An Architecture for Trustworthy and Transparent Digital Supply Chains

Abstract

Traceability of physical and digital Artifacts in supply chains is a long-standing, but increasingly serious security concern. The rise in popularity of verifiable data structures as a mechanism to make actors more accountable for breaching their compliance promises has found some successful applications to specific use cases (such as the supply chain for digital certificates), but lacks a generic and scalable architecture that can address a wider range of use cases.

This document defines a generic, interoperable and scalable architecture to enable transparency across any supply chain with minimum adoption barriers. It provides flexibility, enabling interoperability across different implementations of Transparency Services with various auditing and compliance requirements. Producers can register their Signed Statements on any Transparency Service, with the guarantee that all Consumers will be able to verify them.

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-architecture/.

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-architecture.
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 11 January 2024.

Copyright Notice

Copyright (c) 2023 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
   1.1. Requirements Notation
2. Use Cases
3. Terminology
4. Definition of Transparency
5. Architecture Overview
   5.1. Signed Statement Issuance and Registration
      5.1.1. Issuer Identity
      5.1.2. Naming Artifacts
      5.1.3. Signed Statement Metadata
   5.2. Transparency Service
      5.2.1. Service Identity, Remote Attestation, and Keying
      5.2.2. Registration Policies
      5.2.3. Registry Security Requirements
   5.3. Verifying Transparent Statements
1. Introduction

This document describes a scalable and flexible, decentralized architecture to enhance auditability and accountability across various existing and emerging supply chains. It achieves this goal by enforcing the following complementary security guarantees:

1. Statements made by Issuers about supply chain Artifacts must be identifiable, authentic, and non-repudiable;

2. such Statements must be registered on a secure append-only Log, so that their provenance and history can be independently and consistently audited;

3. Issuers can efficiently prove to any other party the Registration of their Signed Statements; verifying this proof ensures that the Issuer is consistent and non-equivocal when producing Signed Statements.

The first guarantee is achieved by requiring Issuers to sign their Statements and associated metadata using a distributed public key
infrastructure. The second guarantee is achieved by storing the Signed Statement on an immutable, append-only Log. The next guarantee is achieved by implementing the append-only Log using a verifiable data structure (such as a Merkle Tree \[\text{Merkle}\]). Lastly, the Transparency Service verifies the identity of the Issuer, and conformance to a Registration Policy associated with the instance of the Transparency Service. As the Issuer of the Signed Statement and conformance to the Registration Policy are confirmed, an endorsement is made as the Signed Statement is added to the append-only Log.

The guarantees and techniques used in this document generalize those of Certificate Transparency \[\text{RFC9162}\], which can be re-interpreted as an instance of this architecture for the supply chain of X.509 certificates. However, the range of use cases and applications in this document is much broader, which requires much more flexibility in how each Transparency Service is implemented and operates. Each service \textit{MAY} enforce its own Registration Policies for authorizing entities to register their Signed Statements to the append-only Log. Some Transparency Services may also enforce authorization policies limiting who can write, read and audit specific Feeds or the full registry. It is critical to provide interoperability for all Transparency Services instances as the composition and configuration of involved supply chain entities and their system components is ever-changing and always in flux, so it is implausible to expect all participants to choose a single vendor or registry.

A Transparency Service provides visibility into Signed Statements associated with various supply chains and their sub-systems. These Signed Statements (and corresponding Statement payload) make claims about the Artifacts produced by a supply chain. A Transparency Service endorses specific and well-defined metadata about these Artifacts that is captured in Statements. Some metadata is selected (and signed) by the Issuer, indicating, e.g., "who issued the Statement" or "what type of Artifact is described" or "what is the Artifact's version"; whereas additional metadata is selected (and countersigned) by the Transparency Services, indicating, e.g., "when was the Signed Statement about the Artifact registered in the Registry". Producing a Transparent Statement may be considered a form of notarization. A Statements payload content \textit{MAY} be encrypted and opaque to the Transparency Services, if so desired; however the metadata \textbf{MUST} be transparent in order to warrant trust for later processing. Transparent Statements provide a common basis for holding Issuers accountable for the Statement payload about Artifacts they release and (more generally) principals accountable for auxiliary Signed Statements from other Issuers about the original Signed Statement about an Artifact. Issuers may Register new Signed Statements about Artifacts, but they cannot delete or alter Signed Statements previously added to the append-only Log. A Transparency Service may restrict access to Signed Statements
through access control policies. However, third parties (such as Auditors) would be granted access as needed to attest to the validity of the Artifact, Feed or the entirety of the Transparency Service.

Trust in the Transparency Service itself is supported both by protecting their implementation (using, for instance, replication, trusted hardware, and remote attestation of a system's operational state) and by enabling independent audits of the correctness and consistency of its Registry, thereby holding the organization that operates it accountable. Unlike CT, where independent Auditors are responsible for enforcing the consistency of multiple independent instances of the same global Registry, each Transparency Service is required to guarantee the consistency of its own Registry (for instance, through the use of a consensus algorithm between replicas of the Registry), but assume no consistency between different Transparency Services.

Breadth of access is critical so the Transparency Service specified in this architecture cater to two types of audiences:

1. Producers: organizations, stakeholders, and users involved in creating or attesting to supply chain artifacts, releasing authentic Statements to a definable set of peers; and

2. Consumers: organizations, stakeholders, and users involved in validating supply chain artifacts, but can only do so if the Statements are known to be authentic. Consumers MAY be producers, providing additional Signed Statements, attesting to conformance of various compliance requirements.

Signed Statement Issuers rely on being discoverable and represented as the responsible parties for their registered Signed Statements via Transparency Services in a believable manner. The issuer of a Signed Statement should be authenticated and authorized according to the registration policy of the Transparency Service. Analogously, Transparent Statement Consumers rely on verifiable trustworthiness assertions associated with Transparent Statements and their processing provenance in a believable manner. If trust can be put into the operations that record Signed Statements in a secure, append-only log via online operations, the same trust can be put into the resulting transparent statement, issued by the Transparency Services and that can be validated in offline operations.

The Transparency Services specified in this architecture can be implemented by various different types of services in various types of languages provided via various variants of API layouts.
The interoperability guaranteed by the Transparency Services is enabled via core components (architectural constituents) that come with prescriptive requirements (that are typically hidden away from the user audience via APIs but can be relied upon as non functional requirements). Many of the data elements processed by the core components are based on the Concise Signing and Encryption standard specified in [RFC9052], which is used to produce Signed Statements about Artifacts and to build and maintain a Merkle tree that functions as an append-only Log for corresponding Signed Statements.

1.1. Requirements Notation

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.

2. Use Cases

The building blocks defined in SCITT are intended to support applications in any supply chain that produces or relies upon digital artifacts, from the build and supply of software and IoT devices to advanced manufacturing and food supply.

Detailed use cases are maintained in a separate document [I-D.ietf-scitt-software-use-cases].

