Secure Evidence and Attestation Transport
Internet-Draft
Intended status: Standards Track
Expires: 2 September 2026

N. Ritz
Independent

1 March 2026

Factor-based Attestation and Credential Transport Scheme (FACTS) over TLS 1.3
draft-ritz-seat-facts-00

Abstract

This document describes FACTS (Factor-based Attestation and Credential Transport Scheme) over TLS 1.3. Conceptually acting as "multi-factor authentication" for machine identities, factor-based attestation derives session trust from multiple independent cryptographic inputs rather than a single point of failure. Specifically, it utilizes a dual-key scheme that binds identity to attestation evidence through the use of key encapsulation material keys (KEM) and traditional identity signing keys (IK), establishing per-session freshness.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ritz-seat-facts/.

Discussion of this document takes place on the Secure Evidence and Attestation Transport Working Group mailing list (mailto:seat@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/seat. Subscribe at https://www.ietf.org/mailman/listinfo/seat/.

Source for this draft and an issue tracker can be found at https://github.com/nathanaelritz/facts-tls.

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/.

Copyright Notice

Table of Contents

1. Introduction

Remote attestation enables an entity to demonstrate the integrity of its runtime environment to a remote peer. When combined with TLS [I-D.ietf-tls-rfc8446bis], attestation allows a relying party to verify that the TLS endpoint operates within a Trusted Execution Environment (TEE) with known-good software before exchanging application data.

This document combines the latency properties of intra-handshake attestation with the security binding of post-handshake attestation through a dual-key authentication architecture that introduces four targeted architectural changes that preserve the standard TLS 1.3 state machine and key schedule:

1. _LTK and KEM Keys:_ Introduction of a dedicated encapsulation key (pubKEM), alongside the usual long-term signing or identity key (pubIK / LTK).

2. _Early authentication:_ Accessing signed identity documents (e.g. signed JWT/CWT) just before starting the TLS handshake enables cryptographic binding of pubKEM to the server identity and pubIK jointly, closing the identity/key confusion gap that diversion attacks exploit.

3. _Dual-Nonce Encapsulation:_ Encapsulation of mutual ’challenge’ and ’counter-challenge’ nonces (CN1, CN2) to the Server’s pubKEM and the Client’s per-session pubKEM, respectively.

4. _Session-Specific Quote Encryption:_ Encrypting the attestation quote under psk_attest (derived from both challenge nonces), with the rdata committing to the full suite of parameters (pubIK_S, CN1, CN2, pubKEM_C).

The pre-handshake Identity Document is formatted as a JSON Web Token [RFC7519] carrying keys in a JWKS cnf claim [RFC7800]. No novel cryptographic structures are introduced by this document. JWT provides a self-contained, channel-independent integrity guarantee suitable for distribution over untrusted transports such as DNS.

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.

The reader is assumed to be familiar with the vocabulary and concepts defined in [RFC9334], [I-D.ietf-tls-rfc8446bis], and [RFC9180]. FACTS relies on the RATS Conceptual Message Wrapper (CMW) [I-D.ietf-rats-msg-wrap] for encapsulating attestation payloads and supports both the background-check and passport topologies defined in [RFC9334].

TLS Identity Key (privIK / pubIK aka LTK): A long-term signing key pair used by a TLS peer for authentication during the handshake via CertificateVerify. In the threat model of this document, privIK is assumed to be potentially compromisable by an adversary with sufficient access to the software layer.

Encapsulation Key (privKEM / pubKEM): A key pair used for Authenticated Hybrid Public Key Encryption (AuthHPKE, mode 2 of [RFC9180]). In the threat model of this document, privKEM is assumed to be potentially compromisable independently of privIK.

Attestation Key (privAK / pubAK): A hardware-bound key pair rooted in the TEE’s hardware root of trust (e.g., a TPM endorsement key, an Intel TDX report signing key, or an Arm CCA platform attestation key). The private component (privAK) MUST NOT be extractable from the hardware security boundary. This key pair is the trust anchor for the protocol’s Evidence integrity.

Challenge Nonce (CN1, CN2): Ephemeral random values exchanged between client and server during the handshake using AuthHPKE. CN1 is generated by the client and sealed to the server's pubKEM. CN2 is generated by the server and sealed to the client's pubKEM. Together they provide mutual freshness and form the basis for attestation key derivation.

Session Binding Commitment (rdata / eat_nonce): The 32-byte SHA-256 digest SHA-256(pubIK_S || CN1 || CN2 || pubKEM_C) that binds the attester's identity key, both challenge nonces, and the client's ephemeral KEM key into a single compound nonce. No single party controls all four inputs. This value is conveyed in the platform-specific report data field of the TEE attestation report (where it is referred to as rdata) and as the eat_nonce claim when serialized as an EAT. Both carry the identical digest; the distinction is purely one of serialization layer.

Attestation Key Material (psk_attest): A symmetric key derived from both challenge nonces. psk_attest is used to encrypt Evidence during the handshake and is subsequently injected into the Extended Key Update to rotate session traffic keys.

Trust Authority (TA / CA): A trusted third party that signs the Identity Document, establishing a verifiable chain of trust. This document uses "CA" for simplicity.

Platform Measurements (quote): The set of cryptographic measurements characterizing the Trusted Computing Base (TCB) of the platform, reduced to a 32-byte SHA-256 commitment (the Evidence Commitment) for use as input to key derivation.

