Supply Chain Integrity, Transparency, and Trust (SCITT)

IETF#115 WG

10 November 2022

Note Well

Agenda

• Welcome and Introduction (5 min):	Chairs
• Problem Statement (5 mins):	Orie Steele
• Software Supply Chain Uses Cases (15min):	Yogesh Deshpande
• Architecture (30 mins):	Antoine Delignat-Lavaud
• SCITT Receipts (15 min):	Maik Riechert
• Hackathon Report (30 min):	Henk Birkholz, Orie Steele
• AOB (Open Mic) & Next Steps (15 min):	Chairs
• Wrap-up and Conclusion (5 min):	Chairs

Updates From Last Time

- Working group formed ▯ Thank you Roman
- Chairs selected
- Continue with our regular conference calls – switching to IETF tools

Supply Chains

Adam Hayes – Investopedia

“A supply chain is a network of individuals and companies who are involved in creating a product and delivering it to the consumer. Links on the chain begin with the producers of the raw materials and end when the van delivers the finished product to the end user.”

Problem Statement

Software is an inherent part of everyday digitally-enabled life, from smartphones to IoT to datacenters. Widely discussed attacks on the software supply chain have helped raise awareness of the risks.

Many other vulnerabilities highlight the need for greater visibility into supply chain integrity, transparency, and trust to make an informed decision.

Use Case:

Software Supply Chain focusing on SBOM as evidence to a claim

Transparency and Trust

“Transparency does not prevent dishonest or compromised Issuers, but it holds them accountable

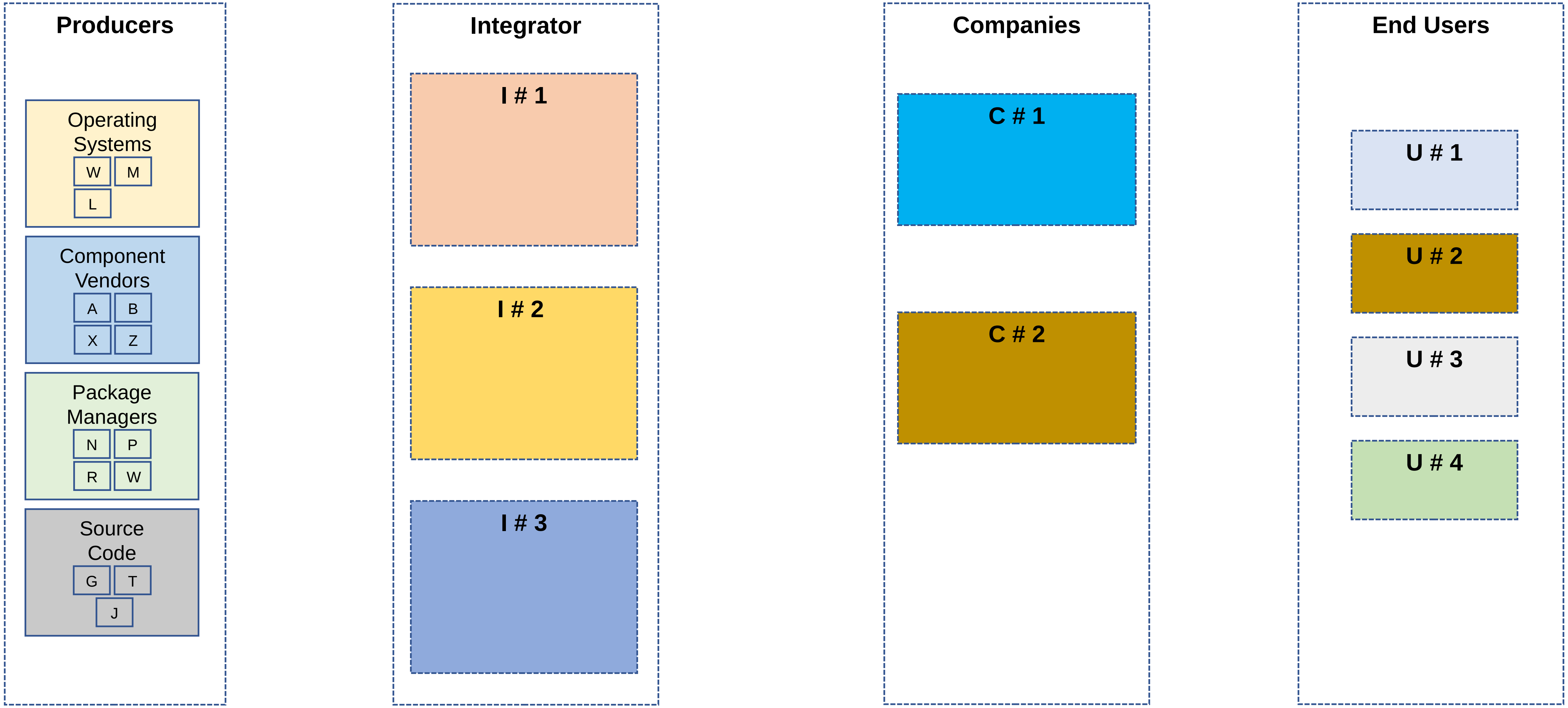
Any Artifact that may be used to target a particular user that checks for Receipts must have been recorded in the tamper-proof Registry,
and will be subject to scrutiny and auditing by other parties.”

