Stream: Internet Engineering Task Force (IETF)
RFC: 9943
Category: Standards Track
Published: March 2026
ISSN: 2070-1721
Authors:
H. Birkholz, *Fraunhofer SIT*
A. Delignat-Lavaud, *Microsoft Research*
C. Fournet, *Microsoft Research*
Y. Deshpande, *ARM*
S. Lasker

RFC 9943
An Architecture for Trustworthy and Transparent Digital Supply Chains

Abstract

Traceability in supply chains is a growing security concern. While verifiable data structures have addressed specific issues, such as equivocation over digital certificates, they lack a universal architecture for all supply chains. This document defines such an architecture for single-issuer signed statement transparency. It ensures extensibility and interoperability between different transparency services as well as compliance with various auditing procedures and regulatory requirements.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9943.

Copyright Notice

Table of Contents

1. Introduction

This document defines an architecture, a base set of extensible message structures, and associated flows to make signed content transparent via verifiable data structures maintained by corresponding transparency services. The goal of the transparency enabled by the Supply Chain Integrity, Transparency, and Trust (SCITT) architecture is to enhance auditability and accountability for single-issuer signed content (statements) that are about supply chain commodities (artifacts).

Registering signed statements with a transparency service is akin to a notarization procedure. Transparency Services (TSs) confirm a policy is met before recording the statement on the ledger. The SCITT ledger represents a linear and irrevocable history of statements made. Once the signed statement is registered, the transparency service issues a receipt.

Similar approaches have been implemented for specific classes of artifacts, such as Certificate Transparency (CT) [RFC9162]. The SCITT approach follows a more generic paradigm than previous approaches. This "content-agnostic" approach allows SCITT transparency services to be

either integrated in existing solutions or an initial part of new emerging systems. Extensibility is a vital feature of the SCITT architecture, so that requirements from various applications can be accommodated while always ensuring interoperability with respect to registration procedures and corresponding auditability and accountability. For simplicity, the scope of this document is limited to use cases originating from the software supply chain domain. However, the specification defined is applicable to any other type of supply chain statements (also referred to as "value-add graphs"), for example, statements about hardware supply chains.

This document also defines message structures for signed statements and transparent statements, which embed CBOR Object Signing and Encryption (COSE) receipts [RFC9942], i.e., signed verifiable data structure proofs). These message structures are based on the Concise Binary Object Representation (CBOR) Standard [STD94] and corresponding signing is facilitated via the COSE Standard [STD96]. The message structures are defined using the Concise Data Definition Language (CDDL) [RFC8610]. The signed statements and receipts are, respectively, based on the COSE_Sign1 specification in Section 4.2 of RFC 9052 [STD96] and on COSE receipts [RFC9942]. The application-domain-agnostic nature of COSE_Sign1 and its extensibility through COSE Header Parameters are prerequisites for implementing the interoperable message layer defined in this document.

In summary, this specification supports relying parties obtaining proof that signed statements were recorded and checked for their validity at the time they were registered. How these statements are managed or stored as well as how participating entities discover and notify each other of changes is out of scope of this document.

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. Software Supply Chain Scope

To illustrate the applicability of the SCITT architecture and its messages, this section details the exemplary context of Software Supply Chain (SSC) use cases. The building blocks provided by the SCITT architecture are not restricted to SSC use cases. SSCs serve as useful application guidance and first usage scenarios.

2.1. Generic SSC Problem Statement

Supply chain security is a prerequisite to protecting consumers and minimizing economic, public health, and safety threats. Supply chain security has historically focused on risk management practices to safeguard logistics, meet regulatory requirements, forecast demand, and optimize inventory. While these elements are foundational to a healthy supply chain, an integrated cyber-security-based perspective of the SSCs remains broadly undefined. Recently, the

global community has experienced numerous supply chain attacks targeting weaknesses in SSCs. As illustrated in Figure 1, an SSC attack may leverage one or more life-cycle stages and directly or indirectly target the component.

