Attestation Verifier Theory of Operation

Ned Smith

Terminology

- Claimset* – Typically, consists of an EMT – Environment-measurement-tuple – that names an environment to which certain measurements belong
- Authority – the entity(ies) that asserted a Claimset – typically a cryptographic key / key-id.
- Accepted Claims Set (ACS) – a Claimset that describe a particular Attester
- Condition – a Claimset that is compared with ACS
- Augmentation – a process of extending the ACS through condition matching
- Endorsement – a Claimset that augments the ACS
- Validation Function (VF) – A function that is applied to a Claimset
- View – A Claimset that is a subset of the ACS
- Reference Value (RV), Reference Value Provider (RVP) – See RFC9334

*For consistency, a Claimset can be a set of Claimset

Theory of Operation: Goals

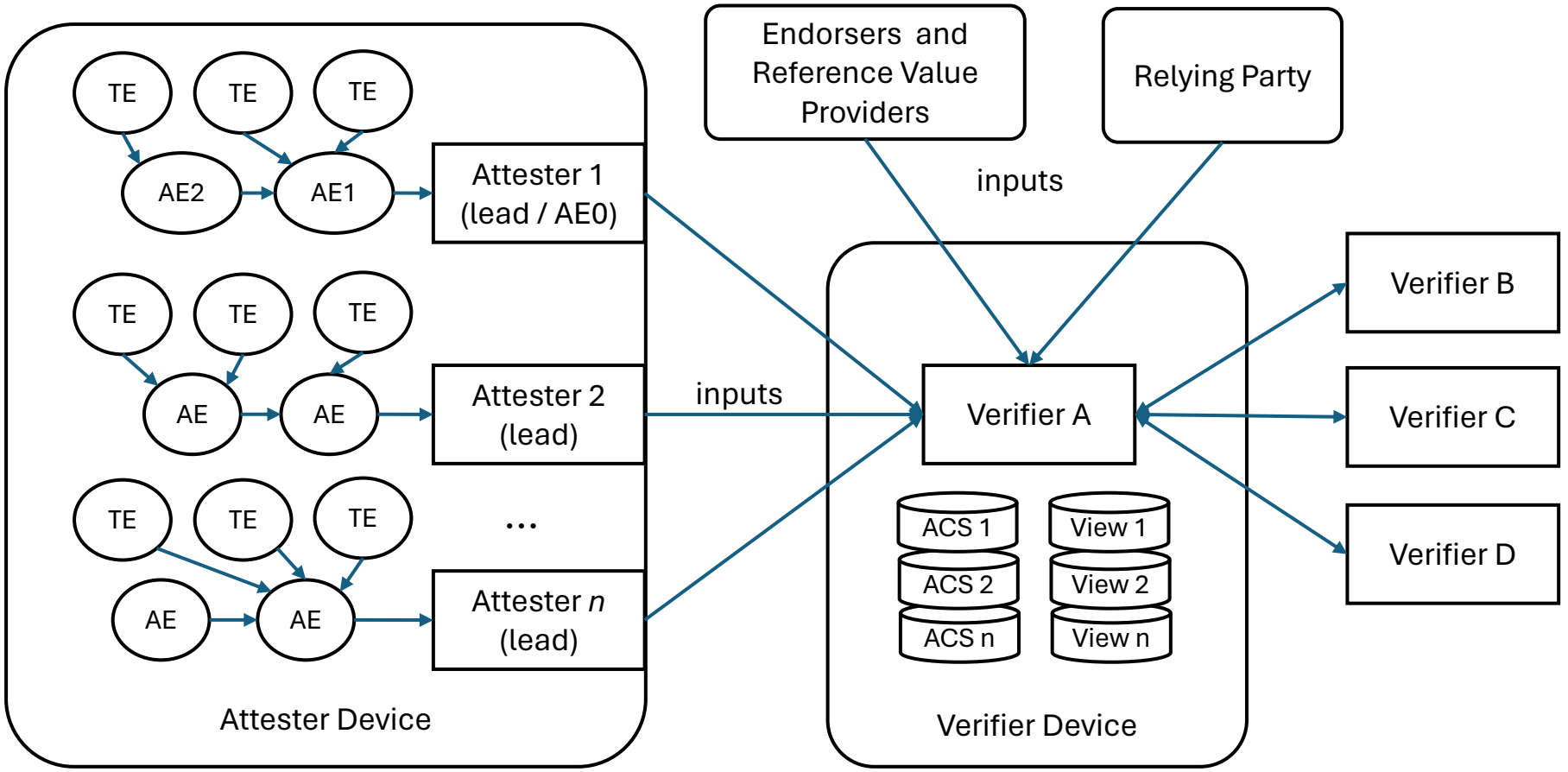
- Basic operational goals
  - Construct an ACS for a given Attester that is non-lossy
  - Inputs are constrained by RATS roles
    - (Attester, RVP, Endorser, V-Owner, RP*)
  - All Verifiers will produce the same ACS given the same inputs
  - Inputs can occur in any order and can be spaced in time for multiple Verifiers
  - Multiple Verifiers can cooperate to construct a common ACS
    - Partial ACS(es) are another type of input
  - Verifiers can augment the ACS by following the same operational rules available to other RATS roles and inputs.
  - Verifiers may constrain the ACS by presenting a read-only View of the ACS

