

From Score to Evidence

Recomputation, Provenance, and Measurement Boundaries

This paper is informative. It explains assurance and verification boundaries in the current CAI design. Where it conflicts with the specifications and schemas published in the CAI repository, or with the applicable versioned rubric catalog, those control. Every statement here about current CAI state, including counts, versions, constants and tolerances, describes that reference state rather than the repository as it stands at the time of reading. The CAI implementation is under active development and changes several times a week, so a reader consulting this paper later should expect the repository to have moved and should recheck any current-state figure against the reference state recorded below.

Version	1.0
Date	17 September 2026
Signed	17 September 2026 by Jimmy Borch
Author	Jimmy Borch
Originating organisation	Canine Development
Series	CAI Founding Papers
Status	Informative, non-normative technical paper
Technical reference state	CAI repository at commit a24b47b2eb46971dcd530f3a705ebb419aea158c, 16 September 2026

Contents

Abstract

1. A score you can inspect

2. Define the verification object before claiming verification

3. Layer one: repository to evidence

4. Layer two: evidence to score

5. Layer three: package, provenance and integrity

6. Why canonicalisation exists

7. Signature semantics and trust

8. Challenge routing by claim type

9. What a neutral verification surface should expose

10. Make the trust boundary visible in the report

Verification matrix

Figure 1. Trust-boundary model

Figure 2. Two independent failure examples

Limitations and maturity boundary

Conclusion

References

Abstract

A CAI result includes the evidence and rule identity needed to inspect and recompute the published score. The Codebase Assurance Index is designed so that a result can carry its evidence, identify the rules under which it was calculated, be recomputed by a recipient from the supplied evidence under the published rules, and travel with provenance that makes its origin inspectable. Under the current CAI design, the scoring contract is deterministic: given the same eligible evidence and the same rubric version, the scoring process produces the same result. The versioned rubric identifies the score-moving semantics, allowing a recipient to inspect both the result and the rule set that produced it.[1][2][5]

A recipient can therefore work backwards from the published result into the scoring inputs, the governing rubric and the provenance of the delivered artifact. Those checks establish different things. Recomputation establishes that a result follows from stated evidence and rules; provenance establishes where an artifact came from and what it claims to describe; cryptographic integrity can establish that a signed payload has not been altered under the applicable trust model; and measurement validity concerns whether the evidence correctly represents the repository that was measured. CAI treats these as separate assurance layers rather than reporting them under a single undifferentiated claim of being "verified".

This paper describes those layers, the boundaries between them, and the information a technically meaningful CAI verification surface should expose. Each claim is tied to the mechanism that can establish or challenge it, so the recipient can see what has been tested and where the remaining evidential responsibility sits.

1. A score you can inspect

A software score presented without its method, inputs or subject identity leaves the reader with little to inspect beyond the conclusion. A CAI result is designed to carry the material needed to examine how that conclusion was produced. That changes the recipient's task from deciding whether to trust an unexplained number to checking a set of identifiable claims about evidence, rules, subject and provenance.

A published CAI result can identify four things about itself.[1][3][5] The first is the evidence from which the score was derived: the structured record of what an inspecting tool observed in the codebase, item by item, in the form the scoring process consumes. The second is the versioned rubric that defines the scoring semantics: the named and dated document that fixes which items are scored, how they group, and how their numbers combine into a headline. The third is the software subject the artifact claims to describe, normally a repository and, where the producer records it, the revision that was inspected. The fourth is the provenance associated with its delivery: what the artifact states about who issued it, when, and through which path it reached the reader.

With those elements attached, the score can be taken apart. Instead of receiving only the statement:

This codebase scored 72.4.

the recipient can ask:

Which evidence produced 72.4?
Which version of the scoring rules was used?
Does 72.4 actually follow from those inputs?
Which repository and revision does the result describe?
Who produced the measurement?
Has the delivered evidence changed since it was issued?

Those questions determine what a recipient can actually check about the result. Each proposition can be tested independently and can fail independently. Reporting only that a result is *verified* hides which claim was tested, by what mechanism, and which claims remain outside the check.[2][3]

A recipient may want to establish:

1. that the published result follows from the supplied evidence;
2. that the correct versioned scoring rules were used;
3. that the evidence and result belong to the stated software subject;
4. that the artifact retains its published integrity and provenance;
5. and, separately, that the underlying measurement is a defensible representation of the repository.

Each question has a different technical object and a different way of being challenged. The first concerns arithmetic and is settled by performing it again. The last concerns judgement about a codebase and is settled, if at all, by examining the producer's evidence and method. Between them sit questions about identity and integrity, which are settled by inspecting an artifact rather than by inspecting a repository. The questions remain separable: satisfying four of them does not settle the fifth, and each mechanism should be described only at the scope it actually reaches.

The deterministic scorer tests the arithmetic. The reference implementation is public, so a recipient holding the evidence and the named rubric can perform the fold again, meaning run the same aggregation over the same inputs, and compare the result with what was published. Agreement or disagreement is then a mechanical finding. The

operation begins after the evidence exists, so it has no independent view of the repository from which that evidence was produced.

The versioned rubric identifies the scoring rule. The August 2026 scoring change made the catalog the place where the parameters capable of moving the number or the label are pinned, so that a rubric version fixes the scoring semantics rather than merely labelling them.[5][13] Section 4 records how far the published catalogs have carried that. A scorer build or package version may travel alongside as provenance, but it is not a second authority for what a given rubric version means. Two results calculated under different rubric versions were calculated under different rules.

The delivery artifact can bind subject, producer, evidence and result. Left as separate files, those elements have to be assembled correctly by whoever holds them, and an assembly error may leave no trace. Bound into a single payload, the association becomes part of the delivered object: this evidence, from this producer, under this rubric, describing this subject, yielding this result.

Cryptographic verification tests integrity under the stated trust model. A signature over the payload lets a recipient detect alteration after issue and check that the payload was signed by the holder of a key associated with the named issuer. It does not establish that the underlying measurement is correct; the signature protects the statement that was signed, while the strength of the issuer binding depends on the arrangements around the key.

Measurement evidence and the issuer's review process address whether the repository was measured correctly. That question remains with the producer because the mechanisms above operate on the evidence and artifact after measurement has taken place. A challenge to the measurement is therefore a challenge to the producer's judgement about the codebase, not to the arithmetic subsequently applied to the resulting evidence.

The current CAI challenge process follows the same division. It distinguishes disputes about arithmetic, reference-scorer behaviour, repository measurement and the rubric itself rather than routing all disagreement through one appeals process.[2] A dispute can therefore be stated in a technically useful form: which part of the evidence chain is contested, and what evidence would settle that particular claim?

2. Define the verification object before claiming verification

Verification starts by identifying what is being tested. The word names a relation between a claim and a procedure, so until the claim has been stated there is nothing a procedure could establish or refute. This discipline is prior to any question about cryptography, recomputation or trust, and the rest of the paper depends on it.

A score without a subject, evidence, rule identity or issuer is a poor verification object. There is no defined proposition to test against the number, so the reader is left deciding how much credence to extend to whoever published it.

72.4 has no repository, no revision, no evidence, no measurement provenance, no scoring rule and no issuer attached to it. It is only a number. Two results both reading 72.4 may describe different codebases, inspected on different days, under different versions of the scoring rules, and nothing in the figure distinguishes that case from one where the same repository was measured twice.

For a CAI result to function as technical evidence rather than an isolated assertion, several identities need to travel with it. Each closes off a specific way the result could be misread, and each is separately checkable, which is what distinguishes a verification object from a presentation of a conclusion.

At minimum, the recipient needs to know:

- what software subject the result claims to describe;
- which repository revision or other point-in-time identity applies, where available;
- who produced the measurement;
- what evidence was produced;
- which rubric version governs the fold;
- what result was claimed;
- what provenance or integrity mechanism, if any, binds those elements together.[1][3]

The current CAI delivery-package design carries repository identity, optional commit identity, producer information, the rubric version, evidence, verdict, issue time and signing information inside one payload.[3] Holding them in a single payload rather than in adjacent files is a substantive choice and not a packaging convenience, because it makes the association between the elements part of what is delivered. A recipient inspecting the result is then inspecting the same object the issuer bound together, rather than a collection somebody assembled afterwards.

This general design principle has precedents outside CAI. The in-toto Statement specification, for example, explicitly binds an attestation to identified software subjects and normally identifies them by cryptographic digest.[8] SLSA similarly defines provenance as verifiable information describing where, when and how an artifact was produced.[9] Both were developed for software supply-chain problems rather than assurance scoring. Their relevance here is the shared requirement that a statement identify the software subject it describes.

These are precedents, not CAI dependencies. CAI does not become in-toto or SLSA by using the same subject-identification principle, and nothing here should be read as a claim of conformance to either specification.

The underlying requirement is straightforward: a claim about software should identify the software it is a claim about.

