Workgroup: SCITT
Internet-Draft:
draft-ietf-scitt-architecture-01
Published: 14 March 2023
Intended Status: Standards Track
Expires: 15 September 2023
Authors: H. Birkholz
Fraunhofer SIT
A. Delignat-Lavaud
Microsoft Research
C. Fournet
Microsoft Research
Y. Deshpande
ARM

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 memo defines a generic and scalable architecture to enable transparency across any supply chain with minimum adoption barriers for producers (who can register their Signed Statements on any Transparency Service, with the guarantee that all consumers will be able to verify them) and enough flexibility to allow different implementations of Transparency Services with various auditing and compliance requirements.

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 15 September 2023.

Copyright Notice

Table of Contents

1. Introduction
  1.1. Requirements Notation
2. Use Cases
  2.1. Software Bill of Materials (SBOM)
  2.2. Confidential Computing
  2.3. Cold Chains for Seafood
3. Terminology
4. Definition of Transparency
5. Architecture Overview
  5.1. Signed Statement Issuance and Registration
    5.1.1. Issuer Identity
    5.1.2. Naming Artifacts
    5.1.3. Signed Statement Metadata
  5.2. Transparency Service
    5.2.1. Service Identity, Remote Attestation, and Keying
    5.2.2. Registration Policies
    5.2.3. Registry Security Requirements
  5.3. Verifying Transparent Statements
6. Signed Statement Issuance, Registration, and Verification
  6.1. Envelope and Signed Statement Format

6.2. Signed Statement Issuance
6.3. Standard Registration Policies
6.4. Registering Signed Statements
6.5. Validation of Transparent Statements
7. FederationThis section needs work.
8. Transparency Service APIThis may be moved to appendix.
8.1. Messages
8.1.1. Register Signed Statement
8.1.2. Retrieve Operation Status
8.1.3. Retrieve Signed Statement
8.1.4. Retrieve Registration Receipt
9. Privacy Considerations
10. Security Considerations
10.1. Threat Model
10.1.1. Signed Statement Authentication and Transparency.
10.1.2. Confidentiality and privacy.
10.1.3. Cryptographic Assumptions
10.1.4. Transparency Service Clients
10.1.5. Identity
11. IANA Considerations
11.1. URN Sub-namespace for SCITT (urn:ietf:params:scitt)
12. References
12.1. Normative References
12.2. Informative References
Appendix A. Attic
Authors' Addresses

1. Introduction

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

1. statements made by issuers about supply chain artifacts must be identifiable, authentic, and non-repudiable;

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

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

The first guarantee is achieved by requiring issuers to sign their statements and associated metadata using a distributed public key infrastructure. The second guarantee is achieved by storing the

signed statement in an immutable, append-only, transparent Registry. The last guarantee is achieved by implementing the Registry using a verifiable data structure (such as a Merkle Tree [MERKLE]), and by requiring a Transparency Service that operates the Registry to endorse its state at the time of registration.

The guarantees and techniques used in this document generalize those of Certificate Transparency [RFC9162], which can be re-interpreted as an instance of this architecture for the supply chain of X.509 certificates. However, the range of use cases and applications in this document is much broader, which requires much more flexibility in how each Transparency Service implements and operates its Registry. Each service may enforce its own policy for authorizing entities to register their Signed Statements on the Transparency Service. Some Transparency Services may also enforce access control policies to limit who can audit the full Registry, or keep some information on the Registry encrypted. Nevertheless, it is critical to provide global interoperability for all Transparency Services instances as the composition and configuration of involved supply chain entities and their system components is ever-changing and always in flux.

A Transparency Services provides visibility into Signed Statements originally created as Statements and issued as Signed Statements by supply chain entities and their sub-systems. These Signed Statements (and corresponding Statement payload) are about the objects produced by supply chain objects: Artifacts. A Transparency Service vouches for specific and well-defined metadata about these Artifacts that is captured in Statements. Some metadata is selected (and signed) by the Issuer, indicating, e.g., "who issued the Statement" or "what type of Artifact is described" or "what is the Artifact's version"; whereas additional metadata is selected (and countersigned) by the Transparency Services, indicating, e.g., "when was the Signed Statement about the Artifact registered in the Registry". A Statements payload content typically is opaque to the Transparency Services, if so desired: it is the metadata that must always be transparent in order to warrant trust for later processing.

