Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 1 April 2026

N. Ritz
Independent

28 September 2025

Ephemeral Compute Attestation (ECA) - Implementation Guide
draft-ritz-eca-impl-00

Abstract

This document provides implementation guidance, deployment patterns, and integration considerations for the Ephemeral Compute Attestation (ECA) protocol. It explores concrete Instance Factor patterns ranging from hardware-rooted to artifact-based approaches, documents operational experiences from prototype implementations, and describes how ECA integrates with existing identity frameworks such as ACME and SPIFFE/SPIRE. The document includes practical examples, test vectors, and architectural guidance for deploying ECA across diverse compute environments from cloud VMs to bare-metal systems.

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 1 April 2026.

Copyright Notice

Table of Contents

1. Introduction

The Ephemeral Compute Attestation (ECA) protocol [I-D.ritz-eca] defines a transport-agnostic, three-phase ceremony for establishing compute instance identity without pre-shared operational credentials. While the core protocol document specifies the normative requirements and security properties, this companion document addresses the practical aspects of implementing and deploying ECA in real-world environments.

This guide serves three primary audiences:

1. *Implementers* seeking concrete examples, test vectors, and reference code for building ECA-compliant systems
2. *Operators* evaluating deployment patterns and integration strategies for their infrastructure
3. *Architects* designing systems that leverage ECA for identity bootstrapping in multi-cloud or high-assurance environments

The guidance ranges from minimal deployments suitable for individual developers to enterprise-scale integrations with existing identity management systems. Each pattern is accompanied by security considerations, operational trade-offs, and implementation notes drawn from prototype experience.

2. Instance Factor Patterns (IFP)

ECA supports full integration with hardware roots of trust (HRoT) where available, and such integration is RECOMMENDED. ECA does not replace the need for HRoTs where the threat model must assume a compromised service provider, hypervisor or related platform risks.

The choice of IFP pattern determines the source of the IF and the strength of the resulting security guarantee. The security of the ECA protocol's initial phase depends on the Attester proving possession of this secret IF, which is bound to the public *Boot Factor (BF)*.

The three defined patterns are:

* *IFP Pattern A (Hardware-Rooted)*: The IF is a secret value derived from a hardware root of trust (HRoT), such as a vTPM or TEE. This pattern provides the highest level of security, as it can mitigate threats from a compromised provider.

* *IFP Pattern B (Orchestrator-Provisioned)*: The IF is a secret provided by a trusted orchestrator through a secure channel, like instance metadata. This approach protects against network attackers but assumes the infrastructure provider is trusted.
* *IFP Pattern C (Artifact-Based)*: The IF is the entire content of a larger provisioned file (e.g., an authorized_keys file) that also contains the BF. This pattern is designed to address Trust-on-First-Use (TOFU) vulnerabilities in constrained environments.

3. Security Considerations

See ECA Protocol Security Considerations (https://datatracker.ietf.org/doc/draft-ritz-eca-00/)

3.1. Minimal Deployment and Trust Chain Sketch (Pattern C)

This section illustrates how ECA can be used even at a small, human-driven scale— such as by an individual developer— to provide cryptographic assurance for ephemeral instances without requiring complex infrastructure or hardware roots of trust, using IFP Pattern C. For security considerations with this pattern, see the core protocol document’s Section 6.2 "Impersonation Risk".

In this sketch, the Instance Factor (IF) is an artifact-based secret such as the full content of an injected file containing the Boot Factor (BF). Mapped to RATS architecture roles, the laptop is the Verifier, the VM is the Attester and the individual developer acts effectively as the Relying Party (RP).

1. Developer trusts their local ECA toolchain CLI for BF generation
2. Developer trusts Service Provider to correctly inject BF/SSH public key
3. Developer trusts their laptop to keep VF confidential
4. VM proves possession of BF+IF to receive VF
5. VM proves possession of BF+VF to complete attestation
6. Developer has acceptable assurance to connect directly with VM

| Implementation note: As of this draft (-00), preliminary tests
| with a prototype CLI toolchain suggest a total attestation latency
| of approx. 1.5 seconds— from VM liveliness to actionable results.
| See Section 8 for further implementation details.

4. Post-Attestation Patterns

Once the ceremony has concluded, operators can make policy decisions about how to handle the Attestation Result (AR). This may include transmitting the AR directly to the successful Attester so that it may present the AR to Relying Parties (RPs) who trust the Verifier's signature. The full scope and mechanism of presenting and accepting ARs to RPs is outside the scope of this document.

4.1. Stateful Re-Attestation for Long-Running Instances

The ECA protocol is primarily designed for the initial identity bootstrap of ephemeral compute instances. However, long-running workloads may require a mechanism for renewing their operational credentials. A credential renewal can be modeled as a "re-attestation" ceremony.

In this model, the original, stable eca_attester_id identity would serve as a Renewal Factor (RF), analogous to the BF. The new Instance Factor (IF) would be a manifest of "known good" measurements of the instance's current state (e.g., hashes of critical binaries or configuration files). This turns the renewal into a periodic health and integrity check, ensuring the instance remains in a known-good state throughout its lifecycle. A future profile of ECA may define a renewal protocol based on stateful re-attestation.

4.2. Chaining and Hierarchical Trust

The ECA protocol is inherently composable, enabling the creation of multi-layer trust architectures that can propagate trust from hardware up through layers of software. This is achieved by using the signed Attestation Result (AR) from one ceremony as a cryptographic input— specifically, the Instance Factor (IF)— for a subsequent ceremony.

A straightforward deployment pattern for this is represented by the following "bare-metal-to-VM" attestation chain:

1. *Initial Attestation (Hardware Layer):* A physical host (Attester i) performs an ECA ceremony using an *Instance Factor* derived from a hardware root of trust (e.g., a TPM quote, per IFP Pattern A). It attests to a low-level Verifier (Verifier i) that is trusted to appraise hardware integrity. The successful result is a signed Attestation Result, AR_i.

