Workgroup: SCITT
Internet-Draft:
draft-ietf-scitt-architecture-05
Published: 10 February 2024
Intended Status: Standards Track
Expires: 13 August 2024
Authors: H. Birkholz
Fraunhofer SIT
A. Delignat-Lavaud
Microsoft Research
C. Fournet
Microsoft Research
Y. Deshpande
ARM
S. Lasker
DataTrails

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. Issuers 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 13 August 2024.

Copyright Notice

Table of Contents

1. Introduction

This document describes the scalable, flexible, and decentralized SCITT architecture. Its goal is to enhance auditability and accountability across supply chains.

In supply chains, artifacts travel down the chain until they are eventually consumed by someone. Consumers like to have information about the artifacts that they consume. There are many parties who publish information about artifacts: For example, the original manufacturer may provide information about the state of the artifact when it left the factory. The shipping company may add information about the transport environment of the artifact. Compliance auditors may provide information about their compliance assessment of the artifact. Security companies may publish vulnerability information

about an artifact. Consumers may even publish the fact that they consume an artifact.

SCITT provides a way for consumers to obtain this information in a way that is "transparent", that is, parties cannot lie about the information that they publish without it being detected. SCITT achieves this by having producers publish information in a Transparency Service, where consumers (also called Verifiers) can check the information.

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. Terminology

The terms defined in this section have special meaning in the context of Supply Chain Integrity, Transparency, and Trust, which are used 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.

Editor's Note:: *The label "394" is expected to be reserved by this document, in the COSE Header Parameters Registry.*

The terms "header", "payload", and "to-be-signed bytes" are defined in [RFC9052].

Append-only Log (Ledger): the verifiable append-only data structure that stores Signed Statements in a Transparency Service, often referred to by the synonym, Ledger. SCITT supports multiple Ledger and Receipt formats to accommodate different Transparency Service implementations, and the proof types associated with different types of Append-only Logs.

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

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

Envelope: metadata, created by the Issuer to produce a Signed Statement. The Envelope contains the identity of the Issuer and information about the Artifact, enabling Transparency Service Registration Policies to validate the Signed Statement. A Signed Statement is a COSE Envelope wrapped around a Statement, binding

the metadata in the Envelope to the 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).

Equivocation: a state where it is possible for a Transparency Service to provide different views of its Append-only log to Verifiers about the same Artifact [EQUIVOCATION].

Feed: A collection of Receipts, as recorded by the Transparency Service, based on filtering of properties from the envelope including, but not limited to the sub field of the CWT_Claims. Verifiers may use the Feed to ensure completeness and Non-equivocation in supply chain evidence by identifying all Transparent Statements linked to the Artifact they are evaluating.

Issuer: organizations, stakeholders, and users involved in creating or attesting to supply chain artifacts, releasing authentic Statements to a definable set of peers. An Issuer may be the owner or author of Artifacts, or an independent third party such as an auditor, reviewer or an endorser.

Non-equivocation: a state where it is impossible for a Transparency Service to provide different views of its append-only log to Verifiers about the same Artifact. Over time, an Issuer may register new Signed Statements about an Artifact in a Transparency Service with new information. However, the consistency of a collection of Signed Statements about the Artifact can be checked by all Verifiers.

Receipt: a Receipt is a cryptographic proof that a Signed Statement is recorded in the Append-only Log. Receipts are based on Signed Inclusion Proofs as described in COSE Signed Merkle Tree Proofs [I-D.draft-steele-cose-merkle-tree-proofs]. Receipts can be built on different verifiable data structures, not just binary merkle trees. Receipts consist of Transparency Service-specific inclusion proofs, a signature by the Transparency Service of the state of the Append-only Log, 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, adding to the Append-only Log, and producing a Receipt.

Registration Policy: the pre-condition enforced by the Transparency Service before registering a Signed Statement, based on information in the non-opaque header and metadata contained in its COSE Envelope.

Signed Statement:
an identifiable and non-repudiable Statement about an Artifact signed 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]). A Statement may represent a Software Bill Of Materials (SBOM) that lists the ingredients of a software Artifact, an endorsement or attestation about an Artifact, indicate the End of Life (EOL), redirection to a newer version, or any content an Issuer wishes to publish about an Artifact. The additional Statements about an Artifact are correlated by the Subject defined in the [CWT_CLAIMS] protected header. The Statement is considered opaque to Transparency Service, and **MAY** be encrypted.

Subject: This term has the same definition as in RFC8392, which relies on the definition in RFC7519. The sub (subject) claim identifies the principal that is the subject of the CWT. The claims in a CWT are normally statements about the subject. In SCITT, sub identifies the entity about which statements, and receipts are made. The subject value **MUST** either be scoped to be locally unique in the context of the Issuer or be globally unique. The processing of this claim is generally application specific. The sub value is a case-sensitive string containing a StringOrURI value. Issuer's use sub to identify the entity about which they are making Signed Statements. Transparency Services use sub to identify the entity about which they are issuing a Receipt.

Transparency Service: an entity that maintains and extends the Append-only Log, and endorses its state. A Transparency Service can be a complex distributed system, and SCITT requires the Transparency Service to provide many security guarantees about its Append-only Log. 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: organizations, stakeholders, and users involved in validating supply chain Artifacts. Verifiers consume Transparent Statements, verifying their proofs and inspecting the Statement

payload, either before using corresponding Artifacts, or later to audit an Artifact's provenance on the supply chain.

3. Definition of Transparency

In this document, the definition of transparency is intended to build over abstract notions of Append-only Logs 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 Append-only Log of a Transparency Service. By extension, the document may say an Artifact (a firmware binary) is transparent if it comes with one or more Transparent 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 verified, is subject to scrutiny and auditing by other parties. The Transparency Service provides a history of Statements, which may be made by multiple Issuers, enabling Verifiers to make informed decisions.

