

# Trust Packet Format & Canonicalization Specification

## Executive Summary (Non-Normative)

The UniKey Trust Packet is a universal, transport-agnostic cryptographic container for proving identity, authority, and intent.

It enables any verifier—without prior integration, shared state, or platform dependency—to deterministically decide whether an action is authorized.

Trust Packets:

- Survive hostile or lossy internet transports
- Are independently verifiable by any party
- Fail closed under ambiguity or attack
- Support delegation without expanding authority

This specification defines the invariant foundation for secure agent action, payments, and authentication across the open internet.

**Status:** Draft (Normative)

**Purpose:** Ensure cryptographic trust packets remain verifiable across heterogeneous internet transports, intermediaries, and storage systems.

## 1. Overview

A *Trust Packet* is a self-contained, cryptographically signed data structure that asserts identity, authority, and intent. It is designed to survive transmission across hostile or lossy environments (email, APIs, message queues, QR/NFC handoff, storage) without breaking signature verification.

This specification defines:

- The canonical packet structure
- Required and optional fields
- Canonicalization and normalization rules
- Hashing and signature requirements

- Multi-signature and delegation encoding
- Error handling and conformance expectations

This specification intentionally does **not** define:

- User experience or UI flows
- Transport protocols
- Business logic or policy interpretation
- Specific cryptographic algorithms (RSA, ECC, etc.)

## 2. Trust Packet Structure

A Trust Packet is a deterministic, serializable object composed of:

1. **Header** – protocol metadata
2. **Claims** – identity, authority, and context assertions
3. **Payload** – action- or session-specific data
4. **Signatures** – one or more cryptographic attestations

Logical structure:

```
None
TrustPacket {
  header
  claims
  payload
  signatures[]
}
```

## 3. Required and Optional Fields

### 3.1 Header (Required)

Field	Description
<code>tp_version</code>	Trust Packet format version
<code>packet_id</code>	Globally unique identifier

<code>issued_at</code>	UTC timestamp (ISO 8601)
<code>expires_at</code>	UTC expiration timestamp
<code>nonce</code>	Single-use entropy value
<code>canonicalization</code>	Canonicalization scheme identifier

### 3.2 Claims (Required)

Field	Description
<code>subject</code>	Identity being asserted
<code>issuer</code>	Signing authority
<code>scope</code>	Allowed actions
<code>audience</code>	Intended verifier(s)
<code>context</code>	Domain / session binding

### 3.3 Payload (Optional)

The payload contains application-specific data (e.g., transaction details, session references). Payload contents are opaque to this specification but **must** be included in the canonical hash if present.

### 3.4 Signatures (Required)

One or more cryptographic signatures over the canonicalized packet.

## 4. Canonicalization Rules

Canonicalization ensures that semantically identical packets always produce the same hash, regardless of transport-induced transformations.

### 4.1 Field Ordering

- Fields **MUST** be ordered lexicographically by field name

- Nested objects follow the same ordering rule
- Arrays preserve order unless explicitly marked unordered

## 4.2 Encoding

- UTF-8 encoding only
- No BOM markers
- Binary data MUST be base64url encoded (no padding)

## 4.3 Whitespace and Normalization

- No insignificant whitespace
- No line breaks
- Numbers represented without leading zeros
- Boolean values strictly `true` or `false`

## 5. Hashing Rules

1. Canonicalize the packet excluding the `signatures` field
2. Apply the approved cryptographic hash function
3. The resulting digest is the **Trust Packet Hash (TPH)**

All signatures MUST be computed over the identical TPH.

## 6. Multi-Signature / Delegation Chain Encoding

### 6.1 Multiple Signatures

The `signatures` array supports multiple independent attestations.

Each signature entry includes:

Field	Description
<code>signer</code>	Identity of signer
<code>key_id</code>	Public key reference

<code>algorithm</code>	Signature algorithm identifier
<code>m</code>	
<code>signature</code>	Base64url-encoded signature
<code>e</code>	
<code>delegation</code>	Optional delegation reference
<code>on</code>	

## 6.2 Delegation Chains

Delegation is expressed as a linked sequence of Trust Packets, where each packet:

- References the hash of the prior packet
- Narrows (never expands) scope
- Is independently verifiable

Verifiers **MUST** validate the entire chain.

