Workgroup: TBD
Internet-Draft:
draft-ietf-scitt-architecture-00
Published: 8 December 2022
Intended Status: Standards Track
Expires: 11 June 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 claims on any Transparency Service (TS), 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 11 June 2023.

Copyright Notice

Table of Contents

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 claims; verifying this proof ensures that the issuer is consistent and non-equivocal when making claims.

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), and by requiring a

TS 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 TS implements and operates its Registry. Each service may enforce its own policy for authorizing entities to register their claims on the TS. Some TS 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 TS instances as the composition and configuration of involved supply chain entities and their system components is ever changing and always in flux.

A TS provides visibility into claims issued by supply chain entities and their sub-systems. These claims are called Digital Supply Chain Artifacts (DSCA). A TS vouches for specific and well-defined metadata about these DSCAs. Some metadata is selected (and signed) by the issuer, indicating, e.g., "who issued the DSCA" or "what type of DSCA is described" or "what is the DSCA version"; whereas additional metadata is selected (and countersigned) by the TS, indicating, e.g., "when was the DSCA registered in the Registry". The DSCA contents can be opaque to the TS, if so desired: it is the metadata that must always be transparent in order to warrant trust.

Transparent claims provide a common basis for holding issuers accountable for the DSCA they release and (more generally) principals accountable for auxiliary claims they make about DSCAs. Hence, issuers may register new claims about their artifacts, but they cannot delete or alter earlier claims, or hide their claims from third parties such as auditors.

Trust in the TS 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, we require each TS 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 TS specified in this architecture caters to two types of audiences:

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

2. DSCA Consumers: entities, stakeholders, and users involved in supply chain interactions that need to access, validate, and trust DSCAs.

DSCA Issuers rely on being discoverable and represented as the responsible parties for released DSCAs by the TS in a believable manner. Analogously, DSCA Consumers rely on verifiable trustworthiness assertions associated with DSCAs and their processing in a believable manner. If trust can be put into the operations that record DSCAs in a secure, append-only Registry via an online operation, the same trust can be put into a corresponding receipt that is the result of these online operations issued by the TS and that can be validated in offline operations.

The TS 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 TS 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 [RFC8152], which is used to sign released DSCAs and to build and maintain a Merkle tree that functions as the append-only Registry for DSCAs. 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 and 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.

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.

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

Issuer: an entity that makes Claims about Artifacts in the supply chain. The Issuer may be the owner or author of the Artifact, or an independent third party such as a reviewer or an endorser.

Envelope:
the metadata added to the Statement by the Issuer to make it a Claim. It contains the identity of the Issuer and other information to help Verifiers identify the Artifact referred in the Statement. A Claim binds the Envelope to the Statement. In COSE, the Envelope consists of protected headers.

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 Claims about the same Artifact. In COSE, Feed is a dedicated header attribute in the protected header of the Envelope.

Registry: the verifiable append-only data structure that stores Claims 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.

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 TS to provide many security guarantees about its Registry . The identity of a TS is captured by a public key that must be known by Verifiers in order to validate Receipts.

Receipt: a Receipt is a special form of COSE countersignature for Claims that embeds cryptographic evidence that the Claim is recorded in the Registry . It 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 protected headers) to assist in auditing.

Registration: the process of submitting a Claim to a Transparency Service, applying its registration policy, storing it in the Registry, producing a Receipt, and returning it to the submitter.

Registration Policy: the pre-condition enforced by the TS before registering a Claim, based on its Envelope (notably the identity of its Issuer) and on prior claims already in the Registry.

Transparent Claim: a Claim that is augmented with a Receipt of its registration. A Transparent Claim remains a valid Claim (as the Receipt is carried in the countersignature), and may be registered again in a different TS.

Verifier: an entity that consumes Transparent Claims (a specialization of Claim Consumer), verifying their proofs and

inspecting their Statements, either before using their Artifacts, or later to audit their provenance on the supply chain.

Auditor: an entity that checks the correctness and consistency of all Claim registered by a TS (a specialization of Claim Consumer).

4. Definition of Transparency

In this document, we use a definition of transparency built over abstract notions of Registry and Receipts. Existing transparency systems such as Certificate Transparency are instances of this definition.