3. Protocol Architecture

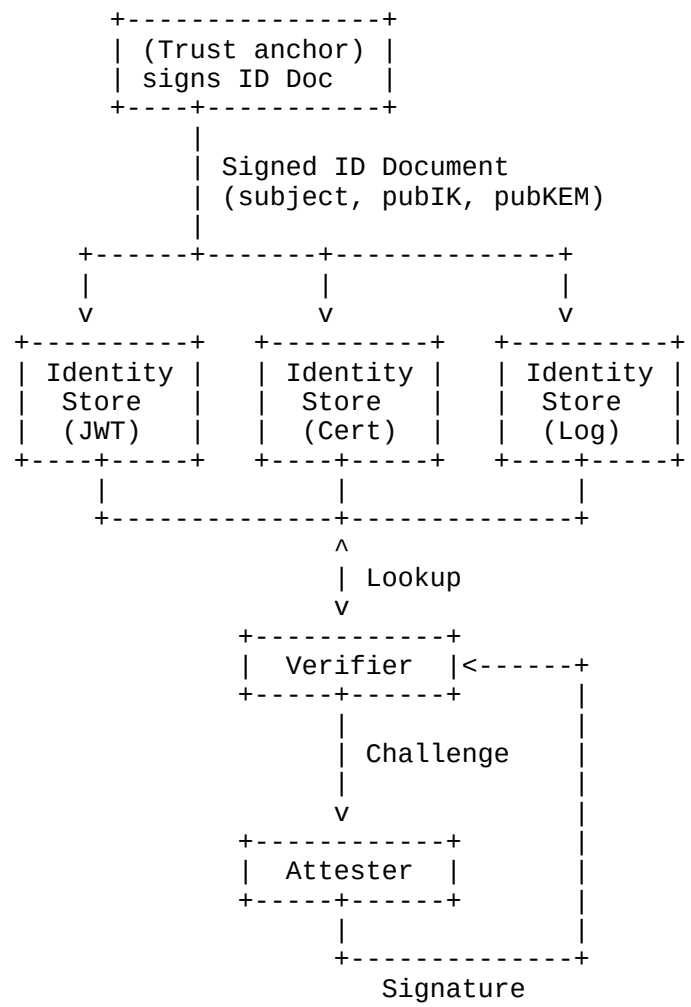
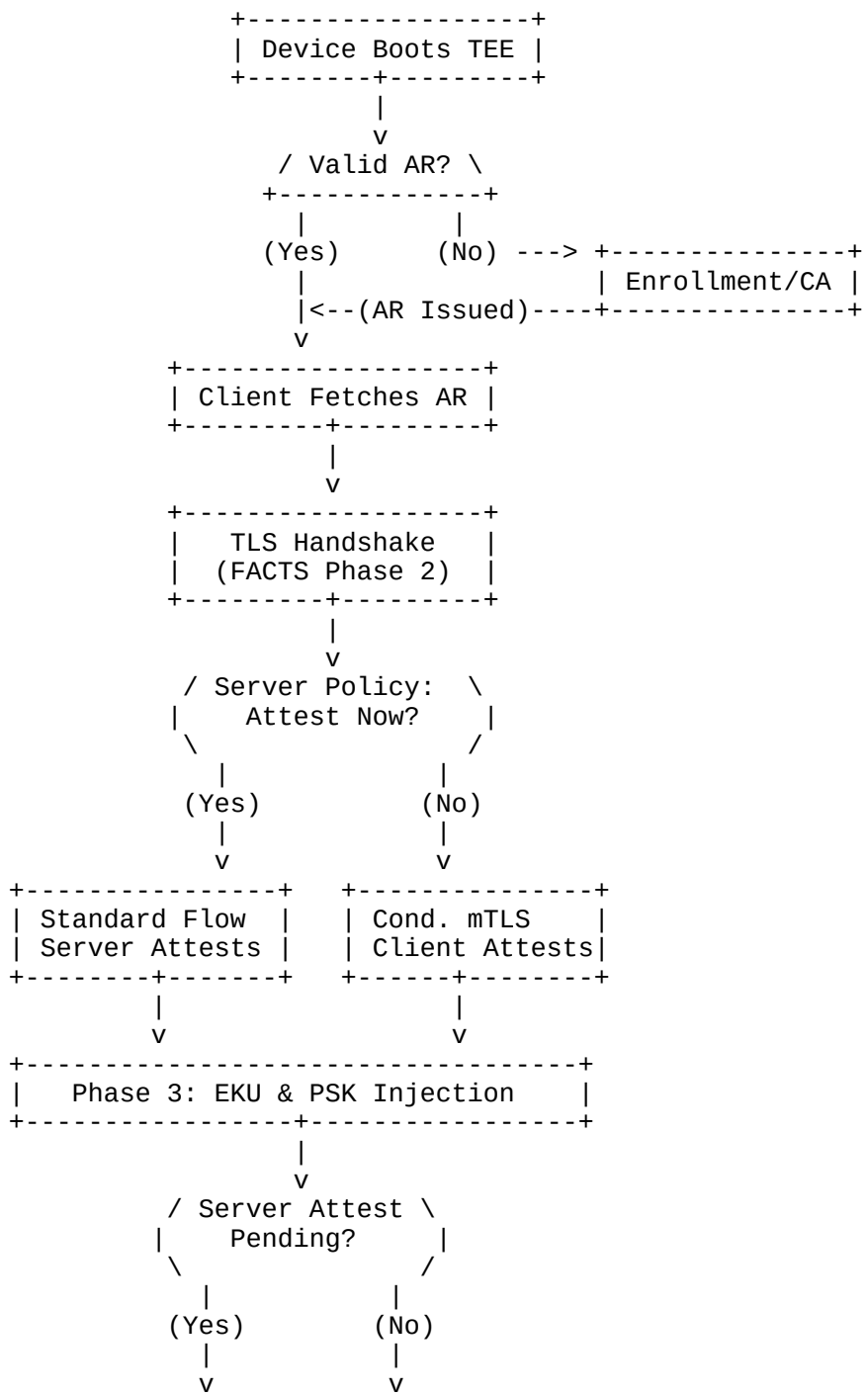


Figure 1: FACTS Architectural Topology

FACTS proceeds in three phases, preceded by an enrollment step that executes once per server instance. The protocol's trust model rests on two independently operating dimensions: the attestation ordering (which peer attests first) and the appraisal topology (how evidence is evaluated). These dimensions combine freely.


Device Boots TEE
Valid AR?
(Yes)
(No)
Enrollment/CA
(AR Issued)
Client Fetches AR
TLS Handshake
(FACTS Phase 2)
Server Policy:
Attest Now?
(Yes)
(No)
Standard Flow
Server Attests
Cond. mTLS
Client Attests
Phase 3: EKU & PSK Injection
Server Attest
Pending?
(Yes)
(No)

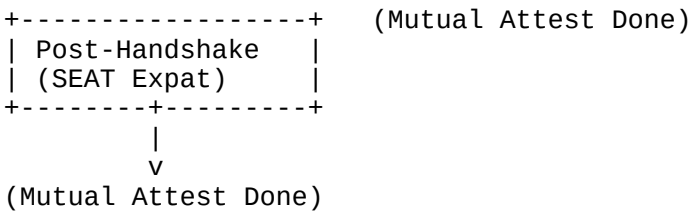


Figure 2: FACTS Decision Flow

Enrollment. A device with no valid AR generates its identity and encapsulation key pairs and produces an EAT [RFC9711] signed by its hardware-bound attestation key. The Verifier appraises the platform measurements and, if accepted, issues an AR carrying the dual-key binding: cnf [RFC7800] for pubIK (_signing_) and attested_kem for pubKEM (_encapsulation_). This AR is distributed via DNS, HTTPS, or any untrusted channel; the Verifier's signature protects integrity. See Section 9 for the enrollment protocol and Section 10 for the AR and EAT claim structures.

Phase 1 -- Identity Lookup. The client fetches and verifies the AR, extracting pubIK_S and pubKEM_S. It generates an ephemeral pubKEM_C, unique to this session and never reused.

Phase 2 -- Attested TLS Handshake. The client seals a challenge nonce (CN1) to pubKEM_S; the server proves possession of privKEM_S by unsealing it, then seals its counter-challenge (CN2) to the client's ephemeral pubKEM_C. Both peers derive psk_attest.

The handshake then branches on server policy. In the standard flow, the server includes its encrypted evidence in the Certificate message. In the conditional _mTLS_ flow, the server withholds its evidence and issues a facts_attest_req, requiring the client to attest first.