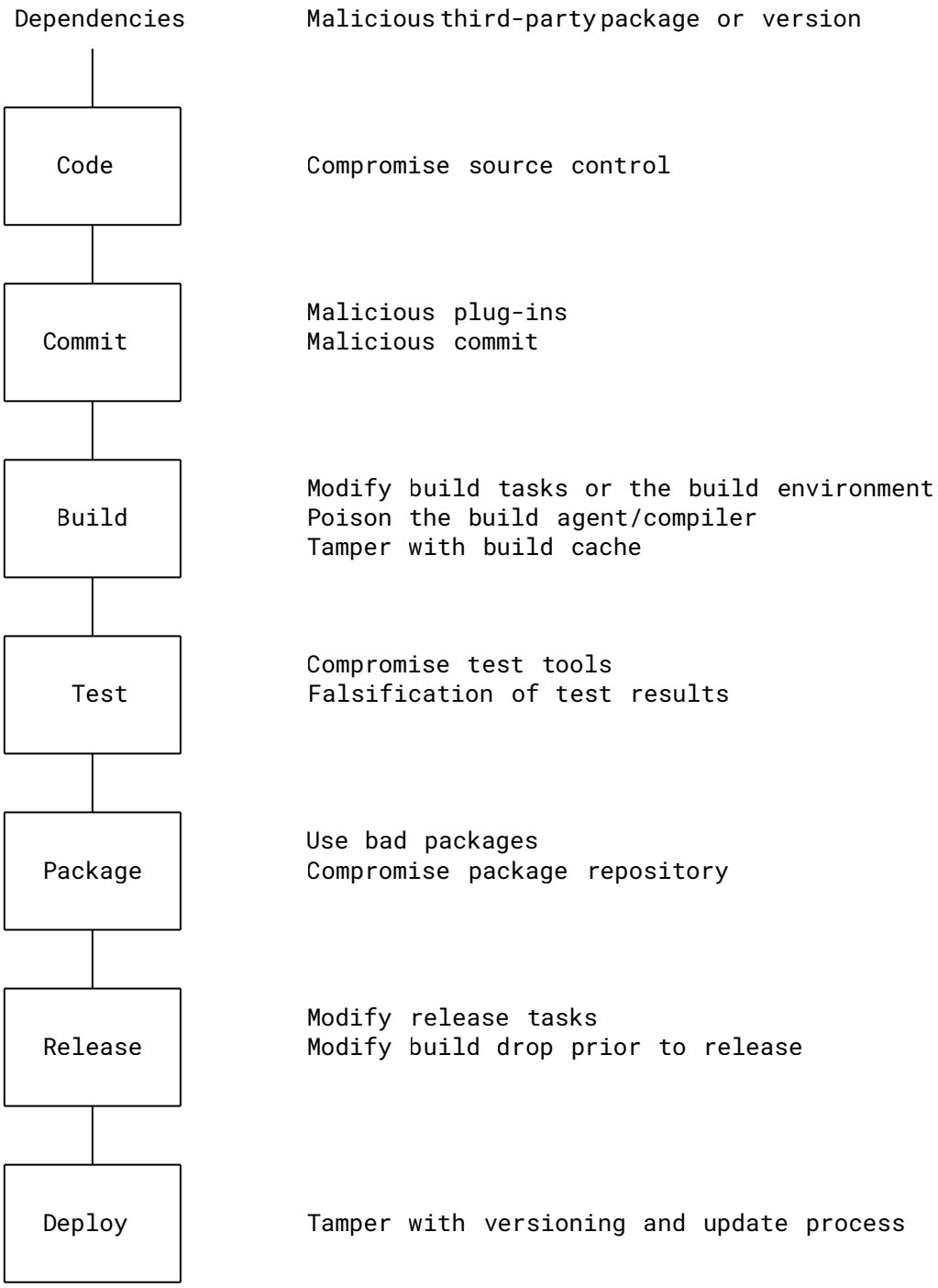


Figure 1: Example SSC Life-Cycle Threats

DevSecOps, as defined in [NIST.SP.800-204C], often depends on third-party and open-source software. These dependencies can be quite complex throughout the supply chain, so checking provenance and traceability throughout their life cycle is difficult. There is a need for manageable auditability and accountability of digital products. Typically, the range of types of statements about digital products (and their dependencies) is vast, heterogeneous, and can differ between community policy requirements. Taking the type and structure of all statements about digital products into account might not be possible. Examples of statements may include commit signatures, build environment and parameters, Software Bill of Materials (SBOM), static and dynamic application security testing results, fuzz testing results, release approvals, deployment records, vulnerability scan results, and patch logs. Consequently, instead of trying to understand and describe the detailed syntax and semantics of every type of statement about digital products, the SCITT architecture focuses on ensuring statement authenticity, ensuring visibility/transparency, and intends to provide scalable accessibility. Threats and practical issues can also arise from unintended side effects of using security techniques outside their proper bounds. For instance, digital signatures may fail to verify past their expiry date even though the signed item itself remains completely valid. Or a signature may verify even though the information it is securing is now found unreliable but fine-grained revocation is too hard.

Lastly, where data exchange underpins serious business decision-making, it is important to hold the producers of those data to a higher standard of auditability. The SCITT architecture provides mechanisms and structures for ensuring that the makers of authoritative statements can be held accountable and not hide or shred the evidence when it becomes inconvenient later.

The following use cases illustrate the scope of SCITT and elaborate on the generic problem statement above.

2.2. Eclectic SSC Use Cases

The three following use cases are a specialization derived from the generic problem statement above.

2.2.1. Security Analysis of a Software Product

A released software product is often accompanied by a set of complementary statements about its security properties. This gives enough confidence to both producers and consumers that the released software meets the expected security standards and is suitable to use.

Subsequently, multiple security researchers often run sophisticated security analysis tools on the same product. The intention is to identify any security weaknesses or vulnerabilities in the package.

Initially, a particular analysis can identify a simple weakness in a software component. Over a period of time, a statement from a third party illustrates that the weakness is exposed in a way that represents an exploitable vulnerability. The producer of the software product provides a statement that confirms the linking of a software component vulnerability with the software product by issuing a product vulnerability disclosure report and also issuing an advisory statement on how to mitigate the vulnerability. At first, the producer provides an updated software product that still uses the vulnerable software component but shields the issue in a

fashion that inhibits exploitation. Later, a second update of the software product includes a security patch to the affected software component from the software producer. Finally, a third update includes a new release (updated version) of the formerly insecure software component. For this release, both the software product and the affected software component are deemed secure by the producer and consumers.

A consumer of a released software wants to:

- know where to get these security statements from producers and third parties related to the software product in a timely and unambiguous fashion
- attribute them to an authoritative issuer
- associate the statements in a meaningful manner via a set of well-known semantic relationships
- consistently, efficiently, and homogeneously check their authenticity

SCITT provides a standardized way to:

- know the various sources of statements
- express the provenance and historicity of statements
- relate and link various heterogeneous statements in a simple fashion
- check that the statement comes from a source with authority to issue that statement
- confirm that sources provide a complete history of statements related to a given component

2.2.2. Promotion of a Software Component by Multiple Entities

A software component (e.g., a library or software product), open-source or commercial, is often initially released by a single trusted producer who can choose to attach a statement of authenticity to it. As that component becomes used in a growing range of other products, providers other than the original trusted producer often re-distribute or release their own version of that component.

Some providers include it as part of their release product or package bundle and provide the package with proof of authenticity using their issuer authority. Some packages include the original statement of authenticity, and some do not. Over time, some providers no longer offer the exact same software component source code but pre-compiled software component binaries. Some sources do not provide the exact same software component but include patches and fixes produced by third parties as these emerge faster than solutions from the original producer. Due to complex distribution and promotion life-cycle scenarios, the original software component takes myriad forms.

A consumer of a released software wants to:

- understand if a particular provider is a trusted originating producer or an alternative party
- know if and how the source, or resulting binary, of a promoted software component differs from the original software component
- check the provenance and history of a software component's source back to its origin

- assess whether to trust a component or product based on a downloaded package location and source supplier

SCITT provides a standardized way to:

- reliably discern if a provider is the original, trusted producer or is a trustworthy alternative provider or is an illegitimate provider
- track the provenance path from an original producer to a particular provider
- check the trustworthiness of a provider
- check the integrity of modifications or transformations applied by a provider

2.2.3. Software Integrator Assembling a Software Product for a Vehicle

Software Integration is a complex activity. Typically, it involves getting various software components from multiple suppliers and producing an integrated package deployed as part of device assembly. For example, car manufacturers source integrated software for their vehicles from third parties that integrate software components from various sources. Integration complexity creates a higher risk of security vulnerabilities to the delivered software.