Transparency is implemented by providing a consistent, append-only, cryptographically verifiable, publicly available record of entries. A SCITT instance is referred to as a Transparency Service. Implementations of Transparency Services may protect their Append-only Log 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 Append-only Log. 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 Transparency Service can independently verify its consistency and review the complete list of Transparent Statements registered by each Issuer. However, the Registrations on a separate Transparency Service is generally disjoint, though it is possible to take a Transparent Statement (i.e. a Signed Statement with a Receipt in its unprotected header, from a from the first Transparency Service) and register it on another Transparency Service, where the second receipt will be over the first Receipt in the unprotected header.

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 Append-only Log, as any inconsistency can easily be pinpointed by any Auditor with read access to the Transparency Service.

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.

SCITT is a generalization of Certificate Transparency [RFC9162], which can be interpreted as a transparency architecture for the supply chain of X.509 certificates. Considering CT in terms of SCITT:

* CAs (Issuers) sign X.509 TBSCertificates (Artifacts) to produce X.509 certificates (Signed Statements)

* CAs submit the certificates to one or more CT logs (Transparency Services)

* CT logs produce Signed Certificate Timestamps (Transparent Statements)

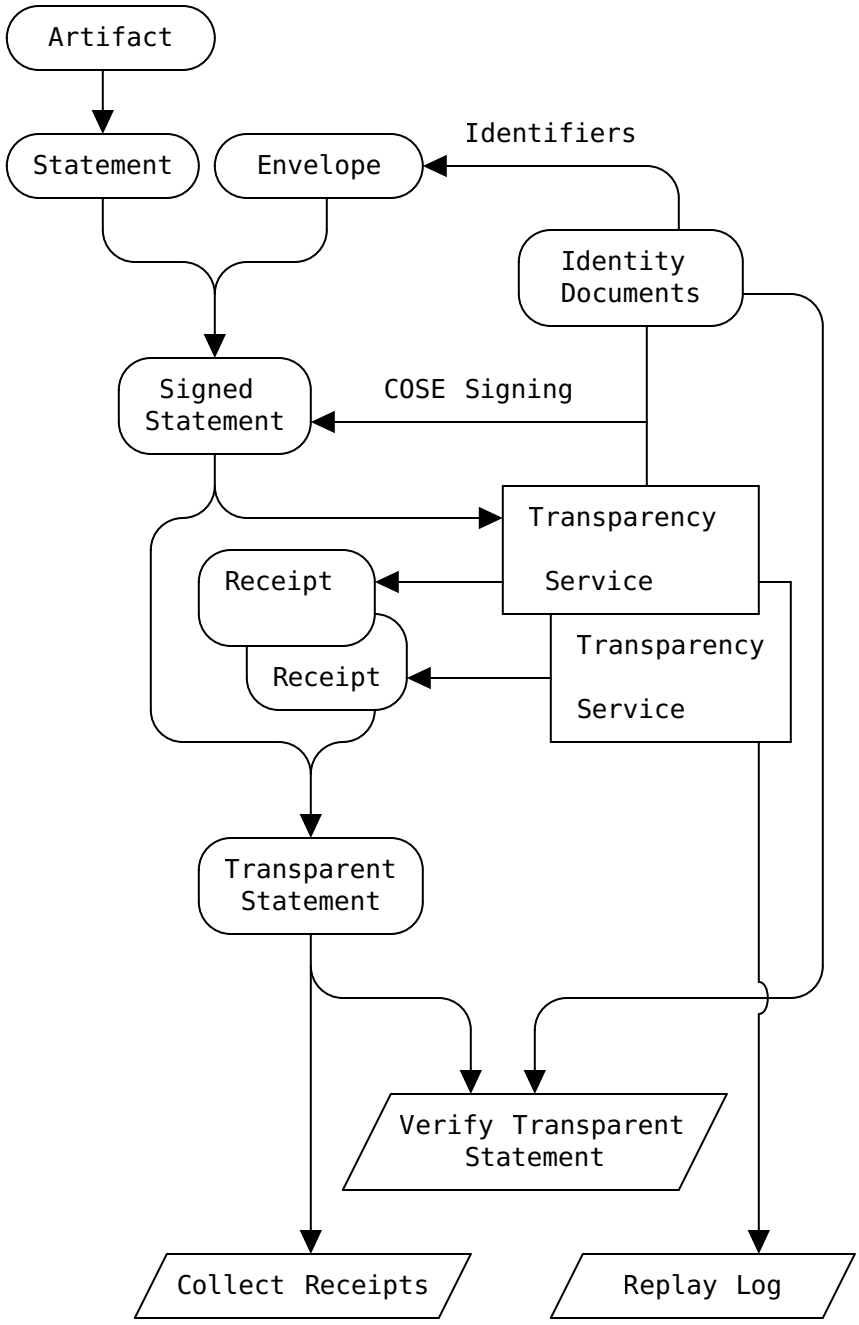
* Signed Certificate Timestamps are checked by Verifiers

* The Append-only Log can be checked by Auditors

4. 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 Signed Inclusion Proofs. Most of the details of the Receipt's contents are specified in the COSE Signed Merkle Tree Proof document [I-D.draft-steele-cose-merkle-tree-proofs].



This section describes at a high level, the three main roles and associated processes in SCITT: Issuers and Signed Statements, Transparency Service and the Signed Statement Registration process, as well as Verifiers of the Transparent Statements and the Receipt validation process.

