Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 18 October 2026

G. Rodriguez
Rolab

16 April 2026

Peer-to-Peer Presence Verification for Relationship-Bound Authorization
draft-rodriguez-h2h-presence-attestation-00

Abstract

Existing protocols authenticate users to services, negotiate session keys, or protect message content from eavesdroppers. None verify that a remote party is the same individual whose key was accepted during an earlier in-person exchange and has just now physically authorized a signature on the device holding that key. Advances in synthetic media make it increasingly difficult to trust unauthenticated audio, video, or text for sensitive authorizations.

This document defines transport-independent CBOR- and COSE-based objects that chain every remote interaction back to a bilateral in-person contact exchange -- not a certificate authority, identity provider, or key server. The protocol specifies Key Binding Objects, short-lived Session Credentials, replay-protected Signed Messages, a Presence Challenge requiring fresh platform-mediated user verification, and a Relationship Fingerprint for out-of-band key confirmation. It does not claim to prove biological humanness, legal identity, or voluntary action.

About This Document

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

The latest revision of this draft can be found at https://github.com/rolabtech/h2h. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-rodriguez-h2h-presence-attestation/.

Source for this draft and an issue tracker can be found at https://github.com/rolabtech/h2h.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Copyright Notice

Table of Contents

1. Introduction

1.1. Problem Statement

Existing protocols authenticate users to services (WebAuthn [WebAuthn], FIDO2 [FIDO2]), services to users (TLS [RFC8446]), or protect message content from eavesdroppers (Signal Protocol [Signal], MLS [RFC9420]). However, some deployments require a narrower but different property: one peer wishes to verify that the remote party is continuing a relationship previously anchored by an in-person exchange, and that the remote party has recently produced a local authorization event on the device bound to that relationship.

Examples include high-trust messaging, approval workflows, operator handoffs, and other interactions where account-level authentication alone is insufficient.

Recent advances in synthetic media and automated impersonation make unauthenticated audio, video, or text unreliable for sensitive approvals. This document does not detect synthetic media or prove biological humanness. Instead, it specifies interoperable signed objects and verification rules that bind sensitive authorizations to a prior relationship ceremony and to a fresh, platform-mediated local user-verification event.

1.2. Design Goal

This protocol does not prove a universally valid human identity. It lets a relying peer verify all of the following:

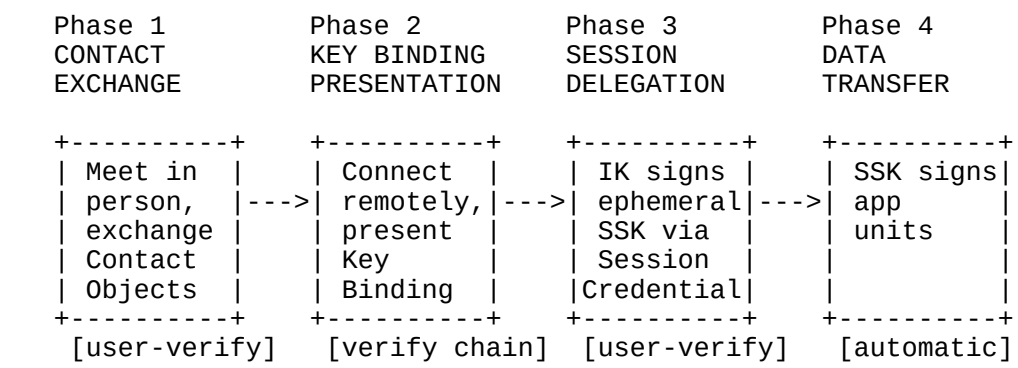
1. A public key was previously accepted during an in-person exchange.

2. The corresponding private key remains non-exportable and device-bound according to the local platform's key store semantics.

3. The remote device recently produced a signature using that key, or a key delegated by it, following a local user-verification event reported by the platform.

1.3. Overview

The protocol operates in four logical phases:



At any point: Presence Challenge --> fresh IK signature

Phases 1 and 3 require platform-mediated user verification (e.g., biometric or PIN) because they use the hardware-bound Identity Key. Phase 4 is automatic because the SSK is a software key delegated during Phase 3 -- no user interaction is needed for signing.

1. *Contact Exchange*: Two peers meet in person and exchange Contact Objects -- signed structures containing their Identity Keys, Transport Keys, and addressing information.

2. *Key Binding Presentation*: When connecting remotely, peers present Key Binding Objects and verify the trust chain: stored IK (from the in-person ceremony) = IK in binding = key that signed TK = TK of the transport peer.

3. *Session Delegation*: Peers create Session Credentials (IK signs an ephemeral SSK) for efficient signing without repeated user-verification prompts.

4. *Data Transfer*: Peers exchange Signed Messages (SSK signs application-defined units). Every signed unit chains back to the in-person ceremony.

At any point, either peer MAY issue a Presence Challenge requesting a fresh IK signature that requires a local user-verification event on the challenged device.

1.4. Scope

This document specifies:

* The Contact Object format and verification rules (Section 4)

* The Key Binding Object format and verification rules (Section 5)

* The Session Credential format and verification rules (Section 6)

* The Signed Message format and verification rules (Section 7)

* The Presence Challenge and Presence Response formats (Section 8)

* A Relationship Fingerprint computation (Section 9)

* Transport Key rotation guidance (Section 3.3)

* Protocol constants for object types, assurance values, and transport key algorithms

This document does not specify:

* Proof-of-personhood

* Legal, civil, or government identity

* Transport protocol selection or configuration
* End-to-end encryption
* Multi-party relationships
* Revocation or recovery
* Application-layer trust policy
* UX requirements beyond protocol-visible semantics

1.5. Reuse and Interoperability Scope

The objects and verification procedures in this document are intended for reuse across multiple peer-to-peer applications and transports that require relationship-bound authorization. This document does not define a single application protocol or user experience. Instead, it standardizes compact object formats and verification rules that can be embedded into messaging, approval, or rendezvous systems.

1.6. Trust Model Summary

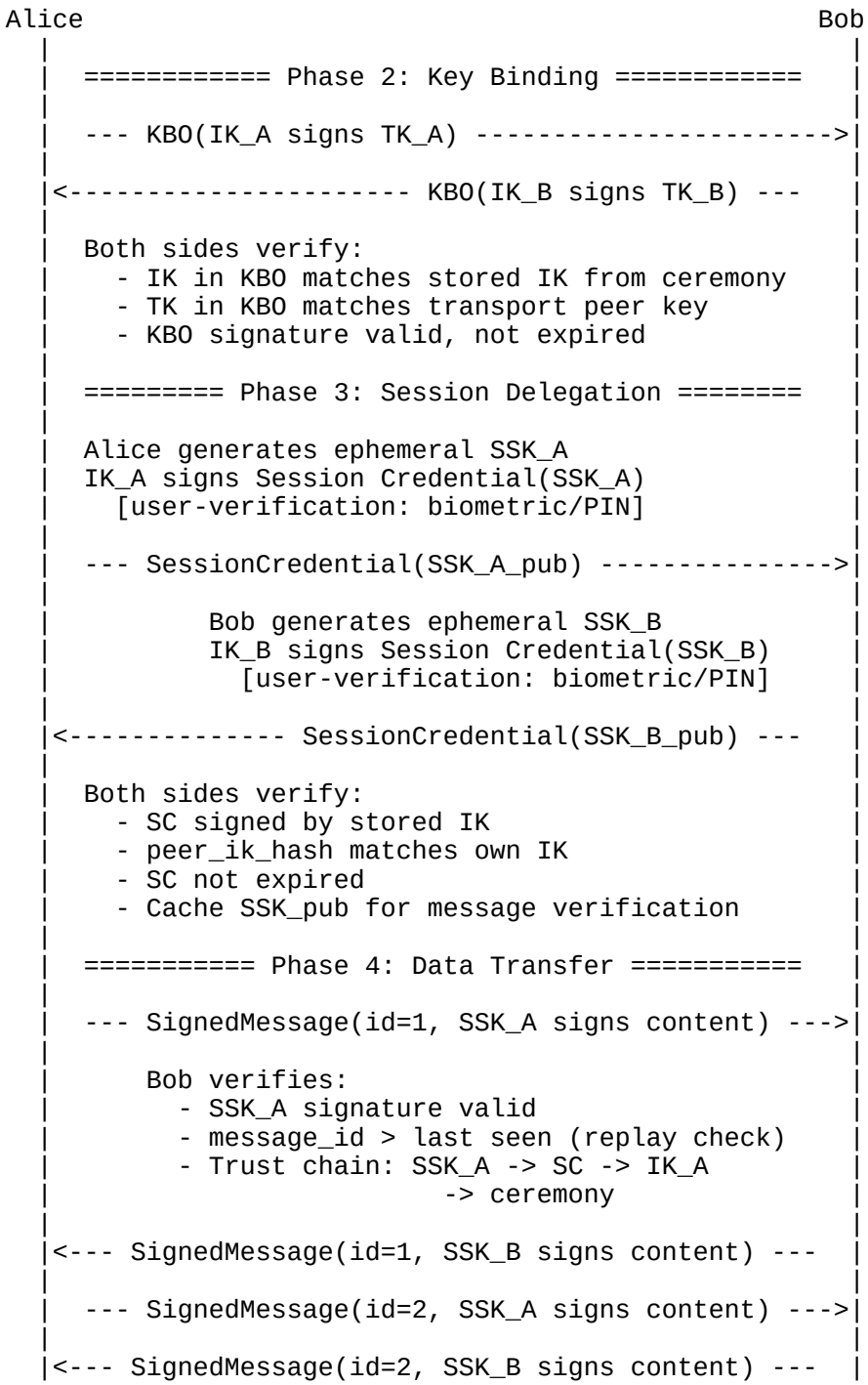
This protocol relies on two external trust assumptions.

