

SCITT  
Internet-Draft  
Intended status: Standards Track  
Expires: 14 November 2026

H. Birkholz  
Fraunhofer SIT  
J. Geater  
Bowball Technologies Ltd  
A. Delignat-Lavaud  
Microsoft Research  
13 May 2026

Supply Chain Integrity, Transparency, and Trust (SCITT) Reference APIs  
draft-ietf-scitt-scrapi-10

## Abstract

This document describes a REST API with the HTTP resources, request and response messages, and error handling needed for an interoperable implementation of a SCITT Transparency Service, as defined by the Supply Chain Integrity, Transparency, and Trust (SCITT) Architecture.

## About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at  
<https://datatracker.ietf.org/doc/draft-ietf-scitt-scrapi/>.

Discussion of this document takes place on the SCITT Working Group mailing list (<mailto:scitt@ietf.org>), which is archived at <https://mailarchive.ietf.org/arch/browse/scitt/>. Subscribe at <https://www.ietf.org/mailman/listinfo/scitt/>.

Source for this draft and an issue tracker can be found at  
<https://github.com/ietf-wg-scitt/draft-ietf-scitt-scrapi>.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 14 November 2026.

## Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Scope and Relation to the SCITT Architecture . . . . .	3
1.2. Terminology . . . . .	5
2. Resources . . . . .	5
2.1. Transparency Service Keys . . . . .	7
2.2. Individual Transparency Service Key . . . . .	8
2.3. Register Signed Statement . . . . .	10
2.3.1. Status 201 - Registration is successful . . . . .	12
2.3.2. Status 303 - Registration is running . . . . .	13
2.3.3. Status 400 - Invalid Client Request . . . . .	13
2.4. Query Registration Status . . . . .	14
2.4.1. Status 302 - Registration is running . . . . .	15
2.4.2. Status 200 - Asynchronous registration is successful . . . . .	15
2.4.3. Status 400 - Invalid Client Request . . . . .	17
2.4.4. Status 404 - Operation Not Found . . . . .	18
2.4.5. Status 429 - Too Many Requests . . . . .	19
2.5. Resolve Receipt . . . . .	19
2.5.1. Status 200 - OK . . . . .	19
2.5.2. Status 404 - Not Found . . . . .	20
3. Privacy Considerations . . . . .	21
4. Security Considerations . . . . .	21
4.1. General Scope . . . . .	21
4.2. Applicable Environment . . . . .	21
4.3. Authentication . . . . .	21
4.4. Threat Model . . . . .	21
4.4.1. In Scope . . . . .	21
4.4.2. Out of Scope . . . . .	23
5. Operational Considerations . . . . .	24
5.1. Client Retry Behavior . . . . .	24

5.2. Server-Side Retry Configuration . . . . .	24
5.3. Rate Limiting . . . . .	24
6. IANA Considerations . . . . .	24
6.1. Well-Known URI for Key Discovery . . . . .	25
6.1.1. Registration Template . . . . .	25
7. References . . . . .	25
7.1. Normative References . . . . .	25
7.2. Informative References . . . . .	27
Contributors . . . . .	27
Authors' Addresses . . . . .	28

## 1. Introduction

The Supply Chain Integrity, Transparency, and Trust (SCITT) Architecture [I-D.draft-ietf-scitt-architecture] defines the core objects, identifiers and workflows necessary to interact with a SCITT Transparency Service:

- \* Signed Statements
- \* Receipts
- \* Transparent Statements
- \* Registration Policies

SCITT Reference APIs (SCRAPI) defines HTTP resources for a Transparency Service using COSE ([RFC9052]). The following resources MUST be implemented for conformance to this specification:

- \* Registration of Signed Statements (Section 2.3, Section 2.4)
- \* Issuance and resolution of Receipts (Section 2.5)
- \* Discovery of Transparency Service Keys (Section 2.1, Section 2.2)

### 1.1. Scope and Relation to the SCITT Architecture

The SCITT Architecture [I-D.draft-ietf-scitt-architecture] specifies the conceptual roles, message structures, and workflows of a Transparency Service, but does not define a concrete protocol by which clients interact with that service. This document specifies one such concrete protocol: an HTTP-based REST API that realizes those interactions in an interoperable way. References in this specification to "normative requirements of the SCITT Architecture" are to the requirements expressed using BCP 14 keywords [RFC2119] [RFC8174] in [I-D.draft-ietf-scitt-architecture] that pertain to the externally observable behavior of a Transparency Service, such as the

registration of Signed Statements, the issuance and validation of Receipts, and the publication of the keys used to verify Receipts.

