

RATS
Internet-Draft
Intended status: Informational
Expires: 13 September 2026

B. Tsyrlunikov
Cyntrisee
12 March 2026

Attested Inference Receipt (AIR): A COSE/CWT Profile for Confidential AI
Inference
draft-tsyrlunikov-rats-attested-inference-receipt-01

Abstract

This document defines the Attested Inference Receipt (AIR), an application-layer COSE_Sign1 envelope carrying CWT claims profiled per the Entity Attestation Token (EAT) framework. An AIR receipt binds model identity, input/output hashes, attestation-linked metadata, and operational telemetry into a single signed artifact suitable for independent third-party verification of a confidential AI inference event.

AIR v1 targets single-inference receipts emitted by workloads running inside hardware-isolated Trusted Execution Environments (TEEs). AIR is attestation-linked: it carries measurements and a hash reference to the platform attestation evidence associated with the inference, but it does not replace platform-specific attestation verification. This version defines AWS Nitro Enclaves and Intel TDX measurement profiles only. Pipeline chaining, multi-inference receipts, and extensibility mechanisms for additional claim or platform profiles are out of scope.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 13 September 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	4
1.1. Goals	4
1.2. Non-Goals	5
2. Requirements Language	5
3. Terminology	5
4. AIR v1 Receipt Format	6
4.1. COSE_Sign1 Envelope	6
4.2. Protected Header	7
4.3. Unprotected Header	7
4.4. Payload: CWT Claims Map	7
4.5. CDDL Schema	7
5. Claim Semantics	9
5.1. Standard CWT/EAT Claims	9
5.1.1. iss (Issuer) -- key 1	9
5.1.2. iat (Issued At) -- key 6	9
5.1.3. cti (CWT ID) -- key 7	9
5.1.4. eat_profile -- key 265	9
5.1.5. eat_nonce -- key 10	10
5.2. AIR Private Claims	10
5.2.1. model_id -- key -65537	10
5.2.2. model_version -- key -65538	10
5.2.3. model_hash -- key -65539	10
5.2.4. request_hash -- key -65540	10
5.2.5. response_hash -- key -65541	10
5.2.6. attestation_doc_hash -- key -65542	11
5.2.7. enclave_measurements -- key -65543	11
5.2.8. policy_version -- key -65544	12
5.2.9. sequence_number -- key -65545	12
5.2.10. execution_time_ms -- key -65546	12
5.2.11. memory_peak_mb -- key -65547	12
5.2.12. security_mode -- key -65548	12
5.2.13. model_hash_scheme -- key -65549	12

6.	EAT Profile Declaration	13
7.	Verification Procedure	14
7.1.	Layer 1: Parse	15
7.2.	Layer 2: Cryptographic Verification (AIR-local)	15
7.3.	Layer 3: Claim Validation	15
7.4.	Layer 4: Policy Evaluation	16
8.	Relationship to Other Work	16
8.1.	draft-messous-eat-ai	17
8.2.	SCITT	17
8.3.	RATS Architecture	17
9.	Security Considerations	18
9.1.	Receipt Integrity	18
9.2.	Algorithm Pinning	18
9.3.	Replay Protection	18
9.4.	Model Hash Limitations	18
9.5.	Attestation Document Not Verified by Receipt	19
9.6.	Workload Honesty and Evidence Scope	19
9.7.	Signing Key Binding	19
9.8.	TEE Compromise	20
9.9.	Clock Integrity	20
9.10.	Deterministic Encoding	20
9.11.	Closed Claims Map	20
10.	Privacy Considerations	20
10.1.	Input/Output Hashes	20
10.2.	Correlation Metadata	21
10.3.	Nonce Privacy	21
11.	IANA Considerations	21
12.	Implementation Status	21
12.1.	Reference Implementation (Rust)	21
12.2.	Python Interop Verifier	22
12.3.	E2E Validation	22
13.	Examples	22
13.1.	Valid Receipt Walkthrough	22
13.2.	Invalid Receipt Categories	23
14.	References	24
14.1.	Normative References	24
14.2.	Informative References	25
	Appendix A. Full CDDL Schema	26
	Appendix B. Golden Vector Summary	27
	Acknowledgments	28
	Author's Address	28

1. Introduction

Deployments that run machine learning models on cloud infrastructure lack a standardized, interoperable mechanism to prove what happened during a specific inference. Existing attestation frameworks such as RATS [RFC9334] establish platform identity and code integrity, but they do not produce per-inference evidence binding a model, its inputs and outputs, and the platform state into a single verifiable artifact.

Platform attestation and ordinary logs are both necessary but insufficient for this purpose. Platform attestation proves properties of the workload and its execution environment, typically at a point in time, while ordinary logs are implementation-specific and often unsigned. Neither provides a standard, portable object that binds one inference event to model identity, request/response hashes, and attestation-linked metadata in a form a third party can verify independently of the underlying platform.

The Attested Inference Receipt (AIR) fills this gap. An AIR receipt is an application-layer COSE_Sign1 [RFC9052] envelope whose payload is a CWT [RFC8392] claims set profiled as an EAT [RFC9711]. The receipt is signed with Ed25519 [RFC8032] by the workload running inside a Trusted Execution Environment (TEE). AIR verification is split into two concerns: AIR-local verification of the signed receipt itself, and platform-specific verification of the underlying attestation evidence and key binding. This document standardizes the former and references the latter, but does not replace platform-specific attestation procedures.