The same principle applies to the scoring rule. A result is a claim about a codebase, but it is equally a claim made under a particular set of rules, and a reader who cannot name those rules cannot say what the number was asserting.

The current CAI scorer resolves score-moving semantics from the named rubric where the rubric supplies them. Following the August 2026 scoring-parameter change, the catalog format defines a block for the parameters capable of

moving the numeric or the categorical result, and those parameters divide according to which of the two they reach.[5]
[13]

Three of them reach the number. Two are the decay constants of the ordered weighted averaging applied at each stage of the fold, meaning the arithmetic that combines many measured items into fewer: the values being combined are sorted from worst to best, and the weights fall geometrically down that order, so the weakest value carries the largest share and the decay constant sets how far it dominates the rest. One constant governs the fold inside a lens, meaning one of the ten grouped areas of assessment such as Code Health or Security and Compliance, while a separate and sharper one governs the fold of lens results into the headline. The third parameter is the architecture-surface floor, which caps the Architecture lens where a repository presents too little analyzable surface for cross-project measurements to carry signal, on the reasoning that such measurements are trivially satisfied on a repository that has almost nothing for them to examine.

The remaining parameters reach the label and leave the number untouched. The band cutlines are the thresholds at which the headline stops being read as a figure and starts being read as a word. The critical gate caps a lens's displayed band where any measured contributor falls below a stated threshold, so that one serious weakness cannot be averaged into a comfortable description of the whole. The quality bar shifts the cutlines together according to how strict the reading should be for the kind of software in question, since what is reported as adequate for a prototype and what is reported as adequate for a mission-critical system are not set at the same place. In each of the three the arithmetic is untouched and only its description moves, so the figure remains the figure the fold produced, whatever expectation it is being reported against.

`rubricVersion` therefore identifies the rule set under which the result was produced. A result published without it has not stated its scoring semantics. Two figures folded under different rubric versions were produced under different rules even where the evidence beneath them was identical, and the numbers alone do not reveal that difference.

A scorer build or package version may remain useful provenance, recording which implementation performed the fold on a given occasion, but it is not a second competing authority for what a particular rubric version means. One result should have one rule authority. Under the current design that authority is the rubric, and the arrangement is examined further in section 4.

3. Layer one: repository to evidence

The first assurance layer is where the software itself is turned into structured evidence. Nothing in the layers above can begin until this one has finished, because every later operation acts on the record this layer produces rather than on the codebase that record describes.

Before CAI can aggregate anything, somebody has to inspect a repository and produce measurements. That party is the producer: the tool, service or organisation that reads the source, decides what it is looking at, applies the criteria and records the outcome.

That is measurement: it forms judgements about a body of code. Every later stage operates on the resulting record without returning to the codebase, so the repository-to-evidence step has a different evidential character from the operations that follow.

The producer decides, through its implementation and measurement process:

- which repository state is being inspected;
- which criteria apply;
- what evidence can be observed;
- which measurements resolve;
- what coverage supports them;
- when the producer should abstain;
- which values are emitted into the score-bearing evidence.[1][6]

Each item on that list is a decision that could have gone otherwise, and several are decisions a reader of the final number will never see. Coverage is one of them: a dimension, meaning a single assessed criterion in the rubric, carries a score on a scale of zero to ten together with a coverage figure between zero and one recording how much of the ground the criterion was meant to examine was actually reached, and the value the fold consumes is the product of the two.[3][13] Abstention is another, since a producer that declines to resolve a criterion has exercised judgement quite as much as one that resolves it.

Much of the technical judgement sits in this layer. A criterion has to be recognised as applicable before it can be assessed, the assessment has to be grounded in something observable in the source, and the outcome has to be reduced to a value on a fixed scale. None of those steps is mechanical, and the quality of the eventual number is bounded by them.

Keeping that boundary visible prevents the rigour of the deterministic arithmetic from being attributed to the upstream measurement judgements. The score can be recomputed exactly even when the measurement that supplied one of its inputs remains open to challenge.

The reference scorer is intentionally downstream from measurement. Its role is to turn eligible evidence, meaning the score-bearing items the rubric admits into the fold, into a deterministic result under a named rubric. That gives the recipient a clean boundary: repository measurement produces the inputs; CAI scoring applies the published rules to those inputs.

If a producer evaluates a dimension incorrectly as 4.0 when the defensible measurement was 8.0, a perfect scorer will reproduce the 4.0. It has no independent access to the repository against which the discrepancy could be noticed, and no representation of what the dimension was meant to be measuring beyond the value it was handed.

Determinism describes the relation between input and output: the same eligible evidence under the same scoring rules produces the same result. It does not establish the separate relation between the evidence and the repository that evidence claims to describe.

The current CAI delivery specification makes this boundary explicit. Its signed package can bind and authenticate the supplied evidence under the stated trust model, and the embedded evidence can be folded again through the open scorer. The specification also states that this does not prove that the evidence is a truthful measurement of the actual repository.[3] The limit is therefore stated in the specification itself and can be checked against the normative delivery design rather than inferred from commentary around the result.

Keeping the boundary explicit prevents arithmetic verification from being reported as repository re-measurement. The delivery specification itself states that limit, so the verification claim can be checked against the specification rather than inferred from presentation language.

The same boundary appears in established measurement practice. ISO/IEC/IEEE 15939 treats measurement as a process involving information needs, definition and application of measures, analysis, and evaluation of whether analysis results are valid for their intended use.[7] Deterministic calculation of an output is one part of a broader measurement chain. The standard is cited here for the shape of that chain and not as an accreditation, since CAI is not claimed to conform to ISO/IEC/IEEE 15939 and the citation would not support such a claim.

The phrase *reproducible score* therefore needs a defined object. Under the current model it refers to reproduction of the score from supplied evidence under the named rubric, not independent reproduction of the repository assessment. Conflating those claims would attribute a property of the arithmetic to the upstream measurement process.

Under the current CAI model, the reference scorer begins after score-bearing evidence already exists. Its starting position is what fixes the boundary, and every claim it can support is a claim about what happens from that point onward.

It can therefore answer:

Does this result follow from these inputs under this rubric?

A recipient can test that claim directly from the supplied evidence and published rules. The check requires no privileged access or cooperation from the issuer, and it does not depend on the recipient sharing the issuer's judgement about what good software looks like.

The separate question is:

Were these the correct inputs to derive from the repository?

That remains a measurement question, and it belongs with the producer and the evidence supporting the measurement. It is answered, where it is answered at all, by examining how the producer works and what it recorded, and section 8 sets out the path by which a recipient raises it.

3.1 What open recomputation establishes

Open scoring addresses the aggregation layer, not the measurement layer. The producer still decides how the repository is inspected and which evidence is emitted, so making the scoring rule public does not make those upstream judgements public or independently reproducible. It does, however, remove a separate class of discretion from the point where fixed evidence becomes a published result.

Once the evidence and governing rubric are fixed, the headline is determined by the scoring rule. An issuer that wanted to publish a different number would have to change the evidence or the rubric rather than adjust the output after the fact, and either change can be inspected in the artifact and rule identity. A recipient can perform the same fold independently and establish whether the published result follows from those inputs.

This assurance property covers recomputation of the published result rather than reproduction of the entire repository assessment. The test is available to any recipient holding the artifact, its outcome does not depend on the recipient's opinion of the issuer, and disagreement can be demonstrated with the same evidence and rule set. The measurement layer remains open to a different form of challenge because recomputation has no independent access to the repository.

The scope of the claim therefore needs to remain stable in both directions. Open recomputation should not be described as independent re-measurement, but its inability to settle measurement validity does not erase the constraint it places on scoring. The property is specific: fixed evidence and a fixed rubric imply a fixed result that a recipient can recompute.

4. Layer two: evidence to score

The evidence-to-score layer supports a claim that can be settled by computation alone. A recipient can determine whether the published result follows from the supplied evidence under the named scoring rules without returning to the repository or asking the producer to adjudicate the calculation.

The current public documentation states the invariant as:

same evidence + same rubric version gives the same score.[1][2][13]

The project's own summary description compresses this considerably further, to the effect that the same code yields the same score.[1] The shorter form serves as a tagline but does not survive as an invariant, because the code does not by itself determine the evidence: two surveys of one repository may resolve different criteria to different depths and emit different bundles, from which the fold will correctly produce different numbers. This paper therefore uses the evidence-level invariant throughout: the evidence, not the codebase by itself, is the input to the deterministic fold.

The August 2026 scoring change materially strengthened the version boundary behind that statement. Before it, naming a rubric version identified only part of what governed the result.

Previously, some score-moving behaviour existed as constants in the scorer rather than as data selected by the rubric version. The catalog format now defines a scoring block for the parameters capable of moving the fold, covering the within-lens and across-lens ordered-weighting decays, critical-gate behaviour, architecture-surface parameters, band cutlines and quality-bar shifts.[5][13] Section 2 sets out what each of those governs. The list must be exhaustive: a score-moving parameter left outside the versioned rule set would be sufficient to break the invariant.