Transparent Statements provide a common basis for holding Issuers accountable for the Statement payload about Artifacts they release and (more generally) principals accountable for auxiliary Signed Statements from other Issuers about the original Signed Statement about an Artifact. Hence, Issuers may register new Signed Statements about their Artifacts, but they cannot delete or alter earlier Signed Statements about certain Artifacts, or hide their Signed Statements from third parties such as auditors.

Trust in the Transparency Service itself is supported both by protecting their implementation (using, for instance, replication,

trusted hardware, and remote attestation of systems) and by enabling independent audits of the correctness and consistency of its Registry, thereby holding the organization accountable that operates it. Unlike CT, where independent auditors are responsible for enforcing the consistency of multiple independent instances of the same global Registry, each Transparency Service is required to guarantee the consistency of its own Registry (for instance, through the use of a consensus algorithm between replicas of the Registry), but assume no consistency between different Transparency Services.

The Transparency Services specified in this architecture caters to two types of audiences:

1. Signed Statement Issuers: entities, stakeholders, and users involved in supply chain interactions that need to release authentic Statements to a definable set of peers; and

2. Transparent Statement Consumers: entities, stakeholders, and users involved in supply chain interactions that need to access, validate, and trust authentic Statements.

Signed Statement Issuers rely on being discoverable and represented as the responsible parties for their registered Signed Statements via Transparency Services in a believable manner. Analogously, Transparent Statement Consumers rely on verifiable trustworthiness assertions associated with Transparent Statements and their processing provenance in a believable manner. If trust can be put into the operations that record Signed Statements (i.e., a believable notarization function) in a secure, append-only Registry via online operations, the same trust can be put into a corresponding Receipt that is the resulting documentation of these online operations issued by the Transparency Services and that can be validated in offline operations.

The Transparency Services specified in this architecture can be implemented by various different types of services in various types of languages provided via various variants of API layouts.

The global interoperability enabled and guaranteed by the Transparency Services is enabled via core components (architectural constituents) that come with prescriptive requirements (that are typically hidden away from the user audience via APIs). The core components are based on the Concise Signing and Encryption standard specified in [RFC9052], which is used to sign released Statements about Artifacts and to build and maintain a Merkle tree that functions as an append-only Registry for corresponding Signed Statements. The format and verification process for Registry-based transparency receipts are described in [I-D.birkholz-scitt-receipts].

1.1. Requirements Notation

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

2. Use Cases

This section presents representative and solution-agnostic use cases to illustrate the scope of SCITT and the processing of Digital Supply Chain Artifacts.

2.1. Software Bill of Materials (SBOM)

As the ever-increasing complexity of large software projects requires more modularity and abstractions to manage them, keeping track of their full Trusted Computing Base (TCB) is becoming increasingly difficult. Each component may have its own set of dependencies and libraries. Some of these dependencies are binaries, which means their TCB depends not only on their source, but also on their build environment (compilers and tool-chains). Besides, many source and binary packages are distributed through various channels and repositories that may not be trustworthy.

Software Bills of Materials (SBOM) help the authors, packagers, distributors, auditors and users of software understand its provenance and who may have the ability to introduce a vulnerability that can affect the supply chain downstream. However, the usefulness of SBOM in protecting end users is limited if supply chain actors cannot be held accountable for their contents. For instance, consider a package repository for an open source operating system distribution. The operator of this repository may decide to provide a malicious version of a package only to users who live in a specific country. They can write two equivocal SBOMs for the honest and backdoored versions of the package, so that nobody outside the affected country can discover the malicious version, but victims are not aware they are being targeted.

2.2. Confidential Computing

Confidential Computing can leverage hardware-protected trusted execution environments (TEEs) to operate cloud services that protect the confidentiality of data that they process. It relies on remote attestation, which allows the service to prove to remote users what is the hash of its software, as measured and signed by the hardware.