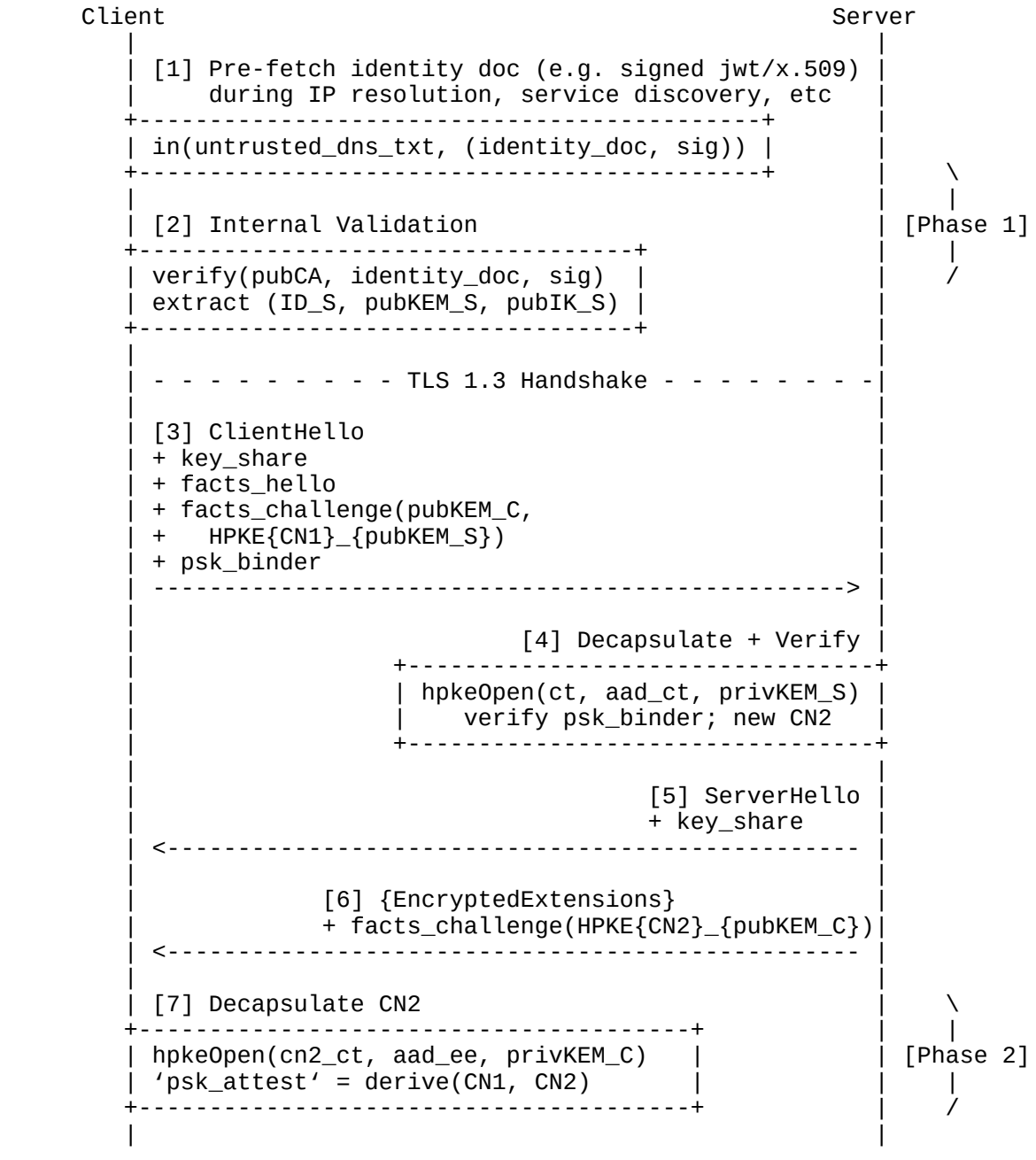
The choice between Passport and Background Check topologies [RFC9334] is orthogonal to this ordering and applies independently to either flow.

Phase 3 -- EKU with PSK Injection. After evidence appraisal, an Extended Key Update [I-D.ietf-tls-extended-key-update] mixes psk_attest into the traffic keys. Any adversary lacking privKEM_C cannot derive the rotated keys and is evicted.

Post-Handshake. If the server deferred attestation, it delivers evidence via Exported Authenticators [I-D.fossati-seat-expat] over the rotated channel. Periodic reattestation is available to any flow via the same mechanism.

4. Protocol Flow

The following diagram illustrates the 3 phase flow:



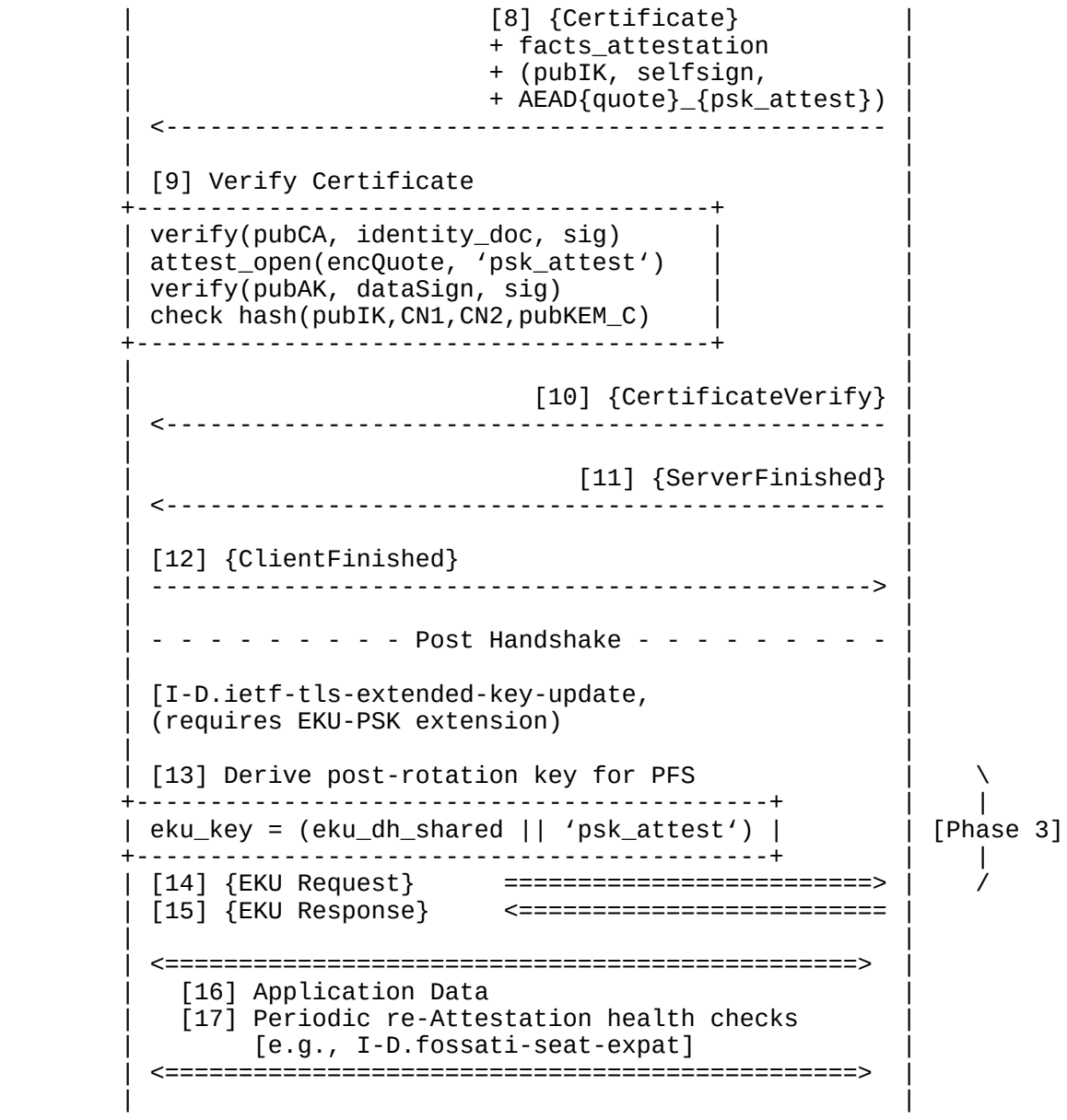


Figure 3: The 3 phase flow of FACTS-TLS

4.1. Split-Session Alternative

An alternative design uses a sacrificial bootstrap handshake followed by PSK-DHE resumption on a new TCP connection, rather than EKU within a single connection.