A Claim 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. Claims can be made transparent by attaching a proof of Registration by a TS, in the form of a Receipt that countersigns the Claim and witnesses its inclusion in the Registry of a TS. By extension, we may say an Artifact (e.g. a firmware binary) is transparent if it comes with one or more Transparent Claims from its author or owner, though the context should make it clear what type of Claim 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.

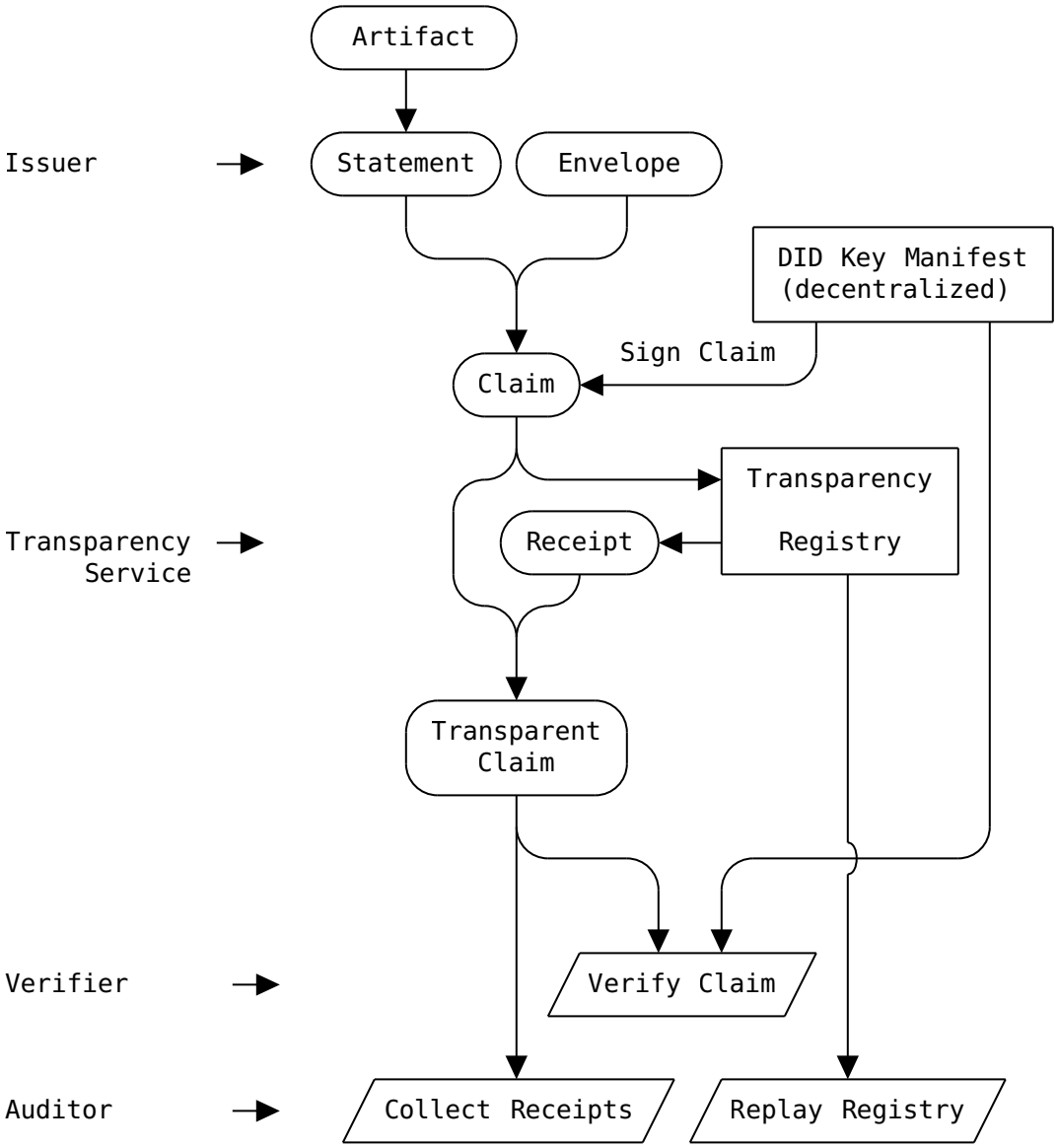
Transparency is implemented by a Registry that provides a consistent, append-only, cryptographically verifiable, publicly available record of entries. Implementations of TS may protect their Registry using a combination of trusted hardware, replication and consensus protocols, and cryptographic evidence. 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 that subsume older entries.

