Current State of Affairs

Henk Birkholz

Meanwhile…

- Updated Internet-Draft
  - An Architecture for Trustworthy and Transparent Digital Supply Chains
  - Detailed Software Supply Chain Use Cases
- Related Internet-Drafts
  - ~~Countersigning COSE Envelopes in Transparency Services~~ (Receipts)
  - Concise Encoding of Signed Merkle Tree Proofs

New Challenges

- Merkle Trees Proofs are not specific to SCITT or even services.
  - **New Challenge:** Creating a Merke Tree Proof I-D that SCITT Receipts can be a profile of
- The way how SCITT Signed Statements and SCITT Receipts are using COSE in specific ways are not specializations of COSE-use only required by SCITT.
  - **New Challenge:** Creating a more generic way to profile the use of COSE for various uses – one of them being the specific usage of COSE in SCITT
- All text about API specification is not specific to SCITT but of general interest to all systems considering to use SCITT as building blocks.
  - **New Challenge**: Creating a dedicated I-D specifying operations and a reference API to address use case requirements (using all other SCITT I-Ds and maybe more!)

The Numbers & The Process

- PR #16 (Vast Terminology Overhaul) created **113** comments:
  1. That's a lot!
  2. There is no doubt that there is interest in the topic.
  3. The editors were able to extract ~30 github issues from that input.
  4. If **YOUR** comment of interest was lost in transition, **PLEASE** raise an issue by your own again. The whole WG, all authors, contributors, and supports value your feedback. If something got, lost that was not intentional. (we are trying to keep up with the immense feedback, but please help!)
  5. In the future, try to create **individual PR**s, or create **Suggestions on a PR**. (If you are not sure how to do that, the editors and chairs will help every single contributor getting familiar with that procedure!)

And then again... Terminology

If you can't say it right, how should you get it right?

- The thing that an Issuer creates about a supply chain thing of interest?
  - An Issuer produces a **Signed Statement** (message to be added to **Append-Only Log)** from **Statements** about supply chain **Artifacts**
- Registry, Ledger, Log... the thing all Signed Statements will end up in?
  - **Append-Only Log**
- A Signed Statement that carries a corresponding Receipt in it's unprotected header?
  - **Transparent Statement**
- **Next Step:** Getting consensus on **Artifact** (the subject of interest moving along the supply chain that the Issuer maintains statements about)
- **Next Step:** Mapping and/or aligning the semantics of RATS **Endorsements** with SW Supply Chain **Audit Results**

Software Supply Chain Uses Cases

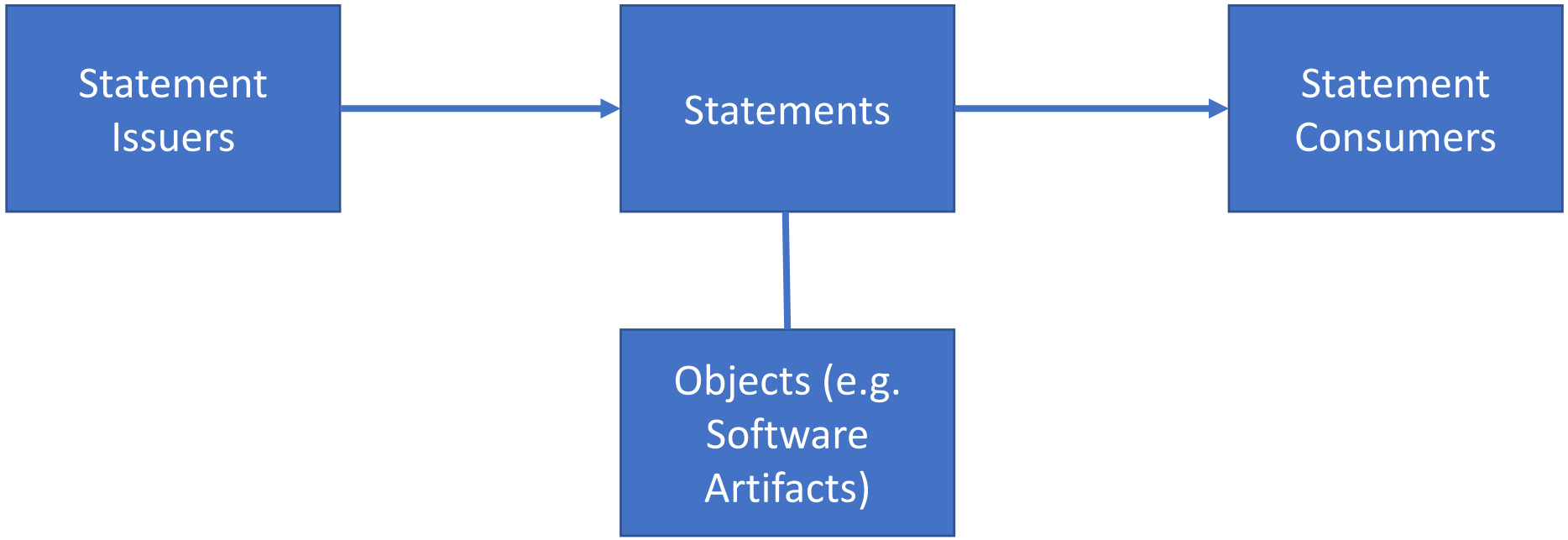
Kay Williams

Use Cases

- Verification that Signing Certificate is Authorized by Supplier
- Multi Stakeholder Evaluation of a Released Software Product
- Security Analysis of a Software Product
- Promotion of a Software Component by multiple entities
- Post-Boot Firmware Provenance
- Auditing of Software Product
- Authentic Software Components in Air-Gapped Infrastructure
- Firmware Delivery to large set of constrained IoT Devices
- Software Integrator assembling a software product for a smart car

Detailed Software Supply Chain Use Cases

General Workflow



Statement Issuers

- Statement Issuers
  - Individuals, organizations, processes
  - Examples - developers, source control management systems, build systems, CI/CD systems, packaging systems, release management systems, auditors, analysts
- Activities
  - Make statements about software artifacts
  - Relate statements to other statements
  - Make statements to update, augment, or invalidate other statements
  - Distribute statements

Statements

- Who produced the software artifact
- What is contained in the software artifact
- Who distributed the software artifact
- What assessments were performed on the software artifact
- Weakness or vulnerabilities identified in a software artifact
- Mitigations to vulnerabilities
- Corrections, updates, augmentations or revocations of previous statements
- Relationships between statements or software artifacts

Objects

Objects (e.g. Software Artifacts)

- Software Artifacts
  - Source Files
  - Source Code Repository Commits
  - Compiled Binaries
  - Software Packages and Containers
  - Disk Images
  - Firmware Images
- Other Statements
- Non-Software Artifacts
  - Build environments, Physical artifacts, etc.

Statement Consumers

- Statement Consumers
  - Individuals, processes
  - Integrators, suppliers, procurement officers, risk managers, auditors, analysts, end customers
- Activities
  - Locate sources of statements
  - Filter sources to those that are ‘trusted’ by the consumer
  - Filter and download relevant statements
  - Verify statement authenticity and integrity
  - Obtain new, updated statements as they become available
  - Ensure statements are current and have not been invalidated
  - Aggregate statements
  - Use statements to apply policy regarding use of software artifacts