A catalog publishing no scoring block resolves to the scorer's default values, which are the constants it has always applied, so every previously published version keeps folding to the number it was computed to.[5][13] The design record is explicit about why pinning them in the versioned catalog matters: a constant frozen in code is an unenforced promise of stability, since nothing prevents its being edited and no archived catalog can detect that it was.[13]

As published at the reference state of this paper, no catalog carries the block. All thirty-eight catalogs published as at 16 September 2026, the most recent included, resolve to the defaults.[13] The mechanism is therefore in place while the property it exists to establish is not yet exercised, so a rubric version currently fixes the catalog contents and the dimension-to-category map, while the fold's constants and the cutlines still come from the build performing the fold. Two recipients running different scorer builds over the same evidence and the same rubric version are therefore relying, for those constants, on the builds agreeing. At present that agreement is a fact about the implementation rather than a property carried by the artifact.

A result published under an early rubric replays under the rules that produced it rather than under current ones, which is what allows a historical score to remain checkable rather than merely archived. That holds under the defaults as it will hold under the block, the defaults being the values those versions were computed under.[5][13]

Carrying the rule with the rubric identity avoids a hidden version switch inside the scorer. If the interpretation of an old result depended on the build that produced it, the recipient would need implementation history in addition to the rubric identity in order to replay the score.

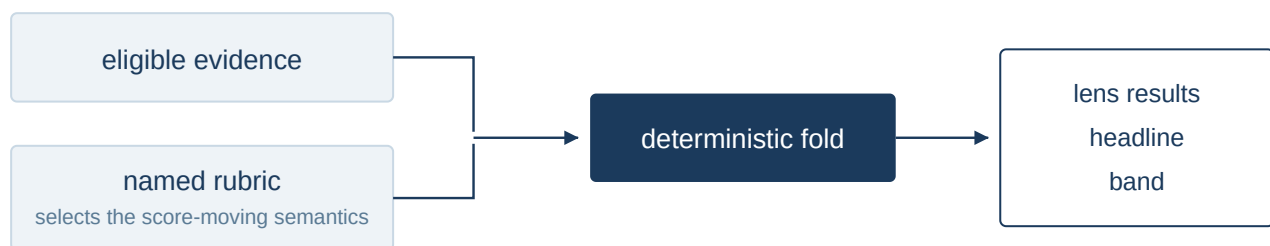
The delivery specification also defines a second version axis, `schemaVersion`, with a different failure mode. The rubric version identifies the scoring semantics. The package carries its own `schemaVersion`, which identifies the wire shape of the artifact rather than the rules applied inside it, and the specification states that neither axis implies the other.[3] That field sits inside the signed payload, so the version cannot be stripped or lowered in transit.[3]

A recipient can therefore meet two failures a report should not present alike. In the first, the package is read and the recomputed result does not match the claim, which is the mechanical discrepancy this section has described. In the second, the verifier declines the package before attempting any arithmetic, because its major format version is one the verifier does not implement.[3] The first is evidence about a result. The second is evidence about a tool, and a recipient told only that verification failed may conclude the number was false when what happened is that their verifier is older than the artifact.

The axes are independent in both directions. A change to the payload structure does not alter the meaning of a score computed under a named rubric, since the rubric governs the fold and the format governs the container. A new rubric version does not require a new package format, since the fold's output has the same shape whatever constants produced it. A verification report should identify which axis failed so that an unsupported package format is not reported as an arithmetic discrepancy.

Conceptually:

THE DETERMINISTIC FOLD



The same two inputs produce the same output, and nothing in this step returns to the repository.

The verification question at this layer is deliberately mechanical:

Does the claimed result follow from these score-bearing inputs under these versioned rules?

If yes, the arithmetic claim is reproduced. The recipient has established a fact about the relation between the published inputs and the published output, and has established nothing beyond it.

If no, there is a falsifiable discrepancy. Something in the chain is wrong, and the disagreement can be exhibited rather than asserted.

The CAI challenge document treats this as a mechanical challenge because it does not require an adjudicator to choose between competing interpretations. A minimal reproduction case can demonstrate that the published result and the reference method disagree.[2] A dispute of that kind is settled by whoever runs the computation, and the party that published the result holds no better position from which to settle it than the party that questioned it.

The recipient can test the relationship between evidence and result without privileged access to the issuer. Within that scope, the same evidence and rubric provide the same recomputation test to any recipient holding the artifact.

4.1 Determinism and verifier tolerance

At the reference state, the Draft delivery-package verifier accepts headline reproduction within a tolerance of ± 0.5 . [3] A recomputed headline differing from the claimed headline by less than half a point is therefore recorded as agreement.

The ± 0.5 tolerance is a verifier acceptance policy, not part of the scoring function. It describes how much divergence the current Draft delivery profile permits before reporting failure; it does not redefine the deterministic contract of the

fold.

The scorer's deterministic property and the delivery verifier's acceptance profile are different things. The first is a claim about a function, that identical inputs yield identical outputs. The second is a policy governing how much divergence a particular checking tool will permit before reporting a failure, and a policy of that kind can be tightened, loosened or replaced without the function changing at all.

The current tolerance has a practical consequence. Two independent implementations disagreeing by four tenths of a point would pass this verifier as though they had agreed, and the disagreement would leave no record. That is a defensible position for a Draft profile operating inside a single implementation, and a position that a later conformance regime intended to establish agreement between independent implementations would need to revisit.

The current delivery specification also defines fixed output precision for values embedded in the package.[3] Scores are stored rounded to two decimal places, and the canonicalisation step depends on that pre-rounding so that no question of numeric representation remains at the point of signing.[3] The package is thus precise to a hundredth of a point at rest while its verifier accepts a difference fifty times larger, and the interval between those two figures is the space in which a genuine implementation divergence could presently pass unnoticed. Final conformance work should continue to make the canonical numeric and display rules explicit enough that independent implementations can agree on the published value without relying on a broad tolerance.

Until then, the technically careful statement is:

CAI scoring is intended to be deterministic under the named evidence and rubric. The current Draft delivery verification profile separately applies a ± 0.5 headline acceptance tolerance.

The two statements describe different layers: one defines the scoring function and the other defines the verifier's acceptance policy. The acceptance policy should therefore not become an implicit definition of the scoring contract. A reader who encounters only the tolerance will conclude that CAI scores are approximate; a reader who encounters only the contract will assume that any two conforming verifiers must already agree to the last digit. Neither conclusion is supported by the current design, and each becomes harder to dislodge once it has been drawn.

4.2 Recomputation as a recipient test

Suppose an evidence bundle contains:

```
D17 = 6.2  
D18 = 7.7  
D19 = 9.1  
...
```

A verifier can use the applicable rubric to reproduce the categories, lenses and headline. The rubric assigns each dimension to a category, meaning a group of related criteria that are averaged together before anything larger sees them; categories then combine into lenses, and the lens results into the single headline figure.[13] Every stage of that sequence is published, so the verifier is not reconstructing the method but executing it.

This gives the published score a testable derivation. The recipient can execute the same aggregation against the supplied evidence instead of relying on a displayed result alone.

The verifier does not need to trust a screenshot, exported PDF or hidden calculation. None of those is evidence of anything beyond what somebody elected to display.

It can perform the fold again. The recomputation is itself the check, and it either yields the published number or it does not.

What that operation does not infer from those numbers alone is whether D17 deserved 6.2. The fold treats each value as given, and nothing in the arithmetic could register the difference between a well-founded value and a careless one.

That claim belongs upstream. It is settled, where it is settled at all, by the producer's own evidence and by the review procedures surrounding it, which is the subject of section 3 and of the dispute path in section 8.3.

This is the central trust boundary in this paper:

Recomputation verifies the fold from evidence to result. It does not independently reproduce the repository-to-evidence measurement.

The architecture therefore supports two separate findings at once: the score can reproduce correctly from the supplied evidence while the validity of one or more measurements remains open to challenge. Each finding belongs to a different layer and requires different evidence.

5. Layer three: package, provenance and integrity

A successful recomputation establishes that the published result follows from the supplied evidence under the named rules. Provenance supplies the context that arithmetic alone cannot: what subject the artifact describes, who produced and issued it, and when it was issued.

The remaining questions concern the origin and scope of the artifact:

- Whose evidence is this?
- Which repository does it claim to describe?
- When was it issued?
- Which rubric was used?
- Has anything changed since the issuer created the artifact?

This is the provenance and integrity layer. Provenance concerns where an artifact came from and what it is bound to; integrity concerns whether it still says what it said when it was issued. Neither is a claim about the codebase, and both must hold before a claim about the codebase can be attributed to anybody.