In particular, this document defines HTTP resources that satisfy the requirements in the following sections of [I-D.draft-ietf-scitt-architecture]:

- \* Registration of Signed Statements (Section 6.3 of [I-D.draft-ietf-scitt-architecture]), realized by the Signed Statement registration resources defined in Section 2.3, Section 2.4 and Section 2.5.
- \* Issuance of Receipts and construction of Transparent Statements (Section 7 of [I-D.draft-ietf-scitt-architecture]), realized by the Receipt resolution resource defined in Section 2.5.
- \* Discovery of the Transparency Service verification keys used by Verifiers to validate Receipts (Section 5.1.2 and Section 9.4 of [I-D.draft-ietf-scitt-architecture]), realized by the resource defined in Section 2.1.

The mandatory-to-implement resources listed above are sufficient for an interoperable Transparency Service.

The following aspects of [I-D.draft-ietf-scitt-architecture] are intentionally out of scope for this document and are not covered by this API:

- \* The internal structure and operation of the Transparency Service's Verifiable Data Structure (Section 5.1.3 of [I-D.draft-ietf-scitt-architecture]).
- \* The contents and evaluation of Registration Policies (Section 5.1.1 of [I-D.draft-ietf-scitt-architecture]); this document only defines how the outcome of a policy decision is communicated to clients.
- \* The format and semantics of Signed Statements, Receipts, and Transparent Statements themselves, which are defined in [I-D.draft-ietf-scitt-architecture] and the COSE specifications referenced therein.
- \* Transports other than HTTP, and bindings of these resources to other application protocols.

## 1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This specification uses the terms "Signed Statement", "Receipt", "Transparent Statement", "Artifact Repositories", "Transparency Service" and "Registration Policy" as defined in [I-D.draft-ietf-scitt-architecture].

This specification uses "payload" as defined in [RFC9052].

## 2. Resources

All messages are sent as HTTP GET or POST requests.

If the Transparency Service cannot process a client's request, it MUST return either:

1. an HTTP 3xx code, indicating to the client additional action they must take to complete the request, such as follow a redirection, or
2. an HTTP 4xx or 5xx status code, and the body MUST be a Concise Problem Details object (application/concise-problem-details+cbor) [RFC9290] containing:
  - \* title: A human-readable string identifying the error that prevented the Transparency Service from processing the request, ideally short and suitable for inclusion in log messages.
  - \* detail: A human-readable string describing the error in more depth, ideally with sufficient detail enabling the error to be rectified.

SCRAPI is not a CoAP API, but Constrained Problem Details objects [RFC9290] provide a useful encoding for problem details and avoid the need to mix CBOR and JSON in resource or client implementations.

NOTE: Examples use '\ ' line wrapping per [RFC8792]

Examples of errors may include:

```
{
  / title /          -1: \
    "Bad Signature Algorithm",
  / detail /         -2: \
    "Signing algorithm 'WalnutDSA' not supported"
}
```

Most error types are specific to the type of request and are defined in the respective subsections below. The one exception is the "malformed" error type, which indicates that the Transparency Service could not parse the client's request because it did not comply with this document:

```
{
  / title /          -1: \
    "Malformed request",
  / detail /         -2: \
    "The request could not be parsed"
}
```

Per the guidance in Section 4.6 of [RFC9205], the specific HTTP status codes shown in the examples throughout this document are illustrative. Status codes can be generated by generic HTTP components (caches, intermediaries, captive portals, gateways, etc.) that are not part of the Transparency Service, and the set of registered HTTP status codes can be extended over time. Clients MUST therefore be prepared to handle any HTTP status code by falling back to the generic class semantics (1xx, 2xx, 3xx, 4xx, or 5xx) of the response when a more specific code is not recognized, and MUST rely on the Concise Problem Details [RFC9290] object (when present) rather than the status code alone to determine the application-level cause of an error.

Clients SHOULD treat 5xx HTTP status code responses as transient failures and MAY retry the same request without modification at a later date.

Note that in the case of any error response, the Transparency Service MAY include a Retry-After header field per [RFC9110] in order to request a minimum time for the client to wait before retrying the request. In the absence of this header field, this document does not specify a minimum.

The following subsections specify the HTTP resources required for conformance, as listed in Section 1.

## 2.1. Transparency Service Keys

This resource, located at `/.well-known/scitt-keys` (registered in accordance with [RFC8615]; see Section 6.1), is used to discover the public keys that can be used by relying parties to verify Receipts issued by the Transparency Service.

Clients interact with this resource by issuing an HTTP GET request, and SHOULD include `Accept: application/cbor` in the request. The Transparency Service MUST respond with a COSE Key Set, as defined in Section 7 of [RFC9052], serialized as `application/cbor`.

