Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 6 December 2026

V. Vasylenko
4 June 2026

End-to-End Encryption for HTTP APIs Using X25519 and AES-GCM
draft-vasylenko-e2ee-http-00

Abstract

This document specifies an application-layer end-to-end encryption (E2EE) scheme for HTTP APIs. The scheme uses X25519 Elliptic Curve Diffie-Hellman (ECDH) for key agreement, HKDF-SHA256 for key derivation, and AES-GCM (with 128-, 192-, or 256-bit keys) for authenticated encryption of request and response payloads. Server public keys are discovered through a Well-Known URI and authenticated either by TLS or by a stronger out-of-band trust mechanism, depending on the deployment threat model. The scheme is designed to provide confidentiality and integrity of payloads independent of, and in addition to, transport-layer security such as TLS. It provides replay protection and key rotation.

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 6 December 2026.

Copyright Notice

Table of Contents

1. Introduction

Transport Layer Security (TLS) [RFC8446] protects HTTP [RFC9110] traffic between two endpoints that share a direct connection. In many modern deployments, however, request and response payloads traverse intermediaries such as reverse proxies, load balancers, content delivery networks, and API gateways. Each intermediary terminates TLS and observes plaintext, which expands the attack surface and weakens the confidentiality guarantee provided to end users.

This document specifies a payload-level end-to-end encryption (E2EE) scheme for HTTP APIs. The scheme protects the request and response payload between the originating client and the terminating application server. Intermediaries that handle the HTTP message can route, log metadata, and apply policy. They cannot read or modify the protected payload when clients authenticate the server key set according to the deployment threat model described in Section 11.

The scheme combines:

* X25519 [RFC7748] for Elliptic Curve Diffie-Hellman (ECDH) key agreement.

* HKDF with SHA-256 [RFC5869] for deriving symmetric keys from the shared secret.

* AES-GCM [NIST-SP-800-38D] with 128-, 192-, or 256-bit keys for authenticated encryption of payloads.

* A Well-Known URI [RFC8615] for publishing the server's static or rotating public key set.

The scheme does not replace TLS; it is intended to be deployed on top of TLS so that transport metadata such as the Host header and request path remain protected on the wire.

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.

This document uses the following terms:

Client: The party that initiates an HTTP request and that owns an ephemeral X25519 key pair for the duration of a session.

Server: The party that terminates the HTTP request, owns one or more X25519 key pairs whose public components are published, and processes the decrypted payload.

Encryption Key (EK): A direction-specific symmetric key of 128, 192, or 256 bits derived from the X25519 shared secret using HKDF-SHA256 and used as the AES-GCM key. The length is determined by the negotiated AEAD identifier (see Section 5).

AEAD Identifier (AEAD): A short string naming the AES-GCM variant in use. One of "AES-128-GCM", "AES-192-GCM", or "AES-256-GCM".

Key Identifier (KID): A short, opaque label that identifies one public key in the server’s published key set.

Fingerprint: A base64url-encoded prefix of the SHA-256 hash of a public key, intended for human-readable verification.

3. Protocol Overview

The protocol has three phases: key discovery, key agreement, and authenticated message exchange.

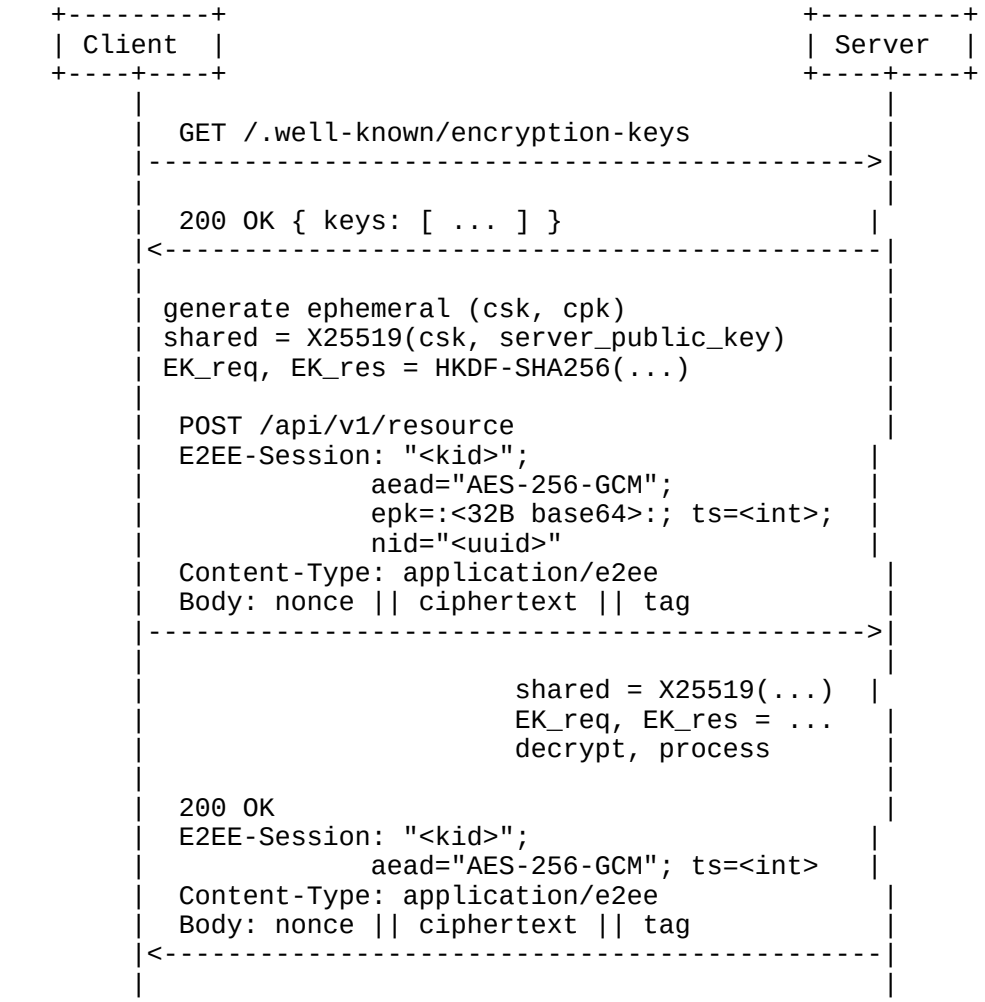

Figure 1: E2EE HTTP message flow

4. Key Discovery

4.1. Well-Known URI

A server that supports this scheme MUST publish its public key set at the Well-Known URI [RFC8615]:

/.well-known/encryption-keys

The resource MUST be served over HTTPS and MUST be retrievable with an HTTP GET request. The response MUST have media type application/json.

4.2. Key Set Document

The response body is a JSON object with the following members:

issuer: A string identifying the server origin. The value MUST be an HTTPS origin and MUST match the origin from which the key set was retrieved unless the client has explicit out-of-band configuration allowing another issuer. REQUIRED.

keys: A JSON array of one or more key objects, ordered from most-preferred to least-preferred. REQUIRED.

Each key object has the following members:

kid: A short opaque string identifying this key within the set. REQUIRED. KIDs MUST be unique within a single key set. The value MUST contain between 1 and 128 characters and MUST match the regular expression ^[A-Za-z0-9._~-]+$. KID comparison is case-sensitive.

alg: The key agreement algorithm. MUST be the string "X25519" for this version of the protocol. REQUIRED.

aeads: A non-empty JSON array of AEAD Identifier strings (see Section 5) that the server is willing to accept under this key, in order of server preference. REQUIRED.

public_key: The 32-byte X25519 public key, base64url-encoded [RFC4648] without padding. REQUIRED.

fingerprint: The base64url-encoded [RFC4648] first 16 bytes of the SHA-256 hash of the raw 32-byte public key. RECOMMENDED.

not_before: An RFC 3339 [RFC3339] date-time value before which the key MUST NOT be used. OPTIONAL.

not_after: An RFC 3339 [RFC3339] date-time value after which the key MUST NOT be used. REQUIRED.

max_skew: A non-negative integer specifying the maximum acceptable absolute difference, in seconds, between the request ts parameter and the server clock. This value bounds the timestamp check in addition to the key validity window and determines the minimum replay-cache retention period (see Section 11). REQUIRED.

Clients MUST treat a key object that is missing a required member, has a member of the wrong JSON type, or contains an invalid public_key, not_before, not_after, or max_skew value as unusable. Clients MUST treat a key set containing duplicate kid values as invalid.

Example:

```
{
  "issuer": "https://api.example.com",
  "keys": [
    {
      "kid": "2026-06",
      "alg": "X25519",
      "aeads": ["AES-256-GCM", "AES-128-GCM"],
      "public_key": "B6N8vBQgk8i3VdwbEOhstCY3StFqqFPtC9_AsrhtHHw",
      "fingerprint": "qqj_9wO1CyKX9PbhNQj3JA",
      "not_before": "2026-06-09T00:00:00Z",
      "not_after":  "2026-07-09T00:00:00Z",
      "max_skew": 300
    }
  ]
}
```

4.3. Caching

Responses SHOULD include a Cache-Control header. Clients MAY cache the key set for up to the duration indicated, but MUST refresh the key set before using a key whose not_after is in the past, and SHOULD refresh on receipt of a key_unknown error (see Section 9).

4.4. Out-of-Band Trust

The integrity of the key set ultimately depends on the integrity of the channel used to retrieve it. Clients MUST verify the TLS server certificate of the server hosting the Well-Known URI per [RFC8446] and [RFC9325]. Clients MUST also verify that the issuer value in the key set matches the HTTPS origin used to retrieve it, unless an out-of-band configuration explicitly authorizes a different issuer.

If the deployment threat model includes TLS-terminating intermediaries that are not trusted with plaintext payloads, TLS authentication of the Well-Known URI is not sufficient to authenticate the encryption keys: such an intermediary could substitute its own key set. In that threat model, clients MUST authenticate the key set independently of the TLS connection by using either a pinned key fingerprint or a verified HTTP Message Signature whose signing key was distributed out of band; see Section 11.

Servers SHOULD additionally protect the key set response with HTTP Message Signatures [RFC9421], signed under a long-lived signing key distributed out of band, and SHOULD include a Content-Digest field [RFC9530] over the key set body so that the signature covers a content hash rather than the raw bytes. Clients that have a pinned signing key MUST reject any key set response whose signature is missing or invalid.

5. AEAD Identifiers

This document defines three AEAD Identifiers, all based on AES-GCM [NIST-SP-800-38D]:

AEAD Identifier	Key length	Nonce length	Tag length
AES-128-GCM	16 octets	12 octets	16 octets
AES-192-GCM	24 octets	12 octets	16 octets
AES-256-GCM	32 octets	12 octets	16 octets

Table 1

All three variants share the same nonce and tag length and differ only in key length. Implementations MUST support AES-128-GCM and AES-256-GCM. Support for AES-192-GCM is OPTIONAL.

The client selects one AEAD Identifier from the server’s aeads list for each session and signals it in the request (see Section 8). The server MUST reject any request whose AEAD Identifier is not advertised in the current key set entry for the chosen kid.

6. Key Agreement and Derivation

6.1. Ephemeral Client Keys and Sessions

A _session_ is a contiguous sequence of requests sent by one client under a single X25519 key pair (csk, cpk). The client public key cpk is sent on the wire as the epk parameter of the E2EE-Session field.

