Network Working Group
Internet-Draft
Intended status: ExperimentalCentre for Development of Advanced Computing (CDAC)
Expires: 29 November 2026

A. Mallick
I. Chebolu

28 May 2026

The Micro Agent Communication Protocol (袖ACP)
draft-mallick-muacp-03

Abstract

This document specifies the Micro Agent Communication Protocol (袖ACP), a resource-efficient messaging protocol for autonomous agents operating on resource-constrained Edge and IoT devices (including Class 1 and Class 2 devices per [RFC7228]). Existing agent communication protocols assume unbounded computational and energy resources, 袖ACP provides mechanisms for bounded resource consumption with deterministic memory bounds (8-byte header, explicitly delimited TLV region, and profile-dependent payload limits) and bounded processing time per message, while maintaining expressiveness sufficient for finite-state coordination patterns. The protocol defines four core message types, a fixed 64-bit header, TLV-based extensibility, and mandatory OSCORE security binding for operation in adversarial environments.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 29 November 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction

The Internet of Things (IoT) and Edge computing domains increasingly demand autonomous coordination among resource-constrained devices. Multi-agent systems, in which software entities collaborate to achieve distributed goals, are a natural fit for these environments. However, existing agent communication protocols, such as FIPA-ACL [FIPA-ACL], assume unlimited computational resources and are unsuitable for devices with kilobytes of RAM operating on battery power.

At the same time, constrained IoT protocols like CoAP [RFC7252] provide efficient request/response primitives but lack the structured semantics necessary for multi-agent coordination patterns such as state dissemination, event-driven subscriptions, and conversation management. This gap leaves developers without a standard, interoperable way to implement lightweight agent communication on microcontroller-class platforms (Class 1 and Class 2 devices per [RFC7228]).

The Micro Agent Communication Protocol (袖ACP) addresses this gap by defining a resource-efficient, semantically expressive protocol tailored for autonomous agents on constrained devices. 袖ACP provides:

* A compact wire format with deterministic resource bounds: 64-bit fixed header, explicitly delimited TLV region, and profile-dependent payload limits.

* Four orthogonal communication primitives (PING, TELL, ASK, OBSERVE) sufficient for liveness detection, request/response, state updates, and publish/subscribe patterns.

* Mandatory OSCORE [RFC8613] security binding ensuring end-to-end confidentiality, integrity, and replay protection without requiring heavyweight TLS stacks.

* Finite-state machine semantics enabling deterministic, verifiable implementations with bounded processing time and memory consumption.

袖ACP is designed to operate over CoAP [RFC7252] as the transport substrate, leveraging CoAP’s congestion control and reliability mechanisms while adding agent-oriented communication semantics. Unlike general-purpose messaging protocols, 袖ACP targets specific coordination patterns found in distributed autonomous systems, making it complementary to, rather than a replacement for, existing IoT protocols.

An accompanying formal model [MUACP] explores resource bounds and safety properties of a simplified 袖ACP core. This model is informative and non-normative, providing an informative exploration of design trade-offs related to determinism. A reference implementation demonstrates 袖ACP operating on ARM Cortex-M microcontrollers with as little as 10 KB of RAM, validating its suitability for severely constrained environments.

The authors welcome feedback on technical approach, scope, and design trade-offs via the 袖ACP reference implementation repository [MUACP-IMPL] and the IETF mailing lists.

1.1. Motivation and Problem Space

Modern IoT deployments face three converging challenges:

1. *Resource Scarcity:* Billions of deployed devices (Class 1: ~10 KB RAM/100 KB flash, Class 2: ~50 KB RAM/250 KB flash per [RFC7228]) cannot support traditional agent communication frameworks or heavy middleware.

2. *Semantic Gap:* Existing constrained protocols (CoAP, MQTT-SN) provide transport-level primitives but lack structured conversation management, subscription scoping, and error semantics required for agent coordination.

3. *Security Requirements:* Autonomous agents operating in adversarial environments (smart cities, industrial control, healthcare) require authenticated, confidentiality-protected communication without the overhead of TLS.

Representative use cases include:

* *Smart Agriculture:* Soil moisture sensors (agents) autonomously coordinate irrigation decisions by subscribing to weather events and sharing state with actuator agents.

* *Industrial IoT:* Machine monitoring agents request diagnostic information from equipment agents and subscribe to fault notifications, all under strict latency and energy budgets.

* *Healthcare Monitoring:* Wearable device agents exchange physiological data with edge gateway agents, requiring authenticated communication and conversation-scoped correlation.

* *Autonomous Robotics:* Swarm robots coordinate via lightweight request/response and event notification without centralized infrastructure.

These scenarios share common requirements: deterministic resource usage, secure multi-party interaction, structured conversation management, and operation on devices where every byte and CPU cycle matters. 袖ACP provides a standardized, interoperable foundation for these patterns.

1.2. Design Principles

袖ACP is guided by the following principles:

* *Deterministic Bounds:* All message processing completes in bounded time and memory. Static allocation and preallocated tables eliminate dynamic memory risks.

* *Minimal Overhead:* The 64-bit fixed header, binary TLV encoding, and optional payload design minimize wire overhead and parsing complexity.

* *Orthogonal Primitives:* Four verbs (PING, TELL, ASK, OBSERVE) compose to support request/response, publish/subscribe, and liveness patterns without semantic overloading.

* *Security by Default:* OSCORE protection is mandatory for all messages except unencrypted PING, ensuring that security is not an afterthought.

* *Transport Compatibility:* 袖ACP builds upon CoAP’s proven constrained-network optimizations (congestion control, blockwise transfer, DTLS/OSCORE) rather than reinventing them.

* *Extensibility Without Bloat:* TLV-based options allow future extensions while maintaining backward compatibility and enabling parsers to skip unknown options.

1.3. Goals

袖ACP aims to:

* Provide minimal, low-overhead communication primitives for constrained autonomous agents with structured, agent-oriented semantics.

* Ensure deterministic and bounded resource usage suitable for Class 1 and Class 2 devices.

* Support essential multi-agent coordination patterns (request/response, publish/subscribe, liveness detection) using four orthogonal primitives.

* Define a secure, interoperable transport binding with mandatory OSCORE protection.

* Enable extensibility via TLV options without breaking compatibility or imposing mandatory overhead.

* Establish IANA registries and interoperability profiles to facilitate independent implementations and ecosystem growth.

1.4. Scope

This specification defines:

* The normative wire format for 袖ACP messages, including header structure, TLV encoding rules, and payload processing.

* The semantics of four core communication verbs (PING, TELL, ASK, OBSERVE) and their state-machine behavior.

* The mandatory CoAP/OSCORE transport binding, including QoS mapping, congestion control, and security context requirements.

* Error handling, version negotiation, and extensibility mechanisms.

* IANA registries for TLV types, QoS codes, verbs, error codes, content formats, and well-known URIs.

* Interoperability profiles (Minimum Interoperability Profile, Constrained Node Profile, Infrastructure Node Profile) and a conformance checklist.

This specification does NOT define:

* *Application-Level Semantics:* The meaning of specific payloads, negotiation protocols, or high-level agent behaviors is application-specific. 袖ACP provides the communication substrate, not the application logic.

* *Ontologies and Knowledge Representation:* Unlike FIPA-ACL, 袖ACP does not prescribe ontology languages, content languages, or semantic frameworks. Developers may use CBOR schemas, JSON-LD, or domain-specific formats as needed.

* *Agent Discovery and Registry:* While /.well-known/muacp provides feature discovery, agent registry protocols and service discovery are out of scope. Existing mechanisms like CoRE Resource Directory [RFC9176] or DNS-SD may be used.

* *Alternative Transport Bindings:* This specification defines CoAP/OSCORE as the mandatory-to-implement transport. Other bindings (e.g., DTLS/UDP, QUIC) may be specified in future companion documents but are not normative here.

* *Non-Goals:* 袖ACP is not a general-purpose messaging queue (like MQTT), not a data-centric pub/sub system (like DDS), and not suitable for Class 0 devices (<10 KB RAM) without significant adaptation.

The scope intentionally focuses on the protocol layer, allowing higher-level agent frameworks, ontologies, and application semantics to evolve independently while maintaining wire-level interoperability.

1.5. Relationship to Other Protocols

袖ACP builds upon and complements existing IETF protocols:

* *CoAP (RFC 7252):* 袖ACP uses CoAP as its transport substrate, leveraging CoAP's request/response model, congestion control (RFC 7252 Section 4.7), and optional blockwise transfer (RFC 7959). 袖ACP messages are carried as CoAP POST payloads.

* *OSCORE (RFC 8613):* 袖ACP mandates OSCORE for end-to-end security, providing authentication, integrity, confidentiality, and replay protection without requiring TLS.

* *CoAP Observe (RFC 7641):* While CoAP Observe provides resource observation, 袖ACP OBSERVE adds agent-oriented subscription semantics with explicit correlation IDs, cancellation TLVs, and subscription state management independent of CoAP's Observe mechanism.

* *MQTT and MQTT-SN:* These protocols provide publish/subscribe for constrained networks but lack request/response semantics, conversation correlation, and structured agent interaction patterns that 袖ACP targets.

* *DDS-XRCE:* DDS for eXtremely Constrained Environments focuses on data-centric publish/subscribe. 袖ACP focuses on agent-to-agent communication with explicit conversations, request/response, and lightweight state machines.

袖ACP is positioned as a specialized protocol for multi-agent coordination on constrained devices, occupying a niche between generic IoT messaging (CoAP, MQTT) and heavyweight agent platforms (FIPA-ACL, JADE).

1.6. Document Structure

The remainder of this document is organized as follows:

* Section 2 defines conventions, terminology, notation, and abbreviations.

* Section 3 specifies the message model and encoding rules, including header format, TLV encoding, payload processing, and OSCORE protection boundaries.

* Section 4 defines the normative semantics of the four 袖ACP verbs (PING, TELL, ASK, OBSERVE) and their processing requirements.

* Section 5 specifies the mandatory CoAP/OSCORE transport binding, including message mapping, QoS semantics, and congestion control.