2. *Second-Level Attestation (VM Layer):* A virtual machine (Attester ii) is instantiated on the host. Its provisioned *Instance Factor* is the signed AR_i from the hardware layer.

Attester ii performs its own ECA ceremony with a higher -level cloud orchestrator (Verifier ii). To validate, Verifier ii first cryptographically verifies AR_i (confirming it trusts Verifier i), and if valid, proceeds with the rest of the ECA ceremony.

The final result, AR_ii, is a portable credential that cryptographically proves a healthy VM is running on a specific, healthy, and previously attested physical host. The same pattern can be used to bridge trust domains, for example by consuming an SVID from an existing SPIFFE/SPIRE infrastructure as the Instance Factor for an attester in a separate cloud environment. A future profile of ECA may define a specific profile for chaining and hierarchical trust.

4.3. Risks and Mitigations for Composable Deployments

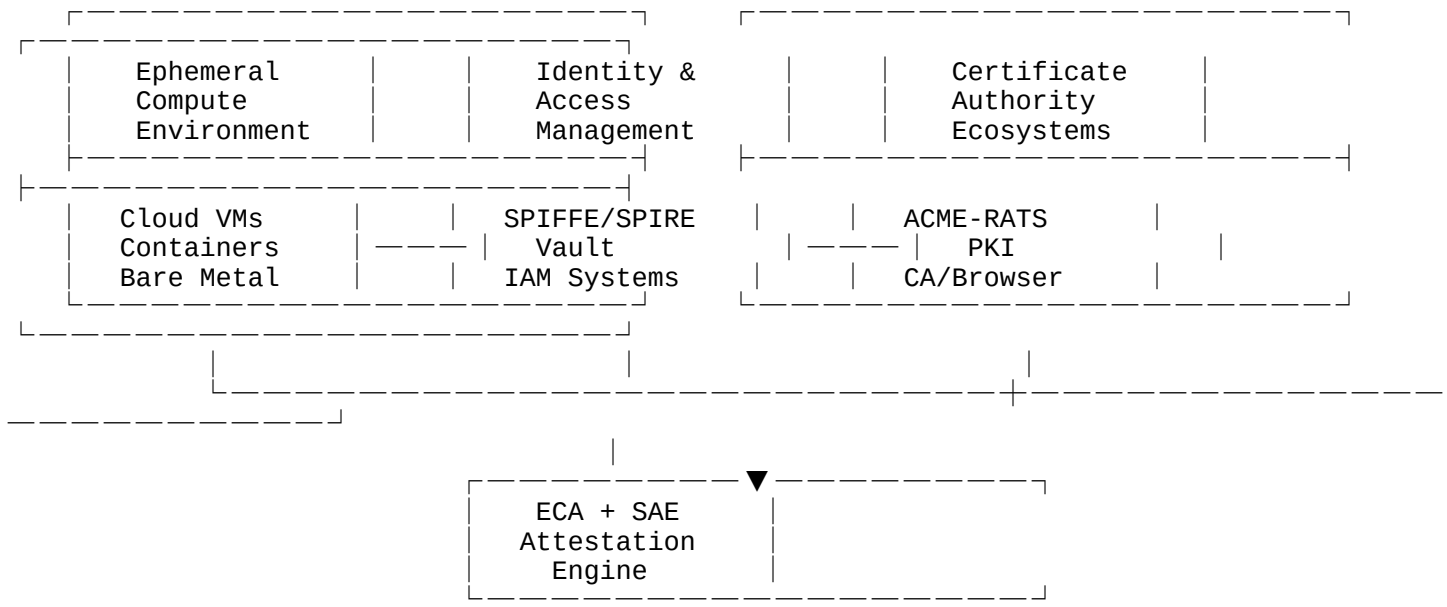
While ECA is designed to be composable (e.g., chaining attestations), realizing this benefit in large teams is expected to require significant operational discipline. Operators should be aware of the following risks:

The "Glue Code" Trap
: The security of the overall system depends on the integrity of each link in the chain. Custom scripts or shims used to connect different attestation layers can inadvertently reintroduce the very vulnerabilities (e.g., parsing flaws, state management bugs) that SAE [I-D.ritz-sae] is designed to eliminate. It is STRONGLY RECOMMENDED to use standardized, well-vetted integrations (e.g., official plugins for tools like Vault or SPIRE) over bespoke "glue code."

Organizational Friction
: In multi-team environments, clear ownership of the end-to-end attestation process is critical. Without a shared governance model, configuration drift between what DevOps provisions, what Security expects, and what the application implements can lead to systemic failures.

5. Integration with Existing Frameworks

The ECA protocol is designed to complement, not replace, existing identity and attestation systems. It acts as a foundational "attestation engine" that fills specific gaps in cross-domain portability and high-assurance bootstrapping for ephemeral workloads. Its role is to provide a verifiable, portable proof of identity that can be consumed by a wide range of higher-level identity frameworks and certificate issuance protocols, as illustrated below.



5.1. Realizing the RATS Passport Model

ECA aligns with the Passport Model of the RATS Architecture [RFC9334], where the Attester obtains a portable Attestation Result (e.g., an EAT [RFC9711]) from the Verifier for presentation to Relying Parties. While RATS provides the roles and terminology for remote attestation, it does not specify a concrete protocol for cross-cloud identity bootstrapping. ECA fills this gap by defining a phased exchange that produces standardized EATs bound to joint ephemeral factors, enabling interoperability across heterogeneous providers.

5.2. ECA + ACME-RATS

A powerful use case for ECA is as a mechanism to satisfy the attestation challenges proposed within the ACME working group, as described in the "(ACME) rats Identifier and Challenge Type" (ACME-RATS) Internet-Draft [I-D.liu-acme-rats]. The ACME-RATS specification defines an abstract challenge/response mechanism for device attestation but intentionally leaves the implementation of the attestation procedure itself out of scope. ECA can act as a bridge, providing the full three-phase ceremony— from initial bootstrap to final proof-of-possession— that an ACME client can execute to produce the verifiable Attestation Result (AR) required by the attestation-result-01 challenge (Passport Model).