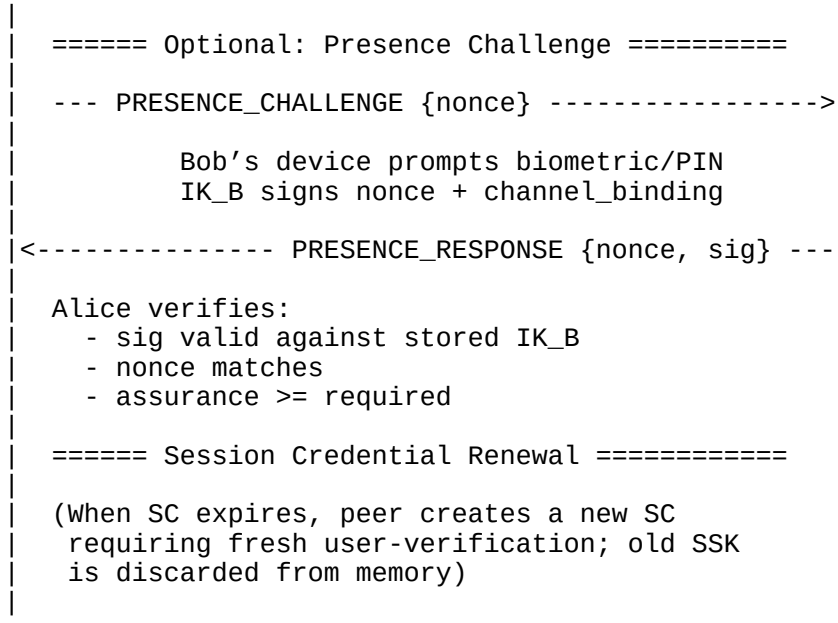
First, peers are assumed to have completed an earlier in-person ceremony and to have accepted each other’s Contact Objects during that ceremony.

Second, relying parties trust the local platform’s key store and user-verification semantics. A valid signature under the stored identity key indicates only that the platform permitted use of that key under its configured policy. It does not, by itself, prove biological humanness, voluntary action, or legal identity.

1.7. End-to-End Message Flow

The following ladder diagram shows a complete post-contact session between two peers, Alice and Bob, who have already exchanged Contact Objects in person. Phases 2 through 4 occur each time the peers reconnect remotely.


Alice
Bob
========== Phase 2: Key Binding ==========
--- KBO(IK_A signs TK_A) ----------
<---------- KBO(IK_B signs TK_B) ---
Both sides verify:
- IK in KBO matches stored IK from ceremony
- TK in KBO matches transport peer key
- KBO signature valid, not expired
========= Phase 3: Session Delegation ========
Alice generates ephemeral SSK_A
IK_A signs Session Credential(SSK_A)
[user-verification: biometric/PIN]
--- SessionCredential(SSK_A_pub) ---------->
Bob generates ephemeral SSK_B
IK_B signs Session Credential(SSK_B)
[user-verification: biometric/PIN]
<---------- SessionCredential(SSK_B_pub) ---
Both sides verify:
- SC signed by stored IK
- peer_ik_hash matches own IK
- SC not expired
- Cache SSK_pub for message verification
=========== Phase 4: Data Transfer ===========
--- SignedMessage(id=1, SSK_A signs content) --->
Bob verifies:
- SSK_A signature valid
- message_id > last seen (replay check)
- Trust chain: SSK_A -> SC -> IK_A
-> ceremony
<--- SignedMessage(id=1, SSK_B signs content) ---
--- SignedMessage(id=2, SSK_A signs content) --->
<--- SignedMessage(id=2, SSK_B signs content) ---



The diagram above is informative. Normative requirements for each object are specified in their respective sections (Section 4, Section 5, Section 6, Section 7, Section 8).

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.

2.1. Terminology

Identity Key (IK): A long-term P-256 ECDSA key pair [FIPS186-5] generated in a platform key store. The private key is non-exportable. Use of the private key requires local user verification according to platform policy.

Transport Key (TK): A transport-facing key pair used by a peer-to-peer transport or rendezvous mechanism. The TK is bound to the IK by a Key Binding Object. The algorithm is determined by the transport layer and is not constrained by this document.

Session Signing Key (SSK): A short-lived P-256 ECDSA key pair

generated in software, delegated by the IK for signing application-defined units. The SSK private key MUST be held in memory only and MUST NOT be persisted to disk. Not all application data requires SSK signing; see Section 7.1 for guidance on when Signed Messages add value beyond transport authentication.

Platform Key Store: A platform component that generates, stores, and uses cryptographic keys without exposing private key material to application software. Examples include secure enclaves, secure elements, TPMs, and trusted execution environments.

Contact Object: A COSE_Sign1 object exchanged during an in-person ceremony that conveys the sender’s Identity Key, Transport Key, and related metadata.

Key Binding Object (KBO): A COSE_Sign1 object that binds a Transport Key to a previously exchanged Identity Key.

Session Credential: A COSE_Sign1 object delegating signing authority from the IK to a short-lived SSK. Analogous to TLS Delegated Credentials [RFC9345].

Signed Message: A COSE_Sign1 object wrapping application content, signed by the SSK and verifiable via the Session Credential trust chain.

Presence Response: A COSE_Sign1 object produced in response to a challenge and bound to a connection context.

Assurance Value: A protocol-visible value reported by the signer describing the type of local verification event used for a signing operation. Assurance values are claims reported by the platform, not independently verifiable facts.

2.2. Assurance Values

This document defines three assurance values. Higher numeric values indicate stronger assurance:

Value	Label	Meaning
1	CREDENTIAL	The platform reported knowledge-based verification (PIN, password, pattern) for this signing operation.
2	BIOMETRIC	The platform reported biometric verification (face, fingerprint, iris) for this signing operation.
3	HARDWARE_VERIFIED	The signing operation includes hardware attestation evidence from the platform’s root of trust, proving the key is hardware-bound and the verification event is genuine. See Section 2.3.

Table 1

A response with assurance value N satisfies a requirement for assurance value M if and only if N >= M. Values 4 and above are reserved for future extensions; the N >= M rule ensures forward compatibility.

Values 1 and 2 are application-reported claims; a compromised application could misrepresent them. Value 3 includes cryptographic evidence from the platform’s hardware root of trust that cannot be forged by a compromised application (see Section 2.3). A verifier MUST treat assurance values as local policy inputs; they do not imply equivalent security across platforms. Relying parties MUST be informed of the assurance value and SHOULD make it visible to the user.

2.3. Hardware Attestation

When the assurance value is set to 3 (HARDWARE_VERIFIED), the signed object MUST include hardware attestation evidence in an additional field (attestation_evidence). This evidence is a platform-specific certificate chain or attestation object that allows the verifier to confirm:

1. The Identity Key was generated inside a genuine hardware security module (Secure Enclave, StrongBox, TPM).

2. The Identity Key is non-exportable, as enforced by the hardware -- not merely claimed by the application.

3. The key is configured to require user verification (biometric or credential) before each signing operation.

Implementations running on platforms that support hardware key attestation (e.g., Android Key Attestation, Apple Secure Enclave key attestation) SHOULD include attestation evidence in Contact Objects exchanged during the in-person ceremony. This allows the receiving peer to verify the Identity Key's hardware properties at the moment of initial trust establishment.

