

QVTM Protocol Specification

Quantum Verifiable Trust Model

Architecture, Protocol Core, Reference Ecosystem, and Deployment Scenarios

Status: Final Release **Scope:** Protocol Specification

Authors: Fernando Guerrero Patricio Guerrero

July the 10th, 2026

Copyright (c) 2026

Part I — Introduction

1. Scope

1.1 Purpose

This specification defines the externally observable and normative behavior required for interoperable implementations of the Quantum Verifiable Trust Mesh (QVTM) protocol.

The Position Paper defines the scientific motivation for Trust Continuity. The White Paper describes the reference architecture and existing reference realizations. This specification defines the protocol obligations that conformant implementations MUST satisfy.

This specification intentionally excludes proprietary implementation details, including internal Graph Puzzle algorithms, continuity-scoring methods, optimization strategies, heuristic selection rules and implementation-specific storage models.

1.2 Scope

This specification defines protocol entities, Protocol Operations, Protocol State Semantics, handshake procedures, message semantics, Continuity Evidence and CEP semantics, synchronization, recovery, canonical representation requirements, version negotiation and interoperability.

The specification defines behavior visible at protocol boundaries. Internal implementation choices MAY differ provided that they preserve all Protocol Invariants, produce semantically equivalent outputs and satisfy the applicable conformance requirements.

1.3 Design Goals

A conformant implementation MUST preserve Trust Continuity semantics, support independent verification, remain cryptographically agile, support disconnected operation, preserve historical Continuity Evidence and interoperate across compatible implementations and Deployment Profiles.

2. Terminology

The terms in this section are normative and preserve the canonical meanings established by the QVTM Position Paper and White Paper.

Term	Normative definition
Trust Continuity	The independently verifiable continuity of a trust relationship across continuity-relevant cryptographic, identity, system, service, organizational and governance changes.
Continuity Evidence	Immutable, independently verifiable protocol information describing one continuity-relevant event, condition or relationship.
Continuity Evidence Package (CEP)	The canonical semantic container used to preserve or exchange one or more Continuity Evidence objects together with their protocol, integrity and relationship context.
Trust State	An observable trust condition evaluated within a defined protocol context.
Continuity Path	An ordered set of semantically related Trust States supported by correlated Continuity Evidence.
Continuity Graph	The protocol representation of Trust States, Continuity Evidence and the semantic relationships among them.
Graph Puzzle	The correlation capability that correlates Continuity Evidence to derive and validate continuity relationships without exposing implementation-specific algorithms.

Term	Normative definition
Trust Evaluation	A Protocol Operation that produces a continuity result for a specified context and applicable policy.
Protocol Operation	A bounded normative operation that validates preconditions, consumes protocol inputs, produces an observable result and records continuity-relevant outcomes.
Observable Determinism	The requirement that semantically equivalent inputs, the same observable Protocol State and the same applicable profile constraints produce semantically equivalent observable results.
Semantic Equivalence	The property by which differently encoded or internally represented protocol objects carry the same normative meaning.
Deployment Profile	A constrained extension that selects transports, encodings, cryptographic suites or policy requirements without redefining core protocol semantics.
Logical Validator	A protocol role that validates an operation, recovery, identity or synchronization condition and MAY be implemented locally, distributively or by an authorized remote participant.

A conformant implementation and every Deployment Profile **MUST** preserve these meanings without reinterpretation.

3. Normative Language

The key words **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **NOT RECOMMENDED**, **MAY** and **OPTIONAL** are to be interpreted as described by BCP 14, comprising RFC 2119 and RFC 8174, when and only when they appear in all capitals [N1, N2].

3.1 Protocol Invariants

The following invariants apply to every conformant implementation and every Deployment Profile:

Invariant	Requirement
Evidence immutability	Valid Continuity Evidence MUST NOT be modified after issuance.
Append-only continuity	New activity MUST extend continuity history rather than rewrite it.
CEP preservation	A valid CEP MUST NOT be silently discarded, rewritten or semantically reinterpreted.
Observable determinism	Equivalent protocol inputs MUST produce equivalent observable results.

Invariant	Requirement
Cryptographic independence	Protocol semantics MUST NOT depend on a particular algorithm generation.
Application separation	Protocol results MUST remain distinct from final authorization or business decisions.
Failure containment	A failed operation MUST NOT corrupt previously valid continuity history.

3.2 Protocol Trust Boundary

QVTM governs protocol semantics, continuity semantics, Continuity Evidence semantics, CEP processing and observable interoperability behavior. It does not govern endpoint integrity, malware prevention, user intent, application authorization policy, operating-system security or physical protection of secret material.

A conformant implementation MUST enforce the boundary between protocol evaluation and application decision-making. Applications MAY use Trust Evaluation results, but MUST NOT reinterpret or rewrite the underlying Continuity Evidence.

3.3 Mandatory Enforcement and Failure Containment

Normative restrictions in this specification are enforceable requirements, not statements that safeguards are absent. A conformant implementation MUST detect and reject invalid state transitions, malformed messages, incompatible mandatory extensions, unauthorized synchronization and any operation that would violate a Protocol Invariant.

When such behavior is detected, the implementation MUST contain the failure by preserving previously valid Protocol State, Continuity Evidence and CEPs; producing a defined error result; and preventing partial or ambiguous continuity updates. Implementations SHOULD record continuity-relevant rejection evidence when required by the applicable Deployment Profile.

3.4 Safeguard Enforcement

Normative safeguards are part of conformant protocol behavior and SHOULD be enforced by the implementation rather than left to operator discipline. A conformant Protocol Core MUST reject operations that violate Protocol Invariants before committing state or continuity changes.

Implementations SHOULD enforce safeguards at three complementary layers:

Enforcement layer	Required behavior
Type and structure enforcement	Invalid or incomplete protocol objects are not representable as accepted objects.
Transition and operation enforcement	Invalid transitions, unauthorized operations and incompatible capabilities are rejected before commit.
Commit and persistence enforcement	Historical Continuity Evidence and valid CEPs cannot be overwritten through ordinary protocol operations.

Where an implementation language or storage system permits it, safeguards SHOULD make invariant-violating operations structurally unavailable rather than merely detect them after execution. Conformance tests MUST verify rejection behavior, failure containment and preservation of previously valid continuity information.

An implementation that bypasses these safeguards is non-conformant even when it emits syntactically valid messages.

3.5 Figure Conventions

This specification uses three figure types with distinct normative purposes:

Figure type	Purpose
Sequence Figure	Shows information exchange between protocol participants over time.
Workflow Figure	Shows transformation, validation or movement of protocol information.
Lifecycle Figure	Shows the temporal evolution of one protocol object or continuity history.

Sequence Figures identify the information carried by each message rather than wire-level headers. Workflow Figures distinguish transient information, persistent Continuity Evidence and derived evaluation results. Lifecycle Figures are used only where the same object or continuity history evolves over time.

