Provisioning Access Domain is a SIM Toolkit/UICC Toolkit parameter that specifies the identities or access rights (CHV & ADM) granted to an application to access GSM/UICC files and perform actions on these files. For example, if the Access Domain value is "FF" (No Access to the File System), attempts to access a file cause an exception.
Token The Ezio Mobile SDK's representation of a user's credentials.
User The mobile device user.