RATS
Internet-Draft
Intended status: Informational
Expires: 2 September 2026

B. Tsyrulnikov
Cyntrisec

1 March 2026

Attested Inference Receipt (AIR): A COSE/CWT Profile for Confidential AI Inference

draft-tsyrulnikov-rats-attested-inference-receipt-00

Abstract

This document defines the Attested Inference Receipt (AIR), a COSE_Sign1 envelope carrying CWT claims profiled per the Entity Attestation Token (EAT) framework. An AIR receipt binds model identity, input/output hashes, platform attestation metadata, and operational telemetry into a single signed artifact suitable for audit, compliance, and 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). It supports AWS Nitro Enclaves and Intel TDX measurement formats, with extension points for additional platforms. Pipeline chaining and multi-inference receipts are out of scope for this version.

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 2 September 2026.

Copyright Notice

Table of Contents

1. Introduction

Regulated industries increasingly deploy machine learning models on cloud infrastructure but 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.

The Attested Inference Receipt (AIR) fills this gap. An AIR receipt is a 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). A verifier can confirm the receipt's integrity, the signing algorithm, and the claim values using only widely available COSE/CWT libraries and the workload's Ed25519 public key.

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.

AIR v1 defines the per-inference receipt as the base primitive. Future AIR profiles may define aggregation mechanisms for high-throughput deployments (for example, Merkle-root commitments over multiple inference events) while preserving the same verification semantics. Such aggregation mechanisms are out of scope for AIR v1.

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 verification by any party with access to the Ed25519 public key, without TEE-specific tooling.

4. Provide a portable measurement map that accommodates multiple TEE platforms (currently Nitro PCR and Intel TDX MRTD/RTMR).

5. Establish extension points for future platforms and claims without breaking v1 verifiers.

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).

* 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): A 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 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. 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: The TEE hardware vendor (e.g., AWS for Nitro, Intel for TDX) whose attestation infrastructure anchors trust in the platform measurements carried by the receipt.

Measurement Map: The enclave_measurements claim containing platform-specific register values (PCRs for Nitro, MRTD/RTMRs for TDX) that identify the workload code and configuration.

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 verify_strict semantics per [RFC8032] Section 5.1.7. Verifiers MUST reject non-canonical S values (S >= L where L is the Ed25519 group order).

4.3. Unprotected Header

The unprotected header MUST be empty for AIR v1 receipts. The CDDL permits an optional kid (label 4, type bstr) for forward compatibility, but the reference implementation rejects non-empty unprotected headers because unprotected header parameters are not covered by the COSE signature and can be tampered in transit.

Verifiers SHOULD 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 (shorter encoded key sorts first, then bytewise lexicographic comparison).

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 = {
  ? 4 => bstr,      ; kid: key identifier (reserved)
}

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.cyntrisec.com/air/v1",  ; eat_profile
  ? 10 => bstr,                 ; 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., "cyntrisec.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: now - max_age <= iat <= now + clock_skew. Verifiers SHOULD reject future timestamps.

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.cyntrisec.com/air/v1". Verifiers MUST reject receipts with unknown eat_profile values. The value is an identifier, not a dereference requirement. Verifiers MUST NOT require network retrieval of this URI during validation.

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. Keys -65550 through -65599 are reserved for v1.x extensions.

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.

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.

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 SHOULD independently obtain and verify the attestation document, then compare its 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 field names for cross-platform verifier simplicity. The measurement_type field disambiguates the actual register semantics.

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).

New scheme values MAY be registered in v1.x minor updates. Implementations MUST NOT invent unregistered scheme values.

6. EAT Profile Declaration

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

1. *Profile identifier*: URI "https://spec.cyntrisec.com/air/v1" (carried in eat_profile, key 265). This URI is used as an opaque identifier and does not imply that validation depends on a hosted verifier service.

2. *Encoding*: CBOR only ([RFC8949]). JSON serialization is not defined.

3. *Envelope*: COSE_Sign1 ([RFC9052] Section 4.2), CBOR tag 18. Untagged COSE_Sign1 MUST be rejected.

4. *Payload content type*: COSE content_type = 61 (application/cwt). The payload is a CWT claims map.

5. *HTTP media type*: application/eat+cwt ([RFC9782]). Receivers SHOULD accept both application/cwt and application/eat+cwt.

6. *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). Optional kid in the unprotected header is reserved but currently rejected by the reference implementation.

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 (shorter encoded form first, then bytewise lexicographic).

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. The CDDL permits an optional kid (label 4) for forward compatibility, but unprotected parameters are not signed and can be tampered in transit.

15. *Private claim keys*: Keys -65537 through -65549 are assigned in the CWT private-use range ([RFC8392]). No IANA registration is required. Keys -65550 through -65599 are reserved for v1.x extensions.

7. Verification Procedure

The AIR v1 verification procedure is organized into four layers. Each layer MUST complete successfully before proceeding to the next. If any check fails, the verifier MUST reject the receipt and SHOULD report the specific failure.