4.1. Transparency Service

An important function of a Transparency Service is to maintain Registration Policies for the Append-only Log. The Append-only Log is the verifiable data structure which registers Signed Statements and supports the production of Receipts.

All Transparency Services **MUST** expose APIs for the registration of Signed Statements and issuance of Receipts.

Transparency Services **MAY** support additional APIs for auditing, for instance, to query the history of Signed Statements.

Typically a Transparency Services has a single Issuer identity which is present in the iss claim of Receipts for that service.

Multi-tenant support can be enabled through the use of identifiers in the iss claim, for example, ts.example may have a distinct Issuer identity for each sub domain, such as customer1.ts.example and customer2.ts.example.

4.1.1. Initialization

The Append-only Log is empty when the Transparency Service is initialized. The first entry that is added to the Append-only Log **MUST** be a Signed Statement including key material. The second set of entries are Signed Statements for additional domain-specific Registration Policy. The third set of entries are Signed Statements for Artifacts. From here on a Transparency Service can check Signed Statements on registration via policy (that is at minimum, key material and typically a Registration Policy) and is therefore in a reliable state to register Signed Statements about Artifacts or a new Registration Policy.

4.1.2. Registration Policies

Registration Policies refer to the checks that are performed before a Signed Statement is registered to an Append-only Log, and a corresponding Receipt becomes available.

As a minimum, a Transparency Service **MUST** authenticate the Issuer of Signed Statements, which requires a trust anchor in the form of an already registered Signed Statement including key material (see Section 4.1.1). 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." Typical representations of a trust anchor include certificates or raw public keys.

The x5t and kid Claims in the protected header of Signed Statements can be used as hints for discovering trust anchors. Before a Registration Policy is used to decide if a Signed Statement is registered, the policy **MUST** be registered. Before a Signed Statement is registered, the trust anchor used to verify it **MUST** be registered (e.g., via a registered Registration Policy). In order to register a trust anchor, the trust anchor **MUST** be converted to a Signed Statement with a matching content type Claim. During initialization of a Transparency Service, the first Signed Statements registered will be for a trust anchor that is not validated by any Registration Policy.

Transparency Services **MUST** specify their supported signature algorithms in their Registration Policies.

This specification leaves implementation, encoding and documentation of Registration Policies to the operator of the Transparency Service.

4.1.3. Append-only Log

The security properties of the Append-only Log are determined by the choice of the verifiable data structure used to produce Receipts.

In addition to Receipts, some verifiable data structures might support additional proof types, such as proofs of consistency, or proofs of non inclusion.

Specific verifiable data structures, such those describes in [RFC9162] and [I-D.draft-steele-cose-merkle-tree-proofs] are out of scope for this document.

4.1.4. Adjacent Services

Transparency Services can be deployed along side other database or object storage technologies. For example, a Transparency Service that is supporting a software package management system, might be referenced from the APIs exposed for package management. Providing an ability to request a fresh receipt for a given software package, or to request a list of Signed Statements and Artifacts associated with a software package.

4.2. Signed Statements

This specification prioritizes conformance to [RFC9052] and its required and optional properties. Profiles and implementation specific choices should be used to determine admissability of conforming messages. This specification is left intentionally open to allow implementations to make the restrictions that make the most sense for their operational use cases.

At least one identifier for an identity document **MUST** be included in the protected header of the COSE envelope, either x5t or kid.

*Support for x5t is mandatory to implement.

*Support for kid is optional.

When x5t is present, iss **MUST** be a string with a value between 1 and 8192 characters in length that fits the regular expression of a distinguished name.

The mechanisms for how Transparency Services obtain identity documents is out-of-scope of this document.

The kid header parameter **MUST** be present when the x5t header parameter is not present. Key discovery protocols are out-of-scope of this document.

The protected header of a Signed Statement and a Receipt **MUST** include the CWT Claims header parameter as specified in Section 2 of [CWT_CLAIMS_COSE]. The CWT Claims value **MUST** include the Issuer Claim (Claim label 1) and the Subject Claim (Claim label 2) [IANA.cwt].

A Receipt is a Signed Statement, (cose-sign1), with addition claims in its protected header related to verifying the inclusion proof in its unprotected header. See [I-D.draft-steele-cose-merkle-tree-proofs].

Figure 1 illustrates a normative CDDL definition for of the protected header for Signed Statements and Receipts.

Everything that is optional in the following CDDL can potentially be discovered out of band and Registration Policies are not assured on the presence of these optional fields. A Registration Policy that requires an optional field to be present **MUST** reject any Signed Statements or Receipts that an invalid according to the policy.

```
Signed_Statement = #6.18(COSE_Sign1)
Receipt = #6.18(COSE_Sign1)

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

Protected_Header = {
  &(CWT_Claims: 15) => CWT_Claims
  ? &(alg: 1) => int
  ? &(content_type: 3) => tstr / uint
  ? &(kid: 4) => bstr
  ? &(x5t: 34) => COSE_CertHash
  * int => any
}

CWT_Claims = {
  &(iss: 1) => tstr
  &(sub: 2) => tstr
  * int => any
}

Unprotected_Header = {
  ? &(receipts: 394)  => [+ Receipt]
}
```

Figure 1: CDDL definition for Signed Statements and Receipts

```
{                                 / Protected                   /
  1: -7,                          / Algorithm                   /
  3: application/example+json,    / Content type                /
  4: h'50685f55...50523255',      / Key identifier              /
  15: {                           / CWT Claims                  /
    1: software.vendor.example,   / Issuer                      /
    2: vendor.product.example,    / Subject                     /
  }
}
```

Figure 2: CBOR Extended Diagnostic Notation example of a Signed Statement's Protected Header

```
18(                                 / COSE Sign 1                   /
    [
      h'a4012603...6d706c65',       / Protected                     /
      {},                           / Unprotected                   /
      nil,                          / Detached payload              /
      h'79ada558...3a28bae4'        / Signature                     /
    ]
)
```

Figure 3: CBOR Extended Diagnostic Notation example of a Signed Statement

Figure 3 illustrates a payload that is detached. This is to support very large supply chain artifacts, and to ensure that Transparent Statements can integrate with existing file systems.