AIR v1 is scoped to a single inference: one request processed by one model inside one attested workload produces one receipt. Pipeline chaining, multi-stage proofs, and integration with transparency logs (such as SCITT [SCITT]) are deferred to future versions.

1.1. Goals

The goals of AIR v1 are:

1. Define a receipt wire format using existing IETF standards (COSE_Sign1, CWT, EAT).
2. Bind model identity (cryptographic hash), input/output hashes, attestation metadata, and operational telemetry in a single signed envelope.

3. Support AIR-local verification using standard COSE/CWT tooling, while allowing deployments to combine AIR with separate platform-specific attestation verification as needed.
4. Carry platform measurements in a portable receipt shape while preserving their platform-specific semantics via `measurement_type`.
5. Define an intentionally closed v1 profile with fail-closed parsing semantics.

1.2. Non-Goals

AIR v1 explicitly does not:

- * Define a transport protocol or session management scheme.
- * Specify attestation document verification procedures (these are platform-specific).
- * Define an extension registry or compatibility mechanism for new claim or platform profiles.
- * Prove data deletion or model correctness.
- * Provide regulatory certification or compliance guarantees.
- * Define pipeline chaining or multi-inference receipts.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Terminology

Attested Inference Receipt (AIR): An application-layer COSE_Sign1 signed CWT/EAT artifact emitted by a workload after processing a single AI inference request inside a TEE. The receipt binds model identity, input/output hashes, attestation-linked metadata, and operational telemetry.

Confidential Workload: The software executing inside a TEE that

loads a model, processes inference requests, and generates AIR receipts. In RATS [RFC9334] terminology, the confidential workload acts as the Attester.

Verifier: An entity that validates an AIR receipt's signature, claim values, and policy constraints. AIR-local verification can be performed with standard COSE/CWT tooling, but full TEE assurance additionally requires platform-specific verification of the underlying attestation evidence and signing-key binding. In RATS [RFC9334] terminology, this maps to the Verifier role.

Relying Party: An entity that consumes the verification result to make trust decisions (e.g., an auditor, compliance officer, or end user). In RATS [RFC9334] terminology, this maps to the Relying Party role.

Endorser: A party that provides endorsements or reference values used to appraise the attested platform measurements. In typical AIR deployments, this role is fulfilled by the TEE platform vendor or attestation service.

Measurement Map: The `enclave_measurements` claim containing platform-specific register values carried in portability-oriented slots. The `measurement_type` field determines the actual semantics of those slots on each platform.

Receipt: In this document, "receipt" always refers to an AIR receipt. Note that this differs from the SCITT usage of "receipt" (which refers to a countersigned statement from a transparency service). The two are complementary: a future version could register an AIR receipt with a SCITT transparency service and receive a SCITT receipt in return.

4. AIR v1 Receipt Format

4.1. COSE_Sign1 Envelope

An AIR v1 receipt is a tagged `COSE_Sign1` structure (CBOR tag 18) as defined in [RFC9052] Section 4.2:

```
COSE_Sign1 = [  
  protected   : bstr,           ; serialized protected header  
  unprotected : map,           ; unprotected header map  
  payload     : bstr,           ; serialized CWT claims map  
  signature   : bstr .size 64   ; Ed25519 signature  
]
```

The signature covers `Sig_structure1 = ["Signature1", protected, external_aad, payload]` where `external_aad` is empty (`h''`).

Verifiers MUST reject untagged `COSE_Sign1` structures. The CBOR tag 18 is mandatory.

4.2. Protected Header

The protected header is a CBOR map containing exactly two entries:

Label	Name	Value	Description
1	alg	-8	EdDSA (Ed25519)
3	content type	61	application/cwt

Table 1

Verifiers MUST reject receipts where `alg` is not -8 or where `content type` is not 61. Additional protected header parameters are not defined in v1 and MUST NOT be present.

The signing algorithm is Ed25519 with strict verification semantics per [RFC8032] Section 5.1.7. Verifiers MUST reject non-canonical `S` values ($S \geq L$ where `L` is the Ed25519 group order).

4.3. Unprotected Header

The unprotected header MUST be empty for AIR v1 receipts. AIR v1 does not use `kid` or any other unprotected header parameter. Because unprotected header parameters are not covered by the COSE signature, verifiers MUST reject receipts with non-empty unprotected headers.

4.4. Payload: CWT Claims Map

The payload is a CBOR-encoded CWT claims map. The map uses deterministic encoding per [RFC8949] Section 4.2.1, with keys sorted by the bitwise lexicographic order of their deterministic encodings.

The claims map is closed: verifiers MUST reject maps containing unknown integer keys. Duplicate keys MUST be rejected.

