Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 3 April 2026

N. Ritz
Independent

30 September 2025

Static Artifact Exchange (SAE) Protocol
draft-ritz-sae-01

Abstract

This document specifies the Static Artifact Exchange (SAE) protocol, an asynchronous transport pattern for exchanging cryptographic artifacts between two parties via a shared, stateless repository. SAE uses a "publish-then-poll," pull-only communication model where peers use the presence and size of immutable artifacts to coordinate a sequenced exchange. By enforcing a strict set of invariants, including a prohibition on parsing arbitrary content to drive the transport’s state machine, SAE is designed to minimize common attack surfaces like injection and parser vulnerabilities. The pattern is transport-agnostic and is intended for use by higher-level cryptographic protocols, like Ephemeral Compute Attestation (ECA), that require a secure, minimal channel.

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 3 April 2026.

Copyright Notice

Table of Contents

1. Introduction

Many cryptographic protocols require a coordinated, multi-phase exchange of artifacts. Traditional request-response patterns often introduce security risks at the transport layer, including complex state management, parser vulnerabilities, and injection attacks.

This document specifies the Static Artifact Exchange (SAE) protocol, an alternative model designed for security and simplicity. In SAE, parties do not communicate directly but interact asynchronously through a simple, stateless repository. One party publishes a set of immutable, pre-computed artifacts and then a status indicator to signal completion. The other party polls for this status indicator and, upon observing it, retrieves the artifacts.

This "publish-then-poll" pattern, governed by a strict set of invariants, reduces the need for active listeners or dynamic request processing, limiting common attack surfaces. The security of the higher-level protocol relies on the cryptographic validity of the artifacts themselves, while the SAE transport provides a hardened, minimal-surface communication channel.

SAE is intended as a reusable transport pattern for higher-level protocols, motivated by the requirements of Ephemeral Compute Attestation (ECA) [I-D.ritz-eca].

2. Terminology and Core Concepts

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.

Peer
: An entity participating in the SAE protocol.

Artifact
: An immutable blob of data published by a peer.

Status Indicator
: A signal indicating the completion of a phase. A zero-byte artifact indicates success, while a non-zero-byte artifact contains an authenticated and encrypted error signal.

Repository
: A durable, addressable, and immutable store for artifacts.

Exchange Identifier
: A unique identifier for a specific exchange instance, used to construct artifact paths.

3. Protocol Requirements (Normative)

SAE implementations MUST satisfy the following invariants:

1. *Static Artifact Model*: All communication MUST conform to a strict artifact exchange model where both peers operate as passive repositories of pre-computed, immutable artifacts.

2. *Pull-Only Communication*: Peers MUST NOT push data. All artifact retrieval MUST be initiated by the consuming peer through polling.

3. *Phased Atomic Progression*: All artifacts for a phase MUST be published before its corresponding status indicator is published.

4. *Immutability*: Once a status indicator for a phase is published, all associated artifacts for that phase MUST NOT be changed or removed.

5. *Prohibited Processing (Transport Layer)*: The SAE transport-level state machine MUST be driven solely by the presence and size of artifacts. A non-zero size for a status indicator MUST be treated by the SAE layer as a terminal failure, ending the exchange. This invariant prevents the SAE layer from parsing arbitrary or variable-length content to determine its outcome. This rule does *not* prohibit the higher-level application protocol (e.g., ECA) from performing its own content-based validation, such as verifying a digital signature, as part of its own application-level logic after retrieving an artifact.

6. *Bounded Polling*: Polling for status indicators MUST use exponential backoff.

7. *Transport Simplicity*: The protocol logic MUST remain independent of transport-level features beyond basic artifact retrieval (e.g., HTTP GET/HEAD) and presence signaling (e.g., HTTP 200/404).

8. *Repository Consistency*: Any repository used to host artifacts MUST provide strong read-after-write consistency.

9. *Resilient Artifact Retrieval*: The peer’s retrieval logic MUST be resilient to transient transport- or storage-layer delays.

4. Protocol Mechanics

4.1. Artifact Repository Structure

Each peer MUST organize its repository using a consistent path structure based on the Exchange Identifier.

