Article

Licensing alternative distribution apps

Build a license server that supports the installation of your apps and the apps available in your marketplace.

Overview

iOS and iPadOS require each app that installs outside of the App Store to have a license issued by the developer. As the developer of an alternative app marketplace or other app that installs over the web, you use the App License Delivery SDK to generate a license for each download request for your app. Alternative app marketplaces also create a license for each download request for the apps that they distribute.

Before continuing, ensure you complete the steps in either Creating an alternative app marketplace or Distributing your app from your website.

The MarketplaceKit installation methods trigger the device's operating system to request a license from your web server before installing a particular app. To support installation of your app or the apps on your marketplace, implement a license server to process the requests. Your license server consists of two endpoints that use this SDK: one that creates licenses for new installations and another that updates the licenses of existing installations.

Important

Prepare your App License Delivery (ALD) encryption assets and development environment for use with the licensing workflow. For more information, see Configuring your app licensing environment.

Publish your licensing endpoint details

To inform the system of the details of your license server, publish a `marketplace-kit` configuration file in the standard location. The system checks for the file at the following relative path:

```
https://<fully qualified domain>/.well-known/marketplace-kit
```

The base domain, `<fully qualified domain>`, of the above URL is from the app's domain that you add to App Store Connect. For more information about adding and managing domains in App Store Connect, see Alternative Distribution Domains.

This URL is your licensing endpoint, so your web server needs to serve a JSON configuration file that identifies your license server details:

```
{
    "license": {
        "dynamicLicenseURL" : "<licensing endpoint>",
        "licenseRenewalURL" : "<renewal endpoint>",
        "licenseResolutionURL": "<resolution webpage>",
        "signingCertificateURL" : "SIGNING_CERTIFICATE_URL",
        "encryptionCertificateURL" : "ENCRYPTION_CERTIFICATE_URL",
    }
    "restore": ...
    "updates": ...
}
```

The system requires that the "`license`" key specifies the following properties:

`license` key	Property
`dynamicLicenseURL`	An endpoint that generates licenses for apps and app versions new to a device.
`licenseRenewalURL`	An endpoint that generates updated licenses for apps and app versions already installed on the device. For more information, see Renewing and revoking app licenses.
`licenseResolutionURL`	A webpage that gives a person more information about an expired app license or the opportunity to renew the license. For more information, see Renewing and revoking app licenses.

`license` key	Property
`signingCertificateURL`	A certificate that signs liceneses you create.
`encryptionCertificateURL`	A certificate that the system uses to encrypt license requests.

> **Note**
>
> The "`restore`" and "`updates`" keys configuration require you to set a value. For more information, see Installing your app from your website for alternative marketplace apps, and Installing apps from an alternative marketplace for other apps.

Host the file using the `https://` URL scheme with a valid certificate, and don't use redirects. This configuration publication follows the same pattern as the `apple-app-site-association` file. For more information, see Supporting associated domains.

For more information about the signing and encryption certificates, see Configuring your app licensing environment.

Handle a dynamic license request

A license request consists of a single POST to your licensing endpoint. Each POST can contain a request for a license for one or more apps. Your license server sends a single response to the POST which typically includes one license for each requested app. The license request is *dynamic*, meaning that your license server provides a new license for each install request.

If you limit app licenses to authenticated devices, check the POST header for an access token that links the license request to an account. The following is an example header:

```
POST /resource HTTP/1.1
Host: server.example.com
Authorization: Bearer mF_9.B5f-4.1JqM
```

Your token endpoint issues the bearer token to the signed-in person at the start of the app download request and the system adds it to all further communication with your server for their account. Use your rules to decide whether to issue the licenses in the request for the account. For more information about authorization, see Installing your app from your website.

The following example reflects the body of a dynamic license request the system makes to your licensing endpoint:

```
"licenseRequest": "ZpGAfk4LjzwrVVrBo9mwll0vQ/tbtsrdP18t8bK[...]",
"licenseKey": "T3b3pO5Z1BCsnBFDAMJv8PHW0JXQ/WYOYzZoSJJMUH/[...]",
"appsById": {
      "6476788646": {
          "appleVersionId": "863214182",
          "assetPublicId": "d48693c5-214e-42e3-ae52-2fa4fa9326bc"
      },
      "6763788646": {
          "appleVersionId": "891242134",
          "assetPublicId": "3bbb71c5-ad23-4df4-a779-dd9e797fbb40"
      },
}}
```

The license request payload contains the following data:

License request payload key	Value
`licenseRequest`	A `base64` encoded license request that's AES CBC encrypted.
`licenseKey`	An AES key/IV that's `base64` encoded and RSA-OAEP encrypted. This key encrypts the license request.
`appsById`	A set of one or more identifiers (`AppleItemID`) that refer to an app that your marketplace distributes.