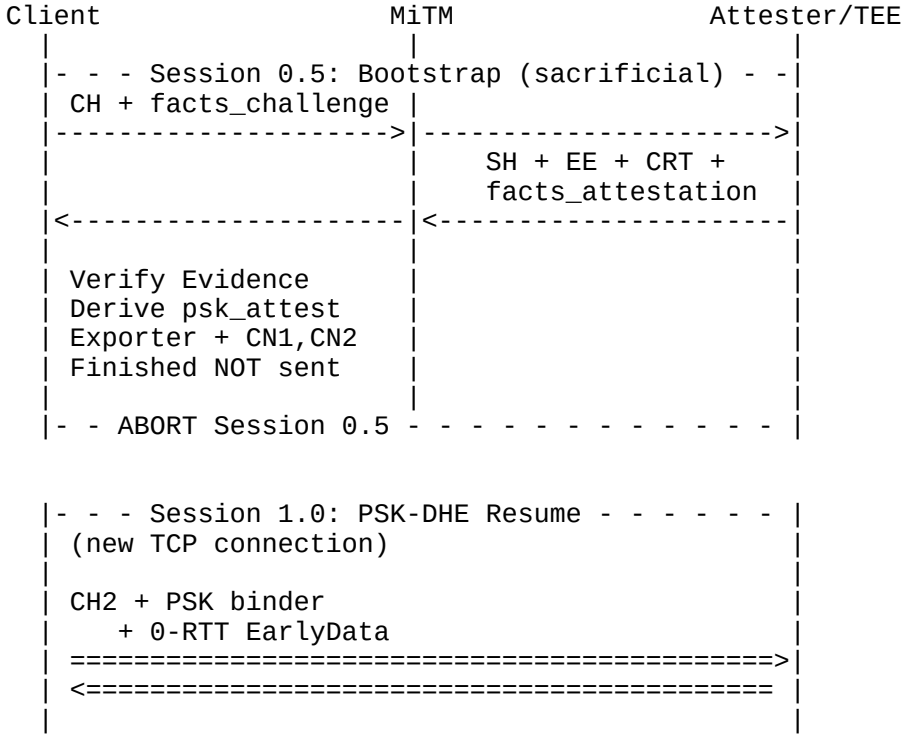


Figure 4: Split-Session Alternative: Bootstrap + PSK-DHE Resume

The EKU-based design is the practical extension of this concept: it replaces the connection tear-down and PSK-DHE resumption with an in-band EKU exchange, saving one TCP round-trip (SYN/SYN-ACK) plus the second ClientHello flight. Both produce equivalent post-rotation security. The split-session variant is useful for legacy stacks without EKU support.

5. Key Schedule Integration

FACTS integrates with the TLS 1.3 key schedule at two points: the Early Secret (for PSK binder validation) and the Extended Key Update (for traffic key rotation).

5.1. Early Secret Binding

CN1 is used to derive a PSK that feeds into the Early Secret computation, enabling PSK binder validation in the ClientHello.

```
PSK          = HKDF-Extract(salt=0, IKM=CN1)
Early Secret = HKDF-Extract(salt=0, IKM=PSK)
binder_key   = Derive-Secret(Early Secret, "res binder", "")
binder       = HMAC(binder_key, Transcript(ClientHello_truncated))
```

5.2. psk_attest Derivation

Both peers derive the attestation key material from the concatenation of both challenge nonces:

```
psk_attest = HKDF-Expand-Label(
    HKDF-Extract(salt=0, IKM=CN1 || CN2),
    "facts:v1:psk", "", Hash.length)
```

psk_attest serves two purposes: it encrypts Evidence during the handshake (via AEAD), and it is injected into the EKU key schedule after Evidence appraisal.

5.3. AAD Construction

Two distinct AAD values are used, one per HPKE operation. Both MUST be computed as specified here; deviations invalidate the session-binding properties of the protocol.

aad_ct — used when sealing CN1 in facts_challenge (ClientHello):

```
aad_ct = Hash(pubKEM_S || ClientHello.random || NegotiationOffer)
```

This binds the CN1 ciphertext to the target server’s KEM key, the client’s fresh random, and the negotiated parameters. An adversary cannot replay the ciphertext to a different server or across sessions.

aad_ee — used when sealing CN2 in facts_challenge (EncryptedExtensions):

```
aad_ee = Hash(ClientHello || ServerHello)
```

This binds the CN2 ciphertext to the full handshake transcript up to that point (log_SH). The CN2 ciphertext is therefore specific to the handshake in flight and cannot be transferred to any other session.

Implementations MUST use the negotiated handshake hash algorithm for both computations. The NegotiationOffer in aad_ct is the offer value from the key exchange negotiation as carried in the ClientHello.

6. Evidence Binding

The attester binds the TEE Evidence to the TLS session by including session-specific data in the TEE-signed attestation report.

6.1. Report Data Construction

```
rdata = SHA-256(pubIK_S || CN1 || CN2 || pubKEM_C)
```

The report data (rdata) binds the Evidence to both the attester’s identity key, the mutually exchanged nonces and the Client’s own ephemeral per-session pubKEM KEM.

7. Extended Key Update with PSK Injection

After Evidence appraisal succeeds, the client initiates an Extended Key Update [I-D.ietf-tls-extended-key-update] that injects psk_attest into the key schedule, rotating the traffic keys to values that incorporate attestation-derived keying material.

7.1. Procedure

1. The client generates an ephemeral key pair for the EKU exchange: (eku_x, g^eku_x).

2. The client sends an EKU request containing g^eku_x.

3. The server generates its ephemeral key pair: (eku_y, g^eku_y).

4. Both peers compute the EKU Diffie-Hellman shared secret: eku_dh = DH(g^eku_x, eku_y) (equivalently, DH(g^eku_y, eku_x)).

5. Both peers construct the combined input keying material: combined_ikm = eku_dh || psk_attest

6. Both peers derive new traffic keys:

```
combined_ikm     = eku_dh_shared || psk_attest

stored_secret    = Derive(main_secret_N, "derived", "")
main_secret_N+1  = HKDF-Extract(stored_secret, combined_ikm)

eku_cts = Derive(main_secret_N+1, "c_app_traffic_N+1", eku_log)
eku_sts = Derive(main_secret_N+1, "s_app_traffic_N+1", eku_log)
```

Figure 5: FACTS Key Schedule for Extended Key Update

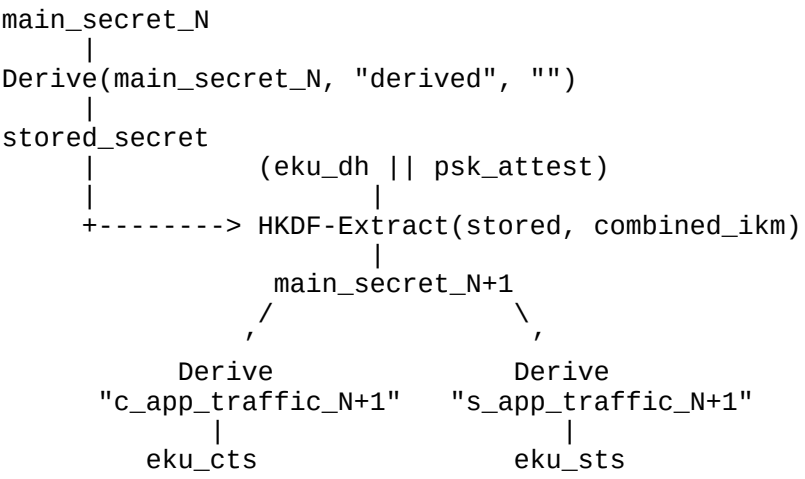


Figure 6: EKU Key Schedule with PSK Injection

Because combined_ikm includes psk_attest, which requires knowledge of CN2, any entity that cannot derive CN2 (because it lacks privKEM_C to unseal the server's Ctx2) cannot compute main_secret_N+1 or any traffic keys derived from it.