*RATS Arch doesn't name RP as an input, but TCG Attestation Framework does.

Verifier Assumptions

- RFC9334 defined assumptions
  - Multiple “Attester” roles can exist on the same attesting device, each lead Attester has an independent session with the Verifier.
    - Attesting Environments rely on a lead Attester to forward Evidence to the Verifier
    - A lead Attester might re-publish Evidence, in which case it becomes another Attesting Environment (i.e., AE0)
  - Multiple ecosystem entities can supply the same Endorsements and Reference Values but have different authority (keys).
  - There is one Verifier role, but the role can be distributed across multiple entities.
  - Appraisal policies from Verifier Owner can constrain Attestation Results.
  - Appraisal policies from Verifier Owner can limit the set of Endorsers, Reference Value Providers, and Attesters that can provide Verifier inputs.
- Additional assumptions
  - Relying Parties can supply inputs to the Verifier that constrain Attestation Results
  - The Verifier normally can’t tell if multiple independent lead Attesters are on the same device or have separate devices
    - However, a Verifier could in theory discover that an attester device has multiple lead Attesters
      - E.g., Endorsements could link lead Attesters.
      - E.g., An ontology could link components that belong to a common device.

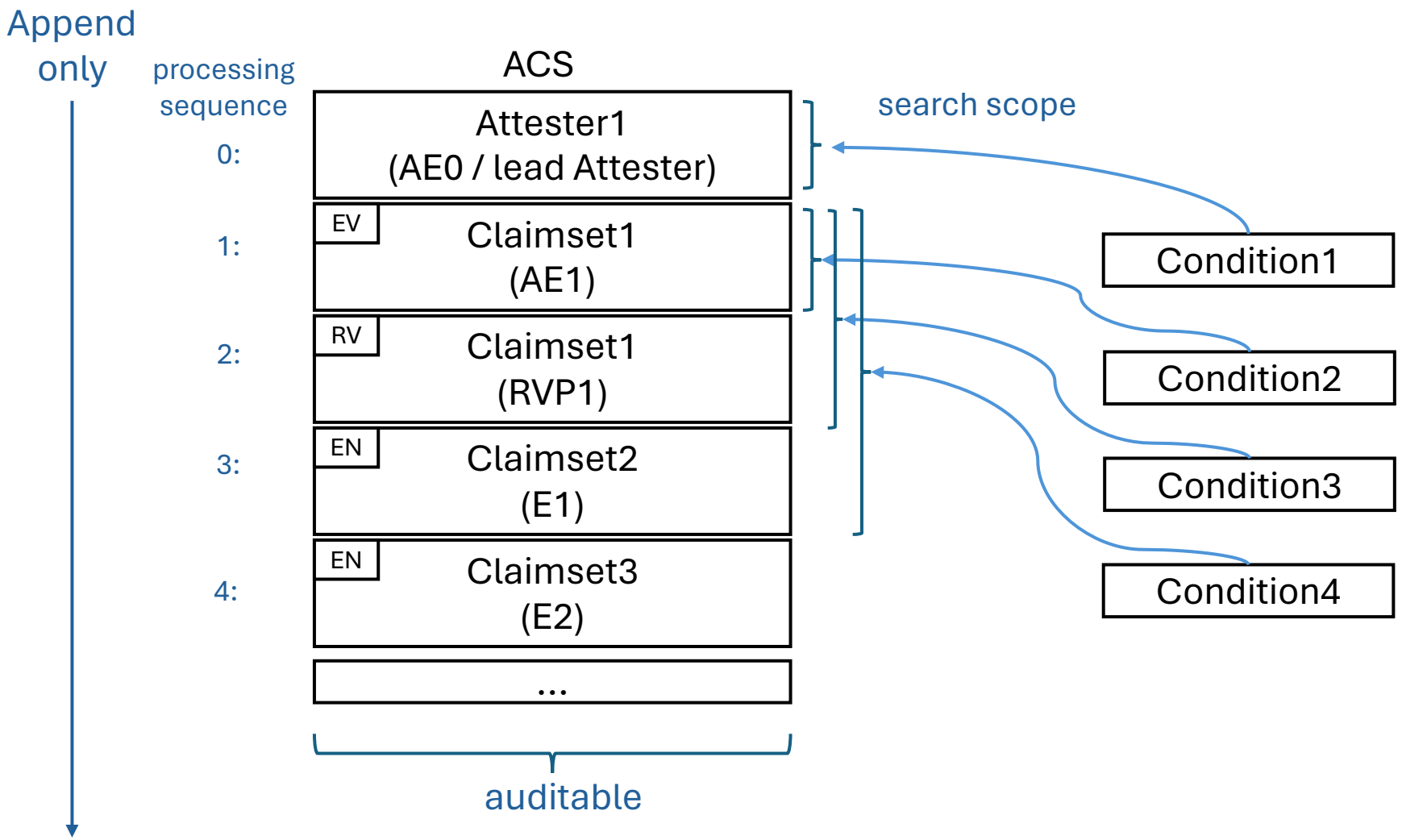
Example Deployment



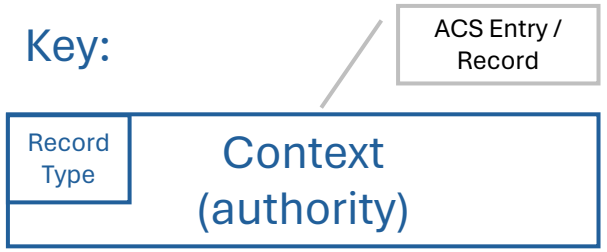
Theory of Operation

- Start by initializing the ACS as the empty set
- Add an Attester binding that associates a lead Attester instance with an ACS.
  - Verifier authenticates the lead Attester which forms a lead Attester–ACS binding (a.k.a., session).