* Section 6 covers error handling, version negotiation, downgrade protection, and extensibility mechanisms.

* Section 7 defines IANA registries for TLV types, QoS codes, verbs, error codes, content formats, and well-known URIs.

* Section 8 specifies finite-state machines governing conversation lifecycles and resource bounds.

* Section 9 provides comprehensive security considerations, including threat model, authentication, DoS mitigation, and key management.

* Section 10 defines interoperability profiles (MIP, CNP, INP) and feature negotiation mechanisms.

* Section 11 presents normative wire examples with hexadecimal encodings and OSCORE-protected payloads.

* Section 12 provides a conformance checklist for 袖ACP-compliant implementations.

1.7. Implementation Experience

A reference implementation of 袖ACP has been developed and is available as open source. The implementation targets multiple platforms including ESP32 (Xtensa), ARM Cortex-M4, and Linux, demonstrating portability across Class 1, Class 2, and infrastructure devices.

Measured resource usage:

* RAM: ~8 KB (MIP profile, 8 conversations), ~15 KB (CNP profile, 8 conversations with static buffers)

* Flash: ~25 KB (protocol code + OSCORE stack)

* Message throughput: ~200 msg/sec (ESP32 @ 160MHz), ~1000 msg/sec (Linux x86_64)

* Latency: <10ms for local delivery, <100ms over constrained 6LoWPAN networks

Interoperability testing between ESP32 and Linux implementations has validated the wire format, OSCORE protection, and state machine behavior. The implementation confirms that 袖ACP is practical for severely constrained devices while maintaining deterministic resource bounds.

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when they appear in ALL CAPS. These words may also appear in lowercase or mixed case as plain English words, absent their normative meanings.

2.1. Terminology

Key terms: *Agent*: autonomous software entity participating in 袖ACP communication, *Verb*: one of four primitives (PING, TELL, ASK, OBSERVE) encoded in 2 bits, *TLV*: Type-Length-Value encoding (8-bit Type, 8-bit Length, variable Value), *Correlation ID*: 16-bit identifier grouping messages into a conversation, *Sequence ID*: 16-bit monotonically increasing identifier for duplicate detection, *Conversation*: sequence of related messages identified by Correlation ID, *OSCORE*: Object Security for Constrained RESTful Environments [RFC8613], *CoAP*: Constrained Application Protocol [RFC7252], *Constrained Device*: device with limited resources per [RFC7228] (Class 1: ~10 KB RAM/100 KB flash, Class 2: ~50 KB RAM/250 KB flash).

2.2. Notation

Notation: hexadecimal values prefixed with "0x", binary values prefixed with "0b", network byte order (big-endian) unless otherwise specified, bit positions within an octet numbered with bit 7 as the most significant bit (MSB) and bit 0 as the least significant bit (LSB), message formats shown using ASCII art diagrams.

2.3. Abbreviations

Abbreviations: CBOR (Concise Binary Object Representation [RFC8949]), CID (Correlation ID), CoAP (Constrained Application Protocol), COSE (CBOR Object Signing and Encryption [RFC9052]), EDHOC (Ephemeral Diffie-Hellman Over COSE [RFC9528]), FSM (Finite State Machine), IANA (Internet Assigned Numbers Authority), IoT (Internet of Things), MTI (Mandatory to Implement), OSCORE (Object Security for Constrained RESTful Environments), QoS (Quality of Service), SID (Sequence ID), TLV (Type-Length-Value), URI (Uniform Resource Identifier).

3. Message Model and Encoding Rules

This section defines the normative wire-level encoding of 袖ACP messages, including the fixed header, TLV format, payload processing rules, byte ordering, and OSCORE protection boundaries. All compliant implementations MUST follow these encoding rules exactly unless otherwise specified.

3.1. Message Structure

A 袖ACP message consists of three components encoded in the following order:

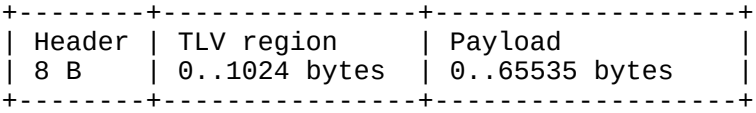


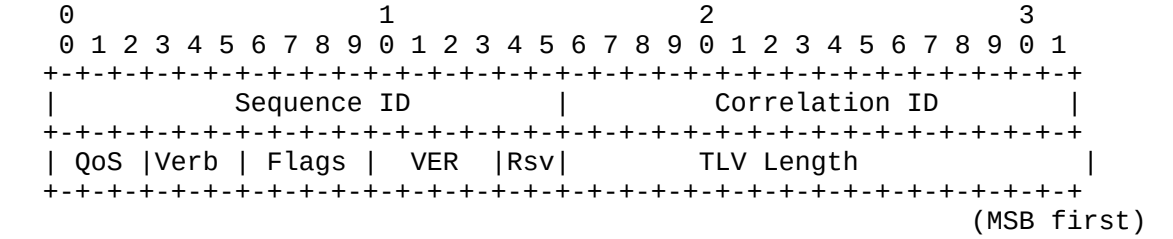
Figure 1: Figure 1: 袖ACP Message Layout

The header format is fixed-length and MUST always appear. TLVs and payloads are optional. The 16-bit TLV Length field in the header delimits the TLV region; the payload is the remaining octets after the header and TLV region. Messages MUST NOT exceed transport-imposed size limits; for CoAP/OSCORE, these limits are determined by underlying MTU constraints and CoAP Blockwise Transfer [RFC7959] if used.

All fields are encoded in network byte order (big-endian).

3.2. Header Format

The 袖ACP header consists of 64 bits arranged as follows:



Byte 0-1: Sequence ID (16 bits, big-endian, MSB first)
Byte 2-3: Correlation ID (16 bits, big-endian, MSB first)
Byte 4: Bits 7-6: QoS (2 bits)
Bits 5-4: Verb (2 bits)
Bits 3-0: Flags (4 bits)
Byte 5: Bits 7-4: VER (4 bits, protocol version)
Bits 3-0: Reserved (4 bits, all zeros)
Bytes 6-7: TLV Length (16 bits, big-endian)

Figure 2: Figure 2: 袖ACP Header Bit Layout

The VER field in byte 5 carries the protocol version used for this message, allowing a receiver to select the correct parsing rules before examining any TLV content. This field addresses the bootstrapping problem inherent in TLV-based version negotiation: a receiver MUST parse byte 5 first and apply the corresponding wire-format rules before proceeding to the TLV region. The current specification defines version 0x0. Senders MUST set VER to the highest version mutually negotiated for the conversation (or 0x0 if no negotiation has occurred). Receivers MUST reject messages whose VER value exceeds their maximum supported version with ERR_VERSION_MISMATCH.

Sequence ID (16 bits, bytes 0-1): Monotonically increasing identifier used for duplicate detection and replay-window tracking. Sequence ID is per-sender (per OSCORE security context) and monotonically increases within each sender’s message stream. MUST wrap modulo 2^16. Sequence ID MUST be initialized to a cryptographically random value to prevent predictability and traffic analysis. The only exception is when ALL of the following conditions are met: (1) a new OSCORE security context is being established, (2) no prior communication history exists with the peer, AND (3) the initialization is synchronized with the establishment of the new OSCORE context. In this specific case, initialization to 0 is acceptable. In all other cases, random initialization is mandatory.

Correlation ID (16 bits, bytes 2-3): Identifies all messages belonging to the same conversation. Correlation ID MUST be unique among active conversations from the same sender (same OSCORE security

context). Different senders may independently use the same Correlation ID values, as conversations are scoped per OSCORE security context. SHOULD be randomly generated in security-sensitive deployments.

QoS (2 bits, byte 4 bits 7-6): Encodes transmission semantics (fire-and-forget, confirmable transfer, or non-confirmable transfer without 袖ACP retransmission). Values are defined in the IANA Considerations section.

Verb (2 bits, byte 4 bits 5-4): Identifies one of the four 袖ACP operations: PING(0), TELL(1), ASK(2), OBSERVE(3).

Flags (4 bits, byte 4 bits 3-0): Control bits reserved for protocol-level features such as fragmentation, retransmission hints, or message cancellation. Future specifications MAY define additional meanings for individual flag bits.

VER (4 bits, byte 5 bits 7-4): Protocol version number for this message. MUST be set to 0x0 in this specification. A non-zero value indicates a later version of 袖ACP. Receivers encountering an unsupported VER value MUST return ERR_VERSION_MISMATCH and MUST NOT attempt to parse the remainder of the message under the current version’s rules. Version 0x0 is this specification. Future VER values are defined only by Experimental or Standards Track RFCs that explicitly update or obsolete this document; no separate IANA registry is created for VER values.

Reserved (4 bits, byte 5 bits 3-0): MUST be set to zero on transmission and MUST be ignored by receivers.

TLV Length (16 bits, bytes 6-7): Specifies the number of octets in the TLV region immediately following the header. TLV Length MUST NOT exceed 1024. A value of 0 indicates that no TLVs are present. Receivers MUST reject messages where TLV Length exceeds the remaining message size or the active profile limit. The payload length is the remaining message length after the 8-byte header and TLV Length octets.

3.3. TLV Encoding

TLVs (Type-Length-Value structures) convey optional metadata and extensibility information. They appear immediately after the header, occupy exactly the number of octets specified by the header TLV Length field, and MUST appear in Type-increasing order to allow binary search and deterministic parsing.