When you combine ECA with the ACME-RATS framework, you create a complete, end-to-end automated flow.

This integration approach enables a powerful vision: just as ACME enabled the automation of web server certificates and brought about ubiquitous HTTPS, the combination of ACME-RATS and ECA can enable the automated issuance of high-assurance identities to ephemeral workloads, realizing a "Let’s Encrypt" for Machines."

5.2.1. Conceptual Integration

An integration of an ACME client with an ECA Attester would follow this sequence:

1. *ACME Challenge:* The ACME client (running on the ephemeral instance) requests a certificate and receives an attestation-result-01 challenge from the ACME server. This challenge includes a server-provided token (acting as a nonce for freshness) and optional claimsHint (e.g., required claims like FIPS_mode or OS_patch_level).

2. *ECA Initiation:* The ACME client triggers an ECA ceremony with a trusted Verifier (separate from the ACME Server). The ACME token is passed to the ECA Verifier to be used as (or bound to) the vnonce for the ceremony, ensuring freshness binding.

3. *ECA Ceremony:* The Attester (ACME client/instance) and Verifier execute the full, three-phase ECA protocol as defined in draft-ritz-eca-core. If a claimsHint was provided, the Attester collects corresponding measurements/claims in its Evidence (e.g., EAT). The Verifier ensures the ACME token is included as the nonce claim in the final Evidence EAT (validated at Gate 8: Nonce Match).

4. *Attestation Result:* Upon successful validation (including appraisal against Verifier policy), the Verifier produces a signed Attestation Result (AR) and delivers it to the Attester (e.g., via SAE transport).

5. *ACME Response:* The ACME client wraps the signed AR in a Conceptual Message Wrapper (CMW) with type=attestation-result and submits it to the ACME server, completing the challenge.

6. *ACME Validation (as RP):* The ACME Server verifies the Verifier’s signature on the AR (using pre-configured trust anchors), checks the nonce matches its issued token, appraises claims against policy (including any required claimsHint), and— if valid— issues the certificate.

5.2.2. Roles and Artifacts

This section provides a conceptual, speculative composition where an ECA ceremony supplies an Attestation Result (AR) that satisfies the ACME attestation-result-01 challenge.

Attester
: Ephemeral instance (e.g., VM/workload) running the ACME client and ECA Attester logic. It possesses initial factors (BF + IF), collects Evidence (e.g., from TPM/TEE/platform measurer), and performs the ECA ceremony to obtain an AR.

Evidence Source
: Implementation-specific (e.g., TPM/TEE for hardware-rooted IFP Pattern A; see Section 2).

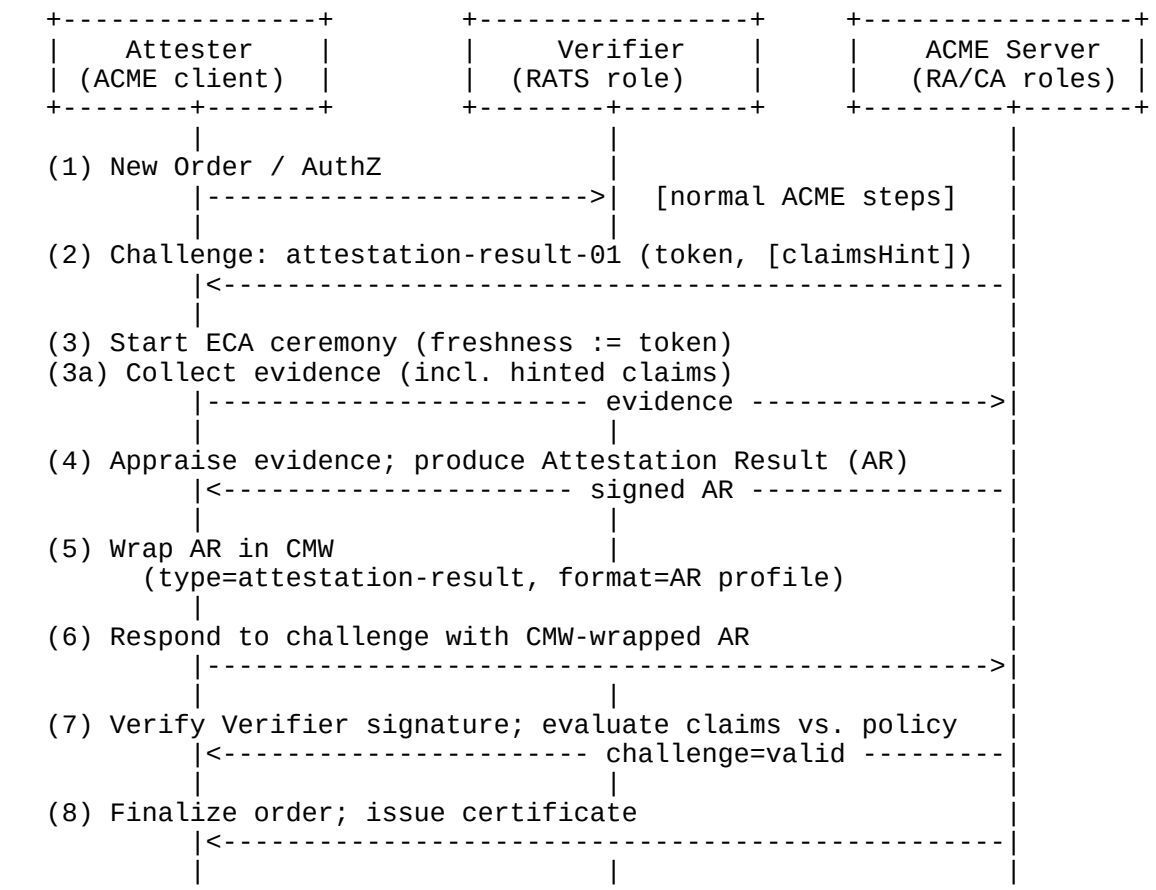
Verifier
: Separate trusted entity (e.g., enterprise-operated or manufacturer-designated). Appraises Evidence against policy, issues signed AR. Trusts anchors for Evidence sources but is _not_ the ACME Server.