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 (3: payload type) to serialize the Statement payload. For a software supply chain, payloads describing the software artifacts may include:

- *[COSWID]
- *[CycloneDX]
- *[in-toto]
- *[SPDX-CBOR]
- *[SPDX-JSON]
- *[SLSA]
- *[SWID]

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).

Issuers may produce Signed Statements about different Artifacts under the same Identity. Issuers and Verifiers must be able to recognize the Artifact to which the statements pertain by looking at the Signed Statement. The iss and sub claims, within the CWT_Claims protected header, are used to identify the Artifact the statement pertains to.

See Subject under Section 2 Terminology.

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.

An Issuer can make multiple Statements about the same Artifact. For example, an Issuer can make amended Statements about the same Artifact as their view changes over time.

Multiple Issuers can make different, even conflicting Statements, about the same Artifact. Verifiers can choose which Issuers they trust.

Multiple Issuers can make the same Statement about a single Artifact, affirming multiple Issuers agree.

4.2.1. Registration

The same Signed Statement may be independently registered in multiple Transparency Services. To register a Signed Statement, the Transparency 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 Verification:** The Transparency Service **MUST** perform resolution of the Issuer's identity. This step may require that the service retrieves the Issuer ID in real-time, or rely on a cache of recent resolutions. For auditing, during Registration, the Transparency Service **MUST** store evidence of the lookup, including if it was resolved from a cache.

3. **Signature verification:** The Transparency Service **MUST** verify the signature of the Signed Statement, as described in [RFC9360], using the signature algorithm and verification key of the Issuer.

4. **Signed Statement validation:** The Transparency Service **MUST** check that the Signed Statement includes the required protected headers listed above. The Transparency Service **MAY** verify the Statement payload format, content and other optional properties.

5. **Apply Registration Policy:** The Transparency Service **MUST** check the attributes required by a policy are present in the protected headers. Custom Signed Statements are evaluated given the current Transparency Service state and the entire Envelope, and may use information contained in the attributes of named policies.

6. **Register the Signed Statement** to the append-only log.

7. **Return the Receipt**, which **MAY** be asynchronous from registration. Details about generating Receipts are described in Section 4.3.

The last two steps may be shared between a batch of Signed Statements recorded in the Append-only Log.

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 Append-only Log. Conversely, the Transparency Service **MAY** re-issue Receipts for the Append-only Log content, for instance after a transient fault during Signed Statement registration.

4.3. Transparent Statements

The client (which is not necessarily the Issuer) that registers a Signed Statement and receives a Receipt can produce a Transparent Statement by adding the Receipt to the Unprotected Header of the Signed Statement.

When a Signed Statement is registered by a Transparency Service a Receipt becomes available. When a Receipt is included in a Signed Statement a Transparent Statement is produced.

Receipts are based on Signed Inclusion Proofs as described in COSE Signed Merkle Tree Proofs ([I-D.draft-steele-cose-merkle-tree-proofs]).

The registration time is defined as the timestamp at which the Transparency Service has added this Signed Statement to its Append-only Log.

Editor's Note: The WG is discussing if existing CWT claims might better support these design principles.

Figure 4 illustrates a normative CDDL definition of Transparent Statements.

```
Transparent_Statement = #6.18(COSE_Sign1)

Unprotected_Header = {
  &(receipts: 394)  => [+ Receipt]
}
```

Figure 4: CDDL definition for a Transparent Statement

```
18(                                 / COSE Sign 1                   /
    [
      h'a4012603...6d706c65',       / Protected                     /
      {                             / Unprotected                   /
        394: [                      / Receipts (1)                  /
          h'd284586c...4191f9d2'    / Receipt 1                     /
        ]
      },
      nil,                          / Detached payload              /
      h'79ada558...3a28bae4'        / Signature                     /
    ]
)
```

Figure 5: CBOR Extended Diagnostic Notation example of a Transparent Statement

Figure 5 illustrates a payload that is detached.

The unprotected header can contain multiple receipts.

```
{                                   / Protected                     /
  1: -7,                            / Algorithm                     /
  4: h'50685f55...50523255',        / Key identifier                /
  -111: 1,                          / Verifiable Data Structure     /
  15: {                             / CWT Claims                    /
    1: transparency.vendor.example, / Issuer                        /
    2: vendor.product.example,      / Subject                       /
  }
}
```

Figure 6: CBOR Extended Diagnostic Notation example of a Receipt's Protected Header

Notice the verifiable data structure used is RFC9162_SHA256 in this case. We know from the COSE Verifiable Data Structure Registry that RFC9162_SHA256 is value 1, and that it supports -1 (inclusion proofs) and -2 (consistency proofs).

```
18(                                 / COSE Sign 1                   /
    [
      h'a4012604...6d706c65',       / Protected                     /
      {                             / Unprotected                   /
        -222: {                     / Proofs                        /
          -1: [                     / Inclusion proofs (1)          /
            h'83080783...32568964', / Inclusion proof 1             /
          ]
        },
      },
      nil,                          / Detached payload              /
      h'10f6b12a...4191f9d2'        / Signature                     /
    ]
)
```

Figure 7: CBOR Extended Diagnostic Notation example of a Receipt

Notice the unprotected header contains verifiable data structure proofs, see the protected header for details regarding the specific verifiable data structure used.

```
[                                   / Inclusion proof 1             /
  8,                                / Tree size                     /
  7,                                / Leaf index                    /
  [                                 / Inclusion hashes (3)          /
     h'c561d333...f9850597'         / Intermediate hash 1           /
     h'75f177fd...2e73a8ab'         / Intermediate hash 2           /
     h'0bdaaed3...32568964'         / Intermediate hash 3           /
  ]
]
```

Figure 8: CBOR Extended Diagnostic Notation example of a Receipt's Inclusion Proof

This is a decoded inclusion proof for RFC9162_SHA256, other verifiable data structures might encode inclusion proofs differently.