Consumers of integrated software want:

- a list of all components present in a software product
- the ability to identify and retrieve all components from a secure and tamper-proof location
- verifiable proofs on build process and build environment with all supplier tiers to ensure end-to-end build quality and security

SCITT provides a standardized way to:

- provide a tiered and transparent framework that allows for verification of integrity and authenticity of the integrated software at both component and product level before installation
- provide valid annotations on build integrity to ensure conformance

3. Terminology

The terms defined in this section have special meaning in the context of SCITT and are used throughout this document.

This document has been developed in coordination with the COSE, OAUTH, and RATS working groups (WGs) and uses terminology common to these WGs as often as possible.

The terms "header", "payload", and "to-be-signed bytes" are defined in [STD96].

The term "claim" is defined in [RFC8392].

When used in text, the following terms are capitalized. To ensure readability, only a core set of terms is included in this section.

Append-only Log: a Statement Sequence comprising the entire registration history of the Transparency Service. To make the Append-only property verifiable and transparent, the Transparency Service defines how Signed Statements are made available to Auditors.

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, or the transparent Statement Sequence, issued by a Transparency Service. An Auditor is an example of a specialized Relying Party.

Client: an application making protected Transparency Service resource requests on behalf of the resource owner and with its authorization.

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 a Transparency Service provides inconsistent proofs to Relying Parties, containing conflicting claims about the Signed Statement bound at a given position in the Verifiable Data Structure.

Issuer: an identifier representing an organization, device, user, or entity securing Statements about supply chain Artifacts. An Issuer may be the owner or author of Artifacts or an independent third party such as an Auditor, reviewer, or endorser. In SCITT Statements and Receipts, the `iss` Claim is a member of the COSE header parameter `15: CWT Claims` defined in [RFC9597], which embeds a CBOR Web Token (CWT) Claim Set [RFC8392] within the protected header of a COSE Envelope. This document uses the terms "Issuer" and "Subject" as described in [RFC8392]; however, the usage is consistent with the broader interpretation of these terms in both JSON Object Signing and Encryption (JOSE) and COSE, and the guidance in [RFC8725] generally applies the COSE equivalent terms with consistent semantics.

Non-equivocation: a state where all proofs provided by the Transparency Service to Relying Parties are produced from a single Verifiable Data Structure describing a unique sequence of Signed Statements and are therefore consistent [EQUIVOCATION]. 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 Relying Parties.

Receipt: a cryptographic proof that a Signed Statement is included in the Verifiable Data Structure. See [RFC9942] for implementations. Receipts are signed proofs of verifiable data-structure properties. Receipt Profiles implemented by a Transparency Service **MUST** support inclusion proofs and **MAY** support other proof types, such as consistency proofs.

Registration: the process of submitting a Signed Statement to a Transparency Service, applying the Transparency Service's Registration Policy, adding to the Verifiable Data Structure, and producing a Receipt.

Registration Policy: the precondition 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.

Relying Party: Relying Parties 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.

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.

Attestation: [NIST.SP.1800-19] defines "attestation" as:

> The process of providing a digital signature for a set of measurements securely stored in hardware, and then having the requester validate the signature and the set of measurements.

NIST guidance "Software Supply Chain Security Guidance EO 14028" uses the definition from [NIST_EO14028], which states that an "attestation" is:

> The issue of a statement, based on a decision, that fulfillment of specified requirements has been demonstrated.

It is often useful for the intended audience to qualify the term "attestation" in their specific context to avoid confusion and ambiguity.

Statement: any serializable information about an Artifact. To help interpret Statements, they must be tagged with a relevant media type (as specified in [RFC6838]). A Statement may represent an SBOM that lists the ingredients of a software Artifact, contains an endorsement or attestation about an Artifact, indicates the End of Life (EOL), redirects to a newer version, or contains any content an Issuer wishes to publish about an Artifact. Additional Statements about an Artifact are correlated by the Subject Claim as defined in the IANA CWT registry [IANA.cwt] and used as a protected header parameter as defined in [RFC9597]. The Statement is considered opaque to Transparency Service and **MAY** be encrypted.

Statement Sequence: a sequence of Signed Statements captured by a Verifiable Data Structure. See Verifiable Data Structure.

Subject: an identifier, defined by the Issuer, that represents the organization, device, user, entity, or Artifact about which Statements (and Receipts) are made and by which a logical collection of Statements can be grouped. It is possible that there are multiple Statements about the same Artifact. In these cases, distinct Issuers (`iss`) might agree to use the `sub` CWT Claim, defined in [RFC8392], to create a coherent sequence of Signed Statements about the same Artifact, and Relying Parties can leverage `sub` to ensure completeness and Non-equivocation across Statements by identifying all Transparent Statements associated to a specific Subject.

Transparency Service: an entity that maintains and extends the Verifiable Data Structure and endorses its state. The identity of a Transparency Service is captured by a public key that must be known by Relying Parties 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.

Verifiable Data Structure: a data structure that supports one or more proof types, such as "inclusion proofs" or "consistency proofs", for Signed Statements as they are Registered to a Transparency Service. SCITT supports multiple Verifiable Data Structures and Receipt formats as defined in [RFC9942], accommodating different Transparency Service implementations.

4. 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 CT [RFC9162] are instances of this definition. SCITT supports multiple Verifiable Data Structures, as defined in [RFC9942].

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. Receipts demonstrate inclusion of Signed Statements in the Verifiable Data Structure of a Transparency Service. By extension, the Signed Statement may say an Artifact (for example, 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 Statement 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 Relying Parties to make informed decisions.

Transparency is implemented by providing a consistent, append-only, cryptographically verifiable, publicly available record of entries. Implementations of Transparency Services may protect their registered sequence of Signed Statements and Verifiable Data Structure using a combination of trusted hardware, consensus protocols, and cryptographic evidence. A Receipt is a signature over one or more Verifiable Data Structure Proofs that a Signed Statement is registered in the Verifiable Data Structure. It is universally verifiable without online access to the TS. Requesting a Receipt can result in the production of a new Receipt for the same Signed Statement. A Receipt's verification key, signing algorithm, validity period, header parameters or other claims **MAY** change each time a Receipt is produced.

Anyone with access to the Transparency Service can independently verify its consistency and review the complete list of Transparent Statements registered by each Issuer.

Thus, reputable Issuers are incentivized to carefully review their Statements before signing them to produce Signed Statements. Similarly, reputable Transparency Services are incentivized to secure their Verifiable Data Structure, as any inconsistency can easily be pinpointed by any Auditor with read access to the Transparency Service.

The building blocks specified in this document enable the unequivocal and auditable production of statements about software supply chain artifacts. The extensible design of the SCITT architecture potentially allows future usage with other supply chains in different domains, for example, advanced manufacturing or food supply.