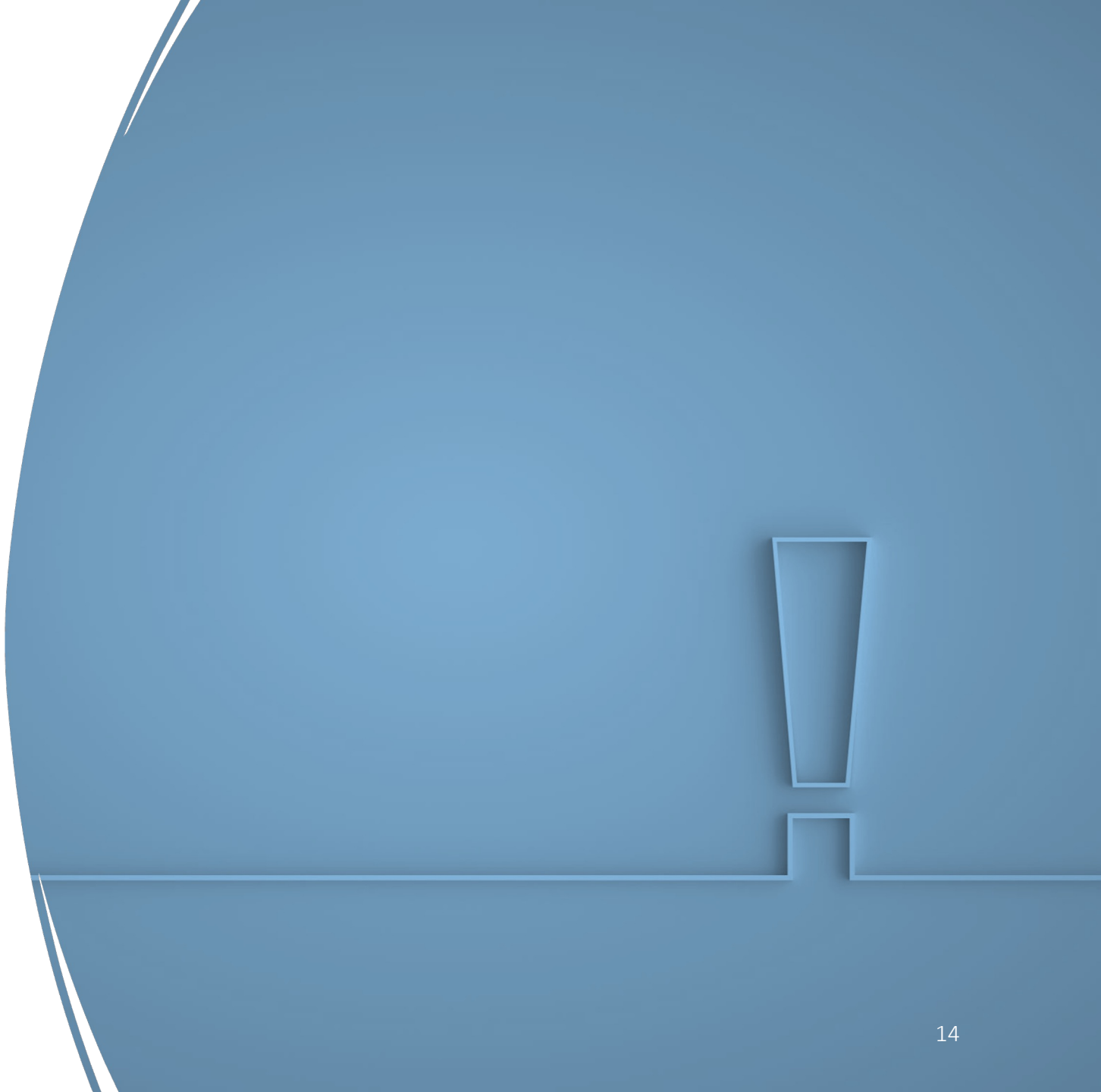
Experience from the Hackathon

Jon Geater

! INFO !

Most of the brain power expended at the hackathon itself was actually spec hacking on the architecture. The fruits of that effort will be revealed and discussed in the next section.

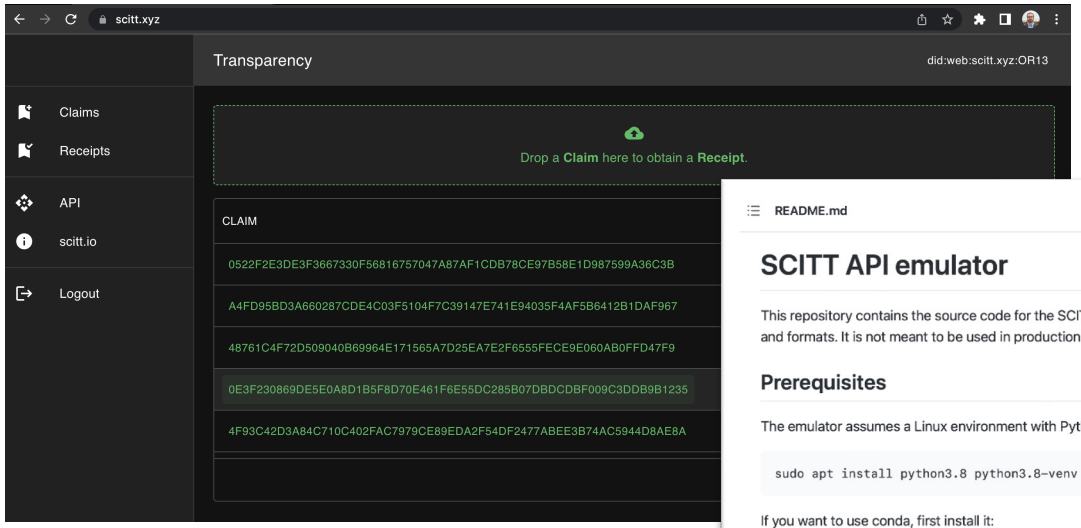
This section is confined to the code exercise.

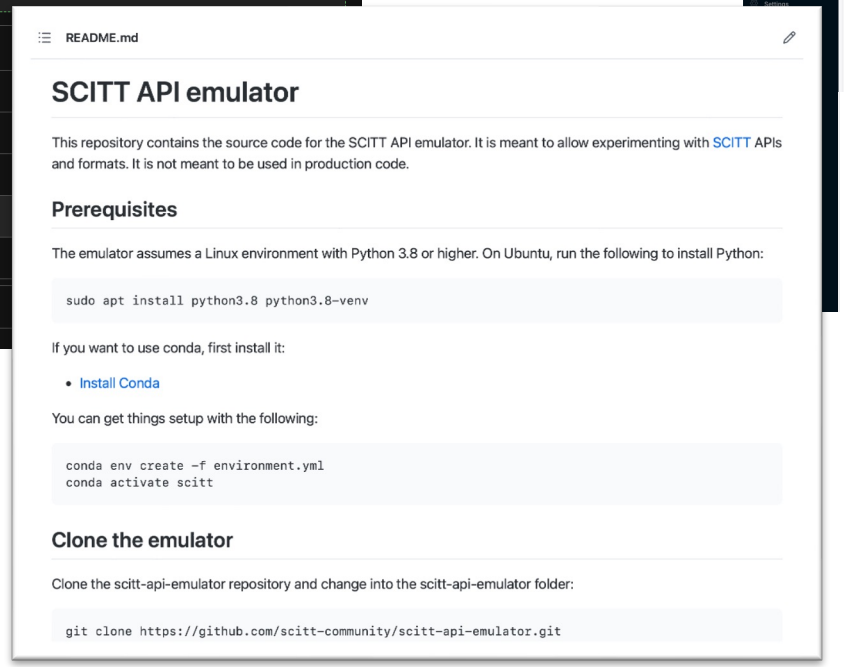


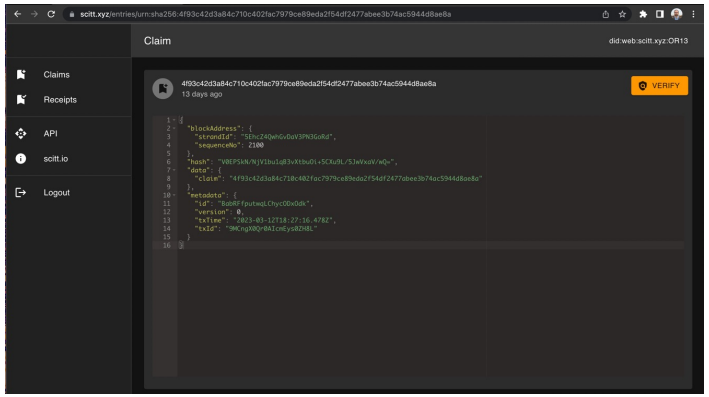
Intent of the code hack

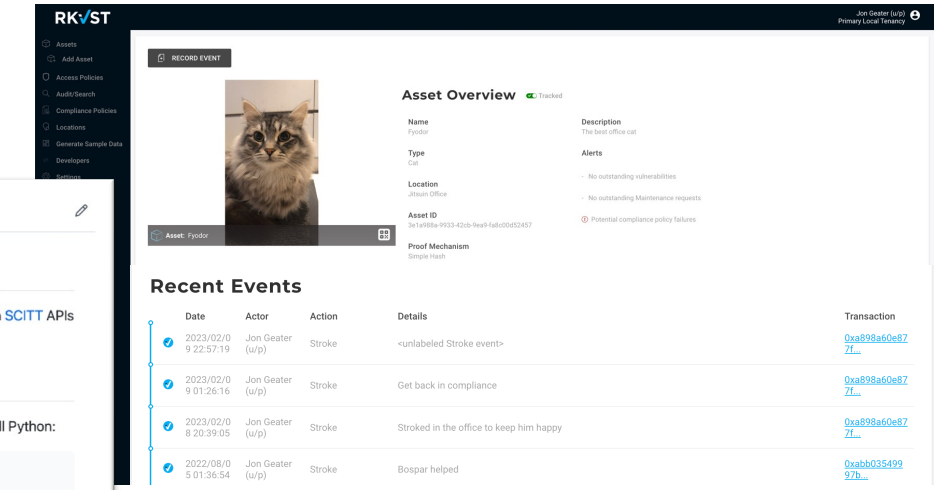
- Multiple implementations proving out the current state of arch – are we on the right track?
  - Testing practicality of interface flows
    - Retrieve receipt à la 8.1.2 and any dependent interfaces
    - Synchronous vs location headers & polling
  - Testing practicality of data structures
    - Claims must conform to 6.1 Envelope and claim format.
    - Register API must take this form and return a COSE receipt
    - Actual content of receipts is out of scope as they're not defined in any adopted work yet. This should be considered a research activity to inform later development.
  - Testing interoperability of interface and data structures
    - BUT NOT PAYLOAD INTEROPERABILITY!