SCITT: An Architecture for Trustworthy and Transparent Digital Supply Chains

Software Supply Chain Uses Cases

Yogesh Deshpande

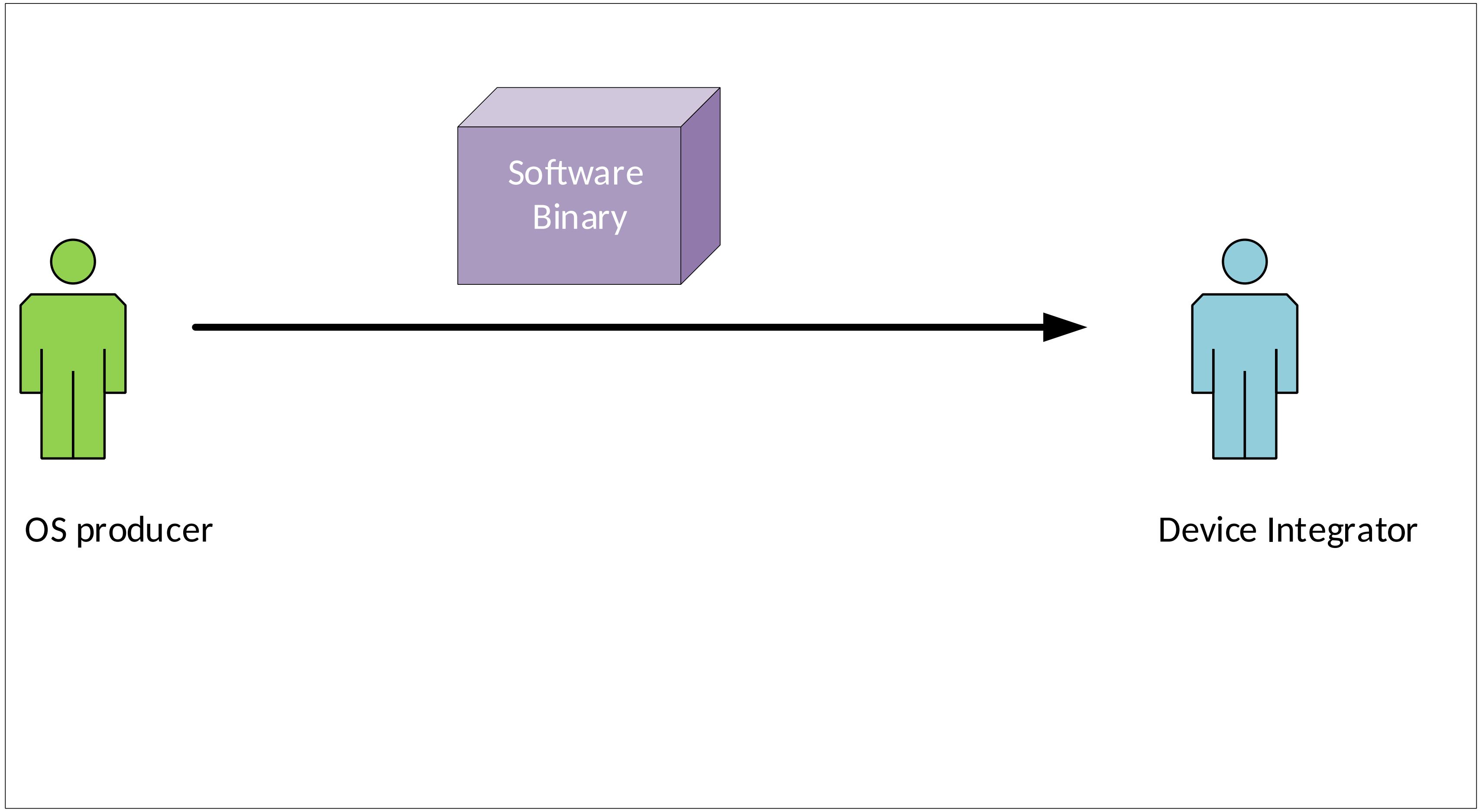
Software Supply Chain



Integrator problems

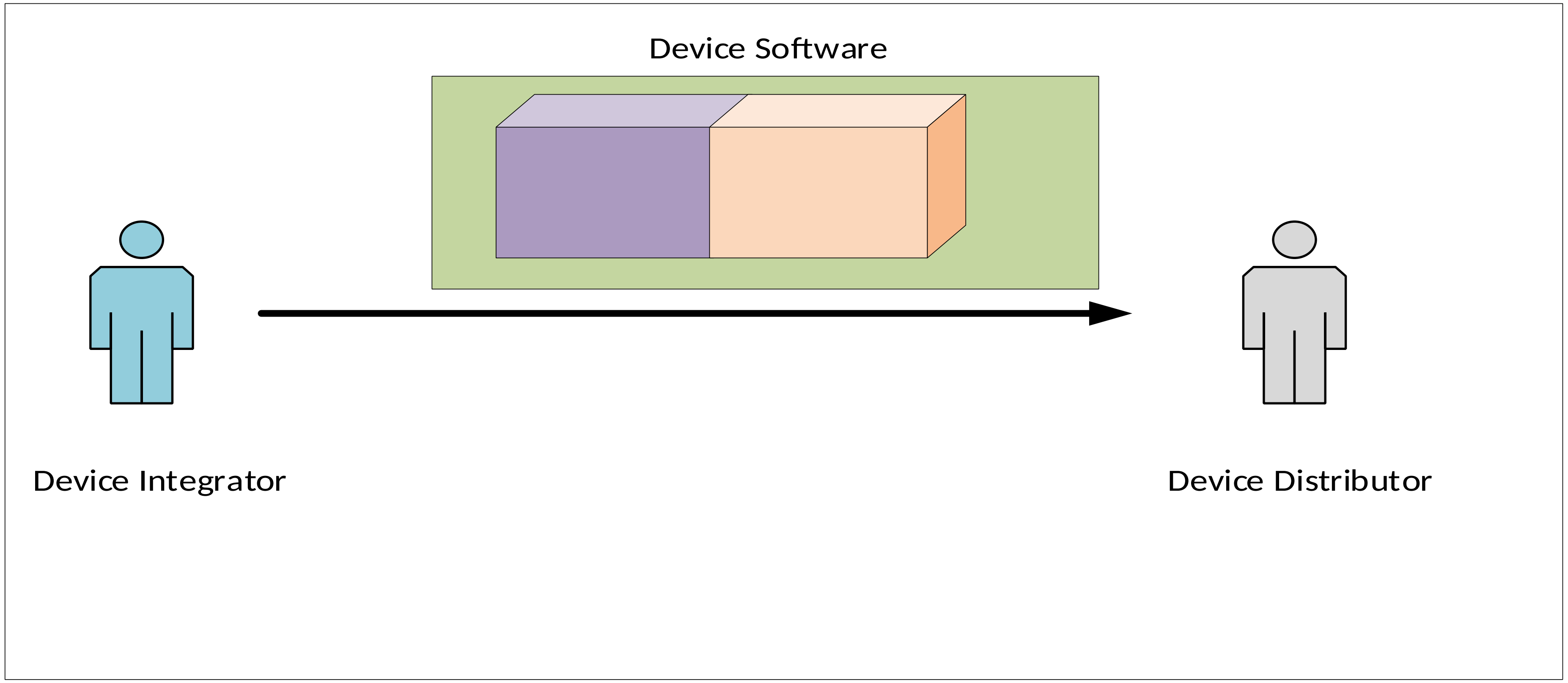
- Integrators need to know the current state of each published component
- Integrators have no standard way to receive this information
- Even if this information is delivered, via a standard way example: (SBOM) how can the integrator ensure:
  - The producer's identity is trustworthy by the integrator
  - The supplied component information is not unexpected modified or maliciously tampered with ?
  - How does an integrator deal with the situation, when a Malware is detected later?

Operating System Produced



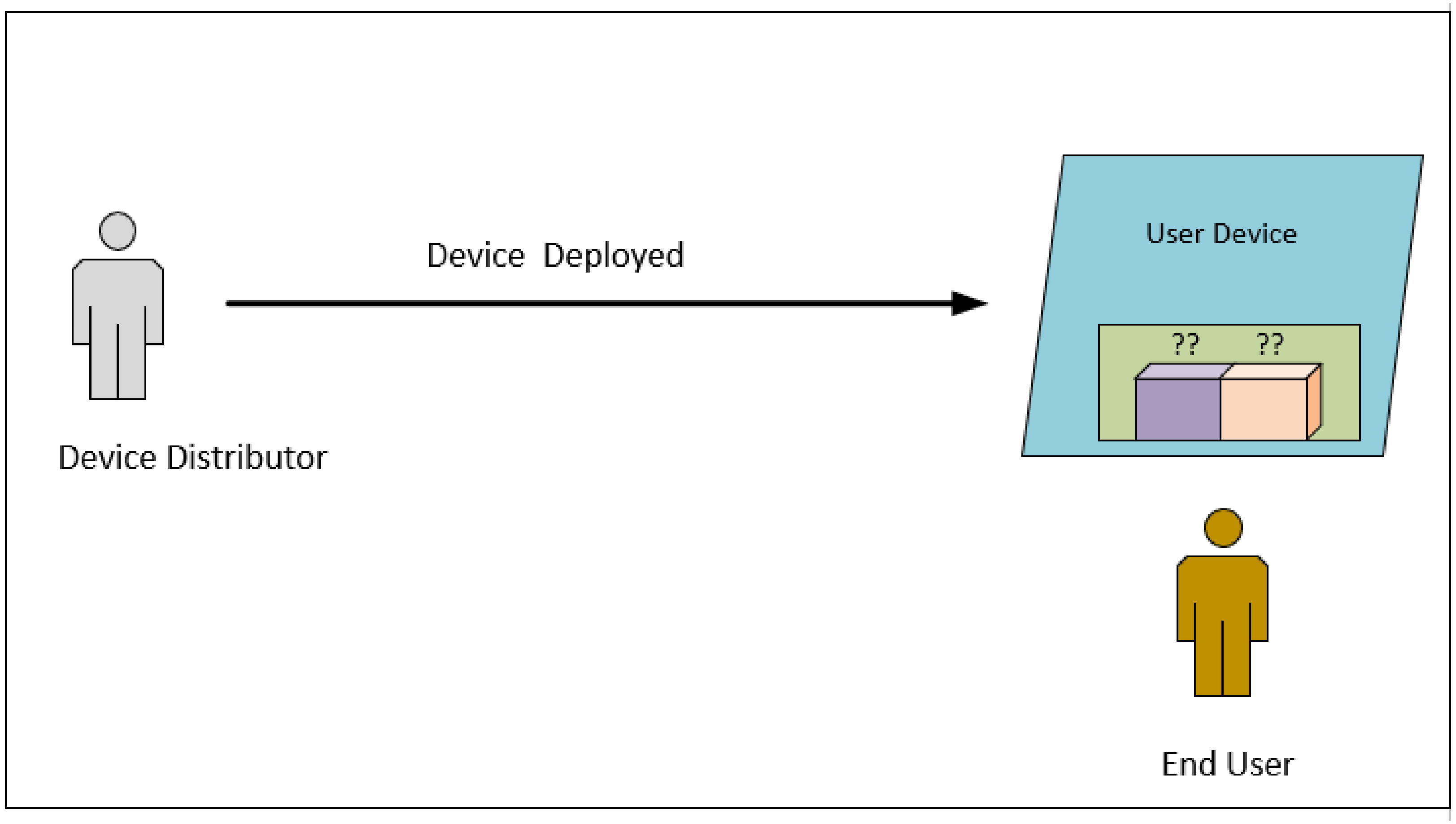
An operating system software producer has built and shipped binaries to a device integrator.