The attestation evidence is opaque to this specification. Its format and verification procedure are determined by the platform vendor. A verifier that does not recognize or cannot verify the attestation evidence MUST treat the assurance value as 2 (BIOMETRIC) rather than 3.

Including hardware attestation introduces a trust dependency on the platform vendor's certificate chain for the specific claim that the key store properties are genuine. It does not replace the in-person ceremony as the trust anchor for identity.

2.4. CBOR and COSE Conventions

All protocol payloads in this document are encoded as CBOR maps [RFC8949] with integer keys. All signed objects are encoded as COSE_Sign1 [RFC9052] with algorithm ES256 (ECDSA w/ SHA-256 using P-256, COSE algorithm identifier -7).

Implementations MUST use deterministic CBOR encoding as defined in Section 4.2 of [RFC8949] (Core Deterministic Encoding Requirements). This ensures that the same payload always produces the same byte sequence, which is critical for signature verification.

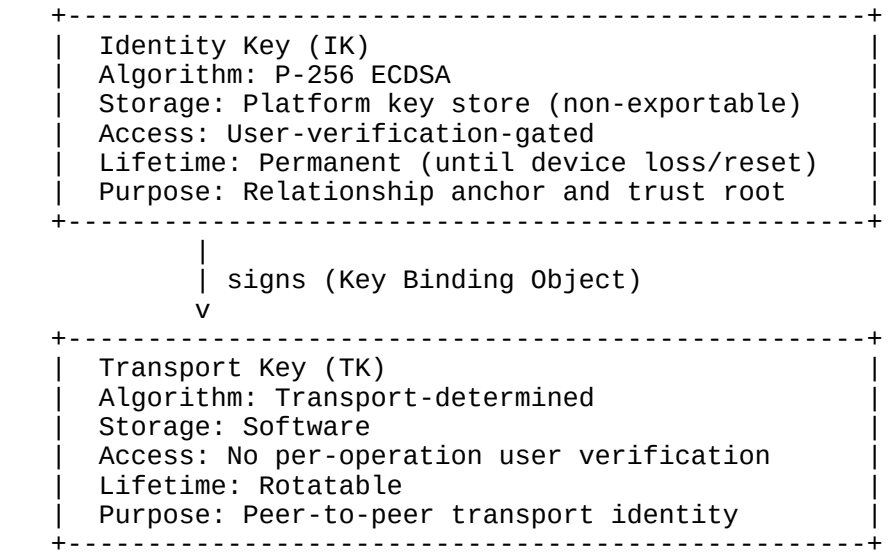
Each payload includes a structure_type discriminator at key 0. A verifier MUST check this value before processing any other payload semantics. This prevents type confusion attacks where a signed structure of one type is substituted for another.

Identity Key public keys are encoded as COSE_Key structures [RFC9053] with kty=2 (EC2) and crv=1 (P-256).

3. Identity and Binding Model

3.1. Two-Key Model

The protocol uses a long-term Identity Key and a rotatable Transport Key per peer.



The Identity Key is the relationship anchor. It is exchanged during the in-person ceremony and later used to verify continuity. Its private key never leaves the platform key store, and every use requires a local user-verification event.

The Transport Key identifies the peer at the transport layer. Transport operations (handshakes, session resumption) require a key that the transport library can use without per-operation user verification. A Key Binding Object (Section 5) links the TK to the IK, extending the trust anchor to the transport layer.

3.2. Identity Key Requirements

An implementation claiming conformance for IK generation MUST ensure:

1. Non-exportable: No interface SHALL allow extraction of the private key material, including backup, sync, or migration interfaces.

2. User-verification-gated: The platform MUST enforce local user verification before each IK signing operation, or before each operation within a bounded and explicitly documented platform policy window. The platform MUST NOT cache verification results indefinitely.

3. Enrollment-sensitive: If the device's biometric enrollment data changes (e.g., a new fingerprint is added), the platform SHOULD invalidate the Identity Key. This prevents an attacker who gains temporary physical access from adding their biometric and subsequently producing valid BIOMETRIC-level signatures.

4. The platform MUST expose sufficient information for the implementation to label the resulting signature with one of the defined assurance values.

5. Hardware attestation: On platforms that support hardware key attestation (e.g., Android Key Attestation, Apple Secure Enclave key attestation), implementations SHOULD obtain attestation evidence for the Identity Key at generation time and include it in Contact Objects (see Section 2.3).

This document does not require any specific hardware architecture. It intentionally abstracts over secure enclaves, secure elements, TPMs, TEEs, and comparable mechanisms. However, platforms that can provide hardware attestation evidence offer a stronger assurance level (HARDWARE_VERIFIED) that is independently verifiable by the receiving peer.

3.3. Transport Key Rotation

The Transport Key MAY be rotated to limit network-level correlation. Rotation comprises:

1. Generate a new Transport Key pair.

2. Sign a new Key Binding Object binding the IK to the new TK (Section 5).

3. Distribute the new KBO to connected peers.

4. Discard old TK private material.

Peers that receive a new KBO for a known IK MUST verify the KBO signature. If valid, the peer updates its stored TK for that contact.

3.3.1. Offline Rotation

When the Transport Key is rotated while a contact is offline, the contact will be unable to locate the key holder on the network using the previous Transport Key. Implementations MUST address this using one or more of the following mechanisms:

Stable addressing (RECOMMENDED): The addressing field in the Contact Object (Section 4, field 8) SHOULD contain addressing information that survives Transport Key rotation (e.g., a relay server address or discovery service endpoint). When a contact reconnects using the stable address, the key holder presents the current KBO containing the new Transport Key.

Mailbox deposit: If the implementation supports a mailbox or store-and-forward mechanism, the key holder SHOULD deposit the new KBO at their mailbox. Offline contacts retrieve the updated KBO upon reconnection.

Overlap period: The key holder MAY continue to accept connections on the old Transport Key for a limited time after rotation (RECOMMENDED: no more than 24 hours). Connections on the old Transport Key SHOULD be used only to deliver the new KBO and redirect the contact. After the overlap period, the old Transport Key MUST be discarded.

Implementations MUST NOT rotate the Transport Key without a mechanism for offline contacts to discover the new key.

4. Contact Object

4.1. Purpose

The Contact Object is exchanged during an in-person ceremony and anchors the long-term relationship.

4.2. Format

The Contact Object is a COSE_Sign1 structure:

```
ContactObject = COSE_Sign1(
  protected: { 1 (alg): -7 (ES256) },
  unprotected: {},
  payload: Contact_payload
)
```

Contact_payload is a CBOR map:

Key	Name	CBOR Type	Description
0	structure_type	uint	MUST be 0x02 (CONTACT_OBJECT)
1	version	uint	MUST be 1
2	ik_public	COSE_Key	Sender Identity Key (P-256)
3	tk_public	bstr	Sender Transport Key public bytes
4	tk_algorithm	int	Algorithm identifier for tk_public
5	display_name	tstr	UTF-8, max 64 bytes
6	timestamp	uint	Unix time in milliseconds
7	nonce	bstr	16 bytes, cryptographically random
8	addressing	bstr	Transport-specific, max 1024 bytes
9	assurance	uint	Assurance value for signing
10	attestation_evidence	bstr	OPTIONAL. Hardware attestation evidence (platform-specific). Present when assurance is 3 (HARDWARE_VERIFIED).

Table 2

The tk_public field MUST contain the raw public key bytes in the canonical encoding for the algorithm identified by tk_algorithm. See the transport key algorithm table (Section 11) for defined encodings.

The addressing field is opaque to this specification. It MUST carry only the minimum information needed for rendezvous or re-contact in the embedding application. Implementations SHOULD prefer application-scoped or short-lived addressing values where practical and SHOULD avoid embedding identifiers that are broader or more stable than operationally required.

4.3. Creation

The sender MUST:

1. Generate a 16-byte nonce (field 7) using a cryptographically secure random number generator (CSPRNG).

2. Set timestamp (field 6) to the current time.

3. Populate all other fields.

4. If the platform supports hardware key attestation, obtain attestation evidence for the Identity Key and include it in field 10 (attestation_evidence). Set assurance (field 9) to 3 (HARDWARE_VERIFIED). Otherwise, set assurance (field 9) to the value the platform will report for the signing event (1 for CREDENTIAL, 2 for BIOMETRIC).