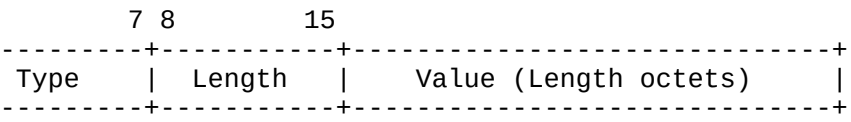


Figure 3: Figure 3: TLV Encoding

Type (8 bits): TLV identifier. Bit 7 (MSB) of the Type byte is the Criticality flag: if bit 7 = 0, the TLV is non-critical (unknown types MUST be silently ignored); if bit 7 = 1, the TLV is critical (unknown types MUST cause the message to be rejected with ERR_UNSUPPORTED_TLV). The remaining 7 bits (bits 6-0) identify the specific TLV within its criticality class. Registration ranges are as follows:

* 0x00-0x1F (non-critical): Expert Review

* 0x20-0x7F (non-critical): Expert Review

* 0x80-0x9F (critical): Expert Review

* 0xA0-0xBF (critical): Expert Review

* 0xC0-0xFF (critical): Private Use

Length (8 bits): Specifies the number of octets in the Value field (0-255). The Length field MUST NOT exceed 255. Receivers MUST validate that the declared Length does not exceed the remaining message buffer before reading the Value field. Each individual TLV's Value field MUST NOT exceed 255 octets. The total TLV region (sum of all TLV lengths plus their Type and Length fields) MUST NOT exceed 1024 bytes. Implementations MUST validate both constraints: individual TLV length <= 255 and total TLV region <= 1024 bytes.

Value: Encoded according to Type. For Types other than 0x00 (Raw Octets), the Value is subject to OSCORE protection (Section 5).

Critical TLVs: Criticality is encoded in bit 7 of the Type byte as defined above. Receivers MUST reject messages containing unknown critical TLVs (bit 7 = 1) with ERR_UNSUPPORTED_TLV. Unknown non-critical TLVs (bit 7 = 0) MUST be silently ignored.

3.3.1. TLV Processing Rules

Receivers MUST apply the following rules when processing TLVs:

* TLVs MUST be parsed strictly in order.

* If Length exceeds the remaining octets in the TLV region, the message MUST be discarded.

* Unknown TLV Types MUST be ignored unless they are designated critical.

* TLV order MUST be strictly increasing by Type, violating this is a format error.

* TLV Type 0x00 (Raw Octets) MUST NOT appear in encrypted messages; its use is restricted to unencrypted PING messages and its Value field MUST NOT exceed 255 bytes.

* TLV Type 0x10 (Reserved Fragmentation) MUST be silently ignored. A future specification will define its semantics.

3.4. Payload Encoding

The 袖ACP payload is an optional octet string of 0-65535 bytes used for application data, action parameters, event notifications, or encoded content (CBOR, JSON). Payload length is inferred from the enclosing transport object after subtracting the 8-byte header and the TLV Length value. Payloads MUST be OSCORE-protected unless the message Verb is PING. Payload sizes MUST be validated before allocation. If encoded using CBOR (Type=0x03), receivers MUST treat it as a single CBOR data item. If JSON (Type=0x02), it MUST be UTF-8 encoded.

3.5. Byte Ordering

All multi-octet integer fields in 袖ACP (Sequence ID, Correlation ID, header composites) MUST be encoded in network byte order (big-endian). TLV and payload content MAY use other encoding rules (e.g., CBOR or UTF-8) as determined by their Types.

3.6. Fragmentation (Optional Feature)

袖ACP does not mandate fragmentation. TLV Type 0x10 is reserved for future fragmentation specification but MUST NOT be used until fully specified. Deployments using CoAP Blockwise Transfer [RFC7959] SHOULD avoid 袖ACP-level fragmentation.

3.7. OSCORE Protection Boundaries

When 袖ACP is transported over CoAP with OSCORE, the OSCORE-protected CoAP payload MUST contain the complete 袖ACP message (Header | TLVs | Payload). OSCORE MUST protect: all TLVs except those in unencrypted PING messages, the entire payload, and header fields other than those needed for outer CoAP routing. Implementations MUST NOT leak semantics (e.g., Verb, QoS) through the CoAP outer header beyond what OSCORE permits.

3.8. Canonical Encoding Rules

Canonical encoding rules: fields MUST NOT be padded, TLVs MUST be sorted by ascending Type, no two TLVs may share the same Type unless explicitly defined, the TLV region MUST contain exactly TLV Length octets, payload MUST begin immediately after the TLV region, and implementations MUST normalize line endings, whitespace, or internal representations before hashing or signing application content.

4. Protocol Semantics

This section defines the normative semantics of the four 袖ACP verbs: PING, TELL, ASK, and OBSERVE. Each verb represents a fundamental communication primitive intended to support higher-level agent behaviors, including liveness detection, request/response interactions, state dissemination, and event-driven notification.

Agents MUST implement all four verbs. Agents MUST apply OSCORE protection to all messages except PING. The only unprotected 袖ACP message permitted by this specification is PING, and support for unprotected PING is OPTIONAL.

PING Security Policy: OSCORE-protected PING is mandatory to implement. Support for unencrypted PING is OPTIONAL and MUST be explicitly configurable. Deployments that enable unencrypted PING accept the associated privacy and security risks documented in Section 9.

For each verb, this section defines sender behavior, receiver behavior, state-machine interactions, mandatory error cases, and expected processing-time bounds.

4.1. PING

PING provides low-cost reachability and liveness detection. Implementations MUST support OSCORE-protected PING. Implementations MAY support unencrypted PING for lightweight liveness detection in environments where unauthenticated reachability checking is acceptable. When unencrypted PING is supported, deployments SHOULD carefully consider the privacy and security implications.

Sender behavior: MAY emit PING at any time, MUST increment Sequence ID, SHOULD use unique Correlation ID, SHOULD rate-limit PING transmissions (the RECOMMENDED default is no more than one per 10 seconds per peer on constrained networks; deployments on higher-capacity links MAY use shorter intervals). Unencrypted PING MUST carry no TLVs other than RAW_OCTETS (Type=0x00) and MUST carry a zero-length payload. Receiver behavior: MUST reply to OSCORE-protected PING with an OSCORE-protected TELL carrying the same Correlation ID and an empty payload. When unencrypted PING mode is explicitly enabled, a receiver MUST reply with an unencrypted TELL carrying the same Correlation ID, Verb=TELL (1), and an empty payload — this is the only permitted unprotected TELL in the protocol. Receivers MUST rate-limit PING processing to mitigate abuse.

Security note: Unencrypted PING messages may leak topology and presence information through timing analysis, message frequency patterns, and correlation tracking. Implementations supporting unencrypted PING SHOULD use rate limiting, randomized response timing, and Correlation ID randomization to reduce information leakage. For authenticated liveness detection, use OSCORE-protected PING or ASK/TELL with OSCORE.

4.2. TELL

TELL conveys information, updates, or asynchronous notifications, and responds to ASK messages. TELL messages MUST be OSCORE-protected. Sender: MUST increment Sequence ID; when responding to ASK, MUST use the same Correlation ID; MAY carry an empty payload and empty TLV set (e.g., for ACK-like responses). Receiver: MUST validate OSCORE, MUST associate via Correlation ID, MUST incorporate content per application policy. Errors: TELL without OSCORE MUST be rejected, malformed TLVs MUST cause discard.

4.3. ASK

ASK initiates a request for information or action and typically elicits a TELL response. ASK messages MUST be OSCORE-protected. Sender: MUST allocate new conversation entry indexed by Correlation ID, MUST increment Sequence ID, MUST start request timer (default timeout of 30 seconds is RECOMMENDED for constrained devices, with exponential backoff for retransmissions when QoS=1), MUST enforce conversation limits. Receiver: MUST validate OSCORE, MUST associate ASK with Correlation ID, MUST generate TELL response with result or error TLV. Errors: malformed TLVs result in TELL(error), security validation failure results in silent discard, and correlation-table limits exceeded results in resource exhaustion error.

4.4. OBSERVE

OBSERVE establishes a subscription for future event-driven notifications, scoped to a single authenticated peer. OBSERVE messages MUST be OSCORE-protected. The subscriber (sender of OBSERVE): MUST allocate/update subscription state indexed by Correlation ID, MUST validate subscription limits, MUST increment Sequence ID, MAY include subscription parameter TLVs (topic, conditions, SUBSCRIPTION_LIFETIME). The publisher (receiver of OBSERVE): MUST validate OSCORE, MUST establish/refresh subscription state, MUST enforce subscription expiration and resource ceilings, and when the subscribed condition is met MUST deliver event notifications as OSCORE-protected TELL messages to the subscriber’s CoAP endpoint (see Section 5.6 for the notification delivery mechanism).

Subscription Lifetime: Every subscription has a lifetime after which the publisher MUST free subscription state and SHOULD send TELL(ERR_TIMEOUT) to the subscriber. If the OBSERVE message carries a SUBSCRIPTION_LIFETIME TLV (Type=0x23), the publisher MUST use that value (in seconds, uint32) as the lifetime. If no SUBSCRIPTION_LIFETIME TLV is present, the publisher MUST apply the RECOMMENDED default lifetime of 86400 seconds (24 hours). Implementations MAY use a shorter default appropriate to their device class; the chosen default SHOULD be advertised in the feature-negotiation response. A subscriber MUST refresh a subscription before it expires by sending a new OBSERVE message with the same Correlation ID; the publisher MUST reset the lifetime timer upon receipt of a valid refresh. Subscribers SHOULD refresh at least 60 seconds before the known expiry.

The subscription remains active and the publisher continues sending event-triggered TELL notifications until explicitly cancelled or expired. Cancellation: sender issues OBSERVE or TELL with

CANCEL_SUBSCRIPTION TLV (Type=0x80, critical), and the receiver MUST immediately delete subscription state and cease notifications; the receiver MUST respond with a TELL confirming cancellation. Errors: subscription limits exceeded result in TELL(error), and OSCORE validation failure results in silent drop.

4.5. Summary of Normative Requirements

Summary: PING is a liveness probe, MUST support OSCORE-protected PING, and MAY support unencrypted PING for lightweight deployments; TELL is an update, response, or notification and MUST use OSCORE; ASK is a request, MUST use OSCORE, and MUST generate a TELL response; OBSERVE is a subscription, MUST use OSCORE, and MUST create or update subscription state. Agents MUST NOT overload verbs with incompatible semantics.

5. Mandatory Transport Binding: OSCORE/CoAP

This section defines the mandatory-to-implement (MTI) transport binding for ACP: the combination of the Constrained Application Protocol (CoAP) as the transport substrate and OSCORE as the end-to-end object security mechanism. All compliant ACP implementations MUST support this binding.

