SCITT
Internet-Draft
Intended status: Standards Track
Expires: 2 July 2026

H. Birkholz
Fraunhofer SIT
J. Geater
DataTrails Inc.

29 December 2025

SCITT Reference APIs
draft-ietf-scitt-scrapi-06

Abstract

This document describes a REST API that supports the normative requirements of the SCITT Architecture. Optional key discovery and query interfaces are provided to support interoperability with X.509 Certificates, alternative methods commonly used to support public key discovery and Artifact Repositories.

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 will expire on 2 July 2026.

Table of Contents

1. Introduction

The 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

SCRAPI defines the operations necessary to support supply chain transparency using COSE [RFC9052]:

* Issuance of Signed Statements
* Registration of Signed Statements
* Verification of Signed Statements
* Issuance of Receipts
* Verification of Receipts
* Production of Transparent Statements
* Verification of Transparent Statements

In addition to these operational HTTP endpoints, this specification defines supporting endpoints:

* Resolving Verification Keys for Issuers
* Retrieving Receipts Asynchronously
* Retrieving Signed Statements from an Artifact Repository
* Retrieving Statements from an Artifact Repository
* Exchanging Receipts for refreshed Receipts

1.1. 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. Endpoints

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 mitigation MUST be implemented.

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 SHOULD 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 endpoint 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:

Error code: 'malformed' (The request could not be parsed)

Clients SHOULD treat 500 and 503 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.

2.1. Mandatory

The following HTTP endpoints are mandatory to implement to enable conformance to this specification.

2.1.1. Transparency Configuration

This endpoint is used to discover the capabilities and current configuration of a Transparency Service implementing this specification.

The Transparency Service responds with a CBOR map of configuration elements. These elements are Transparency-Service specific.

Contents of bodies are informative examples only.

Request:

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

Response:

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

Body (in CBOR diagnostic notation)

{
  "issuer": "https://transparency.example"
}
```

Responses to this message are vendor-specific, and out of the scope of this document.

2.1.2. Transparency Service Keys

This endpoint is used to discover the public keys that can be used by relying parties to verify Receipts issued by the Transparency Service.

The Transparency Service responds with a COSE Key Set, as defined in Section 7 of [RFC9052].

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'65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de108de439c08551d',
    -3:h'1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9eecd0084d19c',
    1:2,
    2:'kid1'
  },
  {
    -1:1,
    -2:h'bac5b11cad8f99f9c72b05cf4b9e26d244dc189f745228255a219a86d6a09eff',
    -3:h'20138bf82dc1b6d562be0fa54ab7804a3a64b6d72ccfed6b6fb6ed28bbfc117e',
    1:2,
    2:'kid2'
  }
]
```

2.1.3. Register Signed Statement

This endpoint 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
 /
     h'935b5a91...e18a588a',     / Payload, sha-256 digest of file stored at Location
 /
     h'269cd68f4211dffc...0dcb29c' / Signature
 /
   ])
```

A Transparency Service depends on both the client's authentication context (if present) and 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.

Response:

One of the following:

2.1.3.1. 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.

2.1.3.2. 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.

2.1.3.3. Status 400 - Invalid Client Request

The following expected errors are defined. Implementations MAY return other errors, so long as they are valid [RFC9290] objects.

```
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"
}
```

2.1.4. Query Registration Status

This endpoint 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:

2.1.4.1. 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.

2.1.4.2. 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-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.

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 (Transparent Statement)      --- TS
           Location: .../entries/final123
```

2.1.4.3. Status 400 - Invalid Client Request

The following expected errors are defined. Implementations MAY return other errors, so long as they are valid [RFC9290] objects.

```
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"
}
```

2.1.4.4. Status 404 - Operation Not Found

If no record of the specified running operation is found, the Transparency Service returns a 404 response.

```
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"
}
```

2.1.4.5. 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."
}
```

2.1.5. Resolve Receipt

Authentication SHOULD be implemented for this endpoint.

Request:

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

Response:

2.1.5.1. 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-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'
])
```

2.1.5.2. Status 404 - Not Found

If there is no Receipt found for the specified EntryID the Transparency Service returns a 404 response:

```
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"
}
```

2.2. Optional Endpoints

2.2.1. Exchange Receipt

This endpoint is used to exchange old or expiring Receipts for fresh ones.

The iat, exp and kid claims can change each time a Receipt is exchanged.

This means that fresh Receipts can have more recent issued at times, further in the future expiration times, and be signed with new signature algorithms.

Authentication SHOULD be implemented for this endpoint.

Request:

```
POST receipt-exchange HTTP/1.1
Host: transparency.example
Accept: application/cose
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.example",
      / subject / 2 : "https://green.example/cli@v1.2.3",
      / iat / 6: 1443944944 # Pre-refresh
    },
  }>>,
  / unprotected / {
    / proofs / 396 : {
      / inclusion / -1 : [
        <<[
          / size / 9, / leaf / 8,
          / inclusion path /
          h'7558a95f...e02e35d6'
        ]>>
      ],
    },
  },
  / payload     / null,
  / signature   / h'02d227ed...ccd3774f'
])

Response:
```

2.2.1.1. Status 200 - OK

If a new Receipt can be issued for the given submitted Receipt:

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

Body (in CBOR diagnostic notation)