5. Serialize Contact_payload as deterministic CBOR.

6. Sign using COSE_Sign1 with the Identity Key. This operation triggers a local user-verification event.

4.4. Verification

Upon receiving a Contact Object during the in-person ceremony, the receiver MUST perform the following checks. Checks are ordered so that inexpensive validations precede cryptographic signature verification:

1. Decode the COSE_Sign1 structure. If decoding fails, reject.

2. Extract the Contact_payload.

3. Verify that structure_type (field 0) is 0x02 (CONTACT_OBJECT). If not, reject.

4. Verify that version (field 1) is supported. If unsupported, reject.

5. Verify that timestamp (field 6) is within a 5-minute window of the receiver's current time: abs(now - timestamp) <= 300000 milliseconds. If outside the window, reject.

6. Verify that nonce (field 7) has not been seen in a previous Contact Object within the current 5-minute window. If duplicate, reject.

7. Verify the COSE_Sign1 signature using ik_public (field 2). If signature verification fails, reject.

8. If attestation_evidence (field 10) is present and assurance (field 9) is 3 (HARDWARE_VERIFIED), verify the attestation evidence against the platform vendor's certificate chain. If the attestation evidence is present but verification fails, reject. If the verifier cannot process the attestation format, it MUST treat the assurance value as 2 (BIOMETRIC) rather than 3 and MAY accept the Contact Object at the reduced assurance level.

9. If all checks pass and the exchange is bidirectional (the local device has also transmitted its own Contact Object to this peer), store the contact: ik_public, tk_public, tk_algorithm, display_name, addressing, assurance value, attestation evidence (if present), and the timestamp of the exchange.

A verifier MAY compute and compare a Relationship Fingerprint (Section 9) as an additional defense if the exchange path includes intermediaries.

4.5. Ceremony Requirement

The Contact Object exchange MUST be bidirectional: both parties MUST exchange and verify Contact Objects before either party stores the other as a verified contact. An implementation MUST NOT store a contact from a unidirectional exchange.

The proximity mechanism MUST satisfy the following requirements:

1. Both parties are physically co-present during the exchange.

2. The mechanism can transfer at least 512 bytes of binary data.

3. The mechanism provides a human-verifiable indication that the exchange is occurring with the intended party (e.g., visual confirmation, physical proximity constraint).

4. For mechanisms that do not provide visual confirmation of the peer (e.g., wireless discovery), the implementation MUST require explicit user confirmation of the peer's display_name before storing the contact.

The choice of proximity mechanism is an implementation decision.

4.6. Size Considerations

A Contact Object without attestation evidence typically serializes to 300-500 bytes; with attestation evidence (field 10) it may reach 1-2 KB. Implementations SHOULD ensure their proximity mechanism supports at least 512 bytes, or at least 4,096 bytes when attestation is included. Bandwidth-limited mechanisms (e.g., QR codes) MAY omit attestation evidence from the initial exchange and deliver it over the transport connection after rendezvous.

4.7. Clock Skew

The 5-minute timestamp window accommodates reasonable clock skew between devices. Implementations SHOULD use network time synchronization (NTP) when available but MUST NOT reject payloads solely due to minor clock differences within the window.

4.8. Replay Protection

The nonce field prevents replay of a captured Contact Object within its validity window. After the 5-minute window expires, the timestamp check prevents replay. Implementations MUST maintain a set of seen nonces for at least 5 minutes and SHOULD discard them after 10 minutes.

5. Key Binding Object

5.1. Purpose

The Key Binding Object proves continuity between the stored IK and the currently offered TK.

5.2. Format

The Key Binding Object is a COSE_Sign1 structure:

```
KeyBindingObject = COSE_Sign1(
  protected: { 1 (alg): -7 (ES256) },
  unprotected: {},
  payload: KBO_payload
)
```

KBO_payload is a CBOR map:

Key	Name	CBOR Type	Description
0	structure_type	uint	MUST be 0x01 (KEY_BINDING)
1	version	uint	MUST be 1
2	ik_public	COSE_Key	Identity Key (P-256)
3	tk_public	bstr	Transport Key public bytes
4	tk_algorithm	int	Algorithm identifier
5	timestamp	uint	Unix time in milliseconds
6	expiry	uint	Unix time in milliseconds
7	assurance	uint	Assurance value for signing

Table 3

The tk_public and tk_algorithm fields follow the same encoding rules as in the Contact Object (Section 4).

5.3. Creation

The signer MUST:

1. Populate all fields of KBO_payload, including assurance (field 7) based on the key’s configured authentication policy.

2. Set expiry (field 6) to no more than 30 days (2,592,000,000 milliseconds) after timestamp (field 5).

3. Serialize KBO_payload as deterministic CBOR.

4. Sign using COSE_Sign1 with the Identity Key. This operation triggers a local user-verification event.

5.4. Verification

The verifier MUST perform the following checks. Checks are ordered so that inexpensive validations precede cryptographic signature verification:

1. Decode the COSE_Sign1 structure. If decoding fails, reject.

2. Verify that structure_type (field 0) is 0x01 (KEY_BINDING). If not, reject.

3. Verify that version (field 1) is supported. If unsupported, reject.

4. Verify that the current time is >= timestamp (field 5) and < expiry (field 6). If outside the window, reject.

5. Verify that expiry (field 6) minus timestamp (field 5) does not exceed 30 days (2,592,000,000 milliseconds). If it does, reject.

6. Extract ik_public (field 2) from the payload.

7. Verify that ik_public matches the expected Identity Key for this contact (from a prior contact exchange, Section 4). If no match, reject.

8. Verify that tk_public (field 3) matches the Transport Key of the transport peer as observed during the transport handshake. If no match, reject.

9. Verify the COSE_Sign1 signature using ik_public. If signature verification fails, reject.

If ANY check fails, the verifier MUST reject the KBO and terminate the connection.

5.5. Renewal

Implementations MUST re-sign the KBO before its expiry. Renewal requires a fresh user-verification event, providing periodic evidence of continued platform-authorized use.

6. Session Credential

6.1. Purpose

The Session Credential allows efficient signing of application-defined units without requiring a fresh IK operation for each unit. Not all data types require this signing; embedding applications determine which content warrants Signed Message protection (see Section 7.1). The delegation pattern is analogous to TLS Delegated Credentials [RFC9345]: a long-term trust anchor (the IK) signs a short-lived operational credential (the Session Credential), which in turn authenticates individual operations (messages, file transfers).

```
Stored IK_public (from in-person ceremony)
  |
  v  verifies
Session Credential (COSE_Sign1 signed by IK)
  |  contains SSK_public
  v  verifies
Signed Message (COSE_Sign1 signed by SSK)
```

6.2. Format

The Session Credential is a COSE_Sign1 structure:

```
SessionCredential = COSE_Sign1(
  protected: { 1 (alg): -7 (ES256) },
  unprotected: {},
  payload: SC_payload
)
```

SC_payload is a CBOR map:

Key	Name	CBOR Type	Description
0	structure_type	uint	MUST be 0x03 (SESSION_CREDENTIAL)
1	version	uint	MUST be 1
2	ssk_public	COSE_Key	Ephemeral P-256 public key
3	ik_public	COSE_Key	Issuing Identity Key (reference)
4	peer_ik_hash	bstr	SHA-256 of peer’s IK_public
5	timestamp	uint	Unix time in milliseconds
6	expiry	uint	Unix time in milliseconds
7	assurance	uint	Assurance value

Table 4

6.3. Creation

The signer MUST:

1. Generate an ephemeral P-256 key pair (the Session Signing Key). The SSK is generated in software; it is not required to reside in the platform key store.

2. Compute peer_ik_hash as SHA-256 of the peer's IK_public encoded as an uncompressed P-256 point (65 bytes starting with 0x04). This encoding MUST match the encoding used in the Relationship Fingerprint computation (Section 9).

3. Set expiry to a value after timestamp. Implementations MUST set a finite expiry (unlimited Session Credentials are prohibited). A RECOMMENDED default is 1 hour. Applications with different risk profiles MAY choose shorter durations (e.g., 5 minutes for financial transactions) or longer durations (e.g., 8 hours for a workday session).

4. Set assurance (field 7) based on the key's configured authentication policy.

5. Serialize SC_payload as deterministic CBOR.

6. Sign using COSE_Sign1 with the Identity Key. This operation triggers a local user-verification event.

The SSK private key MUST be held in memory only and MUST NOT be persisted to disk. It MUST be discarded when the session expires or the connection closes. Implementations SHOULD store the SSK in a memory region that is not swappable to disk (e.g., mlock on Linux).