3. Terminology

The terms defined in this section have special meaning in the context of Supply Chain Integrity, Transparency, and Trust throughout this document. When used in text, the corresponding terms are capitalized. To ensure readability, only a core set of terms is included in this section.

**Artifact**: a physical or non-physical item that is moving along the supply chain.

**Auditor**: an entity that checks the correctness and consistency of all Transparent Statements issued by a Transparency Service.

**Consumer of Signed Statements**: Define here.

**Envelope**: metadata and an Issuer's signature is added to a Statement via a COSE Envelope by the Issuer to produce a Signed Statement. An Envelope contains the identity of the Issuer and other information to help components responsible for validation that are part of a Transparency Services to identify the software Artifact referred to in a Signed Statement. In essence, a Signed
Statement is a COSE Envelope wrapped around a Statement binding the metadata included in the Envelope to a Statement. In COSE, an Envelope consists of a protected header (included in the Issuer's signature) and an unprotected header (not included in the Issuer's signature).

**Feed:** an identifier chosen by the Issuer for the Artifact. For every Issuer and Feed, the Registry on a Transparency Service contains a sequence of Signed Statements about the same Artifact. In COSE, Feed is a dedicated header attribute in the protected header of the Envelope.

**Issuer:** an entity that creates Signed Statements about software Artifacts in the supply chain. An Issuer may be the owner or author of Artifacts, or an independent third party such as a reviewer or an endorser.

**Append-only Log (converges Ledger and Registry):** the verifiable append-only data structure that stores Signed Statements in a Transparency Service. SCITT supports multiple Log and Receipt formats to accommodate different Transparency Service implementations, such as historical Merkle Trees and sparse Merkle Trees.

**Receipt:** a Receipt is a cryptographic proof that a Signed Statement is recorded in the Registry. Receipts are based on COSE Signed Merkle Tree Proofs; they consist of a Registry-specific inclusion proof, a signature by the Transparency Service of the state of the Registry, and additional metadata (contained in the signature's protected headers) to assist in auditing.

**Registration:** the process of submitting a Signed Statement to a Transparency Service, applying the Transparency Service's Registration Policy, storing it in the Registry, producing a Receipt, and returning it to the submitting Issuer.

**Registration Policy:** the pre-condition enforced by the Transparency Service before registering a Signed Statement, rendering it a Signed Statement, based on metadata contained in its COSE Envelope (notably the identity of its Issuer) and on prior Signed Statements already added to a Registry.

**Registry:** the verifiable append-only data structure that stores Signed Statements in a Transparency Service often referred to by the synonym log or ledger. Since COSE Signed Merkle Tree Proofs support multiple Merkle Tree algorithms, SCITT supports different Transparency
Service implementations of the Registry, such as historical Merkle Trees or sparse Merkle Trees.

**Signed Statement**: an identifiable and non-repudiable Statement about an Artifact made by an Issuer. In SCITT, Signed Statements are encoded as COSE signed objects; the payload of the COSE structure contains the issued Statement.

**Statement**: any serializable information about an Artifact. To help interpretation of Statements, they must be tagged with a media type (as specified in [RFC6838]). For example, a Statement may represent a Software Bill Of Materials (SBOM) that lists the ingredients of a software Artifact, or some endorsement or attestation about an Artifact.

**Transparency Service**: an entity that maintains and extends the Registry, and endorses its state. A Transparency Service is often referred to by its synonym Notary. A Transparency Service can be a complex distributed system, and SCITT requires the Transparency Service to provide many security guarantees about its Registry. The identity of a Transparency Service is captured by a public key that must be known by Verifiers in order to validate Receipts.

**Transparent Statement**: a Signed Statement that is augmented with a Receipt created via Registration in a Transparency Service (the receipt is stored in the unprotected header of COSE Envelope of the Signed Statement). A Transparent Statement remains a valid Signed Statement, and may be registered again in a different Transparency Service.

**Verifier**: an entity that consumes Transparent Statements (a specialization of Signed Statement Consumer), verifying their proofs and inspecting their Statement payload, either before using corresponding Artifacts, or later to audit an Artifact's provenance on the supply chain.

4. **Definition of Transparency**

In this document, the definition of transparency is intended to build over abstract notions of Registry and Receipts. Existing transparency systems such as Certificate Transparency are instances of this definition.

A Signed Statement is an identifiable and non-repudiable Statement made by an Issuer. The Issuer selects additional metadata and attaches a proof of endorsement (in most cases, a signature) using the identity key of the Issuer that binds the Statement and its metadata. Signed Statements can be made transparent by attaching a proof of Registration by a Transparency Service, in the form of a
Receipt that countersigns the Signed Statement and witnesses its inclusion in the Registry of a Transparency Service. By extension, the document may say an Artifact (e.g., a firmware binary) is transparent if it comes with one or more Transparent Signed Statements from its author or owner, though the context should make it clear what type of Signed Statements is expected for a given Artifact.

Transparency does not prevent dishonest or compromised Issuers, but it holds them accountable: any Artifact that may be used to target a particular user that checks for Receipts must have been recorded in the tamper-proof Registry, and will be subject to scrutiny and auditing by other parties.

Transparency is implemented by a Registry that provides a consistent, append-only, cryptographically verifiable, publicly available record of entries. Implementations of Transparency Services may protect their Registry using a combination of trusted hardware, replication and consensus protocols, and cryptographic evidence. A Receipt is an offline, universally-verifiable proof that an entry is recorded in the Registry. Receipts do not expire, but it is possible to append new entries (more recent Signed Statements) that subsume older entries (less recent Signed Statements).

Anyone with access to the Registry can independently verify its consistency and review the complete list of Transparent Statements registered by each Issuer. However, the Registries of separate Transparency Services are generally disjoint, though it is possible to take a Transparent Statement from one Registry and register it again on another (if its policy allows it), so the authorization of the Issuer and of the Registry by the Verifier of the Receipt are generally independent.

Reputable Issuers are thus incentivized to carefully review their Statements before signing them to produce Signed Statements. Similarly, reputable Transparency Services are incentivized to secure their Registry, as any inconsistency can easily be pinpointed by any Auditor with read access to the Registry. Some Registry formats may also support consistency auditing (Section 5.2.3.2) through Receipts, that is, given two valid Receipts the Transparency Service may be asked to produce a cryptographic proof that they are consistent. Failure to produce this proof can indicate that the Transparency Services operator misbehaved.
5. Architecture Overview

The SCITT architecture consists of a very loose federation of Transparency Services, and a set of common formats and protocols for issuing and registering Signed Statements, and auditing Transparent Statements.

In order to accommodate as many Transparency Service implementations as possible, this document only specifies the format of Signed Statements (which must be used by all Issuers) and a very thin wrapper format for Receipts, which specifies the Transparency Service identity and the agility parameters for the Merkle Tree Proof. Most of the details of the Receipt's contents are specified...
This section describes at a high level, the three main roles and associated processes in SCITT: Issuers and the Signed Statement issuance process, Transparency Service and the Signed Statement Registration process, as well as Verifiers of the Transparent Statements and the Receipt validation process.