Anyone with access to the Registry can independently verify its consistency and review the complete list of Claims registered by each Issuer. However, the Registry of separate Transparency Services are generally disjoint, though it is possible to take a Claim 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 into Claims. Similarly, reputable TS

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 through Receipts, that is, given two valid Receipts the TS may be asked to produce a cryptographic proof that they are consistent. Failure to produce this proof can indicate that the TS 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 Claims. In order to accommodate as many TS implementations as possible, this document only specifies the format of Claims (which must be used by all Issuers) and a very thin

wrapper format for Receipts, which specifies the TS 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, we describe at a high level the three main roles and associated processes in SCITT: Issuers and the Claim issuance process, transparency Registry and the Claim Registration process, and Verifiers and the Receipt validation process.

5.1. Claim Issuance and Registration

5.1.1. Issuer Identity

Before an Issuer is able to produce Claims, it must first create its decentralized identifier (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 TS will be able to register their Claim. To facilitate interoperability, all Transparency Service implementations **SHOULD** support the did:web method from [https://w3c-ccg.github.io/did-method-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 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 Claims issued with those keys. This DID appears in the Issuer header of the Claim's Envelope, while the version of the key from the manifest used to sign the Claim is written in the kid header.

5.1.2. Naming Artifacts

Many Issuers issue Claims 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 Claim 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 Claims under the same key.

5.1.3. Claim Metadata

Besides Issuer, Feed and kid, the only other mandatory metadata in the Claim is the type of the Payload, indicated in the cty 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 Claim to be registered if it was signed recently. While the Issuer is free to add any information in the payload of the Claim, the TS (and most of its auditor) can only be expected to interpret information in the Envelope.

Such metadata, meant to be interpreted by the TS during Registration policy evaluation, should be added to the reg_info header. While the header **MUST** be present in all Claims, 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 TS. 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 (TS)

The role of TS can be decomposed into several major functions. The most important is maintaining a Registry, the verifiable data structure that records Claims, and enforcing a Registration policy. It also maintains a service key, which is used to endorse the state of the Registry in Receipts. All TS **MUST** expose standard endpoints for Registration of Claims and Receipt issuance, which is described in Section 8.1. Each TS 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 TS. Each of these components **SHOULD** be carefully protected against both external attacks and internal misbehavior by some or all of the operators of the TS. 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 Claims made by a given Issuer and Feed. Implementations of TS **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 TS **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 TS 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.

The TS **SHOULD** provide evidence that it is securely implemented and operated, enabling remote authentication of the hardware platforms and/or software TCB that run the TS. This additional evidence **SHOULD** be recorded in the Registry and presented on demand to Verifiers and auditors.

For example, consider a TS 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 TS that accepts to register any valid claim offered by an issuer would end up providing only limited value to verifiers. In consequence, a baseline transparency guarantee policing the registration of claims is required to ensure completeness of audit, which can help detect equivocation. Most advanced SCITT scenarios rely on the TS performing additional domain-specific checks before a claim is accepted: TS may only allow trusted authenticated users to register claims, TS may try to check that a new claim is consistent with previous claims from the same issuers or that claims are registered in the correct order and cannot be re-played; some TS may even interpret and validate the payload of claims.

In general, registration policies are applied at the discretion of the TS, and verifiers use receipts as witnesses that confirm that the

registration policy of the TS was satisfied at the time claim registration. TS 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 TS is opaque to the verifier, who is forced to trust the associated registration policy. If the policy of the TS evolves over time, or is different across issuers, the guarantee derived from receipt validation may not be uniform across all claims over time.

