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

N. Ritz
Independent

28 September 2025

Ephemeral Compute Attestation (ECA) Protocol
draft-ritz-eca-00

Abstract

This document specifies the Ephemeral Compute Attestation (ECA) protocol, which enables ephemeral compute instances to prove their identity without pre-shared operational credentials. ECA uses a three-phase ceremony that cryptographically combines a public Boot Factor (a high-entropy provisioning value), a secret Instance Factor, and a dynamically released Validator Factor to establish attestation evidence. The protocol is transport-agnostic and produces Entity Attestation Tokens (EAT) for consumption by Relying Parties, such as within automated certificate issuance protocols.

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. Scope, Problem Statement and Motivation

ECA profiles the RATS [RFC9334] "passport model". It assumes familiarity with the roles defined in the RATS architecture.

Modern software architecture increasingly relies on ephemeral compute instances, which require a secure and reliable method to bootstrap their identity upon creation. While solutions exist for this problem, they are often tied to a specific vendor's ecosystem or lack the robustness required for certain environments. This creates significant challenges for portability, security, and operational consistency across diverse computing landscapes.

1.1. The Provider-Native Identity Dilemma

Hyperscale cloud providers (e.g., AWS, GCP, Azure) offer Instance Metadata Services (IMDS) that can provide a cryptographically signed token attesting to an instance's identity. This is a mature model for applications developed to run within a single provider's environment. However, IMDS typically relies on HTTP-based access within the instance's network, which can introduce latency in high-throughput scenarios and requires instances to trust the provider's metadata endpoint.

1.2. Limitations of the Current Landscape

Despite the success of provider-native solutions, their approach creates a new set of challenges in a world that is increasingly multi-cloud and security-conscious.

1.2.1. Vendor Lock-In and Portability

Workloads are now frequently designed to be portable across different providers, but their identity bootstrapping mechanisms are not. An application architected to use the AWS Instance Identity Document cannot be moved to GCP, a private cloud, or a bare-metal server without significant re-engineering of its security and trust establishment logic. This friction couples a workload's identity to its location, undermining the core goal of portability.

1.2.2. The Trust Gap in High-Assurance Environments

Provider-native identity mechanisms fundamentally require that the cloud provider itself is a trusted entity. The identity token is issued by the provider's infrastructure and its validity rests on that trust. However, in Confidential Computing and other zero-trust scenarios, the threat model must include a potentially malicious or compromised provider. AMD SEV or Intel TDX, for instance, offer memory encryption and remote attestation, but their reports are tied to specific hardware generations, complicating migration across diverse fleets. In these cases, an identity token issued by the infrastructure is insufficient; trust must be anchored in a separate, verifiable source, such as a hardware root of trust (HRoT). TPM-based systems, such as those in TCG specifications, provide measured boot integrity but often require platform-specific endorsement keys, limiting interoperability in hybrid setups.

1.2.3. Inconsistency in "Alt-Cloud" and On-Premise Environments

For the vast ecosystem of smaller cloud providers, private clouds, and on-premise data centers, a standardized IMDS-like service does not exist. This forces operators into less secure or bespoke bootstrapping patterns, such as injecting pre-shared secrets via user-data. While a practical starting point, this approach re-introduces TOFU risks and creates a broad exposure surface for secrets in logs, state files, and metadata services, compounding operational complexity at scale. Traditional TOFU, as seen in SSH key exchanges, assumes initial connections are secure but can fail in automated deployments where instances are spun up frequently without human oversight. For example, in systems like Kubernetes or OpenStack, user-data injection requires careful configuration management to prevent accidental exposure during cluster scaling or migrations. For concrete patterns addressing these risks, see [I-D.eca-impl] Section 2.1.

1.3. ECA: An Alternative Approach

The Ephemeral Compute Attestation (ECA) protocol is designed to address these limitations directly. It provides a single, open standard that:

* *Decouples Identity from Infrastructure:* ECA establishes instance identity through a transport-agnostic protocol, facilitating portability of workloads across environments.

* *Supports Trust Anchoring:* ECA's design, allows trust to be anchored by a hardware root of trust (HRoT), providing cryptographic proof of identity that remains effective even if the underlying provider is untrusted.

* *Provides a Standard for Various Environments:* ECA offers a standardized bootstrapping mechanism for on-premise, bare-metal, and "alt-cloud" deployments that lack a native identity service.

ECA approaches the bootstrapping problem as a cryptographic challenge based on verifiable proof of factor possession, independent of location.

| *Integration with Existing Frameworks:* ECA design focus was to complement, not replace, existing identity and attestation frameworks. For detailed exploration of how ECA integrates with ACME, BRSKI, SPIFFE/SPIRE, and other systems, see [I-D.eca-impl] Section 4 "Integration with Existing Frameworks".

2. Conventions and Definitions

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.

Boot Factor (BF) An exposure-tolerant, high-entropy value (≥ 128 bits) provisioned during instance creation. The BF value acts as a public challenge token; the protocol security is maintained even if BF may be exposed in logs or metadata.

Instance Factor (IF) A per-instance secret value, known only to the Attester and the Verifier, that is never transmitted over the public channel. The Attester must prove possession of the IF in conjunction with the BF to authenticate. It may be hardware-derived, orchestrator-provided, or artifact-based. For concrete patterns and implementation guidance, see [I-D.eca-impl] Section 2.