Device Executable Produced



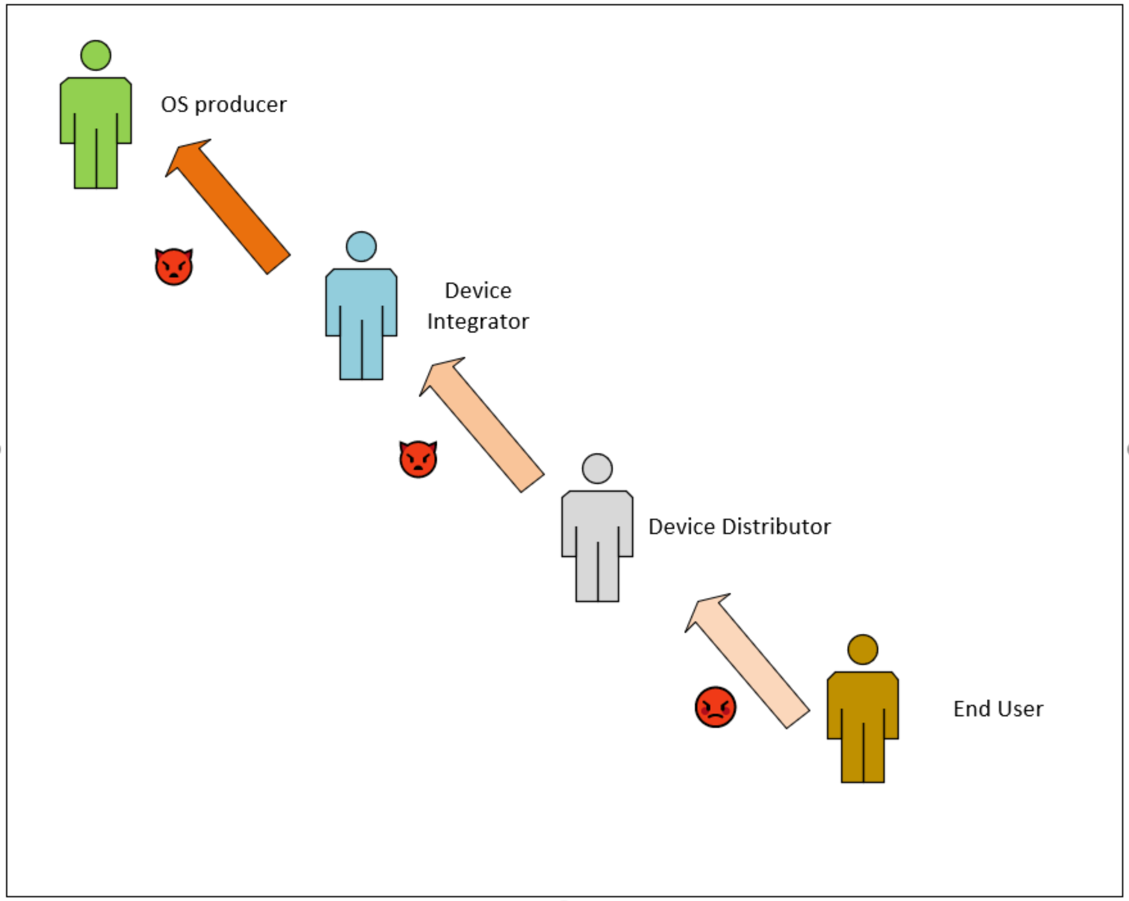
1. A device integrator has received an operating system binary and has integrated additional software components to produce a device executable
2. Device executable is shipped to the device distributor.

Device Deployed



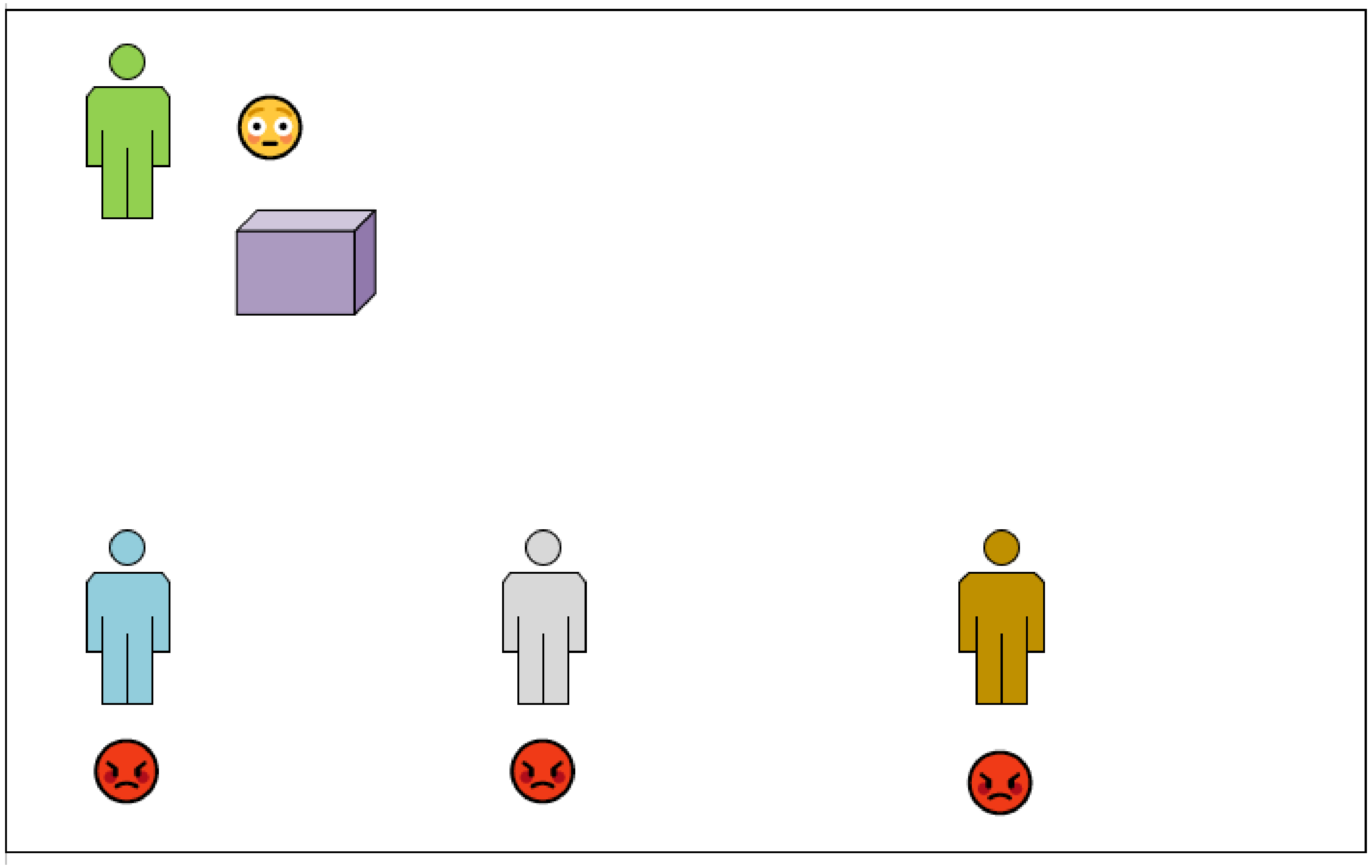
1. Later it is revealed there is a potential compromise when an anomaly is detected in devices sold by a particular distributor
2. The consumers of the device are complaining to the distributor.

Anomaly reported in distributed devices with specific device executables



1. The distributor in a panic demands information about the operating system and software loaded by the device integrator.
2. The device integrator panics and demands the specific environment and architecture details associated with the built operating systems binary to confirm that the software was produced without tampering by the operating system producer.

Confusion, Fear and Panic...



Unfortunately, there is no way for the integrator to know if the binary was compromised, and so the integrator is concerned they may have delivered malware (unknowingly) to their customers.

Deadlock continues …

1. The operating system software producer is now trying to show that it did all the steps correctly!

2. OS producer discloses information about the binary they delivered

3. They also demonstrated the Build Environment and the Architecture they used during the build

4. However, there is no verifiable proof of the statements made by the OS producer ??

5. The device integrator and the chain downline all the way up to the end user, now have to trust without any means of verifying the claims made by the OS producer ?

6. OS producer thinks that the Device Integrator has made a mistake that led to this situation ?

7. There is no clear resolution of this deadlock, as there is no standard means to conclude based on verification

Customer Problems

- Device distributors need to assess the risk of vulnerabilities in the binaries they distribute
- System integrators need to assess the risk of vulnerabilities in the
  - Software they produce
  - Binaries they consume from OS producers
- OS binary producers need to assess the risk of vulnerabilities in the
  - Software they produce
  - Software they consume from others
  - Binaries they distribute

What is SCITT

Supply Chain, Integrity & Trust (SCITT) is a set of specifications aimed at providing Integrity, Transparency and Trust in the exchange of products across end-to-end supply chains

IETF: SCITT Working Group

Defining compact, integrity protected protocols supporting interoperability across multi party supply chains

Focused on **software supply chain** use cases, including **firmware**, **package repositories, container registries, services**, …