## 7. Size Constraints & Performance Targets

Metric	Target
Typical packet size	< 4 KB
Maximum packet size	16 KB
Verification time	< 5 ms (commodity hardware)
Delegation depth	≤ 5 hops recommended

Packets exceeding limits **MAY** be rejected.

## 8. Examples (Normative)

### 8.1 Minimal Trust Packet (Abstract)

None

```
header: {...}
claims: {...}
payload: {...}
signatures: [...]
```

## 8.2 Delegated Action Packet

A delegated packet MUST include:

- Reference to parent packet hash
- Reduced scope
- Independent signature

## 9. Security Considerations

Trust Packets are designed to operate in adversarial environments. Implementations MUST assume:

- Transports may modify formatting, headers, or ordering
- Intermediaries may inspect, store, replay, or delay packets
- Attackers may attempt substitution, replay, truncation, or scope escalation

Security properties enforced by this specification:

- **Integrity:** Any modification invalidates signatures
- **Authenticity:** Only holders of valid private keys can attest
- **Replay Resistance:** Enforced via `nonce`, `issued_at`, and `expires_at`
- **Scope Containment:** Delegation chains can only narrow authority
- **Fail-Closed Behavior:** Partial or ambiguous validation MUST fail

Cryptographic agility is supported via explicit algorithm identifiers, but verifiers MUST reject unknown or deprecated algorithms.

## 9A. Reference Threat Scenarios (Normative)

### TS-1: Packet Replay

**Attack:** An attacker replays a previously valid packet.

**Mitigation:** nonce uniqueness and strict temporal validation.

**Result:** Rejected.

## TS-2: Scope Escalation via Delegation

**Attack:** A delegated packet attempts to expand authority.

**Mitigation:** Verifiers enforce monotonic scope reduction.

**Result:** Rejected.

## TS-3: Transport Mutation

**Attack:** Whitespace, ordering, or encoding altered in transit.

**Mitigation:** Canonicalization prior to hashing.

**Result:** Signature remains valid.

## TS-4: Signature Substitution

**Attack:** Signature replaced or signer spoofed.

**Mitigation:** Public key resolution and signature verification.

**Result:** Rejected.

# 9B. Licensing and Contractual Reference

This specification is intended to be referenced normatively in commercial, licensing, and stewardship agreements.

Example reference language:

"Trust Packet" means a cryptographically signed data structure conforming to **UniKey RFC-001: Trust Packet Format & Canonicalization Specification**, as amended.

Any implementation claiming UniKey compatibility **MUST** conform to this specification.

Trust Packets are designed to operate in adversarial environments. Implementations **MUST** assume:

- Transports may modify formatting, headers, or ordering
- Intermediaries may inspect, store, replay, or delay packets
- Attackers may attempt substitution, replay, truncation, or scope escalation

Security properties enforced by this specification:

- **Integrity:** Any modification invalidates signatures
- **Authenticity:** Only holders of valid private keys can attest
- **Replay Resistance:** Enforced via `nonce`, `issued_at`, and `expires_at`
- **Scope Containment:** Delegation chains can only narrow authority
- **Fail-Closed Behavior:** Partial or ambiguous validation MUST fail

Cryptographic agility is supported via explicit algorithm identifiers, but verifiers MUST reject unknown or deprecated algorithms.

## 10. Transport Agnosticism

Trust Packets are transport-independent by design.

They MAY be conveyed via:

- SMTP (email headers or bodies)
- HTTPS (request or response payloads)
- Message queues or event streams
- QR or NFC handoff
- Local or remote storage

No transport-specific fields are trusted for security decisions. Only the canonicalized packet and verified signatures are authoritative.

