Independent Submission
Internet-Draft
Intended status: Experimental
Expires: 29 October 2026

M.J. Arcan
Arcan Consulting

27 April 2026

The MyClerk Protocol: Tiered Security Communication for Distributed Family Systems
draft-myclerk-protocol-03

Abstract

This document specifies the MyClerk Protocol, a tiered-security communication protocol designed for distributed family orchestration systems. The protocol provides six security tiers ranging from 1-byte minimal overhead for tunneled messages to 144-byte full security for critical operations. It supports multiple transport mechanisms including NATS, Matrix, WebSocket, and direct TCP, while maintaining end-to-end encryption using ChaCha20-Poly1305 AEAD with hybrid post-quantum key establishment combining ML-KEM-768 and X25519. A classical-only mode (X25519 without ML-KEM) remains available for resource-constrained devices and legacy interoperability.

The protocol is transport-agnostic, federation-capable, and optimized for environments ranging from resource-constrained IoT devices to full-featured desktop clients.

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 October 2026.

Table of Contents

1. Introduction

Modern families operate across multiple locations and devices: a primary home with network-attached storage, a vacation house with a mini-PC, grandparents with a Raspberry Pi, and mobile devices requiring access while traveling. The MyClerk Protocol addresses the communication requirements of such distributed family systems.

Traditional protocols impose significant overhead that becomes problematic for constrained channels. The MyClerk Protocol introduces tiered security levels, allowing applications to select appropriate overhead based on the security requirements of each operation.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Terminology

Node A device participating in a MyClerk deployment, such as a server, desktop client, mobile device, or IoT device.

Core Node A node classified as stable and reliable, expected to maintain high availability (>95% uptime over 30 days).

Bonus Node A node with variable availability, used opportunistically for additional redundancy.

Family A group of users and nodes sharing a common trust domain and cryptographic key hierarchy.

Federation The interconnection of multiple families for resource sharing or communication.

Tier A security level (0-5) determining the header structure and cryptographic protections applied to a message.

Session A stateful connection between two endpoints with established cryptographic keys.

2. Protocol Overview

The MyClerk Protocol is a binary protocol using MessagePack [RFC8949] for payload encoding. It defines six security tiers with increasing header sizes and cryptographic protections.

2.1. Design Goals

1. *Tiered Security:* 1-144 bytes overhead depending on security requirements.

2. *Transport Agnostic:* Operates over NATS, Matrix, WebSocket, TCP, or other transports.

3. *Federation Ready:* Supports multi-server, multi-family deployments.

4. *Future Proof:* Extensible operations and feature negotiation.

5. *Post-Quantum Security:* Default key establishment for Tier 4 and Tier 5 is resistant to attacks by cryptographically relevant quantum computers (CRQCs), using a hybrid of ML-KEM-768 [FIPS203] and X25519 [RFC7748], while retaining a classical-only mode for resource-constrained devices.

2.2. Default Port

When using TCP or UDP as the transport protocol, implementations SHOULD use port 5657 as the default listening port. This port is registered with IANA for the "myclerk" service (see Section 11.1).

Implementations MAY use alternative ports when:

* Operating behind a reverse proxy or load balancer

* Running multiple instances on the same host

* Required by local network policies

Service discovery mechanisms (such as mDNS or DNS-SD) SHOULD advertise the actual port in use, regardless of whether it matches the default.

2.3. Security Tiers Overview

Tier	Header	Encryption	Auth	KEX	Use Case
0	1 B	None (tunneled)	None	--	Inside session
1	4 B	None	None	--	Fire-and-forget
2	6 B	Optional	CRC-16	--	Home automation
3	12 B	ChaCha20-Poly1305	HMAC-32	from T4/T5	Conversational
4	14 B	ChaCha20-Poly1305	HMAC-64	hybrid	Key exchange

5	14 B+tag	ChaCha20-Poly1305	HMAC-256	hybrid	Max security

Table 1: Security Tier Summary

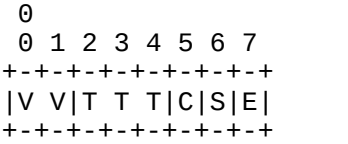
KEX abbreviations: "hybrid" denotes ML-KEM-768 + X25519 hybrid key establishment; "from T4/T5" indicates the session key is derived in a separate Tier 4 or Tier 5 handshake and inherited by subsequent Tier 3 traffic.