We're inspired by COSE, RATs, TEEP, SUIT…

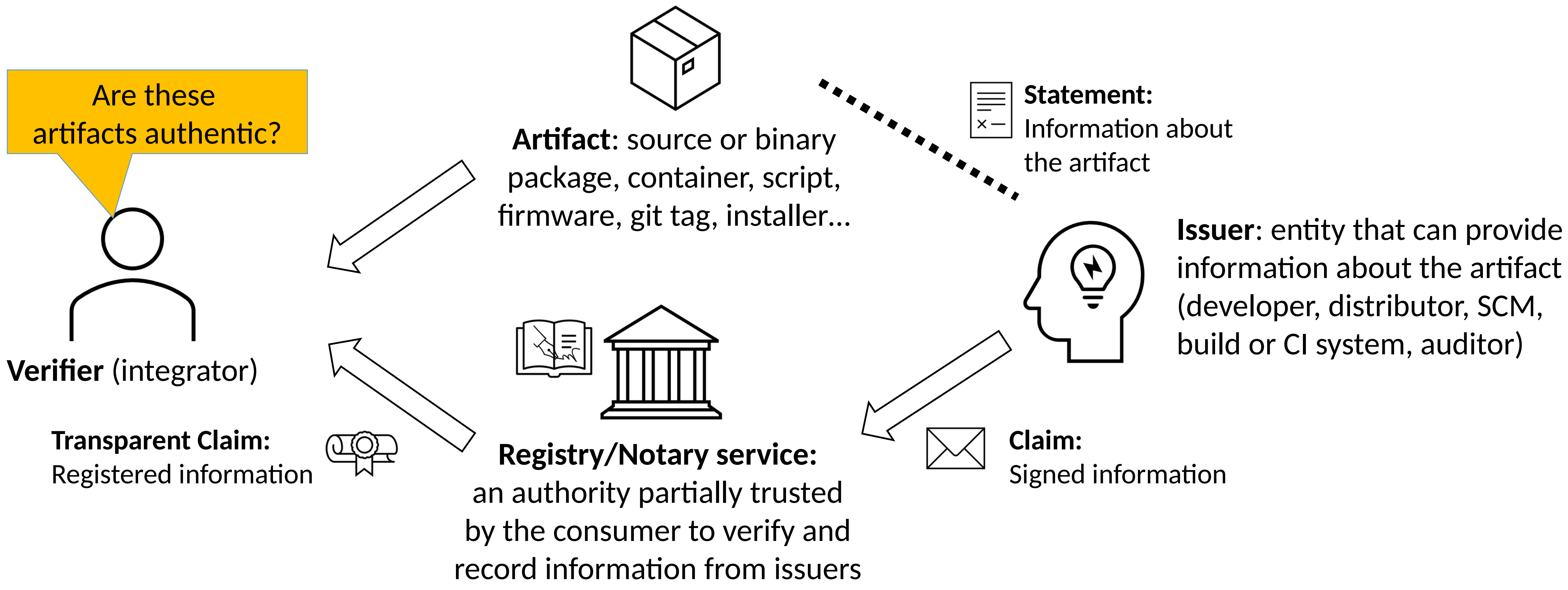
Outcome of 114 meeting for focus of execution

Hardware will come later, as a superset of software

Architecture

Antoine Delignat-Lavaud

Architecture Introduction – Core Concepts



Are these artifacts authentic?

Verifier (integrator)

Artifact: source or binary package, container, script, firmware, git tag, installer…

Statement: Information about the artifact

Issuer: entity that can provide information about the artifact (developer, distributor, SCM, build or CI system, auditor)

Claim: Signed information

Registry/Notary service: an authority partially trusted by the consumer to verify and record information from issuers

Transparent Claim: Registered information

https://datatracker.ietf.org/doc/draft-birkholz-scitt-architecture

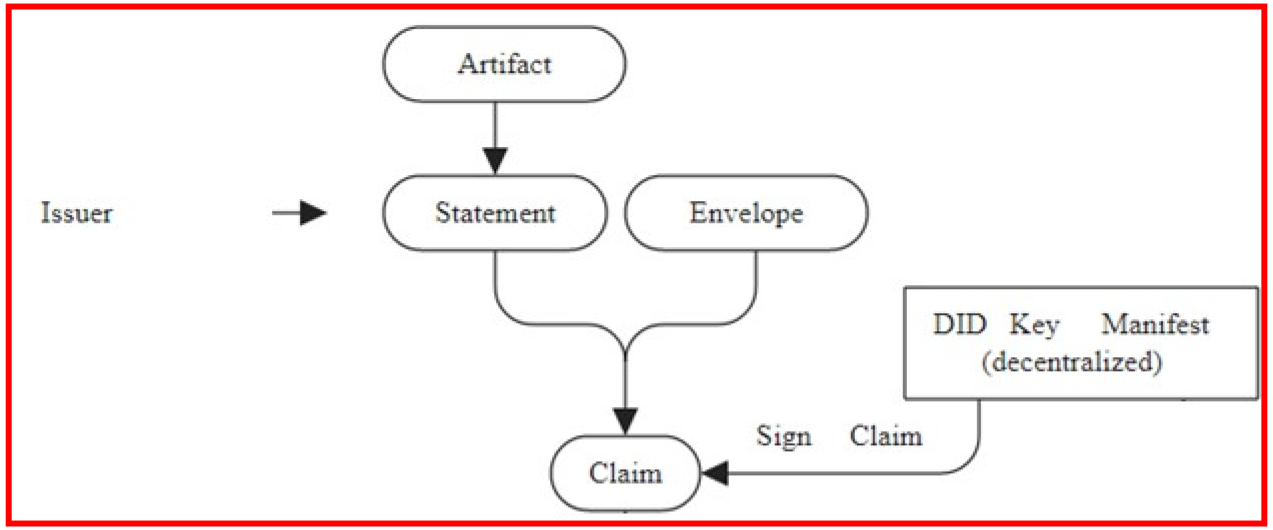
Claim Issuance

Sections 5.1, 6.2

- The issuer~~s~~ serializes a statement using any format of their choice (JSON, XML, SPDX, CycloneDX, SLSA, Reference to external storage…)

- The issuer publishes its signing key using any DID method. This provides a **stable long-term identifier** for the issuer independent of its short-lived credentials (certificates, signing keys, etc.)

- The issuer signs a SCITT claim, which is an instance of a COSE signed statement with specific additional headers:
  - **issuer** is the issuer's DID
  - **feed** is the identifier for the artifact the statement refers to, for instance a firmware image
  - **cty** is the format of the serialized statement (specified using mediatype)



COSE_Sign1

Header	Value
issuer	**did**:web:firmware.sec.fpga.com
alg	ES384
kid	20220101
feed	C910 FPGA Firmware
cty	application/x-c910-firmware-image
registration_info	timestamp, version number, …
Serialized Statement [COSE Payload]	
Signature 3045022100e7d0…	

Claim Registration

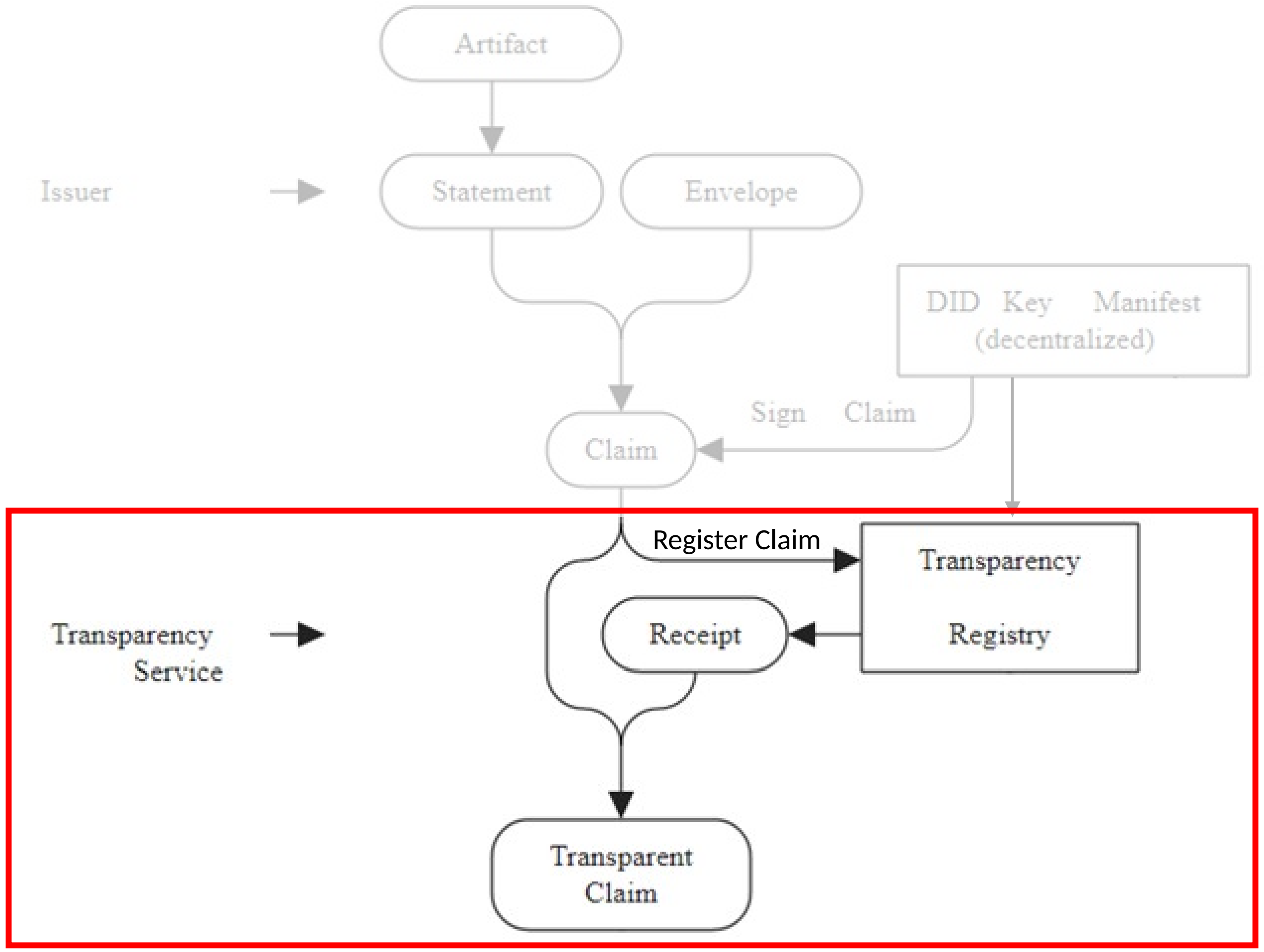
Sections 5.2, 6.4

- Some entity submits the **signed claim** to the registry

- The issuer is authenticated and checked against the registration policy to validate the COSE signature