5.1. Signed Statement Issuance and Registration

5.1.1. Issuer Identity

Before an Issuer is able to produce Signed Statements, it must first create its decentralized identifier [DID-CORE] (also known as a DID). A DID can be resolved into a key manifest (a list of public keys indexed by a key identifier) using many different DID methods.

Issuers MAY choose the DID method they prefer, but with no guarantee that all Transparency Services will be able to register their Signed Statements. To facilitate interoperability, all Transparency Service implementations SHOULD support the did:web method [DID-WEB]. For instance, if the Issuer publishes its manifest at https://sample.issuer/user/alice/did.json, the DID of the Issuer is did:web:sample.issuer:user:alice.

Issuers SHOULD use consistent decentralized identifiers for all their Statements about Artifacts, to simplify authorization by Verifiers and auditing. They MAY update their DID manifest, for instance to refresh their signing keys or algorithms, but they SHOULD NOT remove or change any prior keys unless they intend to revoke all Signed Statements that are registered as Transparent Statements issued with those keys. This DID appears in the Issuer protected header of Signed Statements' Envelopes, while the version of the key from the manifest used to sign the Signed Statement is written in the kid header.

kid MUST either be an absolute URL, or a relative URL. Relative URL MUST be relative to an iss value. When relative URL is used, iss MUST also be present in the protected header.

Resolving kid MUST return an identity document of a registered content type (a set of public keys). In the case of kid being an absolute DID URL, the identity document is called a DID Document, and is expected ot have content type application/did+json.

To dereference a DID URL, it first MUST be resolved. After that the fragment is processed according to the media type.
For example, when resolving did:example:123#key-42, first, the identity document for did:example:123 is resolved as content type application/did+json, next, the fragment #key-2 is dereferenced to a verification method that contains a publicKeyJwk property.

The content type of publicKeyJwk is expected to be application/jwk+json.

The details of both DID resolution and DID dereferencing are out of scope for this document.

The iss or kid, might not be DID URLs, however the following interfaces MUST be satisfied in order to ensure issuer identity documents, and associated keys are discoverable in a consistent manner.

5.1.1.1. Resolving Identity Documents

The value of id might be found the iss or sub claims if they are present in the protected header or payload.

\[
\text{resolve} = (\text{id: string, accept: content\_type = 'application/did+json')} \Rightarrow \text{idDocument (of content type application/did+json)}.
\]

For example:

did:example:123

Might resolve to:

\{
  "id": "did:example:123",
  "verificationMethod": [{
    "id": "#key-42",
    "type": "JsonWebkey",
    "controller": "did:example:123",
    "publicKeyJwk": {
      "kty": "EC",
      "crv": "P-384",
      "alg": "ES384",
      "x": "LCeAt2sW36j94wuFP09gNEIHDzqR6Nh_Udu2bLer3cKFBCaAHY1svmbPV69bP3RH",
      "y": "zz2SkcOGYM6PbYlw19tcbpzo6bEMYHIwGBnN5rd8QWykAprrstPdxx4U0uScvDcYd"
    }
  }
}\}

Editor note, we might wish to eliminate this intermediate identity document content type, by treating it as an alterative encoding of application/jwk-set+json or application/cose-key-set.

However, there is no media type fragment processing directive that would enable dereferencing the known key set content types, listed above.

5.1.1.1.1. Comment on OIDC

For well known token types, such as id_token or access_token.
iss MUST be a URL, and it MUST have keys discoverable in the following way:

iss can be used to build a .well-known URL to discovery the issuer's configuration.

For example, iss contoso.example will have the following open id connect configuration URL.


This URL will resolve to a JSON document which contains the property:

jwks_uri, for example https://contoso.example/.well-known/jwks.json

This URL will resolve to a JSON document of content type application/jwk-set+json, which will contain specific keys... for example:

```
```json
{ "keys": [ { "alg": "RS256", "kty": "RSA", "use": "sig", "n": "wM9TkScbcn5FV3iUJ-812sQTwvTCFrDm6vV2U-g23gn6rRbdFQfDbgEnSkolph6CanOTq1KVHjkJHld6Q4MDVGIdbVbHESxib2YiZyJv US-0oQgizkB8EJxyH4213x3_x4a_snAagLUI-Ykt_gaMPSw_vpxe-pBxu-jd14i-jDfwoPJuDF8ZJGS9orCPRiHCYLDg0scC9XibH9rUtbVg8q4bAPx90x6malx40LvU3pXv jew6LSG31B2YhpCWe6VoMzYXYqCin5MK_0Kod6CcCftDgu3I56SgSfsf7-tI7g1IzO8RMuzqH0kLjVijurjYzRxnM7WqBMPxMHU8i4Z04zw", "e": "AQAB", "kid": "NTBGNTJEMdc3RUE3RUVEOTM4Dc8yeOEFDnZeOsYy5N5NG0Q40EU50A", "x5t": "NTBGNTJEMdc3RUE3RUVEOTM4Dc8yeOEFDnZeOsYy5N5NG0Q40EU50A", "x5c": [ "MiIDczCCAfOgAwIABAgIJAaNPg08XWwstdMA6GqCSqGSIb3DQEBAQUAM8wGjBGVBAM MEOVwnbnRvc28uYXV0aDluYu9tMB4XDEoM0DcxeM2E2NTQyN1oXDTI4MDMxOTE2NTQyN1o wHDeaMgBaIAUeAwwRY24ud9g2by5hDrXRmC5j2b0wggEiMA0GCSqGSIb3DQEBAQUAA4I BDuAwggEKAoIBAQDDBlRJtYf3VXeJQn7zXayp0/ BMYIWsOb8PZ6T6DeCfquF0V1Bt/ ZuA5dKS1wnHoJq5hNwUppEqMct3DgWNuJaJ1tUGERLGJvZgjM1RL7ShCCLOQEJQNI cjhmxFFf+x0D/J6ATSL51S3+Bow9LD++nF76kHg76N3X16MN/ Cg8lR0XXkKL2isIGICjgsOA6xwL1esJf2tR08byrhS/ H07HqZxHg4u9Te1dWN7DosbeIGLZ1Gkj7q7+gy9k1heoLWfkyT8o60ZwI0W0C7cjnP IZ+3wvX66jouboV7xey70oofQulWkuBNjNeucmxtasBw9eYdTyLjP7PajMBaAgjUDPBO BGO4ludQwBBTLarHdkNa5CzPyiKJU51t8Jwn9WTaFbgvN/hSMEGDAwyBTLaRdHdn6aC zPyiKJU51t8Jwn9WTAMBGaNVHRMEBTAQH/ MA6GqCSqGSIb3DQEBAQUA4IcHAQgJbFmo+m+Bpqqk59rQBC0X6ops1wBCXHGc1енькG19qe RwLEwSeF5HPz4TT1f2lcf4pCq2v0hbVNJFvNV+Pj9RACKto+v1n84i/ U4BBezYuXzIpEbV7hV/PwXg8tCvrtPyPajG0UAA/ puAB6CFvyy+LkC4YMRKmD7ZrC4zjxWzhFgNanf2ĚeryxCA3RMNF6Iw7tbg9ZGETeA/ DHVnZgh/e82KVX7ZEknB4MjCQrwj9qswSMBTBIYp0/ vR19cxDFHllwnyYAuzeZDHtWt+Rp2jX7Qwfi9YygcxyjkgAUEZc4WpdNlQ1Wf5G5ep0tH GiwiJS+u/NSVqCFT57+g3R+] }, { "alg": "RS256", "kty": "RSA", 
```
If SCITT wanted to be interoperable with OIDC, we would define key dereferencing in a way that was compatible with how OIDC handles it today.

### 5.1.1.2 Dereferencing Public Keys

kid is always present in the protected header.

If iss is also present, kid *MUST* be a relative URL to iss, otherwise kid *MUST* be an absolute URL that starts with iss.

\[id = \text{kid if iss is undefined, or } \text{iss} + \# + \text{kid when iss is defined.}\]


dereference = (id: string, accept: content_type = 'application/jwk+json') => publicKeyJwk (of content type application/jwk+json).

For example, when DIDs are used:
Might dereference to:

```json
{ "kty": "EC", "crv": "P-384", "alg": "ES384", "x": "LCeAt2sW36j94wuFP0gNEIHDzqR6Nh_Udu20bLer3cKFBcAhY1svmbPV69bP3RH", "y": "zz2SkcOGYM6PbYlw19tcbpzo6bEMYHIwGBnN5rd8QWykAprstPdxx4U0uScvDcYd" }
```

5.1.2. Naming Artifacts

Many Issuers issue Signed Statements about different Artifacts under the same DID, so it is important for everyone to be able to immediately recognize by looking at the Envelope of a Signed Statements what Artifact it is referring to. This information is stored in the Feed header of the Envelope. Issuers MAY use different signing keys (identified by kid in the resolved key manifest) for different Artifacts, or sign all Signed Statements under the same key.

5.1.3. Signed Statement Metadata

Besides Issuer, Feed and kid, the only other mandatory metadata in a Signed Statement is the type of the Payload, indicated in the cty (content type) Envelope header. However, this set of mandatory metadata is not sufficient to express many important Registration Policies. For example, a Registry may only allow a Signed Statement to be registered, if it was signed recently. While the Issuer is free to add any information in the payload of the Signed Statements, the Transparency Services (and most of its Auditors) can only be expected to interpret information in the Envelope.

Such metadata, meant to be interpreted by the Transparency Services during Registration Policy evaluation, should be added to the reg_info header. While the header MUST be present in all Signed Statements, its contents consist of a map of named attributes. Some attributes (such as the Issuer's timestamp) are standardized with a defined type, to help uniformize their semantics across Transparency Services. Others are completely customizable and may have arbitrary types. In any case, all attributes are optional; so the map MAY be empty.

5.2. Transparency Service

The role of Transparency Service can be decomposed into several major functions. The most important is maintaining a Registry, the verifiable data structure that records Signed Statements, and enforcing a Registration Policy. It also maintains a service key, which is used to endorse the state of the Registry in Receipts. All
Transparency Services **MUST** expose standard endpoints for Registration of Signed Statements and Receipt issuance, which is described in Section 8.1. Each Transparency Service also defines its own Registration Policies, which **MUST** apply to all entries in the Registry.

The combination of Registry, identity, Registration Policy evaluation, and Registration endpoint constitute the trusted part of the Transparency Service. Each of these components **SHOULD** be carefully protected against both external attacks and internal misbehavior by some or all of the operators of the Transparency Service. For instance, the code for policy evaluation, Registry extension and endorsement may be protected by running in a TEE; the Registry may be replicated and a consensus algorithm such as Practical Byzantine Fault Tolerance (pBFT [PBFT]) may be used to protect against malicious or vulnerable replicas; threshold signatures may be use to protect the service key, etc.

Beyond the trusted components, Transparency Services may operate additional endpoints for auditing, for instance to query for the history of Signed Statements registered by a given Issuer via a certain Feed. Implementations of Transparency Services **SHOULD** avoid using the service identity and extending the Registry in auditing endpoints; as much as practical, the Registry **SHOULD** contain enough evidence to re-construct verifiable proofs that the results returned by the auditing endpoint are consistent with a given state of the Registry.

### 5.2.1. Service Identity, Remote Attestation, and Keying

Every Transparency Service **MUST** have a public service identity, associated with public/private key pairs for signing on behalf of the service. In particular, this identity must be known by Verifiers when validating a Receipt.

This identity should be stable for the lifetime of the service, so that all Receipts remain valid and consistent. The Transparency Service operator **MAY** use a distributed identifier as their public service identity if they wish to rotate their keys, if the Registry algorithm they use for their Receipt supports it. Other types of cryptographic identities, such as parameters for non-interactive zero-knowledge proof systems, may also be used in the future.

A Transparency Services **SHOULD** provide evidence that it is securely implemented and operated, enabling remote authentication of the hardware platforms and/or software TCB that run the Transparency Service. This additional evidence **SHOULD** be recorded in the Registry and presented on demand to Verifiers and Auditors. Examples for Statements that can improve trustworthy assessments of Transparency
Services are RATS Conceptual Messages, such as Evidence, Endorsements, or corresponding Attestation Results (see [RFC9334]).

For example, consider a Transparency Service implemented using a set of replicas, each running within its own hardware-protected trusted execution environments (TEEs). Each replica **SHOULD** provide a recent attestation report for its TEE, binding their hardware platform to the software that runs the Transparency Service, the long-term public key of the service, and the key used by the replica for signing Receipts. This attestation evidence **SHOULD** be supplemented with transparency Receipts for the software and configuration of the service, as measured in its attestation report.

### 5.2.2. Registration Policies

A Transparency Service that accepts to register any valid Signed Statement offered by anonymous Issuers would only provide limited value, or no value, to verifiers. As a consequence, some form of authorization is needed prior to registration of Signed Statements to ensure completeness of audit. More advanced use case will rely on the Transparency Service performing additional domain-specific checks before a Signed Statement is accepted. For example, some Transparency Services may validate the content of Signed Statements.

We use the term "registration policies" to refer to the checks that are performed before a Signed Statement is registered given a set of input values. This baseline specification leaves the implementation of the registration policy to the provider of the Transparency Services and its users.

As a minimum we expect that a deployment authenticates the Issuer of the Signed Statement, which requires some form of trust anchor. As defined in [RFC6024], "A trust anchor represents an authoritative entity via a public key and associated data. The public key is used to verify digital signatures, and the associated data is used to constrain the types of information for which the trust anchor is authoritative." The Trust Anchor may be a certificate, a raw public key or other structure, as appropriate. It can be a non-root certificate when it is a certificate.

A provider of a Transparency Service is, however, expected to indicate what registration policy is used in a given deployment and inform its users about changes to the registration policy.