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

- Certificate Authorities (CAs) (Issuers) sign the ASN.1 DER-encoded tbsCertificate structure to produce an X.509 certificate (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, Signed Tree Heads, and their respective consistency proofs are checked by Relying Parties
- The Verifiable Data Structure can be checked by Auditors

5. Architecture Overview

The SCITT architecture enables Transparency Services in a given application domain to implement a collective baseline by providing 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. The remaining details of the Receipt's contents are specified in [RFC9942].

Figure 2 illustrates the roles and processes that comprise a Transparency Service independent of any one use case:

- Issuers that use their credentials to create Signed Statements about Artifacts.
- Transparency Services that evaluate Signed Statements against Registration Policies producing Receipts upon successful Registration. The returned Receipt may be combined with the Signed Statement to create a Transparent Statement.
- Relying Parties that:
  - collect Receipts of Signed Statements for subsequent registration of Transparent Statements;
  - retrieve Transparent Statements for analysis of Statements about Artifacts themselves (e.g., verification);
  - or replay all the Transparent Statements to check for the consistency and correctness of the Transparency Service's Verifiable Data Structure (e.g., auditing).

In addition, Figure 2 illustrates multiple Transparency Services and multiple Receipts as a single Signed Statement **MAY** be registered with one or more Transparency Service. Each Transparency Service produces a Receipt, which may be aggregated in a single Transparent Statement, demonstrating the Signed Statement was registered by multiple Transparency Services.

The arrows indicate the flow of information.

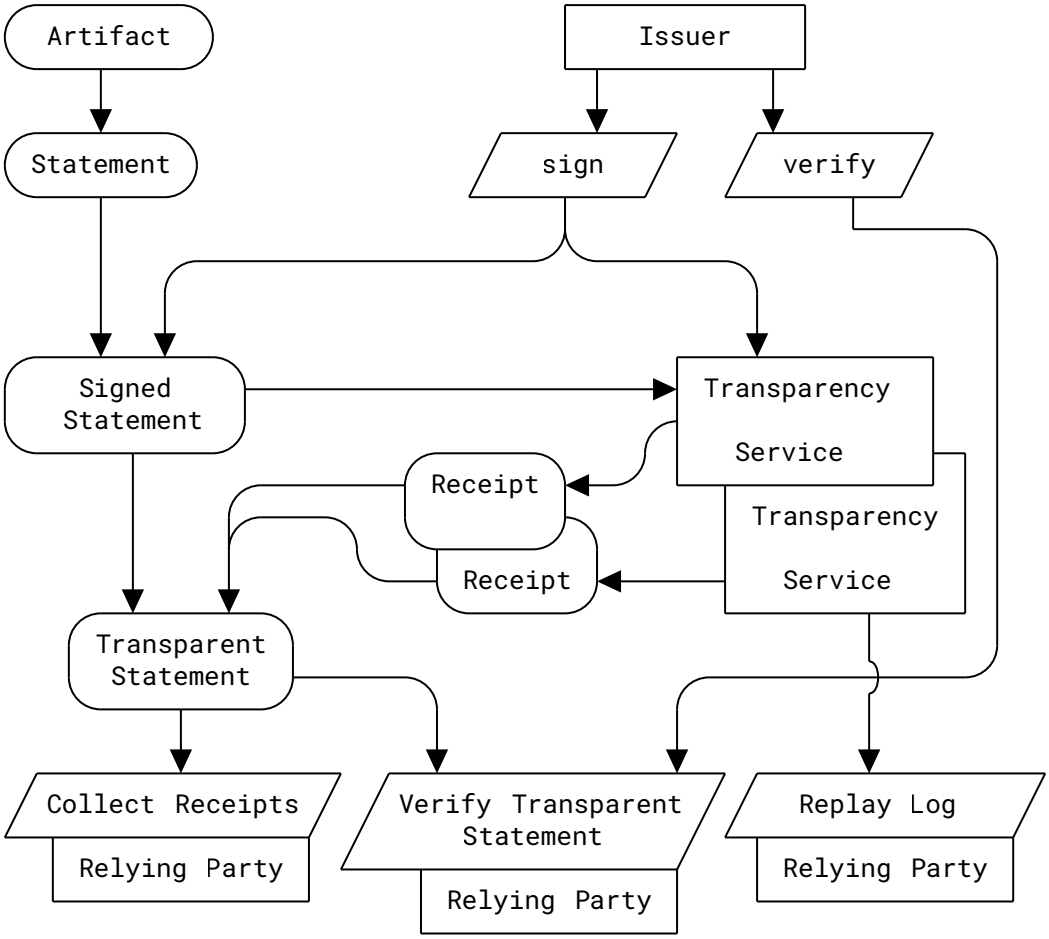


Figure 2: Relationship of Concepts in SCITT

The subsequent sections describe the main concepts, namely Transparency Service, Signed Statements, Registration, and Transparent Statements in more detail.

5.1. Transparency Service

Transparency Services **MUST** produce COSE Receipts [RFC9942].

Typically, a Transparency Service has a single Issuer identity that 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 subdomain, such as `tenant1.ts.example.` and `tenant2.ts.example.`.

5.1.1. Registration Policies

Registration Policies refer to additional checks over and above the Mandatory Registration Checks that are performed before a Signed Statement is registered to the Verifiable Data Structure. To enable auditability, Transparency Services **MUST** maintain Registration Policies. The presence of an explicit transparent registration policy, even if it allows all authenticated submissions, facilitates service audit, and enables potential future changes to that policy.

Beyond the mandatory Registration checks, the scope of additional checks, including no additional checks, is up to the implementation.

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

5.1.1.1. Mandatory Registration Checks

During Registration, a Transparency Service **MUST** syntactically check the Issuer of the Signed Statement by cryptographically verifying the COSE signature according to [STD96]. The Issuer identity **MUST** be bound to the Signed Statement by including an identifier in the protected header. If the protected header includes multiple identifiers, all those that are registered by the Transparency Service **MUST** be checked.

Transparency Services **MUST** maintain a list of trust anchors (see definition of trust anchor in [RFC4949]) in order to check the signatures of Signed Statements either separately or inside Registration Policies. Transparency Services **MUST** authenticate Signed Statements as part of a Registration Policy. For instance, a trust anchor could be an X.509 root certificate (directly or its thumbprint), a pointer to an OpenID Connect identity provider, or any other trust anchor that can be referenced in a COSE header parameter.