Recommended Structure:

/<exchange_id>/<artifact_name>
/<exchange_id>/<phase_name>.status

4.2. Phase Coordination

The protocol progresses using a "publish-then-poll" pattern:

1. *Publication*: A peer publishes all required artifacts for the current phase.

2. *Signaling*: After publication, the peer publishes the status indicator.

   * *Success*: The status indicator MUST be a zero-byte artifact.
   * *Failure*: The status indicator MUST be a non-zero-byte artifact containing an authenticated and encrypted error signal as defined in 則 4.4 (#sec-error-signaling).

3. *Polling*: The counterpart peer polls for the status indicator.

4. *Retrieval and Interpretation*: Upon detecting the status indicator:

   * If the size is zero, the phase was successful; proceed to retrieve phase artifacts.
   * If the size is greater than zero, the SAE exchange has failed and MUST be terminated. Diagnosing the specific error by decrypting the artifact’s content is an OPTIONAL step for the higher-level application.

4.3. Error Signaling

When a peer encounters an error, it MUST publish a status indicator containing a fixed-length, authenticated ciphertext. This is generated by first hashing the canonical error string and then encrypting that hash using an AEAD scheme.

* *Key (K)*: A secret key derived from a value shared between the peers out-of-band or established by the higher-layer protocol for this specific exchange_id.
* *Plaintext*: The raw binary output of SHA-256(canonical_error_string).
* *AEAD*: An AEAD scheme such as AES-256-GCM is RECOMMENDED, with a fixed, zero-byte AAD and a randomly generated 12-byte nonce prepended to the ciphertext.

This method conceals the error from passive observers and provides integrity, while allowing the recipient to perform an efficient O(1) lookup by hashing known error strings and comparing them to the decrypted hash.

5. Security Considerations

5.1. Elimination of Processing Vulnerabilities

The security benefits of SAE are not inherent to the static artifact model itself, but are a direct result of the strict invariants placed upon it. Specifically, the *Prohibited Processing (Transport Layer)* invariant, which forbids the SAE state machine from parsing content to determine its outcome, is what eliminates common attack vectors like parser vulnerabilities and injection flaws at the transport level.

5.2. Prevention of Race Conditions

The atomic "publish-then-signal" model prevents Time-of-Check-to-Time- of-Use (TOCTOU) vulnerabilities.

5.3. Resilience to Replay Attacks

Replay protection across exchanges MUST be provided by the higher-layer protocol (e.g., ECA's accept-once semantics for eca_uuid).

6. IANA Considerations

6.1. SAE Error Codes Registry

IANA is requested to establish a registry for SAE Error Codes. This registry defines the error code identifier and the canonical string that MUST be used as the input to the SHA-256 hash function when generating an error signal.

Code	Canonical Content (UTF-8)
BAD_REQUEST	BAD_REQUEST
UNAUTHORIZED	UNAUTHORIZED
FORBIDDEN	FORBIDDEN
CONFLICT	CONFLICT
GATEWAY_TIMEOUT	GATEWAY_TIMEOUT

Table 1

7. Normative References

[I-D.ritz-eca] Ritz, N., "Ephemeral Compute Attestation (ECA)

Protocol", Work in Progress, draf-ritz-eca-00, 16 September 2025.
[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119.
[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).
[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174.
[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018.
[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022.

8. Acknowledgements

The author wishes to thank Eric Rescorla for his thorough review of the initial draft. His expert feedback led to significant improvements in the protocol’s security claims, the clarification of a key normative invariant, and a more robust and efficient error signaling mechanism.

9. Informative References

[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>.

[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>.

[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>.

[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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

Appendix A. HTTPS Profile (Normative)

This section provides a normative example of how the abstract SAE protocol can be implemented using standard HTTPS.