What got done - code







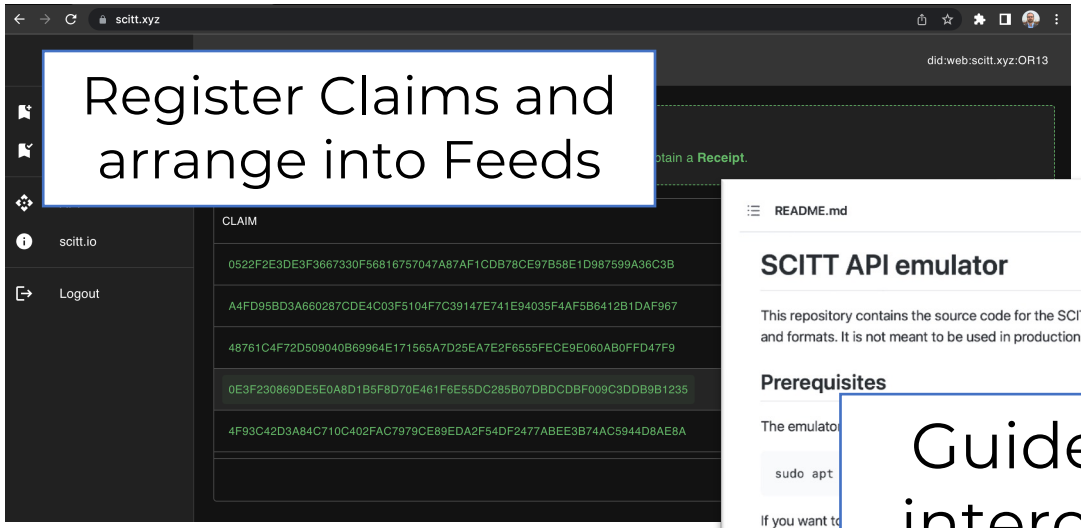


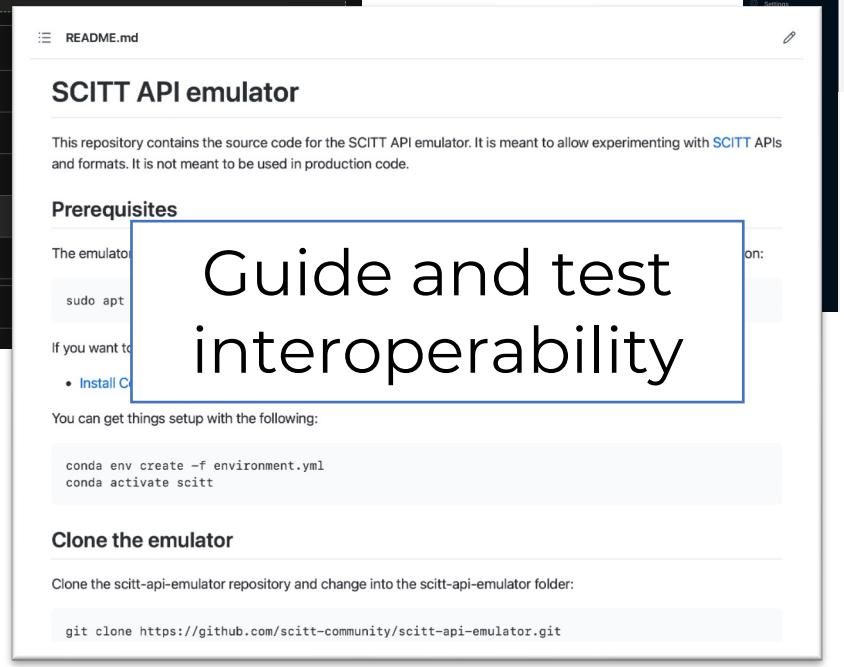
TRANSMUTE

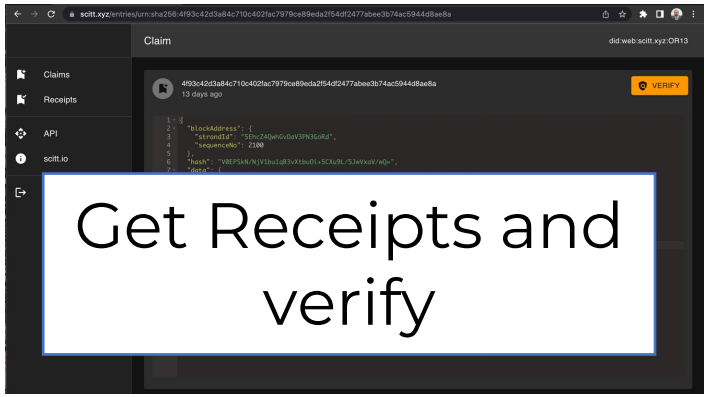
Microsoft

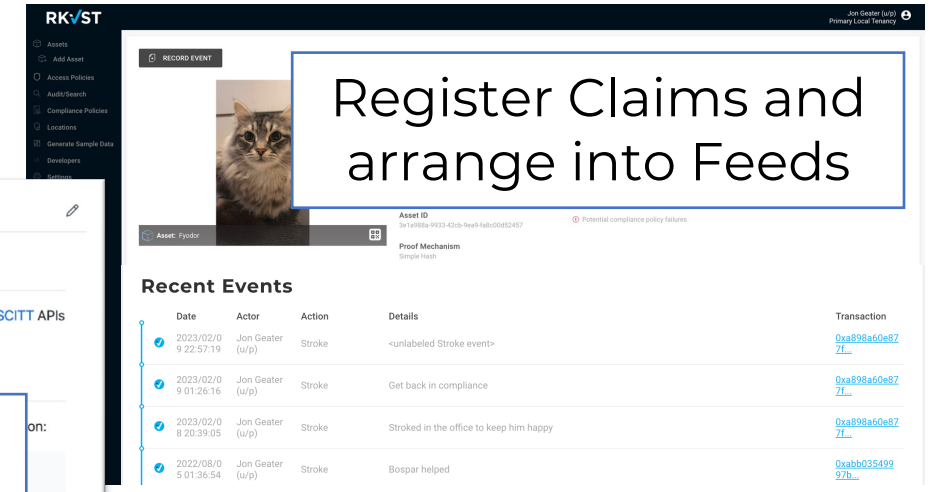
RKVST

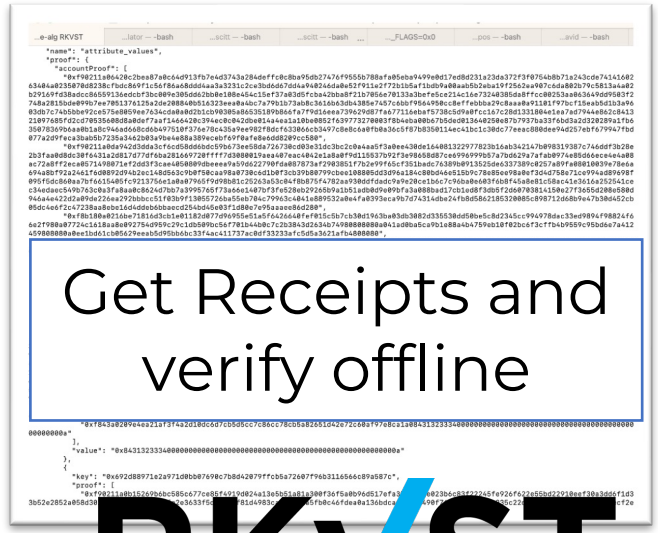
What got done - code







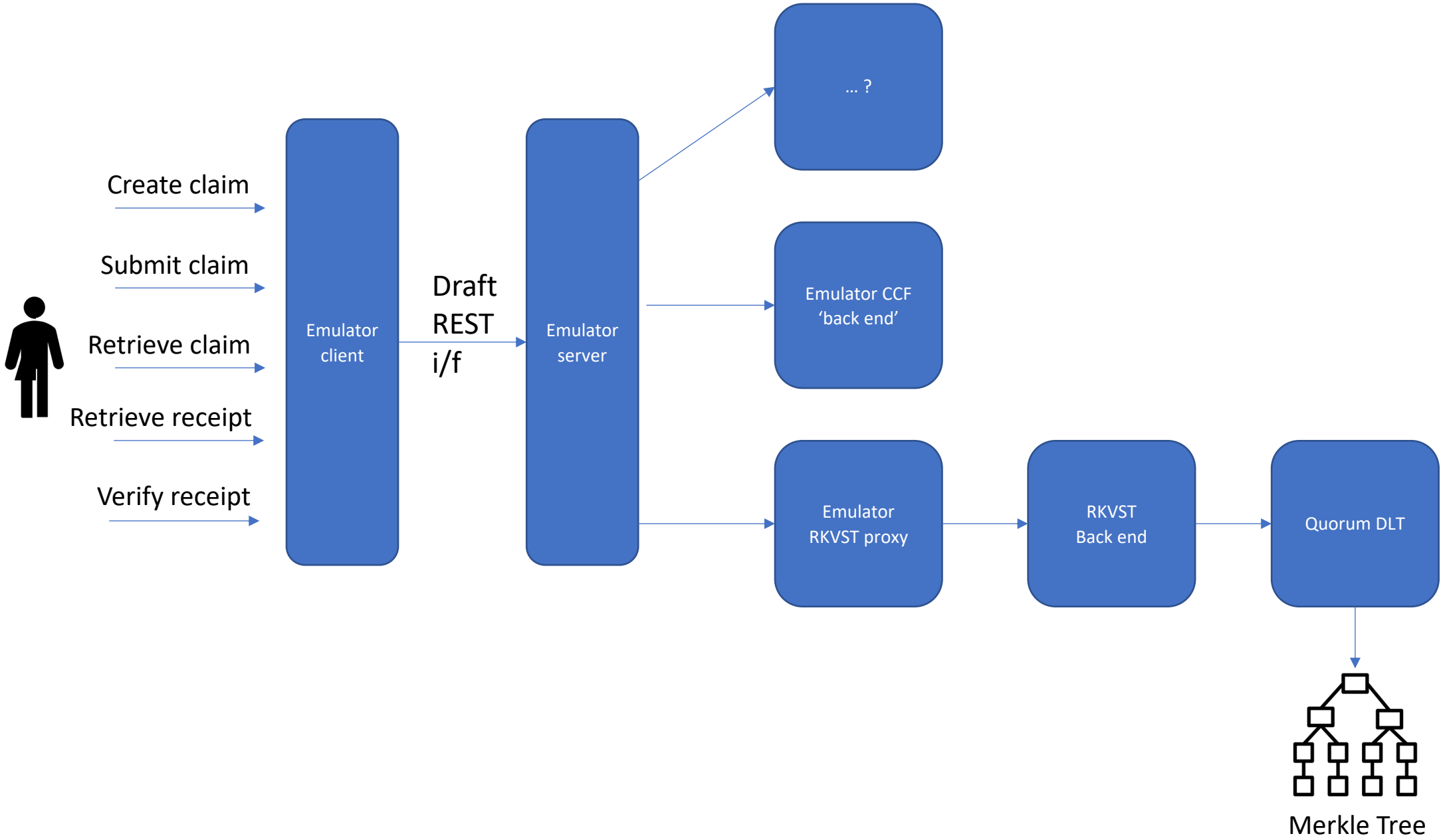


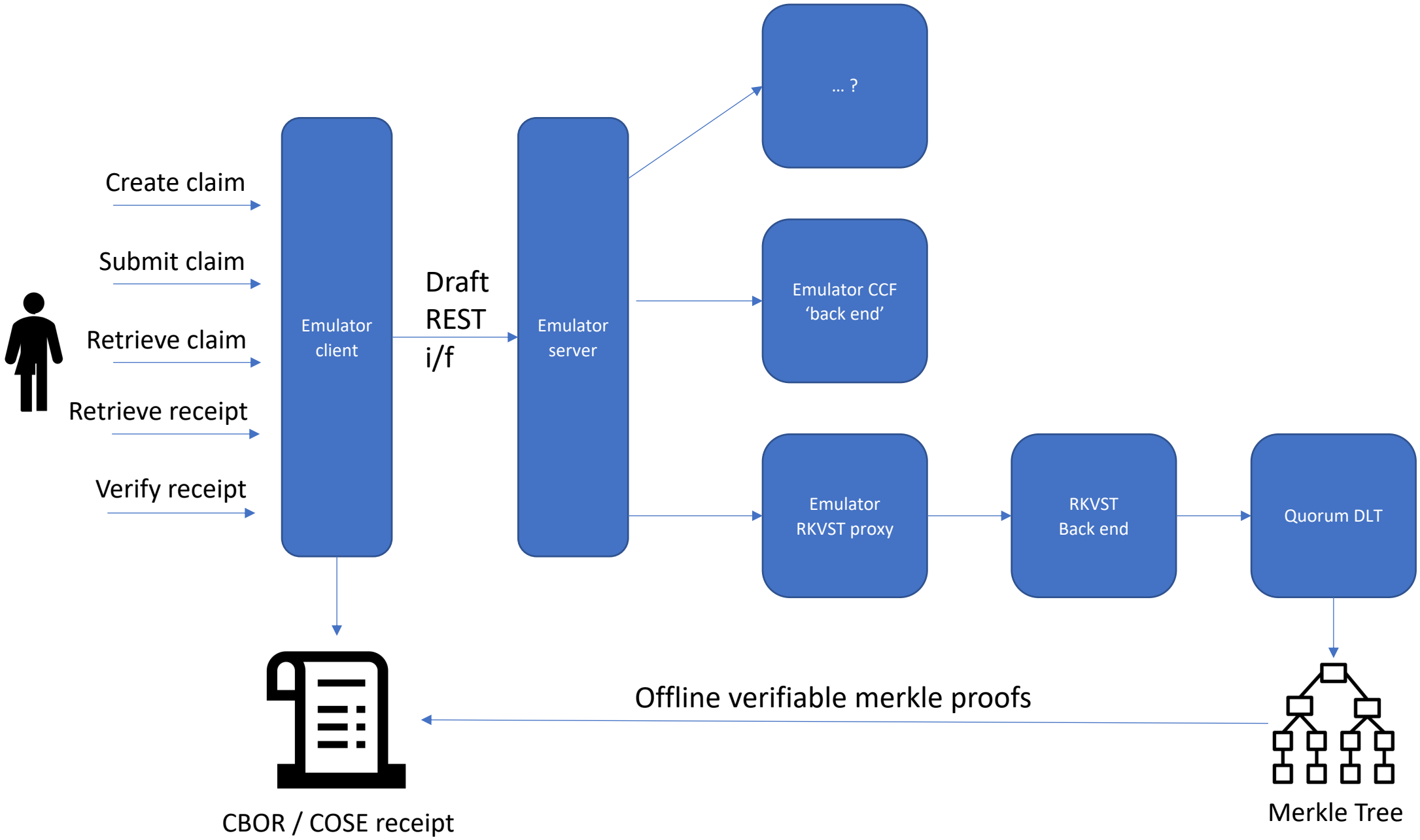


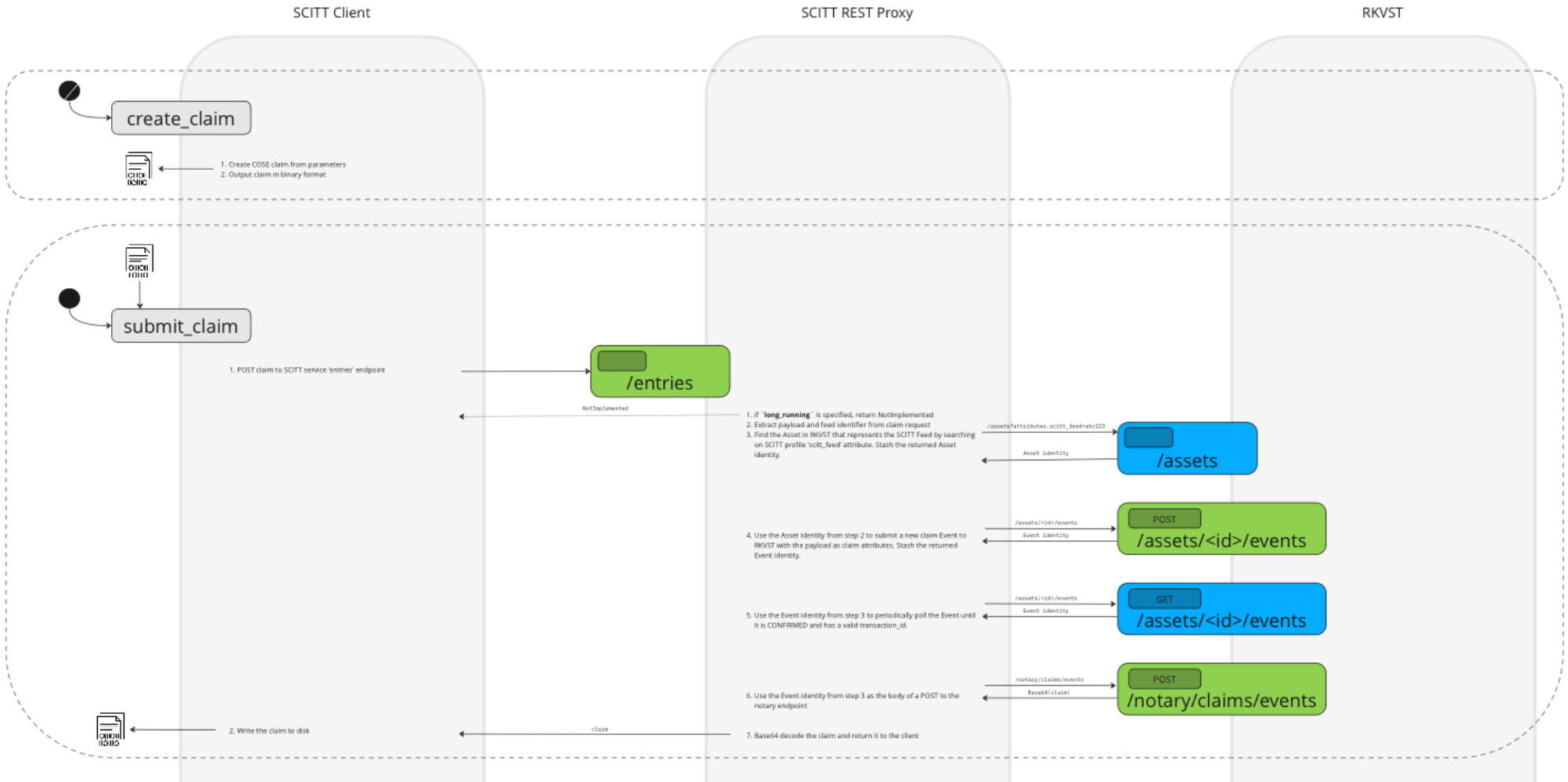
TRANSMUTE

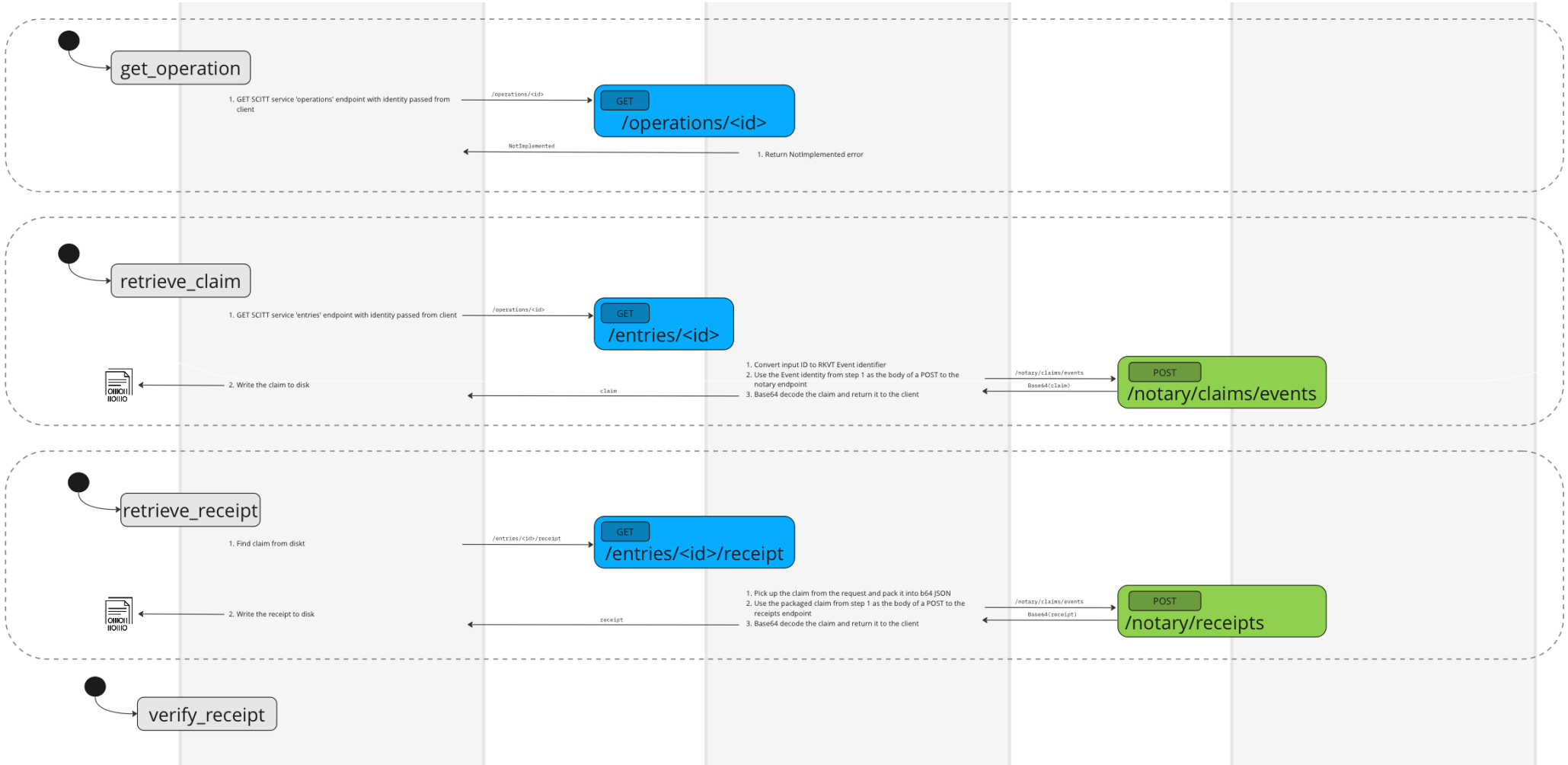
Microsoft

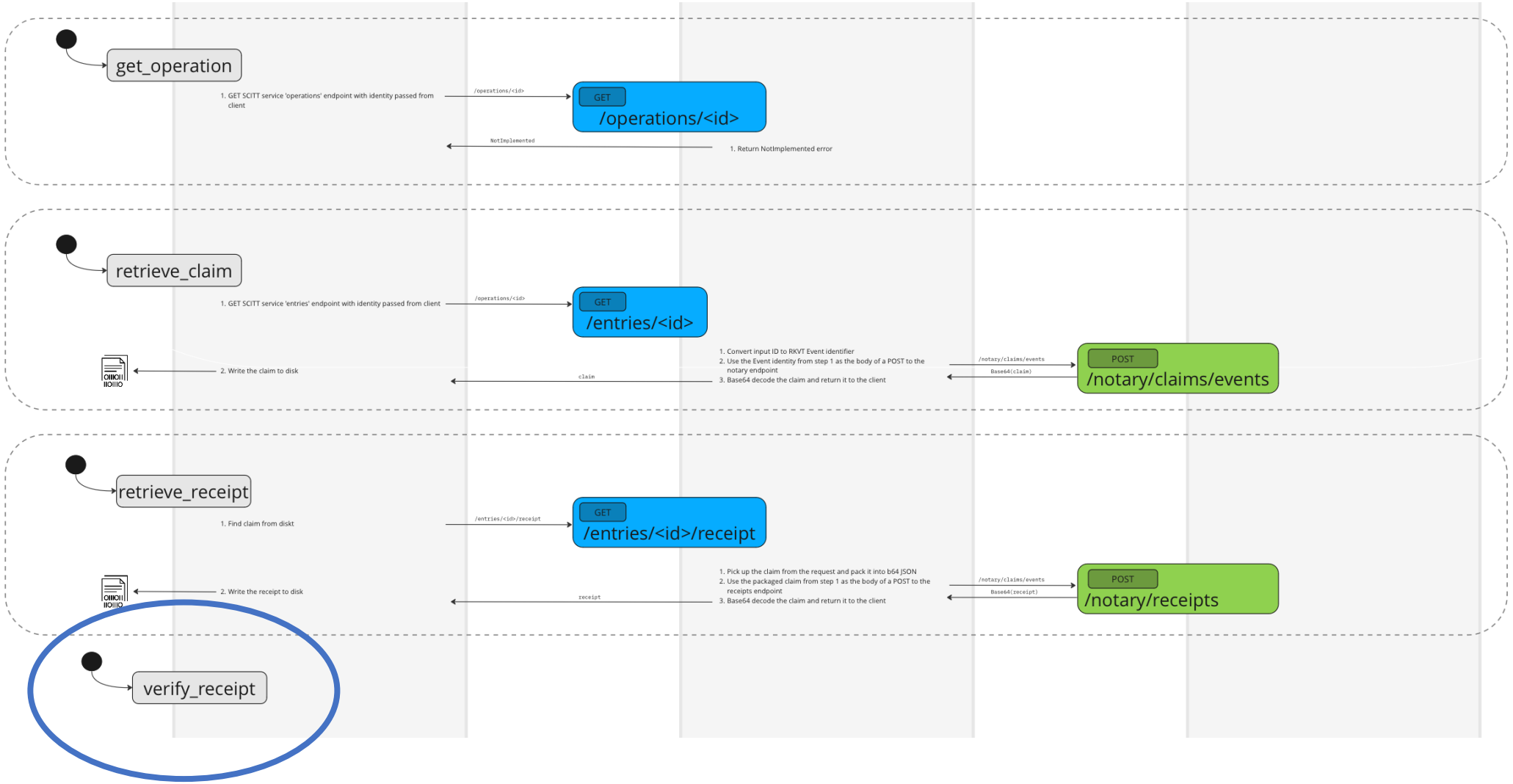
RKVST


Create claim
Submit claim
Retrieve claim
Retrieve receipt
Verify receipt
Emulator client
Draft REST i/f
Emulator server
... ?
Emulator CCF 'back end'
Emulator RKVST proxy
RKVST Back end