6.4. Verification

The verifier MUST:

1. Decode the COSE_Sign1 structure. If decoding fails, reject.

2. Verify that structure_type (field 0) is 0x03 (SESSION_CREDENTIAL). If not, reject.

3. Verify that version (field 1) is supported. If unsupported, reject.

4. Verify that the current time is between timestamp (field 5) and expiry (field 6). If outside the window, reject.

5. Verify that ik_public (field 3) matches the stored Identity Key for the peer. If no match, reject.

6. Verify that peer_ik_hash (field 4) matches SHA-256 of the verifier's own IK_public (confirms the credential is scoped to this relationship). If no match, reject.

7. Verify the COSE_Sign1 signature using the stored IK_public. If signature verification fails, reject.

The Session Credential is typically verified once per session (when first received) and the ssk_public is cached for subsequent message verifications.

6.5. Credential Lifecycle

* A Session Credential SHOULD be created at connection establishment, immediately after KBO exchange.

* The credential MUST have a finite expiry. The maximum duration is application-defined. RECOMMENDED default: 1 hour.

* When a credential expires, the sender MUST create a new one, which requires a fresh user-verification event.

* If the connection closes, the SSK private material MUST be discarded immediately.

* A peer MAY reject an expired Session Credential and request that the sender re-authenticate by issuing a Presence Challenge (Section 8).

6.6. Security Note

Use of the SSK is weaker than direct use of the IK because the SSK is a software key. See Section 10.14 for a detailed analysis of SSK compromise risks and mitigations. Applications SHOULD require direct IK-based Presence Responses for high-risk actions such as key changes or privileged approvals.

7. Signed Message

7.1. Applicability

Not all application data requires Signed Message wrapping. When peers have exchanged and verified Key Binding Objects (Section 5), the transport peer's identity is bound to the stored IK from the ceremony, and data in transit is authenticated by the KBO-verified connection. Signed Messages are useful when content must be verifiable independent of the transport session -- for example, stored messages, forwarded content, or operations requiring non-repudiation. Real-time streams (audio, video) over an authenticated direct connection typically do not require per-unit signatures; the KBO-verified transport and Presence Challenges provide sufficient assurance.

Embedding applications define which data types require Signed Messages and which rely solely on transport authentication.

When Signed Messages are stored for later verification, implementations MUST also persist the Session Credential that authorized the signing key. The full verification chain -- stored IK (from the ceremony), Session Credential (IK signed SSK), and Signed Message (SSK signed the content) -- is required to verify a message after the transport session has ended.

7.2. Format

Application-defined units authenticated by a Session Credential are wrapped in a COSE_Sign1 structure signed by the SSK. A unit MAY be a text message, a video keyframe, a file, a batch of events, or any other application-meaningful boundary. The signing granularity is determined by the embedding application's security requirements:

```
SignedMessage = COSE_Sign1(
  protected: { 1 (alg): -7 (ES256) },
  unprotected: {},
  payload: SM_payload
)
```

SM_payload is a CBOR map:

Key	Name	CBOR Type	Description
0	structure_type	uint	MUST be 0x04 (SIGNED_MESSAGE)
1	message_id	uint	See message ordering below
2	timestamp	uint	Unix time in milliseconds
3	content_type	uint	Application-defined (e.g., 1=text)
4	content	bstr/tstr	Message content

Table 5

The SSK signing operation does NOT require a user-verification event (it uses the software-held ephemeral key). This enables signing application units without user interaction.

7.3. Message Ordering and Replay Protection

The message_id field MUST be unique among all Signed Messages produced by the same sender within the scope of a single Session Credential. The sender MUST assign message_id values as a strictly increasing sequence starting from 1.

7.3.1. Reliable Transports

When the underlying transport guarantees in-order delivery (e.g., TCP, QUIC streams), the receiver MUST reject any Signed Message whose message_id is less than or equal to the highest message_id previously accepted from the same Session Credential.

7.3.2. Unreliable Transports

When the underlying transport does not guarantee ordering (e.g., UDP, DTLS, WebRTC data channels in unordered mode), messages MAY arrive out of order or be lost in transit. In this case, the receiver MUST maintain a sliding replay window of at least 64 entries and apply the following rules. Implementations SHOULD size the window to accommodate the expected maximum reordering delay of the transport (e.g., a larger window for high-throughput video streams than for infrequent text messages):

1. If message_id > highest_seen: accept, update highest_seen.

2. If message_id < highest_seen - window_size: reject (too old).

3. If message_id is within the window and has NOT been seen: accept, mark as seen.

4. If message_id is within the window and HAS been seen: reject (replay).

This approach follows the anti-replay mechanism defined in Section 4.5.1 of DTLS 1.3 [RFC9147].

7.4. Replay State

Verifiers MUST maintain replay state scoped to the validated Session Credential. At minimum, a verifier MUST remember the highest accepted message_id (and, for unreliable transports, the sliding window bitmap) for each active Session Credential until that credential expires or is otherwise discarded.

If replay state is lost, the verifier MUST treat subsequently received Signed Messages under the affected Session Credential as potentially replayed. In that case, the verifier MUST either reject those messages or require a new Session Credential, or a direct Identity-Key-based re-synchronization step, before accepting further messages in that session.

7.5. Verification

The full trust chain from a signed message back to the in-person ceremony:

To verify a Signed Message, the verifier MUST:

1. Obtain the sender’s Session Credential for this connection.

2. Verify the Session Credential (see Section 6).

3. Extract ssk_public (field 2) from the Session Credential.

4. Decode the Signed Message COSE_Sign1 structure. If decoding fails, reject.

5. Verify that structure_type (field 0) is 0x04 (SIGNED_MESSAGE). If not, reject.

6. Verify that timestamp (field 2) is >= the Session Credential’s timestamp and <= the Session Credential’s expiry (plus any locally configured grace period; RECOMMENDED: no more than 60 seconds past expiry). If outside the window, reject.

7. Apply the replay check for message_id (field 1) according to the transport type: strict monotonic for reliable transports, or sliding window for unreliable transports (see Section 7.3). If the check fails, reject.

8. Verify the COSE_Sign1 signature using ssk_public. If signature verification fails, reject.

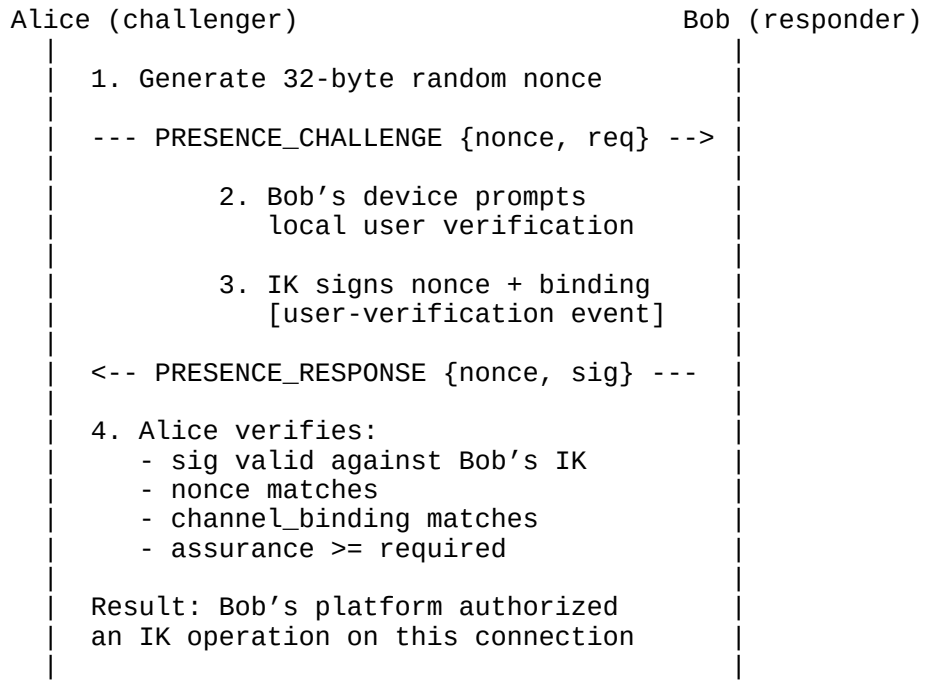
If any check fails, the verifier MUST reject the message.

8. Presence Challenge and Presence Response

8.1. Purpose

A Presence Challenge requests fresh evidence that the peer can perform an IK operation on the current connection.

After a connection is established and authenticated via KBO exchange, either peer MAY issue a Presence Challenge to request on-demand evidence that the remote device has recently produced a locally authorized signature.



