Secure implementation of Certificate pinning

Version: 1.0

2026-03-05

Contents

1. Introduction

When developing mobile applications, if the application communicates with a backend system, a common security solution is certificate pinning.

When using this solution, we specify in the mobile application which certificate we expect the backend system to present. This prevents man-in-the-middle (MITM) attacks, even if the attacker uses a certificate from a compromised or fake certificate authority.

This document shows the correct setting for certificate pinning, so that the configuration always includes a backup pin.

2. Pinning strategies

Possible pinning strategies are summarized in the table below, depending on which element is pinned in the application::

Solution	**Advantage**	**Disadvantage**	**Comment**
Pinning of the complete end entity certificate **Recording the**	The system does not accept certificates other than those specified, making it impossible to misuse other certificates issued by fake or compromised CAs.	If the certificate needs to be replaced (which is relatively common), the application will become inoperable until the new certificate is issued and pinned.	Not recommended.
Pinning of the public key fingerprint contained in end entity certificate	The system does not accept certificates other than the one specified, making it impossible to misuse other certificates issued by fake or compromised CAs. Upon renewal, the certificate can be requested again with the same key, and the application will continue to function.	If the key needs to be replaced (less common), the application will become inoperable until the certificate containing the new key is issued and pinned.	**This is the recommended solution for both Android and iOS.**
Pinning of the intermediate or Root CA certificate (chain of end entity)	Requires minimal maintenance.	If the CA that issued the certificate is compromised, the application will accept all certificates issued by it.	Not recommended because the pinning is too loose.

3. Why is the backup pin mandatory?

Certificate pinning without a secondary certificate/key backup pin is an artificially created single point of failure (SPOF).

Both the Android and iOS developer guides recommend to make redundant the certificate pinning:

- Android: According to the Network Security Configuration documentation, the lack of a backup key will lead to connection failure (bricking) in the event of a key change, so the system design must include at least one backup fingerprint.
- iOS: According to Apple's Apple Identity Pinning guidelines, part of the long-term strategy is to handle planned and unplanned key changes, which cannot be achieved without redundant key pairs.

In addition, the ISO/IEC 27001:2023 standard, which describes guidelines for information security management systems, stipulates that:

> *ICT readiness shall be planned, implemented, maintained and tested based on business continuity objectives and ICT continuity requirements.*

Based on this, if our high-availability application does not have backup solutions for certificate pinning, its design was not adequate for business continuity.

4. Recommendations for creating a backup key

In order for the certificate pinning solution to be redundant, it is not necessary to request two TLS certificates for the backend server at the same time, but it is worth considering.

The key is to generate at least a second (backup) key and store it securely so that you can quickly request a certificate if necessary.

This is essential because if a certificate in use must be revoked by the Trust Service Provider for any reason, depending on the conditions, there is a 1–5-day revocation obligation that cannot be deviated from.

5. Solutions for specific mobile platforms

The following sections provide examples of certificate pinning settings for Android and iOS systems. To do this, you first need to generate the key hash used for pinning.

5.1.1. Generating the keyhash

Egy nyilvános kulcs Base64-kódolt SHA-256 lenyomatát (hash-ét) a következő openssl paranccsal tudjuk előállítani (ez a leggyakrabban használt megoldás; amennyiben más reprezentációra van szükségünk, ahhoz az openssl felhasználói kézikönyve nyújt iránymutatást):

A Base64-encoded SHA-256 hash of a public key can be generated using the following openssl command (this is the most commonly used solution; if you need a different representation, refer to the openssl user manual for guidance):

From a certificate:

```
openssl x509 -in certificate.pem -pubkey -noout | openssl pkey -pubin -outform der | openssl dgst -sha256 -binary | openssl enc -base64
```

From a keypair:

```
openssl pkey -in private_key.pem -pubout -outform der | openssl dgst -sha256 -binary | openssl enc -base64
```

Important!

Apple's ATS system **expects Base64 according to the RFC 4648 standard** and often rejects or considers the URL-safe version to be incorrect, so make sure that it does not contain underscores or hyphens.

5.2. Native Android application

Since Android 7.0 (API 24), the official and declarative way to do certificate pinning is through Network Security Configuration.

To do this, we need to create the **res/xml/network_security_config.xml** configuration file with the appropriate content.

When using the example configuration below, the application works as follows:

- When accessing valami.hu or any of its subdomains, the application accepts certificates issued to the domain name valami.hu and containing any of the specified key hashes, because the `includeSubdomains="true"` option also allows subdomains.
- The configuration includes a backup pin, which is required for security reasons:
  - The official Android developer documentation recommends this in the Network Security Configuration chapter.
  - As part of the Google Play App Security Improvement (ASI) program, Google Play automatically scans uploaded APKs.

If the uploaded app uses Certificate Pinning but does not contain a backup key, the Play Console will send a security warning or a recommendation.

5.2.1. Example configuration

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">valami.hu</domain>

        <pin-set>
            <pin digest="SHA-256">NzbLsXh8uDC9W4fGfq4Z_96mI0-u-6mXN63Ym3lQ020=</pin>
            <!-- backup pin -->
            <pin digest="SHA-256">p1M5fG8zX2y7Q9wL4vK6nJ3mR1tS0vB9A8C7D6E5F4G=</pin>
        </pin-set>

        <trust-anchors>
            <certificates src="system" />
        </trust-anchors>
    </domain-config>
</network-security-config>
```

5.3. Native iOS application

Since iOS 14, it has been possible to configure certificate pinning in Apple products via a configuration file.

This can be done by adding the appropriate configuration XML section to the **Info.plist** file.

When using the configuration in the example below, the application does the following:

- When accessing valami.hu or any of its subdomains, the app accepts any certificate with a key hash specified for the domain name valami.hu (in the string lines under `SPKI-SHA256-BASE64`) because the value of `NSIncludesSubdomains` is `true`.
- The configuration includes a backup pin, because according to Apple's Identity Pinning guidelines, the "long-term strategy" must include the management of planned and unplanned key changes, which cannot be implemented without redundant key pairs.

5.3.1. Example configuration

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSExceptionDomains</key>
    <dict>
        <key>valami.hu</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSRequiresCertificatePinning</key>
            <true/>
            <key>NSPinnedLeafIdentities</key>
            <array>
                <dict>
                    <key>SPKI-SHA256-BASE64</key>
                    <string>NzbLsXh8uDC9W4fGfq4Z/96mI0+u+6mXN63Ym3lQ020=</string>
                </dict>
                <dict>
                    <key>SPKI-SHA256-BASE64</key>
                    <string>p1M5fG8zX2y7Q9wL4vK6nJ3mR1tS0vB9A8C7D6E5F4G=</string>
                </dict>
            </array>
        </dict>
    </dict>
</dict>
```

6. Shorter certificate lifetimes and ACME

Due to the expected shortening of TLS certificate lifetimes, the use of ACME is inevitable, so the following section describes the use/configuration of two ACME clients from this perspective.

6.1. Certbot

By default, certbot generates a new key, but this can be overridden, which is recommended in the case of certificate pinning.

The following command modifies the renewal configuration of the CERTIFICATENAME certificate so that the key will be reused:

```
certbot reconfigure --cert-name CERTIFICATENAME --reuse-key
```

6.2. Acme.sh

With acme.sh, key reuse is the default, so the configuration is suitable for certificate pinning by default.