Quorum DLT
Merkle Tree


Create claim
Submit claim
Retrieve claim
Retrieve receipt
Verify receipt
Emulator client
Draft REST i/f
Emulator server
... ?
Emulator CCF 'back end'
Emulator RKVST proxy
RKVST Back end
Quorum DLT
Offline verifiable merkle proofs
CBOR / COSE receipt
Merkle Tree


SCITT Client
SCITT REST Proxy
RKVST
create_claim
1. Create COSE claim from parameters
2. Output claim in binary format
submit_claim
1. POST claim to SCITT service 'entries' endpoint
/entries
NotImplemented
1. if "long_running" is specified, return NotImplemented
2. Extract payload and feed identifier from claim request
3. Find the Asset in RKVST that represents the SCITT Feed by searching on SCITT profile 'scitt_feed' attribute. Stash the returned Asset identity.
/assets?attributes.scitt_feed=abc123
Asset identity
/assets
4. Use the Asset identity from step 2 to submit a new claim Event to RKVST with the payload as claim attributes. Stash the returned Event identity.
/assets/<id>/events
Event identity
POST
/assets/<id>/events
5. Use the Event identity from step 3 to periodically poll the Event until it is CONFIRMED and has a valid transaction_id.
/assets/<id>/events
Event identity
GET
/assets/<id>/events
6. Use the Event identity from step 3 as the body of a POST to the notary endpoint
/notary/claims/events
Base64(claim)
POST
/notary/claims/events
2. Write the claim to disk
claim
7. Base64 decode the claim and return it to the client

