

Notifications Quickstart

[Suggest Edits](#)

This quickstart guide helps you quickly set up notifications for Programmable Wallets. Follow this to configure a subscriber endpoint that sends a notification every time the status of resource changes. This guide focuses on the challenge resource but applies to any resources available in Programmable Wallet Notifications.

1. Expose a Subscriber Endpoint

To receive notifications on changes in resource status, you must expose a publicly accessible subscriber endpoint on your side. The endpoint must handle both HEAD and POST requests over HTTPS. To quickly expose an endpoint for testing, we will use [webhook.site](#) in this guide, which allows you to easily inspect, test and automate any incoming HTTPS request or e-mail directly in a web browser. When visiting [webhook.site](#) for the first time, you should see a status message that looks like the following:

Your unique URL (Please copy it from here, not from the address bar!) <https://webhook.site/83fa21a0-f00a-4673-bb50-bcf62c78b1f7>

Record the value of the unique URL .

In the example above, the unique URL is: <https://webhook.site/83fa21a0-f00a-4673-bb50-bcf62c78b1f7> . Use the public-facing URL you receive as you progress throughout this guide.

2. Subscribe to Status Notifications

Now that you have a publicly accessible endpoint, you need to register your endpoint as a subscriber to webhook notifications by doing the following:

1. Click Webhooks
2. in your Circle Developer account.
3. Click Add a Webhook
4. in the upper right corner.
5. Enter your endpoint URL from above. It will be similar to the earlier example:
6. <https://webhook.site/83fa21a0-f00a-4673-bb50-bcf62c78b1f7>
7. .
8. Click Add Webhook
9. .
10. At this stage, you should receive a Hello World test notification indicating you have subscribed to your endpoint successfully:

```
JSON {
  "subscriptionId": "00000000-0000-0000-0000-000000000000",
  "notificationId": "00000000-0000-0000-0000-000000000000",
  "notificationType": "webhooks.test",
  "notification": {
    "hello": "world"
  },
  "timestamp": "2023-07-20T16:18:40.816010685Z",
  "version": 2
} Now you have a sample local environment ready to receive transaction and challenge status change notifications!
```

3. Initiate a Challenge Flow to Receive a Notification

To observe a notification message on the status of a challenge, initiate the creation of a wallet (for example, by following the [Programmable Wallets User-Controlled Wallets Tutorial](#) for creating a user-controlled wallet). Once you successfully initiate a challenge payment, you should see a notification message on your local server shell that looks like the following:

```
Sample Challenge Notification :

JSON { "subscriptionId": "a68cd974-d209-46cd-8dbf-b7a081fbd627", "notificationId": "eaa4a4fe-24b8-4329-a4c6-6dfd557dbcb2", "notificationType": "challenges.initialize", "notification": { "id": "9c0a1991-735a-5140-8996-7b95720c5e55", "userId": "2a054cd1-3124-4aa7-b1f0-86c4a4df995c", "type": "INITIALIZE", "status": "COMPLETE", "correlationIds": [ "01890792-a199-77bc-b005-b229f81824fa" ], "errorCode": 0, "errorMessage": "" }, "timestamp": "2023-06-29T14:33:17.785131449Z", "version": 2 }
```

4. Digital Signature Verification

1. Every webhook notification is digitally signed by an asymmetric key. In the headers of each webhook, you can find:
1. X-Circle-Signature
2. : a header containing the digital signature generated by Circle
3. X-Circle-Key-Id
4. : a header containing the UUID you need to retrieve the relevant public key

Headers

```
X-Circle-Key-Id: "879dc113-5ca4-4ff7-a6b7-54652083fcf8"
X-Circle-Signature: "MEYCIQCA9EvPbdEJiy7Cw0eY+KQZA/oFi5ZEInPs8CYpyaJexglhAKIRnDz9QRQmFKx8QFrvawp+8b9Bs2dQ03xD+XaWWDE" 1. Using the X-Circle-Key-Id value, query the following endpoint to retrieve the public key and algorithm used to sign the message:GET /v2/notifications/publicKey/{keyId}

cURL
```