Figure 1. QVTM end-to-end information workflow

sequenceDiagram

autonumber

participant A as Application or Deployment Profile

participant I as Initiating Protocol Core

participant R as Responding Protocol Core

participant K as Cryptographic Services

participant P as CEP Preservation

participant G as Correlation and Trust Evaluation

A->>I: Application data and requested Protocol Operation

I->>I: Validate Protocol State, profile and preconditions

opt Handshake required

 I->>R: Handshake request with operation context

 R-->>I: Handshake response with negotiated capabilities

end

I->>R: Protocol Message carrying operation payload

R->>R: Validate message semantics, state and authorization

alt Operation accepted

 R->>K: Protect continuity-relevant information

 K-->>R: Integrity, authenticity and confidentiality result

 R->>P: Commit immutable Continuity Evidence in CEP

opt Synchronization required

 P->>I: Exchange eligible CEP or evidence inventory

 I-->>P: Accepted, duplicate, conflicting or rejected set

end

P->>G: Provide validated Continuity Evidence

G-->>R: Trust Evaluation result

R-->>I: Observable protocol result

I-->>A: Application result and continuity context

A-->>I: Subsequent continuity-relevant event

else Operation rejected or failed

 R->>P: Preserve prior state and optional failure evidence

 R-->>I: Defined error result without partial commit

 I-->>A: Rejection or failure result

end

Note over A,R: Application data and Protocol Messages are transient interaction data

Note over P,G: Continuity Evidence and CEPs are persistent protocol information

Figure 1 is the principal Sequence Figure for this specification. It shows where a handshake MAY occur, what information crosses each participant boundary and when transient application data becomes persistent Continuity Evidence. Wire-level headers and serialization are intentionally omitted; Parts III through VIII define those requirements separately.

Figure 2. QVTM normative operation cycle

flowchart TD

PRE[Validate preconditions] --> OP[Execute Protocol Operation]

OP --> CE[Generate Continuity Evidence]

CE --> CEP[Commit or extend CEP]

CEP --> EVA[Evaluate continuity]

EVA --> RES[Return observable result]

RES --> EVT[Subsequent event]

EVT --> PRE

PRE --> |reject| ERR[Defined error result]

ERR -. preserves prior state .-> EVT

classDef phase fill:#eef6ff,stroke:#1f77b4,stroke-width:1.5px;

classDef evidence fill:#fff7e6,stroke:#c98200,stroke-width:1.2px;

classDef result fill:#eaf7ee,stroke:#2e7d32,stroke-width:2px;

classDef reject fill:#fdecec,stroke:#b3261e,stroke-width:1.5px;

class PRE,OP phase;

class CE,CEP evidence;

class EVA,RES,EVT result;

class ERR reject;

Every Protocol Operation MUST determine whether its observable result is continuity-relevant. A continuity-relevant result MUST generate at least one Continuity Evidence object. A non-continuity-relevant result MUST NOT generate artificial Continuity Evidence solely to satisfy protocol accounting. A failed Protocol Operation that changes observable continuity risk SHOULD generate failure evidence. As an Operational Figure, Figure 2 provides the normative cycle referenced by later operation-specific sections.

4. Protocol Architecture

4.1 Architectural Components

The information exchange specified in Figure 1 is realized through the conceptual components below. Every compliant implementation SHALL implement the following conceptual components.

Component	Responsibility
Continuity Evidence	Capture continuity-relevant information.
CEP	Preserve and transport evidence.
Continuity Graph	Represent continuity relationships.
Graph Puzzle	Correlate preserved evidence.
Trust Evaluation	Produce continuity evaluation results.

Alternative internal implementations are permitted provided externally observable behavior remains compliant.

4.2 Architectural Separation

Implementations SHALL maintain explicit separation among Cryptographic Services, protocol semantics, deployment logic and application logic. Application-specific behavior MUST NOT modify protocol semantics, and deployment policy MUST NOT be interpreted as Continuity Evidence unless a normative protocol operation records it as such.

4.3 Deployment Profile Contract

A Deployment Profile MAY select or constrain transport protocols, canonical encodings, cryptographic suites, authentication methods, recovery policies, synchronization policies and optional extensions.

A Deployment Profile MUST NOT redefine Trust Continuity, Continuity Evidence, CEP semantics, Protocol Invariants, core Message Semantics or the meaning of a normative result. A profile that violates this restriction is non-conformant.

4.4 Cryptographic Agility Requirement

Implementations MUST NOT embed assumptions about a specific cryptographic algorithm, parameter set or standard generation into protocol semantics. Algorithm identifiers and cryptographic material MUST be carried through profile-defined cryptographic contexts so that algorithm migration can occur without rewriting historical Continuity Evidence.

5. Protocol Entities

5.1 Protocol Participants

The protocol recognizes the following conceptual entities.

Entity	Responsibility
Evidence Producer	Generates Continuity Evidence.
CEP Producer	Packages evidence into CEPs.
Evidence Consumer	Processes received CEPs.
Trust Evaluator	Evaluates Trust Continuity.
Application	Consumes continuity evaluation results.

Individual implementations MAY combine multiple conceptual entities.

5.2 Responsibilities

Evidence Producers MUST generate immutable Continuity Evidence.

CEP Producers MUST preserve evidence integrity.

Trust Evaluators MUST evaluate preserved evidence without modifying historical continuity information.

Applications MUST treat Trust Evaluation results as advisory protocol outputs.

Final authorization decisions remain outside the scope of this specification.

5.3 Protocol Operation Contract

Every Protocol Operation MUST define its purpose, preconditions, consumed protocol objects, produced protocol objects, permitted state transitions, success result and failure result. Enrollment, Authentication, Synchronization, Recovery, Trust Evaluation and Version Negotiation are Protocol Operations.

Operation	Required observable outputs
Enrollment	Enrollment result, resulting Protocol State, initial Continuity Evidence and at least one CEP.

Operation	Required observable outputs
Authentication	Authentication result and continuity-relevant success or failure evidence.
Synchronization	Synchronization result, accepted or rejected evidence set and resulting correlation status.
Recovery	Recovery result, resulting Protocol State and recovery Continuity Evidence.
Trust Evaluation	Evaluation result, evaluated context and referenced continuity inputs.
Version Negotiation	Selected version/profile set or a defined incompatibility result.

Part II — Protocol State Semantics

6. Protocol States

6.1 Overview

Every QVTM protocol instance SHALL occupy exactly one Protocol State at any given time.

Protocol States describe the observable execution state of the protocol and SHALL NOT be interpreted as application states.

6.2 Defined States

The protocol defines the following normative states.

State	Description
Initialized	Protocol context exists but no operational continuity has been established.
Enrolled	Initial continuity establishment has completed successfully.
Operational	Normal protocol operation.
Synchronizing	Continuity Evidence is being exchanged or reconciled.
Recovering	Continuity is being re-established after a disruption.
Suspended	Normal protocol execution is temporarily unavailable.

Archived	Operational activity has ended while preserving historical continuity.

Implementations MAY introduce internal states provided they do not alter externally observable protocol behavior.

6.3 State Constraints

A protocol instance **MUST** begin in **Initialized** and **MUST** reach **Enrolled** before entering **Operational**. It **MUST NOT** transition directly from **Initialized** to **Operational**. Every state transition **MUST** preserve existing Continuity Evidence and the normative protocol semantics, independent of the selected Deployment Profile.