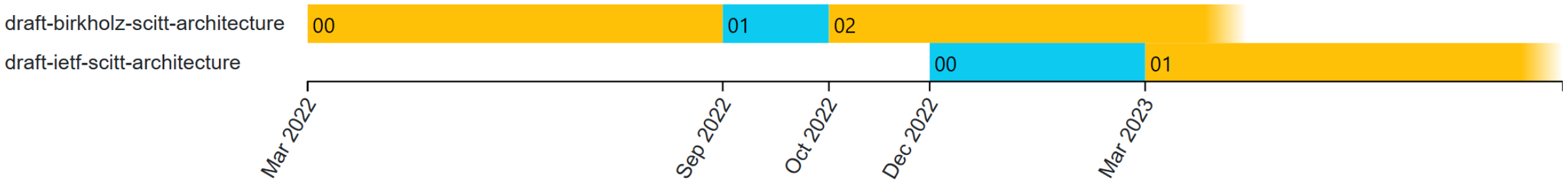

get_operation
1. GET SCITT service 'operations' endpoint with identity passed from client
/operations/<id>
GET
/operations/<id>
NotImplemented
1. Return NotImplemented error
retrieve_claim
1. GET SCITT service 'entries' endpoint with identity passed from client
/operations/<id>
GET
/entries/<id>
2. Write the claim to disk
claim
1. Convert input ID to RKVT Event identifier
2. Use the Event identity from step 1 as the body of a POST to the notary endpoint
3. Base64 decode the claim and return it to the client
/notary/claims/events
Base64(claim)
POST
/notary/claims/events
retrieve_receipt
1. Find claim from diskt
/entries/<id>/receipt
GET
/entries/<id>/receipt
2. Write the receipt to disk
receipt
1. Pick up the claim from the request and pack it into b64 JSON
2. Use the packaged claim from step 1 as the body of a POST to the receipts endpoint
3. Base64 decode the claim and return it to the client
/notary/claims/events
Base64(receipt)
POST
/notary/receipts
verify_receipt


get_operation
1. GET SCITT service 'operations' endpoint with identity passed from client
/operations/<id>
GET
/operations/<id>
NotImplemented
1. Return NotImplemented error
retrieve_claim
1. GET SCITT service 'entries' endpoint with identity passed from client
/operations/<id>
GET
/entries/<id>
2. Write the claim to disk
claim
1. Convert input ID to RKVT Event identifier
2. Use the Event identity from step 1 as the body of a POST to the notary endpoint
3. Base64 decode the claim and return it to the client
/notary/claims/events
Base64(claim)
POST
/notary/claims/events
retrieve_receipt
1. Find claim from diskt
/entries/<id>/receipt
GET
/entries/<id>/receipt
2. Write the receipt to disk
receipt
1. Pick up the claim from the request and pack it into b64 JSON
2. Use the packaged claim from step 1 as the body of a POST to the receipts endpoint
3. Base64 decode the claim and return it to the client
/notary/claims/events
Base64(receipt)
POST
/notary/receipts
verify_receipt

Take-aways: successes and issues raised

- 2.5 implementations proving out the current state of arch
  - Broadly demonstrated practicality of interface flows
    - Statement → Claim → Receipt → Offline Verify
    - Some turbulence in countersigning ideas during code development
      (I think we're better now!)
    - Question marks over how to specify handling of long running operations
    - We will need different verification for different tree algorithms
      (and we really need multiple tree algs: Orie suggested a registry for tree algs in COSE yesterday)
  - Broadly demonstrated practicality of data structures
    - OIDC→ COSE claims → CBOR receipts
    - COSE is great for expressing and serializing the structures, but was a bit awkward for passing across application boundaries. We opted for JSON & base64

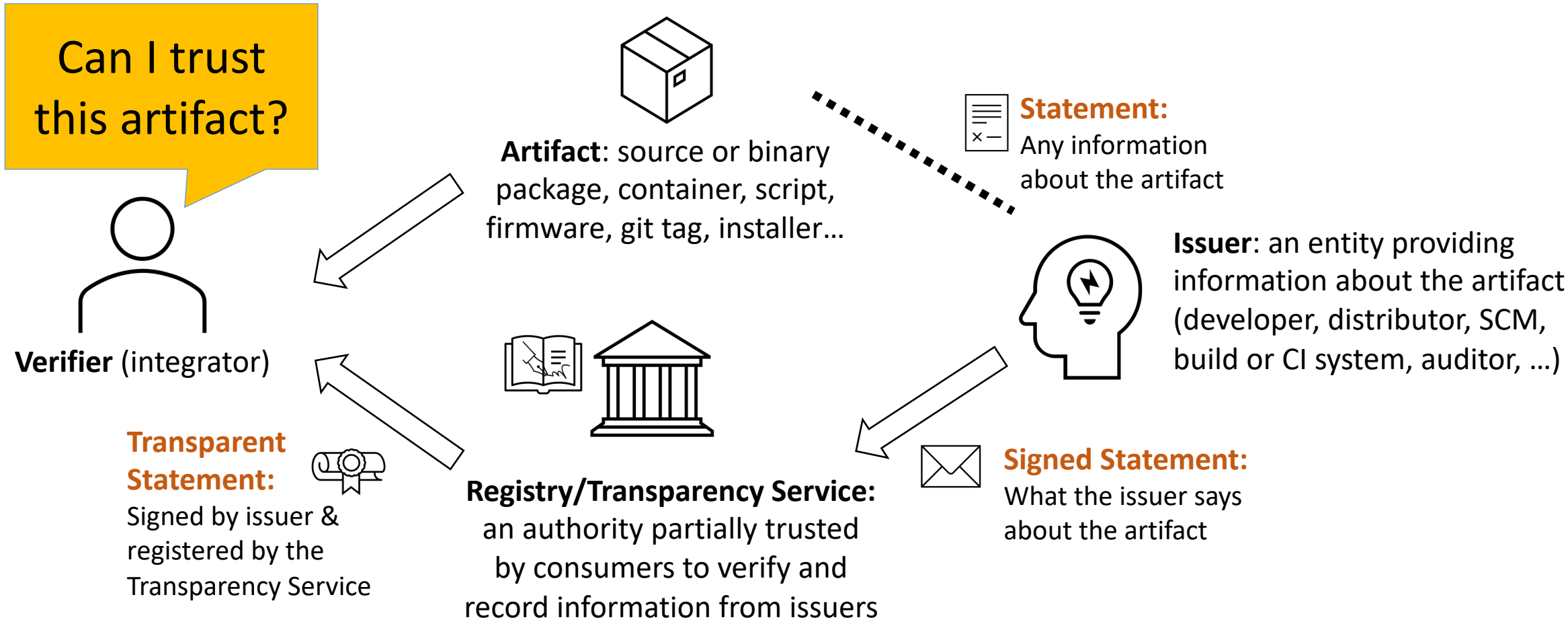
  - Broadly demonstrated interoperability of interface and data structures
    - Single reference client implementation worked with both RKVST and itself
    - RKVST receipts are verifiable entirely offline with 'standard' OSS code – cbor2, pycose, eth_utils
    - Transmute and RKVST implementations of receipt validation structurally very similar



Architecture

Cédric Fournet

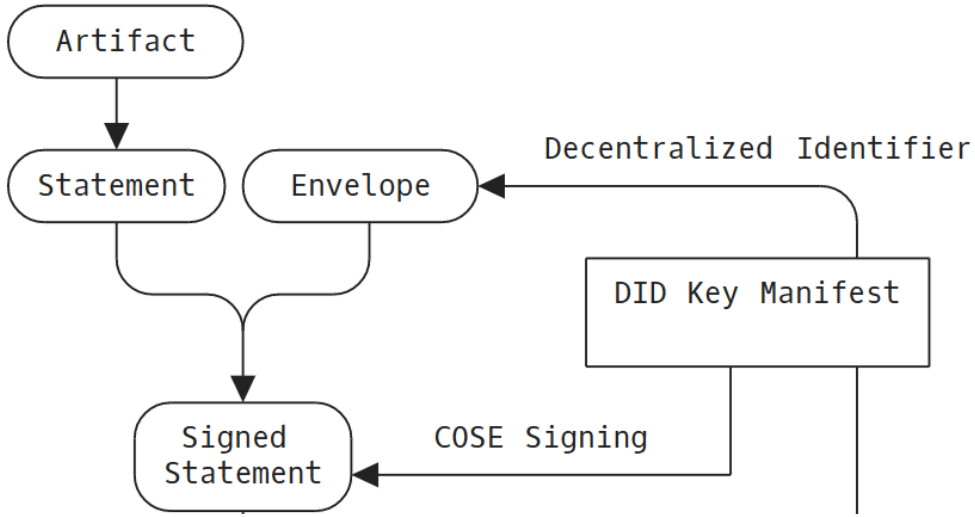
Core Concepts & Revised Terminology



Statement Issuance

Sections 5.1, 6.2

- 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 cryptographic credentials (certificates, signing keys, etc.)
- The Issuers serializes a Statement in a format of their choice (JSON, XML, SPDX, CycloneDX, SLSA, reference to storage…)
- The Issuer produces a COSE signed statement with headers
  - **issuer** is the issuer's DID
  - **feed** is the issuer's identifier for the artifact the statement refers to, e.g. a firmware image
  - **cty** is the format of the serialized statement (specified using mediatype)
  - **registration info** includes input parameters for applying the TS registration policy

