Article

Generating Tokens for API Requests

Create JSON Web Tokens (JWTs) signed with your private key to authorize API requests.

Overview

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a way to securely transmit information. The App Store Connect API requires JWTs to authorize each API request. You create the token, and sign it with the private key you downloaded from App Store Connect.

To generate a signed JWT:

1. Create the JWT header.
2. Create the JWT payload.
3. Sign the JWT.

Include the signed JWT in the authorization header of each App Store Connect API request.

Create the JWT Header

To create a JWT to communicate with the App Store Connect API, use the following fields and values in the header:

Header Field	Value
`alg` - Encryption Algorithm	ES256 All JWTs for App Store Connect API must be signed with ES256 encryption.

Header Field	Value
`kid` - Key Identifier	Your private key ID from App Store Connect, for example, 2X9R4HXF34
`typ` - Token Type	`JWT`

To get your key ID for your team API key from App Store Connect, log in to App Store Connect and:

1. Select Users and Access.
2. Select the Integrations tab.

The key IDs appear in a column under the Active heading.

1. Hover the cursor next to a key ID to display the Copy Key ID link.
2. Click Copy Key ID.

To get your key ID for your individual API key from App Store Connect, log in to App Store Connect and:

1. Go to your user profile.
2. Scroll down to Individual API Key.
3. Click Generate API Key.

> **Tip**
>
> If you have more than one team API key, use the key ID of the same private key that you use to sign the JWT.

Here's an example of a JWT header:

```
{
    "alg": "ES256",
    "kid": "2X9R4HXF34",
    "typ": "JWT"
}
```

Create the JWT Payload for Team Keys

The JWT payload contains information specific to the App Store Connect APIs, such as issuer ID and expiration time. Use the following fields and values in the JWT payload:

Payload Field	Value
`iss` - Issuer ID	Your issuer ID from the API Keys page in App Store Connect, for example, `57246542-96fe-1a63-e053-0824d011072a`
`iat` - Issued At Time	The token's creation time, in UNIX epoch time, for example, `1528407600`
`exp` - Expiration Time	The token's expiration time in Unix epoch time. Tokens that expire more than 20 minutes into the future are not valid except for resources listed in Determine the Appropriate Token Lifetime.
`aud` - Audience	`appstoreconnect-v1`
`scope` - Token Scope	A list of operations you want App Store Connect to allow for this token; for example, `GET /v1/apps/123` (Optional)

To get your issuer ID, log in to App Store Connect and:

1. Select Users and Access.

2. Select the Integrations tab.

The issuer ID appears near the top of the page. To copy the issuer ID, click Copy next to the ID.

Here's an example of a JWT payload:

```
{
    "iss": "57246542-96fe-1a63-e053-0824d011072a",
    "iat": 1528407600,
    "exp": 1528408800,
    "aud": "appstoreconnect-v1",
    "scope": [
        "GET /v1/apps?filter[platform]=IOS"
    ]
}
```

Create the JWT Payload for Individual Keys

The JWT payload for Individual keys contains information specific to the App Store Connect APIs, and expiration time. Use the following fields and values in the JWT payload:

Payload Field	Value
`sub` - Subject	`user` ``This value is always the same for individual keys.
`iat` - Issued At Time	The token's creation time, in UNIX epoch time, for example, `1528407600`
`exp` - Expiration Time	The token's expiration time in Unix epoch time. Tokens that expire more than 20 minutes in the future are not valid except for resources listed in Determine the Appropriate Token Lifetime.
`aud` - Audience	`appstoreconnect-v1`
`scope` - Token Scope	A list of operations you want App Store Connect to allow for this token, for example, `GET /v1/apps/123` (Optional)

> **Note**
>
> Individual keys don't use the Issuer ID key `iss`, but do require the Subject key `sub`.