ACME Server (RP/RA/CA)
: Issues challenges, acts as Relying Party (validates AR signature/claims/nonce), enforces policy, and finalizes certificate issuance. Pre-configured with trust anchors for one or more Verifiers.

Inputs
: * *ACME Challenge Token*: Freshness nonce; bound to ECA vnonce.
  * *Claims Hint* (optional): Guides Evidence collection (e.g., FIPS_mode, OS_patch_level); reflected in AR claims.

Outputs
: * *Attestation Result (AR)*: Verifier-signed (delivered to Attester).
  * *CMW Object*: Wraps AR (Attester → ACME Server); type=attestation-result, format = AR profile (e.g., AR4SI or EAT).

5.2.3. Operational Flow (Conceptual)



5.2.4. ECA + ACME-RATS Trust Chain Sketch

- *ACME Server* is pre-configured with trust anchors (e.g., key set or CA) for one or more Verifiers.
- *Attester* trusts its local Evidence source (e.g., HRoT) and the Verifier (via ECA's cryptographic proofs) but starts in a privileged credential vacuum—no ACME-specific creds prior to challenge completion.
- *Verifier* publishes a stable identifier (e.g., key id) discoverable by the ACME Server (e.g., via directory or config).
- *Freshness/Nonce Binding*: ACME token is bound to ECA vnonce (e.g., vnonce = token or vnonce = SHA-256(token || eca_uuid)), included in Evidence EAT, and reflected in AR. ACME Server checks match at validation.

5.3. The SPIFFE/SPIRE Framework

SPIFFE/SPIRE provides a robust framework for issuing short-lived cryptographic identities (SVIDs) to workloads, enabling zero-trust authentication in distributed systems. While SPIFFE/SPIRE addresses "secret zero" in many scenarios through platform-specific node attestors (e.g., AWS EC2 or Kubernetes), it relies on extensible plugins for custom environments which is a natural fit for an ECA plugin implementation. SPIFFE/SPIRE is a CNCF-graduated community standard rather than an IETF standard.

What ECA/SAE adds. ECA defines an exposure-tolerant, accept-once bootstrap that binds artifacts across dual channels and emits standardized EAT-based evidence and Attestation Results (AR). SAE provides a pull-only, static-artifact transport that works in heterogeneous or constrained networks. These properties make ECA a good fit as a high-assurance _node_ attestor feeding SPIRE, without changing SPIFFE's SVID/workload APIs.

Integration surface (SPIRE). SPIRE supports authoring custom plugins via its extension points, including node-attestors. Prior work integrates TPM-/IMA-based attestation via Keylime, illustrating the "hardware- or higher-assurance attestor" pattern that ECA can follow.

Operational intent. ECA does not replace SPIFFE/SPIRE. It _precedes_ or _augments_ SPIRE node admission where provider metadata is weak, join-token operations are costly, or transports are constrained. After ECA succeeds, SPIRE issues SVIDs and federates as usual.

| _Terminology note._ In SPIRE, node-attestor plugins expose
| *selectors* (key-value attributes used in SPIRE's policy engine)
| that registration policies can match. In the ECA→SPIRE mapping,
| fields from ECA's EAT/AR (e.g., eca_uuid, EUID, JP/PoP artifacts,
| integrity beacons) naturally become such selectors. _(SPIRE
| selector/registration context: SPIRE-CONCEPTS (title:).)_

```
# Example of SPIRE registration entry using ECA-derived selectors
spire-server entry create -spiffeID "spiffe://example.org/my-service" \
  -parentID "spiffe://example.org/spire/agent/eca/<verifier_id>" \
  -selector "eca:euid:a1b2c3d4..." \
  -selector "eca:ihb:e5f6g7h8..."
```

Complementary deployment patterns:

Alt-cloud / bare-metal without signed metadata.

Today: defaults to join tokens or bespoke attestors where robust, signed instance identity is absent. _ECA integration:_ implement ECA as a node-attestor plugin; ECA’s EAT/AR fields (e.g., EUID, IHB, PoP, nonces) become SPIRE selectors for registration, then SPIRE issues SVIDs.

High-assurance with HRoT (Confidential/edge).
Today: SPIRE can leverage TPM/IMA via Keylime but still depends on environment-specific control planes. _ECA integration:_ ECA Pattern A binds identity to HRoT and proves joint possession via SAE; SPIRE consumes the AR to gate SVID issuance. _(Keylime integrations:_ KEYLIME-SPIRE-PLUGIN (title:), REDHAT-KEYLIME-SPIRE (title:); _SPIRE plugin surface:_ SPIRE-EXTENDING (title:), SPIRE-PLUGIN-SDK (title:).)*

Dynamic multi-cluster aliasing/federation.
Today: coordinating join tokens and node aliases across domains can be operationally heavy. _ECA integration:_ selectors derived from ECA’s EAT/AR (e.g., eca_uuid, EUID, JP) provide portable, verifiable bindings without maintaining a separate join-token database.

Standards position.
SPIFFE/SPIRE are CNCF community standards; ECA/SAE are IETF Internet-Drafts. This document positions ECA to _interoperate with_ SPIFFE/SPIRE— augmenting bootstrap where needed— rather than to replace them. _(Status/background:_ CNCF-GRADUATION (title:), SPIFFE-OVERVIEW (title:).)*

5.4. BRSKI (Bootstrapping Remote Secure Key Infrastructure)

BRSKI [RFC8995] solves _manufacturer-anchored onboarding_ for physical devices that ship with an IEEE 802.1AR IDevID and a manufacturer voucher service (MASA). ECA targets _ephemeral compute_ (VMs, containers) that typically lack such an identity.