To help verifiers interpret the semantics of claim registration, SCITT defines a standard mechanism for signalling in the claim itself which policies have been applied by the TS from a defined set of registration policies with standardized semantics. Each policy that is expected to be enforced by the TS is represented by an entry in the registration policy info map (reg_info) in the envelope. The key of the map 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 claim 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 claim is registered, its main downside is that it requires the issuer to include the necessary policies in the envelope when the claim is signed. Furthermore, it makes it impossible to register the same claim on two different TS 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 claims but requires the verifier to be more aware of the particular policies at the TS where the claim 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. We only require the Registry 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, we do not specify any particular Registry format in this document. Instead, we propose two initial formats for Registry in [I-D.birkholz-scitt-receipts] using historical and sparse Merkle Trees. Beyond the format of Receipts, we require generic properties that should be satisfied by the components in the TS that have the ability to write to the Registry.

5.2.3.1. Finality

The Registry is append-only: once a Claim is registered, it cannot be modified, deleted, or moved. In particular, once a Receipt is returned for a given Claim, the Claim 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. TS 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 TS defines and enforces deterministic Registration policies that can be re-evaluated based solely on the contents of the Registry at the time of registraton, 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.

* The TS **SHOULD** store evidence about the resolution of distributed identifiers into manifests.

* The TS **MAY** additionally support verifiability of client authentication and access control.

5.2.3.4. Governance and Bootstrapping

The TS 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 Claims about code updates, secured on the Registry itself, or on some auxiliary TS).

Governance procedures, their auditing, and their transparency are implementation specific. The TS **SHOULD** document them.

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

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

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

5.3. Verifying Transparent Claims

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 TS.

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

Some Verifiers may systematically resolve the Issuer DID to fetch their latest DID document. This strictly enforces the revocation of compromised keys: once the Issuer has updated its document to remove a key identifier, all Claims signed with this 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 TS's Registration policy and the scrutiny of other Verifiers. Although this weakens their guarantees

against key revocation, or against a corrupt TS, they can still keep the Receipt and blame the Issuer or the TS at a later point.

6. Claim Issuance, Registration, and Verification

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

6.1. Envelope and Claim Format

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

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

All Claims **MUST** include the following protected headers:

* algorithm (label: 1): Asymmetric signature algorithm used by the Claim Issuer, as an integer, for example -35 for ECDSA with SHA-384, see COSE Algorithms registry;

* Issuer (label: TBD, temporary: 391): DID (Decentralized Identifier, see W3C Candidate Recommendation) 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 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, Claims **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,
  ? "sequence_no": uint,
  ? "issuance_ts": uint,
  * 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. Claim Issuance

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

- JSON-SPDX
- CBOR-SPDX
- SWID

*CoSWID

*CycloneDX

*in-toto

*SLSA

Once the Statement is serialized with the correct content type, the 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 Claim may only be registered on Transparency Services that implement the associated policy. For instance, if a Claim 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, the Issuer **MAY** use a standard COSE implementation to produce the serialized Claim (the SCITT tag of COSE_Sign1_Tagged is outside the scope of COSE, and used to indicate that a signed object is a Claim).

6.3. Standard registration policies

Editor's note

The technical design for signalling 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.

TS 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 claim can be registered. Any unsupported claims **MUST** be indicated separately and corresponding unknown policy entries in the map of a claim **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 attributes	Implementation
TimeLimited	register_by: uint	Returns true if now () < register_by at registration time. The ledger **MUST** store the ledger time at registration along with the claim, and **SHOULD** indicate it in receipts

Policy Name	Required attributes	Implementation
Sequential	sequence_no: uint	First, lookup in the ledger for claims with the same issuer and feed. If at least one is found, returns true if and only if the sequence_no of the new claim is the highest sequence_no in the existing claims incremented by one. Otherwise, returns true if and only if sequence_no = 0.
Temporal	issuance_ts: uint	Returns true if and only if there is no claim in the ledger with the same issuer and feed with a greater issuance_ts
NoReplay	None	Returns true if and only if the claim doesn't already appear in the ledger

Table 1: An Initial Set of Named Policies

6.4. Registering Signed Claims

The same Claim may be independently registered in multiple TS. To register a Claim, the service performs the following steps:

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

2. Issuer identification. The TS **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 resolve 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 has a payload and the protected headers listed above. The service **MAY** additionally verify the payload format and content.