4.3.1. Validation

Verifiers **MUST** apply the verification process as described in Section 4.4 of RFC9052.

In order to verify the inclusion proof that is included in the Receipt, the verification process for the inclusion proof **MUST** be performed as described in the document that registers corresponding Verifiable Data Structure Parameters (see [I-D.draft-steele-cose-merkle-tree-proofs]).

APIs exposing verification logic for Transparent Statements may wish to provide more details that a single boolean result, for example, indicating if the signature on the Receipt or Signed Statement is valid, if claims related to the validity period are valid, or if the inclusion proof in the Receipt is valid.

The algorithm-specific details of checking inclusion proofs are covered in [I-D.draft-steele-cose-merkle-tree-proofs]. The pseudo-code for validation of a transparent statement is as follows:

```
let verify_transparent_statement(t) =
  let receipt = t.unprotected.scitt-receipt
  let version = receipt.protected.scitt-version or fail "Missing SCITT R
  assert(version == 1)

  let leaf = COSE.serialize(t with .unprotected = {
    334 => receipt.unprotected.statement-registration-info
  })

  let vds = receipt.protected.verifiable-data-structure of fail "Missing
  let root = verify_inclusion_proof(vds, receipt.unprotected.scitt-inclu
    or fail "Failed to verify inclusion proof"

  // Statement registration info has been authenticated by the inclusion
  receipt.protected.statement-registration-info = receipt.unprotected.st
  return COSE.verify(receipt, detached_payload=root)
```

Before checking a Transparent Statement, the Verifier must be configured with one or more identities of trusted Transparency Services.

Verifiers **MAY** be configured to re-verify the Issuer's Signed Statement locally, but this requires a fresh resolution of the Issuer's verification keys, which **MAY** fail if the key has been revoked.

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 Transparent Statement has been verified and validated. Such policies may use as input all information in the Envelope, the Receipt, and the Statement payload, as well as any local state.

Verifiers **MAY** offer options to store or share the Receipt of the Transparent Statement for auditing the Transparency Services in case a dispute arises.

5. Federation

Editor's Note: 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 are either unaware of the other or 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.

For example, a software producer of a supply chain artifact might rely on multiple independent software producers operating transparency services for their upstream artifacts. Downstream producers benefit from upstream producers providing higher transparency regarding their artifacts.

6. Privacy Considerations

Transparency Services are often publicly accessible. Issuers should treat Signed Statements (rendering them as Transparent Statements) as publicly accessible. In particular, a Signed Statement Envelope and Statement payload should not carry any private information in plaintext.

Transparency Services can have an authorization policy controlling who can access the Append-only Log. While this can be used to limit who can read the Log, it may also limit the usefulness of the system.

Some jurisdictions have a Right to be Forgotten. However, once a Signed Statement is inserted into the Append-only Log maintained by a Transparency Service, it cannot be removed from the Log.

7. 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 Append-only Log. Unless advertised in the Transparency Service Registration Policy, the Verifier cannot assume that the ordering of Signed Statements in the Append-only Log 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.

An Issuer that knows of a changed state of quality for an Artifact, **SHOULD** Register a new Signed Statement, using the same 15 CWT iss and sub claims.

Issuers **MUST** 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 **MUST** 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.

Each of these functions **MUST** 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 the Registration Policy evaluation and endorsement may be protected by running in a Trusted Execution Environment (TEE).

The Transparency Service may be replicated with a consensus algorithm, such as Practical Byzantine Fault Tolerance [PBFT] and may be used to protect against malicious or vulnerable replicas. Threshold signatures may be use to protect the service key, etc.

Issuers and Transparency Services **MUST** rotate verification keys for signature checking in well-defined cryptoperiods (see [KEY-MANAGEMENT]).

A Transparency Service **MAY** provide additional authenticity assurances about its secure implementation and operation, enabling remote attestation of the hardware platforms and/or software Trusted Computing Bases (TCB) that run the Transparency Service. If present, these additional authenticity assurances **MUST** be registered in the Append-only Log and **MUST** always be exposed by the Transparency

Services' APIs. An example of Signed Statement's payloads that can improve authenticity assurances are trustworthiness assessments that are RATS Conceptual Messages, such as Evidence, Endorsements, or corresponding Attestation Results (see [RFC9334]).

For example, if a Transparency Service is implemented using a set of redundant replicas, each running within its own hardware-protected trusted execution environments (TEEs), then each replica can provide fresh Evidence or fresh Attestation Results about its TEEs. The respective Evidence can show, for example, the binding of the hardware platform to the software that runs the Transparency Service, the long-term public key of the service, or the key used by the replica for signing Receipts. The respective Attestation Result, for example, can show that the remote attestation Evidence was appraised by a trusted Verifier and complies with well-known Reference Values and Endorsements.

7.1. Security Guarantees

SCITT provides the following security guarantees:

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

2. Statement provenance and history can be independently and consistently audited

3. Issuers can efficiently prove that their Statement is logged by a Transparency Service

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 Append-only Log. The third guarantee is achieved by implementing the Append-only Log using a verifiable data structure (such as a Merkle Tree [MERKLE]).