The mechanisms are complementary: *BRSKI* is for day-0 hardware onboarding based on supply-chain provenance, while ECA is for just-in-time software and instance attestation at runtime. An operator could use BRSKI to securely enroll a physical device into their network, and then use ECA as a subsequent, continuous attestation check to validate the software state running on that device before releasing application-level privileges.

5.5. Summary of Integration Benefits

Adopting ECA as a foundational attestation engine provides several key benefits:

* *Standards-Based:* Built on emerging and established IETF standards like RATS, EAT, and ACME.
* *Portable:* The protocol's transport-agnostic design works across cloud, on-premise, and edge environments.
* *Composable:* Can be layered with existing systems like SPIFFE/SPIRE to enhance their security posture.
* *High-Assurance:* Supports hardware roots of trust (IFP Pattern A) for zero-trust environments.
* *Automation-Friendly:* Designed from the ground up for ephemeral, dynamic, and automated infrastructures.

6. Operational Considerations

Scalability The use of a simple artifact repository allows for high scalability using standard web infrastructure like CDNs and object storage.

Time Synchronization Reasonably synchronized time is REQUIRED for proper validation of the nbf and exp time windows (Gate 4 skew tolerance: ±60s). The use of a time synchronization protocol like NTP [RFC5905] is RECOMMENDED. Polling MUST use exponential backoff with jitter.

Addressing Complexity The multi-phase design of ECA is intentionally confined to the infrastructure layer to provide a simple and secure operational experience. ECA's cryptographic machinery is expected to be abstracted away from the end-user. The prototype implementation demonstrates this, executing a complete, parallel attestation with a single command (e.g. eca-toolchain attest --manifest ./manifest.yml), similar to how a sophisticated suite of standards (SMTP, DKIM, etc.) underpins a simple email "send" button.

6.1. Provisioning and Repository Access

The ECA protocol requires the Attester to publish artifacts while adhering to the *Privileged Credential Vacuum* design principle (see core protocol Section 3). This is achievable using standard cloud primitives that grant ephemeral, narrowly-scoped write capabilities without provisioning long-term secrets. Common patterns include the control plane injecting a time-limited pre-signed URL (e.g., for Amazon S3 or GCS) or a short-lived, scoped OAuth2 token for the instance to use. In this model, the Attester is granted the temporary _capability_ to write to its specific repository path, fulfilling the protocol's needs without violating the zero-trust principle of verify-then-trust. Verifiers MUST NOT rely on any CA or key material delivered by the Attester for appraisal trust establishment. This reinforces the requirement in core protocol

Section 3.1.7.

7. IANA Considerations

See ECA Protocol IANA Considerations
(https://datatracker.ietf.org/doc/draft-ritz-eca-00/)

8. Reference Profile: ECA-VM-v1

| Stability note: This profile documents the concrete choices used
| by the reference prototype to enable experimentation and interop.
| It is non-normative and may change in future drafts based on
| feedback.

8.1. Primitives

* Hash / KDF: HKDF-SHA-256 (RFC5869), SHA-256 (RFC6234)
* MAC: HMAC-SHA-256
* Signatures: Ed25519 (RFC8032)
* KEM/HPKE: X25519 + HPKE base mode (RFC9180) for Verifier -> Attester secrecy in Phase 2. The eca_uuid is used as the AAD, and the info parameter for key derivation is "ECA/v1/hpke".
* Nonces: Verifier freshness vnonce is exactly 16 bytes (encoded base64url, unpadded)

8.2. Integrity Hash Beacon (IHB)

* IHB = SHA-256(BF || IF), rendered as lowercase hex for transport where necessary.

8.3. Deterministic Key Material

All keys are deterministically derived from ceremony inputs via domain-separated HKDF invocations. Notation: HKDF-Extract(salt, IKM) then HKDF-Expand(PRK, info, L). The eca_uuid is appended to the salt in all derivations to ensure session uniqueness.

* *Phase 1 MAC key (Attester artifact MAC)*
  - IKM = BF || IF
  - salt = "ECA:salt:auth:v1" || eca_uuid
  - info = "ECA:info:auth:v1"
  - K_MAC_Ph1 = HKDF-Expand(HKDF-Extract(salt, IKM), info, 32)
  - Usage: HMAC-SHA-256 over the CBOR Phase-1 payload bytes.
* *Phase 2 ECDH/HPKE seed (Attester’s ephemeral X25519 keypair)*
  - IKM = BF || IF

- salt = "ECA:salt:encryption:v1" || eca_uuid
- info = "ECA:info:encryption:v1"
- seed32 = HKDF-Expand(HKDF-Extract(salt, IKM), info, 32)
- The Attester forms an X25519 private key by clamping seed32 per RFC7748; the public key is derived normally.
- The Verifier uses HPKE with the Attester’s public key to encrypt {VF, vnonce}.

* *Phase 3 signing key (Attester’s Ed25519 identity keypair)*

- IKM = BF || VF
- salt = "ECA:salt:composite-identity:v1" || eca_uuid
- info = "ECA:info:composite-identity:v1"
- sk_seed32 = HKDF-Expand(HKDF-Extract(salt, IKM), info, 32)
- The Attester initializes Ed25519 with sk_seed32 as the private key seed and derives the corresponding public key.

* *HPKE KDF info parameter:* info = "ECA/v1/hpke"

8.4. Phase Artifacts

This section provides a high-level description of the payloads. For concrete byte-for-byte examples, see Section 7.

8.4.1. Phase 1 Payload (Attester→Repo)

The Phase-1 payload is a CBOR map containing the following claims, which is then protected by an external HMAC tag.

Claim	Value Type	Description
kem_pub	bstr (raw 32 bytes)	Attester’s ephemeral X25519 public key.
ihb	tstr (lowercase hex)	Integrity Hash Beacon.

Table 1

8.4.2. Phase 2 Payload (Verifier -> Repo)

The Phase-2 payload is a signed CBOR map containing the following claims.