For instance, consider a speech recognition service that implements machine learning inference using a deep neural network model. The

operator of the service wants to prove to its users that the service preserves the user's privacy, that is, the submitted recordings can only be used to detect voice commands but no other purpose (such as storing the recordings or detecting mentions of brand names for advertisement purposes). When the user connects to the TEE implementing the service, the TEE presents attestation evidence that includes a hardware certificate and a software measurement for their task; the user verifies this evidence before sending its recording.

But how can users verify the software measurement for their task? And how can operators update their service, e.g., to mitigate security vulnerabilities or improve accuracy, without first convincing all users to update the measurements they trust?

A supply chain that maintains a transparent record of the successive software releases for machine-learning models and runtimes, recording both their software measurements and their provenance (source code, build reports, audit reports, ...) can provide users with the information they need to authorize these tasks, while holding the service operator accountable for the software they release for them.

2.3. Cold Chains for Seafood

Once seafood is caught, its quality is determined -- amongst other criteria -- via the integrity of a cold chain that ensures a regulatory perspective freshness mandating a continuous storing temperature between 1 °C and 0 °C (or -18 °C and lower for frozen seafood). The temperature is recorded by cooling units adhering to certain compliance standards automatically. Batches of seafood can be split or aggregated before arriving in a shelf so that each unit can potentially have a potentially unique cold chain record whose transparency impacts the accuracy of the shelf-life associated with it. Especially in early links of the supply chain, Internet connection or sophisticated IT equipment are typically not available; sometimes temperature measurements are recorded manually; and digital records are created in hindsight.

3. Terminology

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

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

Auditor: an entity that checks the correctness and consistency of all Transparent Statements registered by a Transparency Service (a subset of potential Transparent Statement Consumers).

Consumer of Signed Statements: Define here.

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

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

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

Receipt: a Receipt is a special form of COSE countersignature for Signed Statements that embeds cryptographic evidence that the Signed Statement is recorded in the Registry. A Receipt consists of a Registry-specific inclusion proof, a signature by the Transparency Service of the state of the Registry, and additional metadata (contained in the countersignature's protected headers) to assist in auditing.

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

Registration Policy: the pre-condition enforced by the Transparency Service before registering a Signed Statement, rendering it a Signed Statement, based on metadata contained in its COSE Envelope

(notably the identity of its Issuer) and on prior Signed Statements already added to a Registry.

Registry: the verifiable append-only data structure that stores Signed Statements in a Transparency Service often referred to by the synonym log or ledger. SCITT supports multiple Registry and Receipt formats to accommodate different Transparency Service implementations, such as historical Merkle Trees and sparse Merkle Trees.