3. Message Format

All messages consist of a header, optional payload, and optional trailer. The header format varies by security tier.

3.1. Common Header Fields

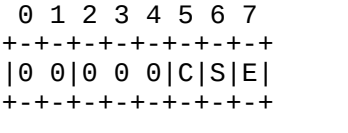
The first byte (Flags) is present in all tiers and has the following structure:



V: Protocol Version (2 bits) - 0 or 1
T: Security Tier (3 bits) - Values 0-5
C: Compressed (1 bit) - Payload is compressed
S: Stream (1 bit) - Server-push stream message
E: Encrypted (1 bit) - Payload is encrypted

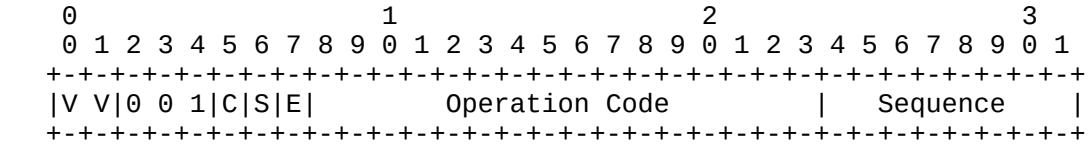
3.2. Tier 0 Header (1 byte)

Tier 0 is used for messages tunneled inside an already-secure session. It provides minimal overhead.



Tier 0 messages MUST only be sent within an established Tier 3+ session. Implementations receiving a Tier 0 message outside of a secure session MUST discard it.

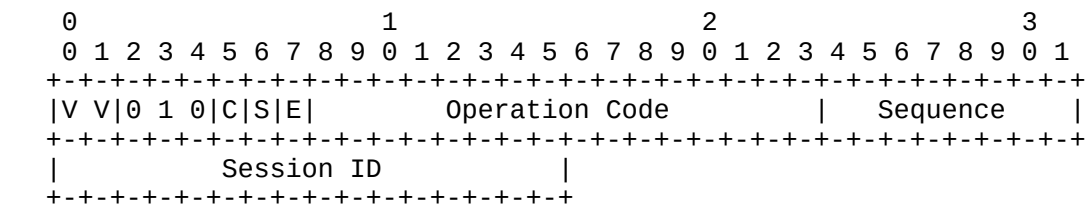
3.3. Tier 1 Header (4 bytes)



Operation Code (16 bits) Identifies the operation. See Section 6.

Sequence (8 bits) Message sequence number, wrapping at 255.

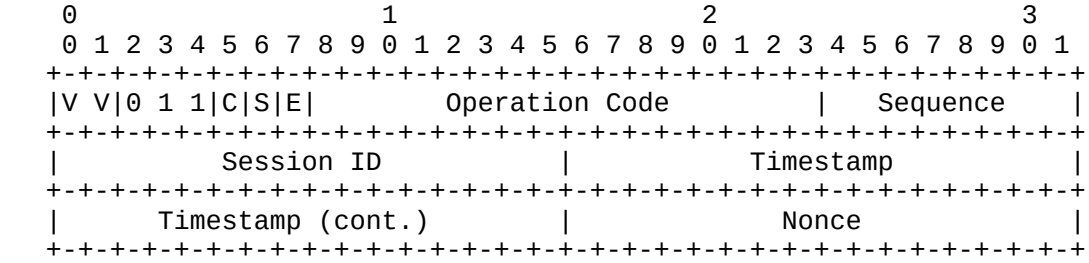
3.4. Tier 2 Header (6 bytes)



Session ID (16 bits) Identifies the session context for this message.

