OMATrust Reputation Specification

Extension to the OMATrust architecture focused on reputation

1. Executive Summary

The OMATrust Reputation Specification defines the trust and reputation layer for applications registered within the OMATrust ecosystem. While the OMATrust Identity Specification provides decentralized identity, DID ownership resolution, data integrity verification, and application registration, this document describes how reputation signals are created, validated, indexed, and consumed.

The goal of the OMATrust reputation system is to enable transparent, tamper-resistant trust signals about applications, services, organizations, and users across both Web3 and Web2 environments. The system supports multiple categories of attestations, including endorsements, certifications, linked identifiers, data URL confirmations, and user reviews.

A key design principle is that reputation must be modular and verifiable. All reputation objects are expressed as structured attestations using standardized schemas. Each schema supports one or more verification mechanisms called ‘Proofs”, which are defined in the OMATrust Proof Specification.

This specification introduces a unified model for provable and unprovable user reviews. For applications capable of producing a Proof (such as blockchain transactions or x402-compliant endpoints), reviews may be “proven” and therefore subsidy-eligible within the ecosystem. Reviews lacking such proof remain valid but are treated as lower-confidence signals and may not qualify for incentives.

In contrast to most decentralized reputation systems, which focus on evaluating users or client identities, OMATrust is explicitly concerned with the reputation of services. This includes applications, APIs, organizations, assessors, and other server-side actors that form the machine-driven internet. The system does not attempt to score, rank, or track end-users. Instead, it establishes verifiable trust signals about the services that users and other machines depend on. This distinction is fundamental to the architecture: OMATrust is a service reputation system, not a client reputation system.

2. Scope

2.1 In-Scope

This document defines the schemas, verification rules, reputation flows, and client requirements for producing and consuming service-level reputation within OMATrust. It is intended for implementers building applications, services, indexers, and validators in the broader OMATrust ecosystem. The scope includes normative definitions for attestations and the mechanisms required for clients to verify and index reputation signals associated with decentralized services.

2.2 Out-of-Scope

This specification does not provide any scoring, weighting, ranking, or recommendation logic for services. OMATrust's role is limited to defining the attestations, proof formats, and verification rules needed to publish service-level reputation data. How clients interpret, aggregate, score, or visualize that data is entirely out of scope.

Approved or standardized verification processes for trusted attesters, including procedural requirements, audits, certification, or governance, is also not in scope. Such processes may be defined by future companion specifications or OMA3 certification programs.

3. References

OMATrust Whitepaper: https://github.com/oma3dao/omatrust-docs/blob/main/whitepaper.md
OMATrust Proof Specification:
OMA3 Github Repository:
https://github.com/oma3dao/rep-attestation-tools-evm-solidity/tree/main/schemas-json
DID Specification: https://www.w3.org/TR/did-core
DID Spec Registries: https://www.w3.org/TR/did-spec-registries/

4. Definitions

4.1 Abbreviations

In the present document, the following abbreviations apply:

DID	Decentralized Identifier
DNS	Domain Name System
HTTP	Hypertext Transfer Protocol

HTTPS	Hypertext Transfer Protocol Secure
ID	Identifier
IdP	Identity Provider
URL	Uniform Resource Locator
VC	(W3C) Verifiable Credential

Relevant External Specification Terminology Sections

DID Spec	https://www.w3.org/TR/did-core/#terminology
VC Spec	https://www.w3.org/TR/vc-data-model-2.0/#terminology

The following applies to the specification:

- “string” means a UTF-8 string
- “URL” means a string in a specific format
- “JSON” means a string in JSON format
- [] is meant to signify an array of whatever is inside the brackets.

4.2 Definitions

The following definitions are used within the present document.

Term	Definition
Application	A software service that can come in many formats, including a self-contained software application on a Device, an API endpoint, or a smart contract.
Client	Software that queries OMATrust to obtain information on an Application.
Verifier	Same as Client, but more compliant with W3C DID specs.
Decentralized Identifier	An identifier that adheres to the W3C DID standard.
Attester	An entity that issues credentials as attestations in the Reputation Service.
Issuer	Same as Attester, but more compliant with W3C DID specs.
Attestation	A structured, signed statement conforming to an OMA3/EAS schema, used to record claims about identity, usage, or trust.
Proof	Data, usually in the form of a JSON object, that proves a particular relationship (OMA3 Proof Specification)
PoP	Proof of Possession

Term	Definition
Encoded Value Transaction	A standardized JSON object embedded inside a User Review attestation that contains a Proof
Verified Review	A user review attestation that includes at least one valid service proof.
Unverified Review	A user review attestation without a service proof, or with a service proof that fails verification.
Trust Level	The classification of a user review based on the strength of its associated proof strength (e.g., High, Medium, Low, Invalid).
x402	A bidirectional, payment-linked protocol used to exchange signed messages between client and server, capable of generating mutually verifiable usage proofs.
Session Identifier	An opaque, non-PII identifier used to correlate client and server messages within x402 or service-proof contexts.
Opaque Identifier	A string with no embedded meaning, used to correlate events without exposing private or internal service information.
Service Signing Key	A public key authorized to sign service proofs for a specific service ID or DID.
Signing Algorithm	The cryptographic algorithm used to produce signatures (e.g., eip712, ed25519, secp256k1).
Signer	The public key or address that produced a signature included in a service proof.
Authorization Set	The set of public keys that a service has published or attested as authorized to sign service proofs.
Proof Verification	The process by which a client or indexer validates the signature, schema compliance, and authority of a service proof.
Review Payload	The structured JSON object inside the User Review attestation that includes review text, rating, and service proofs.
Proof Set	The array of service-proof objects included in a user review payload.
Invalid Proof	A service proof that fails signature verification, schema validation, authorization checks, or timestamp constraints.
Indexer	A node or system that aggregates and indexes attestations (such as EAS or Shinzo-based indexers) for efficient querying and verification.

In addition to the above definitions all OMA3 specifications use requirements language as described in the OMA3 working group process.

5. Core Specification

5.1 Attestation Model

This specification defines a unified model for creating, transporting, and verifying structured attestations used within OMATrust. An attestation is a cryptographically verifiable statement asserting some fact about a subject, produced by an issuer and optionally supported by contextual proofs of service usage or identity.

Clients may derive trust from Reputation attestations in two ways:

1. Proof-based (trustless) validation: Clients independently verify Proof Objects contained in an attestation.
2. Trusted-attester validation: Clients accept attestations based on the authority of the Attester under local trust policy (e.g., allowlists maintained by the client or by a trusted intermediary such as OMA3).

This specification focuses on proof-based validation and the attestation-layer semantics above Proof Objects (OMA3 Proof Specification). While trusted-attester validation is a valid trust model, standardizing approved trusted-attester verification processes is out of scope for this version.

Attestations in OMATrust share the following foundational properties:

- **Canonical JSON representation** for maximum interoperability
- **DID-based identifiers** for subjects, issuers, and service providers
- **Extensible proof structures** allowing multiple trust mechanisms
  Chain-agnostic semantics, enabling storage on EAS, BAS, or other systems
- **Explicit versioning**, allowing schema evolution across time
  Declarative validation rules, enforced through JSON Schema

The attestation model is designed to be transport-independent: the same JSON payload can be stored on a blockchain, in a centralized system, or within a federated attestation indexer. OMATrust distinguishes between two layers:

1. **Semantic Layer (this specification)**
    - Defines the schema, rules, verification logic
      Chain-agnostic
    - Deterministic semantics for all platforms
2. **Transport Layer (binding specifications)**

    - Defines how the attestation is stored, referenced, or queried
    - EAS/BAS/Web2/Hub bindings publish their own mapping rules

This separation ensures that attestations remain interoperable regardless of the infrastructure selected by the service operator, user, or verifier.