Deployments MAY support additional bindings (e.g., DTLS/UDP as specified in [RFC9147] or QUIC) but such bindings are outside the scope of this specification and MUST NOT weaken or replace the OSCORE/CoAP MTI profile.

5.1. Mapping ACP Messages to CoAP

Each ACP message (Header | TLVs | Payload) is encoded as a byte string and placed entirely within the CoAP message payload. Only OSCORE-protected CoAP messages may carry ACP messages (except PING, which MAY be unprotected). ACP messages MUST use: Method=POST, URI-Path="muacp" (fixed), Content-Format=application/muacp, Payload=Full ACP message. Each ACP message corresponds to exactly one CoAP POST.

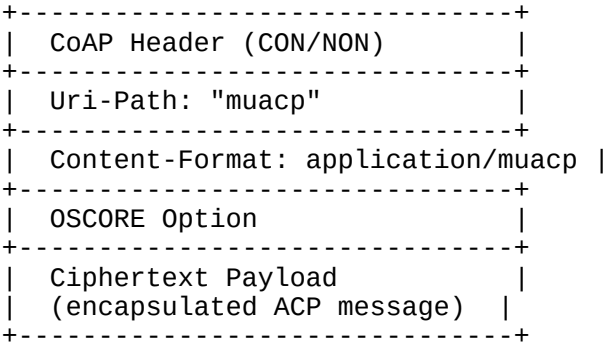


Figure 4: Figure 4: CoAP Envelope Carrying a ACP Message

5.2. OSCORE Protection Requirements

All ACP messages except unencrypted PING MUST be protected using OSCORE [RFC8613], which uses COSE [RFC9052] for cryptographic operations. OSCORE MUST protect: the entire ACP header (except outer CoAP routing metadata), all TLVs except raw TLVs permitted for PING, and the entire ACP payload. OSCORE replay protection MUST be enabled with replay windows configured to match expected message rate and resource constraints. OSCORE MUST use a unique security context per agent-pair.

5.3. Establishing OSCORE Security Contexts

Security contexts for OSCORE MAY be derived by: EDHOC (RECOMMENDED), Pre-Shared Keys (PSK), or out-of-band provisioning. When EDHOC is used, the resulting OSCORE context MUST be bound to the EDHOC handshake transcript to prevent identity misbinding attacks. If a device exhausts its available context storage, it MUST reject new context establishment requests with ERR_RESOURCE_EXHAUSTED. If all OSCORE contexts are active and a new context establishment request arrives, implementations MUST reject it with ERR_RESOURCE_EXHAUSTED. Implementations SHOULD implement context eviction policies (e.g., least-recently-used) for inactive contexts but MUST NOT terminate active conversations. Implementations SHOULD limit concurrent OSCORE contexts according to their device class, see Section 10 for profile-specific recommendations (MIP: 8 contexts minimum, CNP: 8 contexts minimum, INP: 64 contexts minimum).

5.4. CoAP Message Types and Reliability

ACP builds upon CoAP reliability semantics to achieve its QoS model. Implementations MUST map ACP QoS codes to CoAP message types as follows:

ACP QoS	Meaning	CoAP Message Type
0	fire-and-forget	NON (Non-confirmable)
1	confirmable transfer	CON (Confirmable)
2	non-confirmable, no ACP retransmission	NON (Non-confirmable, no retransmission)

Table 1

QoS Semantics: QoS=0 (fire-and-forget) provides best-effort transfer with no ACP delivery tracking; the sender does not expect an application response unless the verb semantics require one. QoS=1 requests CoAP CON processing and CoAP retransmissions, but a CoAP acknowledgment does not prove application processing. Messages may be delivered more than once, and receivers MUST use the OSCORE replay window together with Sequence ID and Correlation ID to suppress duplicate side effects when the requested operation is not idempotent. QoS=2 requests one NON transfer and no ACP retransmission; if delivery fails, no retry occurs. QoS=2 expresses sender intent to avoid protocol retries, not a network-wide guarantee that duplicates can never occur. Implementations MUST NOT retransmit QoS=2 messages at the ACP layer.

CoAP-level acknowledgments MUST NOT be interpreted as ACP-level responses. Application responses are always encoded as TELL messages.

5.5. Mapping ASK/TELL to CoAP Request/Response

ASK messages MUST be sent as CoAP POST requests, TELL responses as CoAP responses. OSCORE MUST protect both directions. The Correlation ID uniquely links ASK with TELL response. CoAP Message IDs MUST NOT be used for application correlation. Receivers MUST respond with TELL even when requests fail, using an Error TLV.

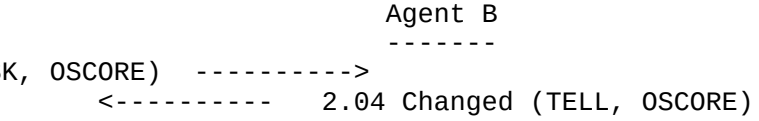


Figure 5: Figure 5: ASK/TELL Over OSCORE-CoAP

5.6. Mapping OBSERVE Subscriptions

OBSERVE establishes a long-lived subscription. ACP defines its own subscription model, independent of CoAP's Observe extension [RFC7641]. The initial OBSERVE message MUST be sent as a CoAP POST to the publisher's "/muacp" endpoint. Event notifications are delivered by the publisher as CoAP POST requests to the subscriber's "/muacp" endpoint, i.e., the publisher acts as a CoAP client for notification delivery. Implementations MUST NOT use CoAP Observe for ACP-defined subscription semantics. This restriction applies only to ACP message exchange and does not preclude concurrent use of CoAP Observe for resource-centric interactions outside ACP.

Notification Delivery Architecture: ACP OBSERVE requires each agent to expose a CoAP server endpoint at "/muacp" and be reachable by its peers for incoming CoAP POST requests on the selected transport path. When a subscriber sends an OBSERVE to a publisher, it implicitly registers its own CoAP endpoint (identified by the OSCORE security context and the CoAP transport address from which the OBSERVE was sent) as the delivery target for notifications. Deployments behind NATs, firewalls, or sleeping-link gateways MUST provide a reachable path by configuration, Resource Directory-assisted discovery, or an application relay outside this specification. A change in the subscriber's transport address invalidates the delivery target until the subscriber refreshes the subscription. The publisher MUST deliver event notifications to the current delivery target as OSCORE-protected CoAP POST requests carrying TELL messages with the matching Correlation ID. If the publisher cannot reach the subscriber's endpoint, it MUST apply the selected QoS and CoAP retransmission policy; after exhausting permitted retries, the publisher MUST terminate the subscription and free the associated state.

CoAP Observe (RFC 7641) is resource-centric: clients observe changes to a specific resource identified by URI. ACP OBSERVE is conversation-centric: agents establish subscriptions scoped to agent-to-agent conversations, identified by Correlation ID and OSCORE security context. While both mechanisms provide event notification, they address different use cases: CoAP Observe targets resource state monitoring, while ACP OBSERVE enables agent interaction patterns with subscription lifecycle tied to conversations. The two mechanisms are complementary and may coexist in deployments.

5.7. Congestion Control Requirements

All ACP-over-CoAP deployments MUST implement congestion control to prevent network collapse and unfair bandwidth usage. Implementations MUST follow CoAP congestion control mechanisms as specified in [RFC7252] Section 4.7.

Agents MUST adhere to: exponential backoff on CoAP CON retransmissions (initial timeout >= 2s, max 247s per [RFC7252]), PING rate limiting (RECOMMENDED default: <= 1 per 10 seconds per peer on constrained networks; deployments SHOULD configure this limit per their link characteristics), OBSERVE throttling when bandwidth pressure is detected, deterministic resource usage, message rate limits per conversation. When Blockwise Transfer [RFC7959] is used, agents MUST ensure block sizes do not exceed memory limits.

5.8. Transport-Layer Error Handling

Transport errors (CoAP timeouts, OSCORE decryption failures, missing acknowledgments) MUST be translated into ACP-level behavior. OSCORE decryption failures cause message drop. Unacknowledged CoAP CON messages apply ACP QoS semantics for retransmission. Repeated timeouts move the conversation to a failure state. Malformed CoAP envelopes are discarded.

5.9. Summary of MTI Requirements

All compliant ACP implementations MUST: support CoAP POST to fixed path "muacp", support Content-Format application/muacp, protect all messages except unencrypted PING with OSCORE, enforce OSCORE replay protection, derive OSCORE contexts using EDHOC or equivalent, map QoS codes to CoAP message types, generate TELL responses for all ASK messages, deliver OBSERVE notifications as TELL messages. This binding establishes interoperability and provides a minimum security baseline.

6. Error Handling, Version Negotiation, and Extensibility

This section defines normative error-handling rules, version-negotiation mechanism, downgrade protection requirements, and the extensibility framework provided by the TLV architecture.

6.1. Error Code TLVs

All protocol-level errors MUST be communicated using a TELL message that includes an Error-Code TLV. Error codes are encoded as unsigned 8-bit integers and MUST follow the registry defined in the IANA Considerations section.

```
Type:   0x22 (Error-Code, see IANA registry)
Length: 1 octet
Value:  uint8 error code
```

Figure 6: Figure 6: Error-Code TLV

The sender MUST set the Correlation ID of the error response to match the ID of the failing message. Receivers MUST interpret the error code as part of the ACP conversation state.

6.2. Standardized Error Conditions

The following error codes are defined for ACP:

Code	Name	Description
0x00	SUCCESS	No error, operation completed successfully. This code is OPTIONAL. If omitted, successful completion is indicated by the absence of an Error-Code TLV. Receivers MUST treat the absence of an Error-Code TLV as equivalent to SUCCESS (0x00).
0x01	ERR_MALFORMED	Malformed header, TLV, or payload.
0x02	ERR_UNSUPPORTED_VERB	Verb not recognized or not supported by receiver.
0x03	ERR_UNSUPPORTED_TLV	Critical TLV not understood.
0x04	ERR_FORBIDDEN	Operation not permitted due to policy or authorization.
0x05	ERR_RESOURCE_EXHAUSTED	Memory, CPU, or subscription/ conversation limits exceeded.
0x06	ERR_VERSION_MISMATCH	Message uses unsupported protocol version.
0x07	ERR_TIMEOUT	Sender or receiver timed out while waiting for a response.
0x08	ERR_INTERNAL	Internal failure not covered by other error categories.
0x09	ERR_REPLAY	Message rejected as potential replay or out-of-order delivery (Sequence ID less than or equal to last observed value for this Correlation ID).