4.5. CDDL Schema

The following CDDL [RFC8610] defines the complete wire shape:

```
air-receipt = #6.18([
  protected:  bstr .cbor air-protected-header,
  unprotected: air-unprotected-header,
  payload:     bstr .cbor air-claims,
  signature:   bstr .size 64
])

air-protected-header = {
  1 => -8,           ; alg: EdDSA (Ed25519)
  3 => 61,           ; content type: application/cwt
}

air-unprotected-header = {}

air-claims = {
  ; --- Standard CWT/EAT claims ---
  1  => tstr,           ; iss: issuer
  6  => uint,           ; iat: issued-at (Unix seconds)
  7  => bstr .size 16,   ; cti: CWT ID (UUID v4, 16 bytes)
  265 => "https://spec.cyntrise.com/air/v1", ; eat_profile
  ? 10 => bstr .size (8..64), ; eat_nonce (optional)

  ; --- AIR private claims ---
  -65537 => tstr,           ; model_id
  -65538 => tstr,           ; model_version
  -65539 => sha256-hash,    ; model_hash
  -65540 => sha256-hash,    ; request_hash
  -65541 => sha256-hash,    ; response_hash
  -65542 => sha256-hash,    ; attestation_doc_hash
  -65543 => enclave-measurements, ; enclave_measurements
  -65544 => tstr,           ; policy_version
  -65545 => uint,           ; sequence_number
  -65546 => uint,           ; execution_time_ms
  -65547 => uint,           ; memory_peak_mb
  -65548 => tstr,           ; security_mode
  ? -65549 => tstr,         ; model_hash_scheme (optional)
}

sha256-hash = bstr .size 32
sha384-hash = bstr .size 48

enclave-measurements = nitro-measurements / tdx-measurements

nitro-measurements = {
  "pcr0"      => sha384-hash,
  "pcr1"      => sha384-hash,
  "pcr2"      => sha384-hash,
  ? "pcr8"    => sha384-hash,
```



```
    "measurement_type" => "nitro-pcr",  
  }  
  
  tdx-measurements = {  
    "pcr0"           => sha384-hash,    ; MRTD  
    "pcr1"           => sha384-hash,    ; RTMR0  
    "pcr2"           => sha384-hash,    ; RTMR1  
    "measurement_type" => "tdx-mrtd-rtmr",  
  }
```

The full CDDL is also provided in Appendix A.

5. Claim Semantics

5.1. Standard CWT/EAT Claims

5.1.1. iss (Issuer) -- key 1

A text string identifying the issuing entity (e.g., "cyntrise.com"). The value is operator-assigned and opaque to the receipt format. Verifiers MAY check against an expected issuer allowlist.

5.1.2. iat (Issued At) -- key 6

An unsigned integer representing the Unix timestamp (seconds since epoch) when the inference completed. Verifiers apply a freshness check: $\text{now} - \text{max_age} \leq \text{iat} \leq \text{now} + \text{clock_skew}$. Verifiers SHOULD reject future timestamps. AIR v1 encodes iat as an unsigned integer only, consistent with the time-encoding restriction in [RFC9711] Section 4.3.1. AIR v1 does not use exp or nbf.

5.1.3. cti (CWT ID) -- key 7

A 16-byte binary string containing a UUID v4 encoded as raw bytes (not the 36-character string form). Each receipt MUST have a unique cti. Verifiers maintaining replay state SHOULD track observed cti values.

5.1.4. eat_profile -- key 265

The fixed string value "https://spec.cyntrise.com/air/v1". Verifiers MUST reject receipts with unknown eat_profile values.

5.1.5. eat_nonce -- key 10

An optional binary string (8-64 bytes per [RFC9711] Section 4.1) provided by the client to bind the receipt to a specific request session. If the verifier supplied a nonce, it MUST check that eat_nonce matches. This is the primary replay resistance mechanism when verifier-side cti deduplication is not feasible.

5.2. AIR Private Claims

AIR uses negative integer keys in the CWT private-use range to avoid collision with IANA-registered claims. Keys -65537 through -65548 are assigned and required. Key -65549 is assigned and optional. No other AIR private claim keys are defined in v1.

5.2.1. model_id -- key -65537

A text string containing the human-readable model identifier (e.g., "minilm-l6-v2"). Operator-assigned, opaque. Not cryptographic; use model_hash for binding.

5.2.2. model_version -- key -65538

A text string containing the human-readable model version (e.g., "1.0.0"). Operator-assigned, opaque.

5.2.3. model_hash -- key -65539

A 32-byte SHA-256 [FIPS180-4] hash of the model weights. This is the cryptographic binding between the receipt and a specific model artifact. Verifiers MUST compare against a known-good hash when model identity matters. The model_hash MUST NOT be all zeros.

5.2.4. request_hash -- key -65540

A 32-byte SHA-256 hash of the inference request payload. Binds the receipt to a specific input. Clients holding the original request can recompute and compare. The hash is computed over the raw wire bytes of the request as received by the workload; AIR v1 does not define a canonicalization scheme for request payloads.

5.2.5. response_hash -- key -65541

A 32-byte SHA-256 hash of the inference response payload. Binds the receipt to a specific output. As with request_hash, the hash is computed over the raw wire bytes of the response as emitted by the workload.

5.2.6. attestation_doc_hash -- key -65542

A 32-byte SHA-256 hash of the platform attestation document (e.g., Nitro COSE attestation document, TDX quote). Links the receipt to TEE evidence without embedding the (potentially large) attestation document itself.

Note: AIR v1 does not define attestation document verification. Verifiers that require TEE assurance MUST independently obtain and verify the attestation document, then compare its hash to attestation_doc_hash.

5.2.7. enclave_measurements -- key -65543

A map containing platform-specific measurement registers. The map structure depends on the measurement_type field within it.