- Basic Augmentation – The ACS state changes by processing input tuples:
  - Augmentation tuple – an abstract tuple that represents most of the current set of CoMID triples:
    - (<condition>, <update>,<authority>) => <output-set>
      - <condition> - a Claimset that is matched to the ACS
      - <update> - the Claimset to be added to the ACS if <condition> is true (note: constrained by schema expressiveness)
      - <authority> - the credential/key of the entity that asserted the tuple
      - <output-set> - the Claimset for a record that is added to the ACS
        - The record includes <authority> and record type
    - If <condition> is true, then copy <update> with entity's <authority> to <output-set> and append it to the ACS
  - Processing begins when new Evidence, RVs, or Endorsements are added to an input queue
    - Each type of input follows the Augmentation tuple structure. E.g.:
      - Evidence: (<empty-set>, <any claimset in the Evidence domain>, <Attester binding context>) => <evidence-set>
      - Reference Values: (<evidence-set>, <any claimset in the Reference Values domain>, <any RVP trust anchor>) => <rvp-set>
      - Endorsement: (<evidence-set or rvp-set or previous endorsement-set>, < any claimset in the Endorsement domain>, <any Endorser trust anchor>) => <endorsement-set>
    - Inputs are processed when it is received (in an append-only fashion – more later)
  - If a <condition> isn't met, then the tuple is returned to an input queue where the <condition> is re-tried
  - Processing terminates when the Attester session closes
    - Tuples with unmet conditions are discarded
    - The ACS can be archived for audit/compliance purposes