5.2 JSON Schema Structure

All attestation types defined by OMATrust use **JSON Schema Draft 2020-12** as the canonical method for describing structure, constraints, field semantics, and validation rules. Definitions for the schemas defined in this document can be found in OMA3 Github Repository.

Each schema MUST include:

- **$schema** – URI of JSON Schema draft
- **$id** – unique schema identifier URL
  title – human-readable name
- **description** – high-level summary
- **type** – object
- **required** – list of mandatory fields
- **properties** – declarative field definitions
- **OMA3-specific extensions**, including:
    - **`x-oma3-subtype`** (e.g., timestamp, rating, key, DID)
    - **`x-oma3-default`** (e.g., auto-timestamp)
    - **`x-oma3-skip-reason`** (e.g., metadata or EAS-handled fields)

Schemas SHOULD provide:

- precise description text for UI rendering
- explicit enums where appropriate
- clearly defined reference paths for nested objects
- consistent naming conventions for DID-related fields

Implementers MUST validate the JSON object against the schema before issuing or accepting an attestation.

The JSON Schema definition acts as the canonical source of truth for:

- field semantics
- validation requirements
- data types
- structural layout

All derived representations (EIP-712 typed data, on-chain encoding, or storage formats) must align with the schema.

5.3 Proofs

Reputation attestations MAY include Proof Objects to enable client-side, trustless validation. Proof Objects are not attestations; they are supporting evidence objects.

This specification does not redefine Proofs, which are defined in the OMATrust Proof Specification. This specification defines only how proofs are used within reputation attestations.

5.3.1 Proof Wrappers

When a reputation attestation includes Proofs, each entry in the **`proofs`** array MUST be a Proof wrapper. Clients MUST verify each wrapper and its native Proof Object according to the declared **`proofType`**.

5.3.2 Attestation-Layer Proof Constraints

For each attestation schema, this specification defines attestation-layer constraints on Proofs, including:

1. Which proof types are permitted or required for that schema.
2. Which **`proofPurpose`** values are permitted or required for that schema.
3. How verified Proofs must bind to the attestation context, i.e., which attestation fields must match which Proof inputs or outputs.

These constraints are specified in the relevant schema sections (e.g., Linked Identifiers in §6.1 and Reputation Attestations in §7). If a Proof is valid but violates any attestation-layer constraint in this document, the attestation MUST be rejected for proof-based trust.

5.3.3 Trust Model Determination

Clients determine how to evaluate an attestation based on proof presence:

- If **`proofs`** is present and non-empty, clients MUST evaluate the attestation using the proof-based (trustless) rules defined in the corresponding schema section.
- If **`proofs`** is absent or empty, clients MAY evaluate the attestation using trusted-attester validation under local trust policy, as described in §5.1.

No field inside an attestation may self-declare trusted status. Trusted acceptance depends solely on local trust policy and attester allowlists.

5.3.4 Multiple Proofs

An attestation MAY include multiple Proofs. Proofs may be redundant, complementary, or provided for different verification pathways. It is possible for one Proof to verify successfully while another fails.

This specification does not define a single global evaluation rule for multiple Proofs. Clients MUST apply the attestation-specific proof requirements in the relevant schema sections, and MAY use local policy to determine whether a failed Proof invalidates the attestation or is ignored.

6. Support Attestations

Support Attestations define the foundational relationships that allow OMATrust clients to reason reliably about who controls which identifiers, which cryptographic keys are authorized to speak for a service, and how identities across different systems are linked. The schemas in this section provide **concrete, transport-ready attestations** that bind identifiers, keys, and accounts in a verifiable way. These attestations establish a durable identity layer upon which all application-level attestations depend. Downstream schemas (e.g., User Reviews, Security Assessments, Certifications) rely on the Support Attestations defined here to verify signatures, resolve service operators, and confirm cross-system identity relationships.

The first two schemas in this section serve two distinct but complementary purposes: **Identity Correlation** and **Key Authorization**. The **Linked Identifier Attestation** (Section 6.1) is used for general-purpose identity linkage to establish common control between a Subject DID and any other external identifier. The **Key Binding Attestation** (Section 6.2) is a specialized schema used exclusively to define the functional authorization, usage scope, and lifecycle (expiration/revocation) for a raw cryptographic key. Implementers **MUST** choose the appropriate schema based on whether the goal is to prove *who* controls an identity or *what* a specific key is permitted to do.

6.1 Linked identifier

6.1.1 Linked Identifier Purpose

This schema defines the Linked Identifier Attestation, a fundamental mechanism for establishing verifiable, common identity control within the OMATrust ecosystem.

The core principle is a cryptographically verifiable statement where an entity (the Attester) asserts that the controller of a DID (Subject ID) also controls a specific external identifier (Linked ID). In other words, It asserts that Subject ID and Linked ID are controlled by the same entity.

- **Subject ID:** MUST be a Decentralized Identifier (DID). This is the base identity to which the new link is being attested.
- **Linked ID:** MUST be a Decentralized Identifier (DID). This is the external "Controller" identifier being linked to the Subject DID. This can be a social media handle via the **`did:handle`** method.

6.1.2 Linked Identifier Fields

A Linked Identifier attestation MUST conform to the following structure. The normative JSON schema is published in the OMA3 schema repository at https://github.com/oma3dao/rep-attestation-tools-evm-solidity/blob/main/schemas-json/linked-identifier.schema.json.

Field	Req	Format	Description
attester	Y	string	Attester ID. DID of the entity making the assertion. This entity has verified the linkage between the **`subject`** and **`linkedId`**.
subject	Y	string	Subject ID. DID of the subject (such as **`did:pkh`** for contract addresses or **`did:web`** for web domains) that is claimed to control the linked identifier. This SHOULD be the DID the clients will most often search for (see below)
linkedId	Y	string	Linked Identifier. The Controller identifier being claimed as controlled by the subject (e.g., a **`did:web`**, **`did:handle`**).
revoked	N	bool	Indicates if this linked identifier attestation has been revoked.
proofs	N	[object]	Required if **`method`** = **`proof`** or **`social-post`**. Each entry MUST be a Proof wrapper. Proof wrapper schema and **`proofType`**-specific verification are defined in the Proof Specification.
issuedAt	Y	integer	Issued Date. Unix timestamp (in seconds) when the attestation was issued.
effectiveAt	N	integer	Effective Date. Optional Unix timestamp (in seconds) when the assessment becomes effective.
expiresAt	N	integer	Expiration Date. Unix timestamp (in seconds) after

			which the assessment expires. Leave empty if the assessment does not expire.

Note: To ensure search efficiency across common indexers and to designate the most durable identity, the Subject ID SHOULD represent the primary, canonical root of the entity (e.g., an organizational **`did:web`**, an immutable **`did:pkh`** for a smart contract, or a **`did:handle`** for a social account that wants to bind keys). Identifiers with higher rotation frequency (e.g., a **`did:pkh`** for a signing key) SHOULD be assigned to the **`linkedId`** field.

6.1.3 Linked Identifier Proof Parameter Binding

A Linked Identifier attestation MAY include one or more Proofs to enable trustless validation. A Proof consists of multiple Parameters that enable a client to self-verify the attestation (see OMATrust Proof Specification). This section describes how the Linked Identifier Fields map to the Proof Parameter values. All Proofs included in a Linked Identifier attestation MUST satisfy the binding requirements in this section. These rules apply to all proof types permitted for Linked Identifier attestations.