5.2.7.1. Nitro PCR Variant (measurement_type = "nitro-pcr")

Field	Type	Required	Description
"pcr0"	bstr 48	Yes	PCR0 (SHA-384)
"pcr1"	bstr 48	Yes	PCR1 (SHA-384)
"pcr2"	bstr 48	Yes	PCR2 (SHA-384)
"pcr8"	bstr 48	No	PCR8 (SHA-384)
"measurement_type"	tstr	Yes	"nitro-pcr"

Table 2

5.2.7.2. TDX MRTD/RTMR Variant (measurement_type = "tdx-mrtd-rtmr")

Field	Type	Required	Description
"pcr0"	bstr 48	Yes	MRTD (SHA-384)
"pcr1"	bstr 48	Yes	RTMR0 (SHA-384)
"pcr2"	bstr 48	Yes	RTMR1 (SHA-384)
"measurement_type"	tstr	Yes	"tdx-mrtd-rtmr"

Table 3

The TDX registers are mapped to pcr0/pcr1/pcr2 portability slots for cross-platform verifier simplicity. The measurement_type field disambiguates the actual register semantics. These slot names do not imply that Nitro PCRs and TDX MRTD/RTMRs are semantically identical.

All pcr0/pcr1/pcr2 values MUST be exactly 48 bytes. Verifiers MUST reject receipts where any required measurement register is the wrong length. The measurement_type MUST be one of the defined values; unknown types MUST be rejected.

5.2.8. policy_version -- key -65544

A text string identifying the version of the policy governing the workload (e.g., "policy-2026.02"). Informational.

5.2.9. sequence_number -- key -65545

An unsigned integer that increases monotonically within a single workload session. Resets on workload restart. Verifiers processing a stream of receipts SHOULD check monotonicity; gaps indicate missed receipts within a session.

5.2.10. execution_time_ms -- key -65546

An unsigned integer representing the wall-clock inference time in milliseconds. Informational; anomalously low or high values may indicate issues but are not a verification failure.

5.2.11. memory_peak_mb -- key -65547

An unsigned integer representing the peak memory usage during inference in megabytes. Informational.

5.2.12. security_mode -- key -65548

A text string identifying the security mode of the workload (e.g., "GatewayOnly", "FullAttestation"). Informational. Verifiers MAY require a specific security mode.

5.2.13. model_hash_scheme -- key -65549

An optional text string declaring how model_hash was computed, enabling verifiers to reproduce the hash from model artifacts.

Defined scheme values:

Scheme	Description
"sha256-single"	SHA-256 of a single model weights file
"sha256-concat"	SHA-256 of deterministically concatenated weight files (lexicographic filename order)
"sha256-manifest"	SHA-256 of a self-describing manifest listing per-file hashes

Table 4

If present, verifiers MUST recognize the scheme value. Unknown schemes MUST be rejected (fail-closed). If absent, verifiers SHOULD treat `model_hash` as opaque (can still compare against a known-good hash, but cannot independently reproduce it).

No additional scheme values are defined in AIR v1. Future revisions MAY define more scheme values. AIR v1 implementations MUST NOT invent new scheme values.

6. EAT Profile Declaration

This section consolidates the mandatory profile positions per [RFC9711] Section 6.3.

- *Profile identifier*:** URI "https://spec.cyntrise.com/air/v1" (carried in `eat_profile`, key 265).
- *Encoding*:** CBOR only ([RFC8949]). JSON serialization is not defined.
- *Envelope*:** COSE_Sign1 ([RFC9052] Section 4.2), CBOR tag 18. Untagged COSE_Sign1 MUST be rejected.
- *Payload content type*:** COSE content_type = 61 (application/cwt). The payload is a CWT claims map.
- *HTTP media type*:** application/eat+cwt ([RFC9782]). Receivers SHOULD accept both application/cwt and application/eat+cwt.
- *Signing algorithm*:** Ed25519 only (COSE alg = -8). `verify_strict` required (canonical S per [RFC8032] Section 5.1.7). No algorithm negotiation in v1.

7. ***Detached bundles***: Not supported in v1. The attestation document is referenced by hash (attestation_doc_hash), not embedded.
8. ***Key identification***: Out of band. The verifier obtains the Ed25519 public key through a platform-specific channel (e.g., attestation document, key registry). AIR v1 does not use kid.
9. ***Mandatory claims***: 16 required claims: iss, iat, cti, eat_profile, model_id, model_version, model_hash, request_hash, response_hash, attestation_doc_hash, enclave_measurements, policy_version, sequence_number, execution_time_ms, memory_peak_mb, security_mode.
10. ***Optional claims***: 2 optional claims: eat_nonce (replay resistance), model_hash_scheme (hash computation method).
11. ***Freshness***: iat carries the execution timestamp (Unix seconds). Verifiers apply max_age + clock_skew policy. eat_nonce provides optional challenge-response replay resistance ([RFC9711] Section 4.1, 8-64 bytes).
12. ***Deterministic encoding***: Required. Map keys sorted per [RFC8949] Section 4.2.1 by the bitwise lexicographic order of their deterministic encodings.
13. ***Closed claims map***: The claims map is closed. Unknown integer keys MUST be rejected. Duplicate keys MUST be rejected.
14. ***Unprotected header***: MUST be empty. All header parameters are carried in the protected header. Receipts with non-empty unprotected headers MUST be rejected.
15. ***Private claim keys***: Keys -65537 through -65549 are assigned in the CWT private-use range ([RFC8392]). No IANA registration is required. AIR v1 defines no extension mechanism or additional private claim keys beyond this set.