Request:

```
GET /.well-known/scitt-keys HTTP/1.1
Host: transparency.example
Accept: application/cbor
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/cbor
```

Body (in CBOR diagnostic notation)

```
[
  {
    -1:1,
    -2:h'65eda5a1...9c08551d',
    -3:h'1e52ed75...0084d19c',
    1:2,
    2:'kid1'
  },
  {
    -1:1,
    -2:h'bac5b11c...d6a09eff',
    -3:h'20138bf8...bbfc117e',
    1:2,
    2:'kid2'
  }
]
```

The Transparency Service MAY stop returning at that resource the keys it no longer uses to issue Receipts, following a reasonable delay. A delay is considered reasonable if it is sufficient for relying parties to have obtained the key needed to verify any previously issued Receipt. Consistent with key management best practices described in [NIST.SP.800-57pt1r5] (Section 5.3.4, which

distinguishes the originator-usage period during which a private key is used to apply cryptographic protection from the recipient-usage period during which the corresponding public key is used to verify that protection), retired public keys used for signing SHOULD remain available for verification for as long as any Receipts signed with them may still need to be verified. Retaining retired keys has operational implications: the Transparency Service is responsible for storing those keys (and their associated metadata, such as kid values and validity periods) securely and continuously, and for serving them via the Individual Transparency Service Key resource (see Section 2.2) for the entire retention period. If retired public keys are not retained, Receipts issued under those keys can no longer be verified by relying parties using only the Transparency Service's published key material, which may break the verifiability of previously issued Receipts and disrupt downstream consumers that depend on long-term verification.

A Transparency Service MAY include the Expires header field, as defined in Section 5.3 of [RFC9111], in responses returned by this resource and by the Individual Transparency Service Key resource (Section 2.2) to indicate how long clients may cache the returned keys. A Transparency Service MAY use the Cache-Control header field with the max-age directive, as defined in Section 5.2.2.1 of [RFC9111], for the same purpose; when both are present, Cache-Control: max-age takes precedence per Section 4.2.1 of [RFC9111]. The cache lifetime indicated by these headers is a hint about server availability and does not constrain client retention. A relying party that holds a Receipt SHOULD retain the verification key for as long as it may need to verify that Receipt, independent of any cache lifetime indicated by the Transparency Service.

The presence of these headers does not constitute a guarantee of key availability. A Transparency Service may still need to retire a key before any indicated cache lifetime has elapsed, for example in response to suspected compromise or cryptographic algorithm deprecation. In such cases, a relying party that holds a Receipt signed with a retired key can request a fresh Receipt for the same Signed Statement at the same position in the Verifiable Data Structure, signed with a current key.

## 2.2. Individual Transparency Service Key

This sub-resource, located at /.well-known/scitt-keys/{kid\_value}, is used to resolve a single public key, from a kid value contained in a Receipt previously issued by the Transparency Service.



Clients interact with this sub-resource by issuing an HTTP GET request, and SHOULD include Accept: application/cbor in the request. The Transparency Service MUST respond with a single COSE Key, as defined in Section 7 of [RFC9052], serialized as application/cbor, or a 404 status if no matching key is found.

Request:

```
GET /.well-known/scitt-keys/{kid_value} HTTP/1.1
Host: transparency.example
Accept: application/cbor
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/cbor
```

Body (in CBOR diagnostic notation)

```
{
  -1:1,
  -2:h'bac5b11c...d6a09eff',
  -3:h'20138bf8...bbfc117e',
  1:2,
  2:'kid_value'
}
```

The following expected error is defined for the condition described below. When this condition is encountered, an implementation MUST return an error response that is a valid [RFC9290] object. Implementations SHOULD use the error defined below, but MAY return another valid [RFC9290] error instead:

```
HTTP/1.1 404 Not Found
Content-Type: application/concise-problem-details+cbor
```

```
{
  / title /           -1: "No such key",
  / detail /          -2: "No key could be found for this kid value"
}
```

If the kid values used by the service ({kid\_value} in the request above) are not URL-safe, the resource MUST accept the base64url encoding of the kid value, without padding, in the URL instead.

Section 2 of [RFC7515] specifies Base64Url encoding as follows:

"Base64 encoding using the URL- and filename-safe character set defined in Section 5 of RFC 4648 [RFC4648], with all trailing '=' characters omitted and without the inclusion of any line breaks, whitespace, or other additional characters. Note that the base64url encoding of the empty octet sequence is the empty string. (See Appendix C of [RFC7515] for notes on implementing base64url encoding without padding.)"

It is RECOMMENDED to use COSE Key Thumbprint, as defined in [RFC9679] as the mechanism to assign a kid to Transparency Service keys. [RFC9679] provides a well-specified, canonical method to deterministically derive a unique kid value directly from the COSE Key itself. Using this mechanism offers several benefits to implementers:

- \* it ensures that the kid is uniquely and reproducibly bound to the key material,
- \* it removes the need for an out-of-band identifier assignment process,
- \* it enables independent parties to compute and verify the same kid for a given key, which simplifies key discovery and reduces the risk of kid collisions across Transparency Services.