Validator Factor (VF) A confidential value generated and released by the Verifier only after successful initial authentication of BF+IF possession. *The VF MUST be bound to the IF* (e.g., VF = SHA-256(seed || IF)). This binding ensures VF secrecy against network attackers, as noted in the formal model (see Appendix A.1).

Joint Possession The cryptographic property where security derives from proving knowledge of multiple factors (BF+VF) rather than secrecy of individual components.

Integrity Hash Beacon (IHB) A SHA-256 binding of BF to IF that

enables exposure-tolerant authentication while preventing pre-computation attacks.

Instance Factor Pattern (IFP) The set of defined methods for sourcing the secret value for Instance Factor (IF). Three patterns are defined: hardware-rooted (Pattern A), orchestrator-provisioned (Pattern B), and artifact-based (Pattern C). For detailed specifications, see [I-D.eca-impl] Section 2.

Entity Attestation Token (EAT) A standardized token format [RFC9711] used to convey attestation evidence in a cryptographically verifiable form.

Exchange Identifier (eca_uuid) A unique identifier for each attestation lifecycle instance, used to construct artifact repository paths and prevent replay attacks.

Artifact Repository A simple, addressable store (e.g., a web server, an object store) where peers can publish and retrieve cryptographic artifacts.

Attestation Ceremony ("ceremony") The RATS architecture [RFC9334] refers to the exchange between participants as "attestation procedures." This document uses "Attestation Ceremony" (or "ceremony") synonymously to describe the complete, multi-phase sequence of cryptographic exchanges required for an attestation. The term "ceremony" is used conventionally throughout this specification.

3. Core Design Principles

Exposure Tolerance: Protocol security is maintained even if the Boot Factor becomes public. This reduces the operational burden of protecting bootstrap secrets in logs, configuration systems, or during provisioning.

Deterministic Identity: All cryptographic keys are derived deterministically from high-entropy factors, ensuring repeatable identity generation without dependence on potentially weak runtime entropy sources.

Transport Agnostic: The protocol’s security is derived from the cryptographic content of exchanged artifacts, not the properties of the transport layer. This allows flexible deployment over any simple retrieval mechanism.

Relationship to Static Artifact Exchange (SAE): While ECA is a transport-agnostic protocol, the Static Artifact Exchange (SAE) I-D.sae-protocol is the recommended transport mechanism. SAE’s static, pull-only model is intentionally minimal to reduce the overall attack surface. This approach reducing common attack surfaces like injection and parser vulnerabilities. By relying on SAE, it re-inforces ECA’s proof-driven design that relies solely from the

cryptographic content of the exchanged artifacts to achieve its security goals, while mitigating risks particularly regarding freshness guarantees (see Appendix A.2.2.).

Privileged Credential Vacuum: The Attester begins its lifecycle with no privileged operational credentials (e.g., API keys, service tokens, or passwords). This operationalizes a "verify-then-trust" model, ensuring that trust is never assumed but must be cryptographically proven through successful attestation. Operational credentials are only delivered after a Relying Party (RP) appraises the Attestation Result (AR) from the Verifier and deems it acceptable. For post-attestation patterns including re-attestation and hierarchical trust, see [I-D.eca-impl] Section 3.

3.1. Protocol Requirements (Normative)

This section defines abstract properties that MUST hold for any conforming implementation. Concrete algorithms and encodings are defined by profiles (see Section 8 "Profiles").

1. *Accept-Once Ceremony*

    * Each attestation ceremony is identified by a globally unique eca_uuid.
    * A Verifier MUST accept each eca_uuid at most once and MUST treat re-observations as replay and abort. Verifiers SHOULD use a persistent store (e.g., a database or file) to track accepted eca_uuid values for at least the expected lifetime of an Attestation Result to prevent replay.

2. *Dual-Channel Binding*

    * The protocol maintains two logically independent channels:

        - an *Attester channel* (artifacts the Attester serves), and
        - a *Verifier channel* (artifacts the Verifier serves).

    * Implementations MUST bind these channels cryptographically so that artifacts from one channel authenticate critical inputs from the other (i.e., no single channel can unilaterally complete the ceremony).

3. *Privileged Credential Vacuum*

    * Any privileged credential or capability MUST NOT be released to the Attester prior to successful appraisal by the Verifier.

   * Success is signaled only by a profile-defined positive terminal artifact; failure is signaled by a profile-defined authenticated failure artifact.

4. *Authenticated Artifacts*

   * Each phase artifact that influences appraisal MUST be integrity-protected under a key bound to ceremony inputs defined by the active profile.
   * Integrity protection MUST cover at minimum: eca_uuid, channel role (Attester/Verifier), and a profile-defined set of claims sufficient for appraisal.

5. *Replay & Freshness*

   * Implementations MUST enforce replay resistance for phase artifacts within the ceremony lifetime.
   * Freshness semantics (e.g., timestamps or nonces) MUST be provided by the active profile and included in the authenticated data.

6. *Termination & State*

   * The Verifier MUST publish a terminal status (success or authenticated failure).
   * After terminalization, subsequent artifacts for the same eca_uuid MUST be ignored.

7. *No Attester-Supplied Trust Pinning*

   * Verifiers MUST NOT establish trust for appraisal by pinning any CA or key material supplied by the Attester.