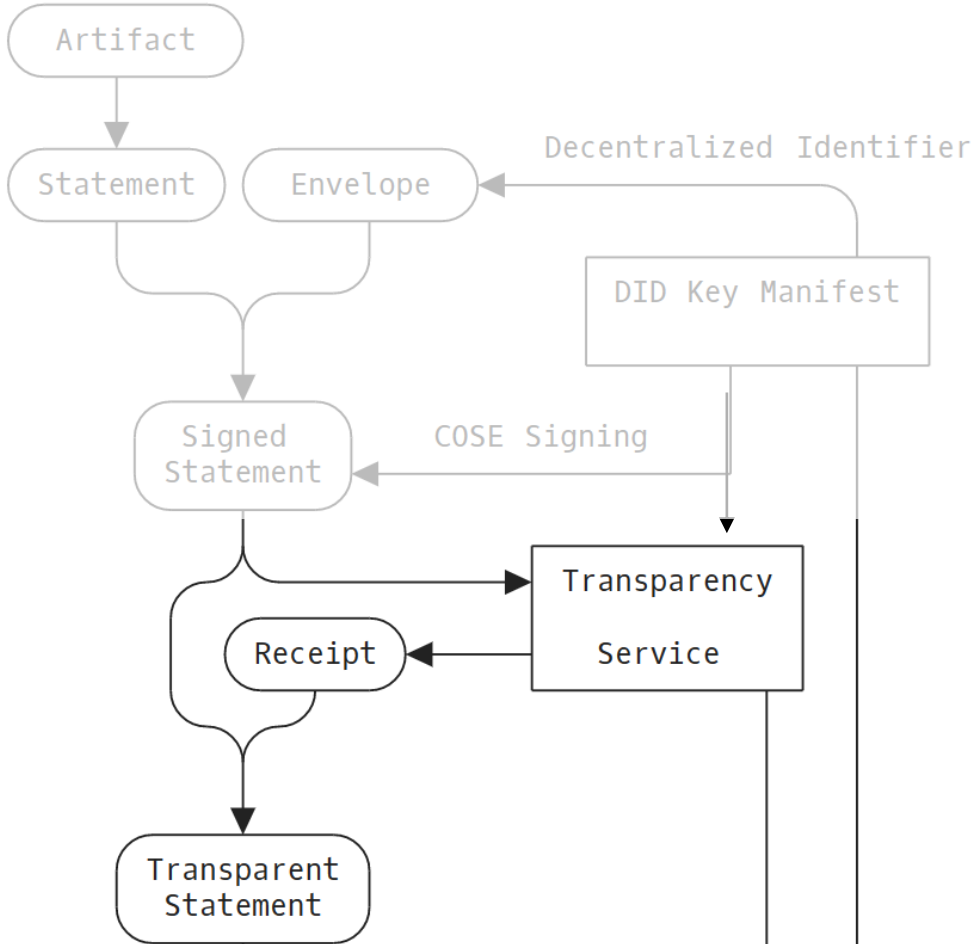


COSE_Sign1

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…	

Statement Registration

Sections 5.2, 6.4



- Some entity submits the **Signed Statement** for registration at the transparency service
- The Transparency Service authenticates the Issuer and checks it against its registration policy to validate the COSE signature
- The Transparency Service may apply additional policy checks including:
  - Restrictions on issuer identity
  - Policies that depend on prior registered claims
  - Policies that depend on the cty, registration info, and payload
- The Transparency Service returns a **Receipt** as proof of registration in its log

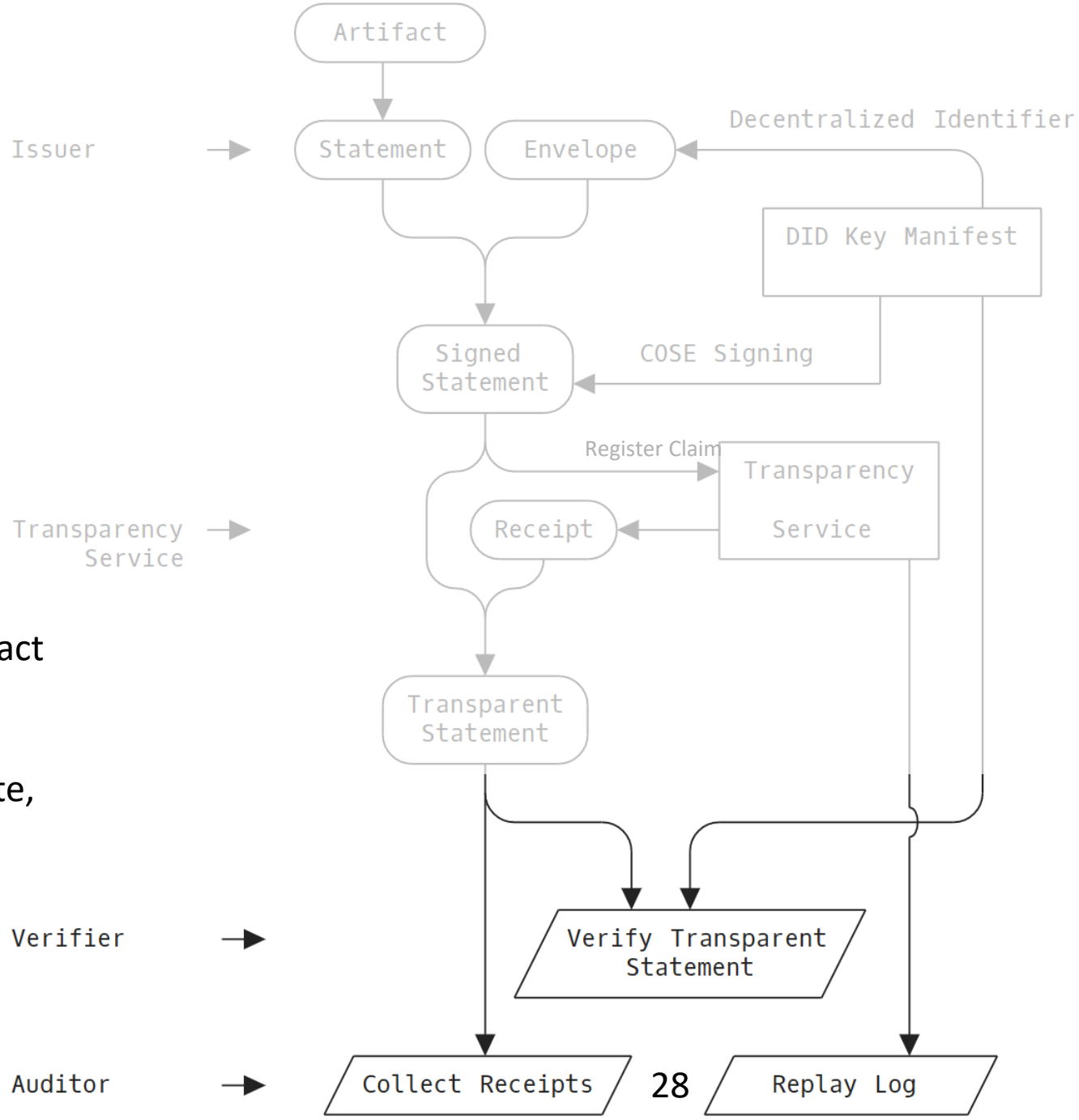
WG work item: which registration policies should be standardized?

Validation & Audit

Sections 5.3, 6.5

- Most Consumers of the Artifact trust a given trusted Transparency Service, and check that they get a valid Transparent Statement associated with the Artifact by verifying its **Receipt**
- Some Consumers may additionally check details of the Statement, re-verify the Issuer's signature, and apply additional policies before accepting the artifact
- The most suspicious consumers (**auditors**) do not fully trust the transparency service. They may keep state, fully re-play the registration of some/all claims, and examine collected receipts from other verifiers

WG work item: do we need additional support for auditors and verifiers to query the registry?



Scalability?