7. State Transitions

7.1 General Rules

A transition is valid only when the current Protocol State permits it, all specified preconditions are satisfied, required Continuity Evidence remains preserved and the observable result satisfies Observable Determinism.

7.2 Valid Transitions

From	To
Initialized	Enrolled
Enrolled	Operational
Operational	Synchronizing
Operational	Recovering
Operational	Suspended
Synchronizing	Operational
Recovering	Operational
Suspended	Operational

Operational	Archived
Recovering	Archived

All other transitions SHALL be considered invalid unless defined by a future deployment profile.

7.3 Invalid Transitions

Implementations MUST reject:

- Archived Operational
- Archived Synchronizing
- Initialized Archived
- Suspended Initialized

Rejected transitions MUST NOT modify preserved Continuity Evidence.

8. Session Execution

8.1 Purpose

Figure 3 refines the bounded interaction shown in Figure 2. A Session is a bounded execution context in which participants negotiate, exchange Protocol Messages and emit Continuity Evidence.

Sessions SHALL NOT modify historical Continuity Evidence.

Instead, sessions SHALL generate additional Continuity Evidence describing protocol activity.

8.2 Execution Sequence

Figure 3. Session execution and evidence emission

```

sequenceDiagram
    autonumber
    participant I as Initiating Participant
    participant R as Responding Participant
    participant C as Protocol Core
    participant P as CEP Preservation

    I->>R: Session request with profile and capability data
    R-->>I: Negotiated capabilities and session context
    I->>C: Protocol Message with operation payload
    C->>C: Validate message, state and operation semantics
    C->>P: Commit negotiation or operation evidence

    alt Normal completion
        C-->>I: Observable session result
        C->>P: Commit completion evidence
    else Termination or failure
        C-->>I: Defined error result
        C->>P: Preserve prior state and optional failure evidence
    end

    Note over I,C: Session and message data are transient
    Note over P: Continuity Evidence and CEPs persist beyond session closure

```

A Session MAY emit Continuity Evidence during negotiation, operation and completion. Completion closes the bounded interaction and makes its evidence eligible for preservation, synchronization and later correlation; it does not consolidate or rewrite historical evidence.

8.3 Session Guarantees

Every completed Session SHALL:

- preserve generated Continuity Evidence;
- preserve chronological ordering;
- preserve protocol integrity;
- produce deterministic observable results.

9. Continuity Lifecycle

9.1 Purpose

Figure 4 extends the information workflow in Figure 1 across successive Protocol Operations. The Continuity Lifecycle defines how Trust Continuity evolves independently from Session execution.

Multiple Sessions MAY contribute to the same continuity history.

9.2 Lifecycle

Figure 4. Continuity lifecycle across successive Protocol Operations

flowchart TD

EVT[Continuity-relevant event] --> OP[Protocol Operation]

OP --> CE[Continuity Evidence]

CE --> CEP[CEP preservation or exchange]

CEP --> COR[Correlation]

COR --> EVAL[Trust Evaluation]

EVAL --> RES[Observable protocol result]

RES --> NEXT[Subsequent event]

NEXT --> OP

```
classDef event fill:#f4f4f4,stroke:#666,stroke-dasharray:4 3;
classDef operation fill:#eef6ff,stroke:#1f77b4,stroke-width:1.5px;
classDef evidence fill:#fff7e6,stroke:#c98200,stroke-width:1.2px;
classDef correlation fill:#f3e8ff,stroke:#7b2cbf,stroke-width:1.8px;
classDef result fill:#eaf7ee,stroke:#2e7d32,stroke-width:2px;
class EVT,NEXT event;
class OP operation;
class CE,CEP evidence;
class COR,EVAL correlation;
class RES result;
```

The lifecycle is intentionally cyclic.

Trust Continuity SHALL be preserved through successive protocol operations rather than being re-established for every Session.

9.3 Continuity Guarantees

The protocol SHALL ensure:

- Continuity Evidence remains immutable.
- Historical continuity information is never overwritten.
- New protocol activity extends existing continuity.
- Independent verification remains possible throughout the lifecycle.
- Cryptographic migration SHALL NOT invalidate preserved continuity.

9.4 Relationship Between Sessions and Continuity

Sessions represent operational interactions.

Trust Continuity represents architectural evolution.

A single Continuity History MAY include multiple independent Sessions.

Likewise, a Session MAY contribute Continuity Evidence to multiple continuity relationships.

Implementations MUST preserve this separation.

Part III — Handshake Specification

10. Enrollment Handshake

10.1 Purpose

The Enrollment Handshake establishes the initial Trust Continuity relationship between protocol participants.

Successful completion SHALL transition the protocol instance from Initialized to Enrolled.

10.2 Preconditions

Before initiating enrollment:

- both participants MUST support a compatible protocol version;
- required cryptographic capabilities MUST be available;
- local protocol context MUST be initialized;
- no active enrollment Session SHALL already exist.

10.3 Handshake Sequence

Figure 5. Enrollment handshake

```
sequenceDiagram
    participant I as Initiator
    participant R as Responder
    I->>R: Enrollment Request
    R-->>I: Enrollment Response
    I->>R: Continuity Initialization
    R-->>I: Enrollment Complete
```

10.4 Postconditions

Successful completion SHALL:

- establish the initial continuity context;
- generate Continuity Evidence;
- generate one or more CEPs;
- preserve protocol integrity.

10.5 Failure Conditions

Enrollment SHALL fail if:

- protocol versions are incompatible;
- mandatory capabilities are unavailable;
- required evidence cannot be generated;
- protocol integrity cannot be established.

Failure SHALL NOT generate partial continuity history.

11. Authentication Handshake

11.1 Purpose

The Authentication Handshake verifies an existing continuity relationship.

It SHALL extend previously established continuity rather than creating a new identity relationship.

11.2 Preconditions

The protocol instance **MUST** have completed Enrollment and **MUST** be in a Protocol State that permits Authentication, normally **Operational**. It **MUST** possess valid Continuity Evidence and a valid protocol context.

The **Enrolled** state records successful completion of initial continuity establishment and **MAY** be transient before entry into **Operational**. Authentication therefore depends on the completed Enrollment history, not on remaining indefinitely in the **Enrolled** state.

11.3 Handshake Sequence

Figure 6. Authentication handshake

```
sequenceDiagram
    participant Q as Requester
    participant V as Verifier
    Q->>V: Authentication Request
    V-->>Q: Continuity Challenge
    Q->>V: Challenge Response
    V-->>Q: Authentication Result
```

11.4 Guarantees

Successful authentication **SHALL**:

- preserve Trust Continuity;
- generate new Continuity Evidence;
- preserve historical continuity;
- avoid modifying previous evidence.

11.5 Failure

Authentication failure **SHALL**:

- preserve historical Continuity Evidence;
- preserve existing CEPs;
- generate failure evidence when required by deployment policy.

12. Synchronization Handshake

12.1 Purpose

The Synchronization Handshake exchanges Continuity Evidence between protocol participants.

Synchronization SHALL exchange evidence rather than protocol state.

12.2 Preconditions

Participants SHALL possess:

- valid protocol context;
- compatible protocol versions;
- synchronization authorization.