### 2.3. Register Signed Statement

This resource instructs a Transparency Service to register a Signed Statement on its log. Since log implementations may take many seconds or longer to reach finality, this API provides an asynchronous mode that returns a locator that can be used to check the registration's status asynchronously.

The following is a non-normative example of an HTTP request to register a Signed Statement:

Request:

```

POST /entries HTTP/1.1
Host: transparency.example
Accept: application/cbor
Accept: application/cose
Content-Type: application/cose

```

Body (in CBOR diagnostic notation)

```

18([ / COSE Sign1
  <<{
    / signature alg / 1: -35, # ES384
    / key identifier / 4: h'75726e3a...32636573',
    / cose sign1 type / 16: "application/example+cose",
    / payload-hash-alg / 258: -16, # sha-256
    / preimage-content-type / 259: "application/spdx+json",
    / payload-location / 260: "https://.../manifest.json",
    / CWT Claims / 15: {
      / Issuer / 1: "vendor.example",
      / Subject / 2: "vendor.product.example",
    }
  }>>, / Protected Header /
  {}, / Unprotected Header /
  / Payload, sha-256 digest of file stored at payload-location /
  h'935b5a91...e18a588a',
  h'269cd68f4211dffc...0dcb29c' / Signature /
])

```

A Transparency Service depends on the verification of the Signed Statement in the Registration Policy.

The Registration Policy for the Transparency Service MUST be applied before any additional processing. The details of Registration Policies are out of scope for this document.

Signed Statements MAY use detached payloads, as described in [I-D.draft-ietf-scitt-architecture]. When a Signed Statement is submitted with a detached payload, the Transparency Service still requires access to the payload content in order to verify the signature as part of applying the Registration Policy. The mechanism by which the payload is made available to the Transparency Service is implementation-specific.

Response:

One of the following:

### 2.3.1. Status 201 - Registration is successful

If the Transparency Service is able to produce a Receipt within a reasonable time, it MAY return it directly.

Along with the receipt the Transparency Service MAY return a locator in the HTTP response Location header, provided the locator is a valid URL.

HTTP/1.1 201 Created

Location: https://transparency.example/entries/67ed...befe

Content-Type: application/cose

Body (in CBOR diagnostic notation)

```
/ cose-sign1 / 18([
  / protected / <<{
    / key / 4 : "mxA4KiOkQFZ-dkLebSo3mLOEPR7rN8XtxkJe45xuyJk",
    / algorithm / 1 : -7, # ES256
    / vds / 395 : 1, # RFC9162 SHA-256
    / claims / 15 : {
      / issuer / 1 : "https://blue.notary.example",
      / subject / 2 : "https://green.software.example/cli@v1.2.3",
    },
  }>>,
  / unprotected / {
    / proofs / 396 : {
      / inclusion / -1 : [
        <<[
          / size / 9, / leaf / 8,
          / inclusion path /
          h'7558a95f...e02e35d6'
        ]>>
      ],
    },
  },
  / payload / null,
  / signature / h'02d227ed...ccd3774f'
])
```

The response contains the Receipt for the Signed Statement. Fresh Receipts may be requested through the resource identified in the Location header.

### 2.3.2. Status 303 - Registration is running

In cases where the registration request is accepted but the Transparency Service is not able to produce a Receipt in a reasonable time, it MAY return a locator for the registration operation, as in this non-normative example:

```
HTTP/1.1 303 See Other
Location: https://transparency.example/entries/67ed...befe
Content-Type: application/cose
Content-Length: 0
Retry-After: <seconds>
```

The location MAY be temporary, and the service may not serve a relevant response at this Location after a reasonable delay.

The Transparency Service MAY include a Retry-After header in the HTTP response to help with polling.

### 2.3.3. Status 400 - Invalid Client Request

The following expected errors are defined for the conditions described below. When such a condition is encountered, an implementation MUST return an error response that is a valid [RFC9290] object. Implementations SHOULD use the corresponding error defined below, but MAY return another valid [RFC9290] error instead.

```
HTTP/1.1 400 Bad Request
Content-Type: application/concise-problem-details+cbor
```

```
{
  / title /          -1: \
                        "Bad Signature Algorithm",
  / detail /          -2: \
                        "Signed Statement contained a non supported algorithm"
}
```

```
HTTP/1.1 400 Bad Request
Content-Type: application/concise-problem-details+cbor
```

```
{
  / title /          -1: "\
                        Confirmation Missing",
  / detail /          -2: \
                        "Signed Statement did not contain proof of possession"
}
```

HTTP/1.1 400 Bad Request  
Content-Type: application/concise-problem-details+cbor

```
{
  / title /          -1: \
    "Payload Missing",
  / detail /          -2: \
    "Signed Statement payload must be present"
}
```

HTTP/1.1 400 Bad Request  
Content-Type: application/concise-problem-details+cbor

```
{
  / title /          -1: \
    "Rejected",
  / detail /          -2: \
    "Signed Statement not accepted by the current\
    Registration Policy"
}
```

HTTP/1.1 400 Bad Request  
Content-Type: application/concise-problem-details+cbor

```
{
  / title /          -1: "Invalid locator",
  / detail /          -2: "Operation locator is not in a valid form"
}
```

#### 2.4. Query Registration Status

This resource lets a client query a Transparency Service for the registration status of a Signed Statement they have submitted earlier, and for which they have received a 303 or 302 - Registration is running response.