When using X.509 Signed Statements, the Transparency Service **MUST** build and validate a complete certification path from an Issuer's certificate to one of the root certificates currently registered as a trust anchor by the Transparency Service. The protected header of the COSE_Sign1 Envelope **MUST** include either the Issuer's certificate as `x5t` or the chain including the Issuer's certificate as `x5chain`, as defined in [RFC9360]. If `x5t` is included in the protected header, an `x5chain` with a leaf certificate corresponding to the `x5t` value **MAY** be included in the unprotected header.

Registration Policies and trust anchors **MUST** be made Transparent and available to all Relying Parties of the Transparency Service by Registering them as Signed Statements on the Verifiable Data Structure.

The Transparency Service **MUST** apply the Registration Policy that was most recently committed to the Verifiable Data Structure at the time of Registration.

5.1.1.2. Auditability of Registration

The operator of a Transparency Service **MAY** update the Registration Policy or the trust anchors of a Transparency Service at any time.

Transparency Services **MUST** ensure that for any Signed Statement they register, enough information is made available to Auditors to reproduce the Registration checks that were defined by the Registration Policies at the time of Registration. At a minimum, this consists of the Signed Statements themselves, any additional collateral data required to perform their authentication, and the applicable Registration Policy at the time of Registration.

5.1.2. Initialization and Bootstrapping

Since the mandatory Registration checks rely on having registered Signed Statements for the Registration Policy and trust anchors, Transparency Services **MUST** support at least one of the three following bootstrapping mechanisms:

- Preconfigured Registration Policy and trust anchors;
- Acceptance of a first Signed Statement whose payload is a valid Registration Policy, without performing Registration checks; or
- An out-of-band authenticated management interface.

5.1.3. Verifiable Data Structure

The security properties are determined by the choice of the Verifiable Data Structure (see [RFC9942]) used by the Transparency Service implementation. This verifiable data structure **MUST** support the following security requirements:

Append-Only: a property required for a verifiable data structure to be applicable to SCITT, ensuring that the Statement Sequence cannot be modified, deleted, or reordered.

Non-equivocation: there is no fork in the registered sequence of Signed Statements accepted by the Transparency Service and committed to the Verifiable Data Structure. Everyone with access to its content sees the same ordered collection of Signed Statements and can check that it is consistent with any Receipts they have verified.

Replayability: the Verifiable Data Structure includes sufficient information to enable authorized actors with access to its content to check that each data structure representing each Signed Statement has been correctly registered.

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 [RFC9942], and the review of their security requirements for SCITT are out of scope for this document.

5.1.4. Adjacent Services

Transparency Services can be deployed alongside other database or object storage technologies. For example, a Transparency Service that supports a software package management system, might be referenced from the APIs exposed for package management. It can also provide the ability to request a fresh Receipt for a given software package or a list of Signed Statements associated with that package.

6. Signed Statements

This specification prioritizes conformance to [STD96] and its required and optional properties. Signed Statements produced by Issuers must be COSE_Sign1 messages, as defined by [STD96]. Profiles and implementation-specific choices should be used to determine admissibility of conforming messages. This specification is left intentionally open to allow implementations to make Registration restrictions that make the most sense for their operational use cases.

There are many types of Statements (such as an SBOM, 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] Concise Software Identification Tags format
- [CycloneDX] Bill of Materials format
- [in-toto] Supply chain description metadata
- [SPDX-CBOR] Software component description format
- [SPDX-JSON] Software component description format
- [SLSA] Supply-chain Levels for Software Artifacts
- [SWID] Software Identification Tag format

Issuers can produce Signed Statements about different Artifacts under the same Identity. Issuers and Relying Parties 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 in Section 3.)

Issuers **MAY** use different signing keys (identified by `kid` in the protected header from [STD96]) 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. Relying Parties can choose which Issuers they trust.

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

Additionally, an `x5chain` that corresponds to either `x5t` or `kid` identifying the leaf certificate in the included certification path **MAY** be included in the unprotected header of the COSE Envelope.

- When using x.509 certificates, support for either `x5t` or `x5chain` in the protected header is **REQUIRED** to implement.
- Support for `kid` in the protected header and `x5chain` in the unprotected header is **OPTIONAL** to implement.

When `x5t` or `x5chain` is present in the protected header, `iss` **MUST** be a string that meets URI requirements defined in [RFC8392]. The `iss` value's length **MUST** be between 1 and 8192 characters in length.

The `kid` header parameter **MUST** be present when neither `x5t` nor `x5chain` is present in the protected header. 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 [RFC9597]. 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 additional Claims in its protected header related to verifying the inclusion proof in its unprotected header. See [RFC9942].

6.1. Signed Statement Examples

Figure 3 illustrates a normative CDDL definition [RFC8610] for the protected header and unprotected header of Signed Statements and Receipts.

The SCITT architecture specifies the minimal mandatory labels. Implementation-specific Registration Policies may define additional mandatory labels.

```
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
  ? &(x5chain: 33) => COSE_X509
  * label => any
}

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

Unprotected_Header = {
  ? &(x5chain: 33) => COSE_X509
  ? &(receipts: 394)  => [+ Receipt]
  * label => any
}

label = int / tstr
```

Figure 3: CDDL Definition for Signed Statements and Receipts

Figure 4 illustrates an instance of a Signed Statement in Extended Diagnostic Notation (EDN), with a payload that is detached. Detached payloads support large Statements and ensure Signed Statements can integrate with existing storage systems.

```
18(                                 / COSE_Sign1       /
    [
      h'a4012603...6d706c65',       / Protected        /
      {},                           / Unprotected      /
      nil,                          / Detached payload /
      h'79ada558...3a28bae4'        / Signature        /
    ]
)
```

