Article

Verifying a postback

Ensure the validity of a postback you receive after an ad conversion by verifying its cryptographic signature.

Overview

Postbacks contain data that validate an ad conversion. You need to verify the postback signature to make sure that Apple signed it before you count the conversion. Follow these steps to validate the postback:

1. Receive the postback at a URL you configure.
2. Respond to the postback request when you receive it.
3. Validate the JSON Web Signature (JWS) by verifying its signature.
4. Count only unique postbacks with signatures you verify. Check the did-win parameter to learn whether the postback represents a winning attribution.

Receive the postback

Devices send postbacks to ad networks and developers within a defined timeframe after a conversion event occurs and the advertised app updates the conversion value at least once. Ad networks receive postbacks at the URL they provide when they register to use AdAttributionKit. Developers who opt in receive copies of winning postbacks at the URL they configure in the app's property list key, `AttributionCopyEndpoint`. For more information about setting up your postback URL, see Registering an ad network and Configuring an advertised app. For more information about the timeframe for receiving attribution, see Receiving ad attributions and postbacks. You may receive postbacks in three conversion windows. For more information, see Receiving postbacks in multiple conversion windows.

Verify Apple’s signature

Apple signs postbacks with a different key depending on the environment and whether the system creates the postbacks in an end-to-end flow or in developer settings.

For production flows, always use the production key to validate that Apple generated the postbacks. Ignore postbacks that don’t pass signature validation in production flows.

You can also validate postback signatures in development environments, whether you’re testing an end-to-end flow or generating postbacks directly from the developer settings.

Determine which NIST P-256 public key you need from the list below:

- To validate production postbacks:

  Key identifier: apple-cas-identifier/0

  Base64-encoded key: `MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEWdp8GPcGqmhgzEFj9Z2nSpQVddayaPe4FMzqM9wib1+aHaaIzoHoLN9zW4K8y4SPykE3YVK3sVqW6Af0lfx3gg==`

- To validate development postbacks the system creates from end-to-end flows:

  Key identifier: apple-development-identifier/0

  Base64-encoded key: `MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAELeEDzpJEP+/qRSE5hJVC1p1J0ssUnQGMzBBbvnACBok8OVGGLgxL0myrKiy6lvRtSlLRsWit87i+vftD8AEqeQ==`

- To validate development postbacks the system generates from developer settings:

  Key identifier: apple-development-identifier/1

  Base64-encoded key: `MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE8YzdO7eM97s/IJ25kdW5CZ3A14USE5IJ5Ha/vhWaxI6UBI1ZxCEvjrKxVluVGe6qWwF1BDFq+QHqKfH5u+wxHQ==`

The following code demonstrates how to determine which public key to use, depending on the `kid` property you receive in the postback’s JWS header:

```swift
import CryptoKit
import Foundation

enum PublicKeyError: Error {
    case unknownKeyID
    case invalidPublicKey
}
```

```
func getPublicKey(forKeyID keyID: String) throws -> P256.Signing.PublicKey {
    let publicKeyBase64: String
    switch keyID {
    case "apple-cas-identifier/0":
        publicKeyBase64 = "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEWdp8GPcGqmhgzEFj9Z2nS
    case "apple-development-identifier/0":
        publicKeyBase64 = "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAELeEDzpJEP+/qRSE5hJVC1
    case "apple-development-identifier/1":
        publicKeyBase64 = "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE8YzdO7eM97s/IJ25kdW5C
    default:
        throw PublicKeyError.unknownKeyID
    }

    guard let publicKeyData = Data(base64Encoded: publicKeyBase64) else {
        throw PublicKeyError.invalidPublicKey
    }
    return try P256.Signing.PublicKey(derRepresentation: publicKeyData)
```

To verify a signature, follow these steps:

1. Decode Apple's public key, which appears as a Base64-encoded string above. The result is a byte array in DER format.
2. Create an X.509 standard public key from the DER byte array.
3. Use the public key to verify the signature of the JWS using the outlined process in Section 5.2 of RFC 7515.

If your verification passes, the postback is valid.

Important

If the signature fails your verification, consider the postback message invalid. Don't use it to count a conversion.

Verify the JWS components

The JWS components include the following:

- A header that has `alg` and `kid` parameters.
- A payload that contains the `impression-type`, `ad-network-identifier`, `source-identifier`, `advertised-item-identifier`, `conversion-type`, `postback-`

`identifier`, `did-win`, and `postback-sequence-index` properties, as well as the `marketplace-identifier` and `publisher-item-identifier` properties, both of which are optional.

- The signature that Apple generates, which is the last component of the JWS.

Respond to the postback

When you receive a postback, respond with an HTTP 200 OK status code. If the device receives a 500 status code, it attempts to send the postback up to nine times over a maximum of 9 days.

Count unique messages only

Use the `postback-identifier` value to ensure that you're counting unique ad conversions. If you receive more than one postback with the same `postback-identifier`, discard the duplicate report.

See Also

Postback verification and parameter identification

Identifying the parameters in a postback

Interpret postback properties to understand the attribution report.