8.2. Challenge Format

The Presence Challenge is an unsigned CBOR map:

Key	Name	CBOR Type	Description
0	structure_type	uint	MUST be 0x10 (PRESENCE_CHALLENGE)
1	challenge_nonce	bstr	32 bytes, cryptographically random
2	required_assurance	uint	Minimum required assurance value

Table 6

The challenger MUST generate challenge_nonce using a cryptographically secure random number generator.

The required_assurance field allows the challenger to demand a minimum assurance level. If set to 2 (BIOMETRIC), the responder MUST authenticate via biometric or equivalent; a CREDENTIAL-level response to a BIOMETRIC-required challenge MUST be rejected by the challenger. If set to 1 (CREDENTIAL), any assurance level is acceptable. If set to 3 (HARDWARE_VERIFIED), the response MUST include hardware attestation evidence (see Section 2.3).

8.3. Response Format

The Presence Response is a COSE_Sign1 structure:

```
PresenceResponse = COSE_Sign1(
  protected: { 1 (alg): -7 (ES256) },
  unprotected: {},
  payload: PR_payload
)
```

PR_payload is a CBOR map:

Key	Name	CBOR Type	Description
0	structure_type	uint	MUST be 0x11 (PRESENCE_RESPONSE)
1	challenge_nonce	bstr	Echoed from challenge
2	assurance	uint	Reported assurance value
3	channel_binding	bstr	SHA-256 of connection context
4	timestamp	uint	Unix time in milliseconds
5	attestation_evidence	bstr	OPTIONAL. Hardware attestation evidence. Present when assurance is 3 (HARDWARE_VERIFIED).

Table 7

The responder signs the response with their Identity Key. This signing operation triggers a local user-verification event.

The channel_binding field (key 3) MUST contain the SHA-256 hash of a connection-specific value agreed upon during the transport handshake. This prevents relay attacks where an attacker forwards a challenge to the victim's device and relays the response to a different connection.

The mandatory-to-implement channel binding computation is:

```
cb_input = sort_lexicographic(TK_local, TK_remote)
channel_binding = SHA-256(cb_input[0] || cb_input[1])
```

where TK_local and TK_remote are the raw Transport Key public bytes (as carried in tk_public fields) and sort_lexicographic orders them by byte-wise lexicographic comparison.

Transport profiles MAY define alternative channel binding computations (e.g., a transport-provided session identifier or TLS Exporter value). Both peers MUST agree on the computation method before issuing or responding to challenges. When no transport profile is in effect, implementations MUST use the mandatory-to-implement computation above.

8.4. Verification

The challenger MUST:

1. Decode the COSE_Sign1 structure. If decoding fails, reject.

2. Verify that structure_type (field 0) is 0x11 (PRESENCE_RESPONSE). If not, reject.

3. Verify that challenge_nonce (field 1) matches the nonce sent. If no match, reject.

4. Verify that timestamp (field 4) is within 60 seconds of the challenge issuance time. If outside the window, reject.

5. Verify that channel_binding (field 3) matches the expected SHA-256 hash of the current connection context. If no match, reject.

6. Verify that assurance (field 2) is >= required_assurance from the challenge. If not, reject.

7. If assurance is 3 (HARDWARE_VERIFIED), verify the attestation_evidence (field 5) against the platform vendor's certificate chain. If attestation evidence is present but verification fails, reject. If the verifier cannot process the attestation format, it MUST treat the assurance value as 2 (BIOMETRIC) rather than 3.

8. Verify the COSE_Sign1 signature using the stored Identity Key for the peer. If signature verification fails, reject.

8.5. Timing

* Implementations SHOULD NOT issue challenges more frequently than once per 300 seconds (5 minutes).

* The responder MUST produce a response within 60 seconds.

* If no valid response is received within 60 seconds, the challenger SHOULD inform the user.

* Failure to respond MUST NOT automatically terminate the connection.

A successful Presence Response demonstrates only that the peer's platform authorized an IK operation on the current connection. See Section 1.6 for scope limitations.

9. Relationship Fingerprint

9.1. Overview

The Relationship Fingerprint enables two peers to verify out-of-band that they hold each other's correct Identity Keys, detecting man-in-the-middle key substitution.

Fingerprint verification is OPTIONAL. Its necessity depends on the contact exchange mechanism:

* When Contact Objects are exchanged via a physical proximity mechanism where both parties can visually confirm the exchange, man-in-the-middle substitution is not feasible and fingerprint verification MAY be skipped.

* When the exchange mechanism involves an intermediary (e.g., a code displayed on one device and photographed by another, or a wireless exchange without visual confirmation of the peer), fingerprint verification SHOULD be performed even when both parties are physically co-present.

This specification does not support Contact Object exchange over channels that do not guarantee physical proximity (e.g., server relay, email, or messaging application). Such usage violates the Ceremony Requirement (Section 4) and falls outside this document's trust model; any security claims in such deployments derive from the relay channel's own authentication, not from this protocol.

9.2. Computation

Given two Identity Key public keys, IK_A and IK_B, encoded as uncompressed P-256 points (65 bytes each, starting with 0x04):

```
sorted = sort_lexicographic(IK_A, IK_B)
prefix = "H2H-RelationshipFingerprint-v1"
digest = SHA-256(prefix || sorted[0] || sorted[1])
```

SHA-256 is as defined in [FIPS180-4].

This document defines only the digest computation. Any human-readable rendering is application-defined.

Implementations MAY encode the digest for human comparison (e.g., as grouped numeric blocks, hexadecimal, base32, or an emoji sequence mapped from digest bits). The security of any such encoding depends on the number of digest bits it exposes. Renderings that expose fewer than 30 bits of the digest offer only weak resistance against man-in-the-middle substitution and SHOULD NOT be used for security-critical comparisons. See Section 10.8 for analysis.

9.3. Properties

* The fingerprint is deterministic: the same pair of Identity Keys always produces the same fingerprint.

* The fingerprint is symmetric: it is the same regardless of which party computes it (due to the lexicographic sort).

* The fingerprint changes if and only if either party's Identity Key changes.

9.4. Key Change Notification

Implementations SHOULD alert the user when a stored contact's Identity Key changes (and therefore the fingerprint changes). This may indicate a new device (benign) or a man-in-the-middle attack.

10. Security Considerations

10.1. Limitations

This protocol does not provide:

* proof-of-personhood
* universal identity proof
* protection against coercion
* protection against a compromised platform key store
* protection against a compromised application falsely describing the platform’s assurance event unless the deployment adds attestation
* unlinkability across contacts that receive the same IK
* revocation or recovery

10.2. Platform Trust, Compromised Applications, and Device State

All meaningful security properties of this protocol depend on platform behavior when using the Identity Key. If the platform allows the key operation without the expected local verification event, the protocol cannot detect that failure by itself.

A compromised or malicious application can alter what is being signed, misrepresent why a signing operation is requested, or present application-specific content to the user in a misleading way. Unless the deployment provides trusted display, platform attestation, or other out-of-band confirmation of signing intent, a verifier MUST NOT assume that a valid protocol object implies that the signer saw or understood the application semantics associated with that object. Deployments SHOULD distinguish between cryptographic authorization of a protocol object and user consent to higher-level application meaning.

If an attacker obtains control of an unlocked device, or if the device holder is coerced into satisfying local verification, the protocol provides no defense against this scenario. Verifiers MUST treat successful verification as evidence only that the platform authorized key use under its current policy (see Section 1.6).

10.3. Contact Object Capture

A captured Contact Object (e.g., photographed visual encoding or intercepted wireless transmission) gives the attacker the victim's public Identity Key and addressing information but the attacker cannot:

* Sign as the victim (lacks the device-bound private key).

* Replay the payload after 5 minutes (timestamp check).

* Replay within 5 minutes if the nonce was already seen.

The attacker CAN attempt to connect to the victim's transport address but cannot complete authentication (they lack a KBO for an Identity Key the victim trusts).

10.4. Key Binding Object Expiry

A 30-day KBO expiry limits the window during which a compromised Transport Key can be used. Frequent TK rotation (Section 3.3) reduces this window further.

10.5. Assurance Semantics, Policy, and Application Compromise

The assurance values defined by this document are signer-reported, platform-mediated claims. They are intentionally coarse and do not capture the full variety of operating-system behavior, biometric modes, passcode fallback rules, trusted-path properties, or reuse windows.