Figure 4: CBOR-Extended Diagnostic Notation Example of a Signed Statement

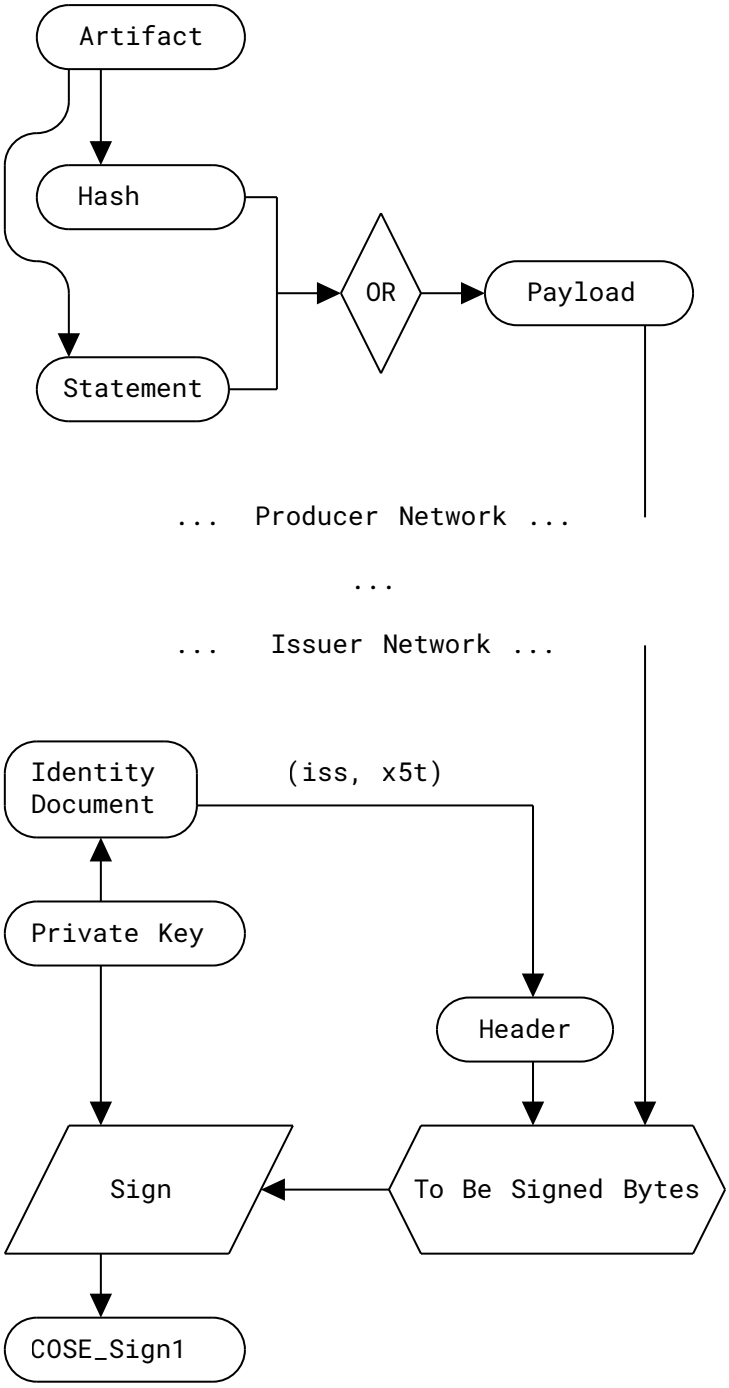
Figure 5 illustrates the decoded protected header of the Signed Statement in Figure 4. It indicates the Signed Statement is securing a JSON content type and identifying the content with the `sub` Claim "vendor.product.example".

```
{                                   / 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 5: CBOR-Extended Diagnostic Notation Example of a Signed Statement's Protected Header

6.2. Signing Large or Sensitive Statements

Statement payloads might be too large or too sensitive to be sent to a remote Transparency Service. In these cases, a Statement can be made over the hash of a payload rather than the full payload bytes.


Artifact
Hash
Statement
OR
Payload
... Producer Network ...
...
... Issuer Network ...
Identity Document
(iss, x5t)
Private Key
Header
Sign
To Be Signed Bytes
COSE_Sign1

6.3. Registration of Signed Statements

To register a Signed Statement, the Transparency Service performs the following steps:

1. Client Authentication

   A Client authenticates with the Transparency Service before registering Signed Statements on behalf of one or more Issuers. Authentication and authorization are implementation specific and out of scope of the SCITT architecture.

2. TS Signed Statement Verification and Validation

   The Transparency Service **MUST** perform signature verification per Section 4.4 of RFC 9052 [STD96] and **MUST** verify the signature of the Signed Statement with the signature algorithm and verification key of the Issuer per [RFC9360]. The Transparency Service **MUST** also check that the Signed Statement includes the required protected headers. The Transparency Service **MAY** validate the Signed Statement payload in order to enforce domain-specific registration policies that apply to specific content types.

3. Apply Registration Policy

   The Transparency Service **MUST** check the attributes required by a Registration 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.

4. Register the Signed Statement
5. Return the Receipt

   This **MAY** be asynchronous from Registration. The Transparency Service **MUST** be able to provide a Receipt for all registered Signed Statements. Details about generating Receipts are described in Section 7.

The last two steps may be shared between a batch of Signed Statements registered in the Verifiable Data Structure.

A Transparency Service **MUST** ensure that a Signed Statement is registered before releasing its Receipt.

A Transparency Service **MAY** accept a Signed Statement with content in its unprotected header and **MAY** use values from that unprotected header during verification and registration policy evaluation.

However, the unprotected header of a Signed Statement **MUST** be set to an empty map before the Signed Statement can be included in a Statement Sequence.

The same Signed Statement may be independently registered in multiple Transparency Services, producing multiple independent Receipts. The multiple Receipts may be attached to the unprotected header of the Signed Statement creating a Transparent Statement.

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.

7. 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. Client applications **MAY** register Signed Statements on behalf of one or more Issuers. Client applications **MAY** request Receipts regardless of the identity of the Issuer of the associated 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 Receipts [RFC9942], which also provides the COSE header parameter semantics for label 394.

The Registration time is recorded as the timestamp when the Transparency Service added the Signed Statement to its Verifiable Data Structure.

Figure 6 illustrates a normative CDDL definition of Transparent Statements. See Figure 3 for the CDDL rule that defines 'COSE_Sign1' as specified in Section 4.2 of RFC 9052 [STD96].