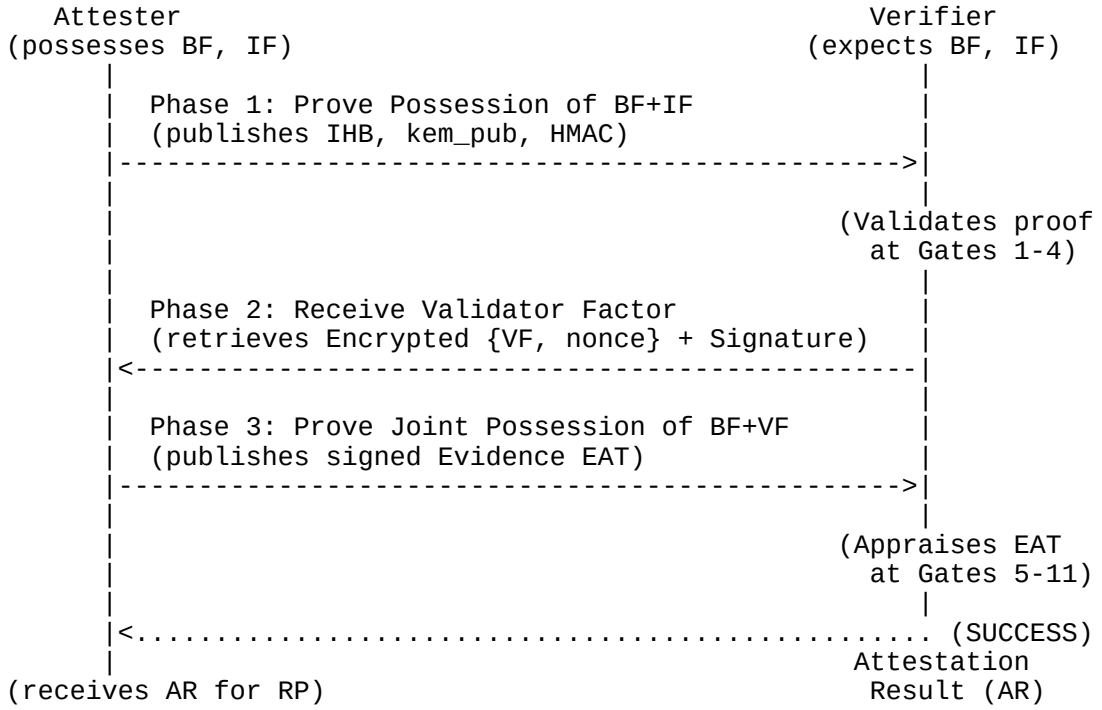
   | Note: The security properties of ceremony isolation depend
   | significantly on the transport mechanism. See Section 6.5 for
   | transport-specific security considerations regarding Verifier
   | key management.

8. *Transport Minimalism*

   * The protocol MUST be realizable over a static artifact repository (poll/pull). Profiles MAY specify additional transports but MUST NOT weaken the requirements above.

4. Protocol Overview

The ECA protocol follows a three-phase ceremony, as illustrated in the figure below. The ceremony begins with the Attester in a privileged credential vacuum, possessing only its initial factors. It concludes with the Verifier producing a signed Attestation Result (AR) upon successful validation, which can then be delivered to the Attester for presentation to Relying Parties (RP).



4.1. Validation Gates

The Verifier enforces a sequence of fail-closed validation gates in a specific order derived from the protocol's formal model. Each gate represents a critical check that must pass before proceeding.

1. *MAC Verification*: Verifies the integrity of the Phase-1 payload using an HMAC tag derived from BF and IF.

   * Failure Action: Immediate termination. Publish error status MAC_INVALID.

2. *Instance Authorization*: Checks if the Attester's identity (e.g., derived from eca_uuid or IF) is authorized to proceed.

   * Failure Action: Immediate termination. Publish error status ID_MISMATCH.

3. *IHB Validation*: Confirms that the received Integrity Hash Beacon (IHB) matches the expected value for the authorized instance.

   * Failure Action: Immediate termination. Publish error status IHB_MISMATCH.

4. *KEM Public Key Match*: Ensures the ephemeral encryption public key in the payload matches the expected key for the session.

   * Failure Action: Immediate termination. Publish error status KEM_MISMATCH.

5. *Evidence Time Window*: Validates that the iat, nbf, and exp claims in the final EAT are within an acceptable time skew (e.g., ツア60 seconds).

   * Failure Action: Immediate termination. Publish error status TIME_EXPIRED.

6. *EAT Schema Compliance*: Checks that the EAT contains all required claims with the correct types and encodings.

   * Failure Action: Immediate termination. Publish error status SCHEMA_ERROR.

7. *EAT Signature*: Verifies the Ed25519 signature on the EAT using the public key derived from BF and VF.

   * Failure Action: Immediate termination. Publish error status SIG_INVALID.

8. *Nonce Match*: Ensures the nonce in the EAT matches the nonce the Verifier issued in Phase 2, proving freshness.

   * Failure Action: Immediate termination. Publish error status NONCE_MISMATCH.

9. *JP Validation*: Verifies the Joint Possession proof, ensuring the final identity key is bound to the ceremony context.

   * Failure Action: Immediate termination. Publish error status KEY_BINDING_INVALID.

10. *PoP Validation*: Verifies the final Proof-of-Possession tag, confirming the Attester’s knowledge of both BF and VF.

    * Failure Action: Immediate termination. Publish error status POP_INVALID.

11. *Identity Uniqueness (Replay)*: Persists the terminal state for the eca_uuid and rejects any future attempts to use it.

    * Failure Action: Immediate termination. Publish error status IDENTITY_REUSE.

These gates align with the formal model’s events (see Appendix A.1): - Gate 8 Nonce Match (per AttesterUsesNonce event). - Gate 9 JP Validation (per VerifierValidatesWithKey event). - Gate 10 PoP Validation (See Section 8) (per VerifierAccepts event).