12.3 Handshake Sequence

Figure 7. Synchronization handshake

```
sequenceDiagram
    participant A as Node A
    participant B as Node B
    A->>B: Synchronization Request
    B-->>A: Capabilities and evidence summary
    A->>B: Evidence Transfer
    B-->>A: Evidence Transfer / acknowledgement
    A->>B: Synchronization Complete
```

12.4 Guarantees

A successful Synchronization operation SHALL:

- preserve evidence immutability;
- avoid destructive updates;
- preserve chronological consistency;
- preserve independent verification.

12.5 Failure

Synchronization failure SHALL NOT:

- invalidate previously preserved evidence;
- delete CEPs;
- modify historical continuity.

13. Recovery Handshake

13.1 Purpose

The Recovery Handshake restores operational capability while preserving Trust Continuity.

Recovery SHALL extend continuity rather than restart protocol history.

13.2 Preconditions

Recovery SHALL require:

- recoverable continuity history;
- sufficient Continuity Evidence;
- successful verification of recovery conditions.

13.3 Handshake Sequence

Figure 8. Recovery handshake

```
sequenceDiagram
    participant N as Recovering Node
    participant A as Recovery Authority
    N->>A: Recovery Request
    A-->>N: Recovery Challenge
    N->>A: Recovery Evidence
    A-->>N: Recovery Result
```

The term **Recovery Authority** denotes a Logical Validator for recovery under the selected Deployment Profile. It does not imply a centralized service and MAY be implemented locally, distributively or by an authorized remote participant.

13.4 Guarantees

A successful Recovery operation SHALL:

- preserve historical Trust Continuity;
- generate new Continuity Evidence;
- extend the Continuity Graph;
- preserve protocol interoperability.

13.5 Failure

Failed recovery SHALL:

- preserve historical evidence;
- preserve previous CEPs;
- avoid discontinuity unless explicitly determined by Trust Evaluation.

13.6 Common Handshake Requirements

All protocol handshakes:

- MUST be deterministic from the perspective of observable protocol behavior.
- MUST preserve previously established Continuity Evidence.
- MUST apply the continuity-relevance rule defined in Section 4 and generate Continuity Evidence only for continuity-relevant outcomes.
- MUST remain independent of specific cryptographic algorithms.
- MUST support future protocol extensions through Deployment Profiles without altering the normative semantics defined by this specification.

Part IV — Message Specification

14. Message Model

14.1 Purpose

Every interaction between QVTM protocol participants SHALL be performed through Protocol Messages. Figure 1 places those exchanges within the complete information workflow, while Figures 5 through 8 specify the operation-specific handshakes.

A Protocol Message represents a single, immutable protocol transaction exchanged between two or more protocol entities.

Protocol Messages SHALL be interpreted according to their semantic type rather than their serialized representation.

14.2 General Properties

Every Protocol Message SHALL:

- identify exactly one Message Type;
- contain sufficient metadata for processing;
- preserve protocol version information;
- be suitable for independent verification;
- remain immutable after transmission.

14.3 Message Lifecycle

Figure 9. Protocol Message lifecycle

flowchart TD

C[Create immutable Protocol Message] --> T[Transmit or store]

T --> R[Receive]

R --> V{Validate syntax, integrity and state}

V --> |valid| P[Process semantic type]

V --> |invalid| X[Reject without state change]

P --> O[Produce observable result]

P -. continuity-relevant outcome .-> E[Generate Continuity Evidence]

E --> B[Bind into CEP]

O --> A[Archive processing disposition]

B --> A

classDef transient fill:#eef6ff,stroke:#1f77b4,stroke-width:1.5px;

classDef decision fill:#f3e8ff,stroke:#7b2cbf,stroke-width:1.8px;

classDef persistent fill:#fff7e6,stroke:#c98200,stroke-width:1.4px;

classDef result fill:#eaf7ee,stroke:#2e7d32,stroke-width:2px;

classDef reject fill:#fdecec,stroke:#b3261e,stroke-width:1.5px;

class C,T,R,P transient;

class V decision;

class E,B persistent;

class O,A result;

class X reject;

As shown in Figure 9, a Protocol Message MUST remain immutable after transmission. Processing MUST create new protocol objects rather than modify the received message.

14.4 Processing Requirements

Implementations MUST:

- process messages deterministically;
- reject malformed messages;
- preserve received messages until processing completes;
- avoid modifying message semantics.

15. Message Types

15.1 Overview

The protocol defines the following conceptual Message Types.

Message Type	Purpose
Enrollment Request	Initiates enrollment.
Enrollment Response	Responds to enrollment.
Authentication Request	Requests continuity verification.
Authentication Response	Returns authentication result.
Synchronization Request	Initiates evidence synchronization.
Synchronization Response	Returns synchronization capabilities.
Evidence Transfer	Exchanges Continuity Evidence.
Recovery Request	Initiates recovery.
Recovery Response	Returns recovery result.
Error Message	Reports protocol failure.

Deployment Profiles MAY define additional Message Types.

15.2 Message Identification

Every Protocol Message SHALL contain:

- Message Type;
- Protocol Version;
- Message Identifier;
- Session Identifier (when applicable).

Message Identifiers SHALL be unique within the associated Session.

15.3 Message Ordering

The protocol SHALL NOT require global message ordering.

Ordering requirements apply only within the scope of a single protocol Session unless otherwise specified by a Deployment Profile.

16. Message Semantics

16.1 Processing Rules

Figure 10. Message processing semantics

flowchart TD

```
M[Received Protocol Message] --> S[Resolve semantic type]
S --> V{Validate version, integrity, completeness and state}
V --> |invalid| ERR[Return defined error result]
V --> |valid| P[Select permitted Protocol Operation]
P --> X[Execute permitted Protocol Operation]
X --> R[Produce protocol result]
X -. if continuity-relevant .-> CE[Generate Continuity Evidence]
CE --> CEP[Commit CEP]
CEP --> R
ERR -. preserve prior state .-> END[Processing complete]
R --> END
```

```
classDef transient fill:#eef6ff,stroke:#1f77b4,stroke-width:1.5px;
classDef decision fill:#f3e8ff,stroke:#7b2cbf,stroke-width:1.8px;
classDef persistent fill:#fff7e6,stroke:#c98200,stroke-width:1.4px;
classDef result fill:#eaf7ee,stroke:#2e7d32,stroke-width:2px;
classDef reject fill:#fdecec,stroke:#b3261e,stroke-width:1.5px;
class M,S,P,X transient;
class V decision;
class CE,CEP persistent;
class R,END result;
class ERR reject;
```

16.2 Validation

Before execution, implementations MUST validate:

- protocol version;
- message integrity;
- message completeness;
- semantic correctness;
- state compatibility.

Messages failing validation SHALL NOT modify protocol state.

16.3 Idempotency

Whenever possible, Protocol Messages SHOULD be idempotent.

Repeated processing of the same Message Identifier MUST NOT generate duplicate Continuity Evidence.

16.4 Observable Behavior

Protocol behavior SHALL depend exclusively upon:

- message semantics;
- preserved Continuity Evidence;
- current Protocol State;
- applicable Deployment Profile.