7. Verification Procedure

The AIR v1 verification procedure is organized into four layers. These layers define AIR-local verification only. A deployment that requires full TEE assurance MUST additionally obtain and verify the underlying platform attestation evidence and the binding between that evidence and the AIR signing key using platform-specific procedures. If any check in any layer fails, the verifier MUST reject the receipt and SHOULD report the specific failure. Implementations MAY continue executing later layers after a failure to collect a complete set of

diagnostic results, but the overall verification verdict MUST be rejection if any layer fails.

7.1. Layer 1: Parse

1. Decode the input as CBOR. Confirm the outer structure is tagged with CBOR tag 18.
2. Decode the COSE_Sign1 array (4 elements).
3. Confirm the receipt size does not exceed 65,536 bytes.
4. Decode the protected header. Confirm it is a well-formed CBOR map.
5. Confirm alg (label 1) in the protected header is -8 (EdDSA). Reject receipts with any other algorithm.
6. Confirm content type (label 3) in the protected header is 61 (application/cwt).
7. Confirm the unprotected header is empty.
8. Decode the payload. Confirm it is a well-formed CBOR map.
9. Confirm eat_profile (key 265) equals "https://spec.cyntrise.com/air/v1". Reject receipts with unknown profile values.

7.2. Layer 2: Cryptographic Verification (AIR-local)

1. Construct Sig_structure1 = ["Signature1", protected, h'', payload].
2. Verify the Ed25519 signature over Sig_structure1 using the provided public key. The verification MUST use verify_strict semantics (reject non-canonical S values).

7.3. Layer 3: Claim Validation

1. Confirm cti (key 7) is exactly 16 bytes.
2. Confirm iat (key 6) is a non-zero unsigned integer.
3. Confirm model_hash (key -65539) is exactly 32 bytes and not all zeros.

4. Confirm all required text string claims (iss, model_id, model_version, policy_version, security_mode) are non-empty and within reasonable bounds (implementation-defined, RECOMMENDED maximum 1024 bytes each).
5. Confirm enclave_measurements (key -65543) is a map.
6. Confirm measurement_type within enclave_measurements is one of the defined values ("nitro-pcr" or "tdx-mrtd-rtmr").
7. Confirm all pcr0/pcr1/pcr2 values are exactly 48 bytes.
8. If measurement_type is "tdx-mrtd-rtmr", confirm pcr8 is absent. TDX measurement maps MUST NOT contain pcr8.
9. If model_hash_scheme (key -65549) is present, confirm it is one of the defined values ("sha256-single", "sha256-concat", "sha256-manifest"). Unknown values MUST be rejected.
10. Confirm the claims map contains no unknown integer keys and no duplicate keys.

7.4. Layer 4: Policy Evaluation

Policy checks are configurable per verifier deployment. The following checks are defined:

- *FRESH* (timestamp bounds): If configured, verify $\text{now} - \text{max_age} \leq \text{iat} \leq \text{now} + \text{clock_skew}$.
- *NONCE* (challenge binding): If the verifier supplied a nonce, verify eat_nonce matches.
- *MODEL* (expected model): If configured, verify model_hash and/or model_id match expected values.
- *PLATFORM* (expected platform): If configured, verify measurement_type matches expected value.
- *REPLAY* (deduplication): If the verifier maintains a seen-cti store, reject duplicate cti values.

Verifiers SHOULD document which Layer 4 policies they enforce.

8. Relationship to Other Work

8.1. draft-messous-eat-ai

[I-D.messous-eat-ai] defines AI-related claims for EAT, including model identification, training metadata, and performance metrics. AIR v1 is complementary: where draft-messous-eat-ai focuses on per-agent identity, provenance, and authorization metadata, AIR v1 focuses narrowly on per-inference execution evidence from a confidential workload. AIR intentionally binds a specific request/response event to attestation-linked metadata; it is not a general AI agent identity profile. A future version of AIR could adopt registered claim keys from draft-messous-eat-ai once they stabilize, replacing the current private-use integer keys.

8.2. SCITT

The Supply Chain Integrity, Transparency and Trust [SCITT] framework uses "receipt" to mean a countersigned statement from a transparency service. In AIR, "receipt" means a workload-signed inference proof. The two are complementary: an AIR receipt could be registered as a SCITT statement, and the resulting SCITT receipt (countersignature from the transparency service) would provide independent auditability. This document uses "AIR receipt" consistently to avoid ambiguity.

8.3. RATS Architecture

AIR can be incorporated into a RATS [RFC9334] architecture as follows:

- * The confidential workload is the **Attester**.
- * A deployment-specific **Verifier** can evaluate AIR receipts together with platform attestation evidence and applicable endorsements or reference values.
- * The end user, auditor, or compliance officer is the **Relying Party** (they consume verification results).
- * The TEE platform vendor or attestation service providing endorsements or reference values acts as the **Endorser**.

AIR v1 is a workload-emitted signed application-layer artifact, not a verifier-emitted attestation result. AIR does not replace the underlying platform attestation evidence or the verifier procedures needed to evaluate it. Instead, AIR packages per-inference execution claims together with measurements and a hash reference to the platform attestation document. It is therefore distinct from IETF EAR (EAT Attestation Result), which is produced by a verifier after

evaluating platform evidence. In a complete deployment, a verifier may evaluate platform evidence and AIR together, and an EAR might reference an AIR receipt as part of the evidence it considered.

9. Security Considerations

9.1. Receipt Integrity

The Ed25519 signature over the COSE Sig_structured1 protects the protected header and all claims against tampering. The unprotected header is not covered by the signature; AIR v1 requires it to be empty (Section 4.3).