### 5.2.3. Registry Security Requirements

There are many different candidate verifiable data structures that may be used to implement the Registry, such as chronological Merkle Trees, sparse/indexed Merkle Trees, full blockchains, and many other variants. The Registry is only required to support concise Receipts
(i.e., whose size grows at most logarithmically in the number of entries in the Registry) that can be encoded as a COSE Signed Merkle Tree Proof.

It is possible to offer multiple signature algorithms for the COSE signature of receipts' Signed Merkle Tree, or to change the signing algorithm at later points. However, the Merkle Tree algorithm (including its internal hash function) cannot easily be changed without breaking the consistency of the Registry. It is possible to maintain separate Registries for each algorithm in parallel but the Transparency Service is then responsible for proving their mutual consistency.

5.2.3.1. Finality

A Registry is append-only: once a Signed Statement is registered and becomes a Transparent Statement, it cannot be modified, deleted, or moved. In particular, once a Receipt is returned for a given Signed Statement, the registered Signed Statement and any preceding entry in the Registry become immutable, and the Receipt provides universally-verifiable evidence of this property.

5.2.3.2. Consistency

There is no fork in the Registry: everyone with access to its contents sees the same sequence of entries, and can check its consistency with any Receipts they have collected. Transparency Service implementations SHOULD provide a mechanism to verify that the state of the Registry encoded in an old Receipt is consistent with the current Registry state.

5.2.3.3. Replayability and Auditing

Everyone with access to the Registry can check the correctness of its contents. In particular,

*the Transparency Service defines and enforces deterministic Registration Policies that can be re-evaluated based solely on the contents of the Registry at the time of Registration, and must then yield the same result.

*the ordering of entries, their cryptographic contents, and the Registry governance may be non-deterministic, but they must be verifiable.

*a Transparency Service SHOULD store evidence about the resolution of distributed identifiers into manifests.

*a Transparency Service MAY additionally support verifiability of client authentication and access control.
5.2.3.4. Governance and Bootstrapping

The Transparency Service needs to support governance, with well-defined procedures for allocating resources to operate the Registry (e.g., for provisioning trusted hardware and registering their attestation materials in the Registry) and for updating its code (e.g., relying on Transparent Statement about code updates, secured on the Registry itself, or on some auxiliary Transparency Service).

Governance procedures, their auditing, and their transparency are implementation specific. A Transparency Service SHOULD document them.

*Governance may be based on a consortium of members that are jointly responsible for the Transparency Services, or automated based on the contents of an auxiliary governance Transparency Service.

*Governance typically involves additional records in the Registry to enable its auditing. Hence, the Registry may contain both Transparent Statements and governance entries.

*Issuers, Verifiers, and third-party Auditors may review the Transparency Service governance before trusting the service, or on a regular basis.

5.3. Verifying Transparent Statements

For a given Artifact, Verifiers take as trusted inputs:

1. the distributed identifier of the Issuer (or its resolved key manifest),
2. the expected name of the Artifact (i.e., the Feed),
3. the list of service identities of trusted Transparency Services.

When presented with a Transparent Statement for an Artifact, Consumers verify its Issuer identity, signature, and Receipt. They may additionally apply a validation policy based on the protected headers present both in the Envelope, the Receipt, or the Statement itself, which may include security-critical or Artifact-specific details.

Some Verifiers may systematically resolve Issuer DIDs to fetch the latest corresponding DID documents. This behavior strictly enforces the revocation of compromised keys: once the Issuer has updated its Statement to remove a key identifier, all Signed Statements include
the corresponding kid will be rejected. However, others may delegate DID resolution to a trusted third party and/or cache its results.

Some Verifiers may decide to skip the DID-based signature verification, relying on the Transparency Service's Registration Policy and the scrutiny of other Verifiers. Although this weakens their guarantees against key revocation, or against a corrupt Transparency Services, they can still keep the Receipt and blame the Issuer or the Transparency Services at a later point.

6. Signed Statement Issuance, Registration, and Verification

This section details the interoperability requirements for implementers of Signed Statements issuance and validation libraries, and of Transparency Services.

6.1. Envelope and Signed Statement Format

The formats of Signed Statements and Receipts are based on CBOR Object Signing and Encryption (COSE [RFC9052]). The choice of CBOR [RFC8949] is a trade-off between safety (in particular, non-malleability: each Signed Statement has a unique serialization), ease of processing and availability of implementations.

At a high-level that is the context of this architecture, a Signed Statement is a COSE single-signed object (i.e., a COSE_Sign1) that contains the correct set of protected headers. Although Issuers and relying parties may attach unprotected headers to Signed Statements, Transparency Services and Verifiers MUST NOT rely on the presence or value of additional unprotected headers in Signed Statements during Registration and validation.

All Signed Statements MUST include the following protected headers:

*algorithm (label: 1): Asymmetric signature algorithm used by the Issuer of a Signed Statement, as an integer, for example -35 for ECDSA with SHA-384, see COSE Algorithms Registry [IANA.cose];

*Issuer (label: TBD, temporary: 391): DID (Decentralized Identifier [DID-CORE]) of the signer, as a string, for example did:web:example.com;

*Feed (label: TBD, temporary: 392): the Issuer's name for the Artifact, as a string;

*payload type (label: 3): media-type of Statement payload as a string, for example application/spdx+json

*Registration Policy info (label: TBD, temporary: 393): a map of additional attributes to help enforce Registration Policies;
*Key ID (label: 4): Key ID, as a bytestring.

Additionally, Signed Statements MAY carry the following unprotected headers:

*Receipts (label: TBD, temporary: 394): Array of Receipts, defined below. This allows the Receipt to be attached to the Signed Statement, thus making a Transparent Statement.

In CDDL [RFC8610] notation, the Envelope is defined as follows:

```
SCITT_Envelope = COSE_Sign1_Tagged

COSE_Sign1_Tagged = #6.18(COSE_Sign1)

COSE_Sign1 = [
    protected : bstr .cbor Protected_Header,
    unprotected : Unprotected_Header,
    payload : bstr,
    signature : bstr
]

Reg_Info = {
    ? "register_by": uint .within (~time),
    ? "sequence_no": uint,
    ? "issuance_ts": uint .within (~time),
    ? "no_replay": null,
    * tstr => any
}

; All protected headers are mandatory, to protect against faulty impleme
; that may accidentally read a missing protected header from the unprote
Protected_Header = {
    1 => int ; algorithm identifier
    3 => tstr ; payload type
    4 => bstr ; Key ID
    ; TBD, Labels are temporary
    391 => tstr ; DID of Issuer
    392 => tstr ; Feed
    393 => Reg_Info ; Registration Policy info
}

Unprotected_Header = {
    ; TBD, Labels are temporary
    ? 394 => [+ Receipt]
}
```
6.2. Receipts

Receipts are based on COSE Signed Merkle Tree Proofs ([I-D.draft-steele-cose-merkle-tree-proofs]) with an additional wrapper structure that adds the following information:

*version: Receipt version number; this should be set to 0 for implementation of this document. We envision that future version of SCITT may add support for more complex receipts; for instance, registrations on multiple TS, receipts for dependency graphs and endorsements of Signed Claims, etc.

*ts_identifier: The DID of the Transparency Service that issued the claim. Verifiers MAY use this DID as a key discovery mechanism to verify the COSE Merkle Root signature; in this case the verification is the same as for Signed Claims and the signer should include the Key ID header. Verifiers MUST support the did:web method, all other methods are optional.

We also introduce the following requirements for the COSE signature of the Merkle Root:

*The SCITT version header MUST be included and its value match the version field of the Receipt structure.

*The DID of issuer header (like in Signed Claims) MUST be included and its value match the ts_identifier field of the Receipt structure.

*TS MAY include the Registration policy info header to indicate to verifiers what policies have been applied at the registration of this claim.

*Since [I-D.draft-steele-cose-merkle-tree-proofs] uses optional headers, the crit header (id: 2) MUST be included and all SCITT-specific headers (version, DID of TS and Registration Policy) MUST be marked critical.

The TS may include the registration time to help verifiers decide about the trustworthiness of the Transparent Statement. The registration time is defined as the timestamp at which the TS has added this Signed Statement to its Registry.
6.3. Signed Statement Issuance

There are many types of Statements (such as SBOMs, malware scans, audit reports, policy definitions) that Issuers may want to turn into Signed Statements. An Issuer must first decide on a suitable format to serialize the Statement payload. For a software supply chain, payloads describing the software artifacts may, for example, include

* JSON-SPDX
* CBOR-SPDX
* SWID
* CoSWID
* CycloneDX
* in-toto
* SLSA

Receipt = [
   version: int,
   ts_identifier: tstr,
   proof: SignedMerkleTreeProof
]

; Additional protected headers in the COSE signed_tree_root of the SignedMerkleTreeProof
Protected_Header = {
   390 => int ; SCITT Receipt Version
   394 => tstr ; DID of Transparency Service (required)
   ? 395 => RegistrationInfo ; Registration policy information (optional)

   ; Other COSE Signed Merkle Tree headers
   ; (e.g. tree algorithm, tree size)

   ; Additional standard COSE headers
   2 => [+ label] ; Critical headers
   ? 4 => bstr ; Key ID (optional)
   ? 33 => COSE_X509 ; X.509 chain (optional)
}

; Details of the registration info, as provided by the TS
RegistrationInfo = {
   ? "registration_time": uint .within (~time),
   * tstr => any
}
Once the Statement is serialized with the correct media-type/content-format, an Issuer should fill in the attributes for the Registration Policy information header. From the Issuer's perspective, using attributes from named policies ensures that the Signed Statement may only be registered on Transparency Services that implement the associated policy. For instance, if a Signed Statement is frequently updated, and it is important for Verifiers to always consider the latest version, Issuers SHOULD use the sequence_no or issuer_ts attributes.

Once all the Envelope headers are set, an Issuer MUST use a standard COSE implementation to produce an appropriately serialized Signed Statement (the SCITT tag of COSE_Sign1_Tagged is outside the scope of COSE, and used to indicate that a signed object is a Signed Statement).

6.4. Registering Signed Statements

The same Signed Statement may be independently registered in multiple Transparency Services. To register a Signed Statement, the service performs the following steps:

1. Client authentication. This is implementation-specific and MAY be unrelated to the Issuer identity. Signed Statements may be registered by a different party than their Issuer.

2. Issuer identification. The Transparency Service MUST store evidence of the DID resolution for the Issuer protected header of the Envelope and the resolved key manifest at the time of Registration for auditing. This MAY require that the service resolves the Issuer DID and record the resulting document, or rely on a cache of recent resolutions.

3. Envelope signature verification, as described in COSE signature, using the signature algorithm and verification key of the Issuer DID document.

4. Envelope validation. The service MUST check that the Envelope includes a Statement payload and the protected headers listed above. The service MAY additionally verify the Statement payload format and content.

5. Apply Registration Policy: for named policies, the Transparency Service should check that the required Registration info attributes are present in the Envelope and apply the check described in Table 1. A Transparency Service MUST reject Signed Statements that contain an attribute used for a named policy that is not enforced by the service. Custom Signed Statements are evaluated given the current Registry state and the entire
Envelope, and MAY use information contained in the attributes of named policies.

6. Commit (register) the new Signed Statement to the Registry

7. Sign and return the Receipt.

The last two steps MAY be shared between a batch of Signed Statements recorded in the Registry.

A Transparency Service MUST ensure that a Signed Statement is registered before releasing its Receipt, so that it can always back up the Receipt by releasing the corresponding entry (the now Transparent Statement) in the Registry. Conversely, the service MAY re-issue Receipts for the Registry content, for instance after a transient fault during Signed Statement Registration.

6.5. Validation of Transparent Statements

This section provides additional implementation considerations. The high-level validation algorithm is described in Section 5.3; the Registry-specific details of checking Receipts are covered in [I-D.draft-steele-cose-merkle-tree-proofs].

Before checking a Transparent Statement, the Verifier must be configured with one or more identities of trusted Transparency Services. If more than one service is configured, the Verifier MUST return which service the Transparent Statement is registered on.

In some scenarios, the Verifier already expects a specific Issuer and Feed for the Transparent Statement, while in other cases they are not known in advance and can be an output of validation. Verifiers SHOULD offer a configuration to decide if the Issuer's signature should be locally verified (which may require a DID resolution, and may fail if the manifest is not available or if the key is revoked), or if it should trust the validation done by the Transparency Service during Registration.

Some Verifiers MAY decide to locally re-apply some or all of the Registration Policies, if they have limited trust in the Transparency Services. In addition, Verifiers MAY apply arbitrary validation policies after the signature and Receipt have been checked. Such policies may use as input all information in the Envelope, the Receipt, and the Statement payload, as well as any local state.

Verifiers SHOULD offer options to store or share Receipts in case they are needed to audit the Transparency Services in case of a dispute.