Proof Parameter	Value	Notes
Subject	**`subject`**	**`subject`** needs to approve the attestation of common control with **`linkedId`**.
Controller	**`linkedId`**	
proofPurpose	**`shared-control`**	

6.1.4 Linked Identifier Proof Types

Linked Identifier Proofs MUST use one of the **`proofTypes`** specified in this section.

6.1.4.1 Direct Proofs Types

Direct proofs provide a cryptographic assertion of control by the Subject over the **`subject`** identifier and allowance of explicit linkage to the **`linkedId`**.

The following Proof Specification proof types are permitted:

- **`pop-eip712`** — for Subjects capable of producing EIP-712 typed-data signatures.
- **`pop-jws`** — for other signer-capable Subjects.
- **`tx-encoded-value`** — for Subjects that cannot sign but can originate deterministic native-asset transfers.

Constraints:

- The Proof MUST use **`proofPurpose = shared-control`**.

- The chosen **`proofType`** MUST be appropriate for the Subject's capabilities as defined in the OMATrust Proof Specification.

No additional Linked Identifier–specific parameters apply to direct proofs.

6.1.4.2 Evidence Pointer Proof Types

`evidence-pointer` Proofs allow a linkage to be expressed through evidence hosted at a public URL controlled by the Linked Identifier (the controller).

The following proof type is permitted: **`evidence-pointer`**

The **`proofObject.url`** MUST resolve to publicly accessible evidence controlled by the **`linkedId`** (i.e., the Controller).

When the resolved evidence is a human-readable statement (i.e., does not contain an embedded cryptographic Proof Object), it MUST satisfy all of the following:

- Identifier coverage:
  - The statement MUST explicitly reference both:
    - the attestation's **`subject`**, and
    - the attestation's **`linkedId`**.
- Explicit linkage semantics
  - The statement MUST clearly express linkage, acknowledgement, or common control between subject and linkedId. Mere co-occurrence of identifiers, or generic marketing text that mentions both, is insufficient.
- Specificity / non-ambiguity:
  - The statement MUST be specific enough that a reasonable verifier would not confuse it with an unrelated mention, impersonation, or coincidental string match.

If these conditions are not met, a Handle-Link Statement evidence artifact MUST NOT be treated as valid proof of a Linked Identifier relationship.

6.2 Key Binding

6.2.1 Key Binding Purpose

A Key Binding Attestation declares that a particular cryptographic public key is authorized to act on behalf of a subject DID (e.g.- a service). This allows verifiers to confirm that signatures—whether used in Proofs, webhook signing, or automated agents—originate from a key legitimately controlled by the service. Key Binding Attestations also provide a chain-agnostic, verifiable mechanism for key lifecycle management, including publication, rotation, expiration, and revocation.

Unlike Linked Identifiers (which link abstract IDs), Key Binding Attestations link the raw cryptographic material (**`publicKeyJwk`**) required for digital signatures. Key Binding Attestations can be interpreted as a more specialized Linked Identifier attestation (the linkedId/keyId must represent a key) with more functionality for lifecycle management.

6.2.2 Key Binding Fields

A Key Binding attestation MUST conform to the following structure. The normative JSON schema is published in the OMA3 Github Repository.

Field	Req	Format	Description
attester	Yes	string	Attester ID. DID of the entity making the assertion. This entity has verified the linkage between the subject and linkedId.
subject	Yes	string	The DID of the entity that authorizes the key. Ownership MUST be verified.
keyId	Yes	string	The DID that represents the key (e.g.- **`did:pkh`**).
publicKeyJwk	No	object	JWK of the keyId. Includes the full public key, type, and curve.
keyPurpose	Yes	[string]	A list of permitted uses for the key.
revoked	No	boolean	Used if the transport does not support natively
proofs	Yes	[object]	At least one **Proof** object that uses **`proofPurpose=shared-control`**
issuedAt	Yes	integer	Issued Date. Unix timestamp (in seconds) when the attestation was issued.
effectiveAt	No	integer	Effective Date. Optional Unix timestamp (in seconds) when the assessment becomes effective.
expiresAt	No	integer	Unix timestamp

The **`keyPurpose`** field is an array of strings defining the authorized operations for the bound key. These values map directly to W3C DID Core Verification Relationships:

Value	Description
authentication	The key is authorized to authenticate on behalf of the **`subject`** (e.g., logging into a website, establishing a session).
assertionMethod	The key is authorized to sign statements on behalf of the **`subject`** (e.g., issuing attestations or **`proofs`**).

keyAgreement	The key is used for encryption or key derivation (e.g., ECDH) to establish secure communication channels.
capabilityInvocation	The key is authorized to update the **`subject`** DID document itself or invoke high-level permissions.
capabilityDelegation	The key is authorized to delegate rights to other keys.

6.2.3 Key Binding Proof Parameter Binding

A Key Binding attestation MUST include at least one Proof establishing that the Subject authorizes the keyId to act on its behalf.

All included Proofs MUST satisfy the binding requirements in the table below. These rules apply uniformly across all permitted proof types used for Key Binding.

Proof Parameter	Value	Notes
Subject	**`subject`**	**`subject`** needs to authorize **`keyId`**.
Controller	**`keyId`**	
proofPurpose	**`shared-control`**	

A Proof that does not match the required attestation identifiers after canonicalization MUST be ignored for proof-based validation.

6.2.4 Key Binding Proof Types

Key Binding Proofs MUST use one of the **`proofTypes`** specified in this section.

6.2.4.1 Direct Proofs Types

The following direct proof types are permitted for Key Binding Proofs:

- **`pop-eip712`** — for Subjects capable of EIP-712 typed-data signing.
- **`pop-jws`** — for other Subjects capable of producing standard JWS-based signatures.
- **`tx-encoded-value`** — for Subjects unable to sign but capable of originating deterministic native-asset transfers.

Constraints

- The Proof MUST use **`proofPurpose = shared-control`**.

- The chosen **`proofType`** MUST be appropriate for the Subject's capabilities as defined in the OMATrust Proof Specification.

No additional Key Binding–specific proof types apply to direct proofs.

6.2.4.2 Evidence Pointer Proof Types

Proofs MAY be used to bind a **`keyId`** through evidence hosted at a URL controlled by the Subject. This is done using the **`evidence-pointer proofType`**.

Constraints:

- The Proof MUST use **`proofPurpose = shared-control`**.
- The **`proofObject.url`** MUST resolve to evidence controlled by the Subject (the entity authorizing the key).
- If used, evidence SHOULD employ Embedded Cryptographic Proof.
- Handle-Link Statement SHOULD be used with caution, as it provides weaker identity assurance for key authorization.

Additional pointer-specific requirements are defined in the Proof Specification.

6.2.5 Key Lifecycle Semantics

A Key Binding attestation describes the lifecycle state of a key authorized by the Subject. This section defines the semantics for publication, coexistence, expiration, and revocation of key bindings. These semantics apply to all bindings regardless of the proof types used.

6.2.5.1 Publication

A Key Binding attestation that:

- includes at least one valid Proof satisfying §6.2.3,
- is not expired, and
- is not revoked,

constitutes an active authorization of the **`keyId`** to perform the operations listed in **`keyPurpose`**.

A Subject MAY publish multiple valid Key Binding attestations over time to authorize new keys.

6.2.5.2 Key Rotation

Key rotation is represented by issuing additional Key Binding attestations for new **`keyId`** values.

- Multiple non-expired, non-revoked Key Binding attestations MAY coexist.
- Clients MUST treat each non-expired, non-revoked binding as independently valid unless a higher-level policy explicitly defines a preferred or "current" key.