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

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

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

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

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

4. Definition of Transparency

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

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

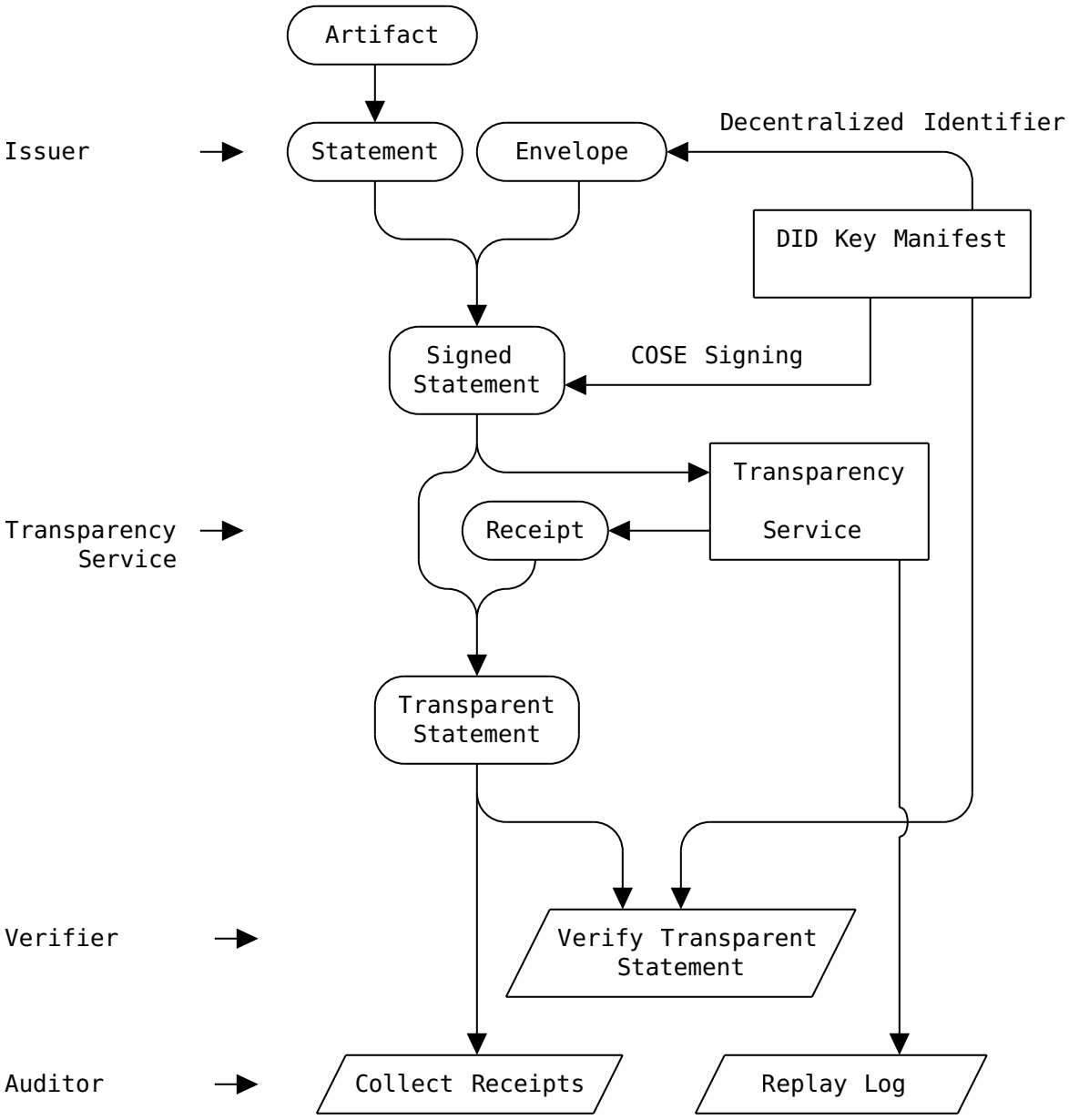
Transparency does not prevent dishonest or compromised Issuers, but it holds them accountable: any Artifact that may be used to target a particular user that checks for Receipts must have been recorded in the tamper-proof Registry, and will be subject to scrutiny and auditing by other parties.

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

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

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

5. Architecture Overview



The SCITT architecture consists of a very loose federation of Transparency Services, and a set of common formats and protocols for issuing, registering 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 Registry algorithm. Most of the details of the Receipt's contents are specific to the Registry algorithm. The [I-D.birkholz-scitt-receipts] document defines two initial Registry algorithms (for historical and sparse Merkle Trees), but other Registry formats (such as

blockchains, or hybrid historical and indexed Merkle Trees) may be proposed later.

In this section, a high level the three main roles and associated processes in SCITT: Issuers and the Signed Statement issuance process, transparency Registry and the Transparent Statement Registration process, as well as Verifiers and the Receipt validation process.

5.1. Signed Statement Issuance and Registration

5.1.1. Issuer Identity

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

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

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

5.1.2. Naming Artifacts

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

5.1.3. Signed Statement Metadata

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

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

5.2. Transparency Service

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

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

Beyond the trusted components, Transparency Services may operate additional endpoints for auditing, for instance to query for the history of Transparent Statements registered by a given Issuer via a

certain Feed. Implementations of Transparency Services **SHOULD** avoid using the service identity and extending the Registry in auditing endpoints; as much as practical, the Registry **SHOULD** contain enough evidence to re-construct verifiable proofs that the results returned by the auditing endpoint are consistent with a given state of the Registry.