Clients MUST generate (csk, cpk) using a cryptographically secure random number generator. The default session scope is a single request: clients SHOULD generate a fresh key pair for every request unless they explicitly opt into a broader scope as described below.

A client MAY reuse one (csk, cpk) across multiple requests provided that all of the following hold:

* The reuse is confined to a single logical client instance (one process, one user, one device).

* The key pair is not persisted to non-volatile storage and is destroyed on process exit, user logout, or after a deployment-defined inactivity timeout, whichever comes first.

* The deployment-defined session lifetime does not exceed the shortest not_after of any server key it references.

* The client maintains its own nid uniqueness within the session (see Section 11).

Clients MUST NOT reuse (csk, cpk) across processes, after restart, across users on a shared device, or after any event that would invalidate the client's secure-random state (for example, virtual machine snapshot restore).

Each unique value of epk observed by a server defines, together with the server kid, the scope of replay-cache state for that client (see Section 11). Per-request session scope therefore minimises both the lifetime of the client private key and the amount of replay-cache state retained for any one client.

6.2. Shared Secret

The shared secret is computed as:

Z = X25519(csk, server_public_key)

per Section 5 of [RFC7748]. Implementations MUST check that Z is not the all-zero value and abort if it is.

6.3. Encryption Key Derivation

The request encryption key (EK_req) and response encryption key (EK_res) are derived from Z using HKDF-SHA256 [RFC5869]:

```
PRK = HKDF-Extract(salt = cpk || server_public_key, IKM = Z)
EK_req = HKDF-Expand(PRK,
                     info = "e2ee/v1:req " || issuer || " "
                            || aead || " " || kid,
                     L    = Nk)
EK_res = HKDF-Expand(PRK,
                     info = "e2ee/v1:res " || issuer || " "
                            || aead || " " || kid,
                     L    = Nk)
```

where:

* || denotes octet-string concatenation,

* cpk is the raw 32-byte client public key,

* server_public_key is the raw 32-byte server public key,

* issuer is the UTF-8 encoding of the key set issuer value,

* aead is the UTF-8 encoding of the selected AEAD Identifier ("AES-128-GCM", "AES-192-GCM", or "AES-256-GCM"),

* kid is the UTF-8 encoding of the Key Identifier used,

* Nk is the key length in octets implied by aead (16, 24, or 32).

Binding issuer, aead, kid, and both public keys into the HKDF inputs binds the derived keys to the specific key agreement transcript and prevents cross-origin, cross-protocol, cross-key, cross-direction, or cross-algorithm confusion. Deriving separate request and response keys prevents an AES-GCM nonce collision in one direction from colliding with a nonce in the other direction.

7. Message Format

7.1. Media Type

Protected payloads MUST be sent with the media type application/e2ee. The media type identifies the ciphertext envelope defined by this document; the plaintext media type, when needed, is carried in the cty parameter of the E2EE-Session field (see Section 8.2).

7.2. Wire Format

The body of a protected request or response is the concatenation:

```
body = nonce || ciphertext || tag
```

where:

* nonce is 12 octets, generated with a cryptographically secure random number generator for every message. Nonces MUST NOT be reused with the same direction-specific EK.

* ciphertext is the AES-GCM encryption of the inner plaintext using EK_req for requests or EK_res for responses and nonce, with additional authenticated data (AAD) as defined in Section 7.4. The AES-GCM key length is determined by the selected AEAD Identifier (see Section 5).

* tag is the 16-octet AES-GCM authentication tag.

Recipients MUST reject a protected body shorter than 28 octets (12-octet nonce plus 16-octet tag) as malformed.

7.3. Interaction with Content-Digest

The AES-GCM tag already provides end-to-end integrity for the ciphertext, so a Content-Digest field [RFC9530] is not required for integrity. Senders MAY include Content-Digest over the raw ciphertext body for caching, deduplication, or intermediary integrity checks; in that case the digest MUST be computed over the exact serialized body (nonce || ciphertext || tag). Recipients MUST NOT treat a present Content-Digest as a substitute for AES-GCM tag verification, and MUST NOT compute or expose a digest over the decrypted plaintext.

7.4. Additional Authenticated Data

The AAD passed to AES-GCM binds only the cryptographic transcript of the message. HTTP semantics such as method, target URI, status code, and selected headers are out of scope for AAD and are bound, when needed, by HTTP Message Signatures [RFC9421] (see Section 7.5).

For AAD construction, an E2EE-Session field value MUST first be parsed as a Structured Field Item and then serialized using the deterministic serialization algorithm for Structured Fields [RFC9651]. The serialized field value does not include the field name, colon, or surrounding whitespace. Recipients MUST NOT use the raw field bytes as received from an HTTP/1.1 connection, because equivalent field values can have different wire serializations and HTTP/2 and HTTP/3 do not preserve an HTTP/1.1 header-field line.

For requests, the AAD MUST be the ASCII concatenation:

AAD = "e2ee/v1:req" || " " || encryption-field

For responses, the AAD MUST be the ASCII concatenation:

```
AAD = "e2ee/v1:res" || " " || req-encryption-field
                        || " " || res-encryption-field
```

where:

* encryption-field is the deterministic serialization of the request's parsed E2EE-Session Structured Field value (see Section 8),

* req-encryption-field is the same value as encryption-field for the request that produced this response,

* res-encryption-field is the deterministic serialization of the response's own parsed E2EE-Session field value.

The leading "e2ee/v1:req" / "e2ee/v1:res" strings combine a protocol-version tag with a domain-separation tag. They prevent a request ciphertext from being accepted as a response (or vice versa) and isolate this protocol from any other use of AES-GCM under the same key.