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

6. Commit the new Claim to the Registry

7. Sign and return the Receipt.

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

The service **MUST** ensure that the Claim is committed before releasing its Receipt, so that it can always back up the Receipt by releasing the corresponding entry in the Registry. Conversely, the service **MAY** re-issue Receipts for the Registry content, for instance after a transient fault during Claim Registration.

6.5. Validation of Transparent Claims

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

Before checking a Claim, 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 Claim is registered on.

In some scenarios, the Verifier already expects a specific Issuer and Feed for the Claim, 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 TS during Registration.

Some Verifiers **MAY** decide to locally re-apply some or all of the Registration policies if they have limited trust in the TS. 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 payload, as well as any local state.

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

7. Federation

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

SCITT instances may be governed and operated by different organizations that do not trust one another.

This may involve registering the same Claims at different transparency services, each with their own purpose and registration policy. This may also involve attaching multiple Receipts to the same Claims, each Receipt endorsing the Issuer signature and a subset of prior Receipts, and each TS verifying prior Receipts as part of their registration policy.

For example, a supplier TS may provide a complete, authoritative Registry for some kind of Claims, whereas a consumer TS may collect different kinds of Claims to ensure complete auditing for a specific use case, and possibly require additional reviews before registering some of these claims.

8. Transparency Service API

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

8.1. Messages

8.1.1. Register Signed Claims

8.1.1.1. Request

POST <Base URL>/entries

Body: SCITT COSE_Sign1 message

8.1.1.2. Response

One of the following:

* HTTP Status 201 - Registration was tentatively successful pending service consensus.

* HTTP Status 400 - Registration was unsuccessful.

  - Error code AwaitingDIDResolutionTryLater

  - Error code InvalidInput

[TODO] Use 5xx for AwaitingDIDResolutionTryLater

The 201 response contains the x-ms-ccf-transaction-id HTTP header which can be used to retrieve the Registration Receipt with the given transaction ID. [TODO] this has to be made generic

[TODO] probably a bad idea to define a new header, or is it ok? can we register a new one? https://www.iana.org/assignments/http-fields/http-fields.xhtml

The 400 response has a Content-Type: application/json header and a body containing details about the error:

json { "error": { "code": "<error code>", "message": "<message>" } }

AwaitingDIDResolutionTryLater means the service does not have an up-to-date DID document of the DID referenced in the Signed Claims but is performing or will perform a DID resolution after which the client may retry the request. The response may contain the HTTP header Retry-After to inform the client about the expected wait time.

InvalidInput means either the Signed Claims message is syntactically malformed, violates the signing profile (e.g. signing algorithm), or has an invalid signature relative to the currently resolved DID document.

8.1.2. Retrieve Registration Receipt

8.1.2.1. Request

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

8.1.2.2. Response

One of the following:

- HTTP Status 200 - Registration was successful and the Receipt is returned.
- HTTP Status 400 - Transaction exists but does not correspond to a Registration Request.
  - Error code TransactionMismatch
- HTTP Status 404 - Transaction is pending, unknown, or invalid.
  - Error code TransactionPendingOrUnknown
  - Error code TransactionInvalid

The 200 response contains the SCITT_Receipt in the body.

The 400 and 404 responses return the error details as described earlier.

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

[TODO] There's also the GET <Base URL>/entries/<Transaction ID> endpoint which returns the submitted COSE_Sign1 with the Receipt already embedded. Is this useful?

9. Privacy Considerations

Unless advertised by the TS, every Issuer should treat its Claims as public. In particular, their Envelope and Statement should not carry any private information in plaintext.

10. Security Considerations

On its own, verifying a Transparent Claim 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 TS. If the Verifier trusts the Issuer, it can infer that the Claim 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 TS, it can independently infer that the Claim passed the TS Registration policy and that has been persisted in the Registry. Unless advertised in the TS Registration policy, the Verifier should not assume that the ordering of Transparent Claims in the Registry matches the ordering of their issuance.

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

Issuers **SHOULD** ensure that the Statements in their Claims are correct and unambiguous, for example by avoiding ill-defined or ambiguous formats that may cause Verifiers to interpret the Claim 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. In case key re-use is unavoidable, they **MUST NOT** sign any other message that may be verified as an Envelope.