4.2. Phase 1: Authenticated Channel Setup

* *Attester* generates an ephemeral X25519 keypair deterministically from BF + IF.
* Computes the Integrity Hash Beacon (IHB): IHB = SHA-256(BF || IF).
* Publishes a CBOR payload containing {kem_pub, ihb} and an associated HMAC tag to the repository.
* *Verifier* retrieves the published artifacts and validates them against Gates 1-4.

4.3. Phase 2: Challenge and Validator Factor Release

* *Verifier* generates a fresh VF (劣 · 128 bits) and a 16-byte nonce.
* Encrypts {VF, nonce} using HPKE to the Attester’s ephemeral public key.
* Signs the encrypted payload with its Ed25519 key and publishes it to the repository.
* *Attester* retrieves the published payload, verifies its authenticity, and decrypts the VF.

4.4. Phase 3: Joint Possession Proof

* *Attester* derives a final Ed25519 signing keypair deterministically from BF+VF.
* Creates a signed EAT containing identity claims, the Verifier’s nonce, and a final Proof-of-Possession HMAC.
* Publishes the signed EAT to the repository.
* *Verifier* retrieves the final EAT and validates it against Gates 5-11, yielding an Attestation Result (AR) upon success.

4.5. Key Lifecycle

When using SAE transport I-D.sae-protocol: - Implementations MAY use long-term or ephemeral Verifier keys - Ephemeral per-ceremony keys are RECOMMENDED for operational best practice

When using other transports: - Implementations MUST use ephemeral per-ceremony Verifier keys (see Security Considerations Section 6.5.2 for rationale)

5. Protocol States

State	Description
INIT	New attestation lifecycle initiated.
AWAITING_ATTESTER_PROOF	Awaiting Phase 1 artifacts.
PROVING_TO_ATTESTER	Publishing Phase 2 artifacts.
AWAITING_EVIDENCE	Awaiting Phase 3 artifacts.
VALIDATING	Appraisal of evidence.
SUCCESS	Terminal success state.
FAIL	Terminal failure state.

Table 1

5.1. State Transitions

State	Event	Next State
INIT	New attestation lifecycle.	AWAITING_ATTESTER_PROOF
AWAITING_ATTESTER_PROOF	Phase 1 artifacts retrieved and validated.	PROVING_TO_ATTESTER
PROVING_TO_ATTESTER	Phase 2	AWAITING_EVIDENCE

	artifacts published.	
AWAITING_EVIDENCE	Phase 3 artifacts retrieved.	VALIDATING
VALIDATING	Appraisal results pass.	SUCCESS
Any	Any validation check fails or timeout.	FAIL

Table 2

6. Security Considerations

Trust Boundaries: Without hardware roots of trust, the security scope is limited to passive network observers rather than compromised infrastructure providers. Hardware-rooted Instance Factor Pattern A addresses this limitation. For detailed pattern specifications, see [I-D.eca-impl] Section 2. This hardware-based protection is critical for mitigating State Reveal attacks; a formal analysis confirmed that a compromise of the Attester's software environment can expose the ephemeral decryption keys used in Phase 2, thereby compromising the ceremony's core secrets (see Appendix A.2.3).

Exposure tolerance: The protocol is designed to tolerate incidental exposure of the unique per-use Boot Factor token (BF) (e.g., in control-plane logs), however this tolerance does not replace the need for sound operational hygiene. Operators SHOULD avoid unnecessary public dissemination of BF to minimize attracting targeted attacks. Security is layered; cryptographic strength complements, but does not replace, good operational practices.

Secrets Handling: Derived keys are sensitive cryptographic material. Implementations MUST handle them securely in memory (e.g., using locked memory pages) and explicitly zeroize them after use.

Exposure Tolerance: A core design principle of this protocol is that the Boot Factor (BF) is considered *public information* and its security does not depend on the BF's confidentiality. This exposure tolerance is a deliberate architectural choice that enables powerful, flexible provisioning patterns, such as using a public key from an ACME certificate as a verifiable Boot Factor.

This design places the entire security burden for the initial authentication on the confidentiality of the *Instance Factor (IF)*. The protocol's security is anchored on the Attester proving its knowledge of the secret IF in conjunction with the public BF.

The operational risk is therefore focused on preventing the concurrent exposure of both BF and IF. This risk is tightly time-bounded by two key factors:

1. *The Accept-Once Policy:* The window of vulnerability is extremely short. Once a Verifier consumes an eca_uuid and successfully completes the ceremony, the "accept-once" rule renders any stolen factors for that specific ceremony useless for future impersonation attacks.
2. *Transport Security (SAE):* When using a transport like SAE, an attacker cannot mount a meaningful impersonation attack without gaining write access to the secure artifact repository, which represents a significant and independent security boundary.

Therefore, operational hygiene should focus on protecting the end-to-end provisioning process to ensure the secrecy of the IF until the ceremony is complete, rather than on attempting to hide the public BF.

6.1. Security Properties (Formal Model)

The protocol's security properties have been analyzed using an exploratory ProVerif model. The model positively identifies key security goals such as authentication, freshness, key binding, and confidentiality against a network attacker, assuming a public Boot Factor (BF). For a detailed summary of the formal model, its queries, and the proven properties within the models, see Appendix A.