Replace {YOUR_API_KEY} with your API key

Replace {PUBLIC_KEY_ID} with your public key id

```
curl --request GET --url 'https://api.circle.com/v2/notifications/publicKey/{PUBLIC_KEY_ID}' \
--header 'accept: application/json' \
--header 'authorization: Bearer ENV_API_KEY:ID:SECRET' \
If successful, you will receive a response with the following shape

Response Body {
  "data": {
    "id": "879dc113-5ca4-4ff7-a6b7-54652083fcf8",
    "algorithm": "ECDSA_SHA_256",
    "publicKey": "MFkwEwYHKoZIzj0DAQcDQgAEsI76SPZBjemW0mJNN4KTvYkLT8b0T4UGhFhzNk3Jfqt6iuPILQLq533FelXwczJbg2U1PHTvQTK7qOQnDL2Tg==",
    "createDate": "2023-06-28T21:47:35.107250Z"
  } } 1. Use the public key and algorithm to verify the integrity of the webhook's payload against the signature found in 2. X-Circle-Signature 3. .

Note: Please make sure the webhook notification is properly formatted JSON string prior to verifying it

Response Body { "subscriptionId": "890a8cae-46bd-40ad-bd7f-2e49b8dea9da", "notificationId": "0bdc3e4b-5070-4530-8ed4-2e2e4c9fd2f0", "notificationType": "webhooks.test", "notification": { "hello": "world" }, "timestamp": "2023-07-12T04: 02: 28.555562821Z", "version": 2 } The following code sample demonstrates how to verify the X-Circle-Signature :

Python Node.js
Java import base64 import json

from cryptography.exceptions import InvalidSignature
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric import ec
```

Load the public key from the base64 encoded string

```
public_key_base64 = "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEsI76SZPBjemW0mJNN4KTvYkLT8bOT4UGhFhzNk3fJqf6iuPILQLq533FelXwczJbjg2U1PHTvQTK7qOQnDL2Tg=="
public_key_bytes = base64.b64decode(public_key_base64) public_key = serialization.load_der_public_key(public_key_bytes)
```

Load the signature you want to verify

```
signature_base64 = "MEQCIBJlPX7t0FDOcozsRK6qlQwik5Fq6mhAtCSSgIB/yQO7AiB9U5lVpdufKvPhk3cz4TH2f5MP7ArnmPRBmhPztpsIFQ==" signature_bytes =
base64.b64decode(signature_base64)
```

Load and format the message you want to verify

```
message = "{\n\"subscriptionId\": \"00000000-0000-0000-0000-000000000000\", \"notificationId\": \"00000000-0000-0000-0000-000000000000\", \"notificationType\": \"webhooks.test\", \"notification\": {\n\"hello\": \"world\", \"timestamp\": \"2024-01-26T18:22:19.779834211Z\", \"version\": \"2\"} }"
json_data = json.loads(message) formatted_json = json.dumps(json_data, separators=(',', ':'))
```