A compromised application could request CREDENTIAL-level verification from the platform and then set the assurance field to BIOMETRIC in the payload before signing. The HARDWARE_VERIFIED (3) assurance level mitigates this attack: when hardware attestation evidence is present, the verifier can confirm the key's properties directly against the hardware vendor's certificate chain. A compromised application cannot forge hardware attestation evidence.

During the Contact Exchange ceremony, a compromised application could substitute attacker-controlled keys while displaying correct-looking UI. Hardware key attestation proves the Identity Key is hardware-bound but does not prove the application is legitimate. Platform app integrity services (Appendix B.2) can mitigate this additional threat. Without either form of attestation, peers SHOULD use out-of-band Relationship Fingerprint comparison (Section 9) as a defense against key substitution.

Deployments that require stronger semantics SHOULD require HARDWARE_VERIFIED assurance where platform support is available, and SHOULD define local policy for acceptable platforms, maximum verification age, session credential lifetimes, and when direct IK use is required instead of delegated signing.

10.6. Relay Attack on Presence Response

Without channel binding, an attacker positioned between two peers could relay a Presence Challenge from Peer A to the victim's device, collect the signed response, and forward it to Peer A. The channel_binding field in the Presence Response (Section 8) prevents this by binding the response to the specific connection context.

10.7. Type Confusion

Without the structure_type discriminator (field 0), an attacker could potentially substitute a signed Contact Object for a Key Binding Object or vice versa, since both are COSE_Sign1 structures signed by the same Identity Key. The structure_type field prevents this: verifiers MUST check that the structure_type matches the expected value before processing any other fields.

10.8. Relationship Fingerprint Limitations

The fingerprint is optional and its security depends on the number of digest bits exposed by the encoding chosen by the application. A rendering that exposes N bits of the digest offers approximately N bits of second-preimage resistance against an attacker who generates candidate Identity Key pairs and searches for one whose fingerprint matches the target.

Indicative strengths:

* A 20-bit rendering (e.g., 6 decimal digits) requires on the order of 10^6 trial key pairs. Trivially brute-forceable on commodity hardware.

* A 26-bit rendering (e.g., 8 decimal digits) requires on the order of 10^8 trial key pairs. Feasible on modern GPU hardware within hours.

* A 60-bit rendering is practically infeasible to brute-force for realistic attackers.

For applications requiring strong assurance, implementations SHOULD support machine-readable comparison (for example, a QR code exchanged over a second authenticated channel) that conveys the full 256-bit digest without human error.

10.9. Privacy Considerations

10.9.1. Identity Key as a Stable Identifier

The Contact Object transmits the Identity Key public component in cleartext. Any party that observes or captures a Contact Object obtains a persistent, unique identifier for the user.

Mitigations:

* Contact Objects are exchanged only during deliberate, in-person ceremonies, limiting the exposure surface.

* The 5-minute validity window limits the useful lifetime of a captured object.

* The Transport Key (used for networking) is a separate key that does not reveal the Identity Key to network observers.

10.9.2. Relationship Correlation

If the same Identity Key is reused across multiple relationships or applications, receivers that exchange identifiers or protocol transcripts can determine that the same long-term key participated in multiple relationships. This may reveal social-graph structure or cross-context linkage.

Deployments that treat such correlation as sensitive SHOULD consider future profiles that use pairwise-derived relationship identifiers or per-relationship keys. Those mechanisms are not defined by this document.

10.9.3. Metadata Exposure

The Contact Object can also reveal metadata through fields such as display_name, addressing, timestamps, and transport key continuity. Even when application payloads are encrypted elsewhere, these fields may reveal account aliases, transport reachability hints, device migration timing, or relationship existence.

Applications using this protocol SHOULD minimize metadata disclosure, avoid unnecessary stable identifiers, and avoid embedding data in the addressing field that is broader than required for rendezvous.

10.9.4. Fingerprint Handling

Relationship Fingerprints are intended for optional out-of-band comparison. Applications SHOULD avoid displaying full fingerprints where they could be captured by screenshots, shoulder surfing, or logging.

See [RFC6973] for a general discussion of privacy considerations in Internet protocols.

10.10. Denial of Service via Presence Challenges

An authenticated peer could issue excessive Presence Challenges, forcing repeated user-verification prompts. The 300-second rate limit (Section 8) mitigates this but does not eliminate it. Implementations SHOULD allow users to disable Presence Challenge responses from specific contacts.

10.11. Post-Quantum Considerations

P-256 ECDSA is vulnerable to quantum computers implementing Shor's algorithm. When post-quantum signature algorithms are available in platform key stores, the protocol SHOULD migrate. The version field in all structures enables this transition.

10.12. Forward Secrecy of Authentication

This protocol does not provide forward secrecy of authentication. If the Identity Key private key is ever extracted from the platform key store (despite hardware protections), all Key Binding Objects, Session Credentials, and Presence Responses previously signed by that key can be verified by the attacker. Past signed objects remain verifiable indefinitely.

Deployments that require forward secrecy of authentication evidence SHOULD layer an ephemeral key agreement protocol (e.g., TLS 1.3 [RFC8446] or a Diffie-Hellman-based transport) beneath this protocol. This document's objects provide authentication, not confidentiality or forward secrecy.

10.13. Session Credential Expiry and In-Flight Messages

A Session Credential has a finite expiry. Messages signed by the SSK before the credential expires but received after expiry present a time-of-check/time-of-use consideration. The Signed Message verification procedure (Section 7) requires that the message timestamp falls within the Session Credential's validity window, with an optional grace period for in-flight messages. RECOMMENDED grace

period: no more than 60 seconds past expiry. After the grace period, verifiers MUST reject the message and require a new Session Credential.

10.14. Session Signing Key Compromise

The SSK is a software key held in memory. An attacker who extracts it can forge Signed Messages for the remaining validity period of the Session Credential. A read-only memory disclosure (e.g., a side-channel attack) that reveals the SSK but not the TK private key would allow message forgery without transport impersonation.

The Session Credential intentionally omits a channel_binding field because Session Credentials must remain valid across delivery paths, including mailbox deposit. As a consequence, a stolen SSK and SC can be used from any network position for the remaining credential lifetime.

Mitigations:

* The Session Credential MUST have a finite expiry, limiting the forgery window.

* The SSK MUST be discarded when the session ends.

* The peer_ik_hash field scopes the credential to a single relationship.

* Implementations SHOULD store the SSK in a non-swappable memory region.

* For operations requiring the strongest assurance, implementations SHOULD sign directly with the Identity Key rather than delegating to the SSK.

10.15. Device Loss and Recovery

When a device is lost or destroyed, the Identity Key is permanently unrecoverable (non-exportable from the platform key store). All contacts verified against that Identity Key become orphaned.

Revocation and recovery are out of scope for this document, but that omission has concrete deployment consequences. In such cases, deployments will typically require an out-of-band re-verification ceremony, explicit peer approval of a new Identity Key, or an application-defined recovery mechanism.

Similarly, if an Identity Key is believed to be compromised, this document does not define a protocol-native revocation object or distribution method. Applications MUST therefore define local policy for suspending trust in a stored key and for handling replacement or termination of the affected relationship.

10.16. Terminology Caution

The "h2h" identifier in this document’s name refers to the intended use case -- communication between devices held by two specific humans who have met in person -- and is not a claim that the protocol cryptographically verifies biological humanness, legal personhood, or proof-of-personhood in any form.

Implementations and profiles built on this document SHOULD avoid using terms such as "proves human" or "non-repudiation" without additional legal and deployment analysis. The protocol produces signed objects that attribute actions to a key; it does not prove those actions were intentional or legally binding.

11. IANA Considerations

This document defines protocol constants for structure types, assurance values, and transport key algorithms. No IANA registries are requested. These values are defined in the body of this document and in the CDDL schema (Appendix A). They are summarized here for convenience.

11.1. Structure Type Values

Value	Name
0x01	KEY_BINDING
0x02	CONTACT_OBJECT
0x03	SESSION_CREDENTIAL
0x04	SIGNED_MESSAGE
0x10	PRESENCE_CHALLENGE
0x11	PRESENCE_RESPONSE

Table 8

11.2. Assurance Values

Value	Name
1	CREDENTIAL
2	BIOMETRIC
3	HARDWARE_VERIFIED

Table 9

11.3. Transport Key Algorithms

COSE alg	Name	tk_public Encoding
-8	EdDSA	32 bytes, Ed25519 public key (RFC 8032 S5.1.5)
-7	ES256	65 bytes, uncompressed P-256 point (0x04 || x || y)

Table 10

Implementations MUST support at least one of the above transport key algorithms.