Condition Processing



Key:



Record Type

- EV : Evidence
- RV : Reference Values
- EN : Endorsements

(authority)

- AE1 : Attesting Environment 1
- RVP1 : Reference Value Provider 1
- E1 : Endorser 1
- E2 : Endorser 2

More on Append-only Semantics

- ACS records are marked with the conceptual message type
  - This removes the need to protect processing ordering to achieve deterministic results.
  - Record order is an artifact of (unpredictable) workflow processing dynamics.
  - Since records can appear in any order, records can be added following append-only semantics which reduces update contention and facilitates distributed verifier architectures.
- Conceptual messages have scoped search semantics
  - E.g., Reference Values are scoped to Evidence
  - Conditions rely on message type context to scope search targets

Basic ACS Augmentation Examples

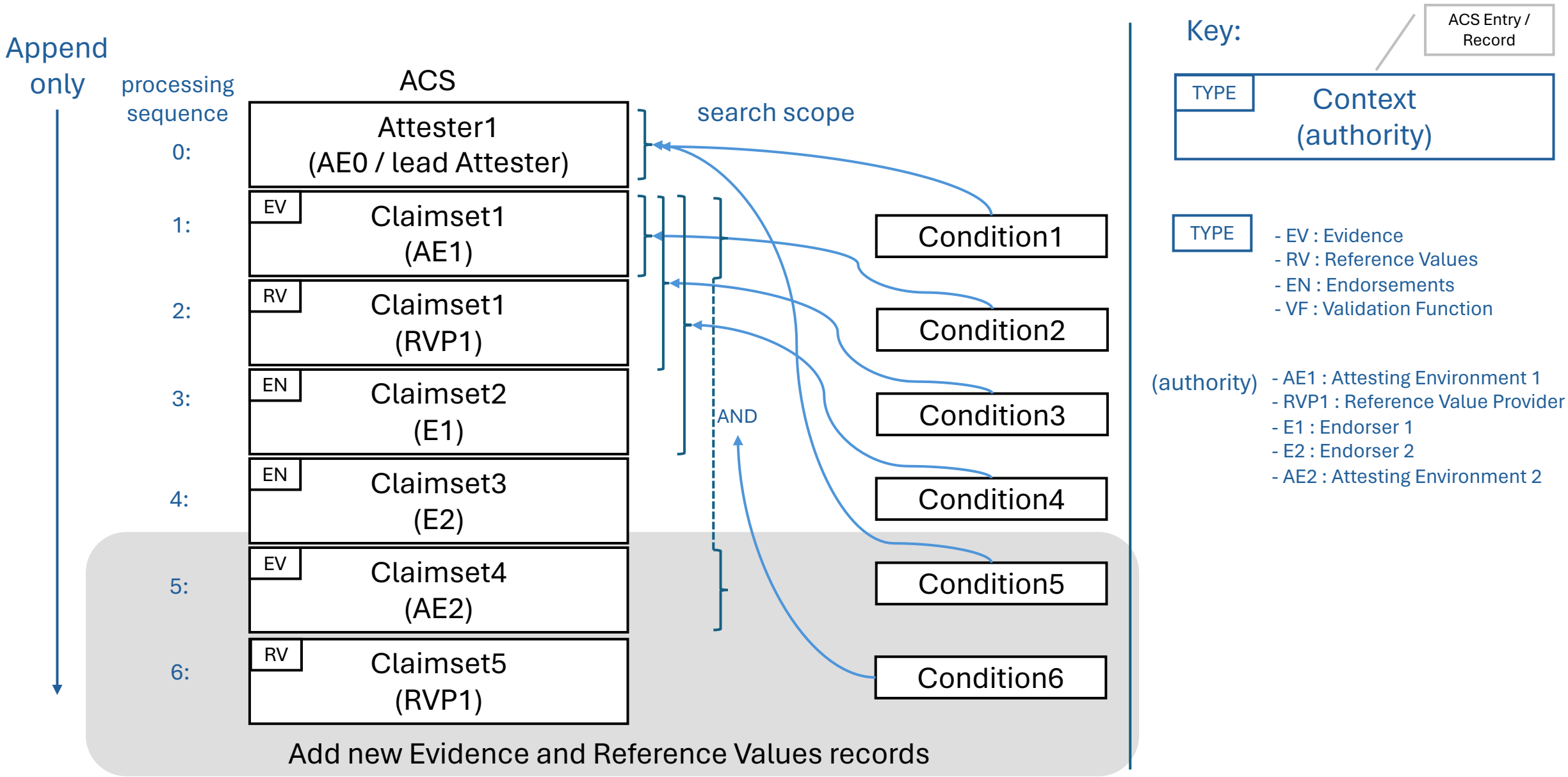
- Evidence
  - (<>, <class-id=.3.2.1 : digest=h’FED4’>, <key-id=h’01’>) =>
    - <[class-id=.3.2.1 : digest=h’FED4’], key-id=h’01’>
- RVP
  - (<class-id=.3.2.1 : digest=h’FED4’>, <>, <key-id=h’02’>) =>
    - <[class-id=.3.2.1 : digest=h’FED4’], key-id=h’02’>
- Endorsement #1
  - (<class-id=.3.2.1 : digest=h’FED4’>, <class-id=.3.2.1 : svn=7>, <key-id=h’03’>) =>
    - <class-id=.3.2.1 : svn=7>, key-id=h’03’>
- Endorsement #2
  - (<class-id=.3.2.1 : svn=7, key-id=h’03’>, <class-id=.3.2.2 : version=“1.0”>, <key-id=h’04’>) =>
    - <class-id=.3.2.2 : version=“1.0”, key-id=h’04’>