6.2. Impersonation Risk

The security properties described in Section 6.1 depend on the secrecy of the joint factors. These properties will be compromised if both the Boot Factor (BF) and Instance Factor (IF) are exposed concurrently before a successful ceremony completes. Therefore, BF and IF MUST NOT be transmitted together over an unsecured channel prior to the conclusion of the ceremony. Such exposure would allow

an adversary to intercept the Validator Factor (VF) and perfectly impersonate the intended Attester.

To reduce this risk, operators SHOULD minimize the time window between when an Attester becomes operational with its BF and when a Verifier is available to appraise the Attester’s evidence.

6.3. Threat Models

ECA is designed to address two key threat models: the *Network Attacker* (a Dolev-Yao-style MiTM who controls communication but not participant state) and the *Malicious Provider* (a privileged insider with control-plane access). The analysis from an exploratory ProVerif model suggests that the protocol, as modelled, defeats the Network Attacker through its Phase 1 MAC and joint possession proofs.

The choice of Instance Factor Pattern directly maps to the desired security goals:

* *IFP Patterns B and C* are sufficient to achieve *workload portability and standardization*. They protect against Network Attackers but assume the underlying infrastructure provider is trusted.
* *IFP Pattern A* is designed for *high-assurance and zero-trust environments*. By anchoring the IF in a hardware root of trust (HRoT), it enables strong isolation and is sufficient to mitigate the Malicious Provider threat model.

For detailed pattern specifications and implementation guidance, see [I-D.eca-impl] Section 2.

6.4. Attester State Compromise

The formal model confirms that the protocol cannot maintain secrecy of the Validator Factor (VF) if the Attester’s runtime state is compromised and the ephemeral decryption key is extracted. The confidentiality of VF is critically dependent on the secrecy of the Attester’s ephemeral private decryption key. A formal "State Reveal" analysis was conducted, where the Attester’s ephemeral private key was deliberately leaked to an attacker (see Appendix A.2.3). The model confirmed that this compromise allows a passive network attacker to intercept the Phase 2 ciphertext from the Verifier and successfully decrypt it, thereby revealing the VF.

This result establishes the protocol's security boundary regarding the Attester's runtime state. The only viable mitigation for this threat is the use of IFP Pattern A (hardware-rooted), where the Instance Factor (IF), and by extension all keys derived from it, are protected by a hardware root of trust, making them resilient to software-level compromise.

6.5. Verifier Key Compromise Impact Analysis

Formal analysis (Appendix A.2.2) identified that long-term Verifier keys enable freshness attacks in theory. However, the protocol's cryptographic binding design *ensures* these attacks cannot produce valid authentication, limiting impact to denial of service at worst.

When using SAE transport I-D.sae-protocol, compromise of Verifier signing keys has negligible security impact:

* *Authentication remains secure*: Attackers cannot forge acceptable evidence
* *Protocol integrity maintained*: All validation gates (8-10) will reject evidence derived from attacker-injected values
* *Maximum impact*: Denial of service only

This resilience results from two factors: 1. SAE's pull-only architecture prevents message injection without repository access 2. ECA's cryptographic binding ensures evidence from corrupted ceremonies fails appraisal

Given these mitigations, implementations using SAE MAY use long-term Verifier keys with acceptable security properties, though ephemeral keys remain RECOMMENDED for operational hygiene and ceremony isolation.

Note: Implementations using push-capable or direct-communication transports MUST use ephemeral per-ceremony keys, as these transports enable active injection attacks that compromise freshness.

6.5.1. With SAE Transport (Pull-Only Model)

When using the Static Artifact Exchange (SAE) protocol

* Compromise of Verifier signing keys is limited to denial-of-service impact
* Attackers cannot inject forged Phase 2 artifacts without repository write access
* Evidence produced under attacker-controlled inputs will fail appraisal at legitimate Verifiers (Gates 8-10 will reject the malformed evidence)

This mitigation arises from SAE's architectural properties: - Pull-only communication (no push channel to Attester) - Repository-based artifact exchange with access control - Immutability requirements preventing artifact replacement

6.5.2. With Direct Communication Transports

For implementations using direct peer-to-peer communication or push-capable transports, the formal model (Appendix A.2.2) demonstrates that:

* Long-term Verifier keys enable injection of (VF', nonce') pairs.
* This breaks the formal Freshness property.
* While authentication still fails (corrupted Evidence is rejected), the DoS potential justifies mandatory ephemeral keys.

Therefore, ephemeral per-ceremony keys are normatively mandated (MUST) when not using SAE I-D.sae-protocol or equivalent pull-only, repository-based transports.

6.5.3. Recommendation Rationale

While SAE mitigates the immediate security impact of key compromise, ephemeral keys remain RECOMMENDED for all implementations because they provide:

* Ceremony isolation (compromise affects only single attestation)
* Operational hygiene through regular key rotation
* Clear security boundaries for audit and analysis
* Future-proofing against transport mechanism changes

7. Non-Goals

ECA explicitly does not attempt to address several related but distinct problems:

Software-Based Mitigation of Hypervisor Threats: 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, including protections against Attester state compromise (see Section 6.4).

Replacement for Single-Cloud IMDS: ECA is not intended to replace provider-native IMDS for simple workloads operating within a single, trusted cloud environment. For such use cases, IMDS provides a simpler, adequate solution. ECA's value is realized in multi-cloud, high-assurance, or non-IMDS environments.

Infrastructure Trust Bootstrapping: ECA assumes operational mechanisms exist for manifest distribution, verifier discovery, and PKI infrastructure. It integrates with existing trust foundations rather than replacing them.