7. Federation

This topic is still under discussion, see issue 79.

Multiple, independently-operated Transparency Services can help secure distributed supply chains, without the need for a single, centralized service trusted by all parties. For example, multiple Transparency Service instances may be governed and operated by different organizations that do not trust one another.

This may involve registering the same Signed Statements at different Transparency Services, each with their own purpose and Registration Policy. This may also involve attaching multiple Receipts to the same Signed Statements, each Receipt endorsing the Issuer signature and a subset of prior Receipts, and each Transparency Service verifying prior Receipts as part of their Registration Policy.

For example, a supplier's Transparency Service may provide a complete, authoritative Registry for some kind of Signed Statements, whereas a Consumer's Transparency Service may collect different kinds of Signed Statements to ensure complete auditing for a specific use case, and possibly require additional reviews before registering some of these Signed Statements.

8. Transparency Service API

8.1. Messages

All messages are sent as HTTP GET or POST requests.

If the Transparency Service cannot process a client's request, it MUST return an HTTP 4xx or 5xx status code, and the body SHOULD be a JSON problem details object ([RFC7807]) containing:

* **type**: A URI reference identifying the problem. To facilitate automated response to errors, this document defines a set of standard tokens for use in the type field within the URN namespace of: "urn:ietf:params:scitt:error:"

* **detail**: A human-readable string describing the error that prevented the Transparency Service from processing the request, ideally with sufficient detail to enable the error to be rectified.

Error responses SHOULD be sent with the Content-Type: application/problem+json HTTP header.

As an example, submitting a Signed Statement with an unsupported signature algorithm would return a 400 Bad Request status code and the following body:
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 a 503 response, the Transparency Service **MAY** include a Retry-After header field per [RFC7231] 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.

### 8.1.1. Register Signed Statement

#### 8.1.1.1. Request

POST <Base URL>/entries

Headers:

*Content-Type: application/cose

Body: SCITT COSE_Sign1 message

#### 8.1.1.2. Response

One of the following:

*Status 201 - Registration is successful.*

-Header Location: <Base URL>/entries/<Entry ID>

-Header Content-Type: application/json

-Body { "entryId": "<Entry ID>" }

*Status 202 - Registration is running.*

-Header Location: <Base URL>/operations/<Operation ID>

-Header Content-Type: application/json
-(Optional) Header: Retry-After: <seconds>

-Body { "operationId": "<Operation ID>" , "status": "running" }

*Status 400 - Registration was unsuccessful due to invalid input.  
-Error code badSignatureAlgorithm

-TBD: more error codes to be defined, see #17

If 202 is returned, then clients should wait until Registration succeeded or failed by polling the Registration status using the Operation ID returned in the response. Clients should always obtain a Receipt as a proof that Registration has succeeded.

8.1.2. Retrieve Operation Status

8.1.2.1. Request

GET <Base URL>/operations/<Operation ID>

8.1.2.2. Response

One of the following:

*Status 200 - Registration is running

-Header: Content-Type: application/json

-(Optional) Header: Retry-After: <seconds>

-Body: { "operationId": "<Operation ID>" , "status": "running" }

*Status 200 - Registration was successful

-Header:Location: <Base URL>/entries/<Entry ID>

-Header: Content-Type: application/json

-Body: { "operationId": "<Operation ID>" , "status": "succeeded" , "entryId": "<Entry ID>" }

*Status 200 - Registration failed

-Header Content-Type: application/json

-Body: { "operationId": "<Operation ID>" , "status": "failed" , "error": { "type": "<type>" , "detail": "<detail>" } }

-Error code: badSignatureAlgorithm
*Status 404 - Unknown Operation ID

-Error code: operationNotFound

-This can happen if the operation ID has expired and been deleted.

If an operation failed, then error details **SHOULD** be embedded as a JSON problem details object in the "error" field.

If an operation ID is invalid (i.e., it does not correspond to any submit operation), a service may return either a 404 or a running status. This is because differentiating between the two may not be possible in an eventually consistent system.

8.1.3. **Retrieve Signed Statement**

8.1.3.1. **Request**

GET <Base URL>/entries/<Entry ID>

Query parameters:

*(Optional) embedReceipt=true

If the query parameter embedReceipt=true is provided, then the Signed Statement is returned with the corresponding Registration Receipt embedded in the COSE unprotected header.

8.1.3.2. **Response**

One of the following:

*Status 200.

-Header: Content-Type: application/cose

-Body: COSE_Sign1

*Status 404 - Entry not found.

-Error code: entryNotFound

8.1.4. **Retrieve Registration Receipt**

8.1.4.1. **Request**

GET <Base URL>/entries/<Entry ID>/receipt
8.1.4.2. Response

One of the following:

*Status 200.
   - Header: Content-Type: application/cbor
   - Body: SCITT_Receipt

*Status 404 - Entry not found.
   - Error code: entryNotFound

The retrieved Receipt may be embedded in the corresponding COSE_Sign1 document in the unprotected header.

9. Privacy Considerations

Unless advertised by a Transparency Service, every Issuer should treat Signed Statements it registered (rendering them Transparent Statements) as public. In particular, Signed Statements' Envelopes and Statement payload should not carry any private information in plaintext.

10. Security Considerations

On its own, verifying a Transparent Statement does not guarantee that its Envelope or contents are trustworthy---just that they have been signed by the apparent Issuer and counter-signed by the Transparency Service. If the Verifier trusts the Issuer, it can infer that an Issuer's Signed Statement was issued with this Envelope and contents, which may be interpreted as the Issuer saying the Artifact is fit for its intended purpose. If the Verifier trusts the Transparency Service, it can independently infer that the Signed Statement passed the Transparency Service Registration Policy and that has been persisted in the Registry. Unless advertised in the Transparency Service Registration Policy, the Verifier should not assume that the ordering of Signed Statements in the Registry matches the ordering of their issuance.

Similarly, the fact that an Issuer can be held accountable for its Transparent Statements does not on its own provide any mitigation or remediation mechanism in case one of these Transparent Statements turned out to be misleading or malicious---just that signed evidence will be available to support them.

Issuers SHOULD ensure that the Statement payloads in their Signed Statements are correct and unambiguous, for example by avoiding ill-
defined or ambiguous formats that may cause Verifiers to interpret the Signed Statement as valid for some other purpose.

Issuers and Transparency Services SHOULD carefully protect their private signing keys and avoid these keys being used for any purpose not described in this architecture document. In cases where key re-use is unavoidable, keys MUST NOT sign any other message that may be verified as an Envelope as part of a Signed Statement.

10.1. Threat Model

The document provides a generic threat model for SCITT, describing its residual security properties when some of its actors (identity providers, Issuers, Transparency Services, and Auditors) are corrupt or compromised.

This model may need to be refined to account for specific supply chains and use cases.


SCITT primarily supports checking of Signed Statement authenticity, both from the Issuer (authentication) and from the Transparency Service (transparency). These guarantees are meant to hold for extensive periods of time, possibly decades.

It can never be assumed that some Issuers and some Transparency Services will not be corrupt.