7.2. Interaction with draft-ietf-tls-extended-key-update

The PSK injection mechanism defined in this section extends the EKU key schedule specified in [I-D.ietf-tls-extended-key-update]. The standard EKU derivation uses only the DH shared secret as IKM; FACTS concatenates psk_attest to the DH shared secret before extraction. All other aspects of the EKU exchange (message formats, transcript construction, key switch timing) remain as specified in [I-D.ietf-tls-extended-key-update].

8. Extension Definitions

8.1. The facts_hello Extension

The facts_hello extension is carried in the ClientHello and serves as the capability signal for FACTS. Its presence indicates that the client supports the FACTS protocol and is prepared to send facts_challenge. It optionally carries a hw_id hint that identifies the hardware platform, enabling the CA or Verifier to select the correct endorsement key bundle, manufacturer reference values, and appraisal policy before the attestation quote arrives.

```
   enum {
       facts_hello_v1(1),
       (255)
   } FactsHelloVersion;

   enum {
       no_hw_id(0),
       hw_id_present(1),
       (255)
   } FactsHelloFlags;

   struct {
       FactsHelloVersion version;        /* MUST be facts_hello_v1 */
       FactsHelloFlags   flags;
       select (FactsHello.flags) {
           case hw_id_present:
               opaque hw_id<1..2^16-1>; /* opaque platform identifier */
           case no_hw_id:
               /* empty */
       };
   } FactsHello;
```

version MUST be facts_hello_v1. Servers that do not recognize the version MUST ignore the extension.

hw_id is an opaque value whose encoding is platform-specific. It is not a credential and is not secret. It is transmitted in the clear in ClientHello. Deployments with elevated privacy requirements SHOULD use no_hw_id and rely on the facts_attest_req context for platform identification, or deliver platform identity inside the encrypted facts_attestation payload instead.

8.2. The facts_challenge Extension

The facts_challenge extension carries HPKE-sealed challenge nonces. It appears in two messages with distinct structures and HPKE modes.

8.2.1. ClientHello Variant

```
struct {
    opaque initiator_id<0..2^16-1>;  /* ID_C: connecting peer identity */
    opaque pubKEM_C<1..2^16-1>;      /* ephemeral per-session KEM public key */
    opaque ct<1..2^16-1>;            /* sealed CN1 */
} FactsChallengeClient;
```

The ciphertext ct MUST be computed as:

```
/* Runtime flow — HPKE Mode 0 (RFC 9180 §5.1) */
ct = HPKE-Seal(plaintext=CN1, aad=aad_ct, pkR=pubKEM_S)

/* Enrollment flow — Auth HPKE Mode 2 (RFC 9180 §5.2.1) */
ct = AuthHPKE-Seal(plaintext=CN1, aad=aad_ct, pkR=pubKEM_S, skS=privLTK_C)
```

where aad_ct is computed per Section 5.3.

The server MUST verify that pubKEM_C is a valid public key for the negotiated KEM group before proceeding.

8.2.2. EncryptedExtensions Variant

```
struct {
    opaque ct<1..2^16-1>;            /* sealed CN2 */
} FactsChallengeServer;
```

The ciphertext ct MUST be computed as:

```
/* Runtime flow — HPKE Mode 0 */
ct = HPKE-Seal(plaintext=CN2, aad=aad_ee, pkR=pubKEM_C)

/* Enrollment flow — Auth HPKE Mode 2 */
ct = AuthHPKE-Seal(plaintext=CN2, aad=aad_ee, pkR=pubKEM_C, skS=privIK_S)
```

where aad_ee is computed per Section 5.3.

The facts_challenge extension in EncryptedExtensions MUST NOT appear unless the corresponding facts_challenge was present in the ClientHello. Servers that receive facts_challenge in ClientHello without a facts_hello extension MUST abort with missing_extension.

8.3. The facts_attestation Extension

The facts_attestation extension is carried in the Certificate message and delivers the encrypted TEE Evidence along with the self-signed key binding. It is the sole extension defined by this document that is permitted in the Certificate message.

```
struct {
    opaque pubIK<1..2^16-1>;         /* attester's identity public key */
    opaque selfsign<1..2^16-1>;      /* sign(privIK, (pubIK || encEvidence)) */
    opaque encEvidence<1..2^16-1>;   /* AEAD{CMW-wrapped quote}_{psk_attest} */
} FactsAttestation;
```

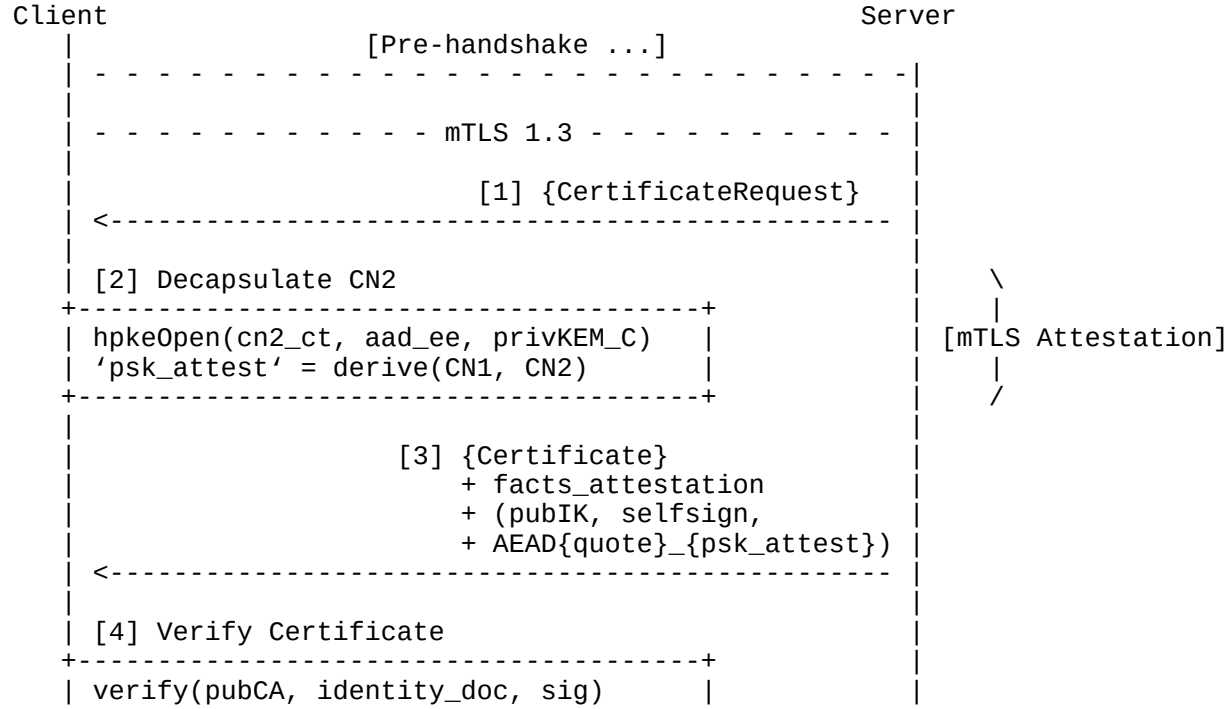
pubIK MUST match the public key in the Certificate's leaf entry. The client MUST abort with illegal_parameter if they differ.

selfsign is a signature over the concatenation of pubIK and encEvidence, produced by privIK. It binds the identity key to the encrypted Evidence payload within the same Certificate message, preventing substitution of one attester's Evidence under another attester's certificate.

encEvidence is the attestation quote encrypted under psk_attest using AEAD (ChaCha20-Poly1305 per [RFC8439]). The plaintext is a RATS CMW-wrapped Evidence payload per [I-D.ietf-rats-msg-wrap].