/ cose-sign1 / 18([
  / protected   / <<{
    / key / 4 : "0vx7agoebGc...9nndrQmbX",
    / algorithm / 1 : -35,  # ES384
    / vds       / 395 : 1,  # RFC9162 SHA-256
    / claims / 15 : {
      / issuer  / 1 : "https://blue.example",
      / subject / 2 : "https://green.example/cli@v1.2.3",
      / iat / 6: 2443944944, # Post-refresh
    },
  }>>,
  / unprotected / {
    / proofs / 396 : {
      / inclusion / -1 : [
        <<[
          / size / 9, / leaf / 8,
          / inclusion path /
          h'7558a95f...e02e35d6'
        ]>>
      ],
    },
  },
  / payload     / null,
  / signature   / h'123227ed...ccd37123'
])
```

A TS may limit how often a new receipt can be issued, and respond with a 503 if a client requests new receipts too frequently.

The following HTTP endpoints are optional to implement.

2.2.2. Resolve Signed Statement

This endpoint enables Transparency Service APIs to act like Artifact Repositories, and serve Signed Statements directly, instead of indirectly through Receipts.

Request:

```
GET /signed-statements/9e4f...688a HTTP/1.1
Host: transparency.example
Accept: application/cose
```

Response:

One of the following:

2.2.2.1. Status 200 - Success

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

Body (in CBOR diagnostic notation)

18([                              / COSE Sign1         /
  h'a1013822',                    / Protected Header   /
  {},                             / Unprotected Header /
  null,                           / Detached Payload   /
  h'269cd68f4211dffc...0dcb29c' / Signature          /
])
```

2.2.2.2. Status 404 - Not Found

The following expected errors are defined. Implementations MAY return other errors, so long as they are valid [RFC9290] objects.

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

{
  / title /         -1: \
          "Not Found",
  / detail /        -2: \
          "No Signed Statement found with the specified ID"
```

2.2.2.3. Eventual Consistency

For all responses additional eventually consistent operation details MAY be present. Support for eventually consistent Receipts is implementation specific, and out of scope for this specification.

2.2.3. Resolve Issuer

This endpoint is inspired by [I-D.draft-ietf-oauth-sd-jwt-vc].

The following is a non-normative example of a HTTP request for the Issuer Metadata configuration when iss is set to https://transparency.example/tenant/1234:

Request:

```
GET /.well-known/issuer/tenant/1234 HTTP/1.1
Host: transparency.example
Accept: application/json

Response:

HTTP/1.1 200 OK
Content-Type: application/json

{
  "issuer": "https://transparency.example/tenant/1234",
  "jwks": {
    "keys": [
      {
        "kid": "urn:ietf:params:oauth\
                 :jwk-thumbprint:sha-256:Dgyo...agRo",
        "alg": "ES256",
        "use": "sig",
        "kty": "EC",
        "crv": "P-256",
        "x": "p-kZ4uOASt9IjQRTrWikGnlbGb-z3LU1ltwRjZaOS9w",
        "y": "ymXE1yltJPXgjQSRe9NweN3TLlSUALYZTzy83NVfdg0"
      },
      {
        "kid": "urn:ietf:params:oauth\
                 :jwk-thumbprint:sha-256:4Fzx...0ClE",
        "alg": "HPKE-Base-P256-SHA256-AES128GCM",
        "use": "enc",
        "kty": "EC",
        "crv": "P-256",
        "x": "Vreuil95vzR6ixutgBBf2ota-rj97MvKfuJWB4qqp5w",
        "y": "NkUTeaoNlLRRsVRxHGDA-RsA0ex2tSpcd3G-4SmKXbs"
      }
    ]
  }
}
```

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. User-host Authentication

[I-D.draft-ietf-scitt-architecture] defines 2 distinct roles that require authentication: Issuers who sign Statements, and Clients that submit API calls on behalf of Issuers. While Issuer authentication and signing of Statements is very important for the trustworthiness of systems implementing the SCITT building blocks, it is out of scope of this document. This document is only concerned with authentication of API clients.

For those endpoints that require client authentication, Transparency Services MUST support at least one of the following options:

* HTTP Authorization header with a JWT

* domain-bound API key

* TLS client authentication

Where authentication methods rely on long term secrets, both clients and Transparency Services implementing this specification SHOULD allow for the revocation and rolling of authentication secrets.

4.4. Primary Threats

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-liable operation, any attack which could cause the _silent_ failure of Signed Statement registration, for example, should be considered in scope.

In principle DoS attacks are easily mitigated by the 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 DoS attacks are not a major issue.

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 SHOULD implement general good practice around network attacks, flooding, rate limiting etc.

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.

Transparency Services MAY also implement additional protections such as anomaly detection or rate limiting in order to mitigate the impact of any breach.

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 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 MAY be added to the protected header signed by the Issuer.

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.

5. IANA Considerations

5.1. Well-Known URI for Transparency Configuration

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

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

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

6. References

6.1. 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>.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://doi.org/10.17487/RFC8174>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://doi.org/10.17487/RFC8615>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://doi.org/10.17487/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://doi.org/10.17487/RFC9110>.

[RFC9290] Fossati, T. and C. Bormann, "Concise Problem Details for Constrained Application Protocol (CoAP) APIs", RFC 9290, DOI 10.17487/RFC9290, October 2022, <https://doi.org/10.17487/RFC9290>.

6.2. Informative References

[I-D.draft-ietf-oauth-sd-jwt-vc]
Terbu, O., Fett, D., and B. Campbell, "SD-JWT-based Verifiable Credentials (SD-JWT VC)", Work in Progress, Internet-Draft, draft-ietf-oauth-sd-jwt-vc-13, 6 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-sd-jwt-vc-13>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://doi.org/10.17487/RFC2046>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://doi.org/10.17487/RFC6838>.

[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://doi.org/10.17487/RFC8792>.

Contributors

Orie Steele
Transmute
United States
Email: 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

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

Authors' Addresses

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

Jon Geater
DataTrails Inc.
United States
Email: jon.geater@datatrails.ai