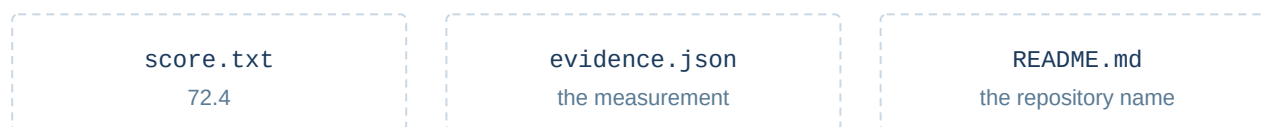
The current CAI delivery-package design puts the evidence and verdict inside a signed payload together with fields describing the subject, producer, issuer, rubric version, measurement context and issue time.[3] Two of those fields require separating, because this paper has so far treated them as one. The producer is the party that measured the code, recorded as a name together with the scanning implementation and its version. The issuer is the party that signed the payload, recorded as a name and a key identifier. Under the current deployment they are not the same party: the producer pushes its evidence to the registry, and the registry stamps the producer field from the authenticated push and signs the result.[3][4] Measurement context, in turn, is descriptive material the producer supplies alongside the scored evidence, which travels with the result without entering the arithmetic.[3]

The resulting artifact carries several related claims together and creates a specific chain of attribution. The signature establishes that the issuer vouches for the payload; one field in that payload identifies the producer that supplied the evidence. A recipient who verifies the signature has therefore established that the issuer attributes the evidence to that producer, not that the producer independently signed the measurement.

Binding the elements into one delivered object reduces the risk that evidence, result and subject identity are separated or recombined later. The specification makes the producer field part of that binding by having the registry stamp it from the authenticated push rather than accepting an arbitrary producer declaration in the payload.[3]

Consider three files:

THREE FILES, NOTHING BINDING THEM



Nothing in the set records that these three belong together.

A substitution leaves no trace, and no party need have acted dishonestly for one to occur.

If `score.txt` says 72.4, `evidence.json` contains the measurement, and `README.md` names the repository, a recipient has to trust that somebody assembled the correct three files. Nothing in the set resists recombination:

evidence from one survey can be filed beside the score from another without leaving any trace of the substitution, and no party need have acted dishonestly for that to happen.

In a bound artifact, the association between the elements is a property of the delivered object rather than a convention maintained by whoever stores the files. A later substitution changes the object that was signed and can therefore be detected under the stated trust model.

It can say, in one authenticated object:

WHAT THE BINDING CARRIES

ONE SIGNED PAYLOAD	
this subject	was measured by
this producer	using
this evidence	under
this rubric	producing
this result	at this time

The association is part of the delivered object, so a later substitution changes what was signed.

This is the provenance claim: an identified issuer binds the subject, producer, evidence, rubric, result and issue context into one artifact that can be checked without returning to the codebase. It also makes the result more portable because the context needed to interpret it can travel with it. A score separated from its rubric version, its subject and its issue time can be re-contextualised only by somebody who still remembers the circumstances, whereas a bound artifact carries them and remains interpretable long after the transaction that produced it has been forgotten.

The same idea appears in established software supply-chain approaches. SLSA describes provenance as verifiable information about where, when and how an artifact was produced.[9] The in-toto Statement structure binds authenticated metadata to identified subjects.[8]

CAI does not claim conformance to either system through this comparison. They are cited as precedents for binding provenance to an identified software subject, not as interchangeable specifications.

In CAI, provenance has a bounded technical role: it describes and binds the artifact's origin and subject. It does not establish whether the repository measurement itself was correct.

It tells the recipient what the artifact says about its origin and subject, and it gives integrity mechanisms something concrete to protect. A signature over an unstructured claim protects very little, because there is little in it whose alteration a reader would notice; a signature over a bound payload protects every field the binding contains.

A recipient may also encounter a content digest alongside a delivery. The registry records a SHA-256 over the same canonical bytes the signature covers, and uses it as the handle by which a delivery is stored, deduplicated and referred to.[4]

The digest identifies the delivery bytes. It lets two parties confirm that they are discussing the same delivery, lets the registry recognise a resubmission it already holds, and lets a recipient match a copy against the registry record without resolving a signing key. It does not add a second integrity mechanism: a change large enough to alter the digest would also cause signature verification over the protected bytes to fail. Reporting the digest and signature as independent

confirmations would therefore count the same byte-level alteration twice. The delivered artifact carries no digest field of its own; the digest is the registry's record of the artifact.[3][4]

5.1 If the producer is already trusted, why use a registry?

The registry provides distribution and provenance functions; it does not determine whether the measurement or CAI arithmetic is correct. In the current v1 design it verifies the delivery package on ingest, checks the trusted signing path, stores the accepted delivery under the stable delivery identity assigned when the package is pushed, and makes the artifact and verification material available to recipients.[4] That gives a recipient a stable object that can be retrieved and checked after the original transaction, including in cases where the producer itself is already trusted but the recipient still needs to establish that the artifact in hand is the one that was issued.

Whether a registry should become a permanent normative CAI requirement is a separate architecture and governance question. The current implementation demonstrates one workable delivery pattern. It should not be mistaken for the only implementation CAI could permit, and the present arrangement, in which the registry is also the signing issuer, is a property of this deployment rather than a requirement the standard imposes.

6. Why canonicalisation exists

Cryptographic signatures operate on bytes. The operation takes a sequence of octets and a key, and produces a value that can later be checked against the same sequence and the corresponding public key, with no notion of what those octets were intended to mean.

Humans usually think of JSON as structured information. A reader who opens a payload sees fields and values, and treats two documents carrying the same fields with the same values as the same document.

Those are different representations. The difference becomes operational when one party signs a byte sequence and another party later has to reproduce the same sequence for verification.

These two JSON objects communicate the same simple data to a normal parser:

```
{"score":72.4,"band":"Strong"}
```

The same values can also be written as:

```
{  
  "band": "Strong",  
  "score": 72.4  
}
```

Their byte representations are different. They differ in the order of the two members and in the whitespace between them, and either difference on its own would be sufficient.

A signature created over one byte sequence will not automatically verify over the other. It will fail, and it will fail in the same manner as a tampered payload, leaving the verifier no means of distinguishing a formatting difference from an attack.

Independent signing and verification therefore require a deterministic method of turning the structured payload into the bytes that are actually signed. The requirement is representational rather than cryptographic: two parties who have agreed on the meaning of a document must also agree on its spelling before either can sign it.

RFC 8785 defines the JSON Canonicalization Scheme for this purpose. Its stated objective is an invariant JSON representation suitable for repeatable hashing and signing, including deterministic serialization and property ordering.[10]

The current CAI delivery format identifies its canonicalisation profile as `RFC8785-json` and uses a reference canonicaliser before Ed25519 signing.[3] The canonical form is obtained by re-emitting the payload with its keys sorted, without insignificant whitespace, encoded as UTF-8, and with number tokens preserved as written, the values in a package having been rounded to fixed precision beforehand so that no question of numeric representation remains at that point.[3] Both of the objects above reduce to the same canonical bytes.

The closed-loop system therefore has a stable byte representation for signing and verification. Because the signature covers the canonical form rather than the presentation formatting of the downloaded JSON, a recipient can re-indent or reformat the human-readable package without changing the canonical bytes used for verification.[3] The specification explicitly chose this approach over an opaque encoded wrapper so that the evidence artifact remains readable and comparable while still having a deterministic signing representation.[3]

Canonicalisation also affects how a verifier handles a field it does not recognise, and the specification defines that behaviour rather than leaving it to implementations. The compatibility rule is additive. A minor version may introduce new optional fields, a verifier is expected to ignore ones it does not know, and older packages keep verifying; a major version, which may change the shape or the signing scheme itself, must be rejected by a verifier that does not implement it.[3]

Ignoring a field is not the same as discarding it. The signature covers the canonical form of the whole payload, so every member present at signing contributes to the bytes that were signed, including members a given verifier has no interpretation for.[3] A verifier that drops an unfamiliar field before re-canonicalising is not checking the artifact it was given but a shortened version of it, and it will observe a failure indistinguishable from tampering.

Forward compatibility therefore rests on an obligation about handling rather than about understanding. The re-emitted payload must preserve what arrived, whatever the verifier makes of it, which is a stronger requirement than parsing and one an implementation can fail while appearing correct, since a package containing no unfamiliar fields verifies either way.

That is the same maturity distinction this section is about to draw. A compatibility rule stated in a specification and a compatibility rule demonstrated against a package the verifier does not fully understand are different kinds of evidence, and only the second survives contact with an implementation somebody else wrote.

The reference implementation shows where the rule will first be tested. It parses a package into typed objects and canonicalises from those rather than from the bytes received, and it models no catch-all for members it does not know, so a field introduced by a later minor version would be absent from the form the signature is checked against.[14] Nothing fails today, no such field having been minted, which is precisely the condition under which a compatibility rule goes unexercised.