Request:

```
GET /entries/67ed...befe HTTP/1.1
Host: transparency.example
Accept: application/cbor
Accept: application/cose
Content-Type: application/cose
```

Response:

One of the following:

#### 2.4.1. Status 302 - Registration is running

Registration requests may fail, in which case the Location MAY return an error when queried.

If the client requests (GET) the location when the registration is still in progress, the TS MAY return a 302 Found, as in this non-normative example:

```
HTTP/1.1 302 Found
Location: https://transparency.example/entries/67ed...befe
Content-Type: application/cose
Content-Length: 0
Retry-After: <seconds>
```

The location MAY be temporary, and the service may not serve a relevant response at this Location after a reasonable delay.

The Transparency Service MAY include a Retry-After header in the HTTP response to help with polling.

#### 2.4.2. Status 200 - Asynchronous registration is successful

Along with the receipt the Transparency Service MAY return a locator in the HTTP response Location header, provided the locator is a valid URL.

HTTP/1.1 200 OK  
Location: https://transparency.example/entries/67ed...befe  
Content-Type: application/cose

Body (in CBOR diagnostic notation)

```
/ cose-sign1 / 18([  
  / protected / <<{  
    / key / 4 : "mxA4KiOkQFZ-dkLebSo3mLOEPR7rN8XtxkJJe45xuyJk",  
    / algorithm / 1 : -7, # ES256  
    / vds / 395 : 1, # RFC9162 SHA-256  
    / claims / 15 : {  
      / issuer / 1 : "https://blue.notary.example",  
      / subject / 2 : "https://green.software.example/cli@v1.2.3",  
    },  
  }>>,  
  / unprotected / {  
    / proofs / 396 : {  
      / inclusion / -1 : [  
        <<[  
          / size / 9, / leaf / 8,  
          / inclusion path /  
            h'7558a95f...e02e35d6'  
        ]>>  
      ],  
    },  
  },  
  / payload / null,  
  / signature / h'02d227ed...ccd3774f'  
)
```

The response contains the Receipt for the Signed Statement. Fresh Receipts may be requested through the resource identified in the Location header.

As an example, a successful asynchronous follows the following sequence:



Initial exchange:

```
Client --- POST /entries (Signed Statement) --> TS
Client <-- 303 Location: .../entries/tmp123 --- TS
```

May happen zero or more times:

```
Client --- GET .../entries/tmp123 --> TS
Client <-- 302 Location: .../entries/tmp123 --- TS
```

Finally:

```
Client --- GET .../entries/tmp123 --> TS
Client <-- 200 (Receipt) --- TS
          Location: .../entries/final123
```

#### 2.4.3. Status 400 - Invalid Client Request

The following expected errors are defined for the conditions described below. When such a condition is encountered, an implementation MUST return an error response that is a valid [RFC9290] object. Implementations SHOULD use the corresponding error defined below, but MAY return another valid [RFC9290] error instead.

HTTP/1.1 400 Bad Request  
Content-Type: application/concise-problem-details+cbor

```
{
  / title /      -1: \
    "Bad Signature Algorithm",
  / detail /      -2: \
    "Signed Statement contained a non supported algorithm"
}
```

HTTP/1.1 400 Bad Request  
Content-Type: application/concise-problem-details+cbor

```
{
  / title /      -1: "\
    Confirmation Missing",
  / detail /      -2: \
    "Signed Statement did not contain proof of possession"
}
```

HTTP/1.1 400 Bad Request  
Content-Type: application/concise-problem-details+cbor

```
{
  / title /          -1: \
                      "Payload Missing",
  / detail /          -2: \
                      "Signed Statement payload must be present"
}
```

HTTP/1.1 400 Bad Request  
Content-Type: application/concise-problem-details+cbor

```
{
  / title /          -1: \
                      "Rejected",
  / detail /          -2: \
                      "Signed Statement not accepted by the current\
Registration Policy"
}
```

HTTP/1.1 400 Bad Request  
Content-Type: application/concise-problem-details+cbor

```
{
  / title /          -1: "Invalid locator",
  / detail /          -2: "Operation locator is not in a valid form"
}
```

#### 2.4.4. Status 404 - Operation Not Found

If no record of the specified running operation is found, the Transparency Service SHOULD respond with a 4xx-class status code (typically 404 Not Found) and a Concise Problem Details [RFC9290] object as in the following example:

HTTP/1.1 404 Not Found  
Content-Type: application/concise-problem-details+cbor

```
{
  / title /          -1: \
                      "Operation Not Found",
  / detail /          -2: \
                      "No running operation was found matching the requested ID"
}
```

#### 2.4.5. Status 429 - Too Many Requests

If a client is polling for an in-progress registration too frequently then the Transparency Service MAY, in addition to implementing rate limiting, return a 429 response:

HTTP/1.1 429 Too Many Requests  
Content-Type: application/concise-problem-details+cbor  
Retry-After: <seconds>

```
{
  / title /          -1: \
    "Too Many Requests",
  / detail /         -2: \
    "Only <number> requests per <period> are allowed."
}
```

#### 2.5. Resolve Receipt

Request:

GET /entries/67ed41f1de6a...cfc158694ed0befe HTTP/1.1  
Host: transparency.example  
Accept: application/cose

Response:

##### 2.5.1. Status 200 - OK

If the Receipt is found:

HTTP/1.1 200 OK  
 Location: https://transparency.example/entries/67ed...befe  
 Content-Type: application/cose

Body (in CBOR diagnostic notation)

```
/ cose-sign1 / 18([
  / protected / <<{
    / key / 4 : "mxA4KiOkQFZ-dkLebSo3mLOEPR7rN8XtxkJJe45xuyJk",
    / algorithm / 1 : -7, # ES256
    / vds / 395 : 1, # RFC9162 SHA-256
    / claims / 15 : {
      / issuer / 1 : "https://blue.notary.example",
      / subject / 2 : "https://green.software.example/cli@v1.2.3",
    },
  }>>,
  / unprotected / {
    / proofs / 396 : {
      / inclusion / -1 : [
        <<[
          / size / 9, / leaf / 8,
          / inclusion path /
            h'7558a95f...e02e35d6'
        ]>>
      ],
    },
  },
  / payload / null,
  / signature / h'02d227ed...ccd3774f'
])
```

#### 2.5.2. Status 404 - Not Found

If there is no Receipt found for the specified EntryID the Transparency Service SHOULD respond with a 4xx-class status code (typically 404 Not Found) and a Concise Problem Details [RFC9290] object as in the following example:

HTTP/1.1 404 Not Found  
 Content-Type: application/concise-problem-details+cbor

```
{
  / title / -1: \
    "Not Found",
  / detail / -2: \
    "Receipt with entry ID <id> not known \
    to this Transparency Service"
}
```

### 3. Privacy Considerations

The privacy considerations section of [I-D.draft-ietf-scitt-architecture] applies to this document.

### 4. Security Considerations

#### 4.1. General Scope

This document describes the interoperable API for client calls to, and implementations of, a Transparency Service as specified in [I-D.draft-ietf-scitt-architecture]. As such the security considerations in this section are concerned only with security considerations that are relevant at that implementation layer. All questions of security of the related COSE formats, algorithm choices, cryptographic envelopes, verifiable data structures and the like are handled elsewhere and out of scope for this document.

#### 4.2. Applicable Environment

SCITT is concerned with issues of cross-boundary supply-chain-wide data integrity and as such must assume a very wide range of deployment environments. Thus, no assumptions can be made about the security of the computing environment in which any client implementation of this specification runs.

#### 4.3. Authentication

Authentication is out of scope for this document. Implementations MAY authenticate clients, for example for the purposes of authorization or preventing denial of service attacks. If Authentication is not implemented, rate limiting or other denial of service mitigations MUST be implemented.

#### 4.4. Threat Model

##### 4.4.1. In Scope

The most serious threats to implementations on Transparency Services are ones that would cause the failure of their main promises, to wit:

- \* Threats to strong identification, for example representing the Statements from one issuer as those of another
- \* Threats to payload integrity, for example changing the contents of a Signed Statement before making it transparent

- \* Threats to non-equivocation, for example attacks that would enable the presentation or verification of divergent proofs for the same Statement payload

#### 4.4.1.1. Denial of Service Attacks

While denial of service attacks are very hard to defend against completely, and Transparency Services are unlikely to be in the critical path of any safety-labile operation, any attack which could cause the `_silent_` failure of Signed Statement registration, for example, should be considered in scope.

The impact of DoS attacks can be detected by a client checking that the Transparency Service has registered any submitted Signed Statement and returned a Receipt. Since verification of Receipts does not require the involvement of the Transparency Service, a DoS attack cannot cause the silent loss of a registration. However, this relies on clients actively checking for Receipts and does not prevent the disruption itself.

Clients to Transparency Services SHOULD ensure that Receipts are available for their registered Statements, either on a periodic or needs-must basis, depending on the use case.

Beyond this, implementers of Transparency Services MUST follow general good practice around defending against network attacks such as flooding, including defenses such as rate limiting.