Claim	Value Type	Description
C	tstr (base64url unpadded)	HPKE ciphertext
vnonce	tstr (base64url unpadded)	The Verifier-generated nonce.

Table 2

The plaintext for HPKE encryption is the direct concatenation of the raw bytes: plaintext = VF || vnonce.

8.4.3. Phase 3 Payload (Attester -> Repo)

The Phase-3 payload is a signed EAT as defined in the core protocol Section 11.1. The profile-specific constructions for proofs are as follows:

* *Joint-Possession Proof (concrete for this profile):*
  - jp_proof = SHA-256(BF || VF), rendered as lowercase hex.
* *Proof-of-Possession (concrete for this profile):*
  - First, a bound hash is computed from the session context:
    - bound_data = eca_uuid || IHB_bytes || eca_attester_id_bytes || vnonce_raw_bytes
    - bound_hash = SHA-256(bound_data)
  - Then, a dedicated MAC key is derived:
    - IKM = BF || VF
    - salt = "ECA:salt:kmac:v1" || eca_uuid
    - info = "ECA:info:kmac:v1"
    - K_MAC_PoP = HKDF-Expand(HKDF-Extract(salt, IKM), info, 32)
  - Finally, the PoP tag is computed over the bound hash:
    - pop_tag = base64url(HMAC-SHA-256(K_MAC_PoP, bound_hash))
* The jp_proof and pop_tag are included in the EAT, which is then signed with the Attester’s Ed25519 key.

8.5. Verification (Verifier)

* Verify Phase-1 MAC with K_MAC_Ph1.
* Verify the signed Phase-2 payload with the Verifier’s public key; HPKE-Open with Attester’s kem key to recover {VF, vnonce}.
* Recompute Attester signing key from BF||VF and verify the EAT signature.
* Recompute jp_proof and pop_tag inputs and compare constant-time.
* Apply local appraisal policy; on success, emit an Attestation Result bound to eca_uuid.

8.6. Interop Notes

* *Encodings*: All binary fields referenced in EAT must be explicitly encoded (e.g., base64url) and stated as such in the claims table. NumericDate claims (iat, nbf, exp) use 64-bit unsigned integers.
* *Side-Channel Resistance*: To mitigate timing attacks, implementations SHOULD use constant-time cryptographic comparisons. Payloads that are inputs to cryptographic operations (e.g., Evidence) MAY be padded to a fixed size using a length-prefix scheme to ensure unambiguous parsing.

9. Test Vectors and Payloads

This section provides concrete test vectors generated using the deterministic inputs listed below.

9.1. Deterministic Inputs

Parameter	Value (Base64URL)
eca_uuid	4b6483ee-3d36-4221-ac2e-2c0271aa9d62 (String)
bf	Be80sHHnLhyYH_koGgKTFA
if	aS1kODFhOTc4N2U5MWQ1MTZk
vf	A-g7iYp8nS5Q-1t_1A1gAFpsgAnJb2DE8_2j2b6b2b4
vnonce	VGhpcyBpcyBhIHZub25jZQ
verifier_priv_key	T2vjA9ssm2E-s2e-8a0A6c0d8f0A4c0B8g0B2a0E6e0F8g0B2a0E4c0d6e0F2a0B
iat/nbf	1759020000 (Integer)
exp	1759020300 (Integer)

Table 3

9.2. Phase 1: Attester to Verifier

* *CBOR Payload (Hex):* a2676b656d5f7075625820a7238510a716447881c798033b2591a329d70d473b1f31b25e79c5324ab2a5436369686278407d772921258b68a41a4825ce2de85626305a4e5113d03bb63222338248d5516a

* *HMAC Tag (Hex):*
  47f607144e54619b165b4528174542d9972332617f699043236e7655938d2146

9.3. Phase 2: Verifier to Attester

* *COSE_Sign1 (Hex):* 8443a10127a1045820f18146247c40465243179a32c286
  d933b499a22f4b0653063529b3506e5793085888a261437882692d38642d31624e
  2d397a502d356f4e2d38774d2d39774e2d37764c2d396f4d2d3572502d336f4e2d
  31724d2d3977502d38764e2d377a4d2d386f4e2d3977502d37774d2d36724e2d39
  774d66766e6f6e6365781656476870637942706379426849485a756232356a5a51
  5840a1b5d1e433433ad75a7b5a59334d58045e057139169a838522304130091807
  09b8602b11548a339598818822306716a4a95829810a08e612809e3557e05

9.4. Phase 3: Attester to Verifier (Evidence EAT)

* *COSE_Sign1 (Hex):* 8443a10127a10458200e6d6d4d12228518898144216834
  162002364402633005510680104163486048590181b8f602041a68c48d601a0104
  1a0105021a68c48d68061a68c48d60077824346236343833656 52d336433362d34
  3232312d616332652d326330323731616139643632 0a781656476870637942706 3
  79426849485a756232356a5a5118ff784061396238633764366535663461336232
  633164306539663861376236633564346533663261316230633964386537663661
  356234633364326531663061396238190109782475726e3a696574663a70617261
  6d733a6561743a70726f66696c653a6563612d763119011178403764373732393 2
  313235386236386134316134383235636532646538353632363330356134653531
  313364303362623633323232333338323438643535313661190112782353 4f6d65
  2d504f502d7461472d696e2d624173653634755 26c2d666f526d41741901136b61
  747465737461746 96f6e1901147840663965386437633662356134663365326431
  633062396138663765366435633462336132663165306439633862376136663565
  346433633262316130663965385840d21a50587d40232c45f8f84724335c0a3739
  7b9870020a160868132924300302061418435860361324386001292024103868 03
  260848123048342938164 0