A Key Binding attestation does not implicitly revoke earlier bindings unless explicitly indicated via the revoked field.

6.2.5.3 Expiration

If the **`expiresAt`** field is present:

- The Key Binding attestation MUST be considered inactive after the specified timestamp.
- Expiration MUST be interpreted using Unix time (seconds).

Expiration affects only the authorization state of the key; it does not invalidate the attestation as a historical record.

If **`expiresAt`** is omitted, the binding remains valid indefinitely unless revoked.

6.2.5.4 Revocation

Revocation provides an explicit mechanism to deactivate a key, independent of expiry.

- If the revoked field is **`true`**, the binding MUST be considered inactive regardless of the value of **`expiresAt`**.
- A revoked binding MUST NOT be used to authorize signatures, perform verification relationships, or fulfill the actions defined in **`keyPurpose`**.
- Revocation does not delete or override the historical Key Binding attestation; it only indicates its current lifecycle state.
- Revocation is effective from the timestamp at which the revocation attestation is issued.

6.2.6 Key Binding Failure Conditions

A Key Binding attestation MUST be treated as invalid for proof-based trust if any of the following conditions occur. These failure conditions apply in addition to the verification rules of the OMATrust Proof Specification.

6.2.6.1 Invalid or Missing Proof Bindings

A Key Binding attestation is invalid if:

- No included Proof satisfies the binding requirements in §6.2.3.
- Any included Proof uses a **`proofPurpose`** other than **`shared-control`**.
- The proof's subject does not match the attestation's subject after canonicalization.
- The proof's controller does not match the attestation's keyId after canonicalization.

Proofs that fail binding MUST be ignored; if none remain valid, the attestation fails.

6.2.6.2 Invalid or Unsupported Key Identifier

A Key Binding attestation is invalid if:

- **`keyId`** is not a DID representing a cryptographic key, or
- **`keyId`** uses a method or format not supported by the binding or verifier.

If present, **`publicKeyJwk`** MUST correspond to **`keyId`** according to the rules of the key's DID method.

6.2.6.3 Invalid or Missing Proof Bindings

A Key Binding attestation is inactive and MUST NOT be used for authorization if:

- **`expiresAt`** is present and has passed, or
- **`revoked`** is **`true`**.

Inactive attestations remain valid as historical records but MUST NOT authorize any verification relationship or operation.

6.2.6.4 Invalid or Missing Proof Bindings

A Key Binding attestation is invalid if required fields in §6.2.2 are missing or malformed:

- **`keyPurpose`** contains values not recognized by the binding.
- **`publicKeyJwk`** is present but malformed or inconsistent with **`keyId`**.

These errors prevent interpretation of the binding and must result in rejection.

7. Reputation Attestations

This section specifies the *canonical technical foundations* of OMATrust attestations: the data model, JSON schema definitions, service-proof formats, and the key-binding framework. While earlier sections describe the system conceptually, this section codifies the *normative rules* that issuers, verifiers, clients, and indexing services **MUST** follow for cross-chain and cross-runtime interoperability. The subsections that follow define the attestation object model, the encoding and verification of service-use proofs, the binding of signing keys to DIDs, and the required verification logic that ensures consistency, authenticity, and security throughout the protocol.

7.1 User Review Schema

7.1.1 Purpose

A User Review attestation expresses a reviewer's evaluation of a **`subject`** service or application. User Reviews provide structured, machine-readable feedback that may include:

- a numerical rating,
- a textual summary or full review body,
- optional reviewer identity information, and

- optional cryptographic evidence demonstrating reviewer identity or actual service usage.

User Reviews differ from Linked Identifier and Key Binding attestations in that they do not establish control relationships or identity delegation. Instead, they record subjective experience, optionally augmented with verifiable evidence.

In the trust model of this specification:

- A User Review MAY include zero or more Proofs.
- Absence of Proofs does not invalidate a review; clients determine how to weight unverified reviews.
- When Proofs are present, they MAY demonstrate:
    - reviewer identity (**`proofPurpose`** = **`shared-control`**), and/or
    - service usage or interaction (**`proofPurpose`** = **`commercial-tx`**).

This flexibility supports diverse review ecosystems, ranging from pseudonymous feedback to cryptographically verified usage claims.

7.1.2 User Review Fields

A User Review attestation MUST conform to the structure defined in user-review.schema.json in the OMA3 Github Repository. This section describes the semantics of the primary fields.

Field	Req	Format	Description
attester	Yes	string	The DID of the entity submitting the review. This MAY be the reviewer (self-attestation) or an intermediary platform acting on behalf of the reviewer.
subject	Yes	string	The DID representing the entity being reviewed (e.g., an application, service, protocol, or website). All usage-related proofs MUST refer to this entity as the target of interaction.
version	No	string	Version of the **`subject`** (e.g., app or API version).
ratingValue	Yes	integer	A numerical rating expressing the reviewer's evaluation of the subject. High rating can be a maximum of 5. Low rating can be a minimum of 1.
reviewBody	No	string	Max length of 500 chars.
screenshotUrls	No	[string]	Array of URL strings that point to screenshots.
proofs	No	[object]	See below
issuedAt	Yes	integer	

7.1.3 Proof Usage

A User Review MAY include zero or more Proofs. Proofs are optional and do not affect the structural validity of a review. When present, Proofs provide additional trust signals that clients MAY use to prioritize, score, or annotate reviews.

User Reviews support two categories of evidence:

- Reviewer identity evidence (**`proofPurpose = shared-control`**)
- Service-usage evidence (**`proofPurpose = commercial-tx`**)

Both are permitted, but they serve distinct purposes and MUST be interpreted according to the rules of this section.

7.1.3.1 Proof Binding Requirements

User Review Proofs MUST bind correctly to the reviewer (**`attester`**), and the reviewed service (**`subject`**) according to the specific proof type and scenario. This section defines binding rules for allowed categories of Proofs in User Reviews.

7.1.3.1.1 Commercial Transactions

This Proof proves that a commercial transaction happened between the user and the service.The only allowed proof types for this binding are:

- **`x402-receipt`**

The Proof Parameter values MUST be:

Proof Parameter	Value	Notes
Subject	**`subject`**	**`subject`** confirms a commercial transaction took place with the user/**`attester`**.
Controller	**`attester`**	
proofPurpose	**`commercial-tx`**	

7.1.3.1.2 Transaction Interaction

This Proof proves that an on-chain transaction, identified by (**`chainId, txHash`**), represents a meaningful interaction between a reviewer and a smart contract–based service. The only allowed proof types for this binding are:

- **`tx-interaction`**

The Proof Parameter values MUST be:

Proof Parameter	Value	Notes
Subject	**`subject`**	**`subject`** represents the service contract that the **`attester`** has interacted.
Controller	**`attester`**	
proofPurpose	**`commercial-tx`**	

7.1.3.1.3 User Account Existence

This binding covers Proofs that the reviewer (**`attester`**) has an account on the service being reviewed. The service MUST expose a publicly accessible URL for user accounts that the service *and* the user controls. The only allowed proof type for this binding is:

- **`evidence-pointer`**

The Proof Parameter values MUST be:

Proof Parameter	Value	Notes
Subject	**`subject`**	**`subject`** represents service the **`attester`** is reviewing.
Controller	**`attester`**	**`attester`** represents the reviewer whose account existence on the service is being proven.
proofPurpose	**`shared-control`**	

7.1.4 Updates and Supersession

Attestations are immutable once created. To update a review, the attester MUST create a new User Review attestation for the same Subject. The new attestation supersedes any previous reviews from the same Attester for that Subject.