Identity Framework Replacement: ECA complements rather than competes with systems like SPIFFE/SPIRE, potentially serving as a high-assurance node attestor for existing identity frameworks. For detailed integration patterns, see [I-D.eca-impl] Section 4.

Manufacturer Provenance: ECA does not provide supply-chain attestation or manufacturer-anchored trust. ECA handles runtime attestation for transient instances at the software layer.

Real-time Performance Optimization: The asynchronous design prioritizes security and reliability over minimal latency. Preliminary efforts suggest total latency of less than 2 seconds using SAE for VM attestation, which is minimal compared to standard cloud VM startup time. Sub-second attestation is not a primary goal, however feedback for secure optimizations are welcomed.

8. Profiles (Normative)

This document defines the protocol abstractly. Concrete cryptographic mechanisms are supplied by profiles. A conforming implementation MUST implement at least one profile, and any chosen profile MUST preserve all requirements in Section 3.1.

| Note: No MTI Algorithms in this Revision. This -00 revision does
| not define mandatory-to-implement (MTI) primitives. A reference
| profile ("ECA-VM-v1") is specified in [I-D.eca-impl] Section 6 as
| a candidate to enable experimentation and interop with the
| prototype.

Key Separation (Architecture requirement): Regardless of profile, implementations MUST maintain strict separation between: - Phase 2 encryption keys (used by the Verifier to release VF to the Attester), and - Phase 3 identity/signing keys (used by the Attester to sign Evidence/EAT).

Profiles typically achieve separation via domain-separated KDF invocations; however, any mechanism that guarantees computational unlinkability between Phase 2 and Phase 3 key material is acceptable, provided Section 3 invariants remain intact.

8.1. Proof-of-Possession (PoP) Construction

A profile MUST provide a PoP mechanism that proves joint-possession of both factors used across the ceremony and binds the result to the session context. At minimum, the PoP's authenticated input MUST cover:

* eca_uuid,
* the Integrity Hash Beacon (IHB) or an equivalent BF+IF binding,
* the Attester's Phase-3 signing public key, and
* the Verifier's freshness input (e.g., vnonce).

The PoP output MUST be verifiable by the Verifier without additional round trips and MUST be integrity-protected under a key that is infeasible to compute without both factors required by the active profile.

9. EAT profiles

9.1. Evidence Claims

Claim	EAT Key	Value Type	M/O	Description
ECA UUID	2 (sub)	tstr	M	The unique eca_uuid for the attestation lifecycle. The value of this claim MUST be the eca_uuid.
Expiration	4 (exp)	int	M	NumericDate (epoch seconds). MUST be encoded as a 64-bit unsigned integer.
Not Before	5 (nbf)	int	M	NumericDate (epoch seconds). MUST be encoded as a 64-bit unsigned integer.
Issued At	6 (iat)	int	M	NumericDate (epoch seconds). MUST be encoded as a 64-bit unsigned integer.
Verifier Nonce	10 (nonce)	tstr	M	Verifier-issued vnonce (*base64url*, unpadded) representing exactly 16 bytes of entropy (typically 22 chars).
ECA Identity	256 (EUID)	tstr	M	eca_attester_id = hex SHA-256 of the Ed25519 public key used to sign

				this EAT.
EAT Profile	265	tstr	M	urn:ietf:params:eat:profile:eca-v1.
Measurements	273	tstr	M	Integrity Hash Beacon (IHB) (*lowercase hex*).
PoP	274 (PoP)	tstr	M	Final Proof of Possession tag (*base64url*, unpadded) computed as defined by the active profile.
JP Proof	276	tstr	M	Joint Possession proof (*lowercase hex*), binding the final identity to the ceremony.
Intended Use	275	tstr	M	The intended use of the EAT (e.g., attestation, enrollment credential binding).

Table 3

Values marked "tstr" that carry binary material (e.g., nonces, tags) MUST specify their encoding. In the ECA-VM-v1 profile (see [I-D.eca-impl] Section 6), nonce and PoP are base64url (unpadded); EUID, Measurements, and JP Proof are lowercase hex.

9.2. Attestation Results

Claim	Key	Value Type	Description
Issuer	1	tstr	An identifier for the Verifier that produced the result.
Subject	2	tstr	The eca_attester_id identity of the instance that was successfully attested.
Expiration	4 (exp)	int	OPTIONAL. NumericDate defining the AR’s validity period.
Not Before	5 (nbf)	int	OPTIONAL. NumericDate defining the AR’s validity period.
Issued At	6	int	NumericDate (epoch seconds) of the successful validation.

JWT ID	7	tstr	The unique eca_uuid of the attestation lifecycle to prevent replay.
Key ID	-1 (kid)	bstr	OPTIONAL. The hash of the Verifier's public key used to sign the AR.
Status	-262148	tstr	The outcome of the attestation. MUST be urn:ietf:params:rats:status:success.

Table 4

For failures, the AR payload SHOULD follow the same structure but with a status of urn:ietf:params:rats:status:failure and an additional error_code claim (e.g., -262149 as a tstr) containing the authenticated error. Relying Parties consuming the AR MUST validate the nbf and exp claims to ensure the AR is within its validity period.