Table 2

Implementations MAY define additional private-use error codes in the private-use range but MUST NOT redefine standardized codes.

6.3. Handling Malformed Messages

Receivers MUST apply strict validation: if header TLV Length exceeds remaining bytes or the active profile limit, discard; if an individual TLV Length exceeds the remaining TLV-region bytes, discard; if TLVs appear out of Type order, discard; if required TLV (future versions) is absent, reject; if header fields contain invalid combinations (e.g., reserved bits set), reject; if OSCORE decryption fails, discard without error signaling. Where feasible, receivers SHOULD send TELL(error) unless doing so would amplify a denial-of-service attack.

6.4. Conversation-Lifetime Error Handling

Conversations MAY fail due to timeouts, resource limits, or message corruption. When such failures occur:

* The agent MUST free associated resources (conversation-table entries).

* The agent SHOULD send an ERR_TIMEOUT or ERR_RESOURCE_EXHAUSTED TELL message.

* For resource exhaustion, an agent MUST NOT attempt recovery that risks violating its resource budget.

If a Correlation ID collision is detected (a new message arrives with a Correlation ID matching an active conversation from the same sender, as identified by the OSCORE security context), the receiver MUST apply the following deterministic strategy in order:

1. If the conversation table is full (all entries occupied), reject the new message with ERR_RESOURCE_EXHAUSTED and maintain the existing conversation.

2. If the new message’s Sequence ID (from the same sender) is greater than the existing conversation’s last observed Sequence ID from that sender, terminate the existing conversation (free its resources), accept the new message, and create a new conversation entry. This handles legitimate Correlation ID reuse after conversation completion or timeout. Note: Sequence IDs are per-sender and monotonically increase within each sender’s message stream. To handle Sequence ID wrap-around (modulo 2^16), implementations MUST use sequence number comparison as defined in [RFC1982] Section 3.1: given two Sequence IDs S1 and S2, S1 is considered greater than S2 if (S1 > S2 and S1 - S2 < 2^15) or (S1 < S2 and S2 - S1 > 2^15). This ensures correct ordering even when Sequence IDs wrap from 0xFFFF to 0x0000.

3. If the new message's Sequence ID is less than or equal to the existing conversation's last observed Sequence ID from the same sender, reject the new message as a potential replay or out-of-order delivery. The receiver MUST NOT modify the existing conversation state and SHOULD silently discard the new message (or MAY send ERR_REPLAY to indicate the rejection reason).

This deterministic strategy ensures interoperability while preventing resource exhaustion and replay attacks. Correlation ID collisions are rare when Correlation IDs are randomly generated with sufficient entropy. Collisions from different senders (different OSCORE contexts) are handled separately, as each OSCORE context maintains its own conversation state. The Sequence ID comparison is secure because Sequence IDs are authenticated and integrity-protected by OSCORE.

Example: If an active conversation exists with Correlation ID=0x1234 and last observed Sequence ID=0x0010 from sender A (identified by OSCORE context A), and a new message arrives with Correlation ID=0x1234 and Sequence ID=0x0015 from the same sender A, the receiver terminates the old conversation and accepts the new one. If the new message has Sequence ID=0x0005 from sender A, it is rejected as a potential replay. If a message arrives with Correlation ID=0x1234 from sender B (different OSCORE context), it is treated as a separate conversation, as conversations are scoped per OSCORE security context.

6.5. Version Negotiation

ACP includes a Version-TLV (Type=0x01) that MAY be included in any message to indicate supported protocol versions. If no Version-TLV is present, receivers MUST assume version 0x00 (this specification).

The VER header field is the wire-format selector and MUST be examined before interpreting any TLV, including the Version-TLV. A receiver MUST parse the fixed header, validate that VER identifies a supported wire format, and only then parse the TLV region according to that version's rules. The Version-TLV negotiates the version to use for subsequent messages in a conversation; once a version is selected, senders MUST place the selected value in the VER field of subsequent messages for that conversation.

Version negotiation follows these rules:

* If a message includes a Version-TLV that indicates only unsupported versions (i.e., all versions listed in the Version-TLV are higher than the receiver’s maximum supported version), the receiver MUST return ERR_VERSION_MISMATCH in a TELL error response.

* If the Version-TLV contains at least one supported version, the receiver MUST use the highest mutually supported version for subsequent messages in the conversation.

* When both parties send Version-TLVs (e.g., in ASK and TELL), each party MUST independently select the highest mutually supported version from the union of both Version-TLV lists. If no common version exists, the receiver MUST return ERR_VERSION_MISMATCH.

* The selected version applies to all messages in the conversation identified by the Correlation ID. Once a version is selected, it MUST NOT be changed for that conversation.

* Version-TLV-based negotiation MUST occur under OSCORE protection. Note: the VER header field (byte 5, bits 7-4) carries the active version unprotected by design, since it must be read before OSCORE decryption; this is intentional. PING messages SHOULD NOT carry Version-TLV. If Version-TLV negotiation is required before OSCORE context establishment, implementations SHOULD establish the OSCORE context first, then negotiate versions in a subsequent message.

```
Type:   0x01 (Version)
Length: N (number of supported versions, 1-255)
Value:  Sequence of N unsigned 8-bit integers.
        Each integer represents one supported protocol version
        (e.g., [0x00] for version 0).
```

Figure 7: Figure 7: Version TLV

Encoding: The Value field of the Version TLV is a sequence of N unsigned 8-bit integers (where N is the Length field value). Each integer represents a protocol version number. For example, a Version TLV indicating support for versions 0x00 and 0x01 would have Length=2 and Value=[0x00, 0x01]. Implementations MUST encode version numbers as single octets (0-255). Receivers MUST parse the Value field as a sequence of Length octets, each interpreted as an unsigned 8-bit version number.

6.6. Downgrade and Version-Rollback Protection

Implementations MUST ensure attackers cannot force a peer to use a lower protocol version when a higher mutually supported version is available. The highest mutually supported version MUST be chosen. Version negotiation MUST occur inside OSCORE-protected messages (except PING). Agents MUST NOT downgrade versions unless a failure condition explicitly requires fallback.

6.7. Extensibility Framework

ACP evolves through TLV-based extensibility. Constraints: receivers MUST ignore unknown non-critical TLVs, implementations MUST NOT reuse TLV Types for different semantics, future versions MAY introduce critical TLVs (unsupported critical TLVs trigger ERR_UNSUPPORTED_TLV), all TLVs MUST be sorted by increasing Type value, and private-use TLV values MUST NOT be assumed to interoperate across vendors. Complex extensions SHOULD define new structured TLVs rather than overloading primitive types.

6.8. Summary of Normative Requirements

Malformed messages MUST be rejected and SHOULD trigger TELL(error) unless unsafe. Errors MUST use standardized codes. Version negotiation MUST prefer the highest mutually supported version. Unknown non-critical TLVs MUST be ignored, unknown critical TLVs MUST trigger errors. OSCORE failures MUST cause silent discard. Resource exhaustion MUST lead to conservative cleanup behavior.

7. IANA Considerations

This section requests the creation of new registries and assignments required for ACP to function as an interoperable Internet protocol. This document is published on the Independent Submission stream. Consistent with the independent stream’s scope, all new registries use the Expert Review policy as defined in [RFC8126] Section 4.5, administered by designated experts appointed by IANA in consultation with the ISE. Designated experts are expected to have expertise in constrained IoT protocols (e.g., CoAP, OSCORE, 6LoWPAN). Expert review criteria: the proposed value must have a stable public specification, must not conflict with existing assignments, and must be consistent with the ACP design principles in this document.

7.1. ACP TLV Types Registry

IANA is requested to create a new registry entitled "ACP TLV Types" (8-bit values 0-255). Each entry MUST contain: Value, Name, Description, Value format, Reference. The range is divided as follows:

Bit 7 of the Type byte is the Criticality flag (0 = non-critical, 1 = critical). All ranges use Expert Review unless noted:

* *0x00-0x1F* (non-critical): Expert Review
* *0x20-0x7F* (non-critical): Expert Review
* *0x80-0x9F* (critical): Expert Review
* *0xA0-0xBF* (critical): Expert Review
* *0xC0-0xFF* (critical): Private Use

Initial values, all with Reference "This document":

Value	Name	Critical	Description	Format
0x00	RAW_OCTETS	No	Unstructured data, MUST NOT appear in encrypted messages; restricted to unencrypted PING. Payload MUST be zero-length or limited to 255 bytes.	Opaque
0x01	VERSION	No	Advertised supported protocol versions.	Array of uint8
0x02	CONTENT_TYPE	No	Specifies payload encoding.	uint8
0x03	CBOR_PAYLOAD	No	Payload encoded	CBOR

				as CBOR.	data item
0x10	RESERVED_FRAGMENTATION	No	Reserved for future fragmentation extension. NOT defined by this specification. Receivers MUST silently ignore this TLV until a future document defines its semantics.		N/A
0x20	TOPIC	No	Subscription topic for OBSERVE.		UTF-8 string
0x21	CONDITION	No	Trigger condition for OBSERVE.		UTF-8 or CBOR
0x22	ERROR_CODE	No	Error code returned in TELL(error).		uint8
0x23	SUBSCRIPTION_LIFETIME	No	Requested subscription lifetime in seconds for OBSERVE. uint32, big-endian. If absent, publisher applies default (RECOMMENDED: 86400 seconds).		uint32
0x80	CANCEL_SUBSCRIPTION	Yes	Explicit termination of OBSERVE subscription. Critical: receivers MUST		Empty

			process this TLV and stop notifications. Value field MUST be empty (Length=0).		