Clients MUST implement supersession logic when displaying reviews:

- When multiple User Review attestations exist from the same Attester for the same Subject, clients MUST consider only the most recent attestation (determined by attestation **`issuedAt`**).
- Clients MAY provide UI to view superseded reviews as "review history" if desired.

7.1.5 Verification

A User Review attestation's verification is a multi-step process that establishes its authenticity, data integrity, and the strength of the claim, which ultimately determines the Trust Level of the review.

Because User Review attestations MAY contain cryptographic Proof objects, verification primarily relies on two models: Proof-based (trustless) validation and Trusted-attester validation, as described in §5.1.

Clients MUST apply the following rules to evaluate a User Review:

1. A review that satisfies one of the following SHOULD qualify for a higher Trust Level.
    a. Proof-based Validation (Trustless)
    b. Trusted-Attester Validation
2. The attestation MUST conform to the structure and constraints defined in the User Review JSON schema.
3. Required fields, including attester and subject, MUST be present and well-formed (i.e., valid DIDs).
4. The issuedAt field MUST be a non-negative Unix timestamp.
5. If there are multiple attestations with the same Attester, Subject, and version only the most recent one SHOULD be considered.

7.2 User Review Response Schema

7.2.1 Purpose

A User Review Response attestation records an official response from a service operator (or representative) to a previously issued User Review attestation. It is the protocol's canonical way for app owners / service providers to acknowledge, rebut, clarify, or resolve a specific review in a machine-indexable format.

Key properties:

- Responses are linked to a single User Review via **`refUID`**.
- Responses are service-side statements, not new reviews.
- Responses do not change or override the original review; they exist as an additional reputational signal associated with the same service and reviewer.
- The **`attester`** is the service owner or an authorized delegate. This is typically proven or inferred through Support Attestations (e.g., Linked Identifier or Key Binding) and/or local trust policy.

7.2.2 User Review Response Fields

A User Review Response attestation MUST conform to the normative JSON schema user-review-response.schema.json in the OMA3 Github Repository. The primary fields are:

Field	Req	Format	Description
attester	Yes	string	This SHOULD correspond to the **`subject`** of the reviewing being responded to or a delegate of **`subject`**.
subject	No	string	If present, MUST match the **`attester`** of the referenced User Review attestation.
refUID	Yes	string	UID of the User Review attestation this response targets. This is the canonical linkage between response and review.
responseBody	Yes	string	Max length of 500 chars.
issuedAt	Yes	integer	

7.2.3 Updates and Supersession

Attestations are immutable once created. However, clients SHOULD NOT limit the number of attestations to evaluate for a given **`refUID`**.

7.2.4 Verification

A User Review Response attestation is considered verifiable when it can be consistently linked to an existing User Review and when the responder identity is coherent with that linkage. Because this schema does not carry Proof objects, verification relies on reference resolution and field consistency only.

Clients verifying a User Review Response MUST apply the following rules:

1. Reference Resolution
    a. **`refUID`** MUST resolve to a valid User Review attestation under the transport's lookup rules (transport binding).
    b. If **`refUID`** cannot be resolved, the response MUST be treated as unbound and SHOULD NOT be displayed as a response to any review.
2. Reviewer Consistency
    a. The response **`subject`** MUST match the reviewer identity of the referenced User Review.
    b. If **`subject`** does not match, the response MUST NOT be associated with that review.
    c. Note that **`subject`** is optional. If it is not present then the verifier MUST derive **`subject`** from User Review attestation using **`refUID`**.
3. Responder Coherence
    a. The response **`attester`** verifiably represents the entity identified as **`subject`** in the refUID attestation.

        i. The response **`attester`** MUST be the reviewed service or a verifiable delegate of the reviewed service.
        ii. The reviewed service is the **`subject`** of the referenced User Review attestation (**`refUID`**).
        iii. If the response **`attester`** equals the **`subject`** of the referenced User Review attestation (**`refUID`**), responder coherence holds.
        iv. If not, clients SHOULD verify coherence using:
            1. Support Attestations (e.g., Linked Identifier, Key Binding) defined in §6, and/or
            2. local trusted-attester policy.
        v. If coherence cannot be established, the response MUST be treated as not verified.
4. Field Validity
    a. **`responseBody`** MUST be present.
    b. **`issuedAt`**, if present, MUST be a non-negative Unix timestamp representing issuance time.
5. Updates

If all rules (1)–(2) pass, the response is validly linked to the referenced User Review. Rules (3)–(4) affect trust UI and scoring but do not invalidate linkage.

7.3 Endorsement Schema

7.3.1 Purpose

An Endorsement attestation is a lightweight reputational signal indicating that an Attester supports, trusts, approves, or recognizes a DID-identified Subject. Endorsements are intentionally minimal so they can be issued frequently and interpreted flexibly by different clients.

Endorsements are attester-centric trust signals: the meaning and weight of an endorsement depends primarily on whether the client trusts the endorser. For example, if **`attester`** endorses a **`subject`** as an OMA3 member, a relying party MUST be able to determine that the endorsement was actually issued by OMA3 (or an authorized delegate of OMA3). This determination is made outside the endorsement payload itself, using Support Attestations (e.g., Linked Identifier, Key Binding with **`did:web:oma3.org`**) and/or local trusted-attester policy as described in §6 and §5.

Endorsements MAY be informal (“we like / recommend this”) or formal (“this subject meets program criteria”). The optional **`policyURI`** field allows a formal approval to reference the criteria or process under which it was granted.

7.3.2 Endorsement Fields

An Endorsement attestation MUST conform to the normative JSON schema endorsement.schema.json in the OMA3 Github Repository. The primary fields are:

Field	Req	Format	Description
attester	Yes	string	DID of the entity issuing the endorsement or approval.
subject	Yes	string	DID of the entity being endorsed or approved.
organization	No	string	DID of a parent organization when endorsing a service/resource that belongs to that organization. Omit when endorsing an organization directly.
version	No	string	Optional semantic version of the endorsed subject. Used if **`subject`** is a service with versioning.
policyUri	No	string	Optional URI pointing to the formal criteria or process used for approvals.
payload	No	integer	Optional evolvable details about the endorsement. See Section 8.
payloadVersion	N	string	
payloadSpecURI	N	string	
payloadSpecDigest	N	string	
issuedAt	Yes	integer	
effectiveAt	No	integer	
expiresAt	No	integer	

7.3.3 Endorsement Verification

Because the Endorsement schema does not include Proof objects, verification is based on structural validity, lifecycle semantics, and trusted-attester evaluation.

Clients and indexers verifying an Endorsement MUST apply the following rules:

Schema Validity

1. The Endorsement attestation MUST include the required fields.
2. Lifecycle Semantics
    a. If **`effectiveAt`** is present and greater than the current time, the attestation MUST be treated as not yet effective.
    b. If **`expiresAt`** is present and less than or equal to the current time, the attestation MUST be treated as expired.
    c. **`issuedAt`**, **`effectiveAt`**, and **`expiresAt`**, if present, MUST be non-negative Unix timestamps (seconds).
3. Organization / Version Consistency

    a. If **`organization`** is present, it MUST be interpreted as the parent organization of the endorsed **`subject`**. Clients MAY use this field to group endorsements or apply organization-scoped trust policies.
    b. If **`version`** is present, it MUST be interpreted as an endorsement scoped to that semantic version of the **`subject`**. Omission indicates a general endorsement of the **`subject`** independent of **`version`**.
4. Trusted-Attester Determination
    a. The meaning and weight of an endorsement depend primarily on whether the client trusts the **`attester`**.
    b. Endorsements MUST NOT be treated as trust-verified unless the **`attester`** is recognized as a trusted endorser under one of the following pathways:
    c. Direct Trust
        i. The **`attester`** DID is explicitly trusted by the client (e.g., allowlisted endorser set, program policy, or local trust graph).
    d. Delegated / Authorized Trust
        i. The client SHOULD treat an endorsement as trusted when the **`attester`** is verifiably authorized to speak for a trusted endorser organization.
        ii. Authorization SHOULD be established using support attestations defined in §6, such as:
            1. Linked Identifier from the trusted endorser DID to the **`attester`** DID, indicating delegated control; and/or
            2. Key Binding showing that the signing key used by the **`attester`** is bound to (or delegated by) the trusted endorser DID.
        iii. Clients MAY also apply equivalent local delegation rules (e.g., organizational key registries), provided they are deterministic within that client.