Each item in `appsById` describes the following data:

`appsById` key	Value
`appleVersionID`	The app version (`AppleVersionID`) that the device requests.
`assetPublicId`	The app variant that the system chooses for the device.

To begin processing the request, parse the payload using a `JSONDecoder`:

```
struct LicenseRequest: Decodable {
    let licenseRequest: String
    let licenseKey: String
    let appsById: [String:VersionAndPublicID]
}

struct VersionAndPublicID: Decodable {
    let appleVersionID: String
    let assetPublicId: String
}

func processRequest() {
    let jsonContents: URL = /* The JSON contents. */
    let data = try Data(contentsOf: jsonContents, options: .mappedIfSafe)
    let request: LicenseRequest = try!
        JSONDecoder().decode(LicenseRequest.self, from: data)
```

Organize the payload components in the right format. Decode the `licenseRequest` and `licenseKey` (which are both `base64` encoded) to reveal their raw, encrypted representations:

```
let encryptedLicenseRequest = Data(base64Encoded: request.licenseRequest)!
let encryptedRequestKey = Data(base64Encoded: request.licenseKey)!
```

Validate that `encryptedRequestKey` is RSA3072 by ensuring `count` is the expected number of bytes:

```
guard encryptedRequestKey.count == 384 else { return nil }
```

Then, load your encryption certificate private key (in PEM or DER format) as necessary to decrypt the license request:

```
let encryptionCertificateKeyPath = "path/to/<encryption-certificate-key-filename>"
let encryptionCertificateKey = try String(contentsOf:
    URL(fileURLWithPath: encryptionCertificateKeyPath))
```

Decrypt the request payload

The system encrypts the payload `licenseRequest` and sends the request to your server. Then, your license server uses your App License Delivery assets in combination with the payload

`licenseKey` to decrypt the `licenseRequest` payload and create a license. To assist with decryption, use swift-crypto:

```
import Crypto // Requires the `swift-crypto 3.2.0` package.
import _CryptoExtras
```

Extract the AES key/IV by decrypting the AES key with the encryption certificate's private key. Use `.PKCS1_OAEP_SHA256`, as RSA decryption specifies a padding mode of OAEP and a SHA256 hash function with MGF1 and OAEP:

```
var rsaDecryptResult: Data
do {
    let derPrivKey = try _RSA.Encryption.PrivateKey(
        /* Private key is PEM format in this example. */
        pemRepresentation: encryptionCertificateKey)
    rsaDecryptResult = try derPrivKey.decrypt(encryptedRequestKey,
        padding: .PKCS1_OAEP_SHA256)
} catch {
    print("Failed to decrypt: \(error)")
    return nil
}
```

In the 32-byte decryption as a `UInt8` array, access the AES key in the byte range `[0–15]`, and the IV in byte range `[16–31]`:

```
let keyBytes = rsaDecryptResult.bytes
let keyData = keyBytes[0..<16]
let ivData = keyBytes[16..<32]
```

Next, decrypt the encrypted license request with the extracted AES key and IV using the CBC decryption mode. The encrypted request uses PKCS #7 padding:

```
do {
    let decryptedRequest = try AES._CBC.decrypt(
        encryptedLicenseRequest, using: SymmetricKey(data: keyData),
            iv: AES._CBC.IV(ivBytes: ivData)
    )
    return decryptedRequest
} catch {
    print("Failed to decrypt: \(error)")
```

```
    return nil
}
```

Start a license session

The decrypted payload includes details about the apps for which the system requests a license. To assist with creating licenses for the apps, the framework needs your ALD encryption assets.

The PASK, which you can find at Certificates, Identifiers & Profiles, is a JSON file with a `base64` encoded authorization key inside (`authorizationKey`). Extract the `authorizationKey` value to its own file on disk. The following code decodes the `base64` contents of the value for use with the framework:

```
let paskPath = "path/to/<PASK-authorization-key-filename>"
let pask = Data(base64Encoded:
    try String(contentsOf: URL(fileURLWithPath: paskPath)))!.bytes
```

Load the ALD encryption and signing certificates from disk:

```
let encryptionCertPath = "path/to/<encryption-certificate-filename>"
let encryptionCert = try Data(contentsOf:
    URL(fileURLWithPath: encryptionCertPath)).bytes

let signingCertificatePath = "path/to/<signing-certificate-filename>"
let signingCert = try Data(contentsOf:
    URL(fileURLWithPath: signingCertificatePath)).bytes
```

Prepare a `.DER` file with `ASN.1` encoding for your signing certificate private key and load it as follows:

```
let signingKeyPath = "path/to/<signing-key-filename>"
let signingKey = Data(base64Encoded: try String(contentsOf:
    URL(fileURLWithPath: signingKeyPath)))!.bytes
```

Create an `ALDProvider` instance by calling the `init(encryptionCert:signingCert:PASK:signingKey:)` initializer, passing your ALD encryption assets:

```
let provider = ALDProvider(encryptionCert: encryptionCert,
    signingCert: signingCert, PASK: pask, signingKey: signingKey)
```

Then, hand the decrypted payload to the framework through a session object. Create a session by calling the provider's `createSession(clientRequest:)` method, which returns an `ALDSession` instance:

```
let session = try provider.createSession(clientRequest: decryptedRequest.bytes)
```

Review the session's `requestedAppleItemIDList` list to confirm which of the requested apps you approve for download. If the device isn't eligible for a particular app — for example, if a required subscription lapsed — you can omit a license for that app in the response.