Including both E2EE-Session fields in the response AAD authenticates kid, aead, epk, ts, and nid from both directions jointly with the ciphertext, prevents an intermediary from altering any of them, and prevents response substitution across different requests.

7.5. Binding HTTP Semantics

This specification does not bind HTTP semantics (method, target URI, status code, request or response headers) into the AES-GCM AAD. Doing so would conflict with the routine behaviour of TLS-terminating intermediaries that legitimately rewrite request targets, normalize paths, or add tracing parameters.

Deployments that require an end-to-end binding of HTTP semantics SHOULD apply HTTP Message Signatures [RFC9421] to requests and responses, alongside the encryption defined by this document. The covered-components list SHOULD include at least:

* For requests: @method, @target-uri, the E2EE-Session field, and a Content-Digest field [RFC9530] computed over the ciphertext body.

* For responses: @status, the E2EE-Session field, and a Content-Digest field over the ciphertext body.

Because Content-Digest covers the ciphertext (which already contains the AES-GCM tag), an RFC 9421 signature over those components transitively covers the encrypted payload. The signature key and keyid are independent of the X25519 key set defined here and are managed per [RFC9421].

A deployment that omits HTTP Message Signatures relies on this protocol's confidentiality and integrity guarantees only at the payload layer, and on TLS and application-level checks for HTTP semantics. This is appropriate when the application cannot be confused by a redirected ciphertext (for example, when each endpoint parses a distinct payload schema and rejects unknown shapes).

8. The E2EE-Session Field

All E2EE control metadata is carried in a single HTTP field named E2EE-Session. The field is a Structured Field [RFC9651] whose value is an Item: a String identifying the server key (kid), with named parameters carrying the remaining metadata.

8.1. Syntax

In the ABNF of [RFC9651], the field value is an sf-item:

E2EE-Session = sf-item

The Item value is a String holding the kid. The following parameters are defined:

Name	Type	Request	Response	Meaning
aead	String	required	required	AEAD Identifier (see Section 5).
epk	Byte Sequence	required	prohibited	Client ephemeral X25519 public key.
ts	Integer	required	required	Seconds since Unix epoch.
nid	String	required	required	Per-message replay identifier (e.g. UUID).
cty	String	optional	optional	Media type of the inner plaintext (see Section 8.2).

Table 2

epk is the raw 32-octet X25519 public key carried as a Structured Fields Byte Sequence (RFC 9651 base64, sf-binary, delimited by :), not base64url. Servers MUST reject any value whose decoded length is not 32 octets.

ts MUST be a non-negative Integer.

nid MUST contain between 1 and 128 characters and MUST match the regular expression ^[A-Za-z0-9._~-]+$. nid comparison is case-sensitive.

cty is OPTIONAL and, when present, MUST be the media type of the inner plaintext (see Section 8.2).

Parameters defined by future revisions of this specification or by extensions MUST use distinct names. Recipients MUST ignore unknown parameters semantically, but unknown parameters remain part of the deterministic serialization that is authenticated as AAD. A field value that contains the same parameter name more than once MUST be rejected as malformed; this check MUST occur before deterministic serialization. Implementations whose Structured Fields parser cannot report duplicate parameters MUST use a parser that can for this field.

8.2. Inner Content Type

The plaintext recovered after AES-GCM decryption is opaque to this specification. To allow recipients to dispatch on its format without a separate inner header, senders MAY include a cty parameter on the E2EE-Session field carrying the inner plaintext's media type as a String, for example cty="application/json" or cty="application/cbor".

The value MUST be a valid media type per Section 8.3 of [RFC9110]. Media-type parameters (for example, charset=utf-8) MAY be present in cty. Recipients SHOULD treat the absence of cty as "unspecified" and rely on out-of-band knowledge of the endpoint to interpret the plaintext.

Because cty is a parameter of the E2EE-Session field, it is covered by the AES-GCM AAD (see Section 7.4), so an intermediary cannot alter the advertised inner media type without invalidating the tag.

The outer HTTP Content-Type field, when present, MUST remain application/e2ee (or another media type defined for the ciphertext envelope); cty does not replace it. Recipients that receive both an outer Content-Type of application/e2ee and an inner cty MUST use cty only for the decrypted plaintext.

8.3. Example

```
E2EE-Session: "2026-06"; \
            aead="AES-256-GCM"; \
            epk=:rUOL+uMfbAk9YdQzklXqeYCSyfrdB7l4J/Swrp3ufBw=:; \
            ts=1781006400; \
            nid="3b1c1c2e-2b6a-4a0d-9b6c-2a9f1b6a0e21"; \
            cty="application/json"
```

(Line folding shown for readability; the field is a single Structured Field value. AAD construction uses deterministic Structured Fields serialization, not these presentation line breaks.)

8.4. Use in Requests and Responses

Every request that carries an encrypted payload MUST include exactly one E2EE-Session field with all required parameters. A response that carries an encrypted payload MUST include an E2EE-Session field whose kid Item value and aead parameter echo the request. The epk parameter MUST be omitted from responses; the server reuses the client's ephemeral public key from the request. The ts parameter on a response MUST reflect the server's current time. A response MUST echo the request's nid so that clients can correlate responses and detect duplicate or substituted responses.

Clients MUST verify that the kid Item value and the aead and nid parameters of the response E2EE-Session field match the values they sent in the corresponding request, and MUST discard the response without decrypting it on mismatch. A mismatch indicates either an in-path attacker or a misconfigured intermediary serving a cached or cross-routed response. Because the request's E2EE-Session field is included in the response AAD (see Section 7.4), a mismatch is also detectable as an AES-GCM tag failure during decryption; the pre-decryption check is RECOMMENDED to avoid spending a decryption attempt on an obviously wrong response.