9.2. Algorithm Pinning

AIR v1 pins the signing algorithm to Ed25519 (alg = -8). The algorithm identifier is carried in the protected header and is therefore signed. This prevents algorithm confusion attacks where an attacker substitutes a weaker algorithm.

9.3. Replay Protection

Replay protection in AIR v1 is a shared responsibility:

- * The cti claim provides a unique receipt identifier. Verifiers maintaining state SHOULD track observed cti values and reject duplicates.
- * The eat_nonce claim (optional) provides challenge-response freshness. When present, it binds the receipt to a specific verifier-supplied challenge, preventing replay to other verifiers.
- * The sequence_number claim provides monotonicity within a session. Gaps indicate missed receipts.

Verifiers not maintaining state and not using eat_nonce have limited replay protection (only iat-based freshness). Deployments requiring strong replay resistance MUST use at least one of cti deduplication or eat_nonce.

9.4. Model Hash Limitations

The model_hash claim (SHA-256 of model weights) proves byte-level identity, not model correctness, bias, or safety. Two distinct models with identical hashes are computationally infeasible, but a model with a correct hash may still produce harmful or incorrect outputs.

The `model_hash_scheme` claim (Section 5.2.13) declares how the hash was computed. Unknown scheme values MUST be rejected. This prevents a verifier from accepting a hash computed with an unrecognized method that might weaken integrity guarantees.

9.5. Attestation Document Not Verified by Receipt

The `attestation_doc_hash` claim is a SHA-256 hash of the platform attestation document. AIR v1 does not embed or verify the attestation document. AIR-local verification alone is therefore insufficient to establish TEE assurance. Verifiers requiring such assurance MUST independently obtain and verify the attestation document using platform-specific procedures (e.g., Nitro COSE verification against the AWS root CA, Intel TDX DCAP verification against Intel PCS).

9.6. Workload Honesty and Evidence Scope

An AIR receipt only speaks for what the signing workload observed and emitted. If the workload is malicious, misconfigured, or signs hashes for data that did not come from the claimed inference path, AIR can still produce a syntactically valid receipt. AIR therefore does not protect against an untrusted signer or against semantics outside the measured workload boundary.

Deployments that rely on AIR for end-to-end assurance MUST treat the receipt as meaningful only when the signing key is bound to an attested workload whose measurement set and execution policy are acceptable to the verifier.

9.7. Signing Key Binding

AIR v1 does not define how the Ed25519 signing key relates to the TEE attestation. A verifier MUST NOT infer TEE provenance from the AIR signature alone. A cross-workload substitution or relay attack occurs when a receipt signed by one workload is presented with attestation evidence from a different workload. A deployment that requires end-to-end TEE assurance MUST verify, using a platform-specific binding mechanism, that the AIR signing public key is bound to the attested workload represented by the attestation evidence. AIR v1 does not define these mechanisms because they differ by platform (e.g., Nitro attestation documents carry a `public_key` field, while TDX deployments commonly bind via `REPORTDATA`). Implementations that require end-to-end TEE assurance SHOULD:

1. Generate the Ed25519 key inside the TEE at startup.

2. Include the public key in the platform attestation document (e.g., Nitro `public_key` field, TDX `REPORTDATA`).
3. Provide the attestation document alongside the receipt for end-to-end verification.

9.8. TEE Compromise

AIR v1 assumes the TEE hardware is correct (Trust Assumption TA-1). A hardware vulnerability, firmware bug, or supply chain compromise affecting the TEE breaks all AIR guarantees. AIR v1 does not define revocation mechanisms for compromised platforms.

9.9. Clock Integrity

The `iat` claim depends on the workload's system clock. On AWS Nitro, the enclave uses the host clock (no independent time source). On Intel TDX, the CVM has a TSC but it is subject to frequency scaling. AIR v1 freshness checks are only as accurate as the platform clock.

9.10. Deterministic Encoding

AIR v1 requires deterministic CBOR encoding ([RFC8949] Section 4.2.1). This ensures that the same claims always produce the same payload bytes, preventing signature-valid variants of the same receipt. Implementations **MUST** sort map keys per the CBOR deterministic encoding rules.

9.11. Closed Claims Map

The claims map is closed: unknown integer keys **MUST** be rejected. This prevents downgrade attacks where an attacker adds unrecognized claims that a naive verifier might silently accept as benign. This closed-scope behavior is intentional in AIR v1; future extensions require a revised profile.

10. Privacy Considerations

10.1. Input/Output Hashes

The `request_hash` and `response_hash` claims contain SHA-256 hashes, not plaintext inputs or outputs. However, for low-entropy inputs (e.g., binary classification queries, yes/no questions), an adversary with knowledge of the input space could brute-force the hash to recover the original input. Deployments handling sensitive low-entropy data **SHOULD** consider whether receipt exposure risks input recovery.

10.2. Correlation Metadata

AIR receipts contain timestamps (iat), sequence numbers, and identifiers (cti, iss) that could be used to correlate activity across receipts. In privacy-sensitive deployments, operators SHOULD consider whether the combination of receipt metadata enables unwanted profiling.

10.3. Nonce Privacy

The eat_nonce claim, when present, may leak correlation data if the same nonce is reused across sessions or if the nonce encodes client-identifying information. Verifiers SHOULD use random nonces and avoid embedding client identifiers in nonce values.