If rules (1)–(3) pass, the endorsement is structurally valid and lifecycle-interpretable. Rule (4) governs whether the endorsement contributes to trust-verified reputation versus unverified reputation.

7.4 Certification Schema

7.4.1 Purpose

A Certification attestation records a formal decision by a Certification Body (CB) that a DID-identified **`subject`** has satisfied the requirements of a specific certification program.

Certifications follow a three-party flow:

1. An **`assessor`** (test lab, auditor, or other evaluation entity) evaluates the **`subject`** and produces assessment data.
2. The Certification Body receives that data and makes the certification decision.
3. If the decision is positive, the Certification Body issues this Certification attestation as the **`attester`**.

Certifications are closely related to Endorsements but differ in two fundamental ways:

1. Assessor-mediated decision flow: An Endorsement is issued directly by an Attester to a Subject as a standalone trust signal. A Certification introduces an explicit intermediary: an Assessor evaluates the Subject and produces assessment results, and a Certification Body (Attester) then issues the certification decision based on that assessment. This Assessor role is part of the certification record and is required for interpreting how the decision was reached.

2. Program-formal and program-scoped: Endorsements are intentionally lightweight and typically do not imply adherence to a defined process. Certifications, by contrast, are issued under formal certification programs with defined criteria, governance, and often their own persistent identity. Every Certification is therefore scoped to a specific program identified by programID (with optional descriptive context via programURI). The trust weight of a Certification depends on the client recognizing and trusting both the Certification Body and the program, in addition to the Assessor's authorization within that program.

The Certification schema does not embed assessor authorization lists. Instead, assessor authorization is determined externally through program governance referenced by **`programID/programURI`**, and/or through other reputation or support Attestations (e.g., Endorsements that designate authorized assessors for a program).

7.4.2 Certification Fields

A Certification attestation MUST conform to the normative JSON schema certification.schema.json in the OMA3 Github Repository. The primary fields are:

Field	Req	Format	Description
attester	Yes	string	DID of the assessor issuing the security assessment.
subject	Yes	string	DID of the product, system, service, or organization being assessed.
organization	No	string	DID of the parent organization that owns or governs the certified subject.
version	No	string	Optional software version of the certified subject. Omit when certifying the subject independent of software version.
versionHW	No	string	Optional hardware version of the certified subject, if applicable. Omit when hardware version is not relevant.
subjectURI	No	string	Optional URI controlled by the subject that provides mutable, human-readable metadata.
programId	Yes	string	DID of the certification program under which this

			certification was issued. Program release version should be included in the DID.
programURI	No	string	Optional URI for mutable, human-readable certification program info.
assessor	Yes	string	DID of the authorized assessor (test lab, auditor) who evaluated the subject
assessorURI	No	string	Optional URI with human-readable info about the assessor (e.g., homepage, credentials).
certificationLevel	No	string	Optional classification level of certification (e.g., 'Gold', 'Level 2').
outcome	No	string	Values can be **`pass`** or **`fail`**. If omitted, outcome MUST be interpreted as **`pass`**.
reportURI	N	string	Optional URI of an off-chain assessment report.
reportDigest	N	object	Optional digest enabling integrity verification of reportURI content. Includes algorithm and canonicalization metadata. Uses the same fields verification as **`payloadSpecDigest`** (Section 8).
payload	N	object	
payloadVersion	N	string	
payloadSpecURI	N	string	
payloadSpecDigest	N	string	
issuedAt	Yes	integer	
effectiveAt	No	integer	
expiresAt	No	integer	

7.4.3 Certification Verification

Because the Certification schema does not include Proof objects, verification is based on structural validity, lifecycle semantics, program scoping, **`assessor`** authorization, and trusted-attester evaluation.

Clients verifying a Certification MUST apply the following rules:

1. The Certification attestation MUST include the required fields.
2. Lifecycle Semantics

    a. If **`effectiveAt`** is present and greater than the current time, the certification MUST be treated as not yet effective.
    b. If **`expiresAt`** is present and less than or equal to the current time, the certification MUST be treated as expired.
    c. **`issuedAt`**, **`effectiveAt`**, and **`expiresAt`**, if present, MUST be non-negative Unix timestamps (seconds).
3. Program Scoping
    a. The certification MUST be interpreted as valid only within the rules of the program identified by **`programId`**.
4. Assessor Authorization (Non-Proof)
    a. **`assessor`** is a claim that the named Assessor evaluated the **`subject`** in accordance with the program.
    b. Clients SHOULD verify that `assessor` is authorized by **`programID`** by one or more of:
        i. resolving program governance via **`programId`** or **`programURI`** (e.g., on-chain program registry or program-defined authorization rules), and/or
        ii. locating Endorsements that designate the Assessor as authorized under the program.
    c. If Assessor authorization cannot be established, the certification MUST be treated as not verified for trust purposes, even if structurally valid.
5. Trusted-Attester Determination
    a. The meaning and weight of a Certification depend primarily on whether the client trusts the Certification Body (**`attester`**).
    b. Certifications MUST NOT be treated as trust-verified unless the Attester is recognized as a trusted CB.

If rules (1)–(3) pass, the Certification is structurally valid, lifecycle-interpretable, and program-scoped. Rules (4)–(5) govern whether the certification contributes to trust-verified reputation versus unverified reputation.

7.5 Security Assessment Schema

7.5.1 Purpose

A Security Assessment attestation records the results of a security evaluation performed by a DID-identified assessor over a DID-identified subject.

The roles are:

- **`attester`**: the security assessor or assessing organization issuing the attestation.
- **`subject`**: the product, system, or service being assessed.

Security Assessments are designed as thin envelopes with most methodological detail, evidence, and results carried in **`payload`**. This keeps the core reputation layer stable while allowing different assessment methodologies and reporting styles to evolve through payload schemas.

The meaning and trust weight of a Security Assessment depend primarily on whether the client trusts the **`attester`**. Clients MAY ingest structurally valid assessments from untrusted Attesters but SHOULD treat them as unverified signals for scoring and filtering purposes.

Each assessment explicitly declares its **`assessmentKind`** to support indexing and filtering without requiring clients to parse provider-specific payloads. Values for **`assessmentKind`** are registry-driven via **`x-oma3-enum`** rather than schema-frozen (See Section 9).

Payload interpretation follows a default/override model:

- If **`payloadSpecURI`** is absent, the payload is interpreted using the default payload structure defined in this schema.
- If **`payloadSpecURI`** is present, the payload is interpreted using the referenced payload specification, which MAY extend or override the default structure.