Table 3

Future extensions MUST NOT assign new semantics to existing TLV values.

7.2. ACP QoS Codes Registry

IANA is requested to create a registry entitled "ACP QoS Codes". QoS is encoded as a 2-bit field in the header (values 0-3).

Value	Name	Description	Reference
0	FIRE_AND_FORGET	No reliability, mapped to CoAP NON.	This document
1	CONFIRMABLE_TRANSFER	Use CoAP CON and CoAP retransmission behavior.	This document
2	NON_CONFIRMABLE_NO_RETRY	Use one CoAP NON transfer with no ACP retransmission.	This document
3	RESERVED	Reserved for future use.	This document

Table 4

7.3. ACP Verb Codes Registry

IANA is requested to create a registry entitled "ACP Verb Codes". Verb values occupy 2 bits but are listed numerically (0-3).

Value	Name	Description	Reference
0	PING	Liveness probe.	This document
1	TELL	State update, notification, or response.	This document
2	ASK	Request for information or action.	This document
3	OBSERVE	Subscription to events or state changes.	This document

Table 5

7.4. ACP Error Codes Registry

IANA is requested to create a registry entitled "ACP Error Codes" consisting of integers 0-255.

The assignment policy for values 0-127 is Expert Review. Values 128-255 are Private Use.

Initial values:

Code	Name	Description	Reference
0x00	SUCCESS	No error, operation completed successfully.	This document
0x01	ERR_MALFORMED	Malformed header, TLV, or payload.	This document
0x02	ERR_UNSUPPORTED_VERB	Verb not recognized or not supported.	This document
0x03	ERR_UNSUPPORTED_TLV	Critical TLV not understood.	This document
0x04	ERR_FORBIDDEN	Operation not permitted.	This document
0x05	ERR_RESOURCE_EXHAUSTED	Resource limits exceeded.	This document
0x06	ERR_VERSION_MISMATCH	Unsupported protocol version.	This document
0x07	ERR_TIMEOUT	Request timed out.	This document
0x08	ERR_INTERNAL	Internal failure.	This document
0x09	ERR_REPLAY	Message rejected as potential replay.	This document

Table 6

7.5. CoAP Content-Format Registration

IANA is requested to register the following CoAP Content-Format:

Media Type	Encoding	ID	Reference
application/muacp	binary	TBD (to be assigned by IANA)	This document

Table 7

This Content-Format is mandatory for all ACP-over-CoAP messages.

The Content-Format ID will be assigned by IANA prior to publication. CoAP Content-Format assignments for independent submissions follow the Expert Review policy for the CoAP Content-Formats registry.

7.6. Media Type Registration

IANA is requested to register the following media type in the "application" registry per [RFC6838]:

Type name: application
Subtype name: muacp
Required parameters: none
Optional parameters: none
Encoding considerations: binary
Security considerations: See the Security Considerations section.
Interoperability considerations: Defined by the ACP header, TLV-region, and payload structure.
Published specification: This document.
Intended usage: COMMON
Author/Change controller: IESG

7.7. Well-Known CoAP Resource

IANA is requested to register the following well-known URI suffix per [RFC8615] using the "Specification Required" policy:

URI Suffix	Description	Reference
muacp	Discovery resource indicating ACP support.	This document

Table 8

A CoAP GET to /.well-known/muacp SHOULD return a CBOR structure (Content-Format: application/cbor) describing supported TLVs, maximum sizes, and supported versions as specified in Section 10.5. A successful response (2.05 Content) MUST contain a CBOR map. A 4.04 Not Found response indicates that ACP is not supported by the device. Implementations MUST handle both success and error responses gracefully.

7.8. Summary of IANA Actions

IANA is requested to: create the ACP TLV Types registry (Expert Review) and populate initial values, create the ACP QoS Codes registry (Expert Review), create the ACP Verb Codes registry (Expert Review), create the ACP Error Codes registry (Expert Review), register the CoAP Content-Format application/muacp (Expert Review), register the media type application/muacp (Expert Review per RFC 6838), and register the well-known URI suffix muacp (Specification Required per RFC 8615). All Expert Review registries will be administered by designated experts appointed by IANA on advice of the ISE.

8. State Machines and Processing Logic

This section defines normative finite-state machines (FSMs) governing ACP conversations. Implementations MUST implement these FSMs for deterministic, interoperable behavior. Agents operate according to: receive message, validate OSCORE (if required), validate header/TLVs/payload, identify conversation via Correlation ID, execute verb-specific FSM transition, emit resulting messages. Agents MUST enforce a bounded maximum number of concurrent conversations determined by their resource profile (see Section 10). Agents MUST reject new conversations with ERR_RESOURCE_EXHAUSTED when resource limits are exceeded.

8.1. ASK/TELL Conversation State Machine

ASK initiates a conversation, TELL completes it. States: IDLE -> (send ASK) -> WAIT_RESP -> (recv TELL) -> COMPLETED -> cleanup. On timeout with QoS=1, the CoAP layer retransmits the CON request; after CoAP exhausts MAX_RETRANSMIT retransmissions without acknowledgment, ACP MUST transition to COMPLETED with ERR_TIMEOUT. On timeout with QoS=0 or QoS=2 (no retransmission), ACP MUST immediately transition to COMPLETED with ERR_TIMEOUT. Receiver MUST emit TELL(error) for protocol errors.

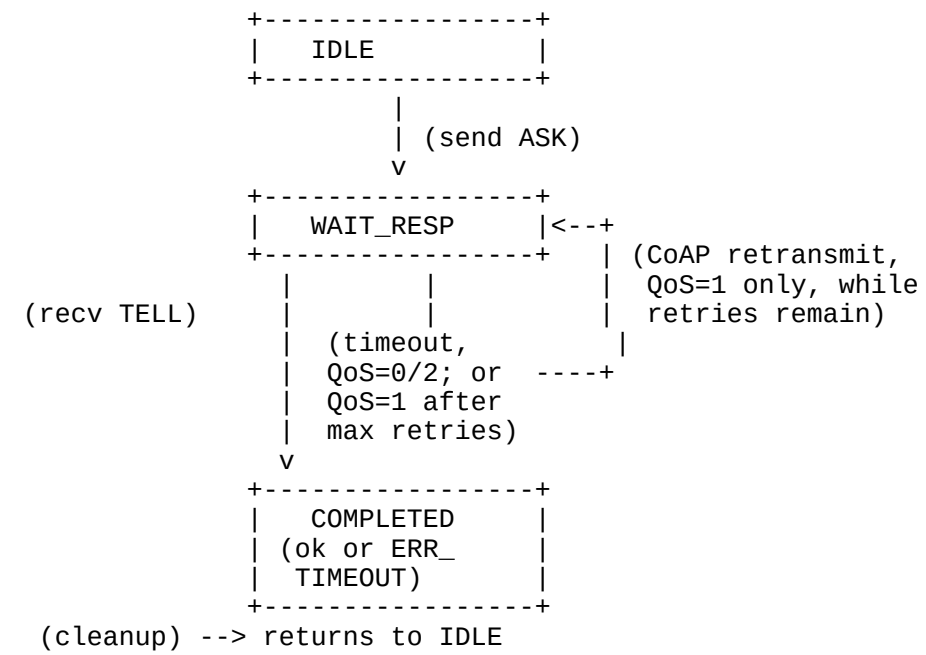


Figure 8: Figure 8: ASK/TELL State Machine

8.2. PING State Machine

PING serves as a minimal liveness check. PING is stateless and does NOT create persistent conversation table entries. States: IDLE -> (send PING) -> WAIT_PONG -> (recv TELL with matching Correlation ID, or timeout) -> COMPLETED. Responses are always Verb=TELL: OSCORE-protected TELL for OSCORE-protected PING; unprotected TELL for unencrypted PING (when permitted). Implementations MUST support OSCORE-protected PING and MAY support unencrypted PING for lightweight liveness detection. PING MUST NOT modify application state and MUST NOT cause retransmissions on timeout.

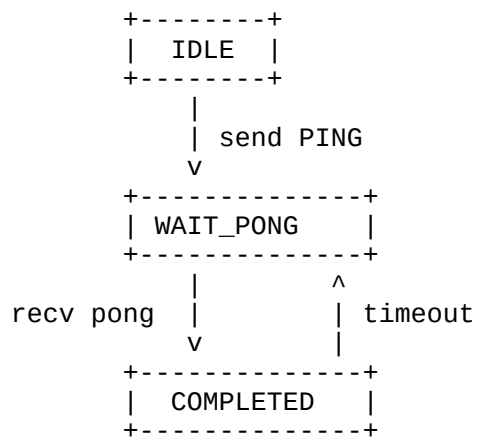


Figure 9: Figure 9: PING Liveness FSM

8.3. OBSERVE Subscription State Machine

OBSERVE establishes a long-lived subscription that persists and delivers multiple event-triggered notifications until cancelled or expired. The FSM resides on the publisher (receiver of OBSERVE). Each event trigger causes a TELL notification; the subscription returns to SUBSCRIBED and awaits the next event. Subscriptions MUST expire after the negotiated or default lifetime (see Section 4.4) and MUST enforce resource ceilings (max subscriptions per peer). A new OBSERVE on the same Correlation ID from the same peer resets the lifetime timer. Upon expiry, the publisher MUST free all subscription state and SHOULD notify the subscriber via a TELL(ERR_TIMEOUT).