Tier 2 messages SHOULD include a CRC-16 trailer for error detection.

3.5. Tier 3 Header (12 bytes)



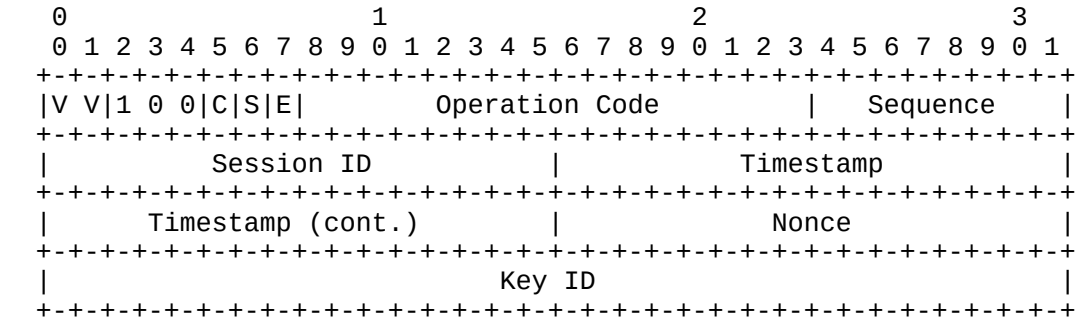
Timestamp (32 bits) Unix timestamp in seconds. Used for replay protection.

Nonce (16 bits) Random nonce component. Combined with timestamp and counter to form the full 96-bit nonce for ChaCha20-Poly1305.

Tier 3 messages with the E flag set MUST be encrypted using ChaCha20-Poly1305 as specified in [RFC8439].

3.6. Tier 4 Header (14 bytes)

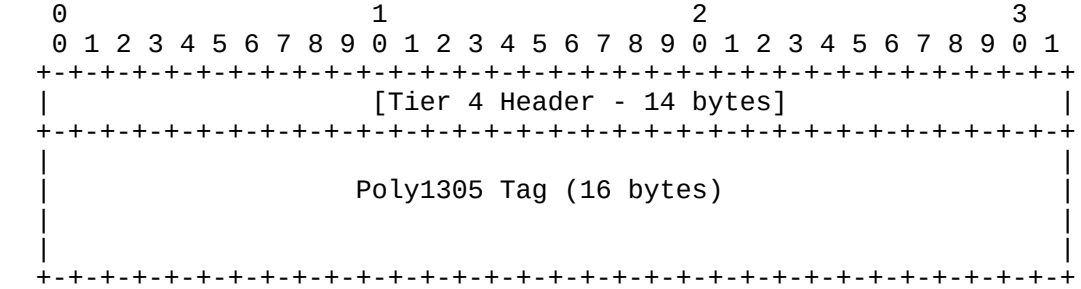
Tier 4 carries the messages that establish a session (SESSION_INIT, SESSION_ACK). Earlier drafts of this document carried the 32-byte X25519 ECDH public key inline in a 42-byte header. With the move to a hybrid post-quantum KEX the key material grew to 1184 bytes (ML-KEM-768 encapsulation key) plus 1088 bytes (ML-KEM-768 ciphertext) plus the existing 32 bytes of X25519 per direction, which no longer fits in any sensibly-sized fixed header. Consequently, Tier 4 headers no longer carry KEX material inline; all public keys and ciphertexts are transported in the SESSION_INIT / SESSION_ACK payloads (see Section 7.1), and the header retains only session and nonce identifiers.



Key ID (32 bits) Identifier for the session key being used or negotiated. Zero during the initial SESSION_INIT before a key has been established.