5.2.1. Service Identity, Remote Attestation, and Keying

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

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

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

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

5.2.2. Registration Policies

A Transparency Services that accepts to register any valid Signed Statement offered by an Issuer would end up providing only limited value to verifiers. In consequence, a baseline transparency guarantee policing the registration of Signed Statements is required to ensure completeness of audit, which can help detect equivocation. Most advanced SCITT scenarios rely on the Transparency Service performing

additional domain-specific checks before a Signed Statement is accepted: Transparency Services may only allow trusted authenticated users to register Signed Statements, Transparency Services may try to check that a new Signed Statement is consistent with previous Signed Statements from the same Issuers or that Signed Statements are registered in the correct order and cannot be re-played; some Transparency Services may even interpret and validate the payload of Signed Statements.

In general, registration policies are applied at the discretion of the Transparency Services, and verifiers use Receipts as witnesses that confirm that the registration policy of the Transparency Services was satisfied at the time of creating a Transparent Statement via Signed Statement registration. Transparency Service implementations **SHOULD** make their full registration policy public and auditable, e.g. by recording stateful policy inputs at evaluation time in the registry to ensure that policy can be independently validated later. From an interoperability point of view, the policy that was applied by the Transparency Services is opaque to the verifier, which is forced to trust the associated registration policy. If the policy of the Transparency Services evolves over time, or is different across Issuers, the assurances derived from Receipt validation may not be uniform across all Signed Statements over time.

To help verifiers interpret the semantics of Signed Statement registration, the SCITT Architecture defines a standard mechanism to include signals the Signed Statement itself which policies have been applied by the Transparency Service from a defined set of registration policies with standardized semantics. Each policy that is expected to be enforced by the Transparency Service is represented by an entry in the registration policy info map (reg_info) in the COSE Envelope of the Signed Statement. The key of the map entry corresponds to the name of the policy, while its value (including its type) is policy-specific. For instance, the register_by policy defines the maximum timestamp by which a Signed Statement can be registered, hence the associated value contains an unsigned integer.

While this design ensures that all verifiers get the same guarantee regardless of where a Transparent Statement is registered, its main downside is that it requires the Issuer to include the necessary policies in the Envelope when the Signed Statement is produced. Furthermore, it makes it impossible to register the same Signed Statement on two different Transparency Services, if their required registration policies are incompatible.

Editor's note

The technical design for signalling and verifying registration policies is a work in progress. An alternative design would be to

include the registration policies in the receipt/countersignature rather than in the envelope. This improves the portability of Signed Statements but requires the verifier to be more aware of the particular policies at the Transparency Service where the Signed Statement is registered.

5.2.3. Registry Security Requirements

There are many different candidate verifiable data structures that may be used to implement the Registry, such as chronological Merkle Trees, sparse/indexed Merkle Trees, full blockchains, and many other variants. The Registry is only required to support concise Receipts (i.e., whose size grows at most logarithmically in the number of entries in the Registry). This does not necessarily rule out blockchains as a Registry, but may necessitate advanced Receipt schemes that use arguments of knowledge and other verifiable computing techniques.

Since the details of how to verify a Receipt are specific to the data structure, no particular Registry format is specified in this document. Instead, two initial formats for Registry in [I-D.birkholz-scitt-receipts] using historical and sparse Merkle Trees are proposed. Beyond the format of Receipts, generic properties that should be satisfied by the components in the Transparency Services that have the ability to write to the Registry are required.

5.2.3.1. Finality

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

5.2.3.2. Consistency

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

5.2.3.3. Replayability and Auditing

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

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

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

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

- a Transparency Service **MAY** additionally support verifiability of client authentication and access control.

5.2.3.4. Governance and Bootstrapping

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

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

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

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

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

5.3. Verifying Transparent Statements

For a given Artifact, Verifiers take as trusted inputs:

1. the distributed identifier of the Issuer (or its resolved key manifest),

2. the expected name of the Artifact (i.e., the Feed),

3. the list of service identities of trusted Transparency Services.