8.5. Validation Order

Recipients MUST validate the E2EE-Session field before any other protocol step.

For requests, servers MUST validate in the following order:

1. Parse as a Structured Fields Item; reject as malformed on failure.

2. Check that the Item value is a String, that all required request parameters are present, that no parameter prohibited in requests by Section 8 is present, that no parameter name appears more than once, and that all known parameters have the types defined in Section 8; reject as malformed on failure.

3. If cty is present, validate it as a media type per Section 8.3 of [RFC9110]; reject as malformed on failure.

4. Resolve kid against the current key set; reject as key_unknown or key_expired.

5. Check aead against the aeads list for that kid; reject as aead_unsupported.

6. Decode and length-check epk; reject as malformed.

7. Check the protected body length; reject as malformed if it is shorter than 28 octets.

8. Check ts against the selected key's not_before/not_after validity window and max_skew; reject as timestamp_skew if outside the window.

9. Check nid against the replay cache; reject as replay_detected on hit, but do not insert the nid yet.

10. Derive EK_req and attempt AES-GCM decryption; reject as decrypt_failed on tag failure.

11. After successful authentication and before invoking application processing, insert the nid into the replay cache atomically.

For responses, clients MUST validate in the following order:

1. Parse as a Structured Fields Item; reject the response on failure.

2. Check that the Item value is a String, that all required response parameters are present, that epk is absent, that no parameter name appears more than once, and that all parameters known to this specification have the types defined in Section 8; reject the response on failure.

3. If cty is present, validate it as a media type per Section 8.3 of [RFC9110]; reject the response on failure.

4. Check that kid, aead, and nid match the request.

5. Check the protected body length; reject the response if it is shorter than 28 octets.

6. Check that ts is a non-negative Integer and is acceptable under local response freshness policy.

7. Derive EK_res and attempt AES-GCM decryption; reject the response on tag failure.

9. Error Handling

When a request cannot be processed due to a protocol error, the server MUST respond with an HTTP error status and a Problem Details object [RFC9457] serialized as application/problem+json. The type member MUST be a URI of the form:

urn:ietf:params:e2ee:error:<code>

where <code> is one of the codes defined below. The status member MUST equal the HTTP status code of the response. The title member SHOULD be a short, fixed human-readable summary of the code. The detail member, if present, SHOULD describe the specific occurrence subject to the constraints in Section 11.6.

Example:

```
HTTP/1.1 400 Bad Request
Content-Type: application/problem+json

{
  "type":   "urn:ietf:params:e2ee:error:key_unknown",
  "title":  "Key identifier is not recognized",
  "status": 400
}
```

The following codes are defined:

key_unknown: HTTP 400. The kid Item value does not match any current key.

key_expired: HTTP 400. The referenced key is outside its not_before/not_after window.

aead_unsupported: HTTP 400. The aead parameter is not advertised for the referenced kid, or is not recognized by the server.

decrypt_failed: HTTP 400. AES-GCM authentication failed.

timestamp_skew: HTTP 400. The ts parameter is outside the acceptable window (see Section 11).

replay_detected: HTTP 425. The server has already processed a message with this nid within the replay window. The 425 (Too Early) status [RFC8470] is reused here to signal that the server is unwilling to process a request that may be a replay.

malformed: HTTP 400. The E2EE-Session field is missing, not parseable as a Structured Field Item, missing a required parameter, or has a parameter of the wrong type or length, or the protected body is too short to contain a nonce and tag.

10. IANA Considerations

This document requests IANA actions in the following registries.

10.1. Well-Known URI

IANA is requested to register the URI suffix encryption-keys in the "Well-Known URIs" registry established by [RFC8615], with this document as the reference.

URI suffix: encryption-keys

Change controller: IETF

Status: permanent

Specification document: This document.

Related information: This resource returns a JSON key set used by the protocol defined in this document.

10.2. Media Type

IANA is requested to register the media type application/e2ee in the "Media Types" registry, with this document as the reference and per the procedures in [RFC6838].

Type name: application

Subtype name: e2ee

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See Section 11.

Interoperability considerations: Implementations need to parse the E2EE-Session Structured Field and process the binary nonce || ciphertext || tag envelope defined by this document. Interoperability depends on agreement on the selected AEAD Identifier, the key set entry referenced by kid, and the AAD construction rules in Section 7.4.

Published specification: This document.

Applications that use this media type: HTTP APIs that use the encrypted payload envelope defined by this document.

Fragment identifier considerations: N/A

Additional information: Deprecated alias names for this type: N/A Magic number(s): N/A File extension(s): N/A Macintosh file type code(s): N/A

Person and email address to contact for further information: See the Authors' Addresses section.

Intended usage: LIMITED USE

Restrictions on usage: This media type is intended for HTTP request and response payloads that use the encrypted envelope defined by this document. It is not a general-purpose stored file format.

Author: See the Authors' Addresses section.

Change controller: IETF

10.3. HTTP Field Names

IANA is requested to register the following entry in the "Hypertext Transfer Protocol (HTTP) Field Name Registry":

* Field name: E2EE-Session
* Status: permanent
* Structured Type: Item
* Reference: This document

10.4. Problem Type URN Parameter Identifier

IANA is requested to add the following entry to the "IETF URN Sub-namespace for Registered Protocol Parameter Identifiers" registry:

* Registered Parameter Identifier: e2ee
* Reference: This document
* IANA Registry Reference: E2EE Error Codes registry, created by this document

The registration policy for this registry is IETF Review [RFC8126], as specified for the urn:ietf:params namespace by [RFC3553].

The template required by [RFC3553] is:

Registry name: e2ee

Specification: This document.