7.2. 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. A Verifier **SHOULD** 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 held accountable and they can be called out by any Auditor that replays their Append-only Log 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 employ Equivocation with Signed Statements to different entities. 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.

7.2.1. 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 Append-only Log
4. refuse access to its Transparency Service (e.g., to Auditors, possibly after storage loss)

An Auditor granted (partial) access to a Transparency Service 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.

7.2.2. Availability of Receipts

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 may want to independently keep their own record of the Signed Statements they issue, endorse, verify, or audit.

7.2.3. 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 may not exclude network traffic analysis.

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.

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

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.

7.2.4. Cryptographic Agility

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.

7.2.5. Transparency Service Clients

Trust in clients that submit Signed Statements for Registration is implementation-specific. 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.

7.2.6. Impersonation

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.

8. IANA Considerations

TBD; Section 3.

8.1. Media Type Registration

This section requests registration of the following media types [RFC2046] in the "Media Types" registry [IANA.media-types] in the manner described in [RFC6838].

To indicate that the content is an scitt configuration represented as JSON:

- *Type name: application
- *Subtype name: scitt-configuration+json
- *Required parameters: n/a
- *Optional parameters: n/a
- *Encoding considerations: binary; application/scitt-configuration+json values are represented as a JSON Object; UTF-8 encoding **SHOULD** be employed for the JSON object.
- *Security considerations: See the Security Considerations section of TBD.
- *Interoperability considerations: n/a
- *Published specification: TBD
- *Applications that use this media type: TBD
- *Fragment identifier considerations: n/a

*Additional information:

-Magic number(s): n/a

-File extension(s): n/a

-Macintosh file type code(s): n/a

*Person & email address to contact for further information: TBD

*Intended usage: COMMON

*Restrictions on usage: none

*Author: TBD

*Change Controller: IETF

*Provisional registration? No

9. References

9.1. Normative References

[COSWID] Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", RFC 9393, DOI 10.17487/RFC9393, June 2023, <https://www.rfc-editor.org/rfc/rfc9393>.

[CWT_CLAIMS_COSE] Looker, T. and M. B. Jones, "CBOR Web Token (CWT) Claims in COSE Headers", Work in Progress, Internet-Draft, draft-ietf-cose-cwt-claims-in-headers-10, 29 November 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-cwt-claims-in-headers-10>.

[IANA.cwt] IANA, "CBOR Web Token (CWT) Claims", <http://www.iana.org/assignments/cwt>.

[IANA.media-types] IANA, "Media Types", <http://www.iana.org/assignments/media-types>.

[IANA.named-information] IANA, "Named Information", <http://www.iana.org/assignments/named-information>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI

10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/rfc/rfc2046>.

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

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

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/rfc/rfc6570>.

[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://www.rfc-editor.org/rfc/rfc6838>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[RFC9360] Schaad, J., "CBOR Object Signing and Encryption (COSE): Header Parameters for Carrying and Referencing X.509 Certificates", RFC 9360, DOI 10.17487/RFC9360, February 2023, <https://www.rfc-editor.org/rfc/rfc9360>.

9.2. Informative References

[CWT_CLAIMS] "CBOR Web Token (CWT) Claims", n.d., <https://www.iana.org/assignments/cwt/cwt.xhtml>.

[CycloneDX] "CycloneDX", n.d., <https://cyclonedx.org/specification/overview/>.

[EQUIVOCATION] Chun, B., Maniatis, P., Shenker, S., and J. Kubiatowicz, "Attested append-only memory: making adversaries stick to their word", Association for Computing Machinery (ACM), ACM SIGOPS Operating Systems Review vol. 41, no. 6, pp. 189-204, DOI

10.1145/1323293.1294280, October 2007, <https://doi.org/10.1145/1323293.1294280>.

[I-D.draft-ietf-core-href] Bormann, C. and H. Birkholz, "Constrained Resource Identifiers", Work in Progress, Internet-Draft, draft-ietf-core-href-14, 9 January 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-core-href-14>.

[I-D.draft-steele-cose-merkle-tree-proofs] Steele, O., Birkholz, H., Delignat-Lavaud, A., and C. Fournet, "Concise Encoding of Signed Merkle Tree Proofs", Work in Progress, Internet-Draft, draft-steele-cose-merkle-tree-proofs-01, 10 July 2023, <https://datatracker.ietf.org/doc/html/draft-steele-cose-merkle-tree-proofs-01>.

[in-toto] "in-toto", n.d., <https://in-toto.io/>.

[KEY-MANAGEMENT] Barker, E. and W. Barker, "Recommendation for key management:: part 2 -- best practices for key management organizations", National Institute of Standards and Technology, DOI 10.6028/nist.sp.800-57pt2r1, May 2019, <https://doi.org/10.6028/nist.sp.800-57pt2r1>.

[MERKLE] Merkle, R., "A Digital Signature Based on a Conventional Encryption Function", Springer Berlin Heidelberg, Advances in Cryptology — CRYPTO '87 pp. 369-378, DOI 10.1007/3-540-48184-2_32, ISBN ["9783540187967", "9783540481843"], 1988, <https://doi.org/10.1007/3-540-48184-2_32>.

[PBFT] Castro, M. and B. Liskov, "Practical byzantine fault tolerance and proactive recovery", Association for Computing Machinery (ACM), ACM Transactions on Computer Systems vol. 20, no. 4, pp. 398-461, DOI 10.1145/571637.571640, November 2002, <https://doi.org/10.1145/571637.571640>.

[RFC2397] Masinter, L., "The "data" URL scheme", RFC 2397, DOI 10.17487/RFC2397, August 1998, <https://www.rfc-editor.org/rfc/rfc2397>.