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

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

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

6. Signed Statement Issuance, Registration, and Verification

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

6.1. Envelope and Signed Statement Format

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

At a high-level that is the context of this architecture, a Signed Statement is a COSE single-signed object (i.e., a COSE_Sign1) that contains the correct set of protected headers. Although Issuers and relaying parties may attach unprotected headers to Signed Statements,

Transparency Services and Verifiers **MUST NOT** rely on the presence or value of additional unprotected headers in Signed Statements during Registration and validation.

All Signed Statements **MUST** include the following protected headers:

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

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

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

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

- Registration policy info (label: TBD, temporary: 393): a map of additional attributes to help enforce Registration policies;

- Key ID (label: 4): Key ID, as a bytestring.

Additionally, Signed Statements **MAY** carry the following unprotected headers:

- Receipts (label: TBD, temporary: 394): Array of Receipts, defined in [I-D.birkholz-scitt-receipts]

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

```
SCITT_Envelope = COSE_Sign1_Tagged

COSE_Sign1_Tagged = #6.18(COSE_Sign1)

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

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

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

Unprotected_Header = {
  ; TBD, Labels are temporary
  ? 394 => [+ SCITT_Receipt]
}
```

6.2. Signed Statement Issuance

There are many types of Statements (such as SBOMs, malware scans, audit reports, policy definitions) that Issuers may want to turn into Signed Statements. An Issuer must first decide on a suitable format to serialize the Statement payload, such as:

- JSON-SPDX
- CBOR-SPDX
- SWID
- CoSWID

*CycloneDX

*in-toto

*SLSA

Once the Statement is serialized with the correct media-type/content-format, an Issuer should fill in the attributes for the Registration policy information header. From the Issuer's perspective, using attributes from named policies ensures that the Signed Statement may only be registered on Transparency Services that implement the associated policy. For instance, if a Signed Statement is frequently updated, and it is important for Verifiers to always consider the latest version, Issuers **SHOULD** use the sequence_no or issuer_ts attributes.

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

6.3. Standard Registration Policies

Editor's note

The technical design for signaling and verifying registration policies is a work in progress. We expect that once the formats and semantics of the registration policy headers are finalized, standardized policies may be moved to a separate draft. For now, we inline some significant policies to illustrate the most common use cases.

Transparency Service implementations **MUST** indicate their support for registration policies and **MUST** check that all the policies indicated as defined in the reg_info map are supported and are satisfied before a Signed Statement can be registered. Any unsupported types of Signed Statements **MUST** be indicated separately and corresponding unknown policy entries in the map of a Signed Statement **MUST** be rejected. This is to ensure that all verifiers get the same guarantee out of the registration policies regardless of where it is registered.

Policy Name	**Required reg_info attributes**	**Implementation**
TimeLimited	register_by: uint .within (~time)	Returns true if now () < register_by at registration time. The Transparency Service **MUST** store the time of registration along with the Signed

Policy Name	Required reg_info attributes	Implementation
		Statement, and **SHOULD** indicate it in corresponding Receipts. The value provided for register_by **MUST** be an unsigned integer, interpreted according to POSIX time, representing the number of seconds since 1970-01-01T00:00Z UTC.
Sequential	sequence_no: uint	First, lookup of existing registered Transparent Statements with same Issuer and Feed. If at least one is found, returns true if and only if the sequence_no of the new Signed Statement to be registered would become the highest sequence_no in the set of existing Transparent Statements, incremented by one. Otherwise, returns true if and only if sequence_no = 0.
Temporal	issuance_ts: uint .within (~time)	Returns true if and only if there is no existing already registered Transparent Statement in the ledger with the same Issuer and Feed with a greater issuance_ts and now () > issuance_ts at registration time. The value provided for issuance_ts **MUST** be an unsigned integer, interpreted according to POSIX time, representing the number of seconds since 1970-01-01T00:00Z UTC.
NoReplay	no_replay: null	If the no_replay attribute is present then the policy returns true if and only if the Signed Statement about to be registered doesn't already appear in the ledger. This policy has no required attributes.