- The registry may apply any additional policy checks including:
  - Issuer type and identity
  - Policies that depend on prior registered claims
  - Policies that inspect the cty and payload

- Registry returns a **SCITT receipt** as proof of registration

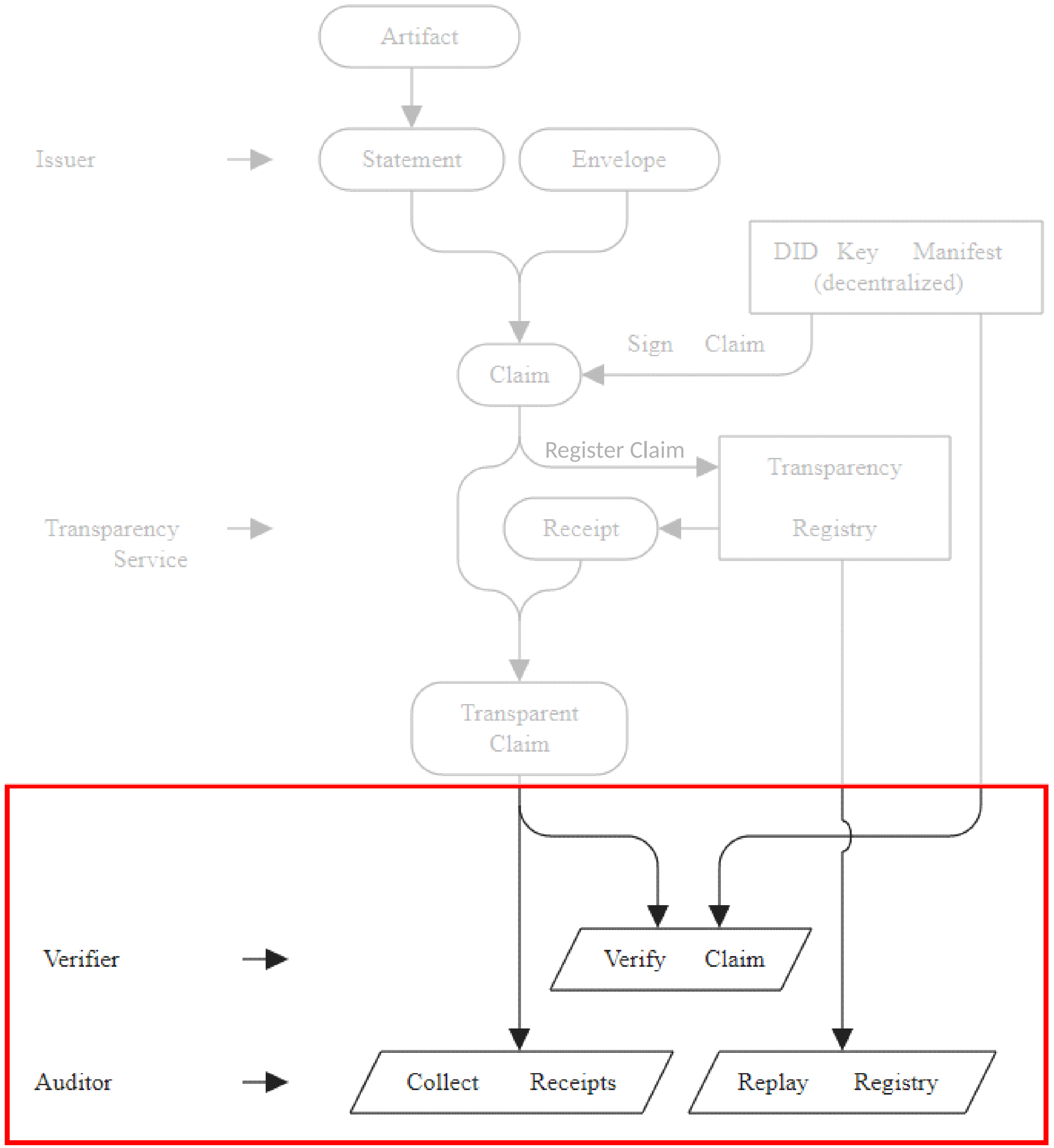


Validation & Audit

Sections 5.3, 6.5

- Most consumers of the artifact will pick a trusted notary. They check that they get a transparent claim associated with the artifact by validating the **SCITT receipt**
- Some sophisticated consumers may additionally check details of the claim, re-verify the issuer’s signature, or apply **additional policies** before they accept the artifact
- Finally, the most suspicious consumers (**auditors**) do not fully trust the notary. They may fully re-play the registration of all claims in the registry and examine collected receipts from verifiers

Should there be a standard way for auditors and verifiers to query the registry?



What Are the Security Goals of SCITT?

Sections 4, 10.1

Accountability of:

- **issuers**: the registry can be used to detect and blame issuers for incorrect or equivocal claims
- **notaries**: if users detect inconsistent receipts, or a receipt that does not comply with the service's registration policy, or a receipt inconsistent with the registry, or fail to retrieve or replay the registry, the service operator can be held accountable.

Auditability of:

- **claims**: if verifiers check receipts, all claims they may accept must be registered on the ledger
- **registration policies**: auditors can replay registration for all claims to verify the correct policy ed.