The current maturity boundary is stated in the Draft delivery specification, which says that the existing closed loop relies on the same reference canonicaliser and that full independent RFC 8785 string and number conformance for specification-only third-party verifiers is deferred.[3] The distinction is between agreement reached by construction and agreement demonstrated by test: parties running one implementation cannot disagree about canonical bytes, which is not the same proposition as having shown that a party working only from the specification would arrive at them too.

Inside the existing closed loop, the components agree on the canonical bytes through a shared implementation. What has been established is that the pipeline is internally consistent, which is a real if modest property, and it is the property the specification claims rather than a stronger one it declines to claim.

This is implementation evidence: it shows that the present components agree on the canonical bytes. It does not yet demonstrate that an independent implementation working only from the specification will produce the same representation.

The next maturity step is to make the same property independently demonstrable through published conformance cases and genuinely separate implementations. Until a second implementation exists, no test can distinguish a specification precise enough to be implemented twice from one that merely appears so to the people who wrote it.

The path is therefore additive:

1. stable canonicalisation inside the current implementation;
2. published conformance vectors;
3. independent third-party implementations reproducing the same canonical bytes and verification outcomes.

This is the kind of progression a technical standard should make visible. Each step is a claim of a different strength, and stating which of them has been reached is more informative than describing the sequence as a whole as interoperability.

7. Signature semantics and trust

The current CAI delivery format uses Ed25519 signatures.[3] Ed25519 is an instance of the Edwards-curve Digital Signature Algorithm specified in RFC 8032.[11] The assurance question in this section is not the choice of Ed25519 itself but the claim a valid signature supports and the trust model required to interpret it.

The cryptographic operation gives the artifact an integrity and authenticity mechanism. Integrity, in that a modified payload will no longer verify; authenticity, in that verification succeeds only for a payload signed with the private key corresponding to the public key used in the check.

A verifier can ask whether a signature validates over the canonical payload using a particular public key. The question is closed and mechanical, it is answered without reference to any party's intentions, and it returns the same answer for everyone who asks it.

If it does, that establishes a cryptographic relationship between:

- the canonical payload;
- the signature;
- the corresponding key pair.

That establishes a relationship among the payload, signature and key pair. Any claim about the identity or competence of the signer depends on information outside the cryptographic operation.

It means a recipient can detect an altered payload under the stated trust model, and can bind the protected fields together rather than trusting an unstructured collection of files. Every field inside the signed payload is protected at once, which is where the binding described in section 5 acquires its enforcement.

The next question is operational:

Why should this public key represent the issuer I think it represents?

That depends on the trust model surrounding the key. The signature establishes that some holder of a private key signed the payload, while the identity attached to that key is supplied by the arrangements around it, and those arrangements are not cryptographic objects.

Key issuance, custody, rotation, retirement and the mapping between a key identifier and an issuer are not solved merely by selecting a strong signature algorithm. Each is an operational commitment that can be documented, audited or neglected independently of the mathematics, and a system with an impeccable algorithm and careless key custody is weaker than one with the reverse.

The current CAI package carries a `keyId`, while the registry exposes trusted verification keys. The Draft delivery design describes key rotation by retaining retired public keys so older artifacts remain verifiable.[3] The key set is published with each key marked active or retired, is retrievable without authentication, and can be pinned by a recipient, which is what allows verification to be performed offline against a set obtained in advance.[3][4] Retention has a specific consequence: because a retired key continues to verify the deliveries signed under it, retirement records that a key is no longer in use rather than that artifacts signed with it should now be doubted, and whether a given key was sound at the moment it signed is not a question the signature can answer.

The current v1 ecosystem is also explicitly closed-loop. The registry contract describes a single trusted producer operating with the CAI-issued signing path, with third-party scanner conformance and certification deferred.[4] The consequence for attribution follows from the separation drawn in section 5: the signature authenticates the issuer,

while the producer named in the payload is an assertion the issuer makes on the strength of having authenticated the party that pushed the evidence. A recipient's confidence in producer attribution therefore rests on the registry's admission process rather than on the signature by itself.

The current deployment therefore has a working trust path: a recipient can obtain the key set, verify the artifact offline and identify the signing authority behind it. A vendor-neutral conformance regime would require more. It would need published criteria for producer admission, a decision process, and evidence requirements that a party outside the present deployment could satisfy; configuration-based admission does not provide those conditions.

7.1 What the signature gives the recipient

Under the stated key and trust model, a valid signature can support the claim that the signed content has not been altered without detection and that the holder of the corresponding signing authority signed that canonical payload.[3] [11] The qualifications are substantive: alteration is rendered detectable rather than prevented, and the signer is identified as the holder of a key rather than as an organisation.

Combined with the payload, that can bind:

- subject;
- producer;
- evidence;
- rubric version;
- result;
- issue time;
- other signed metadata.[3]

A detached score or unsigned export does not provide the same integrity check. With a signed payload, the recipient can verify the artifact against the trusted key material instead of relying on the sender's assertion that the document is genuine.

It means integrity and provenance can travel with the result. The claim survives being forwarded, archived and retrieved, and it survives being read by somebody who was not party to the original exchange, which is the ordinary condition under which assurance evidence is used.

7.2 Where the signature stops

The signature authenticates a specific statement: the canonical payload under a particular key. Its assurance value depends on keeping that object and the surrounding key trust model explicit. Signature verification establishes who, under that trust model, committed to the payload and whether the protected payload has changed; it does not determine whether the statement inside the payload is factually correct. The mathematics signs a false statement as readily as a true one.

That means a valid signature does not establish:

- that the repository is good;
- that the software is secure;
- that the producer measured it correctly;
- that a detector was valid;
- that the rubric is well calibrated;

- that the issuer is competent;
- that the governance process is independent.

The seven limitations occur at different layers: the first two concern the software, the next three the measurement and scoring rule, and the last two the institutions around the system. A valid signature does not move evidence from one layer into another.

The current CAI delivery specification states this limitation directly: verification of the package does not attest that the evidence truthfully measures the real repository. That remains the producer's claim.[3] The boundary is therefore part of the published delivery claim and can be cited directly when interpreting a successful verification.

A signature should therefore be presented as authentication of a payload, not as a generic seal of software quality. The signature records that a particular signing authority committed to a particular document; it does not record an independent examination or approval of the software. Its technical value is the integrity and provenance protection it gives to the claim contained in the payload.

8. Challenge routing by claim type

The CAI challenge model routes disagreement according to the claim being contested.[2] It distinguishes four classes: the arithmetic of a score, a defect in the reference scorer, a measurement within a particular survey, and the rubric itself. The route determines both who can settle the dispute and what evidence the challenger needs to produce. Arithmetic and scorer-implementation challenges use reproduction cases; measurement challenges require evidence about the repository; methodology challenges require an argument about the rule. Localising the dispute prevents a disagreement at one layer from becoming an unbounded argument about confidence in the system as a whole.

8.1 “The result does not follow from the evidence”

This is an arithmetic or reproduction challenge. It is the only one of the four requiring neither permission nor an adjudicator, and the challenge documentation says so directly, describing the question as mechanical and its answer as falsifiable.[2]

The verifier takes the supplied evidence and governing rubric and performs the fold again. Both are published alongside the result, so the operation requires nothing from the issuer beyond what has already been delivered.

If the claimed result does not follow, the discrepancy is mechanical. It can be exhibited as a case anyone else can run, and running it produces the same outcome for the challenger and for the party being challenged.

No opinion about the repository is required.[2] The challenger need not have read the code, need not agree with the rubric, and need not hold any view about whether the software is any good.

The evidence for this class of challenge can be self-contained. A complete dispute of this kind consists of a bundle, a rubric version, the number that was published and the number the fold produces, which is a smaller and more decisive object than an argument.

8.2 “The reference scorer does not implement the published rule”

This is an implementation-conformance problem. The dispute lies not between two parties' judgements but between two artifacts, the written method and the code that claims to implement it.

A minimal evidence bundle should be able to demonstrate a divergence between the reference scorer and the published scoring rule. The challenge documentation asks for that reproduction together with the rubric version, the number obtained and the number implied by the specification.[2] Such a divergence is a scorer-implementation defect: the reference implementation disagrees with the published rule it is meant to implement.[2]

Because the scorer is open and the rule is versioned, this class of defect can be investigated against a concrete reproduction case. A fix that changes a published number is then handled with the same discipline as a rubric change, earlier versions remaining frozen and served, so that correcting the scorer does not silently rewrite results computed before the correction.[2]

8.3 “The evidence mis-measures the repository”

This is a measurement dispute, and the standard is deliberately not the body that settles it.[2] The arithmetic may be completely correct while one of the measurement inputs is wrong. The fold processes the supplied value as given, so it cannot distinguish a well-founded input from an incorrect one that has the same numeric form.

The disputed proposition may concern the measured value:

This dimension should not have been 4.2.

It may instead concern how the criterion was applied:

This criterion was applied to the wrong code.