9. Enrollment

Certificate enrollment has a foundational chicken-and-egg problem in TEE environments: a client needs credentials to authenticate to a CA, but obtaining credentials is the reason it is connecting to the CA in the first place. FACTS uses the mTLS handshake wire format to deliver the client's X.509 certificate over a channel that is cryptographically proven to terminate at the verified hardware boundary: the CA's challenge nonce (CN2) is sealed to the client's ephemeral pubKEM_C, the client's TEE evidence binds that nonce and the client's fresh pubLTK_C in the hardware-signed quote, and the resulting certificate arrives over traffic keys that only a party holding privKEM_C could have derived.




attest_open(encQuote, 'psk_attest')
verify(pubAK, dataSign, sig)
check hash(pubIK,CN1,CN2,pubKEM_C)
[5] {CertificateVerify}
[6] {ServerFinished}
[7] {Certificate}
+ facts_attestation
+ (pubIK, selfsign,
+ AEAD{client_quote}_{psk_attest})
[8] Verify Client Cert
attest_open(client_encQuote,
psk_attest)
verify(pubAK, dataSign, sig)
check hash(pubIK_S, CN1,
CN2, pubKEM_C)
[9] {CertificateVerify}
[10] {ClientFinished}
- - - - - - - - - Post Handshake - - - - - - - - -
[I-D.ietf-tls-extended-key-update,
(requires EKU-PSK extension)
[11] {EKU Request}
[12] {EKU Response}
[13] Application Data
+ issued AR (ID_C, pubIK_C, pubKEM_C)
[14] Periodic re-Attestation health checks
[e.g., I-D.fossati-seat-expat]
[Attestation
Result]

Figure 7: Attestation over mTLS supplying RATS AR

9.1. The facts_attest_req Extension

The facts_attest_req extension is carried in the CertificateRequest message. It signals that the server requires client attestation before it will release its own Evidence, and communicates the Evidence format requirements the client must satisfy. CA enrollment is one application of this signal; privacy-gated disclosure to unauthenticated clients is another.

9.1.1. Extension Structure

```
struct {
    ExtensionType extension_type = facts_attest_req; /* 0xTBD1 */
    opaque extension_data<0..2^16-1>;
} Extension;

struct {
    FactsAttestRequestVersion version;
    FactsEvidenceFormat supported_formats<1..2^8-1>;
    opaque responder_identity<0..2^16-1>;  /* responder's ID_S, for rdata binding */
    opaque request_context<0..2^8-1>;      /* opaque, echoed in client response */
} FactsAttestRequestParams;

enum {
    facts_attest_req_v1(1),
    (255)
} FactsAttestRequestVersion;

enum {
    eat_cbor(1),       /* EAT CBOR per RFC 9711 */
    eat_json(2),       /* EAT JSON */
    cmw(3),            /* RATS CMW per I-D.ietf-rats-msg-wrap */
    csr_attestation(4),/* I-D.ietf-lamps-csr-attestation attribute format */
    (255)
} FactsEvidenceFormat;
```

version identifies the facts_attest_req protocol version. Clients that do not recognize the version MUST abort with handshake_failure.

supported_formats is an ordered list of Evidence formats the responder is willing to accept, in descending preference. The client MUST select the first format it supports. If no common format exists, the client MUST abort with handshake_failure.

responder_identity carries the responder's ID_S value as it appears in its Identity Document. The client MUST verify this matches the ID_S bound in the pre-fetched Identity Document. This closes the identity confirmation loop: the responder explicitly asserts its identity inline rather than relying solely on the certificate chain.

request_context is an opaque value chosen by the responder. The client MUST echo it verbatim in its response. It serves as a lightweight correlation handle and MAY encode responder-internal state such as a request identifier or policy selector. It MUST NOT be used as a freshness mechanism; freshness is provided by CN1 and CN2.

9.1.2. Server Behavior

A server sending facts_attest_req in CertificateRequest:

1. MUST include facts_challenge in EncryptedExtensions. A facts_attest_req without a corresponding facts_challenge in EncryptedExtensions is a protocol error; the client MUST abort with missing_extension.

2. MUST deliver a valid X.509 certificate in its own Certificate message.

3. MUST omit facts_attestation from its own Certificate message. Server Evidence delivery is deferred to post-handshake Exported Authenticators [I-D.fossati-seat-expat] once the client has been verified. See Section 12.4.

9.1.3. Client Behavior

Upon receiving a CertificateRequest containing facts_attest_req, the client:

1. Verifies version is supported.

2. Selects an Evidence format from supported_formats.

3. Verifies responder_identity matches the pre-fetched Identity Document.

4. Derives psk_attest from CN1 and CN2 per Section 5.

5. Produces TEE Evidence bound to the session parameters, encrypts it under psk_attest, and delivers it in facts_attestation within its Certificate message per Section 8.3.

The enrollment-specific client steps (key pair generation, CRTEnroll construction, rdata binding), defined earlier in this section, are a specific application of this general mechanism.

A client that is unable or unwilling to attest MUST send an empty Certificate message and MUST NOT send facts_attestation. The server MUST then abort with certificate_required.

9.1.4. Relationship to cert_req_context

The standard TLS 1.3 CertificateRequest carries a certificate_request_context field that the client echoes in its Certificate message to prevent replay across connections. The request_context in facts_attest_req is distinct from and supplementary to this field. Both MUST be echoed. Implementations MUST NOT conflate them.

10. EAT/AR Claims as Identity Documents

FACTS uses two identity documents to convey device identity and verifier appraisal: an Entity Attestation Token (EAT) [RFC9711] produced by the attesting device, and an Attestation Result (AR) produced by the verifier. The EAT is signed by the device attestation key (privAK). The AR is signed by the verifier under a separate trust anchor.

Both documents MAY be serialized in any format supported by the negotiated FactsEvidenceFormat; JWT serialization is one valid option. The claims and key binding structures defined in this section apply regardless of serialization format.

The AR’s cnf-based key binding follows the same proof-of-possession pattern established for OAuth token binding via mutual TLS [RFC8705] and DPoP [RFC9449], extended with an attested_kem claim that provides a second, independent proof-of-possession channel through HPKE nonce encapsulation.

10.1. EAT Structure

The EAT payload MUST include the following FACTS-specific claims:

```
struct {
    string   sub;         // Stable device identifier bound to pubIK and pubKEM
    uint64   iat;
    uint64   nbf;
    uint64   exp;
    bytes    eat_nonce;   // SHA-256(pubIK_S || CN1 || CN2 || pubKEM_C),
                          //   base64url-unpadded;
    array    keys;        // JWKS array carrying exactly two JWK entries
    string   eat_profile; // EAT profile identifier per RFC 9711
} EatClaims;
```

All other claims MUST conform to [RFC9711].

eat_nonce carries the session binding commitment defined in Section 6. Its value MUST be the base64url-unpadded encoding of SHA-256(pubIK_S || CN1 || CN2 || pubKEM_C). Implementations MUST NOT place a verbatim verifier challenge in this field. How this value is conveyed to the underlying platform attestation primitive is defined in per-platform companion documents.