Internal implementation details SHALL NOT affect externally observable behavior.

17. Error Messages

17.1 Purpose

Error Messages communicate protocol failures while preserving protocol consistency.

Errors SHALL describe observable protocol conditions rather than implementation-specific failures.

17.2 Error Categories

Category	Description
Invalid Message	Message syntax or semantics are invalid.
Invalid State	Current Protocol State does not permit the requested operation.
Version Mismatch	Protocol versions are incompatible.
Authentication Failure	Authentication Handshake did not complete successfully.
Synchronization Failure	Synchronization could not complete.
Recovery Failure	Recovery requirements were not satisfied.
Internal Failure	An implementation-defined failure occurred without exposing implementation details.

17.3 Error Processing

Upon generating an Error Message, an implementation SHALL:

- preserve previously established Continuity Evidence;
- preserve Protocol State unless explicitly modified by this specification;
- preserve protocol interoperability.

Error handling SHALL NOT invalidate previously established Trust Continuity unless explicitly determined through Trust Evaluation.

17.4 Error Reporting

Error Messages SHOULD contain:

- Error Category;
- Error Identifier;
- associated Message Identifier (if applicable);
- protocol version;
- optional implementation-defined diagnostic information.

Diagnostic information MUST NOT reveal confidential implementation details.

17.5 Recovery from Errors

Following successful handling of an Error Message, implementations MAY:

- retry the operation;
- terminate the current Session;
- initiate Recovery;
- remain Operational.

The selected behavior SHALL depend upon the current Protocol State and the applicable Deployment Profile.

Part V — Data Structures

18. Continuity Evidence

18.1 Purpose

Continuity Evidence is the fundamental protocol object representing independently verifiable information contributing to Trust Continuity.

Every Continuity Evidence object SHALL be immutable after creation.

18.2 Responsibilities

Continuity Evidence SHALL:

- represent a single continuity-relevant event;
- remain independently verifiable;
- support long-term preservation;
- remain suitable for CEP encapsulation.

Continuity Evidence SHALL NOT contain implementation-specific state.

18.3 Required Properties

Every Continuity Evidence object SHALL provide:

Property	Requirement
Identifier	Globally unique within the Deployment Profile.
Type	Identifies the semantic category of the evidence.
Creation Context	Describes the protocol context in which the evidence was produced.
Integrity Information	Allows independent integrity verification.
Version	Indicates the semantic version of the evidence structure.

Additional implementation-specific fields MAY be present only when they do not alter the normative meaning of required properties. Implementations MUST be able to determine whether two Continuity Evidence objects are semantically equivalent even when their internal representation differs.

18.4 Lifecycle

Figure 11. Continuity Evidence lifecycle and reuse

flowchart TD

EVT[Continuity-relevant event] --> I[Issue Continuity Evidence]

I --> V{Validate semantics and integrity}

V --> |invalid| X[Reject without commit]

V --> |valid| B[Bind immutable evidence into CEP]

B --> L[Local preservation]

B --> S[Exchange or synchronization]

L --> C[Correlation]

S --> C

C --> E[Trust Evaluation]

E --> REF[Reference from later Protocol Operations]

REF -. references same immutable object .-> L

REF -. may trigger new event .-> EVT

```
classDef event fill:#f4f4f4,stroke:#666,stroke-dasharray:4 3;
classDef action fill:#eef6ff,stroke:#1f77b4,stroke-width:1.5px;
classDef decision fill:#f3e8ff,stroke:#7b2cbf,stroke-width:1.8px;
classDef persistent fill:#fff7e6,stroke:#c98200,stroke-width:1.4px;
classDef result fill:#eaf7ee,stroke:#2e7d32,stroke-width:2px;
classDef reject fill:#fdecec,stroke:#b3261e,stroke-width:1.5px;

class EVT event;
class I action;
class V decision;

class B,L,S persistent;
class C,E,REF result;
class X reject;
```

As shown in Figure 11, Continuity Evidence remains immutable throughout this lifecycle. Later Protocol Operations MAY reference preserved evidence, but MUST NOT alter or reissue it as if it were new evidence. Such references MUST use a stable protocol identity defined by the applicable canonical representation; a Deployment Profile MAY realize that identity through an object identifier, a canonical digest, a semantic relationship identifier or another semantically equivalent mechanism.

19. Continuity Evidence Package (CEP)

19.1 Purpose

The Continuity Evidence Package (CEP) is the canonical protocol container used to preserve and exchange Continuity Evidence.

The CEP defines semantic structure only.

Serialization is defined in Part VIII.

19.2 Responsibilities

Every CEP SHALL:

- encapsulate one or more Continuity Evidence objects;
- preserve evidence integrity;
- preserve semantic interoperability;
- support future protocol evolution.

19.3 Logical Structure

Every CEP SHALL contain the following conceptual sections.

Section	Responsibility
Header	Protocol identification.
Metadata	Contextual protocol information.
Evidence Collection	One or more Continuity Evidence objects.
Integrity Information	Verification material.
Extensions	Deployment-specific extensions.

Implementations MAY optimize storage provided observable protocol semantics remain unchanged.

19.4 CEP Guarantees

A CEP SHALL guarantee the following:

Guarantee	Normative requirement
Immutability	The CEP MUST NOT be modified after issuance.
Deterministic interpretation	Equivalent CEPs MUST produce equivalent protocol semantics.
Version awareness	The CEP MUST identify the protocol version governing interpretation.
Profile context	The CEP MUST identify its applicable Deployment Profile without allowing that profile to redefine core semantics.
Integrity verification	The CEP MUST carry or reference sufficient integrity information for independent verification.

20. Metadata

20.1 Purpose

Metadata provides contextual information required for correct protocol interpretation.

Metadata SHALL NOT modify the semantic meaning of Continuity Evidence.

20.2 Required Metadata

Every CEP SHALL include sufficient metadata to determine:

- protocol version;
- deployment profile;
- protocol context;
- evidence relationships;
- integrity context.

20.3 Metadata Evolution

Future protocol versions MAY introduce additional metadata fields.

Unknown metadata SHALL be ignored unless explicitly defined as mandatory by the applicable protocol version.

20.4 Compatibility

Metadata extensions SHALL preserve backward compatibility whenever possible.

Mandatory metadata changes SHALL require a new protocol version.

21. Extension Mechanism

21.1 Purpose

The Extension Mechanism allows protocol evolution without modifying existing normative semantics.

Extensions SHALL complement existing protocol behavior.

They SHALL NOT redefine it.

21.2 Extension Categories

Extensions MAY introduce:

- additional metadata;
- additional Message Types;
- Deployment Profile capabilities;
- implementation-specific optimizations.

21.3 Compatibility Rules

Every extension SHALL:

- preserve protocol interoperability;
- preserve Trust Continuity semantics;
- preserve CEP compatibility;
- preserve deterministic observable behavior.

Extensions violating these requirements SHALL be considered non-compliant. A conformant implementation MUST reject such an extension before it modifies Protocol State or continuity history.