10. Transport Considerations

The ECA protocol is transport-agnostic. It requires only that peers have a mechanism to publish and retrieve immutable cryptographic artifacts from a pre-defined Artifact Repository. The Static Artifact Exchange (SAE) protocol I-D.sae-protocol is specified as the recommended pattern to fulfill this requirement. SAE's static, "publish-then-poll" model is intentionally chosen to minimize the attack surface associated with traditional, dynamic APIs. By avoiding direct request processing, it eliminates entire classes of vulnerabilities like injection and parser flaws, ensuring that protocol security is derived from the cryptographic content of the artifacts alone.

11. Operational Considerations

For detailed operational guidance including scalability, time synchronization, addressing complexity, and provisioning patterns, see [I-D.eca-impl] Section 5.

12. IANA Considerations

IANA is requested to register:

12.1. EAT Profile

* *Profile*: urn:ietf:params:eat:profile:eca-v1

* *ECA Attestation Result Claims:* IANA is requested to establish a registry for ECA Attestation Result Claims as outlined in Section 9.2. This registry defines the claims used within the signed CBOR object that constitutes an Attestation Result.

12.2. Registries

12.2.1. ECA Error Codes Registry

This registry defines application-specific error codes that are used in addition to the base error codes defined in I-D.sae-protocol. The Canonical Content string defined here MUST be used as the input to the HMAC-SHA256 function when generating an error signal, as specified by the SAE protocol.

Code	Canonical Content (UTF-8)	Gate	Description
MAC_INVALID	MAC_INVALID	1	Provided MAC was invalid.
ID_MISMATCH	ID_MISMATCH	2	Provided instance identity was invalid.
IHB_MISMATCH	IHB_MISMATCH	3	Recomputed IHB did not match expected value.
KEM_MISMATCH	KEM_MISMATCH	4	Did not get expected KEM key for the session.
TIME_EXPIRED	TIME_EXPIRED	5	Evidence timestamp was outside valid time window.
SCHEMA_ERROR	SCHEMA_ERROR	6	Attestation

			token failed schema validation.
SIG_INVALID	SIG_INVALID	7	Attestation token signature failed.
NONCE_MISMATCH	NONCE_MISMATCH	8	Nonce in the EAT did not match the issued nonce.
KEY_BINDING_INVALID	KEY_BINDING_INVALID	9	The key used for validation is not bound to the session’s Boot Factor.
POP_INVALID	POP_INVALID	10	The PoP tag was invalid.
IDENTITY_REUSE	IDENTITY_REUSE	11	Attempt to reassign an existing identity.
PUBLISHER_INVALID	PUBLISHER_INVALID	-	Attester artifacts were observed at a repository not hosted by the Attester.
TIMEOUT_PHASE1	TIMEOUT_PHASE1	-	Attester failed to publish Phase 1 artifacts within timeout
TIMEOUT_PHASE2	TIMEOUT_PHASE2	-	Attester failed to publish Phase 2 artifacts

			within timeout
TRANSPORT_ERROR	TRANSPORT_ERROR	-	Underlying transport protocol error

Table 5

13. 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.

14. References

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

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

[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 (https://www.rfc-editor.org/info/rfc2119).

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, https://www.rfc-editor.org/info/rfc4648 (https://www.rfc-editor.org/info/rfc4648).

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, https://www.rfc-editor.org/info/rfc5869 (https://www.rfc-editor.org/info/rfc5869).

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, https://www.rfc-editor.org/info/rfc6234 (https://www.rfc-editor.org/info/rfc6234).

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, https://www.rfc-editor.org/info/rfc7519 (https://www.rfc-editor.org/info/rfc7519).

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, https://www.rfc-editor.org/info/rfc7748 (https://www.rfc-editor.org/info/rfc7748).

[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/info/rfc8032 (https://www.rfc-editor.org/info/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/info/rfc8174 (https://www.rfc-editor.org/info/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/info/rfc8392 (https://www.rfc-editor.org/info/rfc8392).

[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/info/rfc8949 (https://www.rfc-editor.org/info/rfc8949).

[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 (https://www.rfc-editor.org/info/rfc8995).

[RFC9180] Barnes, R., Bhargavan, K., Lipp, B., and C. Wood, "Hybrid Public Key Encryption", RFC 9180, DOI 10.17487/RFC9180, February 2022, https://www.rfc-editor.org/info/rfc9180 (https://www.rfc-editor.org/info/rfc9180).

[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 (https://www.rfc-editor.org/info/rfc9334).

[RFC9711] Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", RFC 9711, DOI 10.17487/RFC9711, August 2025, https://www.rfc-editor.org/info/rfc9711 (https://www.rfc-editor.org/info/rfc9711).

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

[ECA-PV-BL-MODEL] title: "ECA ProVerif: Baseline Happy Path (model)" target: https://github.com/eca-sae/internet-drafts-eca-sae/blob/pv0.3.0/formal-model/eca-pv.baseline.happy-path.model.txt (https://github.com/eca-sae/internet-drafts-eca-sae/blob/pv0.3.0/formal-model/eca-pv.baseline.happy-path.model.txt) date: 2025-09-26

[ECA-PV-BL-PROOF] title: "ECA ProVerif: Baseline Happy Path (proof)"

target: https://github.com/eca-sae/internet-drafts-eca-sae/blob/pv0.3.0/formal-model/eca-pv.baseline.happy-path.proof.txt (https://github.com/eca-sae/internet-drafts-eca-sae/blob/pv0.3.0/formal-model/eca-pv.baseline.happy-path.proof.txt) date: 2025-09-26