Issuance is fully-distributed

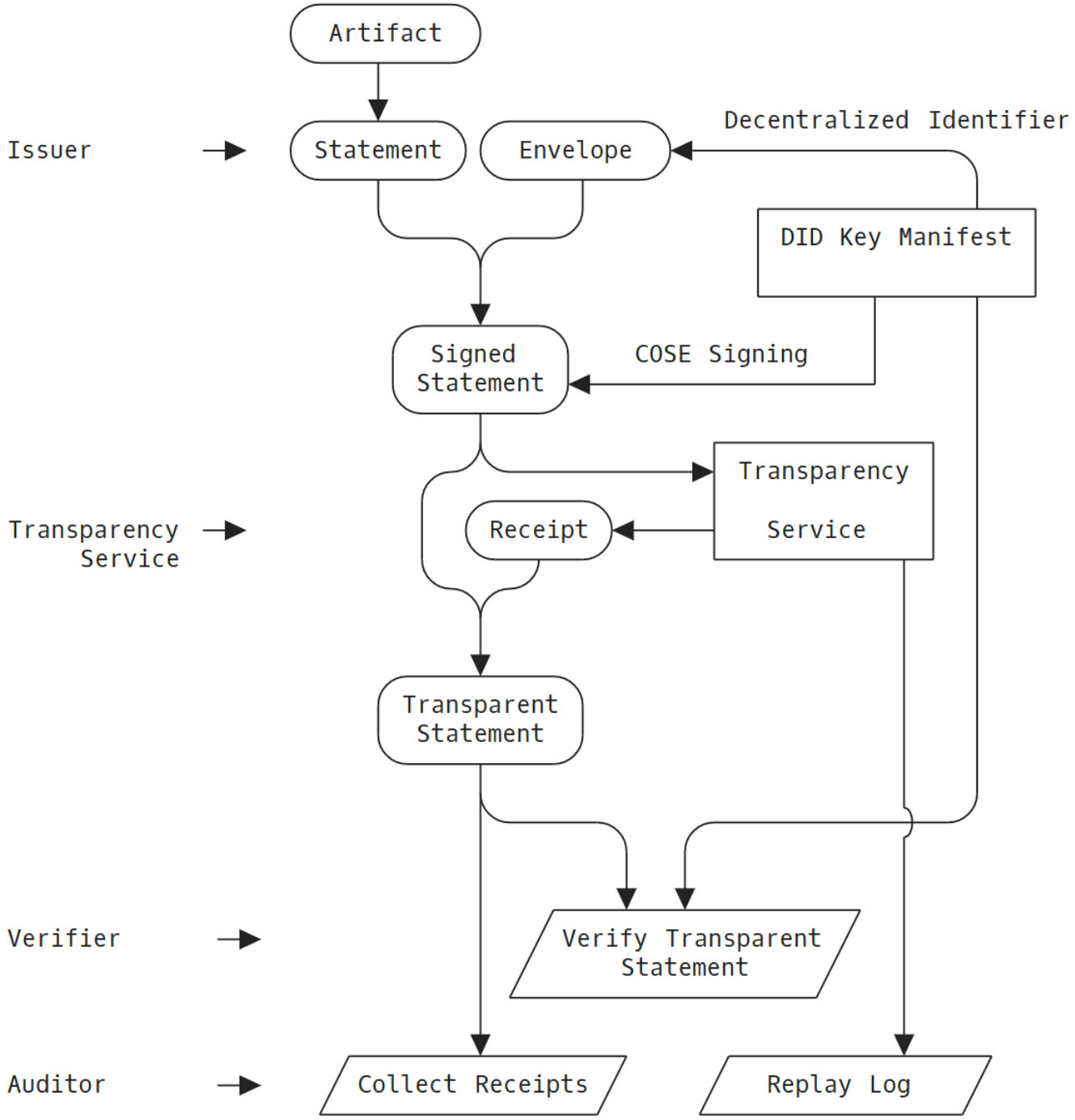
Registration scales by

- batching: Signed Statements can be registered in bulk, by extending the Merkle tree & signing its root once per batch (10k+ Statements/S throughput with mS latency)
- keeping Transparent Statements small, e.g., by detaching payloads & including only commitments to larger documents such as SBOMs.
- federation: not everyone needs to keep track of intermediate Transparent Statements

Verification scales by relying on receipts

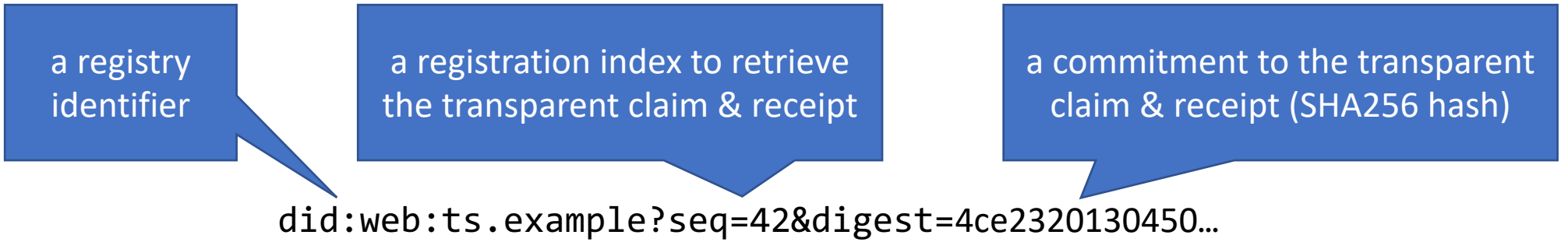
- Relying parties can verify transparent claims offline
- Untrusted stores can replicate the log and serve Receipts without contacting the Transparency Service

WG work item: ensuring freshness without contacting the TS



URLs for Transparent Statements

References are composed of 3 parts:



Use cases:

- Registration info referring to previously-registered statements, e.g.,
  “this statement updates this earlier statement” or
  “this statement is registered based on this policy statement”
- Federation (secondary Transparency Service pointing to primary Transparency Service)

Passing Statements by reference enables their transparent sharing
(Passing them by value opens potential inconsistency attacks)

Recording Registration Data

Registration may produce data that is relevant for verifiers, such as

- a timestamp
- a reference to a transparent statement that defines the applied registration policy
- a reference to a transparent statement that records the registry configuration
- a reference to the last transparent claim with same issuer & feed

…

Can we include such data in the resulting transparent statement?

fresh WG work item

Recording Registration Data

Transparent Claim

issuance

Protected Header	Value
iss	**did**:web:firmware.sec.fpga.com
alg	ES384
kid	#key-3
feed	C910 FPGA Firmware
cty	application/…
registration_info	timestamp, version number, …
payload: serialized Statement	
Issuer signature 3045022100e7d0…	

registration

Unprotected Header	Value
registration_data	policy applied, statement replaced, …
receipt	

authenticated by the receipt

Receipt

Protected Header	Value
iss	did:web:transparency.example
alg	ES384
kid	#key-0
tree_alg	CCF | QLDB | Trillion | Tessera

Unprotected Header	Value
inclusion_path	[extra data, [+ hashes]
payload: Merkle Tree root	
Transparency Service signature 150rbd5a502100e7fe…	

COSE Inclusion Proofs

Orie Steele

What is a SCITT Receipt?

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

- Registration means:
  - Apply registration policies (at minimum, verify issuer/identity)
  - Store the signed statement in the append only log
  - Compute a merkle proof for the signed statement
  - Sign the merkle root and include the proof to produce a receipt
  - Return the receipt to the submitter

How are SCITT Receipts used?

- Verifiers need only trust the issuer of the receipt, not the issuers of all signed statement
- Receipts enables offline verification of the issuer's signed statements
- Receipts are also used in federation...

Why Are Receipts Like Countersignatures?

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

- On a technical level, treating it ***like*** a countersignature + metadata:
  - Allows embedding of receipts in the *unprotected* header of the signed statement
  - Leveraging detached envelope payload forces verifiers to recompute the merkle root

SCITT Receipts are COSE Merkle Proofs

- https://github.com/ietf-scitt/draft-steele-cose-merkle-tree-proofs
- Generic COSE Merkle Proofs can be used for SCITT Receipts

SCITT Receipt

Protected Header	Value
iss	did:web:transparency.example
kid	#key-0
alg	ES256
tree_alg	CCF | QLDB | Trillion | Tessera

Unprotected Header	Value
inclusion_path	[extra data, [+ hashes]

Payload: Merkle Root
Signature 3045022100e7d0…

WG work item: Structure of receipts will hopefully be a COSE WG item, and no longer specific to SCITT.

AOB (Open Mic) & Next Steps

Wrap-Up

Back up

Generalized Use Case

- Scenario: Software consumers want to consume products that meet their requirements.
- Problems Today:
- Software producers need to provide assurance that products meet customer requirements.

- No standard methods for software producers to create and share assurance statements with customers.

Use Case Coverage

- Software types (software, firmware)
- Lifecycle steps (code, commit, package, release, deploy)
- Attack types (signing certificate compromise, vulnerability exploit)
- Distribution scenarios (multiple suppliers, integrators)
- Assessment scenarios (security and compliance audits, security analysis)
- Deployment scenarios (firmware, IoT, air-gapped environments)

Use Case Example: Auditing of Software Product

- Scenario: An organization has established procurement requirements and compliance policies for software use.
- Problems Today:
- Difficult to gather track and manage associate relevant documents and check results required for various types of audits
- assert the authenticity and provenance of documents relevant to audits in a deterministic and uniform fashion
- check the validity of identity statements about relevant documents after the fact (when they were made) in a consistent, long-term fashion
- allow for more than one level of complexity of audit procedures (potentially depending on criticality)

Verification that Signing Certificate is Authorized by Supplier

- Scenario: A malicious actor compromises a supplier, obtains a signing certificate from the supplier, and uses it to sign compromised software. The end user installs the compromised software believing it to be from the supplier.
- Problem Today: No way for suppliers to identify Software consumer wants to verify the authenticity and integrity of software before use
- Standards are needed to:
  - allow verification that certificates used to sign software are authorized by the supplier for signing and are still valid

Multi Stakeholder Evaluation of Released Software

- Scenario: Individuals and organizations want to ensure the software they produce and consume meets best practices (including business and regulatory requirements) for security and privacy.
- Problems: Evaluating whether requirements have been met requires tracking of activities, processes, and evaluations both as the software is produced and following
- Standards are needed to
  - Aggregate related assessments across multiple parties
  - Express relationships between assessments
  - Identify and discover relevant assessment providers

Security Analysis of a Software Product

- Scenario: A critical security issue is identified in a software component. Individuals and organizations want to know if they are exposed, and if so, what they can they do to reduce personal and business risk.
- Problems: Statements of exploitability and mitigation need to be exchanged by many parties (component providers, integrators, deployment administrators) before final assessments can be made to end users.
- Standards are needed to:
  - Facilitate the timely exchange of statements of exploitability and mitigation
  - Express the provenance and history of statements
  - Express the relationship between statements
  - Verify that statements come from authoritative sources

Use Case Coverage

- Software types (software, firmware)
- Lifecycle steps (code, commit, package, release, deploy)
- Attack types (signing certificate compromise, vulnerability exploit)
- Distribution scenarios (multiple suppliers, integrators)
- Assessment scenarios (security and compliance audits, security analysis)
- Deployment scenarios (firmware, IoT, air-gapped environments)