Generate a dynamic license

With the session object, generate a license for each app that you approve for download.

1. Choose a value for the license ID. Ensure the ID is unique across all the licenses your license server distributes.
2. Create a license attribute for the license ID by calling `init(licenseID:)`.
3. Set the `issuedTime` and a `duration`, in seconds, that determines when the license expires. For example, setting `duration` to `86400` prevents the licensed app from launching after a day.
4. Set the `appKey` for the app, or *key blob*, which is unique per app variant. Refer to `appsById` in the request payload for the app ID, and `assetPublicId` for the variant that iOS needs for that app. App Store Connect provides the key blob during app ingestion. For more information, see Ingesting an alternative distribution package.
5. Create the license by calling the `generateLicense(attr:)` method with the license attribute.

```
let licenseID: UInt64 = 1 // Placeholder value.

var attribute = ALDLicenseAttribute(licenseID: licenseID)
attribute.issuedTime = UInt64(Date.now.timeIntervalSince1970)
attribute.duration = 86400 // One day; for example only.

let appKeyBlob = try Data(contentsOf: URL(fileURLWithPath: appKeyBlobPath)).bytes
let appkey = ALDAppKey(appItemID: appID, appKeyBlob: appKeyBlob)
try attribute.addAppKey(appKey: appKey)

try session.generateLicense(attr: attribute)
```

If the system requests a license for multiple apps in its POST, repeat this process for each app. Every time you call `generateLicense(attr:)`, the framework queues an additional license for the response.

> **Important**
>
> In iOS 17.4, set a value less than `Int64.max`; see iOS & iPadOS 17.4 Release Notes.

Respond with the generated licenses

The framework assists you with preparing a response to the original POST. Call `generateLicenseResponse()` begin the response:

```
let response = try session.generateLicenseResponse()
```

Then, call `finalizeLicenseResponse(licenseResponse:signature:)` to retrieve the response data.

```
let staticBlob = try session.finalizeLicenseResponse(licenseResponse: response)
```

The response data contains the license(s), signing, and encryption certificates. To prepare for transit over the network, validate the data and encode it in `base64`:

```
if staticBlob.count > 0 {
    print("Dynamic license created")
    let data = Data(staticBlob)
    license = data.base64EncodedString()
    print(license)
} else { print("Failed to generate dynamic license") }
```

The format of the license response payload is:

```
{
    "license" : "<BASE_64_ENCODED_LICENSE>",
    "unlicensedApps": [
        "<An app's appleItemID>",
        "<Another app's appleItemID>",
        ...
```

```
    ]
}
```

License response payload key	Value
`license`	The `finalizeLicenseResponse` return result that you encode in `base64`. This value contains the license(s) that you added to the session.
`unlicensedApps`	An array that contains an `AppleItemID` for each app from the request for which you choose not to provide a license.

Important

To complete an installation, the system requires a response from your server within 60 seconds of making the request. If you use a web debugging proxy tool to test license generation, make sure it doesn't interfere with your server's ability to respond promptly.

When the system receives the `license` response, it validates the licenses contained within according to the signing and encryption certificates in your JSON configuration file (specifically, `signingCertificateURL` and `encryptionCertificateURL`).

Then, the system downloads the licensed app(s) from your app web server. For more information about serving app downloads, see:

- Installing your app from your website for alternative marketplace apps or other apps that install from your website.
- Installing apps from an alternative marketplace for other apps.

Track the number of simultaneous app installs for an account

When the system sends a request to your license server, it includes the authentication token in the header. Use the token to associate the request with a specific person's account.

The `ALDSession` object's `requestDeviceID` property represents a unique ID for the device on which the person requests to install the app.

You can determine the number of active installs of a particular app for an account by counting the unique device IDs your license server encounters minus any unique device IDs associated with revoked or expired licenses.

See Also

App licensing

Renewing and revoking app licenses

Determine whether an app for which you issue a license launches.

`struct` `ALDAppKey`

A structure that identifies an app and a key that's required to decrypt the app's license request.

`struct` `ALDLicenseAttribute`

A structure that defines the requested license type for the session.

`class` `ALDProvider`

An object that creates a session with the alternative app marketplace's signing assets.

`class` `ALDSession`

A structure that contains the details of a license request and methods to generate license responses.