Resulting ACS

processing sequence

1) [[class-id=.3.2.1 : digest=h’FED4’], key-id=h’01’, evs],
2) [[class-id=.3.2.1 : digest=h’FED4’], key-id=h’02’, rvs],
3) [[class-id=.3.2.1 : svn=7], key-id=h’03’, ens],
4) [[class-id=.3.2.2 : version=“1.0”], key-id=h’04’, ens]

Condition Processing (cont.)



Discussion

- If new Evidence is asserted after ACS has processed basic augmentations
  - Evidence #2 is appended to ACS
  - E.g.,
    1) [[class-id=.3.2.1 : digest=h'FED4'], key-id=h'01', ev],
    2) [[class-id=.3.2.1 : digest=h'FED4'], key-id=h'02', rv],
    3) [[class-id=.3.2.1 : svn=7], key-id=h'03', ens],
    4) [[class-id=.3.2.2 : version="1.0"], key-id=h'04', en],
    5) [[class-id=.3.2.3 : digest=h'EDC3'], key-id=h'07', ev]

- A Reference Values condition could match on either or both evidence records
  - If it matches, Reference Values #2 entry is appended to ACS
  - E.g.,
    6) [[class-id=.3.2.3 : digest=h'EDC3'], key-id=h'02', rv]

Discussion

- Multiple Verifiers may cooperate to produce a distributed ACS1
  - VerifierA partitions ACS1 into $ACS1_A$, $ACS1_B$, and $ACS1_C$
  - VerifierB augments $ACS1_B$
  - VerifierC augments $ACS1_C$
  - Verifier D consumes $ACS1_A$, $ACS1_B$, and $ACS1_C$ to produce a final $ACS1_D$
  - $ACS1_D$ should be equal to ACS1 except for record ordering

$ACS1_B$

- Attester 1 session context

1) [[class-id=.3.2.1 : digest=h'FED4'], key-id=h'01', ev],
2) [[class-id=.3.2.1 : svn=7], key-id=h'03', en],
3) [[class-id=.3.2.3 : digest=h'EDC3'], key-id=h'07', ev]