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. Decode the payload. Confirm it is a well-formed CBOR map.

8. Confirm eat_profile (key 265) equals "https://spec.cyntrisec.com/air/v1". Reject receipts with unknown profile values.

7.2. Layer 2: Cryptographic Verification

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 now - max_age <= iat <= now + 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 an EAT profile for autonomous AI agents, including model identification, training metadata, and performance metrics. AIR v1 is complementary: where draft-messous-eat-ai focuses on broad AI agent provenance metadata (potentially including training and evaluation details), AIR v1 focuses narrowly on per-inference execution evidence from a confidential workload. 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 receipts fit the RATS [RFC9334] architecture as follows:

* The confidential workload is the *Attester* (it generates evidence in the form of receipts).

* The receipt consumer is the *Verifier* (it validates signatures and claims).

* The end user, auditor, or compliance officer is the *Relying Party* (they consume verification results).

* The TEE hardware vendor (AWS, Intel) is the *Endorser* (their attestation infrastructure anchors trust).

AIR v1 is a workload-emitted artifact, not a verifier-emitted attestation result. It is distinct from IETF EAR (EAT Attestation Result), which is produced by a verifier after evaluating platform evidence. In a complete deployment, an EAR might reference an AIR receipt as part of the evidence it evaluated.

9. Security Considerations

9.1. Receipt Integrity

The Ed25519 signature over the COSE Sig_structure1 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. Verifiers requiring TEE 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. Signing Key Binding

AIR v1 does not define how the Ed25519 signing key relates to the TEE attestation. Implementations 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 user data field, TDX REPORTDATA).

3. Provide the attestation document alongside the receipt for end-to-end verification.

9.7. 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.

9.8. 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.

9.9. 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.

9.10. 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.

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.cyntrisec.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: Cyntrisec

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: Deployment-validated. Emitted in E2E paths on three platforms.

Coverage: 575 tests passing (including 16 AIR v1 conformance vector tests).

Performance snapshot (non-normative): 2026-03-01 AWS build-host microbenchmark aggregate (air_v1_aws_build_bench_5runs_2026-03-01) measured crypto and verifier costs on an AWS c6i.xlarge Linux host after compiling ephemeralml-verify from source (valid AIR vector size 599 bytes). Values below are 5-run median and p95:

Metric	Median	p95	Notes
SHA-256 (1 KB)	0.931 us	0.931 us	OpenSSL 3.2.2 speed conversion
SHA-256 (4 KB)	3.227 us	3.227 us	OpenSSL 3.2.2 speed conversion
Ed25519 sign	31.763 us	31.887 us	OpenSSL 3.2.2 speed
Ed25519 verify	102.512 us	103.338 us	OpenSSL 3.2.2 speed
AIR verify (Rust CLI, process-per-call)	1,533.100 us	1,558.608 us	Includes process spawn overhead

Table 5

Estimated receipt emission crypto path for a 1 KB request + 4 KB response plus a 1 KB attestation hash and Ed25519 signing is 36.852 us median (36.958 us p95) per inference on this host.

Separate retest run (non-normative): A second 5-run retest on the same AWS instance class (air_v1_aws_retest_bench_5runs_2026-03-01) reproduced SHA-256 and Ed25519 primitive timings within approximately 1%, and reproduced the receipt emission crypto estimate at 37.035 us median (+0.5% versus the baseline snapshot). The Rust CLI process-per-call verify metric increased by +14.7% in the retest; this metric includes process fork/exec/linker overhead and is environment-sensitive. For this reason, AIR v1 performance interpretation should prioritize the per-inference crypto path estimate rather than CLI process-per-call latency.

Environment-sensitivity check (non-normative): A separate run on GCP n2-standard-4 (Intel Xeon @ 2.80 GHz, OpenSSL 3.0.13, Ubuntu 24.04) measured the emit crypto path at 62.178 us median. The absolute values differ from AWS due to OpenSSL version (3.0 vs 3.2 assembly paths) and CPU generation, not protocol logic. This confirms the AIR crypto path remains in tens of microseconds across tested environments.

These measurements are environment-specific and informative only; AIR v1 does not define performance requirements.

Contact: borys@cyntrisec.com

12.2. Python Interop Verifier

Organization: Cyntrisec (same team, separate Python implementation)

Implementation: 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 6

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 16 required claims plus the EAT profile:

```
{
  1: "cyntrisec.com",                          / iss /
  6: 1740000000,                               / iat /
  7: h'<16 bytes UUID v4>',                    / cti /
  265: "https://spec.cyntrisec.com/air/v1",    / eat_profile /
  -65537: "minilm-l6-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 7

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/publications/detail/fips/180/4/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", 2026.

[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 = {
  ? 4 => bstr,      ; kid: key identifier (reserved)
}

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.cyntrisec.com/air/v1",  ; eat_profile
  ? 10 => bstr,                 ; 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:
  2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a

* 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 vectors/.

Valid vectors:

* v1-nitro-no-nonce.json: Nitro measurements, no eat_nonce (canonical golden vector).

* v1-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