Repository: The E2EE Error Codes registry created by this document.

Index value: A problem type URI has the form

urn:ietf:params:e2ee:error:<code>, where <code> is a lowercase ASCII error code registered in the E2EE Error Codes registry. No transformation or canonicalization is applied. Comparison is by exact string match.

10.5. E2EE Error Codes

IANA is requested to create the "E2EE Error Codes" registry. The registration policy is Specification Required [RFC8126].

Error codes are ASCII strings that MUST contain between 1 and 64 characters and MUST match the regular expression ^[a-z][a-z0-9_]{0,63}$. New registrations MUST provide:

* Error code
* HTTP status code
* Description
* Reference

The initial contents of the registry are:

Error code	HTTP status	Description	Reference
key_unknown	400	Key identifier is not recognized.	This document
key_expired	400	Key identifier is outside its validity.	This document
aead_unsupported	400	AEAD identifier is not supported.	This document
decrypt_failed	400	AEAD authentication failed.	This document
timestamp_skew	400	Timestamp is outside the accepted window.	This document
replay_detected	425	Replay identifier was already observed.	This document
malformed	400	Protocol metadata or body is malformed.	This document

Table 3

11. Security Considerations

11.1. Threat Model

This scheme is designed to protect the confidentiality and integrity of request and response payloads against:

* Intermediaries that terminate TLS (reverse proxies, CDNs, API gateways), and

* Passive observers of any plaintext channel between the TLS terminator and the application backend.

It is not designed to protect HTTP metadata (method, path, headers other than the protected body, response status) or to defeat traffic analysis. It does not authenticate the client by itself. Client authentication MAY be layered on top using the protected payload (for example, bearer tokens carried inside the ciphertext) or, preferably, by signing the request with HTTP Message Signatures [RFC9421]; in the latter case the signature input MUST include the E2EE-Session field and the Content-Digest field over the ciphertext body so that the signature also binds the AAD and ciphertext.

11.2. Transport Layer Security

This scheme MUST be used over TLS [RFC8446] configured per current best practice [RFC9325]. TLS protects request metadata, the Well-Known key set retrieval, and provides server authentication.

11.3. Server Authentication and Key Trust

When the TLS endpoint that serves the Well-Known URI is also the application endpoint trusted with plaintext payloads, the server’s identity is authenticated by the TLS certificate of that host and by the issuer value in the key set.

When TLS-terminating intermediaries are present and are not trusted with plaintext payloads, TLS authentication alone does not authenticate the encryption keys end to end. In that deployment, clients MUST verify the fingerprint of any key they use against a value obtained out of band (for example, distributed with the client software) or MUST verify a signature over the key set using a signing key distributed out of band. Mobile and desktop client implementations are RECOMMENDED to pin at least one fingerprint or signing key.

Where stronger guarantees are required, servers SHOULD sign the key set with HTTP Message Signatures [RFC9421] and cover the body with a Content-Digest field [RFC9530]. The signing key is necessarily distributed out of band; clients that pin the signing key obtain key authenticity that is independent of the Web PKI used by TLS.

11.4. Forward Secrecy

The client’s key pair is ephemeral, but the server’s published key is static for the lifetime of its key set entry. The scheme therefore provides forward secrecy only with respect to client-side compromise. Compromise of a server private key allows decryption of all sessions that used it.

Operators SHOULD rotate server keys frequently and use short not_after windows. Implementations of this specification SHOULD publish at least two overlapping keys in the key set to enable seamless rotation.

A future revision of this protocol MAY define a mode in which the server returns a fresh ephemeral public key on first contact, providing full perfect forward secrecy at the cost of an additional round trip.

11.5. Replay Protection

Servers MUST validate the ts parameter against the key validity window for the referenced kid: any ts before not_before, when present, or after not_after MUST be rejected (timestamp_skew). Servers MUST also reject any ts whose absolute difference from the server clock exceeds the key set entry's max_skew value (timestamp_skew).

Servers SHOULD publish a max_skew value no larger than the maximum retry interval they are willing to support; a value of 300 seconds is RECOMMENDED unless the deployment has stricter clock synchronization or longer retry requirements.

Clients MUST read max_skew from the selected key set entry and account for it when scheduling retries. A retry sent with a fresh nid after the original request's timestamp has aged beyond max_skew is expected to fail timestamp validation.

Servers MUST maintain a cache of recently seen nid values, keyed by (kid, epk), for at least max_skew plus a small tolerance for processing latency and clock granularity. A repeated nid MUST result in replay_detected. Clients MUST generate a fresh, unpredictable nid for every request; a version 4 UUID or any 128-bit value drawn from a cryptographically secure random number generator is sufficient.

Servers MUST NOT insert a nid into the replay cache until the request has been authenticated successfully by AES-GCM. Inserting a nid before tag verification would allow an attacker to poison the replay cache with unauthenticated requests. Servers MUST insert the nid atomically after successful authentication and before application side effects occur; if another request with the same (kid, epk, nid) wins that atomic insertion, the later request MUST fail with replay_detected.

The nid parameter is an anti-replay identifier and is not an application idempotency key. The two have opposite behavior on a cache hit: a duplicate nid MUST cause the server to reject the

request, whereas a duplicate application idempotency key is normally expected to cause the server to return the stored response of the original request. Applications that require both fault-tolerant retries and end-to-end replay protection MUST use distinct values for the two purposes. An application idempotency key, if used, SHOULD be carried inside the encrypted payload so that the application sees it but intermediaries do not, and so that it is bound to the request by the AES-GCM tag.

The 12-byte AES-GCM nonce is independently random per message and MUST NOT be reused under the same direction-specific EK.