#### 4.4.1.2. Eavesdropping

Since the purpose of this API is to ultimately put the message payloads on a Transparency Log there is limited risk to eavesdropping. Nonetheless transparency may mean 'within a limited community' rather than 'in full public', so implementers MUST add protections against man-in-the-middle and network eavesdropping, such as TLS.

#### 4.4.1.3. Message Modification Attacks

Modification attacks are mitigated by the use of the Issuer signature on the Signed Statement.

#### 4.4.1.4. Message Insertion Attacks

Insertion attacks are mitigated by the use of the Issuer signature on the Signed Statement, therefore care must be taken in the protection of Issuer keys and credentials to avoid theft and impersonation.

#### 4.4.2. Out of Scope

##### 4.4.2.1. Replay Attacks

Replay attacks are not particularly concerning for SCITT or SCRAPI: Once a statement is made, it is intended to be immutable and non-repudiable, so making it twice should not lead to any particular issues. There could be issues at the payload level (for instance, the statement "it is raining" may be true when first submitted but not when replayed), but being payload-agnostic implementations of SCITT services cannot be required to worry about that.

If the semantic content of the payload are time-dependent and susceptible to replay attacks in this way then timestamps MUST be added to the protected header signed by the Issuer. The iat claim in a CWT\_Claims header parameter ([RFC9597]) MUST be used when the Issuer provides the timestamp themselves. The COSE header parameters defined in [RFC9921] for including [RFC3161] timestamp tokens or a similar mechanic, for example an Epoch Marker [I-D.ietf-rats-epoch-markers] Claim in the 'CWT\_Claims' header parameter, SHOULD be used, where a timestamp from a third party is required. Other mechanisms for including timestamps in the protected header MAY also be used.

##### 4.4.2.2. Message Deletion Attacks

Once registered with a Transparency Service, Registered Signed Statements cannot be deleted. Thus, any message deletion attack must occur prior to registration else it is indistinguishable from a man-in-the-middle or denial-of-service attack on this interface.

##### 4.4.2.3. Use of Unauthenticated HTTP Metadata

Implementations that serve multiple application profiles MAY use unauthenticated HTTP-layer signals, such as request headers or distinct registration endpoints, to route incoming Signed Statements to profile-specific processing.

However, these signals are not signed, are not committed to the Verifiable Data Structure, and cannot be replayed by Auditors.

Implementations MUST NOT use unauthenticated signals as authoritative inputs to the registration decision.

Implementations that use such signals for early dispatch MUST ensure that any processing decisions that affect the outcome of registration are fully determined by authenticated inputs, or are otherwise captured in the Verifiable Data Structure, such that the registration process remains deterministic and replayable by Auditors.

The authoritative identification of the application profile is carried within the protected header or payload of the Signed Statement, and MUST be verified after signature authentication.

## 5. Operational Considerations

### 5.1. Client Retry Behavior

Aggressive client retry or polling behavior can significantly impact a Transparency Service, increasing load and, in extreme cases, amplifying transient failures into sustained outages.

Clients SHOULD honor any Retry-After header field (defined in {Section 7.1.3 of RFC7231}) returned by the Transparency Service, treating it as a minimum interval before retrying. In its absence, clients SHOULD apply exponential backoff with jitter, cap the total number of retries, and avoid synchronizing retries across clients.

### 5.2. Server-Side Retry Configuration

Operators SHOULD configure a minimum retry interval appropriate for the expected registration latency and service capacity, and SHOULD communicate it to clients via the Retry-After header on relevant responses (e.g., 202, 429, 503). The interval should account for worst-case registration time, sustainable request volume, and intermediary behavior.

### 5.3. Rate Limiting

As noted in Section 4.3 and Section 4.4.1.1, rate limiting or other denial of service mitigations are required. The specific per-client policy is implementation dependent and typically varies with whether and how clients are authenticated (e.g., per-identity for authenticated clients versus per source IP for unauthenticated clients), the cost of the operation, and the deployment environment.

When a client exceeds the configured rate limit, the Transparency Service SHOULD return a 429 response (see Section 2.4.5) including a Retry-After header field.

## 6. IANA Considerations



## 6.1. Well-Known URI for Key Discovery

IANA is requested to register the /.well-known/scitt-keys URI in the "Well-Known URIs" registry defined in [RFC8615]. The normative behavior of this resource and its `{kid_value}` sub-resource is specified in Section 2.1 and Section 2.2.