## 11. Canonical JSON Example (Normative)

JSON

```
{
  "header": {
    "tp_version": "1.0",
    "packet_id": "550e8400-e29b-41d4-a716-446655440000",
    "issued_at": "2026-01-30T12:00:00Z",
    "expires_at": "2026-01-30T12:05:00Z",
    "nonce": "Lk9XQpY7",
    "canonicalization": "unikey-json-v1"
  },
  "claims": {
```

```
    "subject": "user@example.com",
    "issuer": "device:samsung-galaxy",
    "scope": ["pay", "login"],
    "audience": "merchant.example",
    "context": "session:abc123"
  },
  "payload": {
    "amount": "49.99",
    "currency": "USD"
  },
  "signatures": [
    {
      "signer": "example.com",
      "key_id": "dkim-2026",
      "algorithm": "rsa-sha256",
      "signature": "MEUCIQDn..."
    }
  ]
}
```

## 12. Error Handling

Verifiers MUST return deterministic error codes.

Code	Condition
TP-00 1	Invalid canonicalization
TP-00 2	Signature mismatch
TP-00 3	Expired packet

TP-00 Invalid delegation chain  
4

TP-00 Unsupported version  
5

Packets failing validation MUST be rejected without partial trust.

## 13. Verifier Pseudocode (Normative)

The following pseudocode illustrates the minimum required verification logic.

None

```
function verifyTrustPacket(packet):  
    if packet.tp_version not supported:  
        return TP-005  
  
    canonical = canonicalize(packet without signatures)  
    hash = hash(canonical)  
  
    for sig in packet.signatures:  
        pubkey = resolvePublicKey(sig.signer, sig.key_id)  
        if not verifySignature(pubkey, hash, sig.signature):  
            return TP-002  
  
    if now < packet.issued_at or now > packet.expires_at:  
        return TP-003  
  
    if replayDetected(packet.nonce):  
        return TP-001  
  
    if delegationPresent(packet):  
        if not verifyDelegationChain(packet):  
            return TP-004  
  
    return VALID
```

All steps MUST succeed. No partial validation is permitted.

## 14. Registry and Extensibility

### 14.1 Canonicalization Registry

Canonicalization identifiers are globally unique strings.

Example:

- `unikey-json-v1`

Implementations MUST reject unknown canonicalization identifiers.

### 14.2 Error Code Registry

Code	Meaning
TP-001	Canonicalization or replay failure
TP-002	Signature validation failure
TP-003	Temporal validity failure
TP-004	Delegation chain failure
TP-005	Unsupported version or algorithm

### 14.3 Algorithm Agility

Signature and hash algorithms are identified explicitly. Verifiers MUST support algorithm agility and MUST reject deprecated algorithms.

## 15. Conformance and Invariant Mapping

This specification is normative and defines UniKey trust invariants.

It maps directly to core UniKey invariants:

- **Deterministic Verification:** Canonicalization and hashing guarantee identical outcomes
- **Independent Verification:** Any verifier can validate without coordination

- **Infrastructure-Level Enforcement:** Trust is evaluated below applications and above transport

An implementation is conformant if it:

- Produces packets matching this format
- Canonicalizes deterministically
- Rejects malformed or unverifiable packets
- Verifies all required signatures and delegation chains

Non-conformant packets MUST be rejected without partial trust.

This specification is normative and defines UniKey trust invariants.

It maps directly to core UniKey invariants:

- **Deterministic Verification:** Canonicalization and hashing guarantee identical verification outcomes
- **Independent Verification:** Any verifier with public keys can validate without coordination
- **Infrastructure-Level Enforcement:** Trust decisions occur below applications and above transport

An implementation is conformant if it:

- Produces packets matching this format
- Canonicalizes deterministically
- Rejects malformed or unverifiable packets
- Verifies all required signatures and delegation chains

Non-conformant packets MUST be rejected without partial trust.

An implementation is conformant if it:

- Produces packets matching this format
- Canonicalizes deterministically
- Rejects malformed or unverifiable packets
- Verifies all required signatures and delegations

This specification defines the invariant trust substrate. All higher-level behavior is implementation-defined.