The keys member is a JWKS array [RFC7517] carrying exactly two JWK entries:

```
keys: [
    { "kty": "OKP", "crv": "Ed25519", "use": "sig",
      "kid": "pubIK_S",  "x": "<pubIK key material>" },
    { "kty": "OKP", "crv": "X25519",  "use": "enc",
      "kid": "pubKEM_S", "x": "<pubKEM key material>" }
]
```

The first entry carries pubIK_S with "use": "sig" and the second carries pubKEM_S with "use": "enc". Both MUST use the OKP key type per [RFC8037]. The EAT presents these keys as part of the device’s attested evidence; they are bound to the platform measurements by the TEE’s signature over the complete EAT.

Platform-specific EAT claims (ueid, oemid, hwmodel, hwversion, swname, swversion, secboot, dbgstat, and claims defined by the applicable attestation profile such as PSA IoT) MAY be included and are interpreted per the applicable platform profile.

10.2. AR Structure

The AR is issued by the verifier during enrollment alongside the X.509 certificate. It enables the passport model [RFC9334] for subsequent FACTS handshakes, permitting the device to present a cached attestation result without a live background-check round-trip to the verifier.

The AR payload MUST include:

```
struct {
    string   iss;           // Verifier identity
    string   sub;           // MUST match sub from the appraised EAT
    string   aud;           // Intended relying party audience
    object   cnf;           // Confirmation claim per RFC 7800
    object   attested_kem;  // Attested encapsulation key
} ArClaims;
```

The cnf claim [RFC7800] MUST contain a jwk member carrying a single JWK entry [RFC7517] with "use": "sig" carrying pubIK_S:

```
"cnf": {
    "jwk": { "kty": "OKP", "crv": "Ed25519", "use": "sig",
             "kid": "pubIK_S", "x": "<pubIK key material>" }
}
```

A relying party that receives an AR MUST verify that the cnf.jwk key matches the public key presented in the TLS Certificate message. This is the standard proof-of-possession confirmation: CertificateVerify proves the server holds the private key corresponding to the confirmed public key.

The attested_kem claim carries the encapsulation key as a standalone JWK with "use": "enc":

```
"attested_kem": {
    "kty": "OKP", "crv": "X25519", "use": "enc",
    "kid": "pubKEM_S", "x": "<pubKEM key material>"
}
```

This key is the target for the client’s HPKE-sealed challenge nonce (CN1) in the facts_challenge extension. Successful unsealing by the server constitutes proof-of-possession of the attested encapsulation key. Additional appraisal claims are implementation-specific and out of scope for this document.

11. Security Considerations

11.1. Idealized deployment topology assumptions

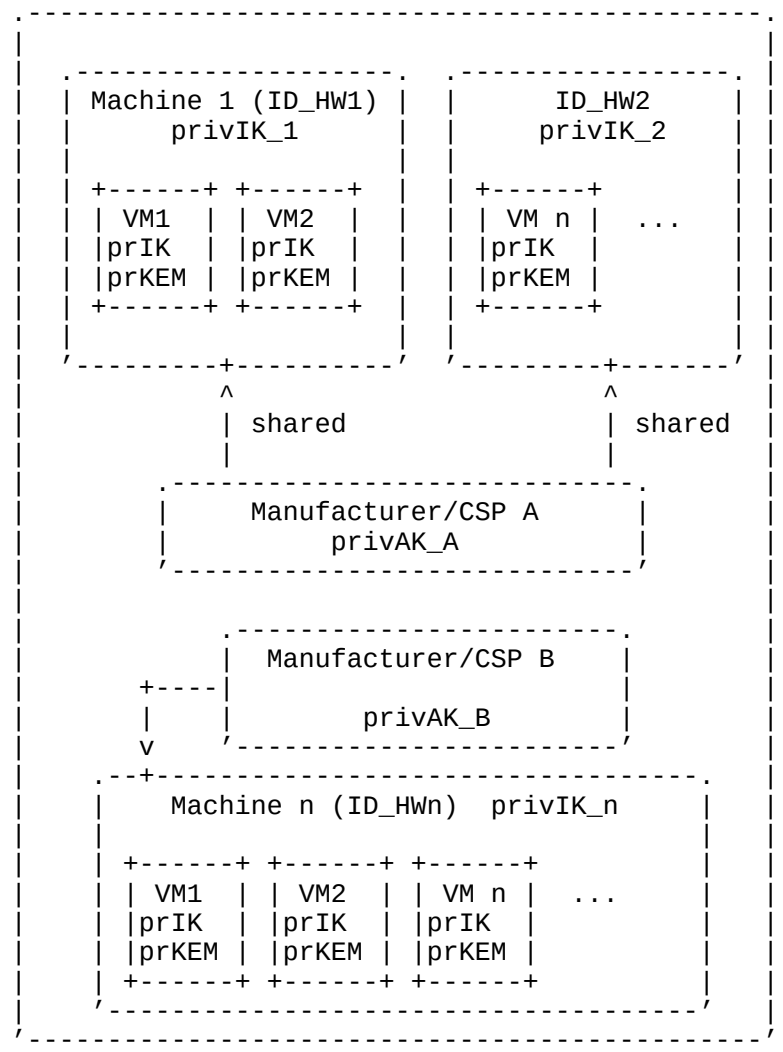


Figure 8: Infrastructure Topology

11.2. Threat Model

FACTS is designed to mitigate compromise under adversarial conditions:

* The adversary is an active MitM: they can intercept, inject, modify, and replay protocol messages on the network.

* The distribution channel for AR is untrusted and may serve attacker-controlled content (such as DNS without DNSSEC).

* Various states of compromise are tested including combinations where privAK is compromised while privIK and privKEM are secured, or when privIK and privKEM are compromised but privAK holds.

11.2.1. ProVerif model (Work in Progress)

A comprehensive ProVerif model, adapted from the open-source release of the [ID-Crisis] formal analysis of [I-D.fossati-tls-attestation] has been extended to evaluate the security properties of FACTS over TLS 1.3 [FACTS-ProVerif].

11.3. Key Separation

The attacks demonstrated in [ID-Crisis] arise from using a single key for both TLS authentication and attestation binding, enabling an adversary to relay evidence across sessions. FACTS separates these roles:

* The IK signs the TLS transcript (CertificateVerify) and the selfsign binding. Compromise of privIK enables TLS impersonation but does not yield a valid quote.

* The EK decapsulates challenge nonces and enables psk_attest derivation. Compromise of privKEM enables decryption of the quote but does not enable transcript forgery.

* The AK signs the TEE quote. Compromise requires breaching the hardware security boundary.

An adversary holding privKEM and privIK simultaneously can impersonate the TLS layer but cannot produce a quote with valid rdata for a session they did not participate in, because rdata binds (pubIK, CN1, CN2, pubKEM_C) and CN2 is sealed to the client’s ephemeral pubKEM_C which the adversary does not control.

11.4. Compound Authentication

The FACTS ProVerif model [FACTS-ProVerif] indicates compound injective-agreement between the Client and Server Pre-Finish ("Property G-C2") holds even when the private Attestation Key (AK) has been leaked. This suggests that the compound authentication property - both for the TLS session and the attestation evidence - refer to the same endpoint even under privAK compromise.

11.5. Pre-Handshake Identity Binding

The Identity Document distributed via DNS closes the identity/key separation gap identified in [ID-Crisis]. By binding pubKEM and pubIK to the server identity under a CA signature before the handshake begins, the client can verify that the keys used in the handshake correspond to the expected server identity. An adversary presenting keys not bound in a valid Identity Document will fail client verification.