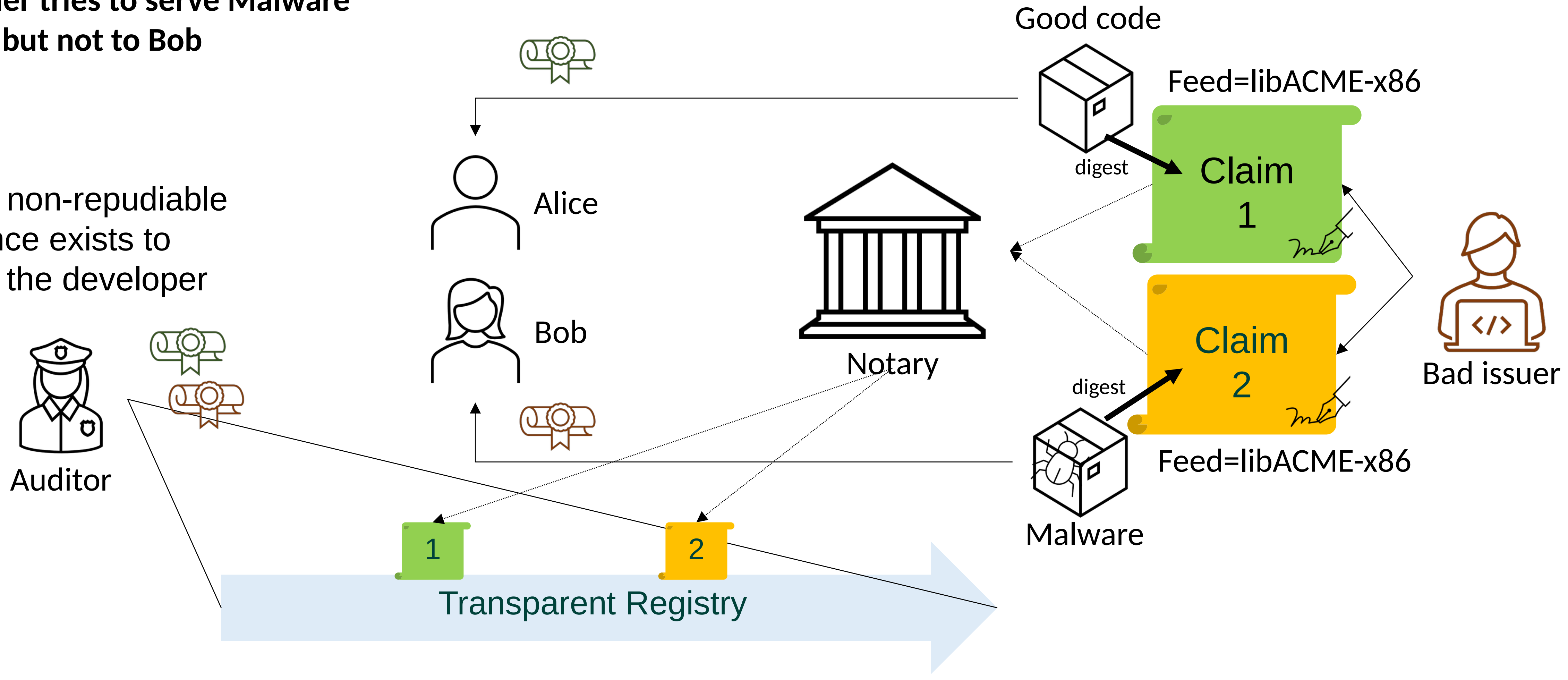
Separate threat model document

Issuer Accountability

Section 10.1.1

Bad issuer tries to serve Malware to Alice but not to Bob

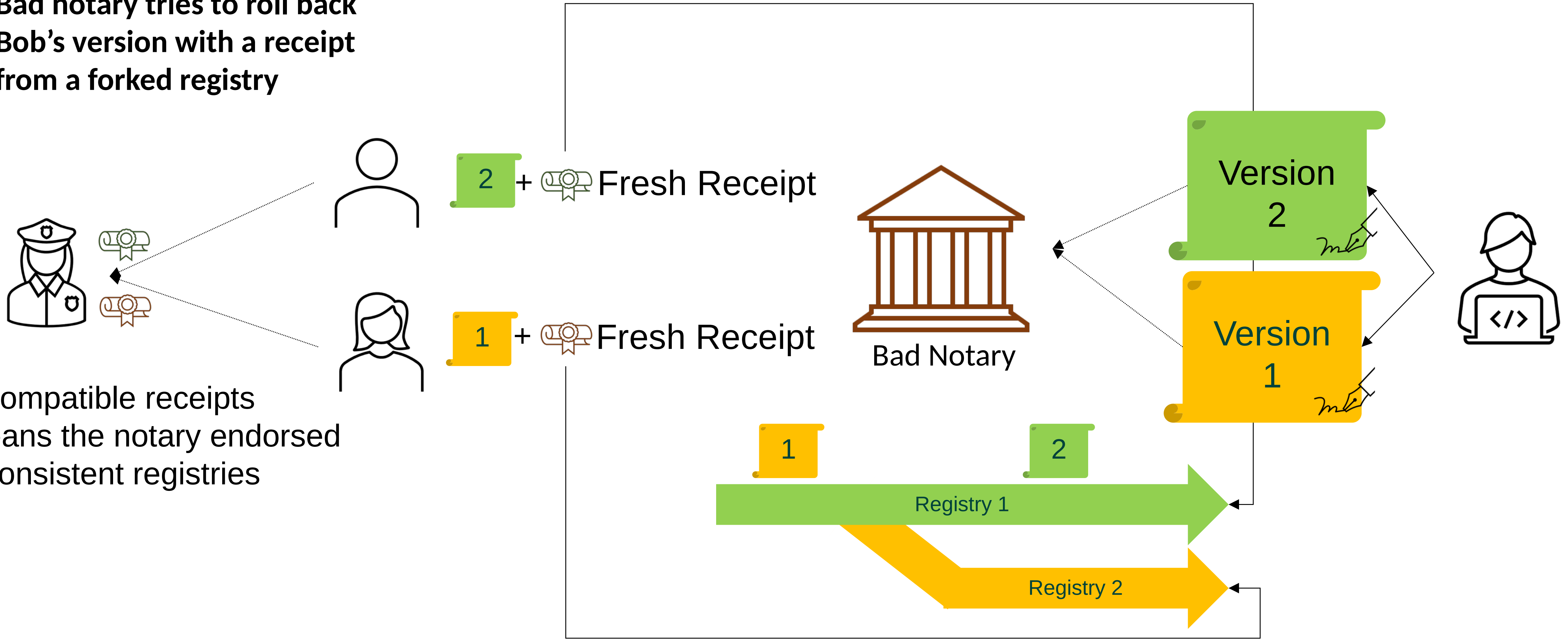
Public non-repudiable evidence exists to blame the developer



Transparency Service Accountability

Section 10.1.1

Bad notary tries to roll back Bob's version with a receipt from a forked registry



Incompatible receipts means the notary endorsed inconsistent registries

Updated in draft 02

SCITT Policies & Standardization

Section 5.2.2, 6.3

Should I sign this statement?

Issuance Policy

- Is the statement valid, and correct?
- Do I accept responsibility for it?
- What format do I use to serialize the statement?
- What artifact does it refer to?

Unspecified: issuers can enforce any policy they want

Should I register this claim?

Registration Policy

- Is this issuer trustworthy?
- Do I accept this type of claim?
- Is the submitter authenticated or authorized to use this registry?
- Is the claim recent enough?
- Does this claim replace an existing one?
- Is it spam / too large for ledger?

Standardized: some policies should have universal semantics for all verifiers

Should I accept this receipt?

Validation Policy

- Do I trust the registry that issued the receipt?
- Do I re-check the issuer's signature?
- Do I check the freshness of the claim or of its receipt?
- Do I look at information in the statement?

Unspecified: verifiers can enforce any policy they want

SCITT Standard Registration Policies

Followed & notified by issuers

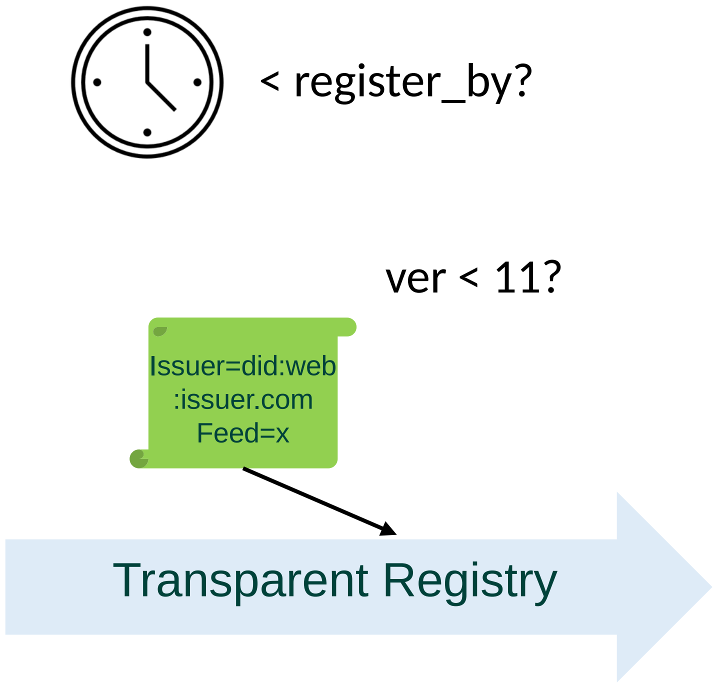
Header	Value
Issuer	did:web:issuer.com
Feed	X
…….	
Registration info	

Key	Value
register_by	221201120000
ver	11
…	

Payload

Signature 3045022100e7d0…

Enforced by notaries



Benefits all verifiers, regardless of where the claim is registered

Claim 1

Header	Value
Issuer	did:web:issuer.com
Feed	X
…….	
Registration info	

Key	Value
ver	10

Payload 1

Claim 2

Header	Value
Issuer	did:web:issuer.com
Feed	X
…….	
Registration info	

Key	Value
ver	11

Payload 1

Claim 2 > Claim 1 in registry

Should there be a mechanism for notaries to indicate to verifiers which registration policies were applied without involving issuers?

How Does the Architecture Meet Customer Requirements?

Customer Requirements: Supply chain parties need to assess the risk of software vulnerabilities over time

Alt: *Supply chain producers and consumers wish to communicate a secure and constant stream of updates for the state of their software*

Architecture supports uniform methods the following:

- issuing statements about software
- verifying the integrity, authenticity, and timeliness of statements
- creating consistent, append-only, verifiable records of statements

User defined

- Policy for determining which statements are trusted
- Assessments of risk based on trusted statements

SCITT Receipts

Maik Riechert

What Is a SCITT Receipt?

- Proof that a SCITT claim has been successfully registered in a Transparency Service

- Registration means:
    - Apply registration policies (at minimum, verify claim issuer/identity)
    - Store the claim in the registry
    - Produce a receipt
    - Return the receipt to the submitter
    - Receipt enables offline verification of the claim

Why Are Receipts Like Countersignatures?

- SCITT Transparency Service acts ***like*** an electronic notary
  - Certifies the authenticity of the claim signature
  - Certifies any additional registration policies apply to the claim

- On a technical level, treating it ***like*** a countersignature:
  - Allows embedding of receipts in the *unprotected* header of the claim
  - Instead, re-use elements of COSE countersignatures (`Countersign_structure`)

Why Do We Need a New Format?

- Isn’t a standard COSE countersignature enough?
- Not quite, because Transparency Services don’t countersign claims
- Rather, they sign a Merkle tree root

- Proposed/abandoned to hide those details in existing COSE concepts:
  - New COSE algs and structured “signature”
  - Discussed on the community mailing list
    - Need more transparency (*mailing list discussion*) [1]

1: https://mailarchive.ietf.org/arch/msg/cose/u0snos1yKnTQHwdNMOY6g8PevXk/

SCITT Receipt Format (DRAFT)

```
Receipt = [
  protected: bstr .cbor { * label => value },
  proof: any
]
```

- Each receipt has a protected header from the Transparency Service:
  - Service ID (lookup of service parameters)
  - Tree Algorithm (CCF, QLDB, Trillian, …)
  - Issued At (time of registration)
- `proof` type depends on the tree algorithm parameter

Should Receipts Become a New Standard COSE Message Type?