[ECA-PV-AT-FS-MODEL] title: "ECA ProVerif: Advanced Threats 窶 Forward Secrecy (model)" target: https://github.com/eca-sae/internet-drafts-eca-sae/blob/pv0.3.0/formal-model/eca-pv.advanced-threat.forward-secrecy.model.txt (https://github.com/eca-sae/internet-drafts-eca-sae/blob/pv0.3.0/formal-model/eca-pv.advanced-threat.forward-secrecy.model.txt) date: 2025-09-26

[ECA-PV-AT-SR-MODEL] title: "ECA ProVerif: Advanced Threats State Reveal (model)" target: https://github.com/eca-sae/internet-drafts-eca-sae/blob/pv0.3.0/formal-model/eca-pv.advanced-threat.state-reveal.model.txt (https://github.com/eca-sae/internet-drafts-eca-sae/blob/pv0.3.0/formal-model/eca-pv.advanced-threat.state-reveal.model.txt) date: 2025-09-26

15. Informative References

[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>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[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/info/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/info/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/info/rfc8392>.

[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/info/rfc8949>.

[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>.

[RFC9180] Barnes, R., Bhargavan, K., Lipp, B., and C. Wood, "Hybrid Public Key Encryption", RFC 9180, DOI 10.17487/RFC9180, February 2022, <https://www.rfc-editor.org/info/rfc9180>.

[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>.

Appendix A. Formal Modelling (Informative)

The protocol's security properties were analyzed using an exploratory formal model in ProVerif [ECA-PV-BL-MODEL]. The model assumes a powerful Dolev-Yao network attacker who can intercept, modify, and inject messages. It also correctly models the Boot Factor (BF) as public knowledge from the start, as per the protocol's "exposure tolerance" principle (Section 3).

The analysis was conducted in two parts: verification of the core security properties against a network attacker, and an analysis of the protocol's behavior under specific key compromise scenarios to define its security boundaries.

A.1. Core Security Properties (Baseline Model)

In the baseline model, all core security goals were successfully shown to hold against a network attacker [ECA-PV-BL-PROOF].

Property	ProVerif Query	Result	Interpretation
Authentication	inj-event(VerifierAccepts(...)) ==> inj-event(AttesterInitiates(...))	*True*	The Verifier only accepts an attestation if a unique Attester legitimately initiated it. This prevents an attacker from impersonating the Attester.
Freshness	event(AttesterUsesNonce(n)) ==> event(VerifierGeneratesNonce(n))	*True*	The Attester will only use a nonce that was genuinely generated by the Verifier for that ceremony. This property is the basis for *Gate 8 (Nonce Match)* and prevents replay attacks.
Key Binding	event(VerifierValidatesWithKey(pk)) ==> event(AttesterPresentsKey(pk))	*True*	The final identity key that the Verifier checks is unambiguously bound to the

			Attester that participated in the ceremony, validating *Gate 9 (JP Validation)*.
Confidentiality	not (event(VFReleased(vf)) && attacker(vf))	*True*	The secret ValidatorFactor (VF) is never revealed to a network attacker, satisfying a fundamental security goal of the protocol.

Table 6

A.2. Boundary Analysis (Advanced Threat Models)

Additional tests were performed to formally define the protocol's security boundaries under specific compromise scenarios.

A.2.1. Key Compromise Impersonation (KCI)

A test was conducted where an attacker compromises an InstanceFactor (IF) from one ceremony and attempts to impersonate a Verifier in a different ceremony [ECA-PV-BL-MODEL]. The model indicated this attack is not possible [ECA-PV-BL-PROOF]. The KCI security query passed, confirming that compromising a secondary factor (IF) does not allow an attacker to forge messages from a primary party (the Verifier), as they still lack the Verifier's private signing key.

A.2.2. Verifier Key Compromise

A test was conducted modeling a compromised long-term Verifier signing key [ECA-PV-AT-FS-MODEL]:

* *Result*: The model demonstrated that an attacker can inject arbitrary (VF', nonce') pairs, breaking the Freshness property (event(AttesterUsesNonce(n)) ==> event(VerifierGeneratesNonce(n)) was *False*).

* *Interpretation*: While the formal model identifies a freshness violation, the protocol's cryptographic design ensures this only enables denial of service, not authentication bypass:

* The attacker can cause the Attester to derive keys from (BF, VF')

* However, the resulting Evidence will contain:

  - Wrong nonce (fails Gate 8)
  - Wrong JP proof (fails Gate 9)
  - Wrong PoP tag (fails Gate 10)
  - No correctly implemented Verifier should accept this Evidence

Furthermore, when using SAE transport I-D.sae-protocol, even this DoS attack becomes infeasible without repository write access, as noted in Section 6.5.1.

* *Mitigation*: This analysis provides the formal rationale for:
  - Section 6.5.2's requirement for ephemeral keys with push-capable transports
  - Section 6.5.1's relaxed guidance when using SAE transport

A.2.3. Attester State Reveal

A test was conducted modeling a compromised Attester whose ephemeral private decryption key is leaked [ECA-PV-AT-SR-MODEL].

* *Result*: The model demonstrated that this allows a passive attacker to decrypt the Phase 2 ciphertext and steal the ValidatorFactor (VF) (not (event(VFReleased(vf)) && attacker(vf)) was *False*).

* *Interpretation*: This result formally establishes the security boundary discussed in Section 6.4

* *Mitigation*: This analysis provides the formal rationale for hardware-rooted Instance Factor Pattern A when the threat model must assume compromise of the underlying provisioning platform. For pattern specifications, see [I-D.eca-impl] Section 2.

Author's Address

Nathanael Ritz
Independent
Email: nathanritz@gmail.com