11. IANA Considerations

This document has no IANA actions at this time.

AIR v1 uses negative integer keys in the CWT private-use range (keys -65537 through -65549). If AIR gains adoption, a future version may request registration of these claims in the CWT Claims registry established by [RFC8392]. The eat_profile URI ("https://spec.cyntrise.com/air/v1") follows the EAT profile naming conventions in [RFC9711] but is not registered in any IANA registry.

The HTTP media type application/eat+cwt referenced in Section 6 is registered by [RFC9782].

12. Implementation Status

Note to RFC Editor: Please remove this section before publication.

This section records the status of known implementations of the protocol defined by this specification at the time of posting, per [RFC7942].

12.1. Reference Implementation (Rust)

Organization: Cyntrise

Implementation: EphemeralML (common/src/air_receipt.rs, common/src/air_verify.rs)

Description: Full AIR v1 emitter and 4-layer verifier. Generates COSE_Sign1 receipts with deterministic CBOR encoding and Ed25519 signing. Verifier implements all four layers (parse, crypto, claims, policy) with structured error codes.

Maturity: Reference implementation with end-to-end validation on three confidential-computing platforms.

Coverage: Validated against the 10-vector AIR corpus (2 valid, 8 invalid) and exercised in cross-cloud E2E runs.

Contact: borys@cyntirsec.com

12.2. Python Interop Verifier

Organization: Cyntrise (same team, independent implementation)

Implementation: spec/v1/scripts/interop_test.py

Description: Minimal Python verifier using pycose and cbor2 libraries. Validates COSE_Sign1 structure, Ed25519 signature, and claim presence.

Maturity: Test/interop.

12.3. E2E Validation

The reference implementation has been validated end-to-end on three confidential computing platforms:

Platform	Status	Date
AWS Nitro Enclaves (m6i)	PASS	2026-02-28
GCP Confidential Space TDX (c3-standard-4)	PASS	2026-02-27
GCP Confidential Space GPU H100 CC (a3-highgpu-1g)	PASS	2026-02-27

Table 5

13. Examples

13.1. Valid Receipt Walkthrough

The following describes a valid AIR v1 receipt in diagnostic notation. This corresponds to the v1-nitro-no-nonce golden vector.

The COSE_Sign1 envelope (tagged with CBOR tag 18):

```

18([
  h'A2012703183D',      / protected: {1: -8, 3: 61} /
  {},                  / unprotected: empty /
  h'B0...',             / payload: CWT claims map /
  h'<64 bytes>',        / signature: Ed25519 /
])

```

The protected header decodes to:

```

{
  1: -8,      / alg: EdDSA /
  3: 61      / content type: application/cwt /
}

```

The payload (CWT claims map) includes the 16 required claims:

```

{
  1: "cyntriseccom",      / iss /
  6: 1740000000,          / iat /
  7: h'<16 bytes UUID v4>', / cti /
  265: "https://spec.cyntriseccom/air/v1", / eat_profile /
  -65537: "minilm-16-v2", / model_id /
  -65538: "1.0.0",        / model_version /
  -65539: h'<32 bytes SHA-256>', / model_hash /
  -65540: h'<32 bytes SHA-256>', / request_hash /
  -65541: h'<32 bytes SHA-256>', / response_hash /
  -65542: h'<32 bytes SHA-256>', / attestation_doc_hash /
  -65543: {                / enclave_measurements /
    "pcr0": h'<48 bytes SHA-384>',
    "pcr1": h'<48 bytes SHA-384>',
    "pcr2": h'<48 bytes SHA-384>',
    "measurement_type": "nitro-pcr"
  },
  -65544: "policy-2026.02", / policy_version /
  -65545: 1,                / sequence_number /
  -65546: 77,               / execution_time_ms /
  -65547: 0,                / memory_peak_mb /
  -65548: "FullAttestation" / security_mode /
}

```

Verification with the corresponding Ed25519 public key succeeds through all four layers.

13.2. Invalid Receipt Categories

The specification includes 8 invalid golden vectors covering failure modes across all verification layers:

Vector	Layer	Expected Failure
wrong-key	L2	SIG_FAILED
wrong-alg	L1	BAD_ALG
zero-model-hash	L3	ZERO_MODEL_HASH
bad-measurement-length	L3	BAD_MEASUREMENT_LENGTH
nonce-mismatch	L4	NONCE_MISMATCH
model-hash-mismatch	L4	MODEL_HASH_MISMATCH
platform-mismatch	L4	PLATFORM_MISMATCH
stale-iat	L4	TIMESTAMP_STALE

Table 6

Complete vector files (JSON with hex-encoded COSE bytes, expected failure codes, and policy overrides) are available in the reference implementation repository.

14. References

14.1. Normative References

- [FIPS180-4] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", August 2015, <<https://csrc.nist.gov/pubs/fips/180-4/upd1/final>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <<https://www.rfc-editor.org/rfc/rfc8032>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.

- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/rfc/rfc8392>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/rfc/rfc8610>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/rfc/rfc8949>>.
- [RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <<https://www.rfc-editor.org/rfc/rfc9052>>.
- [RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation procedures (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <<https://www.rfc-editor.org/rfc/rfc9334>>.
- [RFC9711] Lundblade, L., Mandyam, G., O'Donoghue, J., and C. Wallace, "The Entity Attestation Token (EAT)", RFC 9711, DOI 10.17487/RFC9711, April 2025, <<https://www.rfc-editor.org/rfc/rfc9711>>.