21.4 Extension Enforcement

An extension or Deployment Profile **MUST NOT** rewrite historical Continuity Evidence, silently discard valid CEPs, weaken mandatory integrity checks, reinterpret a previously issued Evidence Type, convert an application decision into a protocol result, or make protocol semantics dependent on an undisclosed implementation state.

Implementations **MUST** enforce these restrictions during extension negotiation and processing. Detection **MUST** result in a defined incompatibility or invalid-extension error and **MUST** preserve all previously valid continuity information.

21.5 Unknown Extensions

Implementations receiving unknown extensions:

- **MUST** ignore optional extensions;
- **MUST** reject mandatory extensions they cannot process;
- **MUST** preserve existing Continuity Evidence.

Unknown extensions **SHALL NOT** invalidate previously established protocol continuity.

21.6 Version Independence

The Extension Mechanism **SHALL** permit protocol evolution while minimizing incompatibilities between successive protocol versions.

Implementations **SHOULD** prefer extensions over incompatible protocol modifications whenever practical.

Part VI — Synchronization

22. Synchronization Protocol

22.1 Purpose

The Synchronization Protocol defines the normative behavior required to exchange Continuity Evidence between protocol participants.

Synchronization **SHALL** exchange Continuity Evidence and CEPs.

Synchronization **SHALL NOT** synchronize implementation state.

22.2 Objectives

The Synchronization Protocol SHALL:

- preserve Trust Continuity;
- preserve evidence immutability;
- support disconnected operation;
- tolerate partial synchronization;
- preserve independent verification.

22.3 Synchronization Preconditions

Before synchronization begins:

- protocol versions **MUST** be compatible;
- both participants **MUST** possess valid protocol context;
- synchronization authorization **MUST** have been established;
- protocol instances **MUST NOT** be Archived.

22.4 Synchronization Sequence

Figure 12. Synchronization sequence between independent participants

```
sequenceDiagram
    participant A as Participant A
    participant B as Participant B
    participant L as Local Validators

    A->>B: Synchronization Request
    B-->>A: Compatible capabilities and profile context
    A->>B: Evidence inventory or discovery request
    B-->>A: Missing CEP identifiers
    A->>B: CEP transfer
    B->>L: Validate semantics, integrity and duplication
    L-->>B: Accepted, duplicate, conflicting or rejected set
    B-->>A: Synchronization Result
    Note over A,B: Previously valid evidence remains unchanged
```

Capability negotiation, evidence discovery, exchange and validation are logically ordered. The discovery exchange **MAY** use identifiers, canonical digests, summaries, ranges or another semantically equivalent mechanism defined by the Deployment Profile. An implementation **MAY**

pipeline or batch these phases only when the resulting observable behavior is semantically equivalent and no unvalidated evidence is committed.

22.5 Synchronization Guarantees

Successful synchronization SHALL:

- preserve all previously established Continuity Evidence;
- preserve CEP integrity;
- preserve chronological consistency;
- preserve protocol interoperability.

23. Conflict Resolution

23.1 Purpose

Conflict Resolution specifies the observable protocol behavior when multiple synchronization participants possess different continuity histories.

This specification intentionally does not define proprietary conflict-resolution algorithms.

23.2 Normative Requirements

Conflict Resolution SHALL:

- preserve all valid Continuity Evidence;
- avoid destructive synchronization;
- avoid implicit evidence replacement;
- preserve independent verification.

23.3 Conflicting Evidence

When conflicting evidence is detected, implementations SHALL:

- preserve all conflicting Continuity Evidence;
- generate additional Continuity Evidence describing the observed conflict when required by the Deployment Profile;
- defer continuity interpretation to Trust Evaluation.

Conflict Resolution SHALL NOT determine trust.

When two participants encode the same normative evidence differently, they **MUST** compare semantic identifiers and canonicalized semantics rather than raw byte equality. Semantically equivalent evidence **MUST** be treated as duplicate evidence, not conflicting evidence.

23.4 Duplicate Evidence

Duplicate Continuity Evidence **SHALL NOT** generate duplicate protocol history.

Implementations **MUST** recognize previously processed Evidence Identifiers.

23.5 Missing Evidence

Missing Continuity Evidence **SHALL NOT** invalidate previously preserved continuity.

Implementations **MAY** request additional synchronization or initiate Recovery according to Deployment Profile policy.

24. Offline Synchronization

24.1 Purpose

Offline Synchronization enables protocol participants to continue generating Continuity Evidence while disconnected.

Connectivity **SHALL NOT** be required for local continuity preservation.

24.2 Offline Operation

During offline operation implementations **SHALL**:

- continue generating Continuity Evidence;
- continue generating CEPs;
- preserve chronological ordering;
- preserve protocol semantics.

24.3 Reconnection

Following reconnection:

- locally generated CEPs **SHALL** become eligible for synchronization;

- historical Continuity Evidence SHALL remain unchanged;
- synchronization SHALL extend existing continuity.

Offline operation SHALL NOT restart protocol history.

24.4 Synchronization Responsibilities

Each participant SHALL remain responsible for preserving its locally generated Continuity Evidence until successful synchronization has completed.

No participant SHALL assume that another participant has preserved equivalent evidence.

24.5 Synchronization Independence

The Synchronization Protocol SHALL remain independent from:

- transport protocols;
- storage technologies;
- deployment architectures;
- communication media.

Only observable synchronization semantics are normative.

24.6 Common Synchronization Requirements

Every conformant implementation MUST preserve immutable Continuity Evidence and previously established CEPs during synchronization. It MUST reject destructive synchronization and MUST preserve Observable Determinism. Support for future Deployment Profile extensions SHOULD be provided without changing the normative synchronization semantics defined by this specification.

Part VII — Recovery

25. Recovery Protocol

25.1 Purpose

The Recovery Protocol defines the normative behavior required to restore operational capability while preserving Trust Continuity.

Recovery SHALL extend an existing continuity history.

Recovery SHALL NOT create a new continuity history unless explicitly defined by the applicable Deployment Profile.

25.2 Recovery Objectives

The Recovery Protocol SHALL:

- preserve previously established Trust Continuity;
- preserve immutable Continuity Evidence;
- support recovery after operational disruption;
- support continuity-preserving migration;
- remain independent from implementation-specific recovery mechanisms.

25.3 Recovery Preconditions

Recovery SHALL require:

- recoverable Continuity Evidence;
- valid protocol context;
- successful verification according to the active Deployment Profile;
- compatibility with the current protocol version.

Recovery SHALL fail if these conditions cannot be satisfied.

25.4 Recovery Sequence

The **Recovery Validator**, **Identity Validator** and equivalent validation participants shown in Figures 13 through 15 are Logical Validators. Their placement in a sequence figure denotes responsibility, not mandatory centralization or network location.