11.6. Plaintext Error Responses

Error responses defined in Section 9 are sent as Problem Details [RFC9457] in plaintext and are therefore visible to any TLS-terminating intermediary. Error responses do not carry encrypted application payloads and do not require an E2EE-Session field; some errors occur before the request E2EE-Session field can be parsed safely.

Operators MUST treat error metadata as observable to intermediaries. In particular:

* The detail and instance members of the Problem Details object MUST NOT contain user identifiers, request contents, stack traces, or any value derived from the decrypted plaintext.

* The title member SHOULD be a fixed string per error code and MUST NOT vary with request content.

* Extension members defined by future revisions or by deployments MUST be subject to the same constraints.

* Error type URIs SHOULD be chosen from the fixed set in Section 9; application-specific error information SHOULD instead be returned as an encrypted payload with an appropriate HTTP status.

* The presence of decrypt_failed or replay_detected reveals to an observer that an attack or a stale retry occurred; this is considered acceptable as it aids defenders more than attackers.

Successful responses MUST carry their application payload encrypted under this specification; servers MUST NOT fall back to a plaintext success response when the request was encrypted.

11.7. Key Rotation

Each request carries a kid, allowing the server to retain the private keys associated with all currently valid kids and decrypt requests that arrive during a rotation. Clients MUST refresh the key set when the cached entry has expired, when no key remains within its validity window, or on key_unknown.

11.8. Algorithm Agility

This document specifies one key agreement algorithm (X25519), one key-derivation function (HKDF-SHA256), and three AEAD variants (AES-128/192/256-GCM). Servers MAY publish keys with alg values or list aeads entries defined by future specifications; clients that do not recognize an alg MUST ignore the key, and clients that do not recognize any of the aeads entries for a key MUST treat the key as unusable.

AES-128-GCM and AES-256-GCM are mandatory to implement; AES-192-GCM is OPTIONAL because the 192-bit variant is rarely accelerated in hardware and provides no security advantage over AES-128-GCM in this setting. Implementations SHOULD prefer AES-256-GCM by default. AES-128-GCM is acceptable when interoperability or constrained-device performance takes precedence. Implementations MUST reject any single plaintext that exceeds the AES-GCM per-invocation input limit (approximately 2^39 - 256 bits per [NIST-SP-800-38D]). Deployments that reuse a client session across multiple requests SHOULD also enforce a maximum number of messages per direction-specific EK; with randomly generated 96-bit nonces, 2^32 messages under one direction-specific EK is an upper bound and is far above typical API usage.

11.9. Side Channels and Implementation

X25519 implementations MUST be constant-time per [RFC7748]. AES-GCM implementations SHOULD use hardware acceleration where available to reduce the risk of cache-timing leaks. HKDF and base64url implementations MUST validate input lengths to avoid out-of-bounds reads.

11.10. Denial of Service

Decryption is comparatively cheap, but X25519 scalar multiplication is not free. Servers SHOULD rate-limit requests bearing unknown or expired kids, and SHOULD reject malformed bodies before performing key agreement.

11.11. Comparison with Transport-Layer Solutions

This scheme is complementary to, and not a replacement for, alternatives such as mTLS or QUIC [RFC9000]. It is specifically targeted at deployments where TLS-terminating intermediaries are part of the application architecture and removing them is not feasible.

12. Privacy Considerations

This protocol improves payload confidentiality from the perspective of TLS-terminating intermediaries, but it does not hide HTTP metadata. Intermediaries can still observe the client and server endpoints, request timing, request method, target URI, status code, message sizes, and all unprotected HTTP fields. Applications that carry privacy-sensitive values in URIs or unprotected headers will still expose those values.

The E2EE-Session field is also visible to intermediaries. Its kid, aead, ts, nid, epk, and cty parameters can reveal deployment state, timing, retry behavior, client session scope, and the media type of the inner plaintext. Clients that need to reduce linkability SHOULD use the default per-request client key scope and generate fresh nid values for every request.

Plaintext error responses expose protocol failure information, as described in Section 11.6. Deployments SHOULD avoid putting user-specific or content-derived information in error details and SHOULD carry application-specific error information inside encrypted response payloads when feasible.

These considerations are intended to supplement the privacy guidance in [RFC6973].

13. References

13.1. Normative References

[NIST-SP-800-38D]
National Institute of Standards and Technology, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST Special Publication 800-38D, 2007.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/rfc/rfc3339>.

[RFC3553] Mealling, M., Masinter, L., Hardie, T., and G. Klyne, "An IETF URN Sub-namespace for Registered Protocol Parameters", BCP 73, RFC 3553, DOI 10.17487/RFC3553, June 2003, <https://www.rfc-editor.org/rfc/rfc3553>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/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/rfc/rfc5869>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

[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/rfc/rfc7748>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[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/rfc/rfc8446>.

[RFC8470] Thomson, M., Nottingham, M., and W. Tarreau, "Using Early Data in HTTP", RFC 8470, DOI 10.17487/RFC8470, September 2018, <https://www.rfc-editor.org/rfc/rfc8470>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

[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/rfc/rfc9110>.

[RFC9325] Sheffer, Y., Saint-Andre, P., and T. Fossati, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 9325, DOI 10.17487/RFC9325, November 2022, <https://www.rfc-editor.org/rfc/rfc9325>.

[RFC9421] Backman, A., Ed., Richer, J., Ed., and M. Sporny, "HTTP Message Signatures", RFC 9421, DOI 10.17487/RFC9421, February 2024, <https://www.rfc-editor.org/rfc/rfc9421>.

[RFC9457] Nottingham, M., Wilde, E., and S. Dalal, "Problem Details for HTTP APIs", RFC 9457, DOI 10.17487/RFC9457, July 2023, <https://www.rfc-editor.org/rfc/rfc9457>.