$ACS1_C$

- Attester 1 session context

1) [[class-id=.3.2.1 : digest=h'FED4'], key-id=h'01', ev],
2) [[class-id=.3.2.1 : digest=h'FED4'], key-id=h'02', rv],
3) [[class-id=.3.2.1 : svn=7], key-id=h'03', en],
4) [[class-id=.3.2.2 : version="1.0"], key-id=h'04', en]

$ACS1_A$

- Attester 1 session context

$ACS1_D$

- Attester 1 session context

1) [[class-id=.3.2.1 : digest=h'FED4'], key-id=h'01', ev],
- ~~[[class-id=.3.2.1 : digest=h'FED4'], key-id=h'01', ev],~~
2) [[class-id=.3.2.1 : digest=h'FED4'], key-id=h'02', rv],
3) [[class-id=.3.2.1 : svn=7], key-id=h'03', en],
- ~~[[class-id=.3.2.1 : svn=7], key-id=h'03', en],~~
4) [[class-id=.3.2.3 : digest=h'EDC3'], key-id=h'07', ev]
5) [[class-id=.3.2.2 : version="1.0"], key-id=h'04', en]

Augmentation by Validation Function

- VF Augmentation – ACS changes state by processing a validation function:
  - VF Augmentation tuple: (<condition>, <function>,<authority>) => <output-set>
    - <condition> - a Claimset that is matched to the ACS
    - <function> - An action applied to <condition> in ACS
    - <authority> - the credential/key of the entity that asserted the tuple
    - <output-set> - The Claimset added to the ACS with authority and tuple context
    - If <condition> is true, then perform <function> and append function result as an <output-set> Claimset to ACS. Note: <output-set> could be <nil>
  - Example VF tuples in CoMID:
    - attest-key-triple-record
    - identity-key-triple-record

VF Augmentation Example

- Identity Key Triple
  - (<class-id=.3.2.1,[key-id=h'01']>, <$f_{KEY\text{-}VERIFY}()$>, <key-id=h'05'>) =>
    - <[class-id=.3.2.1 : [key-id=h'01'], result=VALID, key-id=h'05', vf]>

Discussion

- The identity triple is a bit like an endorsement where the function result is the endorsed Claimset.
- In CoMID, the function is defined as part of the triple predicate
- Note:
  - The Verifier is an agent of the Endorser hence, the result is asserted under the authority of the Endorser rather than the Verifier.
  - Maybe both authorities are needed?