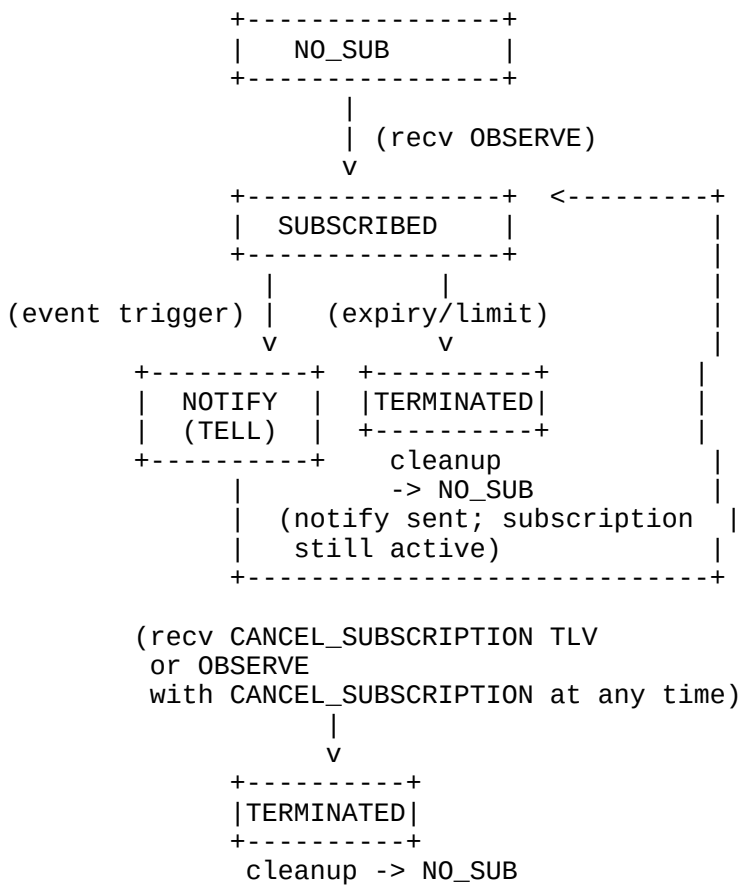


Figure 10: Figure 10: OBSERVE Subscription FSM (Publisher Side)

8.4. Error-State Transitions

Errors MUST transition FSMs to predictable termination states: ERR_MALFORMED causes discard with no state, ERR_UNSUPPORTED_TLV terminates the conversation and sends error TELL, ERR_TIMEOUT completes with error and frees resources, ERR_RESOURCE_EXHAUSTED rejects without new state, and OSCORE failure causes silent discard with no state update.

8.5. Processing Time and Resource Bounds

All FSM transitions MUST complete in bounded time and memory. Required limits: conversation table (a bounded number of entries, determined by the active interoperability profile), subscription table (a bounded number of entries, determined by the active interoperability profile), deterministic message buffer sizes (header plus bounded TLV region plus bounded payload), timers without per-message dynamic allocation. Numeric minimums are defined only in Section 10 (Interoperability Profiles). Platforms MAY use preallocated memory pools or static tables.

9. Security Considerations

This section defines the security properties, assumptions, and mandatory mitigations for ACP. The protocol relies on OSCORE and the underlying transport for security. All implementations MUST follow the requirements in this section to avoid exposure to denial-of-service, spoofing, downgrade, replay, or privacy attacks.

9.1. Threat Model

The ACP threat model assumes attackers may: passively eavesdrop, modify, inject, reorder, or replay messages, exhaust memory/CPU/storage/energy/subscription tables, desynchronize state, conduct traffic analysis, attempt version downgrades, exploit weak random number generation or incorrect OSCORE configuration. The protocol provides security *only* when implemented with OSCORE. Attackers are assumed to have full control of the transport layer but not of OSCORE-protected channels.

9.2. Authentication, Integrity, and Confidentiality

All ACP messages except unencrypted PING MUST be authenticated and integrity-protected using OSCORE. OSCORE provides peer authentication (when derived from EDHOC or provisioned credentials), integrity protection over header/TLVs/payload, replay protection, and request/response binding. Implementations MUST use a unique OSCORE security context per communicating peer. TELL, ASK, and OBSERVE messages MUST be encrypted via OSCORE. Authorization MUST be enforced before performing operations triggered by ASK or OBSERVE.

9.3. Replay Prevention and Freshness

ACP relies on OSCORE replay protection. Implementations MUST enable and correctly maintain OSCORE replay windows. Receivers SHOULD maintain a per-peer sliding window of recent Sequence IDs. Subscription-triggered notifications MUST validate freshness. Agents MUST reject delayed or reordered messages if OSCORE replay windows indicate a stale nonce.

9.4. Denial-of-Service and Resource Exhaustion

Implementations MUST enforce: maximum active conversations (determined by interoperability profile), maximum OBSERVE subscriptions (determined by interoperability profile), rate limits on PING and ASK, TLV region size limits (max 1024 bytes), payload size limits determined by the active profile (MIP default: 1024 bytes; INP: up to 65535 bytes), and static/preallocated memory pools. Numeric minimums are defined in Section 10 (Interoperability Profiles). When limits are exceeded, agents MUST return ERR_RESOURCE_EXHAUSTED or silently drop messages. CoAP-level DoS mitigation (exponential backoff, NON vs CON) MUST also be applied.

9.5. Subscription Security

OBSERVE and CANCEL_SUBSCRIPTION MUST be OSCORE-protected. Subscriptions MUST be bound to an authenticated OSCORE context. Correlation IDs MUST be unpredictable. Subscription deletion MUST require a valid CANCEL_SUBSCRIPTION from the same authenticated peer or timeout/resource exhaustion. Agents MUST reject subscription attempts exceeding resource ceilings.

9.6. Downgrade Protection

The highest mutually supported version MUST be used. Version negotiation MUST occur under OSCORE (except PING). Agents MUST reject messages advertising only unsupported versions and MUST NOT fall back silently to lower versions.

9.7. Key Management

Implementations MUST provide: secure key provisioning (EDHOC, PSK, or manufacturing-time injection), rotation of OSCORE master secrets, secure deletion of expired keys, protection against key reuse across peers, and protection against side-channel extraction. Compromise of OSCORE keys compromises all ACP security properties.

Key Rotation: OSCORE master secrets SHOULD be rotated periodically (e.g., time-based: 30-90 days, usage-based: after 2^32 messages, or event-based: upon compromise suspicion). Rotation procedures MUST preserve active conversations where possible.

9.8. Side-Channel Attacks

Constrained devices may be vulnerable to side-channel attacks (timing, power, electromagnetic). Implementations SHOULD: use constant-time cryptographic operations, minimize observable timing differences, protect against power analysis (HSMs or software countermeasures), avoid leaking information through error timing or resource allocation, use secure random number generators for Correlation IDs and Sequence IDs. While complete side-channel resistance may be impractical on severely constrained devices, implementations SHOULD document their threat model and mitigations.

9.9. Safe Failure Modes

Malformed messages MUST be discarded without modifying state. OSCORE failures MUST be silent and MUST NOT produce error messages usable for oracle attacks. Timeouts MUST clean up state deterministically. Subscription state MUST never persist without authenticated refresh.

10. Interoperability and Deployment Profiles

This section defines the minimum feature set required for interoperability between ACP implementations, along with deployment profiles tailored to different classes of devices and networks.

10.1. Minimum Interoperability Profile (MIP)

MIP defines the absolute floor: every conformant ACP implementation MUST satisfy MIP regardless of device class. All ACP implementations MUST support: the 64-bit header format (including the VER field), all four verbs (PING, TELL, ASK, OBSERVE), TLV processing with ordering and size limits, OSCORE/CoAP transport binding, Content-Format application/muacp, and error-handling and state-machine behavior as defined in this specification.

* Minimum concurrent conversations: 8
* Minimum concurrent subscriptions: 4
* Maximum payload size: 1024 bytes (senders MUST NOT send larger payloads to MIP peers without prior capability negotiation)
* Maximum TLV region: 1024 bytes

10.2. Constrained Node Profile (CNP)

CNP targets severely constrained microcontroller-class devices (Class 1: ~10 KB RAM / 100 KB flash; Class 2: ~50 KB RAM / 250 KB flash per [RFC7228]). CNP is a constrained implementation profile that still satisfies the MIP receive requirements. Implementations declaring CNP compliance MUST satisfy MIP and additionally MUST use static/preallocated buffers, minimize logging, and SHOULD prefer PSK/EDHOC-based OSCORE contexts.

* Minimum concurrent conversations: 8 (same as MIP minimum; CNP nodes SHOULD expose this limit via feature negotiation)

* Minimum concurrent subscriptions: 4 (same as MIP minimum)

* Maximum payload size accepted: 1024 bytes (same as MIP)

* Maximum TLV region accepted: 1024 bytes (same as MIP)

Note: CNP nodes MAY use smaller routine application payloads, for example 512-byte payloads and 256-byte TLV regions, as local sending defaults. Those local defaults MUST NOT be advertised as peer receive limits unless the implementation is explicitly operating outside MIP conformance.

10.3. Infrastructure Node Profile (INP)

INP targets edge gateways and cloud-side collectors. Implementations MUST support full subscription features, extended TLV sets, high-throughput replay windows, EDHOC key exchange, and rate-shaping for constrained peers. INP nodes MAY provide protocol translation and hardware-accelerated crypto.

* Minimum concurrent conversations: 64

* Minimum concurrent subscriptions: 16

* Maximum payload size: 65535 bytes

* Maximum TLV region: 1024 bytes

10.4. Cross-Profile Interoperability

When an INP node communicates with a CNP node, the INP node SHOULD discover the peer's capabilities via feature negotiation (Section 10.5) before sending large messages, and MUST NOT exceed the peer's advertised payload or TLV limits. CNP nodes MUST ignore unknown non-critical TLVs. MIP compliance is always the fallback: when no capability advertisement is available, all parties MUST assume MIP limits. All profile interactions MUST preserve security properties.