12. Privacy Considerations

12.1. DNS Lookup Exposure

Pre-fetching the Identity Document via DNS reveals the client's intent to connect to the target server to any observer with visibility into DNS traffic. This is equivalent to the exposure present in any DNS resolution prior to TLS establishment. Deployments with elevated privacy requirements SHOULD use encrypted DNS (DoH or DoT).

12.2. Evidence Disclosure

The TEE quote is encrypted under psk_attest, which is derived from the challenge nonces exchanged between the two peers. The quote is therefore not visible to passive network observers. The client obtains the decrypted quote after step [9] and appraises it directly (background-check model per [RFC9334]) or forwards it to a Verifier. In the latter case, the Verifier learns the server's platform measurements and authentication timing.

12.3. Identity Document Opacity

The Identity Document carries only public key values and standard JWT metadata. It does not contain platform measurements, hardware identifiers, or personally identifiable information. Distribution via DNS does not create a correlation surface beyond the server's domain name.

12.4. Conditional Evidence Disclosure

[I-D.fossati-seat-expat] observes that when the server acts as Attester, it can require client authentication before releasing Evidence, limiting exposure of hardware-level claims to authorized clients. This document adopts that principle within the handshake.

In the standard FACTS flow, any client that completes the HPKE exchange obtains the server's decrypted TEE Evidence. The facts_attest_req extension (Section 9.1) allows the server to withhold its own Evidence and require client attestation first. Server Evidence is subsequently delivered via Exported Authenticators [I-D.fossati-seat-expat] post-handshake.

Deployments where the server's platform measurements are sensitive SHOULD use facts_attest_req for connections from unauthenticated or untrusted clients. The decision to require client-first attestation is a server-local policy choice.

13. IANA Considerations

13.1. TLS ExtensionType Registry

IANA is requested to register the following entries in the "TLS ExtensionType Values" registry [RFC8447]:

Value	Extension Name	TLS 1.3	DTLS-Only	Recommended
TBD1	facts_hello	CH	N	Y
TBD2	facts_challenge	CH, EE	N	Y
TBD3	facts_attestation	CT	N	Y
TBD4	facts_attest_req	CR	N	Y

Table 1

The facts_challenge extension is permitted in both ClientHello (CH) and EncryptedExtensions (EE). Its structure and HPKE mode differ per message as specified in Section 8.2. The presence of facts_challenge in EncryptedExtensions is conditional on its presence in the corresponding ClientHello, per [RFC8446].

The facts_attestation extension is permitted only in the Certificate message (CT). It carries an encrypted CMW-wrapped Evidence payload and a self-signed key binding as specified in Section 8.3.

14. References

14.1. Normative References

[I-D.ietf-rats-msg-wrap]
Birkholz, H., Smith, N., Fossati, T., Tschofenig, H., and D. Glaze, "RATS Conceptual Messages Wrapper (CMW)", Work in Progress, Internet-Draft, draft-ietf-rats-msg-wrap-23, 11 December 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-msg-wrap-23>.

[I-D.ietf-tls-extended-key-update]
Tschofenig, H., Txen, M., Reddy.K, T., Fries, S., and Y. Rosomakho, "Extended Key Update for Transport Layer Security (TLS) 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-extended-key-update-09, 18 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-extended-key-update-09>.

[I-D.ietf-tls-rfc8446bis]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-rfc8446bis-14, 13 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-rfc8446bis-14>.

[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>.

[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>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/rfc/rfc7800>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/rfc/rfc8032>.

[RFC8037] Liusvaara, I., "CFRG Elliptic Curve Diffie-Hellman (ECDH) and Signatures in JSON Object Signing and Encryption (JOSE)", RFC 8037, DOI 10.17487/RFC8037, January 2017, <https://www.rfc-editor.org/rfc/rfc8037>.

[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>.

[RFC8439] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 8439, DOI 10.17487/RFC8439, June 2018, <https://www.rfc-editor.org/rfc/rfc8439>.

[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>.

[RFC8447] Salowey, J. and S. Turner, "IANA Registry Updates for TLS and DTLS", RFC 8447, DOI 10.17487/RFC8447, August 2018, <https://www.rfc-editor.org/rfc/rfc8447>.

[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/rfc/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/rfc/rfc9334>.

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

14.2. Informative References

[FACTS-ProVerif] Ritz, N., "FACTS: ProVerif Model", March 2026, <https://github.com/nathanaelritz/formal-spec-id-crisis>.

[I-D.fossati-seat-early-attestation]
Sheffer, Y., Mihalcea, I., Deshpande, Y., Fossati, T., and T. Reddy.K, "Using Attestation in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", Work in Progress, Internet-Draft, draft-fossati-seat-early-attestation-03, 1 March 2026, <https://datatracker.ietf.org/doc/html/draft-fossati-seat-early-attestation-03>.

[I-D.fossati-seat-expat]
Sardar, M. U., Fossati, T., Reddy.K, T., Sheffer, Y., Tschofenig, H., and I. Mihalcea, "Remote Attestation with Exported Authenticators", Work in Progress, Internet-Draft, draft-fossati-seat-expat-02, 26 February 2026, <https://datatracker.ietf.org/doc/html/draft-fossati-seat-expat-02>.

[I-D.fossati-tls-attestation]
Tschofenig, H., Sheffer, Y., Howard, P., Mihalcea, I., Deshpande, Y., Niemi, A., and T. Fossati, "Using Attestation in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", Work in Progress, Internet-Draft, draft-fossati-tls-attestation-09, 30 April 2025, <https://datatracker.ietf.org/doc/html/draft-fossati-tls-attestation-09>.

[I-D.ietf-lamps-csr-attestation]
Ounsworth, M., Tschofenig, H., Birkholz, H., Wiseman, M., and N. Smith, "Use of Remote Attestation with Certification Signing Requests", Work in Progress, Internet-Draft, draft-ietf-lamps-csr-attestation-22, 11 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-csr-attestation-22>.

[ID-Crisis]
Sardar, M. U., Moustafa, M., and T. Aura, "Identity Crisis in Confidential Computing: Formal Analysis of Attested TLS", February 2026, <https://doi.org/10.1145/3779208.3785387>.

[RelayAttacks]
Sardar, M. U., Dubeyko, V., and J. Jacquet, "Relay Attacks in Intra-handshake Attestation for Confidential Agentic AI Systems", January 2026, <https://mailarchive.ietf.org/arch/msg/seat/x3eQxFjQFJLceae6l4_NgXnmsDY/>.

[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/rfc/rfc7519>.

[RFC8705] Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI 10.17487/RFC8705, February 2020, <https://www.rfc-editor.org/rfc/rfc8705>.

[RFC9261] Sullivan, N., "Exported Authenticators in TLS", RFC 9261, DOI 10.17487/RFC9261, July 2022, <https://www.rfc-editor.org/rfc/rfc9261>.

[RFC9449] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, <https://www.rfc-editor.org/rfc/rfc9449>.

Acknowledgments

Thanks to Sardar et al. for the open-source release of the Identity Crisis model and for the detailed relay attack disclosures to the community. Thank you to the authors and contributors who have worked tirelessly on the foundational body of work for attested TLS, and for their insights on the IETF mailing lists that helped shape this approach. A special additional thanks to Usama Sardar for sharing valuable time for discussions.

Author’s Address

Nathanael Ritz
Independent
Email: ietf@nritz.com