Table 1: An Initial Set of Named Policies

6.4. Registering Signed Statements

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

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

2. Issuer identification. The Transparency Service **MUST** store evidence of the DID resolution for the Issuer protected header

of the Envelope and the resolved key manifest at the time of Registration for auditing. This **MAY** require that the service resolves the Issuer DID and record the resulting document, or rely on a cache of recent resolutions.

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

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

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

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

7. Sign and return the Receipt.

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

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

6.5. Validation of Transparent Statements

This section provides additional implementation considerations. The high-level validation algorithm is described in Section 5.3; the Registry-specific details of checking Receipts are covered in [I-D.birkholz-scitt-receipts].

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

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

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

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

7. FederationThis section needs work.Henk

Editor's note: This section needs work.

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

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

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

8. Transparency Service APIThis may be moved to appendix.Henk

Editor's Note: This may be moved to appendix.

8.1. Messages

All messages are sent as HTTP GET or POST requests.

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

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

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

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

As an example, submitting a signed statement with an unsupported signature algorithm would return a 400 Bad Request status code and the following body:

```
{
  "type": "urn:ietf:params:scitt:error:badSignatureAlgorithm",
  "detail": "The statement was signed with an algorithm the server does
}
```

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

- *Error code: malformed (The request could not be parsed).

Clients **SHOULD** treat 500 and 503 HTTP status code responses as transient failures and **MAY** retry the same request without modification at a later date. Note that in the case of a 503 response, the transparency service **MAY** include a Retry-After header field per [RFC7231] in order to request a minimum time for the client to wait before retrying the request. In the absence of this header field, this document does not specify a minimum.

8.1.1. Register Signed Statement

8.1.1.1. Request

POST <Base URL>/entries

Headers:

*Content-Type: application/cose

Body: SCITT COSE_Sign1 message

8.1.1.2. Response

One of the following:

*Status 201 - Registration is successful.

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

-Header Content-Type: application/json

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

*Status 202 - Registration is running.

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

-Header Content-Type: application/json

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

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

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

-Error code badSignatureAlgorithm

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

8.1.2. Retrieve Operation Status

8.1.2.1. Request

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

8.1.2.2. Response

One of the following:

- *Status 200 - Registration is running
  - -Header: Content-Type: application/json
  - -(Optional) Header: Retry-After: <seconds>
  - -Body: { "operationId": "<Operation ID>", "status": "running" }
- *Status 200 - Registration was successful
  - -Header: Location: <Base URL>/entries/<Entry ID>
  - -Header: Content-Type: application/json
  - -Body: { "operationId": "<Operation ID>", "status": "succeeded", "entryId": "<Entry ID>" }
- *Status 200 - Registration failed
  - -Header Content-Type: application/json
  - -Body: { "operationId": "<Operation ID>", "status": "failed", "error": { "type": "<type>", "detail": "<detail>" } }
  - -Error code: badSignatureAlgorithm
- *Status 404 - Unknown Operation ID
  - -Error code: operationNotFound
  - -This can happen if the operation ID has expired and been deleted.

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

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

8.1.3. Retrieve Signed Statement

8.1.3.1. Request

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

Query parameters:

- *(Optional) embedReceipt=true

If the query parameter embedReceipt=true is provided, then the signed statement is returned with the corresponding registration receipt embedded in the COSE unprotected header.

8.1.3.2. Response

One of the following:

- *Status 200.
  - -Header: Content-Type: application/cose
  - -Body: COSE_Sign1
- *Status 404 - Entry not found.
  - -Error code: entryNotFound

8.1.4. Retrieve Registration Receipt

8.1.4.1. Request

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

8.1.4.2. Response

One of the following:

- *Status 200.
  - -Header: Content-Type: application/cbor
  - -Body: SCITT_Receipt
- *Status 404 - Entry not found.
  - -Error code: entryNotFound

The retrieved Receipt may be embedded in the corresponding COSE_Sign1 document in the unprotected header, see draft-birkholz-scitt-receipts (TODO: replace with final reference).

9. Privacy Considerations

Unless advertised by a Transparency Service, every Issuer should treat Signed Statements it registered (rendering them Transparent Statements) as public. In particular, Signed Statement's Envelopes

and Statement payload should not carry any private information in plaintext.