Figure 13. Recovery sequence

```

sequenceDiagram
    autonumber
    participant N as Recovering Participant
    participant V as Recovery Validator
    participant C as Protocol Core
    participant P as CEP Preservation

    N->>V: Recovery request with recovery context
    V->>C: Validate profile, protocol state and evidence requirements
    C->>P: Retrieve preserved recovery evidence
    P-->>C: Eligible CEPs and Continuity Evidence
    C->>C: Perform Trust Evaluation

    alt Recovery accepted
        C-->>V: Continuity extension authorized
        V-->>N: Recovery result and new operational context
        C->>P: Commit recovery Continuity Evidence
    else Recovery rejected
        C-->>V: Defined recovery failure
        V-->>N: Failure result
        C->>P: Preserve prior state and optional failure evidence
    end
end

```

25.5 Recovery Guarantees

A successful Recovery operation SHALL:

- preserve historical Continuity Evidence;
- generate additional Continuity Evidence describing the recovery event;
- preserve CEP interoperability;
- preserve independent verification.

26. Device Replacement

26.1 Purpose

Device Replacement defines the observable protocol behavior when operational capability is transferred to a different computing device.

The protocol specifies continuity semantics only.

Device provisioning remains outside the scope of this specification.

26.2 Normative Requirements

Device replacement SHALL:

- preserve Trust Continuity;
- preserve historical Continuity Evidence;
- generate Recovery Evidence;
- preserve protocol interoperability.

The replacement device SHALL NOT inherit protocol state without successful Recovery.

26.3 Replacement Sequence

Figure 14. Device replacement sequence

```
sequenceDiagram
    autonumber
    participant O as Previous Device Context
    participant N as Replacement Device
    participant V as Recovery Validator
    participant P as CEP Preservation

    N->>V: Device replacement request
    V->>P: Retrieve eligible continuity and recovery evidence
    P-->>V: Preserved CEP set
    V->>V: Validate device transition and Trust Continuity

    alt Replacement authorized
        V-->>N: Authorized operational context
        N->>P: Commit device-replacement Continuity Evidence
        Note over O,N: Protocol state is not copied directly from the previous device
    else Replacement rejected
        V-->>N: Defined failure result
        V->>P: Preserve prior state and optional failure evidence
    end
end
```

26.4 Responsibilities

The replacing participant SHALL:

- generate Recovery Evidence;
- preserve received CEPs;
- preserve protocol semantics.

Historical Continuity Evidence SHALL remain immutable.

26.5 Failure

Failed device replacement SHALL:

- preserve existing continuity history;
- preserve previous CEPs;
- avoid generating partial continuity.

27. Identity Restoration

27.1 Purpose

Identity Restoration defines the protocol behavior required to re-establish operational identity while preserving Trust Continuity.

Identity Restoration SHALL extend continuity.

Identity Restoration SHALL NOT redefine identity semantics.

27.2 Identity Continuity

Successful Identity Restoration SHALL:

- preserve historical Trust Continuity;
- generate Identity Restoration Evidence;
- preserve continuity relationships;
- preserve protocol interoperability.

27.3 Restoration Preconditions

Identity Restoration SHALL require:

- sufficient Continuity Evidence;
- successful Trust Evaluation;

- successful validation according to the applicable Deployment Profile.

27.4 Restoration Sequence

Figure 15. Identity restoration sequence

```
sequenceDiagram
    autonumber
    participant N as Restoring Participant
    participant I as Identity Validator
    participant C as Protocol Core
    participant P as CEP Preservation

    N->>I: Identity restoration request and evidence
    I->>C: Validate identity evidence and applicable profile
    C->>P: Retrieve preserved identity continuity evidence
    P-->>C: Eligible CEPs
    C->>C: Evaluate Identity Continuity

    alt Restoration accepted
        C-->>I: Identity continuity verified
        I-->>N: Restored operational identity context
        C->>P: Commit Identity Restoration Evidence
    else Restoration rejected
        C-->>I: Defined restoration failure
        I-->>N: Failure result
        C->>P: Preserve prior state and optional failure evidence
    end
end
```

27.5 Recovery Evidence

Every successful Identity Restoration SHALL generate Continuity Evidence describing:

- restoration context;
- restoration completion;
- protocol version;
- applicable Deployment Profile.

The generated evidence SHALL become eligible for future synchronization.

27.6 Recovery Independence

The Recovery Protocol SHALL remain independent from:

- authentication technologies;
- identity providers;
- hardware devices;
- operating systems;
- cryptographic algorithms.

Only observable protocol behavior is normative.

27.7 Common Recovery Requirements

Every conformant implementation MUST preserve historical Continuity Evidence and immutable CEPs during recovery. Recovery MUST extend existing continuity rather than replace it and MUST preserve Observable Determinism. Future Deployment Profiles SHOULD be supported without changing the normative recovery semantics defined by this specification.

Part VIII — Wire Specification

28. Serialization Rules

28.1 Purpose

Serialization defines how protocol objects are represented for storage or transmission.

This specification defines canonical representation requirements; the selected encoding remains a Deployment Profile decision.

Individual Deployment Profiles MAY define specific serialization formats provided they preserve the canonical semantics defined herein.

28.2 Serialization Requirements

Every serialized protocol object SHALL:

- preserve semantic equivalence;
- preserve object boundaries;
- preserve protocol version information;

- preserve deterministic interpretation.

Serialization SHALL NOT alter protocol behavior.

28.3 Canonical Semantic Representation

All conformant implementations MUST map logically equivalent protocol objects to semantically equivalent canonical representations. Semantic Equivalence applies across different encodings and internal representations; it does not require identical bytes across different encoding technologies.

After deserialization, semantically equivalent canonical representations MUST produce identical normative meaning and equivalent observable protocol behavior.

28.4 Object Independence

Serialization SHALL preserve the independence of:

- Continuity Evidence;
- CEPs;
- Protocol Messages;
- Metadata.

No serialized object SHALL require implementation-specific internal state for interpretation.

29. Canonical Encoding

29.1 Purpose

Canonical Encoding defines byte-level requirements for protocol objects that are hashed, signed, integrity-protected or compared by canonical digest.

The core protocol remains independent of any one encoding technology. Each interoperable Deployment Profile that protects protocol objects cryptographically MUST select exactly one canonical byte encoding for each signed or hashed object class.

29.2 Byte Canonicalization Requirements

Within a selected canonical encoding, equivalent protocol objects MUST produce identical canonical bytes. The profile MUST define field ordering, normalization, treatment of absent and default values, representation of integers and strings, extension ordering and rejection of non-canonical alternatives.

A verifier **MUST** compute hashes and signatures over the canonical bytes defined by the applicable Deployment Profile. It **MUST NOT** accept a non-canonical byte representation as an alternative representation of the same signed object unless the profile explicitly defines a normalization procedure before verification.

29.3 Supported Encoding Technologies

A Deployment Profile **MAY** select JSON, CBOR, Protocol Buffers, ASN.1 or another binary or textual technology, provided that the profile defines deterministic byte canonicalization for every protected object class.

A profile **MAY** support additional non-canonical presentation or transport encodings, but those encodings **MUST** be converted to the selected canonical byte representation before hashing, signing or canonical comparison.

29.4 Cross-Encoding Compatibility

Semantic Equivalence applies across encodings. Byte equality applies only within the same canonical encoding and profile definition.

Encoding selection **MUST NOT** change Trust Evaluation semantics, Continuity Evidence meaning, CEP interpretation or Protocol State. Two implementations claiming the same Profile Conformance **MUST** use the same canonical byte encoding for every object class covered by that profile.