9.5. Verifier Attestation Result (AR)

* *COSE_Sign1 (Hex):* 8443a10127a1045820f18146247c40465243179a32c286
  d933b499a22f4b0653063529b3506e5793085860a501757665726966696572 2d69
  6e7374616e63652d303031027840613962386337643665356634613362326331 64
  306539663861376236633564346533663261316230633964386537663661356234
  633364326531663061396238061a68c48d650778243462363438336565 2d336433
  362d343232312d616332652d326330323731616139643632381a00280004782475
  726e3a696574663a706172616d733a726174733a7374617475733a737563636573
  735840902803852938108428139528094819438902581028509185012948902481
  903829580194890382098590184918093859013849018392859018392850918 30

9.6. CDDL Schemas

The following CDDL definitions describe the canonical structure of the protocol payloads.

```
; Phase 1: Attester’s initial proof
Phase1Payload = {
  "kem_pub": bstr .size 32,
  "ihb": tstr
}

; Phase 2: Verifier’s challenge
Phase2Payload = {
  "C": tstr,
  "vnonce": tstr
}

; Phase 3: Attester’s final evidence
EvidenceEAT = {
  2 => tstr, ; sub (EUID)
  4 => uint, ; exp
  5 => uint, ; nbf
  6 => uint, ; iat
  7 => tstr, ; jti (eca_uuid)
  10 => tstr, ; nonce
  256 => tstr, ; EUID
  265 => tstr, ; eat_profile
  273 => tstr, ; Measurements (IHB)
  274 => tstr, ; PoP
  275 => tstr, ; IntendedUse
  276 => tstr ; JP
}

; Final Attestation Result from Verifier
AttestationResult = {
  1 => tstr, ; iss
  2 => tstr, ; sub (EUID)
  6 => uint, ; iat
  7 => tstr, ; jti (eca_uuid)
  -262148 => tstr ; status
}
```

9.6.1. Interop Fixtures (JSON)

This JSON object contains all deterministic values and artifacts as hex-encoded strings, suitable for automated testing.

```
{
  "deterministic_inputs": {
    "eca_uuid": "4b6483ee-3d36-4221-ac2e-2c0271aa9d62",
    "bf_b64url": "Be80sHHnLhyYH_koGgKTFA",
    "if_b64url": "aS1kODFhOTc4N2U5MWQ1MTZk",
    "vf_b64url": "A-g7iYp8nS5Q-1t_1A1gAFpsgAnJb2DE8_2j2b6b2b4",
    "vnonce_b64url": "VGhpcyBpcyBhIHZub25jZQ",
    "timestamps": {
      "iat": 1759020000,
      "nbf": 1759020000,
      "exp": 1759020300
    }
  },
  "phase_1": {
    "payload_cbor_hex": "a2676b656d5f7075625820a7238510a716447881c798033b2591a329d70d4
73b1f31b25e79c5324ab2a5436369686278407d772921258b68a41a4825ce2de85626305a4e5113d03bb63222
338248d5516a",
    "mac_hex": "47f607144e54619b165b4528174542d9972332617f699043236e7655938d2146"
  },
  "phase_2": {
    "cose_sign1_hex": "8443a10127a1045820f18146247c40465243179a32c286d933b499a22f4b065
3063529b3506e5793085888a261437882692d38642d31624e2d397a502d356f4e2d38774d2d39774e2d37764c
2d396f4d2d3572502d336f4e2d31724d2d3977502d38764e2d377a4d2d386f4e2d3977502d37774d2d36724e2
d39774d66766e6f6e6365781656476870637942706379426849485a756232356a5a515840a1b5d1e433433ad7
5a7b5a59334d58045e057139169a83852230413009180709b8602b11548a339598818822306716a4a95829810
a08e612809e3557e05"
  },
  "phase_3_evidence": {
    "cose_sign1_hex": "8443a10127a10458200e6d6d4d122851889814421683416200236440263300
551068010416348604859018 1b8f602041a68c48d601a01041a0105021a68c48d68061a68c48d600778243462
3634383365652d336433362d343232312d616332652d326330323731616139643632 0a7816564768706379427
06379426849485a756232356a5a5118ff784061396238633764366535663461336232633164306539663861 37
623663356434653366326131623063396438653766366135623463336432653166306139623819010978247 57
26e3a696574663a706172616d733a6561743a70726f66696c653a6563612d763119011178403764373732393 2
313235386236386134316134383235636532646538353632363330356134653531313364303362623633232 3
233338323438643535313661190112782353 4f6d652d504f502d7461472d696e2d62417365363475526c2d66
6f526d41741901136b617474657374617469 6f6e1901147840663965386437633662356134663365326431633
062396138663765366435633462336132663165306439633862376136663565346433632623161306639653 8
5840d21a50587d40232c45f8f84724335c0a37397b9870020a160868132924300302061418435860361324386
001292024103868032608481230483429381640"
  },
  "attestation_result": {
    "cose_sign1_hex": "8443a10127a1045820f18146247c40465243179a32c286d933b499a22f4b065
3063529b3506e5793085860a501757665726966696 5722d696e7374616e63652d303031027840613962386337
643665356634613362326331643065396638613762366335643465336632613162306339643865376636613 56
234633364326531663061396238061a68c48d650778243462363438336565 2d336433362d343232312d616332
652d326330323731616139643632381a0028000478247572 6e3a696574663a706172616d733a726174733a737
4617475733a737563636573735840902803852938108428139528094819438902581028509185012948902481
903829580194890382098590184918093859013849018392859018392850918 30"
  }
}
```


10. Implementation Status

A working prototype demonstrates end-to-end attestation including:

* Complete three-phase protocol implementation
* EAT-compliant evidence generation
* Concurrent execution capability
* Docker-based orchestration for testing

Metric	Value	Notes
Protocol Execution	_1.3s	Phases 1-3, excluding infra
Full Attestation (incl. containers)	_6s	Parallel runs, randomized mode
Scalability	3 concurrent	No failures observed

Table 4

An end-to-end happy-path version of the Prototype is available at [GH-ECA-SAE-PROTO].