SCITT entities explicitly trust one another on the basis of their long-term identity, which maps to shorter-lived cryptographic credentials. Hence, a Verifier would usually validate a Transparent Statement originating from a given Issuer, registered at a given Transparency Service (both identified in the Verifier's local authorization policy) and would not depend on any other Issuer or Transparency Services.

Authorized supply chain actors (Issuers) cannot be stopped from producing Signed Statements including false assertions in their Statement payload (either by mistake or by corruption), but these Issuers can made accountable by ensuring their Signed Statements are systematically registered at a trustworthy Transparency Service.

Similarly, providing strong residual guarantees against faulty/corrupt Transparency Services is a SCITT design goal. Preventing a Transparency Service from registering Signed Statements that do not meet its stated Registration Policy, or to issue Receipts that are not consistent with their append-only Log is not possible. In contrast Transparency Services can be hold accountable and they can be called out by any Auditor that replays their Registry against any
contested Receipt. Note that the SCITT Architecture does not require trust in a single centralized Transparency Service: different actors may rely on different Transparency Services, each registering a subset of Signed Statements subject to their own policy.

In both cases, the SCITT Architecture provides generic, universally-verifiable cryptographic proof to individually blame Issuers or the Transparency Service. On one hand, this enables valid actors to detect and disambiguate malicious actors who issue contradictory Signed Statements to different entities (Verifiers, Auditors, Issuers), otherwise known as 'equivocation'. On the other hand, their liability and the resulting damage to their reputation are application specific, and out of scope of the SCITT Architecture.

Verifiers and Auditors need not be trusted by other actors. In particular, so long as actors maintain proper control of their signing keys and identity infrastructure they cannot "frame" an Issuer or a Transparency Service for Signed Statements they did not issue or register.

10.1.1.1. Append-only Log

If a Transparency Service is honest, then a Transparent Statement including a correct Receipt ensures that the associated Signed Statement passed its Registration Policy and was recorded appropriately.

Conversely, a corrupt Transparency Service may 1. refuse or delay the Registration of Signed Statements, 2. register Signed Statements that do not pass its Registration Policy (e.g., Signed Statement with Issuer identities and signatures that do not verify), 3. issue verifiable Receipts for Signed Statements that do not match its Registry, or 4. refuse access to its Registry (e.g., to Auditors, possibly after storage loss).

An Auditor granted (partial) access to a Registry and to a collection of disputed Receipts will be able to replay it, detect any invalid Registration (2) or incorrect Receipt in this collection (3), and blame the Transparency Service for them. This ensures any Verifier that trusts at least one such Auditor that (2,3) will be blamed to the Transparency Service.

Due to the operational challenge of maintaining a globally consistent append-only Log, some Transparency Services may provide limited support for historical queries on the Signed Statements they have registered, and accept the risk of being blamed for inconsistent Registration or Issuer equivocation.

Verifiers and Auditors may also witness (1,4) but may not be able to collect verifiable evidence for it.
10.1.1.2. Availability of Transparent Signed Statement

Networking and Storage are trusted only for availability.

Auditing may involve access to data beyond what is persisted in the Transparency Services. For example, the registered Transparency Service may include only the hash of a detailed SBOM, which may limit the scope of auditing.

Resistance to denial-of-service is implementation specific.

Actors should independently keep their own record of the Signed Statements they issue, endorse, verify, or audit.

10.1.2. Confidentiality and privacy.

According to Zero Trust Principles any location in a network is never trusted. All contents exchanged between actors is protected using secure authenticated channels (e.g., TLS) but, as usual, this may not exclude network traffic analysis.

10.1.2.1. Signed Statements and Their Registration

The Transparency Service is trusted with the confidentiality of the Signed Statements presented for Registration. Some Transparency Services may publish every Signed Statement in their logs, to facilitate their dissemination and auditing. Others may just return Receipts to clients that present Singed Statements for Registration, and disclose the Append-only Log only to Auditors trusted with the confidentiality of its contents.

A collection of Signed Statements must not leak information about the contents of other Signed Statements registered on the Transparency Service.

Nonetheless, Issuers should carefully review the inclusion of private/confidential materials in their Statements. For example, issuers should remove Personally Identifiable Information (PII) as clear text in the statement. Alternatively, Issuers may include opaque cryptographic statements, such as hashes.

10.1.2.2. Queries to the Registry

The confidentiality of queries is implementation-specific, and generally not guaranteed. For example, while offline Envelope validation of Signed Statements is private, a Transparency Service may monitor which of its Transparent Statements are being verified from lookups to ensure their freshness.
10.1.3. Cryptographic Assumptions

SCITT relies on standard cryptographic security for signing schemes (EUF-CMA: for a given key, given the public key and any number of signed messages, an attacker cannot forge a valid signature for any other message) and for Receipts schemes (log collision-resistance: for a given commitment such as a Merkle-tree root, there is a unique log such that any valid path authenticates a Signed Statement in this log.)

The SCITT Architecture supports cryptographic agility: the actors depend only on the subset of signing and Receipt schemes they trust. This enables the gradual transition to stronger algorithms, including e.g. post-quantum signature algorithms.

10.1.4. Transparency Service Clients

Trust in clients that submit Signed Statements for Registration is implementation-specific. Hence, an attacker may attempt to register any Signed Statement it has obtained, at any Transparency Service that accepts them, possibly multiple times and out of order. This may be mitigated by a Transparency Service that enforces restrictive access control and Registration Policies.

10.1.5. Identity

The identity resolution mechanism is trusted to associate long-term identifiers with their public signature-verification keys. (Transparency Services and other parties may record identity-resolution evidence to facilitate its auditing.)

If one of the credentials of an Issuer gets compromised, the SCITT Architecture still guarantees the authenticity of all Signed Statements signed with this credential that have been registered on a Transparency Service before the compromise. It is up to the Issuer to notify Transparency Services of credential revocation to stop Verifiers from accepting Signed Statements signed with compromised credentials.

The confidentiality of any identity lookup during Signed Statement Registration or Transparent Statement Verification is out of scope.

11. IANA Considerations

TBD; Section 4.

11.1. URN Sub-namespace for SCITT (urn:ietf:params:scitt)

IANA is requested to register the URN sub-namespace urn:ietf:params:scitt in the "IETF URN Sub-namespace for Registered
12. References

12.1. Normative References


12.2. Informative References


Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany

Email: henk.birkholz@sit.fraunhofer.de

Antoine Delignat-Lavaud
Microsoft Research
21 Station Road
Cambridge
CB1 2FB
United Kingdom

Email: antdl@microsoft.com

Cedric Fournet
Microsoft Research
21 Station Road
Cambridge
CB1 2FB
United Kingdom

Email: fournet@microsoft.com

Yogesh Deshpande
ARM
110 Fulbourn Road
Cambridge
CB1 9NJ
United Kingdom

Email: yogesh.deshpande@arm.com
Steve Lasker
Lasker Consulting
Seattle,
United States

Email: stevenlasker@hotmail.com