A future Standards Track document may request the creation of IANA registries for these value spaces if broader interoperability requires coordinated allocation.

12. References

12.1. Normative References

[FIPS180-4]
National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, August 2015.

[FIPS186-5]
National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS PUB 186-5, February 2023.

[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>.

[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>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/rfc/rfc8610>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/rfc/rfc9053>.

12.2. Informative References

[AppAttest] Apple, "DeviceCheck | Apple Developer Documentation", 2024, <https://developer.apple.com/documentation/devicecheck>.

[FIDO2] FIDO Alliance, "FIDO2: Web Authentication (WebAuthn)", 2024, <https://fidoalliance.org/fido2/>.

[PlayIntegrity] Google, "Play Integrity API Overview", 2024, <https://developer.android.com/google/play/integrity/overview>.

[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>.

[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>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/rfc/rfc9147>.

[RFC9345] Barnes, R., Iyengar, S., Sullivan, N., and E. Rescorla, "Delegated Credentials for TLS and DTLS", RFC 9345, DOI 10.17487/RFC9345, July 2023, <https://www.rfc-editor.org/rfc/rfc9345>.

[RFC9420] Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., and K. Cohn-Gordon, "The Messaging Layer Security (MLS) Protocol", RFC 9420, DOI 10.17487/RFC9420, July 2023, <https://www.rfc-editor.org/rfc/rfc9420>.

[Signal] Marlinspike, M. and T. Perrin, "The X3DH Key Agreement Protocol", 2016, <https://signal.org/docs/specifications/x3dh/>.

[WebAuthn] Balfanz, D., Czeskis, A., Hodges, J., Jones, J.C., Jones, M.B., Kumar, A., Liao, A., Lindemann, R., and E. Lundberg, "Web Authentication: An API for accessing Public Key Credentials Level 2", W3C Recommendation, April 2021, <https://www.w3.org/TR/webauthn-2/>.

Appendix A. CDDL Schema

The following CDDL [RFC8610] schema formally defines all protocol structures. This schema is normative.

```
; ============================================================
; P2P Presence Verification CDDL Schema
; Reference: draft-rodriguez-h2h-presence-attestation-00
; ============================================================

; -- Common types --

assurance_value = 1 / 2 / 3   ; 1=CREDENTIAL, 2=BIOMETRIC,
```

```
                                ; 3=HARDWARE_VERIFIED

   ; P-256 public key as COSE_Key (kty=EC2, crv=P-256)
   P256_COSE_Key = {
     1  => 2,                    ; kty: EC2
     -1 => 1,                   ; crv: P-256
     -2 => bstr .size 32,       ; x coordinate
     -3 => bstr .size 32,       ; y coordinate
   }

   ; ============================================================
   ; COSE_Sign1 wrappers
   ; Each signed structure is a COSE_Sign1 with ES256 algorithm.
   ; The payload is a CBOR-encoded map defined below.
   ; ============================================================

   KeyBindingObject = #6.18([
     bstr .cbor {1 => -7},            ; protected: alg ES256
     {},                               ; unprotected
     bstr .cbor KBO_payload,          ; payload
     bstr,                             ; signature
   ])

   ContactObject = #6.18([
     bstr .cbor {1 => -7},
     {},
     bstr .cbor Contact_payload,
     bstr,
   ])

   SessionCredential = #6.18([
     bstr .cbor {1 => -7},
     {},
     bstr .cbor SC_payload,
     bstr,
   ])

   SignedMessage = #6.18([
     bstr .cbor {1 => -7},
     {},
     bstr .cbor SM_payload,
     bstr,
   ])

   PresenceResponse = #6.18([
     bstr .cbor {1 => -7},
     {},
     bstr .cbor PR_payload,
```

```
  bstr,
])

; ============================================================
; Payload definitions (no additional keys permitted)
; ============================================================

; -- Key Binding Object --

KBO_payload = {
  0 => 1,                    ; structure_type: KEY_BINDING (fixed)
  1 => 1,                    ; version (fixed for v1)
  2 => P256_COSE_Key,        ; ik_public
  3 => bstr,                 ; tk_public: raw key bytes
  4 => int,                  ; tk_algorithm: COSE alg id
  5 => uint,                 ; timestamp: Unix ms
  6 => uint,                 ; expiry: Unix ms, max +30d
  7 => assurance_value,      ; assurance
}

; -- Contact Object --

Contact_payload = {
  0 => 2,                    ; structure_type: CONTACT_OBJECT (fixed)
  1 => 1,                    ; version (fixed for v1)
  2 => P256_COSE_Key,        ; ik_public
  3 => bstr,                 ; tk_public: raw key bytes
  4 => int,                  ; tk_algorithm: COSE alg id
  5 => tstr .size (0..64),   ; display_name: UTF-8, max 64 bytes
  6 => uint,                 ; timestamp: Unix ms
  7 => bstr .size 16,        ; nonce: 16 random bytes
  8 => bstr .size (0..1024), ; addressing: transport-specific
  9 => assurance_value,      ; assurance
  ? 10 => bstr,              ; attestation_evidence: OPTIONAL
}

; -- Session Credential --

SC_payload = {
  0 => 3,                    ; structure_type: SESSION_CREDENTIAL
  1 => 1,                    ; version (fixed for v1)
  2 => P256_COSE_Key,        ; ssk_public: ephemeral P-256 key
  3 => P256_COSE_Key,        ; ik_public: issuing Identity Key
  4 => bstr .size 32,        ; peer_ik_hash: SHA-256 of peer IK
  5 => uint,                 ; timestamp: Unix ms
  6 => uint,                 ; expiry: Unix ms
  7 => assurance_value,      ; assurance
}
```

```
; -- Signed Message --

SM_payload = {
  0 => 4,                    ; structure_type: SIGNED_MESSAGE (fixed)
  1 => uint,                 ; message_id: >= 1, strictly increasing
  2 => uint,                 ; timestamp: Unix ms
  3 => uint,                 ; content_type: application-defined
  4 => bstr / tstr,          ; content
}

; -- Presence Challenge (not signed, plain CBOR) --

PresenceChallenge = {
  0 => 16,                   ; structure_type: PRESENCE_CHALLENGE
  1 => bstr .size 32,        ; challenge_nonce: 32 random bytes
  2 => assurance_value,      ; required_assurance
}

; -- Presence Response --

PR_payload = {
  0 => 17,                   ; structure_type: PRESENCE_RESPONSE
  1 => bstr .size 32,        ; challenge_nonce: echoed
  2 => assurance_value,      ; assurance
  3 => bstr .size 32,        ; channel_binding: SHA-256
  4 => uint,                 ; timestamp: Unix ms
  ? 5 => bstr,               ; attestation_evidence: OPTIONAL
}
```

Appendix B. Implementation Considerations

The following implementation guidance is informative and does not alter the protocol requirements in this document.

B.1. Platform Key Store Mapping

The abstract "platform key store" requirement commonly maps to mechanisms such as Secure Enclave, StrongBox or TEE-backed Android keys, TPM-backed Windows keys, or TPM- and PKCS#11-based Linux deployments. Implementations MUST verify that their chosen mechanism meets the Identity Key requirements in Section 3.2 regardless of the platform name used in product documentation.

B.2. App Integrity Attestation

Hardware key attestation proves the Identity Key resides in genuine hardware but does not prove the application is legitimate. Platform app integrity services -- such as Android Play Integrity API [PlayIntegrity] and Apple App Attest [AppAttest] -- can verify the application binary is unmodified. These services require online validation against vendor APIs. Implementations that wish to include app integrity attestation SHOULD obtain a token at ceremony time, include it alongside the Contact Object (not in the signed payload), and validate it against the vendor's API. Deployments handling high-value authorizations SHOULD use both hardware key attestation and app integrity attestation where available.

B.3. Payload Size Limits

Implementations MUST enforce maximum payload sizes to prevent resource-exhaustion attacks. A deployment SHOULD set explicit limits for each structure type before CBOR decoding or signature verification. If a profile does not define stricter limits, a reasonable default is 4,096 bytes for Contact Objects, Key Binding Objects, and Session Credentials; 262,144 bytes for Signed Messages; and 1,024 bytes for Presence Challenges and Presence Responses.

Acknowledgements

This work builds on COSE [RFC9052], CBOR [RFC8949], WebAuthn [WebAuthn], the FIDO2 framework [FIDO2], delegated-credential patterns [RFC9345], the Signal Protocol [Signal], and prior secure messaging designs.

Author's Address

Gamil Rodriguez
Rolab
Email: gamil@rolabtech.com