```
Transparent_Statement = #6.18(COSE_Sign1)

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

Figure 6: CDDL Definition for a Transparent Statement

Figure 7 illustrates a Transparent Statement with a detached payload and two Receipts in its unprotected header. The type of label 394 `receipts` in the unprotected header is a CBOR array that can contain one or more Receipts (each entry encoded as a .cbor-encoded Receipt).

```
18(                                 / COSE_Sign1                /
    [
      h'a4012603...6d706c65',       / Protected                 /
      {                             / Unprotected               /
        394:   [                    / Receipts (2)              /
          h'd284586c...4191f9d2'    / Receipt 1                 /
          h'c624586c...8f4af97e'    / Receipt 2                 /
        ]
      },
      nil,                          / Detached payload          /
      h'79ada558...3a28bae4'        / Signature                 /
    ]
)
```

Figure 7: CBOR-Extended Diagnostic Notation Example of a Transparent Statement

Figure 8 illustrates one of the decoded Receipts from Figure 7. The Receipt contains inclusion proofs for verifiable data structures. The unprotected header contains verifiable data structure proofs. See the protected header for details regarding the specific verifiable data structure used. Per the COSE Verifiable Data Structure Algorithms Registry documented in [RFC9942], the COSE key type RFC9162_SHA256 is value `1`. Labels identify inclusion proofs (`-1`) and consistency proofs (`-2`).

```
18(                                 / COSE_Sign1                /
    [
      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 8: CBOR-Extended Diagnostic Notation Example of a Receipt

Figure 9 illustrates the decoded protected header of the Transparent Statement in Figure 7. The verifiable data structure (`-111`) uses `1` from (RFC9162_SHA256).

```
{                                  / 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 9: CBOR-Extended Diagnostic Notation Example of a Receipt's Protected Header