### 6.1.1. Registration Template

The following value is requested to be registered in the "Well-Known URIs" registry (using the template from [RFC8615]):

URI suffix: scitt-keys Change controller: IETF Specification  
document(s): RFCthis Status: Permanent Related information:  
[I-D.draft-ietf-scitt-architecture]

## 7. References

### 7.1. Normative References

[I-D.draft-ietf-scitt-architecture]

Birkholz, H., Delignat-Lavaud, A., Fournet, C., Deshpande, Y., and S. Lasker, "An Architecture for Trustworthy and Transparent Digital Supply Chains", Work in Progress, Internet-Draft, draft-ietf-scitt-architecture-22, 10 October 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-scitt-architecture-22>>.

[NIST.SP.800-57pt1r5]

Barker, E., "Recommendation for Key Management: Part 1 - General", NIST Special Publication 800-57 Part 1 Revision 5, May 2020, <<https://doi.org/10.6028/NIST.SP.800-57pt1r5>>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.

[RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, DOI 10.17487/RFC3161, August 2001, <<https://www.rfc-editor.org/rfc/rfc3161>>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/rfc/rfc4648>>.

- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/rfc/rfc7515>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.
- [RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <<https://www.rfc-editor.org/rfc/rfc8615>>.
- [RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <<https://www.rfc-editor.org/rfc/rfc8792>>.
- [RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <<https://www.rfc-editor.org/rfc/rfc9052>>.
- [RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <<https://www.rfc-editor.org/rfc/rfc9110>>.
- [RFC9111] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <<https://www.rfc-editor.org/rfc/rfc9111>>.
- [RFC9205] Nottingham, M., "Building Protocols with HTTP", BCP 56, RFC 9205, DOI 10.17487/RFC9205, June 2022, <<https://www.rfc-editor.org/rfc/rfc9205>>.
- [RFC9290] Fossati, T. and C. Bormann, "Concise Problem Details for Constrained Application Protocol (CoAP) APIs", RFC 9290, DOI 10.17487/RFC9290, October 2022, <<https://www.rfc-editor.org/rfc/rfc9290>>.
- [RFC9597] Looker, T. and M.B. Jones, "CBOR Web Token (CWT) Claims in COSE Headers", RFC 9597, DOI 10.17487/RFC9597, June 2024, <<https://www.rfc-editor.org/rfc/rfc9597>>.

- [RFC9679] Isobe, K., Tschofenig, H., and O. Steele, "CBOR Object Signing and Encryption (COSE) Key Thumbprint", RFC 9679, DOI 10.17487/RFC9679, December 2024, <<https://www.rfc-editor.org/rfc/rfc9679>>.
- [RFC9921] Birkholz, H., Fossati, T., and M. Riechert, "CBOR Object Signing and Encryption (COSE) Header Parameter for Timestamp Tokens as Defined in RFC 3161", RFC 9921, DOI 10.17487/RFC9921, February 2026, <<https://www.rfc-editor.org/rfc/rfc9921>>.

## 7.2. Informative References

- [I-D.ietf-rats-epoch-markers]  
Birkholz, H., Fossati, T., Pan, W., Mihalcea, I., and C. Bormann, "Epoch Markers", Work in Progress, Internet-Draft, draft-ietf-rats-epoch-markers-03, 2 March 2026, <<https://datatracker.ietf.org/doc/html/draft-ietf-rats-epoch-markers-03>>.

## Contributors

Orie Steele  
Transmute  
United States  
Email: [orie@transmute.industries](mailto:orie@transmute.industries)

Orie contributed examples, text, and URN structure to early version of this draft.

Amaury Chamayou  
Microsoft  
United Kingdom  
Email: [amaury.chamayou@microsoft.com](mailto:amaury.chamayou@microsoft.com)

Amaury contributed crucial content to ensure interoperability between implementations, improve example expressiveness and consistency, as well as overall document quality.

Dick Brooks  
Business Cyber Guardian  
United States  
Email: [dick@businesscyberguardian.com](mailto:dick@businesscyberguardian.com)

Dick contributed use cases and helped improve example expressiveness and consistency.

Robert Martin  
MITRE Corporation  
United States  
Email: ramartin@mitre.org

Bob contributed use cases and helped with authoring and improving the document.

Steve Lasker  
Email: stevenlasker@hotmail.com

Steve contributed architectural insights, particularly around asynchronous operations and participated in the initial writing of the document.

Nicole Bates  
Microsoft  
United States  
Email: nicolebates@microsoft.com

Nicole contributed reviews and edits that improved the quality of the text.

Roy Williams  
United States of America  
Email: roywill@msn.com

Roy contributed the receipt refresh use case and associated resource definition.

#### Authors' Addresses

Henk Birkholz  
Fraunhofer SIT  
Rheinstrasse 75  
64295 Darmstadt  
Germany  
Email: henk.birkholz@ietf.contact

Jon Geater  
Bowball Technologies Ltd  
United Kingdom  
Email: jonathan@bowball-tech.com

Antoine Delignat-Lavaud  
Microsoft Research  
21 Station Road  
Cambridge  
CB1 2FB  
United Kingdom  
Email: antdl@microsoft.com