7.5.2 Security Assessment Fields

A Security Assessment attestation MUST conform to the normative JSON schema security-assessment.schema.json in the OMA3 Github Repository. The primary fields are:

Field	Req	Format	Description
attester	Yes	string	DID of the Certification Body issuing the certification.
subject	Yes	string	DID of the product, system, service, or organization being certified.
organization	No	string	DID of the parent organization that owns or governs the certified subject. Use when certifying a service/resource of an org; omit when certifying an organization directly.
version	No	string	Optional software version of the certified subject. Omit when certifying the subject independent of software version.
versionHW	No	string	Optional hardware version of the certified subject, if applicable. Omit when hardware version is not relevant.
issuedAt	Yes	integer	
effectiveAt	No	integer	
expiresAt	No	integer	
payload	N	object	
payloadVersion	N	string	

payloadSpecURI	N	string	
payloadSpecDigest	N	string	

A Security Assessment attestation MUST conform to the normative JSON schema security-assessment.schema.json. The primary fields are:

Field	Req	Format	Description
assessmentKind	Yes	string	Enum (pentest, security-audit, code-review, or vulnerability-scan)
period	No	object	{start,end} unix seconds; UI-suppressed.
methodURI	No	string	Methodology location.
reportURI	No	string	Human-readable report location.
reportDigest	No	object	
outcome	No	string	Values can be **`pass`** or **`fail`**. If omitted, outcome MUST be interpreted as **`pass`**.
metrics	No	object	
policy			

If the period field is present it MUST have the following fields:

Field	Req	Format	Description
start	Yes	integer	Unix timestamp
end	Yes	integer	Unix timestamp

If the metrics field is present it MUST have the following fields:

Field	Req	Format	Description
critical	Yes	integer	Number of critical vulnerabilities
high	Yes	integer	
medium	Yes	integer	
low	Yes	integer	
info	Yes	integer	Number of informational findings

7.5.3 Security Assessment Verification

Because the Security Assessment schema does not include Proof objects, verification is based on structural validity, lifecycle semantics, payload interpretation, and trusted-attester evaluation.

Clients and indexers verifying a Security Assessment MUST apply the following rules:

1. Schema Validity
    a. The assessment MUST include the required fields.
    b. Required and optional fields MUST satisfy the type and format constraints of **`security-assessment.schema.json`**.
2. Lifecycle Semantics
    a. **`issuedAt`** MUST be a non-negative Unix timestamp (seconds).
    b. If **`effectiveAt`** is absent, clients MUST treat the assessment as effective at **`issuedAt`**.
    c. If **`effectiveAt`** is present, it MUST be a non-negative Unix timestamp (seconds).
3. Assessment Kind Handling
    a. **`assessmentKind`** MUST be present.
    b. Values not listed in **`x-oma3-enum`** are valid unless a client's local policy rejects them.
4. Outcome Defaulting
    a. If outcome is absent, the assessment MUST be interpreted as a **`pass`**.
5. Payload Interpretation
    a. If **`payloadSpecURI`** is absent, the payload MUST be interpreted using the default payload structure defined in this schema.
    b. If **`payloadSpecURI`** is present, the payload MUST be interpreted using the referenced payload specification.
    c. Clients MAY treat **`payload`** as opaque and ignore them for trust/scoring purposes when the payload specification is missing, unrecognized, or fails local validation.
6. Report Integrity (Optional)
    a. If **`reportURI`** and **`reportDigest`** are present, clients SHOULD fetch the report and verify its digest using the declared algorithm and canonicalization rules.
    b. If digest verification fails, the report MUST be treated as invalid and ignored; the on-chain envelope remains the authoritative assessment record.
7. Trusted-Attester Determination
    a. The meaning and weight of a Security Assessment depend primarily on whether the client trusts the attester.
    b. Assessments MUST NOT be treated as trust-verified unless the attester is recognized as trusted under direct trust, delegated trust via Support Attestations, or local trusted-attester policy as described in §5.1 and §5.3.3.

If rules (1)–(3) pass, the assessment is structurally valid and lifecycle-interpretable. Rules (4)–(7) govern how payload, evidence, and attester trust affect reputation scoring and filtering.

8. Utility Schemas

This section defines reusable utility structures that appear across multiple attestation types. Utility schemas are not primary reputation attestations on their own; they provide shared field patterns and interpretation rules used by attestations in §§6–7.

The utilities in this section are non-proof structures. Proof objects are also reusable utilities attached to attestations, not attestations on their own, but they are defined in the OMATrust Proof Specification.

8.1 Payload Container

Many OMATrust attestations include a **`payload`** field to carry attestation-specific details that are expected to evolve over time. Payloads are intentionally flexible to avoid frequent structural schema updates.

`payload` is a JSON object whose internal fields are attestation specific. Unless a payload specification is referenced via **`payloadSpecURI`**, clients MUST treat payload contents as opaque data that MAY be ignored for trust/scoring purposes.

8.1.1 Default and Custom Payloads

Payload interpretation follows a default/override model:

1. Default payload: If an attestation defines a default payload structure within its schema and omits **`payloadSpecURI`**, then the **`payload`** MUST be interpreted under that default structure.
2. Custom payload: If **`payloadSpecURI`** is present, the payload MUST be interpreted under the referenced payload specification. The referenced specification MAY extend or override the default payload structure.

Clients MAY classify payloads that omit **`payloadSpecURI`** as “default-schema payloads,” and payloads that include it as “custom-schema payloads.”

If the client cannot resolve or recognize a referenced spec, the client MUST still treat the attestation envelope as valid, but MAY ignore payload contents for trust/scoring purposes.

8.1.2 Payload Metadata Fields

Payload metadata fields describe the structure, versioning, and integrity of payloads. These fields are shared utilities referenced by multiple attestation schemas. They are defined in the **`common.schema.json`** file.

Payload Fields	Description
payload	The payload object.
payloadVersion	Identifies the version of the payload structure (**`x.y.z`**).
payloadSpecURI	Points to a document that defines the payload's schema or normative interpretation rules.
payloadSpecDigest	Hash of the payload's schema for schema integrity confirmation.

The **`payloadSpecDigest`** is an object with the following fields:

Field	Description
algo	Hash algorithm (keccak256 or sha256).
canon	Canonicalization method: **`raw`** for binary/text files, **`jcs`** for JSON (RFC 8785).
hex	0x-prefixed hex hash of the payload specification.

Verification:

- If both **`payloadSpecURI`** and **`payloadSpecDigest`** are present, clients SHOULD fetch the spec and verify its digest.
- If digest verification fails, clients MUST treat the referenced spec as invalid and MAY fall back to default-schema interpretation or treat **`payload`** as opaque.

9. Schema Publication and Versioning

This section defines where OMATrust normative JSON schemas are published, how schema versions are managed over time, how clients resolve schema UIDs to specific schema versions, and how to interpret OMA3-specific schema annotations (**`x-oma3-*`**). These rules ensure that attestations remain structurally stable and machine-validated across releases, while allowing the specification's semantic guidance to evolve independently.

9.1 Payload Container

Normative JSON schemas for all Support and Reputation attestations are published in the OMA3 schema repository on GitHub. Implementations MUST validate attestations against the schemas in that repository, as pinned by a tagged schema-set release.

9.1.1 Tagged schema-set releases

Schemas MUST be referenced by immutable tagged releases (e.g., **`schemas/vx.y.z/`**). Mutable branches (including main) are not normative unless explicitly bound to a tagged release. A tagged schema-set release represents the authoritative structural definitions for a specific OMATrust specification version.

9.1.2 Spec ↔ schema-set mapping

Each OMATrust specification version maps to exactly one schema-set tag. References in §§6–7 to “the normative schema” mean the schema file contained in the schema-set tag corresponding to this specification version.

9.1.3 Immutability expectation

Structural changes to any schema require registering a new schema with the underlying attestation framework, producing a new schema UID. Because schema UID changes impose real client migration costs, schemas are expected to evolve infrequently compared to this specification text. Semantic or verification changes that do not alter structure SHOULD NOT require schema changes.