- Discussed in COSE WG on Tuesday
- No immediate objection
- Will continue working on receipts and engage with COSE WG

Hackathon Report

Henk Birkholz, Orie Steele

Detached COSE Envelopes

- RFC 9052 says: "One feature that is present in CMS that is not present in this standard is a digest structure."
- COSE_Hash_Find in RFC 9054 defines:
- **`COSE_Hash_Find = [`**
- **`    hashAlg : int / tstr,`**
- **`    hashValue : bstr`**

  `]`
- **Plus**: Location hints via resolvable {C/U}RI or{C/U]RL
- **Proposal**:
  Define Detached COSE Envelopes for Signed Statements for SCITT applications

Detached COSE Envelopes

Why?

- 1st: Content-Addressable Storage support
- 2nd: Statements can be BIG!
- 3rd: Statements can be confidential or require scoped access-control
- 4th: Legal or Personally Identifiable Information (PII) requirements

We Are Not the Curators of a Thousand Semantics

- Most Statements (from one or multiple Signed Statement Issuers) are semantically related
- Dependencies are impacted by the Statements, yet statement payload is agnostic to SCITT
- **Proposal**:
  Define a **small set** of named and registered standard Statement structures
  (for example: see ‘Detached COSE Envelopes’)

We Are Not the Curators of a Thousand Semantics

Why?

- There are Statement types that appear to be somewhat universal, such as
  - Revocation (Statement refresh)
  - Endorsement
- In support of system design and dependency graph construction

References to Transparent Statements

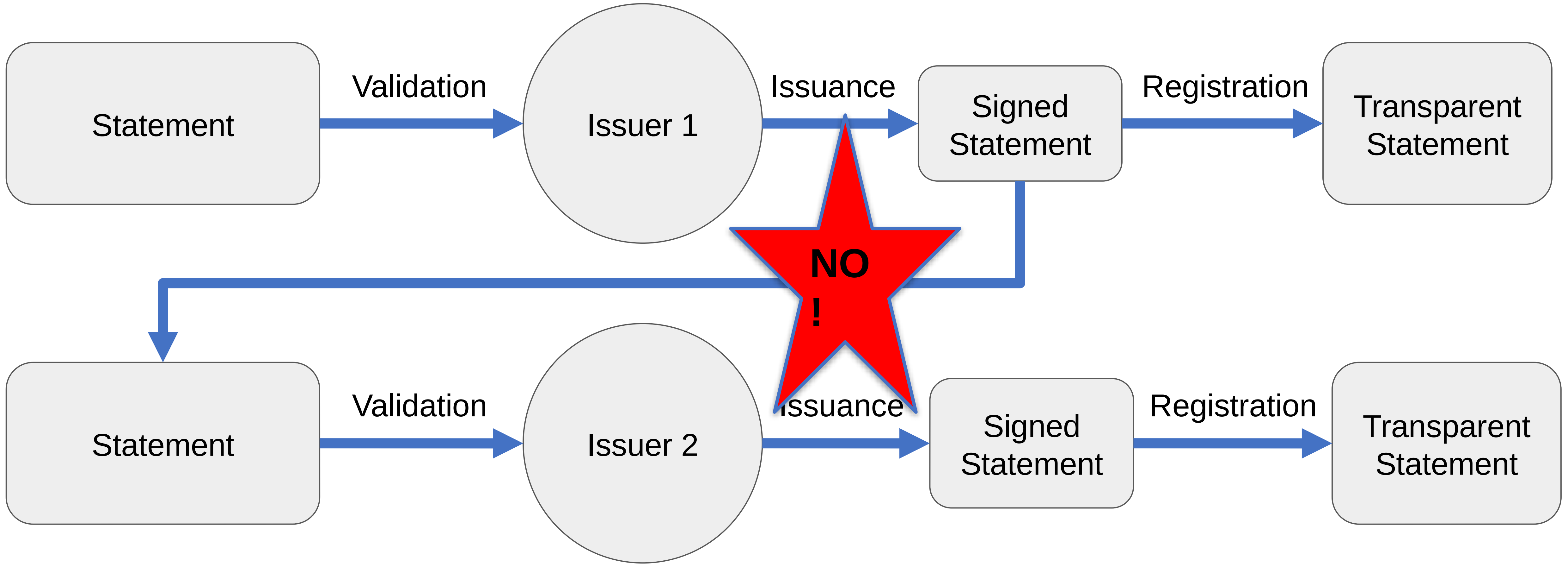
- Most semantics can be clumped into a "**refers-to**" relationship
- Semantics related to standardized, registered Statement structures can be **inferred by Statement content**
  (see ' Curators of a Thousand Semantics ')
- **Proposal:**
  For agnostic statements or detached statement content
  a type of pointer/reference is useful and should be defined
  (see ‘Detached COSE Envelopes’)

References to Transparent Statements

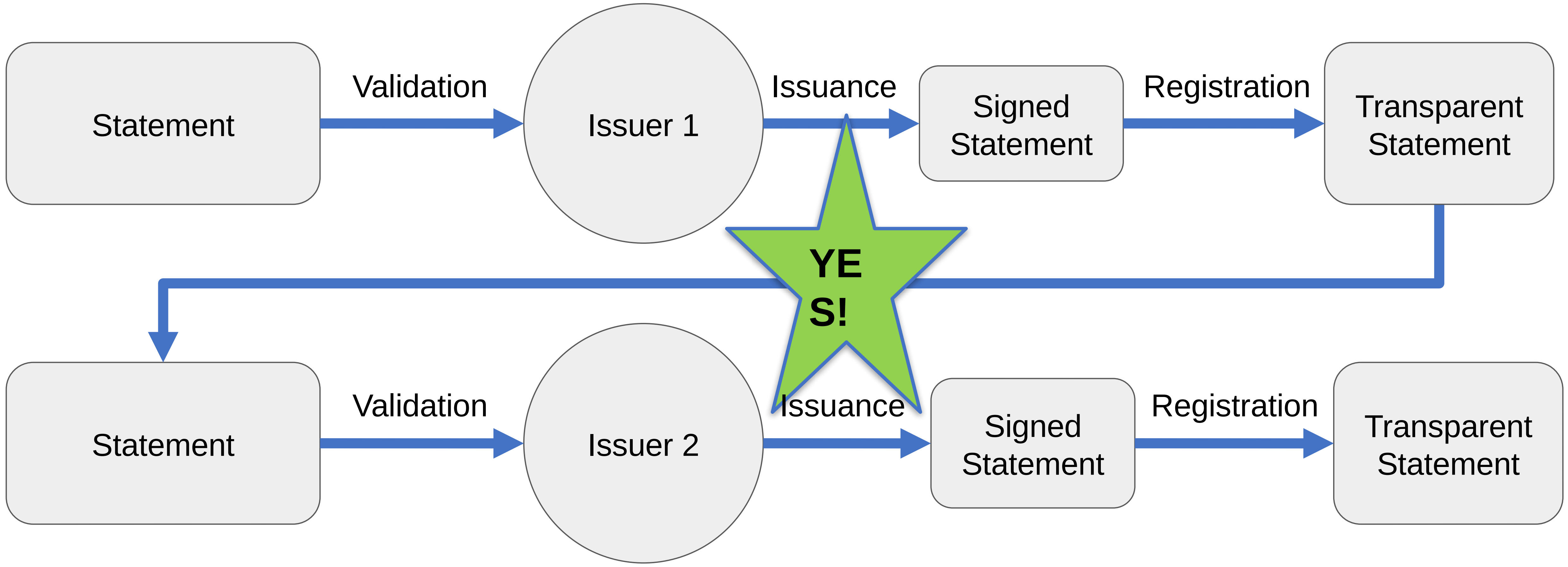
Why?