10. Security Considerations

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

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

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

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

10.1. Threat Model

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

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

10.1.1. Signed Statement Authentication and Transparency.

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 the extensive periods of time, possibly decades.

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

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

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

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

In both cases, the SCITT Architecture provides generic, universally-verifiable cryptographic proof to individually blame Issuers or the Transparency Service. On the one hand, this enables valid actors to detect and disambiguate malicious actors who issue contradictory Signed Statements to different entities (Verifiers, Auditors, Issuers). 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, they cannot "frame" an Issuer or a Transparency Service for Signed Statements they did not issue or register.

10.1.1.1. Append-only Log

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

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

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

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

Verifier and Auditors may also witness (1,4) but may not be able to collect verifiable evidence for it.

10.1.1.2. Availability of Transparent Signed Statement

Networking and Storage are trusted only for availability.

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

Resistance to denial-of-service is implementation specific.

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

10.1.2. Confidentiality and privacy.

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

10.1.2.1. Signed Statements and Their Registration

The Transparency Service is trusted with the confidentiality of the Signed Statements presented for registration. Some Transparency Services may publish every Transparent 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 ledger only to auditors trusted with the confidentiality of its contents.

A collection of Transparent Statements leaks no information about the contents of other Transparent Statements registered at the Transparency Service.

Nonetheless, Issuers should carefully review the inclusion of private/confidential materials in their issued Signed Statements; they may for instance remove any PII, or include instead opaque cryptographic commitments, such as hashes.

10.1.2.2. Queries to the Registry

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

10.1.3. Cryptographic Assumptions

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

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

10.1.4. Transparency Service Clients

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

10.1.5. Identity

The identity resolution mechanism is trusted to associate long-term identifiers with their public signature-verification keys.

(Transparency Services and other parties may record identity-resolution evidence to facilitate its auditing.)

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

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

11. IANA Considerations

TBD; Section 4.

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

IANA is requested to register the URN sub-namespace urn:ietf:params:scitt in the "IETF URN Sub-namespace for Registered Protocol Parameter Identifiers" registry [IANA.params], following the template in [RFC3553]:

Registry name: scitt

Specification: [RFCthis]

Repository: http://www.iana.org/assignments/scitt

Index value: No transformation needed.

12. References

12.1. Normative References

[DID-CORE] W3C, "Decentralized Identifiers (DIDs) v1.0", 22 July 2022, <https://www.w3.org/TR/did-core/>.

[DID-WEB] "did:web Decentralized Identifiers Method Spec", n.d., <https://w3c-ccg.github.io/did-method-web/>.

[IANA.cose] IANA, "CBOR Object Signing and Encryption (COSE)", <https://www.iana.org/assignments/cose>.

[IANA.params] IANA, "Uniform Resource Name (URN) Namespace for IETF Use", <https://www.iana.org/assignments/params>.

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

[RFC3553] Mealling, M., Masinter, L., Hardie, T., and G. Klyne, "An IETF URN Sub-namespace for Registered Protocol Parameters", BCP 73, RFC 3553, DOI 10.17487/RFC3553, June 2003, <https://doi.org/10.17487/RFC3553>.

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

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://doi.org/10.17487/RFC7231>.

[RFC7807] Nottingham, M. and E. Wilde, "Problem Details for HTTP APIs", RFC 7807, DOI 10.17487/RFC7807, March 2016, <https://doi.org/10.17487/RFC7807>.

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

[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://doi.org/10.17487/RFC8610>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://doi.org/10.17487/RFC8949>.

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

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

12.2. Informative References

[I-D.birkholz-scitt-receipts] Birkholz, H., Riechert, M., Delignat-Lavaud, A., and C. Fournet, "Countersigning COSE Envelopes in Transparency Services", Work in Progress, Internet-Draft, draft-birkholz-scitt-receipts-02, 24 October 2022, <https://datatracker.ietf.org/doc/html/draft-birkholz-scitt-receipts-02>.

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

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

[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://doi.org/10.17487/RFC9334>.

Appendix A. Attic

Not ready to throw these texts into the trash bin yet.

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