That dispute belongs with the producer of the measurement.[2] The challenge model routes it to the issuer of the survey, where it is adjudicated by the named human who attested the measurement rather than by the standard.[2]

CAI's role is to make the disputed evidence and governing rule visible enough that the disagreement can be specific. It does this by constraining what a producer may emit: a dimension measured at confidence zero must be recorded as absent rather than as a score of zero, so that “not assessed” can never be read as “failed”; advisory dimensions judged by a language model are band-only and never touch the deterministic number; and a band capped by the critical gate must name the contributor that capped it rather than display an anonymous flag.[2] Each rule exists so that a challenger has something particular to point at.

The open scorer cannot retrospectively observe the repository and adjudicate the producer's measurement merely by folding the same evidence again. Re-running a computation over an input tells nobody where the input came from.

8.4 “The dimension, weighting or formula is wrong”

This is a methodology dispute. The challenger accepts the arithmetic, accepts the measurement, and objects to the rule under which they were combined.

The implementation may be correct and the measurement may be accurate while a reviewer still disagrees with the rule itself. Such a disagreement is not an error report and cannot be resolved by producing a reproduction case, since both parties would obtain the same number and continue to disagree about what it ought to have been.

That disagreement belongs in the evolution of the rubric or standard. The challenge documentation treats it as a first-class contribution rather than as a complaint to be deflected, which is the appropriate posture for a standard whose rules are published in order to be argued with.[2]

Because rubrics are versioned, an accepted methodology change should create a new rule identity rather than silently modifying the meaning of historical results.[2][5] Every earlier version remains published and served, so a result computed long before stays reproducible against the definitions under which it was actually computed.[2]

Versioning lets CAI change the methodology without rewriting the meaning of previously issued results. An accepted rule change receives a new identity, while earlier results remain interpretable and reproducible under the rules that governed them.

8.5 The current governance boundary

The current challenge documentation says that scorer and rubric challenges are decided by the standard's maintainers through the public repository.[2] Of the four paths, that is two of them: the arithmetic path is settled by computation and the measurement path by the issuer, while both paths capable of changing the rule terminate with the maintainers.

The current process is documented in a public venue, and the reasoning attached to a decision is visible. That provides a traceable dispute path, while the authority to decide scorer and rubric challenges still remains with the maintainers.

It should not be described as evidence that CAI has already acquired independent governance. The maintainers deciding in the open are the parties who wrote the rule, and a process in which the authors adjudicate challenges to

their own method is transparent without being independent.

Open adjudication and independent stewardship are different maturity properties. The first concerns whether a decision can be observed, the second who is entitled to make it, and the two can be improved separately.

The present evidence therefore supports a claim of transparent maintainer adjudication, not independent stewardship. A future governance body would require a separate authority structure and cannot be inferred from the current public process.

8.6 Open scoring and producer conformance

The scoring method and the producer ecosystem have separate maturity states. The rubric, scoring semantics, reference scorer and verification method are publicly inspectable today, while the current producer path remains closed-loop and neutral third-party producer conformance is not yet defined.[1][4][13] Describing the scoring method as open therefore refers to the rules and reference implementation; it does not imply that external producers can already obtain neutral conformance status. Conversely, the current closed producer path does not change the public status of the scoring rule.

9. What a neutral verification surface should expose

A verification surface should state the trust model and the status of each check explicitly rather than leaving the recipient to infer them from a JSON schema. A schema defines fields and shapes; it does not by itself explain which propositions were tested or which remain outside the verifier.

The surface should therefore report, layer by layer, what was checked and what result each check returned. At minimum, a CAI verification surface should expose or make retrievable:

Verification element	Question it answers
Subject identity	What software does this claim describe?
Revision or point-in-time identity	Which state of that software is in scope?
Producer identity	Who performed the measurement?
Issuer identity and role	Who signed this artifact, and in what capacity?
Evidence	What inputs were supplied to the scoring process?
Applicability and completeness information where available	What measurement surface actually resolved?
Rubric version	Which score-moving rules govern the result?
Claimed result	What number and band were published?
Recomputed result	Does the claimed result follow from the supplied evidence and rubric?
Provenance and integrity information	What artifact or signing path binds the claim together?
Verification limitations	Which propositions have not been independently verified?

The current CAI delivery model already contains much of this information.[3] Subject, revision, producer, rubric version, evidence and claimed result are carried inside the signed payload, and the provenance and integrity row is answered by the signature and key set described in section 7. What the delivery format supplies is data, whereas what the table asks for is that data organised by the question each element answers, and the second does not follow automatically from the first.

The newer signed `surveyFit` information also allows a delivery to carry descriptive survey-completeness information without feeding that completeness value back into the CAI score itself.[12] It reports two counts rather than a single figure: how many of the domain and architecture lenses applied to this codebase at all, and how many of those actually resolved and returned a result.[12] The gap between the two is the survey's blind spots. They are kept separate because they fail for opposite reasons, since a rich codebase the surveying implementation could not resolve and a fully supported codebase with little structure to find both produce a thin reading, and a reader needs to know which of the two occurred.[12] The producer supplies these numbers, and CAI neither computes nor validates them beyond their shape.[12]

Completeness describes the survey, while the score describes what the resolved evidence says about the codebase. Folding completeness into the score would mix a property of the measurement process with the result derived from

the evidence that did resolve.

A thin measurement is not automatically evidence of a bad codebase. A lens that did not apply is not counted in the denominator at all, so an inapplicable question is never held against the repository, and a lens that applied without resolving is a fact about the survey rather than about the code.[12]

But a recipient should still be able to see that the measurement was thin. The specification puts the reason plainly: a score from a thin survey and a score from a complete one are different claims, and without this information the two look identical.[12]

The reporting layer therefore needs enough information to distinguish the strength of the evidence base from the quality result produced from the evidence that did resolve. These are two independent axes, and a report collapsing them leaves the recipient unable to tell a confident result from a provisional one.

Keeping the evidential context visible lets the recipient distinguish the quality result from the strength and completeness of the evidence base, and decide how much weight to place on the number.

9.1 The word “verified” needs an object

A verification interface should report the status of specific checks:

Score recomputation: passed.
Signature verification: passed.
Subject binding: present.
Measurement validity: not independently tested by this verifier.

Those statements carry more information than the single verdict:

Verified.

The longer wording identifies which claim survived which test and records checks that were not attempted instead of collapsing them into an overall status. A layered report also preserves distinct failure states that a single verdict would collapse. For example, a package can have:

AUTHENTIC ARTIFACT, ARITHMETIC FAILS

signature	PASS
score recomputation	FAIL

That tells us something very different from:

ARITHMETIC HOLDS, SIGNATURE FAILS

signature	FAIL
score recomputation	PASS

The first case says the authenticated artifact contains a result that does not follow from its evidence. The claim is properly attributed and internally wrong, which directs attention to the issuer's process rather than to the artifact's

handling.

The second says the arithmetic may be internally consistent, but the artifact's claimed signing path cannot be established. That directs attention the other way, to transmission, key resolution or tampering, and sound arithmetic is no comfort when the document's origin cannot be established.

The strength of a layered verification surface is that these states stay distinguishable. Each names a different party who would have to act, which is the practical test of whether a verification result is of any use.

A single green or red badge throws away information the system is already capable of providing. The loss is gratuitous, since the underlying checks were performed separately and only the presentation combined them.

10. Make the trust boundary visible in the report

A CAI verification report should make the trust boundary visible without forcing the recipient to reconstruct it from implementation details. The report should show which claims were tested, which mechanism performed each test, and which claims remain outside that mechanism. That level of precision does not require a more complicated verdict; it requires the report to preserve distinctions the underlying system already makes.

A recipient of a CAI artifact should be able to identify the repository and revision the result claims to describe, the producer of the measurement, the rubric governing the scoring semantics, and the evidence supplied to the fold. The same report should state whether the score was recomputed successfully, whether the signed artifact verified under the stated key and trust model, and which parts of the underlying measurement remain the producer's evidential claim. These statements describe different objects, so they can be combined without turning them into one generic assertion that “the CAI is verified”.

Each statement needs an object, a mechanism and a failure condition. The object identifies what is being asserted; the mechanism identifies how it was checked; and the failure condition identifies what observation would withdraw the claim. A report that preserves those elements lets a recipient see whether a disagreement concerns measurement, arithmetic, subject identity, provenance or integrity, and directs the dispute to the layer where evidence can actually settle it.

The model does not eliminate trust. It places trust at identifiable points in the chain and makes some of those points directly testable. Strong cryptography can protect the integrity and attribution of a defined payload, but it cannot supply meaning that the payload does not contain. Deterministic recomputation can establish the evidence-to-result relation, but it cannot extend backwards into a repository measurement it never performed. Those boundaries belong in the report because they determine what a successful check licenses the recipient to say.