- Supporting systems and application use cases is vital in SCITT
- Consumers want to query auditable, interconnected statements
- Many use cases rely on generation of directed property graphs composed of Transparent Statement
  - **Example:**
    Remediation Guidance for a Vulnerability Disclosure Report
    for an SBOM for a software artifact SWID

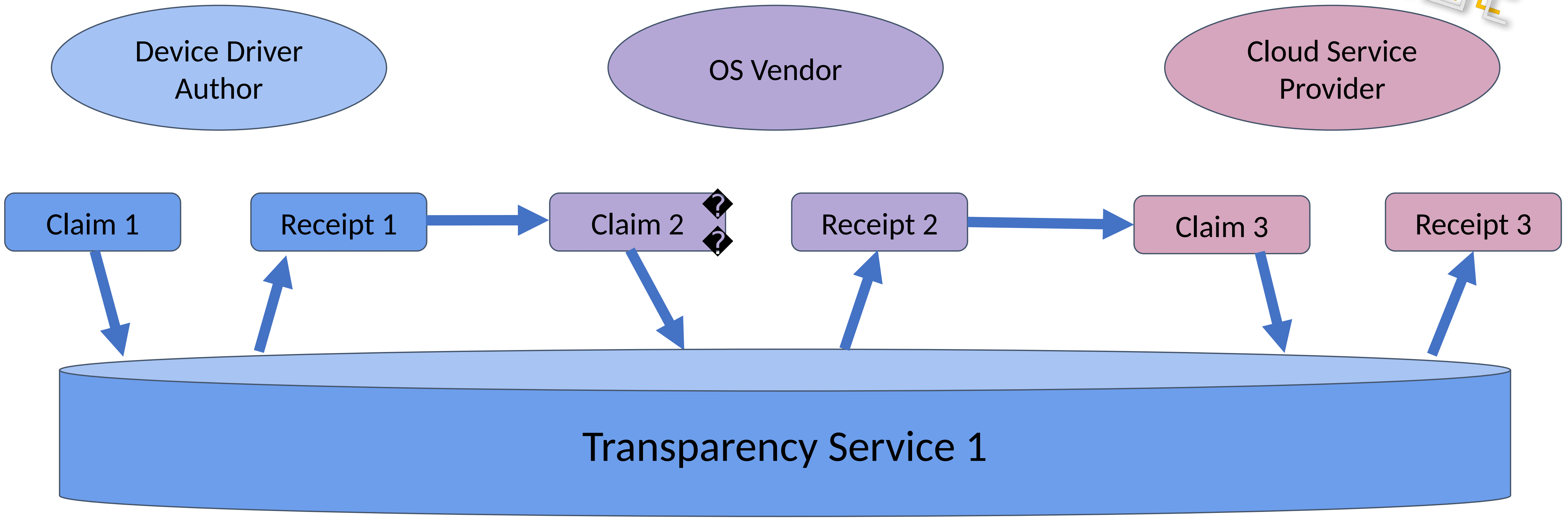
Vital Lesson on Statement References



Vital Lesson on Statement References

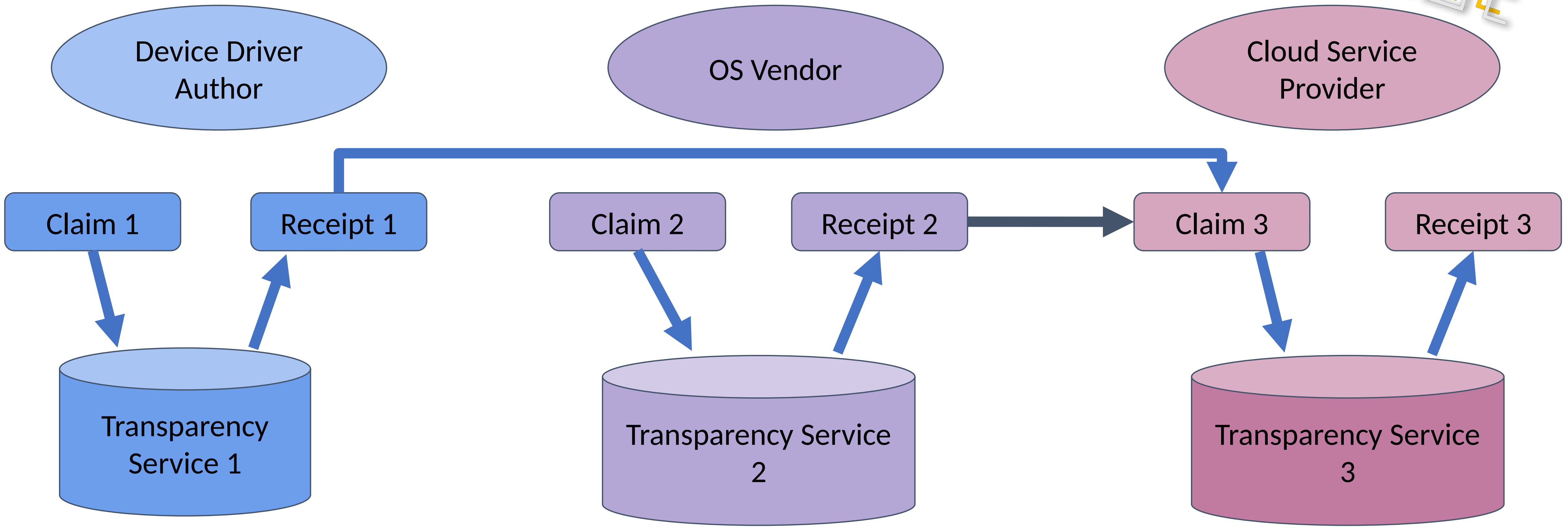


Vital Lesson on Scoping Examples: "Auditor's View"



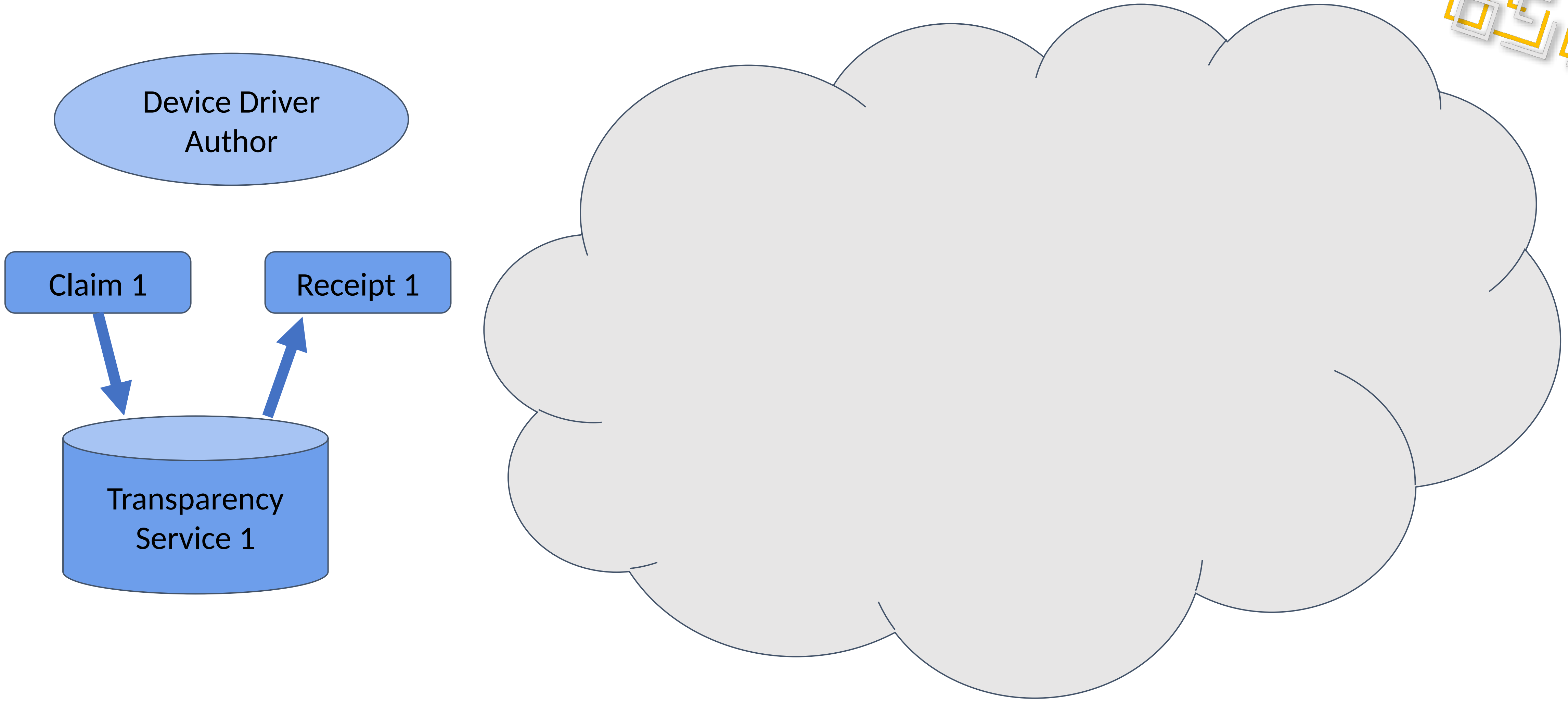
Claim1: GPU hardware version 1.2.3, Firmware measurements sha256:0xdeadbeef, FIPS certified.
Claim2: Vendor name XYZ, FIPS compliant build version "FIPS-4.5.6"
Claim3: Receipt 1, Receipt 2, Workload Measurements, and SLA promises

Vital Lesson on Scoping Examples: "Auditor's View"



Claim1: GPU hardware version 1.2.3, Firmware measurements sha256:0xdeadbeef, FIPS certified.
Claim2: Vendor name XYZ, FIPS compliant build version "FIPS-4.5.6"
Claim3: Receipt 1, Receipt 2, Workload Measurements, and SLA promises

Vital Lesson on Scoping Examples “Device Driver’s View”



AOB (Open Mic) & Next Steps

Wrap-Up and AOB

Next Steps

- Continued participation on mailing list and in community meetings
  - Mailing List
  - Community Meetings
- Review related IETF drafts
  - Countersigning COSE Envelopes in Transparency Services
  - An Architecture for Trustworthy and Transparent Digital Supply Chains
- Resources
  - scitt-api-emulator
  - scitt-ccf-ledger