30. Version Negotiation

30.1 Purpose

Version Negotiation enables interoperable communication between protocol participants implementing different protocol versions.

Version negotiation **SHALL** occur before any protocol operation requiring semantic compatibility.

30.2 Version Information

Every protocol participant **SHALL** advertise:

- supported protocol version;
- supported Deployment Profile;
- supported optional extensions.

30.3 Negotiation Rules

Participants SHALL negotiate:

1. Protocol Version.
2. Deployment Profile.
3. Required Extensions.
4. Optional Extensions.

If semantic compatibility cannot be established, protocol execution SHALL terminate before Continuity Evidence is generated.

30.4 Backward Compatibility

Implementations SHOULD maintain backward compatibility whenever practical.

Backward compatibility SHALL NOT modify normative protocol semantics.

30.5 Forward Compatibility

Unknown optional protocol elements SHALL be ignored.

Unknown mandatory protocol elements SHALL terminate protocol negotiation.

Termination SHALL preserve previously established Trust Continuity.

30.6 Version Negotiation Sequence

Figure 16. Version negotiation sequence

```

sequenceDiagram
    autonumber
    participant A as Participant A
    participant B as Participant B

    A->>B: Supported versions, profiles and extensions
    B-->>A: Supported versions, profiles and extensions
    A->>A: Determine semantic compatibility
    B->>B: Determine semantic compatibility

    alt Compatible selection exists
        A->>B: Selected version and profile context
        B-->>A: Negotiation accepted
    else No compatible selection
        A->>B: Version mismatch result
        Note over A,B: No continuity-relevant operation is committed
    end
end

```

30.7 Common Version Requirements

Every conformant implementation **MUST** negotiate protocol versions before executing an operation that requires semantic compatibility. It **MUST** preserve Observable Determinism and reject incompatible mandatory versions. Implementations **SHOULD** support backward-compatible evolution and **MAY** support multiple protocol versions simultaneously.

Part IX — Interoperability

31. Protocol Compliance

31.1 Purpose

Protocol Compliance defines the minimum observable behavior required for an implementation to claim conformance with this specification.

31.2 Mandatory Requirements

A compliant implementation **MUST** implement the normative protocol semantics and the Protocol Invariants. It **MUST** preserve Trust Continuity semantics, Continuity Evidence immutability, CEP interoperability, Observable Determinism and the safeguards required by Section 3.4.

31.3 Optional Capabilities

Implementations MAY support:

- additional Deployment Profiles;
- additional Message Types;
- additional synchronization strategies;
- implementation-specific optimizations.

Optional capabilities SHALL NOT alter normative behavior.

32. Conformance Levels

32.1 Conformance Classes

Conformance class	Mandatory capabilities
Core Conformance	Protocol State Semantics, Message Specification, Continuity Evidence, CEP, Graph Puzzle input/output contracts, Trust Evaluation result contracts, Synchronization, Recovery, Version Negotiation and safeguard enforcement.
Extended Conformance	Core Conformance plus optional Extensions, multiple protocol versions or additional interoperable capabilities.
Profile Conformance	Core or Extended Conformance plus every mandatory requirement, cryptographic suite and canonical byte encoding of the claimed Deployment Profile.

An implementation MUST NOT claim a conformance class unless every mandatory capability in that class is implemented and testable.

32.2 Profile Conformance Rules

A Deployment Profile MAY define additional conformance requirements. Such requirements MUST remain compatible with the normative protocol semantics, canonical terminology and Protocol Invariants defined by this specification.

A Profile Conformance claim MUST identify the Protocol Version, profile identifier, mandatory extensions, cryptographic suite and canonical byte encoding to which the claim applies.

32.3 Safeguard Conformance

Safeguard Conformance is mandatory for every conformance class. Conformance testing **MUST** verify rejection and containment of at least:

- invalid state transitions;
- malformed or semantically invalid Protocol Messages;
- duplicate Continuity Evidence;
- destructive synchronization attempts;
- historical overwrite attempts;
- incompatible mandatory extensions;
- unauthorized recovery or synchronization;
- partial-commit and rollback conditions.

A failed safeguard test invalidates the associated conformance claim even when normal success-path tests pass.

32.4 Conformance Test Vectors

Implementations claiming interoperability **MUST** be testable against published or mutually agreed conformance vectors. Those vectors **SHOULD** include successful and rejected Protocol Operations, canonical byte examples, semantic-equivalence pairs, state-transition cases, synchronization conflicts, recovery cases and expected Trust Evaluation result classes.

Conformance vectors specify observable inputs and expected observable results. They **MUST NOT** require disclosure of proprietary Graph Puzzle algorithms, scoring logic or internal optimization strategies.

33. Compatibility Rules

33.1 Normative Requirements

Compatible implementations **SHALL**:

- interpret protocol semantics identically;
- preserve observable protocol behavior;
- preserve Trust Continuity;
- preserve interoperability.

Internal implementation details **MAY** differ.

33.2 Interoperability Requirements

Two conformant implementations claiming compatibility with the same Protocol Version, Deployment Profile, mandatory extensions, cryptographic suite and canonical byte encoding **MUST** interoperate for the conformance class they jointly claim.

Implementations that share only Core Conformance but claim different incompatible profiles are not required to interoperate for profile-specific operations.

33.3 Compatibility Guarantees

Implementations **SHALL** preserve compatibility across:

- cryptographic evolution;
- Deployment Profile evolution;
- protocol version evolution;
- implementation evolution.

33.4 Non-Compatibility

Implementations **SHALL** refuse interoperability whenever semantic compatibility cannot be guaranteed.

Failure to interoperate **SHALL NOT** invalidate previously established Trust Continuity.

33.5 Final Compliance Statement

A conformance claim applies only to the explicitly identified Protocol Version, conformance class and Deployment Profile context. Interoperability evidence **SHOULD** include the applicable safeguard results and conformance-vector results.

An implementation claiming compliance with this specification **SHALL** conform to all normative requirements expressed using:

- **MUST**
- **MUST NOT**
- **SHOULD**
- **SHOULD NOT**
- **MAY**

unless explicitly superseded by a future revision of this specification.

Appendix A — Normative Safeguard Summary

Violation class	Required safeguard	Failure outcome
Invalid state transition	Reject before mutation	Invalid State error; prior state preserved
Malformed or semantically invalid message	Reject during validation	Invalid Message error; no Continuity Evidence mutation
Incompatible mandatory extension	Reject during negotiation	Version or extension incompatibility; prior continuity preserved
Evidence rewrite attempt	Block modification	Integrity failure; original evidence retained
Duplicate operation	Apply idempotency controls	No duplicate Continuity Evidence
Unauthorized synchronization	Reject exchange	Synchronization Failure; local evidence preserved
Partial recovery	Abort atomically	Recovery Failure; no partial continuity extension

Part X — References

34. Normative References

[N1] S. Bradner, “Key words for use in RFCs to Indicate Requirement Levels,” RFC 2119, BCP 14, March 1997.

[N2] B. Leiba, “Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words,” RFC 8174, BCP 14, May 2017.

35. Informative References

[I1] QVTM Position Paper, version 1.1.

[I2] QVTM White Paper, version 1.0.