11. Acknowledgments

The design of this protocol was heavily influenced by the simplicity and security goals of the AGE [AGE] file encryption tool. The protocol's core cryptographic mechanisms would not be as simple or robust without the prior work of the IETF community in standardizing modern primitives, particularly Hybrid Public Key Encryption (HPKE) in [RFC 9180]. The author wishes to thank the contributors of these foundational standards for making this work possible.

12. Normative References

[I-D.ritz-eca] Ritz, N., "Ephemeral Compute Attestation (ECA) Protocol", Work in Progress, Internet-Draft, draft-ritz-eca-00, 28 September 2025.

[I-D.ritz-sae] Ritz, N., "Static Artifact Exchange (SAE) Protocol", Work in Progress, Internet-Draft, draft-ritz-sae-00, 28 September 2025.

[I-D.liu-acme-rats] Liu, P., "Automated Certificate Management Environment (ACME) rats Identifier and Challenge Type", Work in Progress, Internet-Draft, draft-liu-acme-rats-02, 24 September 2025.

[GH-ECA-SAE-PROTO] title: "OSS MTI prototype for the ECA & SAE Internet-Drafts." target: https://github.com/eca-sae/prototype-eca-sae/tree/proto-0.1.0 (https://github.com/eca-sae/prototype-eca-sae/tree/proto-0.1.0) date: 2025-09-28

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch,

"Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010.

[RFC8995] Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)", RFC 8995, DOI 10.17487/RFC8995, May 2021.

[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.

[RFC9711] Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", RFC 9711, DOI 10.17487/RFC9711, August 2025.

target: https://spiffe.io/docs/latest/spiffe-about/spiffe-concepts/ (https://spiffe.io/docs/latest/spiffe-about/spiffe-concepts/) date: 2025-01 organization: "CNCF SPIFFE Project"

target: https://spiffe.io/docs/latest/spiffe-about/overview/ (https://spiffe.io/docs/latest/spiffe-about/overview/) date: 2025-01 organization: "CNCF SPIFFE Project"

target: https://spiffe.io/docs/latest/spire-about/spire-concepts/ (https://spiffe.io/docs/latest/spire-about/spire-concepts/) date: 2025-01 organization: "CNCF SPIRE Project"

target: https://spiffe.io/docs/latest/deploying/spire_agent/ (https://spiffe.io/docs/latest/deploying/spire_agent/) date: 2025-01 organization: "CNCF SPIRE Project"

target: https://spiffe.io/docs/latest/deploying/configuring/ (https://spiffe.io/docs/latest/deploying/configuring/) date: 2025-01 organization: "CNCF SPIRE Project"

target: https://spiffe.io/docs/latest/planning/extending/ (https://spiffe.io/docs/latest/planning/extending/) date: 2025-01 organization: "CNCF SPIRE Project"

target: https://github.com/spiffe/spire-plugin-sdk (https://github.com/spiffe/spire-plugin-sdk) date: 2025-08 organization: "CNCF SPIRE Project"

target: https://www.cncf.io/announcements/2022/09/20/spiffe-and-spire-projects-graduate-from-cloud-native-computing-foundation-incubator/ (https://www.cncf.io/announcements/2022/09/20/spiffe-and-spire-projects-graduate-from-cloud-native-computing-foundation-incubator/) date: 2022-09-20 organization: "Cloud Native Computing Foundation"

target: https://developer.hpe.com/blog/spiffe-spire-graduates-enabling-greater-security-solutions/ (https://developer.hpe.com/blog/spiffe-spire-graduates-enabling-greater-security-solutions/) date: 2022-10-24 organization: "HPE Developer Blog"

target: https://www.infoq.com/news/2022/09/spire-graduates-cncf/ (https://www.infoq.com/news/2022/09/spire-graduates-cncf/) date: 2022-09-23 organization: "InfoQ News"

target: https://github.com/keylime/spire-keylime-plugin (https://github.com/keylime/spire-keylime-plugin) date: 2025-01 organization: "Keylime Project"

target: https://keylime.dev/blog/2024/02/07/remote-attestation-blog-part1.html (https://keylime.dev/blog/2024/02/07/remote-attestation-blog-part1.html) date: 2024-02-07 organization: "Keylime Project"

target: https://next.redhat.com/2025/01/24/spiffe-spire-and-keylime-software-identity-based-on-secure-machine-state/ (https://next.redhat.com/2025/01/24/spiffe-spire-and-keylime-software-identity-based-on-secure-machine-state/) date: 2025-01-24 organization: "Red Hat (Emerging Tech)"

target: https://github.com/spiffe/spire/issues/5647
date: 2024-11-12
organization: "SPIRE GitHub Issues"

[AGE] Valsorda, F. and Cartwright-Cox, B., "The age encryption specification", February 2022, https://age-encryption.org/v1 (https://age-encryption.org/v1).

13. Informative References

[I-D.liu-acme-rats]
Liu, P. C., Ounsworth, M., and M. Richardson, "Automated Certificate Management Environment (ACME) rats Identifier and Challenge Type", Work in Progress, Internet-Draft, draft-liu-acme-rats-02, 24 September 2025, <https://datatracker.ietf.org/doc/html/draft-liu-acme-rats-02>.

[I-D.ritz-eca]
Ritz, N., "Ephemeral Compute Attestation (ECA) Protocol", Work in Progress, Internet-Draft, draft-ritz-eca-00, 28 September 2025, <https://datatracker.ietf.org/doc/html/draft-ritz-eca-00>.

[I-D.ritz-sae]
Ritz, N., "Static Artifact Exchange (SAE) Protocol", Work in Progress, Internet-Draft, draft-ritz-sae-00, 28 September 2025, <https://datatracker.ietf.org/doc/html/draft-ritz-sae-00>.

[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/info/rfc2119>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC8995] Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)", RFC 8995, DOI 10.17487/RFC8995, May 2021, <https://www.rfc-editor.org/info/rfc8995>.

[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/info/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/info/rfc9711>.

Author’s Address

Nathanael Ritz
Independent
Email: nathanritz@gmail.com