[RFC6024] Reddy, R. and C. Wallace, "Trust Anchor Management Requirements", RFC 6024, DOI 10.17487/RFC6024, October 2010, <https://www.rfc-editor.org/rfc/rfc6024>.

[RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <https://www.rfc-editor.org/rfc/rfc8141>.

[RFC9162] Laurie, B., Messeri, E., and R. Stradling, "Certificate Transparency Version 2.0", RFC 9162, DOI 10.17487/RFC9162, December 2021, <https://www.rfc-editor.org/rfc/rfc9162>.

[RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <https://www.rfc-editor.org/rfc/rfc9334>.

[SLSA] "SLSA", n.d., <https://slsa.dev/>.

[SPDX-CBOR] "SPDX Specification", n.d., <https://spdx.dev/use/specifications/>.

[SPDX-JSON] "SPDX Specification", n.d., <https://spdx.dev/use/specifications/>.

[SWID] "SWID Specification", n.d., <https://csrc.nist.gov/Projects/Software-Identification-SWID/guidelines>.

[URLs] "URL Living Standard", n.d., <https://url.spec.whatwg.org/>.

Appendix A. Identifiers

This section provides informative examples of identifiers for statements, signed statements, and receipts.

SCITT Identifiers are primarily meant to be understood by humans and secondarily meant to be understood by machines, as such we define text encodings for message identifiers first, and then provide binary translations according to standard transformations for URLs and URNs to binary formats.

SCITT Identifiers for URLs and URNs that are not Data URLs **MUST** be represented in binary using [I-D.draft-ietf-core-href].

For each SCITT conceptual message, we define a Data URL format according to [RFC2397], a URN format according to [RFC8141] and a URL format according to [URLs].

Note that Data URLs require base64 encoding, but the URN definitions require base64url encoding.

Resolution and dereferencing of these identifiers is out of scope for this document, and can be implemented by any concrete api implementing the abstract interface defined as follows:

```
resource: content-type = dereference(identifier: identifier-type)
```

These identifiers **MAY** be present in a tstr field that does not otherwise restrict the string in ways that prevent a URN or URL from being present.

This includes iss, and sub which are used to express the Issuer and subject of a signed statement or receipt.

This also includes kid which is used to express a hint for which public key should be used to verify a signature.

All SCITT identifiers share common parameters to promote interoperability:

Let hash-name be an algorithm name registered in [IANA.named-information].

To promote interoperability, the hash-name **MUST** be "sha-256".

Let base-encoding, be a base encoding defined in [RFC4648].

To promote interoperability, the base encoding **MUST** be "base64url".

In the blocks and examples that follow, note '' line wrapping per RFC 8792.

A.1. For Binary Content

Identifiers for binary content, such as Statements, or even Artifacts themselves are computed as follows:

Let the base64url-encoded-bytes-digest for the message be the base64url encoded digest with the chosen hash algorithm of bytes / octets.

Let the SCITT name for the message be the URN constructed from the following URI template, according to [RFC6570]:

Let the message-type, be "statement" for Statements and Artifacts.

```
urn:ietf:params:scitt:\
{message-type}:\
{hash-name}:{base-encoding}:\
{base64url-encoded-bytes-digest}
```

A.2. For SCITT Messages

Identifiers for COSE Sign 1 based messages, such as identifiers for Signed Statements and Receipts are computed as follows:

Let the base64url-encoded-to-be-signed-bytes-digest for the message be the base64url encoded digest with the chosen hash algorithm of the "to-be-signed bytes", according to Section 8.1 of [RFC9052].

Let the SCITT name for the message be the URN constructed from the following URI template, according to [RFC6570]:

Let the message-type, be "signed-statement" for Signed Statements, and "receipt" for Receipts.

```
urn:ietf:params:scitt:\
{message-type}:\
{hash-name}:{base-encoding}:\
{base64url-encoded-to-be-signed-bytes-digest}
```

Note that this means the content of the signature is not included in the identifier, even though signature related claims, such as activation or expiration information in protected headers are included.

As a result, an attacker may construct a new signed statement that has the same identifier as a previous signed statement, but has a different signature.

A.3. For Transparent Statements

Identifiers for Transparent Statements are defined as identifiers for binary content, but with "transparent-statement" as the message-type.

```
urn:ietf:params:scitt:\
{message-type}:\
{hash-name}:{base-encoding}:\
{base64url-encoded-bytes-digest}
```

Note that because this identifier is computed over the unprotected header of the Signed Statement, any changes to the unprotected header, such as changing the order of the unprotected header map key value pairs, adding additional receipts, or adding additional proofs to a receipt, will change the identifier of a transparent statement.

Note that because this identifier is computed over the signatures of the signed statement and signatures in each receipt, any canonicalization of the signatures after the fact will produce a distinct identifier.