Here's an example of a JWT payload:

```
{
    "sub": "user",
    "iat": 1528407600,
    "exp": 1528408800,
    "aud": "appstoreconnect-v1",
    "scope": [
        "GET /v1/apps?filter[platform]=IOS"
    ]
}
```

Determine the Scope of the Token

To reduce potential attack surface and improve security of your tokens, you can explicitly specify the scope of a token. The scope tells App Store Connect which requests it needs to accept for the token. If you unintentionally share a token with an unauthorized party, a limited scope reduces the requests that a potential attacker can make using the token.

The scope claim is an array of strings, each representing a request. Each scope entry includes:

- The HTTP `GET` method
- The URL path, for example, `/v1/apps` or `/v1/ciWorkflows/1234`
- The optional URL query string, for example, `?filter[platform]=IOS`

App Store Connect rejects a token with a scope claim if none of the scope entries match the attempted request.

> **Note**
>
> The order of query parameters isn't important. Additionally, App Store Connect ignores the following query parameters when it checks the scope: `limit`, `cursor`, and `sort`.

The following code listing shows an example of a JWT payload with a scope.

```
{
    "iss": "57246542-96fe-1a63-e053-0824d011072a",
    "iat": 1528407600,
    "exp": 1528408800,
    "aud": "appstoreconnect-v1",
    "scope": [
        "GET /v1/apps?filter[platform]=IOS"
    ]
}
```

With this payload, App Store Connect only allows you to fetch a list of iOS apps using the `List Apps` endpoint if you use the `filter[platform]=IOS` query parameter.

You can use a JWT without a `scope` for any request as long as the role of the API key allows it.

Determine the Appropriate Token Lifetime

For every request, App Store Connect calculates the valid time for a token, referred as the token's `lifetime`, by subtracting the `iat` claim from the `exp` claim. For increased security, carefully consider the lifetime of tokens you create and choose a lifetime that doesn't allow usage of the

token for longer than necessary. For example, an appropriate lifetime for a token you use for a one-off request is two minutes. In contrast, consider using a token with a lifetime of 20 minutes for a long-running process that makes many requests with the same token. Additionally, consider generating a new token periodically throughout the process, instead of issuing tokens with longer lifetimes.

For most requests, App Store Connect rejects a token with a lifetime greater than 20 minutes. However, it accepts long-lived tokens for some inherently safe requests if:

- The token defines a scope.
- The scope only includes GET requests.
- The resources in the scope allow long-lived tokens.

If a token meets all the above criteria, App Store Connect accepts a token with a lifetime of up to six months for the following resources:

- Build Actions
- Build Runs
- Git References
- Issues
- macOS Versions
- Products
- Providers
- Power and Performance Metrics and Logs
- Pull Requests
- Repositories
- Test Results
- Workflows
- Xcode Versions

Sign the JWT

Use the private key associated with the key ID you specified in the header to sign the token.

Regardless of the programming language you're using with the App Store Connect API, there are a variety of open source libraries available online for creating and signing JWT tokens. See JWT.io for more information.

> **Tip**
>
> You don't need to generate a new token for every API request. To get better performance from the App Store Connect API, reuse the same signed token for multiple requests until it expires.

Include the JWT in the Request's Authorization Header

Once you have a complete and signed token, provide the token in the request's authorization header as a bearer token.

The following example shows a `curl` command using a bearer token. Replace the text `[signed token]` with the value of the signed token itself.

```
curl -v -H 'Authorization: Bearer [signed token]'
"https://api.appstoreconnect.apple.com/v1/apps"
```

See Also

Essentials

- Creating API Keys for App Store Connect API

  Create API keys to sign JSON Web Tokens (JWTs) and authorize API requests.

- Revoking API Keys

  Revoke unused, lost, or compromised private keys.

- Identifying Rate Limits

  Recognize the rate limits that REST API responses provide and handle them in your code.

- Uploading Assets to App Store Connect

  Upload screenshots, app previews, attachments for App Review, and routing app coverage files to App Store Connect.

- App Store Connect API Release Notes

  Learn about new features and updates in the App Store Connect API.