10.5. Feature Negotiation

Feature discovery uses GET /.well-known/muacp, returning a CBOR map describing the device's ACP capabilities. The response MUST use Content-Format application/cbor and MUST conform to the following CDDL schema:

```
muacp-capabilities = {
  ? "max-tlv-size" => uint,           ; Max TLV bytes
  ? "max-payload-size" => uint,       ; Max payload bytes
  ? "supported-tlv-types" => [*uint], ; TLV Type values
  ? "supported-versions" => [*uint],  ; Protocol versions
  ? "congestion-modes" => [*text],    ; Congestion modes
  ? "conversation-limit" => uint,     ; Max conversations
  ? "subscription-limit" => uint,     ; Max subscriptions
  ? "default-sub-lifetime" => uint,   ; Default subscription lifetime (seconds)
  ? "profile" => ("mip" / "cnp" / "inp"),  ; Profile identifier
}
```

Figure 11: Feature Negotiation Response Format (CDDL)

All fields are optional. If a field is omitted, implementations MUST assume the MIP minimum for that capability. Nodes SHOULD cache results until expiration or reboot. If the resource is unavailable (4.04 Not Found), implementations MUST assume MIP defaults: max-tlv-size=1024, max-payload-size=1024, conversation-limit=8, subscription-limit=4, default-sub-lifetime=86400, supported-versions=[0x00]. Senders MUST NOT transmit payloads or TLV regions exceeding the peer's advertised or assumed limits.

11. Wire Examples

This section provides essential normative examples of ACP messages. Additional test vectors are available in the reference implementation repository [MUACP-IMPL]. Byte order is network byte order (big-endian).

11.1. Minimal PING (unencrypted)

A minimal PING contains only the ACP header. The complete 64-bit header is:

```
00 01   # Sequence ID = 0x0001
00 01   # Correlation ID = 0x0001
00      # QoS = 0 (bits7:6), Verb = 0 (bits5:4), Flags = 0
00      # VER = 0 (bits7:4), Reserved = 0 (bits3:0)
00 00   # TLV Length = 0 bytes

Total: 8 bytes
```

Figure 12: Example 1: PING Message (Hex)

No TLVs, no payload. This message may be sent unencrypted over CoAP NON.

11.2. ASK/TELL over OSCORE

ASK messages are sent as CoAP POST requests with OSCORE protection. The unencrypted ACP ASK structure: Header (Sequence ID=0x0002, Correlation ID=0x0003, QoS=1, Verb=2), optional TLVs, optional payload. After OSCORE encryption, the complete ACP message becomes the CoAP payload. TELL responses use the same Correlation ID and are also OSCORE-protected.

Complete Example: The following shows a complete ASK/TELL exchange:

Step 1: ASK before OSCORE encryption:

Header (8 bytes):

```
00 02              # Sequence ID = 0x0002
00 03              # Correlation ID = 0x0003
60                 # QoS=1 (confirmable, bits7:6=01),
                   # Verb=2 (ASK), Flags=0
00                 # VER=0, Reserved=0
00 00              # TLV Length = 0 bytes
```

TLVs (none in this example):

```
[No TLVs]
```

Payload (CBOR-encoded request, 13 bytes):

```
A1                    # CBOR map(1): 1 key-value pair
66 61 63 74 69 6F 6E  # text(6): "action"
64 72 65 61 64        # text(4): "read"
```

Total ACP message:

8 bytes (header) + 0 bytes (TLVs) + 13 bytes (payload)

```
      = 21 bytes

  Step 2: CoAP POST with OSCORE (encrypted):
    CoAP Header: 44 02 7A 10  # CON, POST, MID=0x7A10
    CoAP Options:
      0B 6D 75 61 63 70  # Uri-Path: "muacp"
      [Content-Format: application/muacp; IANA value TBD]
      09 XX              # OSCORE Option

    OSCORE-Protected Payload (encrypted ACP message):
      [Ciphertext depends on OSCORE context]
      Note: the 21-byte message from Step 1 is encrypted here

  Step 3: TELL before OSCORE:
    Header (8 bytes):
      00 03              # Sequence ID = 0x0003
      00 03              # Correlation ID = 0x0003
      10                 # QoS=0, Verb=1 (TELL), Flags=0
      00                 # VER=0, Reserved=0
      00 03              # TLV Length = 3 bytes

    TLVs:
      22 01 00           # Error-Code TLV:
                         # Type=0x22, Len=1, Value=SUCCESS

    Payload (CBOR-encoded result, 10 bytes):
      A1                  # CBOR map(1): 1 key-value pair
      65 76 61 6C 75 65   # text(5): "value"
      F9 4D 60            # float16: 21.5
                          # 0x4D60 decodes to 21.5

  Step 4: CoAP Response with OSCORE (encrypted):
    CoAP Header: 64 44 7A 10  # ACK, 2.04 Changed, MID=0x7A10
    OSCORE-Protected Payload (encrypted ACP TELL):
      [Ciphertext depends on OSCORE context]
```

Figure 13: Example 2: Complete ASK/TELL Exchange

Complete hexdumps of encrypted payloads with actual OSCORE ciphertext are provided in the reference implementation repository, as they depend on specific OSCORE security contexts, nonces, and key material.

Note: The Content-Format numeric option value is intentionally omitted from this example because it will be assigned by IANA during the IESG review process. Conformant implementations MUST use the assigned value for application/muacp.

12. Conformance Checklist

This section summarizes the conformance points that implementations need to satisfy. Compliance is determined by the normative requirements in this specification; external test suites and the reference implementation repository [MUACP-IMPL] are informative aids and do not define additional normative requirements.

A conformant implementation MUST satisfy the normative requirements in the following categories:

* *Header and TLV Encoding:* Correct parsing of all header fields, TLV ordering, size limits (1024 bytes TLV region, 65535 bytes payload), and handling of unknown TLVs.

* *Parser Robustness:* Safe handling of malformed headers, oversized payloads, invalid TLV lengths, and resource exhaustion conditions.

* *State-Machine Behavior:* ASK/TELL conversation lifecycle, OBSERVE subscription management, PING statelessness, and deterministic error transitions.

* *OSCORE Security:* Authentication and decryption of protected messages, replay window enforcement, context isolation, and downgrade protection.

* *Resource Constraints:* Conversation table limits, subscription limits, and bounded processing time (minimums defined per interoperability profile in Section 10).

* *Interoperability:* Successful message exchange between independent implementations under the Minimum Interoperability Profile.

Test vectors can help demonstrate these properties, but a test suite cannot weaken, replace, or add to the requirements in this document.

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014, <https://www.rfc-editor.org/rfc/rfc7252>.

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

[RFC8613] Selander, G., Mattsson, J., and T. Fossati, "OSCORE: Object Security for Constrained RESTful Environments", RFC 8613, April 2019, <https://www.rfc-editor.org/rfc/rfc8613>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, September 2015, <https://www.rfc-editor.org/rfc/rfc7641>.

[RFC7959] Bormann, C. and Z. Shelby, "Blockwise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, August 2016, <https://www.rfc-editor.org/rfc/rfc7959>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

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

[RFC9528] Selander, G., Mattsson, J., and M. Furuhed, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", RFC 9528, March 2024, <https://www.rfc-editor.org/rfc/rfc9528>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014, <https://www.rfc-editor.org/rfc/rfc7228>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996, <https://www.rfc-editor.org/rfc/rfc1982>.

13.2. Informative References

[FIPA-ACL] (FIPA), F. F. I. P. A., "ACL Message Structure Specification", 1997, <https://www.fipa.org/specs/fipa00061/>.

[RFC9176] Amsss, C., Shelby, Z., Koster, M., Bormann, C., and P. V. D. Stok, "Constrained RESTful Environments (CoRE) Resource Directory", RFC 9176, April 2022, <https://www.rfc-editor.org/rfc/rfc9176>.

[MUACP] Mallick, A. and I. Chebolu, "μ ACP: A Formal Calculus for Expressive, Resource-Constrained Agent Communication", Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems AAMAS 2026, DOI 10.65109/PHRW6922, arXiv 2601.00219, 2026, <https://doi.org/10.65109/PHRW6922>.

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

[MUACP-IMPL] Mallick, A., "ACP Reference Implementation", GitHub Repository arnab-m1/miuACP, 2025, <https://github.com/arnab-m1/miuACP>.

Open Questions for Working Group Discussion

The following design choices are highlighted for working group review and consensus-building:

1. *Unencrypted PING:* Should unencrypted PING be mandatory-to-implement (for maximum interoperability), optional (current specification), or prohibited (for maximum security)? The current draft specifies MAY to balance lightweight liveness detection with security concerns.

2. *Resource Minimums:* Are the profile-specific resource limits (MIP: 8/4, CNP: 8/4, INP: 64/16 conversations/subscriptions) appropriate for the target device classes? Should additional profiles be defined?

3. *OBSERVE vs RFC 7641:* Should ACP OBSERVE semantics be more closely aligned with CoAP Observe (RFC 7641), or maintain conversation-centric subscription management? What are the tradeoffs between resource-centric and conversation-centric approaches?

4. *Transport Bindings:* Should additional transport bindings (e.g., DTLS/UDP without CoAP, QUIC) be standardized, or should CoAP/OSCORE remain the only MTI (mandatory-to-implement) binding?

5. *Interoperability Testing:* What interoperability events or conformance test suites are needed to validate independent implementations? Should ACP participate in existing IoT plugfests?

The authors welcome feedback on these and all other aspects of the specification. The authors plan to submit this work to the CoRE Working Group for consideration and to seek ACE review for the OSCORE and authorization aspects.

Acknowledgments

The design of ACP benefited from feedback across multiple research and engineering communities working on IoT systems, multi-agent communication, and distributed protocol design.

The authors thank the early reviewers who provided detailed feedback on the wire format, TLV design, and OSCORE integration. Special thanks to the contributors to the open-source reference implementation who identified edge cases in the state machine implementations and provided interoperability testing reports.

The authors acknowledge participants from the IETF CoRE and ACE working groups whose prior work on OSCORE, EDHOC, and constrained-device protocols informed the security architecture of ACP. Discussions at IETF hackathons and CoAP plugfests helped refine the transport binding specification.

Feedback from researchers working on formal verification of constrained protocols influenced the deterministic resource bounds and finite-state machine specifications in this document.

This work is an individual contribution and does not represent the views of any organization or government entity.

Authors’ Addresses

Arnab Mallick
Centre for Development of Advanced Computing (CDAC)
Hyderabad
India
Email: arnabm@cdac.in

Indraveni Chebolu
Centre for Development of Advanced Computing (CDAC)
Hyderabad
India
Email: indravenik@cdac.in