Verify the signature

```
try: public_key.verify( signature_bytes, formatted_json.encode(), ec.ECDSA(hashes.SHA256()) ) print("Signature is valid.") except InvalidSignature: print("Signature is invalid.") const asn1 =
require('asn1.js'); const base64url = require('base64url'); const crypto = require('crypto'); const jwkToPem = require('jwk-to-pem');

// Define the EC public key ASN.1 syntax const EC_PUBLIC_KEY = asn1.define('ECPublicKey', function () { this.seq().obj( this.key('algorithm').seq().obj( this.key('id').objid()),
this.key('namedCurve').objid() ), this.key('publicKey').bitstr() ) });

// Load the public key from the base64 encoded string const publicKeyBase64 =
'MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEsI76SZPBjemW0mJNN4KTvYkLT8bOT4UGhFhzNk3fJqf6iuPILQLq533FelXwczJbjg2U1PHTvQTK7qOQnDL2Tg=='; const publicKeyDer =
base64url.toBuffer(publicKeyBase64); const publicKeyAsn1 = EC_PUBLIC_KEY.decode(publicKeyDer, 'der'); const publicKeyJwk = { kty: 'EC', crv: 'P-256', x: publicKeyAsn1.publicKey.data.slice(1,
33).toString('base64'), y: publicKeyAsn1.publicKey.data.slice(33).toString('base64') }; const publicKeyPem = jwkToPem(publicKeyJwk);

// Load the signature you want to verify const signatureBase64 = 'MEQCIBJlPX7t0FDOcozsRK6qlQwik5Fq6mhAtCSSgIB/yQO7AiB9U5lVpdufKvPhk3cz4TH2f5MP7ArnmPRBmhPztpsIFQ=='; const
signatureBytes = base64url.toBuffer(signatureBase64);

// Load and format the message you want to verify const message = '{\n\"subscriptionId\": \"00000000-0000-0000-0000-000000000000\", \"notificationId\": \"00000000-0000-0000-0000-000000000000\", \"notificationType\": \"webhooks.test\", \"notification\": {\n\"hello\": \"world\", \"timestamp\": \"2024-01-26T18:22:19.779834211Z\", \"version\": \"2\"} }"; const jsonData = JSON.parse(message); const
formattedJson = JSON.stringify(jsonData, null, 0);

// Verify the signature const verify = crypto.createVerify('SHA256'); verify.update(formattedJson); verify.end();

const isSignatureValid = verify.verify(publicKeyPem, signatureBytes); if (isSignatureValid) { console.log('Signature is valid.')} else { console.log('Signature is invalid.')} } package com.circle.webhook;

import java.nio.charset.StandardCharsets; import java.security.InvalidKeyException; package com.circle.webhook;

import com.fasterxml.jackson.core.JsonProcessingException; import com.fasterxml.jackson.databind.JsonNode; import com.fasterxml.jackson.databind.ObjectMapper;

import java.nio.charset.StandardCharsets; import java.security.InvalidKeyException; import java.security.KeyFactory; import java.security.NoSuchAlgorithmException; import java.security.PublicKey;
import java.security.Signature; import java.security.SignatureException; import java.security.spec.InvalidKeySpecException; import java.security.spec.X509EncodedKeySpec; import java.util.Base64;

public class CircleSignature { public static void main(String[] args) throws NoSuchAlgorithmException, InvalidKeySpecException, InvalidKeyException, SignatureException, JsonProcessingException {
// Load the public key from the base64 encoded string String publicKeyBase64 =
'MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEsI76SZPBjemW0mJNN4KTvYkLT8bOT4UGhFhzNk3fJqf6iuPILQLq533FelXwczJbjg2U1PHTvQTK7qOQnDL2Tg=='; byte[] publicKeyBytes =
Base64.getDecoder().decode(publicKeyBase64); X509EncodedKeySpec keySpec = new X509EncodedKeySpec(publicKeyBytes); KeyFactory keyFactory = KeyFactory.getInstance("EC"); PublicKey
publicKey = keyFactory.generatePublic(keySpec);

// Load the signature you want to verify
String signatureBase64 = "MEQCIBJlPX7t0FDOcozsRK6qlQwik5Fq6mhAtCSSgIB/yQO7AiB9U5lVpdufKvPhk3cz4TH2f5MP7ArnmPRBmhPztpsIFQ==";
byte[] signatureBytes = Base64.getDecoder().decode(signatureBase64);

// Load and format the message you want to verify
String message = "{\n\"subscriptionId\": \"00000000-0000-0000-0000-000000000000\", \"notificationId\": \"00000000-0000-0000-0000-000000000000\", \"notificationType\": \"webhooks.test\", \"notification\": {\n\"hello\": \"world\", \"timestamp\": \"2024-01-26T18:22:19.779834211Z\", \"version\": \"2\"} }";
ObjectMapper objectMapper = new ObjectMapper();
JsonNode jsonData = objectMapper.readTree(message);
String formattedJson = objectMapper.writeValueAsString(jsonData);

// Verify the signature
Signature signature = Signature.getInstance("SHA256withECDSA");
signature.initVerify(publicKey);
signature.update(formattedJson.getBytes(StandardCharsets.UTF_8));

boolean isSignatureValid = signature.verify(signatureBytes);
if (isSignatureValid) {
System.out.println("Signature is valid.");
} else {
System.out.println("Signature is invalid.");
}
}
}
```

5. IP Accesslist Check

To ensure the security of your integration, only trust Circle webhook notifications from the following IP addresses.

```
54.243.112.156
100.24.191.35
54.165.52.248
54.87.106.46 Updated10 days ago
```

What's Next Congratulations, you have received your first notification for a Programmable Wallets challenge! [*Notification Flows * Table of Contents *](#) [* 1. Expose a Subscriber Endpoint *](#) [* 2. Subscribe to Status Notifications *](#) [* 3. Initiate a Challenge Flow to Receive a Notification *](#) [* 4. Digital Signature Verification *](#) [* 5. IP Accesslist Check](#)