[RFC9530] Polli, R. and L. Pardue, "Digest Fields", RFC 9530, DOI 10.17487/RFC9530, February 2024, <https://www.rfc-editor.org/rfc/rfc9530>.

[RFC9651] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 9651, DOI 10.17487/RFC9651, September 2024, <https://www.rfc-editor.org/rfc/rfc9651>.

13.2. Informative References

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/rfc/rfc6973>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[VASYLENKO-BLOG] Vasylenko, V., "End-to-End Encryption for APIs: X25519 and AES", 27 July 2025, <https://blog.vitalvas.com/post/2025/07/27/e2e-encryption-api-x25519-aes/>.

Acknowledgments

The protocol described in this document derives from an informal blog-post implementation of end-to-end encryption for HTTP APIs using X25519 and AES-GCM [VASYLENKO-BLOG]. This document formalizes the wire format, generalizes the cipher to AES-128/192/256-GCM with explicit negotiation, specifies additional authenticated data and replay protection, and defines key rotation semantics.

Worked Example

This appendix shows one complete request/response pair using deterministic inputs. All values are hexadecimal unless noted; lines of the form = are byte-identical to what an interoperable implementation would compute. The inputs (private keys, AES-GCM nonces) are fixed so the example is reproducible; production code MUST NOT reuse these values.

Inputs

```
server private (ssk) =
    0102030405060708090a0b0c0d0e0f10
    1112131415161718191a1b1c1d1e1f20
server public  (spk) =
    07a37cbc142093c8b755dc1b10e86cb4
    26374ad16aa853ed0bdfc0b2b86d1c7c

client private (csk) =
    a1a2a3a4a5a6a7a8a9aaabacadaeafb0
    b1b2b3b4b5b6b7b8b9babbbcbdbebfc0
client public  (cpk) =
    ad438bfae31f6c093d61d4339255ea79
    8092c9fadd07b97827f4b0ae9dee7c1c

kid  = "2026-06"
aead = "AES-256-GCM"
issuer = "https://api.example.com"
ts   = 1781006400
nid  = "3b1c1c2e-2b6a-4a0d-9b6c-2a9f1b6a0e21"
cty  = "application/json"
```

Key Agreement and Derivation

```
   Z   = X25519(csk, spk)
       = 1eadf045f970f3619aa3a82d3ce461d6
         8ee42839f0563ff052d8db20bf927d29

   salt = cpk || spk    (64 octets)
   info_req = "e2ee/v1:req https://api.example.com AES-256-GCM 2026-06"
   info_res = "e2ee/v1:res https://api.example.com AES-256-GCM 2026-06"

   EK_req = HKDF-SHA256(Z, salt, info_req, 32)
          = 88927bb69c7fce5a26b88ccf3b8638c5
            e876080eae5349c7a014787e80382f81

   EK_res = HKDF-SHA256(Z, salt, info_res, 32)
          = 2784f1a637499c327e97ad56a0a199b9
            50680c41e57597cea41a220233304a8b
```

Request

The serialized E2EE-Session field (single logical line):

```
   E2EE-Session: "2026-06"; aead="AES-256-GCM";
               epk=:rUOL+uMfbAk9YdQzklXqeYCSyfrdB7l4J/Swrp3ufBw=:;
               ts=1781006400;
               nid="3b1c1c2e-2b6a-4a0d-9b6c-2a9f1b6a0e21";
               cty="application/json"
```

The AAD is the ASCII string below, using deterministic Structured Fields serialization of the E2EE-Session field value:

```
   e2ee/v1:req "2026-06"; aead="AES-256-GCM"; epk=:...:; \
   ts=1781006400; nid="3b1c1c2e-2b6a-4a0d-9b6c-2a9f1b6a0e21"; \
   cty="application/json"
```

(The :...: here stands for the full sf-binary form shown above and is shortened only for readability; the actual AAD contains the full deterministic serialization.)

```
   plaintext = {"op":"transfer","amount":1000,"to":"acct-42"}

   nonce     = deadbeef0000000000000001
   ciphertext =
       a6b3551bec16e7866943502146d893b2
       baa8bc6a4ef76712f7e4febcb576c821
       41551464b46eb0f096750ed69020
   tag       =
       4cc3c77e4c463d111f81bf6cf83f08d5
```

The HTTP request body is nonce || ciphertext || tag, base64-encoded here for compactness:

```
body (base64) =
    3q2+7wAAAAAAAABprNVG+wW54ZpQ1AhRtiTsrqovGpO92cS
    9+T+vLV2yCFBVRRktG6w8JZ1DtaQIEzDx35MRj0RH4G/bPg/CNU=
```

Response

The response uses a fresh nonce and a new ts, echoes kid, aead, and nid, and omits epk:

```
E2EE-Session: "2026-06"; aead="AES-256-GCM";
            ts=1781006401;
            nid="3b1c1c2e-2b6a-4a0d-9b6c-2a9f1b6a0e21";
            cty="application/json"
```

AAD (the request's deterministic field serialization followed by the response's deterministic field serialization):

```
e2ee/v1:res <request-encryption-field> <response-encryption-field>
```

```
plaintext = {"status":"ok","txid":"a1b2c3"}

nonce     = feedface0000000000000002
ciphertext =
    f111c0a217756b5f967108e32ce392d6
    2f4de9380b2267c53b81cc4679bc59
tag       =
    5b64d39058d1bb23e2cec5f9c69880e1

body (base64) =
    /u36zgAAAAAAAAAC8RHAohd1a1+WcQjjLOOS1i9N6TgLImfFO4H
    MRnm8WVtk05BY0bsj4s7F+caYgOE=
```

Author's Address

Vitaliy Vasylenko
Email: ietf@vitalvas.com