14.2. Informative References

- [I-D.messous-eat-ai] Messous, A., Morand, L., and P. C. Liu, "Entity Attestation Token (EAT) Profile for Autonomous AI Agents", 23 February 2026, <<https://datatracker.ietf.org/doc/draft-messous-eat-ai/>>.
- [RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <<https://www.rfc-editor.org/rfc/rfc7942>>.
- [RFC9782] Lundblade, L., Birkholz, H., and T. Fossati, "Entity Attestation Token (EAT) Media Types", RFC 9782, DOI 10.17487/RFC9782, May 2025, <<https://www.rfc-editor.org/rfc/rfc9782>>.

[SCITT] "Supply Chain Integrity, Transparency, and Trust (SCITT)",
n.d., <<https://datatracker.ietf.org/wg/scitt/about/>>.

Appendix A. Full CDDL Schema

This appendix reproduces the complete CDDL schema from Section 4.5 for convenience.

```
; Attested Inference Receipt (AIR) v1 -- CDDL Schema
; Status: v1.0 FROZEN
; References: RFC 9052, RFC 8392, RFC 9711, RFC 8949, RFC 8610

air-receipt = #6.18([
    protected:  bstr .cbor air-protected-header,
    unprotected: air-unprotected-header,
    payload:     bstr .cbor air-claims,
    signature:   bstr .size 64
])

air-protected-header = {
    1 => -8,           ; alg: EdDSA (Ed25519)
    3 => 61,           ; content type: application/cwt
}

air-unprotected-header = {}

air-claims = {
    ; --- Standard CWT/EAT claims ---
    1  => tstr,           ; iss: issuer
    6  => uint,           ; iat: issued-at (Unix seconds)
    7  => bstr .size 16,   ; cti: CWT ID (UUID v4, 16 bytes)
    265 => "https://spec.cyntrise.com/air/v1", ; eat_profile
    ? 10 => bstr .size (8..64), ; eat_nonce (optional)

    ; --- AIR private claims ---
    -65537 => tstr,           ; model_id
    -65538 => tstr,           ; model_version
    -65539 => sha256-hash,    ; model_hash
    -65540 => sha256-hash,    ; request_hash
    -65541 => sha256-hash,    ; response_hash
    -65542 => sha256-hash,    ; attestation_doc_hash
    -65543 => enclave-measurements, ; enclave_measurements
    -65544 => tstr,           ; policy_version
    -65545 => uint,           ; sequence_number
    -65546 => uint,           ; execution_time_ms
    -65547 => uint,           ; memory_peak_mb
    -65548 => tstr,           ; security_mode
}
```

```

; --- Optional claims (v1.0) ---
? -65549 => tstr,                ; model_hash_scheme
}

sha256-hash = bstr .size 32
sha384-hash = bstr .size 48

enclave-measurements = nitro-measurements / tdx-measurements

nitro-measurements = {
    "pcr0"           => sha384-hash,
    "pcr1"           => sha384-hash,
    "pcr2"           => sha384-hash,
    ? "pcr8"         => sha384-hash,
    "measurement_type" => "nitro-pcr",
}

tdx-measurements = {
    "pcr0"           => sha384-hash,    ; MRTD
    "pcr1"           => sha384-hash,    ; RTMR0
    "pcr2"           => sha384-hash,    ; RTMR1
    "measurement_type" => "tdx-mrtd-rtmr",
}

```

Appendix B. Golden Vector Summary

The reference implementation includes 10 golden test vectors (2 valid, 8 invalid) generated with a deterministic Ed25519 key pair:

```
* Seed:  
2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a  
  
* Public key:  
197f6b23e16c8532c6abc838facd5ea789be0c76b2920334039bfa8b3d368d61
```

Vectors are JSON files containing the COSE_Sign1 bytes (hex-encoded), expected verification outcomes, and policy overrides for Layer 4 tests. They are available in the repository under `spec/v1/vectors/`.

Valid vectors:

- ```
* vl-nitro-no-nonce.json: Nitro measurements, no eat_nonce
 (canonical golden vector).

* vl-tdx-with-nonce.json: TDX measurements, with eat_nonce (tests
 nonce binding and TDX measurement variant).
```

Invalid vectors exercise specific failure modes across all four verification layers:

- \* v1-wrong-key.json (L2: SIG\_FAILED)
- \* v1-wrong-alg.json (L1: BAD\_ALG)
- \* v1-zero-model-hash.json (L3: ZERO\_MODEL\_HASH)
- \* v1-bad-measurement-length.json (L3: BAD\_MEASUREMENT\_LENGTH)
- \* v1-nonce-mismatch.json (L4: NONCE\_MISMATCH)
- \* v1-model-hash-mismatch.json (L4: MODEL\_HASH\_MISMATCH)
- \* v1-platform-mismatch.json (L4: PLATFORM\_MISMATCH)
- \* v1-stale-iat.json (L4: TIMESTAMP\_STALE)

#### Acknowledgments

The author thanks the RATS working group for the foundational architecture ([RFC9334]), the EAT editors for the profiling framework ([RFC9711]), and the COSE editors for the signing structures ([RFC9052]). The measurement of confidential computing overhead referenced in this document was performed on AWS Nitro Enclaves and GCP Confidential Space (Intel TDX).

#### Author's Address

Borys Tsyrulnikov  
Cyntrisec  
Email: borys@cyntrisec.com