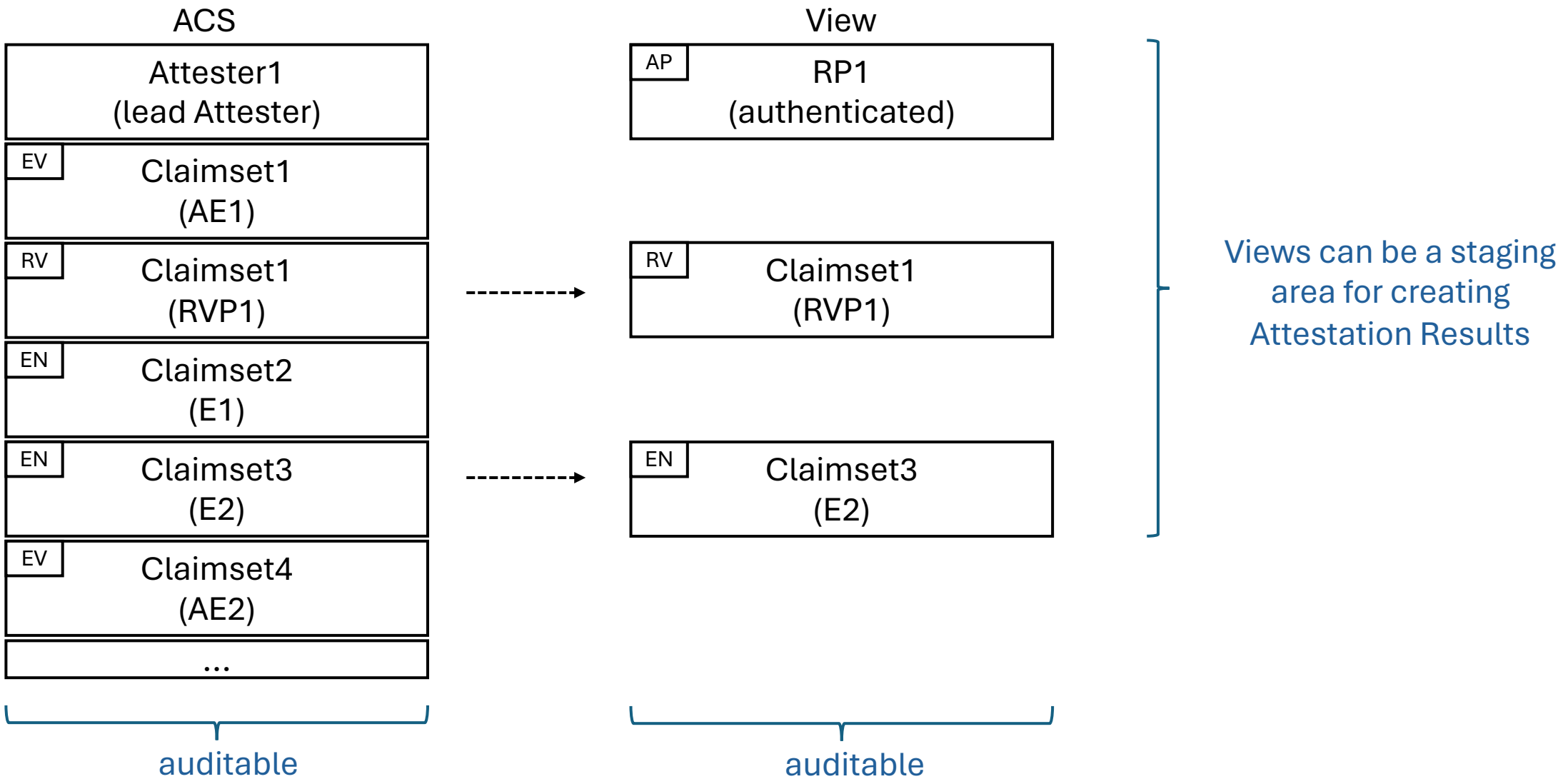
Resulting ACS – with VF Augmentation

- [[class-id=.3.2.1 : digest=h’FED4’], key-id=h’01’, ev],
- [[class-id=.3.2.1 : digest=h’FED4’], key-id=h’02’, rv],
- [[class-id=.3.2.1 : svn=7], key-id=h’03’, en],
- [[class-id=.3.2.2 : version=“1.0”], key-id=h’04’, en],
- [[[class-id=.3.2.1 : [key-id=h’01’], result=VALID], key-id=h’05’, vf]

ACS Restriction / Views

- View Restriction – A “view” of the ACS:
  - Restriction tuple: (<view-name>, <condition>, <authority>) => <output-set>
    - <condition> - A Claimset that selects Claimsets for placement in a View
    - <output-set> - The View
    - If <condition> is true, then select matching Claimsets from ACS and make them visible through <output-set> via a session/context authenticated by <authority>.
  - Receipt of RP Requests or Appraisal Policies triggers processing
    - If the ACS Augmentation inputs are still active, then ACS Restriction results may differ each time the same request is processed.

Processing an ACS Restriction / View



View Examples

- The view restriction tuple has a condition that constrains by Trust Anchors
  - (<“MyView”>, <ta=[key-id=h’02’, key-id=h’04’]>, <key-id=h’06’>) => <view-claimset>
  - The order of records in a View doesn’t matter

View Results:

- [view-name=“MyView”, key-id=h’06’],
- [[class-id=.3.2.1 : digest=h’FED4’], key-id=h’02’, rv],
- [[class-id=.3.2.2 : version=“1.0”], key-id=h’04’, en]

ACS Integrity Checking Example