10.1. Threat Model

We provide a generic threat model for SCITT, describing its residual security properties when some of its actors (identity providers, Issuers, TS, 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. Claim authentication and transparency.

SCITT primarily supports evidence of Claim integrity, both from the Issuer (authentication) and from the TS (transparency). These guarantees are meant to hold for the long term, possibly decades.

We conservatively suppose that some issuers and some TS will 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 signed Claim from a given Issuer, registered at a given TS (both identified in the Verifier's local authorization policy) and would not depend on any other Issuer or TS.

We cannot stop authorized supply chain actors from making false claims (either by mistake or by corruption) but we can make them accountable by ensuring their Claims are systematically registered at a trustworthy TS.

Similarly, we aim to provide strong residual guarantees against a faulty/corrupt TS. We cannot stop a TS from registering Claims that do not meet its stated Registration Policy, or to issue Receipts that are not consistent with their append-only Registry, but we can hold it accountable and guarantee that it will be blamed by any Auditor that replays their Registry against any contested Receipt. Note that SCITT does not require trust in a single centralized TS: different actors may rely on different TS, each registering a subset of claims subject to their own policy.

In both cases, SCITT provides generic, universally-verifiable cryptographic evidence to individually blame the Issuer or the TS. This enables valid actors to detect and disambiguate malicious actors who make contradictory Claims 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 for SCITT.

Verifiers and Auditors need not be trusted by other actors. In particular, they cannot "frame" an Issuer or a TS for claims they did not issue or register.

Append-only log

If a TS is honest, then a transparent signed Claim with a correct Receipt of registration at a given position ensures that the signed

claim passed its Registration Policy and was recorded at that position in its Registry.

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

An Auditor granted (partial) access to the 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 TS for them. This ensures any Verifier that trust at least one such Auditor that (2,3) will be blamed to the TS.

Due to the operational challenge of maintaining a globally consistent append-only Registry, some TS may provide limited support for historical queries on the Claims 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.

Availability of Transparent Signed Claims

Networking and Storage are trusted only for availability.

Auditing may involve access to data beyond what is persisted in the TS log. For example, the registered TS 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 Claims they issue, endorse, verify, or audit.

10.1.2. Confidentiality and privacy.

The network is untrusted. All contents exchanged between actors is protected using secure authenticated channels (TLS) but, as usual, this may not exclude network traffic analysis.

Claims and their registration

The TS is trusted with the confidentiality of the claims presented for registration. Some TS may publish every claim in their logs, to facilitate their dissemination and auditing. Others may just return receipts to the client that present claims for registration, and

disclose the ledger only to auditors trusted with the confidentiality of its contents.

A collection of transparent Claims leaks no information about the contents of other Claims registered at the TS.

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

Queries to the Registry

The confidentiality of queries is implementation-specific, and generally not guaranteed. For example, while offline Claim verification is private, a TS may monitor which of its Claims are being verified from lookups to ensure their freshness.

10.1.3. Cryptographic Assumptions

We rely on standard cryptographic security for signing schemes (EUF-CMA: for a given key, given the public key and any number of signed messages, the 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 claim in this log.)

SCITT 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. TS Clients

Trust in clients that submit Claims for registration is implementation-specific. Hence, an attacker may attempt to register any Claim it has obtained, at any TS that accepts them, possibly multiple times and out of order. This may be mitigated by a TS 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. (The TS and other parties may record identity-resolution evidence to facilitate its auditing.)

If one of the credentials of an Issuer gets compromised, SCITT still guarantee the authenticity of all claims signed with this credential that have been registered on a TS before the compromise. It is up to

the Issuer to notify TS of credential revocation to stop Verifiers from accepting Claims signed with compromised credentials. [See the thread of revocation for additional details.]

The confidentiality of any identity lookup during Claim Registration or Claim Verification is out of scope.

11. IANA Considerations

See Body Section 4.

12. References

12.1. Normative References

[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>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[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>.

[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>.

[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>.

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://www.ietf.org/archive/id/draft-birkholz-scitt-receipts-02.txt>.

[PBFT] Castro, M. and B. Liskov, "Practical byzantine fault tolerance and proactive recovery", ACM Transactions on Computer Systems, Volume 20, Issue 4 , November 2002, <https://doi:10.1145/571637.571640>.

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