Figure 10 illustrates the decoded inclusion proof from Figure 8. This inclusion proof indicates that the size of the Verifiable Data Structure was 8 at the time the Receipt was issued. The structure of this inclusion proof is specific to the verifiable data structure used (RFC9162_SHA256).

```
[                                  / 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 10: CBOR-Extended Diagnostic Notation Example of a Receipt's Inclusion Proof

7.1. Validation

Relying Parties **MUST** apply the verification process as described in Section 4.4 of RFC 9052 [STD96] when checking the signature of Signed Statements and Receipts.

A Relying Party **MUST** trust the verification key or certificate and the associated identity of at least one Issuer of a Receipt.

A Relying Party **MAY** decide to verify only a single Receipt that is acceptable to them and not check the signature on the Signed Statement or Receipts that rely on verifiable data structures they do not understand.

APIs exposing verification logic for Transparent Statements may provide more details than a single boolean result. For example, an API may indicate 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.

Relying Parties **MAY** be configured to re-verify the Issuer's Signed Statement locally.

In addition, Relying Parties **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.

8. Privacy Considerations

Interactions with Transparency Services are expected to use appropriately strong encryption and authorization technologies.

The Transparency Service is trusted with the confidentiality of the Signed Statements presented for Registration. Issuers and Clients are responsible for verifying that the Transparency Service's privacy and security posture is suitable for the contents of the Signed Statements they submit prior to Registration. Issuers must carefully review the inclusion of private, confidential, or Personally Identifiable Information (PII) in their Statements against the Transparency Service's privacy posture.

In some deployments, a special role such as an Auditor might require and be given access to both the Transparency Service and related Adjacent Services.

Transparency Services can leverage Verifiable Data Structures that only retain cryptographic metadata (e.g., a hash) rather than the complete Signed Statement as part of an in-depth defensive approach to maintaining confidentiality. By analyzing the relationship between data stored in the Transparency Service and data stored in Adjacent Services, it is possible to perform metadata analysis, which could reveal the order in which artifacts were built, signed, and uploaded.

9. Security Considerations

SCITT provides the following security guarantees:

1. Statements made by Issuers about supply chain Artifacts are identifiable and can be authenticated.
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. The second guarantee is achieved by proving a Signed Statement is present in a Verifiable Data Structure. The third guarantee is achieved by the combination of both of these steps.

In addition to deciding whether to trust a Transparency Service, Relying Parties can use the history of registered Signed Statements to decide which Issuers they choose to trust. This decision process is out of scope of this document.

9.1. Ordering of Signed Statements

Statements are signed prior to submitting to a SCITT Transparency service. Unless advertised in the Transparency Service Registration Policy, the Relying Party cannot assume that the ordering of Signed Statements in the Verifiable Data Structure matches the ordering of their issuance.

9.2. Accuracy of Statements

Issuers can make false Statements either intentionally or unintentionally; registering a Statement only proves it was produced by an Issuer. A registered Statement may be superseded by a subsequently submitted Signed Statement from the same Issuer, with the same subject in the `CWT Claims` protected header. Other Issuers may make new Statements to reflect new or corrected information. Relying Parties may choose to include or exclude Statements from Issuers to determine the accuracy of a collection of Statements.

9.3. Issuer Participation

Issuers can refuse to register their Statements with a Transparency Service or selectively submit some but not all the Statements they issue. It is important for Relying Parties not to accept Signed Statements for which they cannot discover Receipts issued by a Transparency Service they trust.

9.4. Key Management

Issuers and Transparency Services **MUST**:

- carefully protect their private signing keys
- avoid using keys for more than one purpose
- rotate their keys at a cryptoperiod (defined in [RFC4949]) appropriate for the key-algorithm and domain-specific regulations

9.4.1. Verifiable Data Structure

The security considerations for specific Verifiable Data Structures are out of scope for this document. See [RFC9942] for the generic security considerations that apply to Verifiable Data Structure and Receipts.

9.4.2. Key Compromise

It is important for Issuers and Transparency Services to clearly communicate when keys are compromised so that Signed Statements can be rejected by Transparency Services or Receipts can be ignored by Relying Parties. Transparency Services whose receipt signing keys have been compromised can roll back their Statement Sequence to a point before compromise, establish new credentials, and use the new credentials to issue fresh Receipts going forward. Issuers are encouraged to follow existing best practices if their credentials are compromised. Revocation strategies for compromised keys are out of scope for this document.

9.4.3. Bootstrapping

Bootstrapping mechanisms that solely rely on Statement registration to set and update registration policy can be audited without additional implementation-specific knowledge; therefore, they are preferable. Mechanisms that rely on preconfigured values and do not allow updates are unsuitable for use in long-lived service deployments in which the ability to patch a potentially faulty policy is essential.

9.5. Implications of Media Type Usage

The Statement (scitt-statement+cose) and Receipt (scitt-receipt+cose) media types describe the expected content of COSE envelope headers. The payload media type ('content type') is included in the COSE envelope header. [STD96] describes the security implications of reliance on this header parameter.

Both media types describe COSE_Sign1 messages, which include a signature and therefore provide integrity protection.

9.6. Cryptographic Agility

Because the SCITT Architecture leverages [STD96] for Statements and Receipts, it benefits from the format's cryptographic agility.

9.7. Threat Model

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

SCITT primarily supports checking of Signed Statement authenticity, both from the Issuer (authentication) and from the Transparency Service (transparency). Issuers and Transparency Services can both be compromised.

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. Running multiple, independent Transparency Services provides different organizations to represent consistent or divergent opinions. It is the role of the relying party to decide which Transparency Services and Issuers they choose to trust for their scenario.

In both cases, the SCITT architecture provides generic, universally verifiable cryptographic proofs to hold Issuers or Transparency Services accountable. 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.

Relying Parties and Auditors need not be trusted by other actors. 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. IANA Considerations

IANA has registered the following media types in the "application" subregistry of the "Media Types" registry group [IANA.media-types]:

- application/scitt-statement+cose (see Section 10.1)
- application/scitt-receipt+cose (see Section 10.2)

10.1. Registration of application/scitt-statement+cose

Name	Template	Reference
scitt-statement+cose	application/scitt-statement+cose	Section 6 of RFC 9943

Table 1: SCITT Signed Statement Media Type Registration

Type name: application

Subtype name: statement+cose

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary (CBOR data item)

Security considerations: Section 9.5 of RFC 9943

Interoperability considerations: none

Published specification: RFC 9943

Applications that use this media type: Used to provide an identifiable and non-repudiable Statement about an Artifact signed by an Issuer.

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): .scitt

Macintosh file type code(s): N/A

Person & email address to contact for further information: iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: none

Author/Change controller: IETF

10.2. Registration of application/scitt-receipt+cose Registration

Name	Template	Reference
scitt-receipt+cose	application/scitt-receipt+cose	Section 7 of RFC 9943

Table 2: SCITT Receipt Media Type Registration

Type name: application

Subtype name: receipt+cose

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary (CBOR data item)

Security considerations: Section 9.5 of RFC 9943

Interoperability considerations: none

Published specification: RFC 9943

Applications that use this media type: Used to establish or verify transparency over Statements. Typically emitted by a Transparency Service for the benefit of Relying Parties wanting to ensure Non-equivocation over all or part of a Statement Sequence.

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): .receipt

Macintosh file type code(s): N/A

Person & email address to contact for further information: iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: none

Author/Change controller: IETF

10.3. CoAP Content-Format Registrations

IANA has registered the following Content-Format numbers in the "CoAP Content-Formats" subregistry within the "Constrained RESTful Environments (CoRE) Parameters" registry group [IANA.core-parameters] in the 256-9999 range:

Content Type	Content Coding	ID	Reference
application/scitt-statement+cose	-	277	RFC 9943
application/scitt-receipt+cose	-	278	RFC 9943

Table 3: SCITT Content-Formats Registration

11. References

11.1. Normative References

[IANA.core-parameters] IANA, "Constrained RESTful Environments (CoRE) Parameters", <https://www.iana.org/assignments/core-parameters>.

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

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

[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/info/rfc2119>.

[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/info/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/info/rfc8174>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/info/rfc8392>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[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/info/rfc9360>.

[RFC9597] Looker, T. and M.B. Jones, "CBOR Web Token (CWT) Claims in COSE Headers", RFC 9597, DOI 10.17487/RFC9597, June 2024, <https://www.rfc-editor.org/info/rfc9597>.

[RFC9942] Steele, O., Birkholz, H., Delignat-Lavaud, A., and C. Fournet, "CBOR Object Signing and Encryption (COSE) Receipts", RFC 9942, DOI 10.17487/RFC9942, March 2026, <https://www.rfc-editor.org/info/rfc9942>.

[STD94] Internet Standard 94, <https://www.rfc-editor.org/info/std94>.
At the time of writing, this STD comprises the following:

Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[STD96] Internet Standard 96, <https://www.rfc-editor.org/info/std96>.
At the time of writing, this STD comprises the following:

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/info/rfc9052>.

Schaad, J., "CBOR Object Signing and Encryption (COSE): Countersignatures", STD 96, RFC 9338, DOI 10.17487/RFC9338, December 2022, <https://www.rfc-editor.org/info/rfc9338>.

11.2. Informative 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/info/rfc9393>.

[CycloneDX] "CycloneDX", <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", ACM SIGOPS Operating Systems Review, vol. 41, no. 6, pp. 189-204, DOI 10.1145/1323293.1294280, 14 October 2007, <https://www.read.seas.harvard.edu/~kohler/class/08w-dsi/chun07attested.pdf>.

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

[NIST.SP.1800-19] Bartock, M., Dodson, D., Souppaya, M., Carroll, D., Masten, R., Scinta, G., Massis, P., Prafullchandra, H., Malnar, J., Singh, H., Ghandi, R., Storey, L. E, Yeluri, R., Shea, T., Dalton, M., Weber, R., Scarfone, K., Dukes, A., Haskins, J., Phoenix, C., and B. Swarts, "Trusted cloud: Security Practice Guide for VMware Hybrid Cloud Infrastructure as a Service (IaaS) Environments", DOI 10.6028/NIST.SP.1800-19, NIST SP 1800-19, 20 April 2022, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1800-19.pdf>.

[NIST.SP.800-204C] Chandramouli, R., "Implementation of DevSecOps for a Microservices-based Application with Service Mesh", DOI 10.6028/NIST.SP.800-204C, NIST Special Publications (General) 800-204C, NIST SP 800-204C, DOI 10.6028/NIST.SP.800-204C, March 2022, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204C.pdf>.

[NIST_EO14028] NIST, "Software Supply Chain Security Guidance Under Executive Order (EO) 14028 Section 4e", 4 February 2022, <https://www.nist.gov/system/files/documents/2022/02/04/software-supply-chain-security-guidance-under-EO-14028-section-4e.pdf>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <https://www.rfc-editor.org/info/rfc8725>.

[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/info/rfc9162>.

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

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

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

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

Contributors

Orie Steele
Tradeverifyd
United States of America
Email: orie@or13.io

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

Amaury Chamayou
Microsoft
United Kingdom
Email: amaury.chamayou@microsoft.com

Amaury contributed elemental parts to finalize normative language on registration behavior and the single-issuer design, as well as overall document consistency

Dick Brooks
Business Cyber Guardian
United States of America
Email: dick@businesscyberguardian.com

Dick contributed to the software supply chain use cases.

Brian Knight
Microsoft
United States of America
Email: brianknight@microsoft.com

Brian contributed to the software supply chain use cases.

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

Robert contributed to the software supply chain use cases.

Authors' Addresses

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

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
Email: stevenlasker@hotmail.com