The current CAI architecture already exposes the information needed for such a report. The reference scorer makes the evidence-to-score relationship directly testable by a recipient.[1][2][13] The versioned rubric identifies the scoring semantics that govern the result.[5] The Draft delivery format binds subject, producer, evidence, result and provenance into a signed artifact while stating the limit of that signature.[3] The current registry provides a working closed-loop distribution and verification path while third-party producer conformance remains deferred.[4]

These properties should be reported at the strength supported by the evidence. A recipient who tests an overstated claim and finds the mechanism cannot support it has little reason to trust the accurate claims standing beside it. The safer technical practice is to name the exact property that was established and leave adjacent claims to the mechanisms that actually address them.

The next maturity step is greater independence in verification. Canonical conformance vectors would let implementations test their interpretation of the signing representation; independently implemented verifiers would show whether the specification can be reproduced outside the reference code; explicit third-party producer requirements would define how an external producer can enter the trust model; and independent governance would eventually need authority over those requirements without treating the founding implementation as the standard itself. Those steps extend the evidence for the architecture rather than changing what the present mechanisms do.

The current claim can therefore be stated directly:

A CAI result can make its arithmetic, rule identity, provenance and trust boundaries inspectable while keeping repository measurement validity as a separate evidential question.

Verification matrix

The sections above treat each layer in turn. The matrix below states the same relations in a single view, arranged around the question that has governed the paper throughout: for each property, which mechanism establishes it, what a successful check therefore licenses a reader to say, and what that same check leaves open.

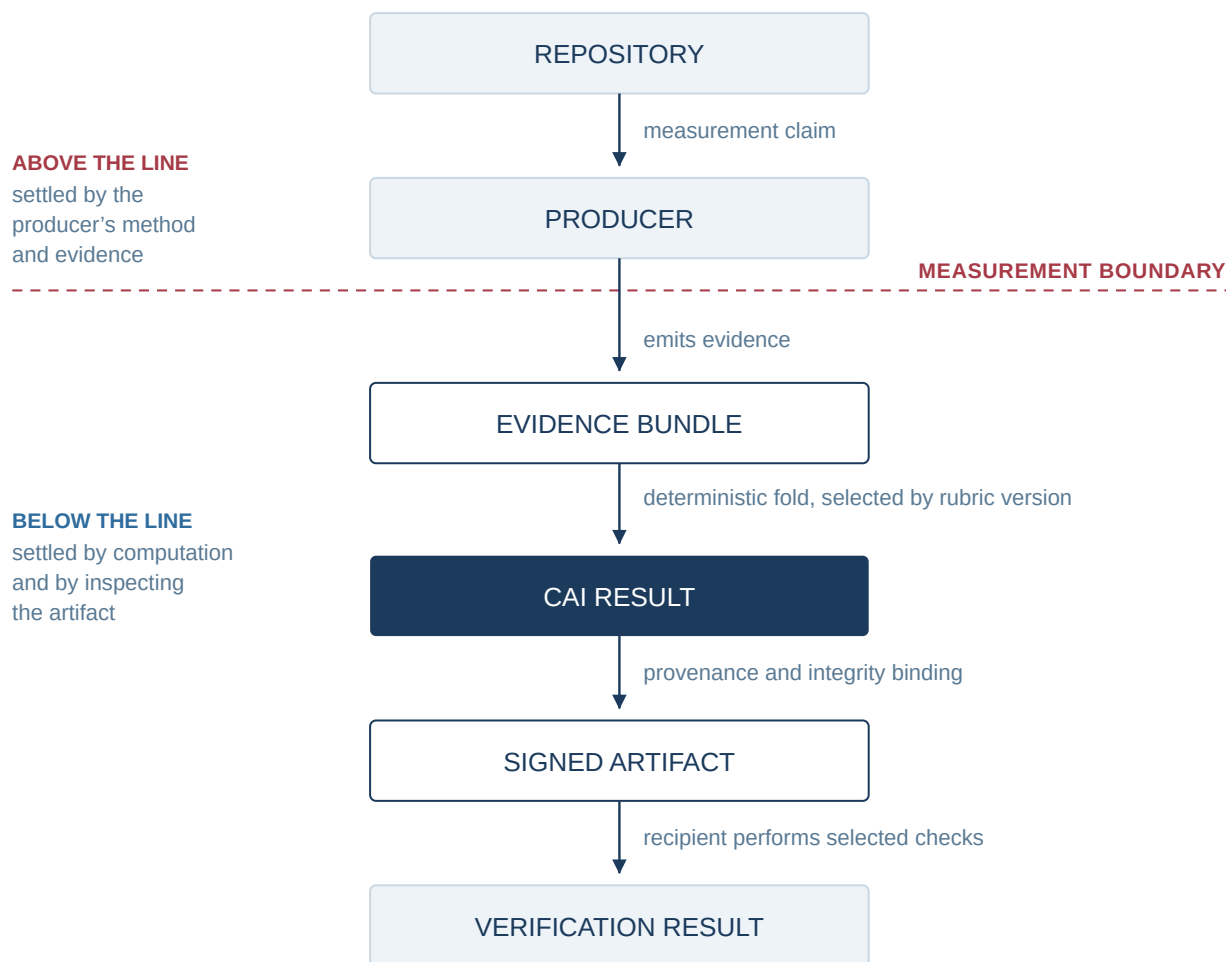
Property	Current verification mechanism	What a successful check establishes	What it does not establish
Arithmetic	Reference scorer / independent implementation using evidence and rubric	Claimed result follows from supplied inputs and scoring rules	Evidence correctly represents repository
Rubric identity	Versioned rubric and archived scoring configuration	Which score-moving rule set governs the result	Rule set is empirically valid
Subject binding	Signed package subject fields	Artifact claims to concern the identified subject	Producer measured that subject correctly
Artifact integrity	Signature over canonical payload	Signed content has not changed without detection	Content is factually correct
Issuer authenticity	Signature verification plus trusted key binding	Signing authority corresponding to trusted key signed the payload	Signer is competent or independent
Provenance	Producer, subject, issue time, rubric and evidence carried together	Origin and production context are inspectable	Upstream process was valid
Measurement validity	Producer evidence, benchmarks, review and separate validation	Depends on the validation performed	Cannot be inferred from recomputation or signing alone
Governance independence	Governance process and independent participation	Depends on actual governance structure	Cannot be inferred from open source code or public issue tracking

The final column records scope, not a catalogue of deficiencies: each entry names a proposition for which another mechanism or layer is responsible. Measurement validity and governance independence also differ from the rows above them because neither has a single mechanical check that settles the question; the table leaves that absence visible instead of inventing a decisive-looking mechanism.

Figure 1. Trust-boundary model

The diagram traces one result from the repository it describes through to the verification a recipient performs upon it. Each stage is an operation carried out by a different party, on a different object, and warranting a different claim.

TRUST-BOUNDARY MODEL



Each arrow creates a different claim. The arrow rather than the box is where the assurance question lives, since every transition is a point at which something could be done incorrectly and at which a distinct kind of evidence would be required to show that it was not.

Re-running a later arrow does not automatically validate every arrow before it. Recomputing the fold establishes the third transition and says nothing whatever about the first, which is this paper's central boundary expressed as a picture.

Each transition can therefore be inspected and improved independently. A weakness at one transition can be addressed without redesigning the others, and a strengthening at one does not have to wait for changes elsewhere in the chain.

Figure 2. Two independent failure examples

The two cases below are illustrative rather than measured. Neither is drawn from a real survey; both are constructed to show what a layered verification surface reports when one check succeeds and another fails.

Example A: authentic artifact, wrong arithmetic

EXAMPLE A

Signature verifies	PASS
Subject / producer binding	PASS
Embedded evidence is intact	PASS
Claimed headline	78 . 2
Recomputed headline	74 . 6
Arithmetic verification	FAIL

The value of the layered model is that authenticity does not hide an arithmetic mismatch. A single combined verdict would have to choose between reporting the artifact as sound, which would conceal the discrepancy, and reporting it as unsound, which would misdescribe a signature that verified correctly.

The verifier can identify the failed layer. The recipient learns that the artifact is genuinely from its issuer and that its published headline does not follow from its own evidence, which directs attention to the issuer's scoring process and to nothing else.

Example B: reproducible result, contested measurement

EXAMPLE B

Signature verifies	PASS
Headline reproduces	PASS
Evidence D17	3 . 8
Repository owner disputes D17	OPEN

The score can reproduce perfectly while the underlying measurement remains disputed. The two facts are independent: the reproduction is not weakened by the dispute, and the dispute is not settled by the reproduction.

The dispute belongs upstream, at the measurement layer. It follows the third challenge path set out in section 8.3, where it is answered by the issuer of the survey rather than by the standard.

The verification model therefore keeps the disagreement at the measurement layer instead of treating the whole artifact as either trusted or untrusted. A model offering only those two states would force the recipient either to discard a sound arithmetic result over an unresolved question about one input, or to accept the disputed input because the arithmetic was sound.