A.1. Example Exchange

Error example: If B encounters a timeout, it publishes an authenticated and encrypted error signal.

```
# B derives the AEAD key (K) and a nonce
# B computes the plaintext: HASH = SHA-256("GATEWAY_TIMEOUT")
# B encrypts the hash: C = AEAD.Seal(K, nonce, HASH, nil)
# B constructs the payload: nonce || C

B: PUT /exchange-12345/response.status (content: <nonce-and-ciphertext>)
A: HEAD /exchange-12345/response.status (polling until 200 OK)
A: Observes Content-Length > 0 // Exchange is dead
A: GET /exchange-12345/response.status (reads payload) // OPTIONAL
A: Parses nonce and ciphertext from payload
A: Decrypts hash: HASH’ = AEAD.Open(K, nonce, C, nil)
A: Looks up HASH’ in a precomputed map of known error hashes
A: Finds match for SHA-256("GATEWAY_TIMEOUT") -> aborts exchange
```

Appendix B. Example Implementations (Informative)

B.1. Generating an Error Signal (Bash)

```
#!/bin/bash
ERROR_CODE="GATEWAY_TIMEOUT"
AEAD_KEY_HEX="deadbeef..." # 32-byte key, securely provided
NONCE_HEX=$(openssl rand -hex 12)

# Compute the hash of the error string
ERROR_HASH=$(printf "%s" "$ERROR_CODE" | sha256sum | awk '{print $1}')

# Encrypt the hash using AES-256-GCM and get ciphertext + tag
# Note: This is a simplified example. Real implementations should
# handle outputs carefully.
ENCRYPTED_OUTPUT=$(echo -n "$ERROR_HASH" | openssl enc -aes-256-gcm \
  -K "$AEAD_KEY_HEX" -iv "$NONCE_HEX" -A "" | xxd -p -c 256)
CIPHERTEXT=${ENCRYPTED_OUTPUT::-32}
TAG=${ENCRYPTED_OUTPUT: -32}

# The final payload is nonce || ciphertext || tag
printf "%s%s%s" "$NONCE_HEX" "$CIPHERTEXT" "$TAG" > error.status
```

B.2. Verifying an Error Signal (Bash)

```
#!/bin/bash
RECEIVED_PAYLOAD=$(cat error.status | xxd -p -c 256)
AEAD_KEY_HEX="deadbeef..." # 32-byte key, securely provided

# Parse the payload
NONCE_HEX=${RECEIVED_PAYLOAD:0:24}
CIPHERTEXT_HEX=${RECEIVED_PAYLOAD:24:-32}
TAG_HEX=${RECEIVED_PAYLOAD: -32}

# Attempt to decrypt
DECRYPTED_HASH=$(printf "%s" "$CIPHERTEXT_HEX" | xxd -r -p | openssl enc -d -aes-256-g
cm -K "$AEAD_KEY_HEX" -iv "$NONCE_HEX" -A "" -T "$TAG_HEX")

# Precompute known error hashes
declare -A KNOWN_ERROR_HASHES
KNOWN_ERROR_HASHES[$(printf "%s" "GATEWAY_TIMEOUT" | sha256sum | awk '{print $1}')]="G
ATEWAY_TIMEOUT"
# ... add other known errors ...

# Check if the decrypted hash is a known error
if [[ -v KNOWN_ERROR_HASHES[$DECRYPTED_HASH] ]]; then
  echo "VERIFIED: The error code is '${KNOWN_ERROR_HASHES[$DECRYPTED_HASH]}'."
else
  echo "UNKNOWN_ERROR or AUTHENTICATION_FAILURE."
fi
```

Appendix C. Changes from -00

* Refined the security claims in the Security Considerations section to more precisely attribute the protocol's security benefits to its strict invariants rather than the static artifact model itself.
* Clarified the normative language for the "Prohibited Processing" invariant to resolve an ambiguity. The rule now explicitly applies to the SAE transport-level state machine and does not prohibit content validation by the higher-level application protocol.
* Replaced the HMAC-based error signaling mechanism with a more efficient AEAD-based design, based on feedback from Eric Rescorla. This improves performance and conceals the error type from passive observers.
* General editorial and language improvements for clarity.

Author's Address

Nathanael Ritz
Independent
Email: nathanritz@gmail.com