9.1.4 Schema-set structure

Each schema-set release SHOULD be stored in a versioned folder (e.g., **`schemas/vx.y.z/`**) containing:

- all current ratified schema JSON files,
- the pinned Common JSON Schema for that release, and
- a registry file as defined in §9.2.

9.1.5 Backward compatibility

Clients MUST NOT assume a single active schema UID for a given attestation type. Clients SHOULD support the set of schema UIDs published in the registry for this spec version and MAY additionally support older UIDs according to local policy.

9.2 Schema UID Registry

Because attestations are often identified by schema UID, each schema-set release MUST publish a machine-readable registry file that maps schema UIDs to their structural definitions.

The registry MUST include, at minimum:

- schema-set version,
- generation timestamp,
- for each schema:
    - schema name,
    - schema file path,
    - schema version,
    - schema UID(s) and associated chain/namespace,
- a pinned Common JSON Schema binding for the set, including the common schema file name and a digest identifying the exact common version used.

Clients MUST rely on the registry to resolve a schema UID to a specific schema version and file. Clients MUST NOT infer schema semantics purely from schema name or from transport metadata.

Within a tagged schema-set release, the registry is immutable. Any change to a published registry requires a new schema-set version and tag.

9.3 x-oma3 Schema Annotation Semantics

Schemas in the OMA3 repository may include OMATrust-specific JSON Schema extension fields prefixed with **`x-oma3-`**. These extensions are not part of normative structural validation; standard JSON Schema validators MUST ignore them.

The purpose of **`x-oma3-*`** extensions is to allow OMATrust to improve UX and interoperability without changing registered schema structures. In practice, updating a registered schema requires a new transport-level schema registration which could result in a new schema UID. New schema UIDs impose real client migration costs (clients must track and support multiple UIDs). Therefore, schemas are designed to change infrequently. **`x-oma3-*`** extensions provide a safe way to:

- add or refine UI behavior,
- publish non-binding “known value” lists (e.g., enums), and
- attach lightweight semantic hints,

without requiring a new schema UID. Accordingly, unless a specific extension is referenced elsewhere in this specification, clients MAY treat **`x-oma3-*`** values as advisory metadata.

9.3.1 x-oma3-default

Indicates that a field MAY be automatically populated with a default value by an implementation when omitted. The value of **`x-oma3-default`** indicates what the default value should be:

- **`current-timestamp`** — Field defaults to the current Unix timestamp (seconds).

Auto-population MUST NOT override any explicit value provided by the Attester.

9.3.2 x-oma3-did-methods

An array of DID method strings that provide hints about which DID methods are recommended or expected for a DID-formatted field. Intended for UI guidance and method support signaling (a front end can provide a drop down of DID methods).

- UIs SHOULD use this list to present appropriate DID method choices.
- Verifiers MUST accept any syntactically valid DID for the field, even if its method is not listed here, unless the field also declares strict JSON Schema constraints.
- This extension does not restrict validity.

9.3.3 x-oma3-enum

This array of strings provides a machine-readable list of the currently recognized values for an extensible string field. The list MUST reflect the value space defined in this specification. It exists to support UI and developer ergonomics without freezing those values into a transport-registered schema.

Using a JSON Schema **`enum`** would make the value list structurally binding. Each time a new value is added, the schema would need to be updated and re-registered with the transport, producing a new schema UID and forcing clients to track additional UIDs. **`x-oma3-enum`** avoids that churn by keeping the field structurally open while still publishing “known values” for clients to present as options.

- Implementations SHOULD surface the values listed in **`x-oma3-enum`** as selectable options in UI and tooling.
- Implementations MUST treat the list as advisory for validation: if the underlying schema does not declare a JSON Schema **`enum`**, values not listed in **`x-oma3-enum`** remain structurally valid.
- If an implementation encounters a value not listed in **`x-oma3-enum`**, it SHOULD treat it as “unrecognized but valid” and apply local policy for scoring or display.
- When the specification expands the recognized value set, **`x-oma3-enum`** SHOULD be updated in a new schema-set release without requiring a new transport schema UID, unless other structural changes are also made.

9.3.5 x-oma3-subtype

Provides semantic type information for fields whose base JSON Schema type is generic. **`x-oma3-subtype`** allows a UI to represent a more specific data concept with suitable controls (date pickers, semver inputs). Valid values:

- **`timestamp`** — Unix timestamp in seconds (integer).
- **`semver`** — Semantic version string (string).

9.3.6 x-oma3-handle-platforms

When a DID field supports did:handle via **`x-oma3-did-methods`**, this extension lists commonly supported social handle platforms that allows a UI to make the inputting of the DID easier.

- UIs SHOULD use this list to suggest platform choices for **`did:handle`** identifiers.
- Verifiers MUST accept any syntactically valid **`did:handle:<platform>:<username>`** DID, even if the platform is not listed, unless strict constraints are declared elsewhere.
- This extension does not restrict validity.

9.3.7 x-oma3-skip-reason

`x-oma3-skip-reason` indicates that a field should not be surfaced in the attestation creation UI, and explains why. The extension is a structured hint about data entry responsibility, not a validity rule.

Fields may be marked with **`x-oma3-skip-reason`** for transport- or schema-level reasons, including but not limited to:

- **`metadata`** — The field is part of the JSON-LD / schema-definition ecosystem and is included for standards compatibility (e.g., **`@context`**, **`@type`**). Implementations MAY omit these from UI and MAY auto-populate them as appropriate.
- **`eas`** — The field is automatically provided by the attestation transport layer for every EAS attestation (e.g., EAS supplies attester, timestamps, or other envelope data). Implementations SHOULD NOT require manual user entry for these fields and MAY omit them from UI. Other transports MAY define additional auto-supplied fields; when they do, another value for the **`x-oma3-skip-reason`** field will be added.
- **`unused`** — The field is reserved for future use, legacy compatibility, or optional features not currently supported by default clients. Implementations MAY omit these from UI but MUST preserve them if present.

Notes:

- **`x-oma3-skip-reason`** MUST NOT affect structural validity or verification outcomes.
- Implementations MAY use the tag to decide whether a field should be user-editable, auto-populated, or hidden in default UI.

- If a field is present in an attestation, clients MUST preserve it in storage, indexing, and transmission regardless of skip reason.

9.3.8 x-oma3-render

`x-oma3-render` indicates how an object or array field should be presented in attestation creation UIs. The extension is a structured hint about input presentation, not a validity rule.

Fields may be marked with **`x-oma3-render`** to control UI rendering behavior:

- **`expanded`** — The field's sub-properties should be rendered as individual form inputs. This is the default behavior for object types when no value is specified. Sub-properties should include UI fields, such as **`title`** and **`description`**.
- **`raw`** — The field should be rendered as a single text input accepting raw JSON. Users paste or type the complete JSON object/array rather than filling individual sub-fields. This is appropriate for technical data structures (e.g., JWKs, digests) that are typically generated or exported from external tools

Notes:

- **`x-oma3-render`** MUST NOT affect structural validity or verification outcomes.
- Implementations MAY use the tag to decide whether an object field should be expanded into sub-fields or presented as a single JSON input.
- When **`x-oma3-render`** is **`raw`**, implementations SHOULD validate that the input parses as valid JSON and conforms to the field's schema before submission.
- Local overrides take precedence: a field definition MAY override the render mode inherited from a **`$ref`** reference.

Change History

Version	Date	Comments
0.1	2025-09-25	Initial draft - Alfred Tom
0.2	2025-12-10	First complete draft- Alfred Tom

Appendix A

TBD