Limitations and maturity boundary

This paper describes the current CAI architecture and the verification claims supported by its present public technical surfaces. It is therefore dated in a specific sense, the architecture being under development and its technical reference state recorded in the header, so a reader consulting the paper later should check these claims against the surfaces as they then stand.

Figures that move

These are the paper's current-state figures, each as at the reference state named above. A later reader should recheck any of them against that commit rather than assume it has held.

Figure	At the reference state	Settled by
Published rubric catalogs	38, none carrying a scoring block	the published catalogs[13]
Lenses in the catalog	10	the published catalogs[13]
Delivery verifier headline tolerance	± 0.5	the delivery specification and verifier[3][14]
Score precision stored in the package	two decimal places	the delivery specification[3]
Delivery format version	1.0, Draft	the delivery specification[3]
Signature and canonicalisation	Ed25519 over RFC8785 - json	the delivery specification[3]
Registry topology	v1, one trusted producer, third-party conformance deferred	the registry contract[4]
Challenge paths	four	the challenge document[2]

The current architecture already supports meaningful assurance properties:

- deterministic recomputation from supplied score-bearing evidence under a named rubric;
- versioned score semantics;
- subject and producer provenance in the delivery artifact;
- cryptographic integrity under the current trust model;
- explicit challenge paths for arithmetic, implementation, measurement and methodology disputes;
- signed descriptive survey-completeness information through `surveyFit`.

Each of those has been demonstrated in the sections above against a published specification, a public implementation, or both, and none of them depends upon a study that has not been run.

Several stronger claims remain future maturity work. They are set out explicitly because a reader who has followed the argument this far might reasonably extrapolate from what the architecture does to what it has been shown to achieve, and the two are not the same thing.

This paper does not establish that:

- CAI's dimensions or lenses are externally validated measures of software quality;
- the current weighting and aggregation choices are empirically optimal;
- a third-party producer can currently obtain neutral CAI conformance status;
- the current registry constitutes independent governance;
- the existing closed-loop canonicalisation path establishes complete third-party RFC 8785 interoperability;
- the fold's constants and band cutlines are pinned in the published rubric catalogs rather than resolved from the scorer's defaults;
- the delivery format's forward-compatibility rule has been demonstrated against a package a verifier does not fully implement;
- signature verification establishes truthfulness of the underlying repository measurement;
- deterministic recomputation establishes outcome validity or predictive validity.

Those are different claims and require different evidence. They are also of two kinds, which the list does not distinguish. The first seven are contingent: a corpus study, an independent implementation, a conformance regime or a governing body would settle them, and each may in time be settled. The last two are not contingent at all but permanent boundaries between categories, since no quantity of future work will make a signature capable of establishing that a measurement was true, or a recomputation capable of establishing that a score predicts anything.

The most immediate technical strengthening opportunities are versioned conformance vectors, independent verification implementations and an explicit future producer-conformance model. They are listed in ascending order of cost rather than of importance, and the first is inexpensive because much of the necessary fixture material already exists in the reference scorer's published tests.[13]

Broader measurement validity belongs to the separate CAI research programme rather than being inferred from successful recomputation. The boundary is deliberate, since a paper concerned with what verification establishes should not also be the place in which measurement validity is argued.

These maturity steps extend the evidence for the current verification architecture. They are not prerequisites for describing the assurance properties already supported by the published specifications and implementation, but they are prerequisites for making stronger claims about independent interoperability, producer conformance or governance.

Conclusion

CAI verification depends on making the object of each check explicit. Repository measurement, deterministic scoring, provenance, artifact integrity and key trust are related parts of one evidence chain, but they are not interchangeable claims. The repository-to-evidence step is assessed through the producer's method and evidence; the evidence-to-score step can be recomputed from the supplied inputs and rubric; the delivery artifact binds subject, producer, rules and result; and the signature protects that bound payload under a stated key and trust model.

This separation gives a verifier a defined job. It can report which checks it performed and the outcome of each one without converting a partial verification into a claim about the entire chain. A successful recomputation establishes that the result follows from the supplied evidence. A successful signature check establishes integrity and attribution under the applicable trust model. Neither operation independently establishes that the producer measured the repository correctly, and a measurement dispute remains a measurement dispute even when the arithmetic and signature both pass.

The same separation makes disagreement tractable. A recipient can identify whether the contested point lies in a measurement, the scorer implementation, the rubric, the subject binding or the integrity of the delivered artifact. Each of those questions calls for different evidence and, in some cases, a different decision-maker. The unresolved layer therefore remains visible instead of being hidden inside one overall verdict, and the recipient can direct the dispute to the evidence relevant to that layer.

Recomputation is the central mechanical property in that model. Once eligible evidence and a rubric version are fixed, the published result can be derived again by a recipient without privileged access to the issuer. Bound provenance then keeps that derivation attached to the subject, rule identity and origin it claims to describe. Cryptographic integrity protects the resulting artifact against undetected alteration under the stated trust model, while the measurement boundary records where those mechanisms stop.

The resulting CAI artifact is an inspectable chain of evidence, rules, provenance and verification steps rather than a detached score. A recipient can examine each link on its own terms and can state what has been established without extending that claim into another layer. Trust in the producer's measurement remains where the architecture cannot replace it with an independent test; where a direct test exists, the artifact should expose enough information for the recipient to perform it.

References

1. **CanineCC/CAI**, *README.md*. Current public description of the CAI scoring model, reference scorer, evidence bundle and reproducibility claim. Repository state checked 23 August 2026.
2. **CanineCC/CAI**, *Challenge & review: how to dispute a CAI*. Current challenge model separating arithmetic, reference-scorer, issuer-measurement and methodology disputes. Repository state checked 23 August 2026.
3. **CanineCC/CAI**, *CAI-delivery package: format specification*, version 1.0, Draft. Primary current source for payload structure, subject and producer binding, Ed25519 signing, RFC 8785 canonicalisation target, offline verification, headline tolerance and the explicit limit of the signature's attestation.
4. **CanineCC/CAI**, *CAI registry: API contract*, implemented v1 closed-loop. Primary current source for the one-producer registry topology, verification-on-ingest, trusted-key model, immutable delivery storage and deferred third-party producer conformance.
5. **CanineCC/CAI**, commit `1d557c36faa6f7650cfe6746db01665ad2ac9bfc`, 22 August 2026, *the catalog pins the fold's constants and the cutlines, so a rubric selects its own semantics*. Primary pinned source for the catalog scoring-parameter mechanism and default resolution.
6. Canine Development, *How the Codebase Assurance Index is constructed*. Informative founding paper describing the separation between evidence production and deterministic aggregation and the CAI scoring architecture.
7. **ISO/IEC/IEEE 15939:2017**, *Systems and software engineering: Measurement process*. Current published edition as of August 2026. Used as general methodological context for separating measurement definition, analysis and validity. It is not evidence that CAI is ISO-conformant or ISO-validated.
8. **in-toto**, *Statement Layer Specification v1*. Used as an external precedent for binding attestation metadata to an identified software subject.
9. **SLSA v1.2**, *Provenance*. Used as an external software-supply-chain precedent for describing verifiable information concerning where, when and how an artifact was produced. CAI does not claim SLSA conformance through this comparison.
10. Rundgren, A., Jordan, B., and Erdtman, S., **RFC 8785**, *JSON Canonicalization Scheme (JCS)*, 2020. Used for the role of invariant JSON representation in repeatable hashing and signing.
11. Josefsson, S., and Liusvaara, I., **RFC 8032**, *Edwards-Curve Digital Signature Algorithm (EdDSA)*, 2017. Used for the technical definition of EdDSA and Ed25519, not as evidence for the sufficiency of CAI's key-management model.
12. **CanineCC/CAI**, commit `fc616ce3bf981194e0e351547a234aa6b22b3645`, 22 August 2026. Primary pinned source for signed, descriptive, non-scored `surveyFit` completeness information.
13. **CanineCC/CAI**, current scoring implementation, published rubric catalogs and conformance tests at commit `a24b47b2eb46971dcd530f3a705ebb419aea158c`, especially `src/Cai.Scoring/CaiScorer.cs`, `src/Cai.Scoring/ScoringParameters.cs`, `src/Cai.Scoring/RubricCatalog.cs`, `docs/adr/0004-versioned-frozen-rubrics.md`, and `tests/Cai.Tests/ScoringParametersTests.cs`. Direct implementation evidence that the catalog's scoring block governs the fold where present, that catalogs publishing no block resolve to the scorer's defaults, and that no catalog published to date carries one.
14. **CanineCC/CAI**, *CAI-delivery reference implementation* at commit `a24b47b2eb46971dcd530f3a705ebb419aea158c`, especially `src/Cai.Delivery/DeliveryPackage.cs`,

`src/Cai.Delivery/DeliverySigning.cs` and `src/Cai.Delivery/CanonicalJson.cs`. Direct implementation evidence for the verification sequence and for the typed parse and canonicalisation path.