A.4. Statements

A.4.1. Statement URN

```
urn:ietf:params:scitt:statement:sha-256:base64url:5i6UeRzg1...qnGmr1o
```

Figure 9: Example Statement URN

A.4.2. Statement URL

```
https://transparency.example/api/identifiers\
/urn:ietf:params:scitt:statement:sha-256:base64url:5i6UeRzg1...qnGmr1o
```

Figure 10: Example Statement URL

A.4.3. Statement Data URL

```
data:application/json;base64,SGVsb...xkIQ==
```

Figure 11: Example Statement Data URL

A.5. Signed Statements

A.5.1. Signed Statement URN

```
urn:ietf:params:scitt:\
signed-statement:sha-256:base64url:5i6UeRzg1...qnGmr1o
```

Figure 12: Example Signed Statement URN

A.5.2. Signed Statement URL

```
https://transparency.example/api/identifiers\
/urn:ietf:params:scitt:\
signed-statement:sha-256:base64url:5i6UeRzg1...qnGmr1o
```

Figure 13: Example Signed Statement URL

A.5.3. Signed Statement Data URL

```
data:application/cose;base64,SGVsb...xkIQ==
```

Figure 14: Example Signed Statement Data URL

A.6. Receipts

A.6.1. Receipt URN

```
urn:ietf:params:scitt:receipt:sha-256:base64url:5i6UeRzg1...qnGmr1o
```

Figure 15: Example Receipt URN

A.6.2. Receipt URL

```
https://transparency.example/api/identifiers\
/urn:ietf:params:scitt:receipt:sha-256:base64url:5i6UeRzg1...qnGmr1o
```

Figure 16: Example Receipt URL

A.6.3. Receipt Data URL

```
data:application/cose;base64,SGVsb...xkIQ==
```

Figure 17: Example Receipt Data URL

A.7. Transparent Statements

A.7.1. Transparent Statement URN

```
urn:ietf:params:scitt:\
transparent-statement:sha-256:base64url:5i6UeRzg1...qnGmr1o
```

Figure 18: Example Transparent Statement URN

A.7.2. Transparent Statement URL

```
https://transparency.example/api/identifiers\
/urn:ietf:params:scitt:\
transparent-statement:sha-256:base64url:5i6UeRzg1...qnGmr1o
```

Figure 19: Example Transparent Statement URL

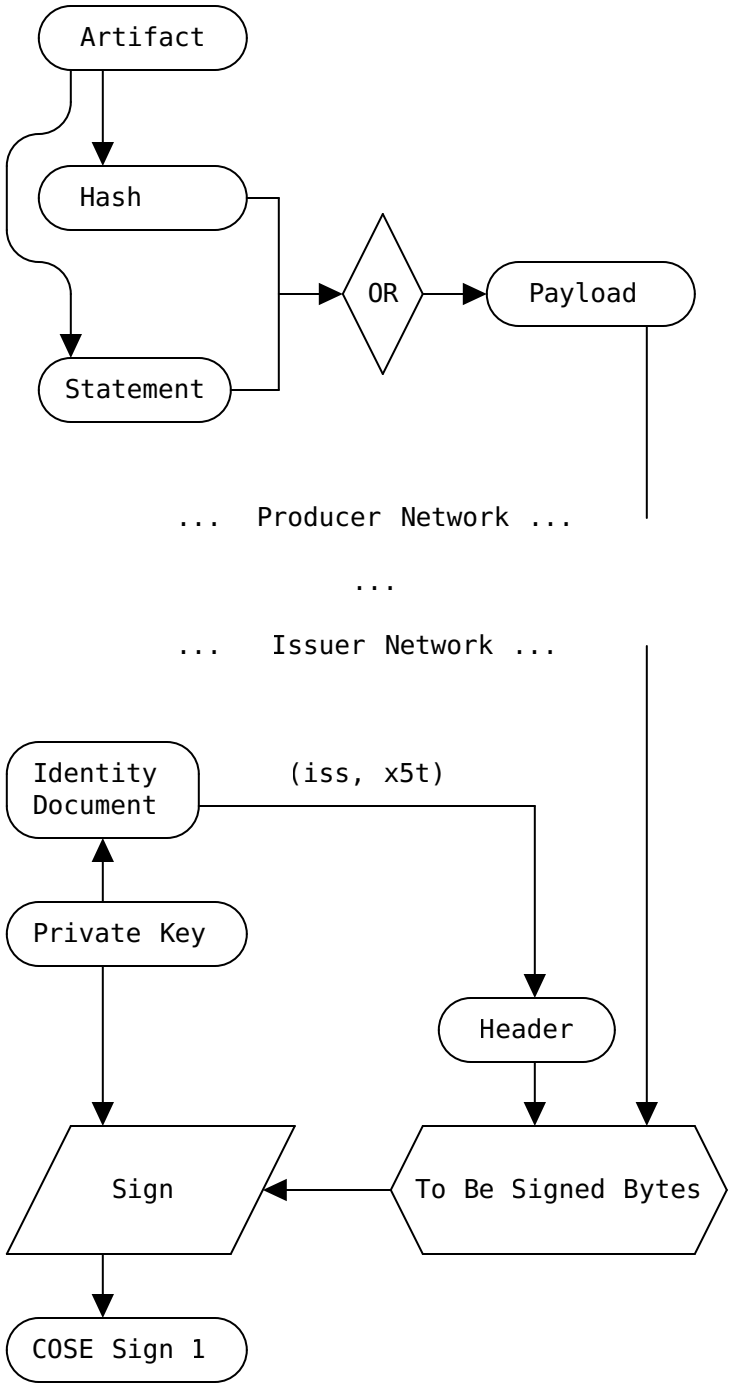
A.7.3. Transparent Statement Data URL

```
data:application/cose;base64,SGVsb...xkIQ==
```

Figure 20: Example Transparent Statement Data URL

Appendix B. Signing Statements Remotely

Statements, such as digital artifacts or structured data regarding artifacts, can be too large or too sensitive to be send to a remote Transparency Services over the Internet. In these cases a statement can also be hash, which becomes the payload included in COSE to-be-signed bytes. A Signed Statement (cose-sign1) **MUST** be produced from the to-be-signed bytes according to Section 4.4 of [RFC9052].



Contributors

Orie Steele
Transmute
United States

Email: orie@transmute.industries

Orie contributed to improving the generalization of COSE building blocks and document consistency.

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
DataTrails
Seattle, WA 98199
United States

Email: steve.lasker@datatrails.ai