3.7. Tier 5 Header (14 bytes + Poly1305 Tag)

Tier 5 uses the same 14-byte header as Tier 4 but appends a full Poly1305 authentication tag after the header (before the payload):



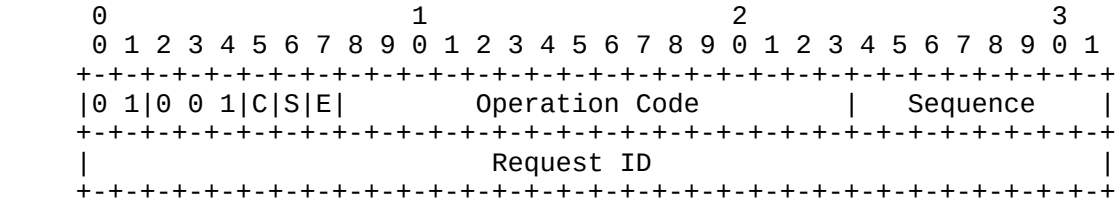
Prior drafts of this document mandated an additional HMAC-SHA256 trailer on Tier 5 as a post-quantum resilience workaround against potential future quantum attacks on Poly1305. With hybrid ML-KEM-768 key establishment now baked into Tier 4/5 (see Section 5), that trailer is redundant and has been removed. Implementations conforming to this draft MUST NOT emit an HMAC-SHA256 trailer on Tier 5 messages.

3.8. Request Correlation (Version 1)

Version 0 of the protocol correlates responses to requests by Operation Code. This limits each connection to one outstanding request per Operation Code, preventing parallel requests for the same operation (e.g., multiple concurrent thumbnail fetches).

Version 1 extends each tier-specific header with a 32-bit Request ID field, inserted after the tier-specific fields and before the payload. The Version field in the Flags byte (bits 0-1) distinguishes V0 from V1.

Example: Tier 1 Version 1 Header (8 bytes):



Header sizes by version:

Tier	V0	V1	Delta
0	1 byte	5 bytes	+4
1	4 bytes	8 bytes	+4
2	6 bytes	10 bytes	+4
3	12 bytes	16 bytes	+4
4	42 bytes	46 bytes	+4
5	58 bytes	62 bytes	+4

Table 2: Header Sizes: Version 0 vs Version 1

3.8.1. Request ID Semantics

Format 32-bit unsigned integer, big-endian byte order.

Generation Client-generated, monotonically increasing per connection.

Echo The server MUST echo the Request ID from the request in its response.

Fire-and-Forget A Request ID of 0x00000000 indicates fire-and-forget; no response correlation is expected.

Wrap When the counter reaches 0xFFFFFFFF, it MUST wrap to 0x00000001, skipping zero.

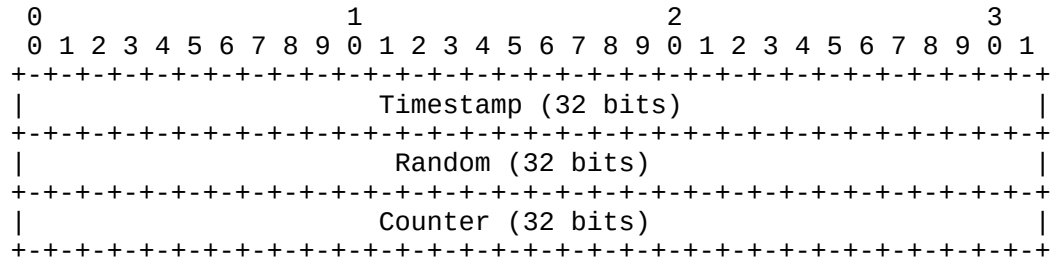
3.8.2. Backward Compatibility

Servers MUST support Version 0 and Version 1 simultaneously. The version is determined by bits 0-1 of the Flags byte. A response MUST use the same version as the request that triggered it.

Clients that support Version 1 SHOULD set their version bits to 01 and include a Request ID in every request. When communicating with a Version 0 server that ignores the Request ID, the client MUST fall back to Operation Code correlation.

4. Nonce Construction

ChaCha20-Poly1305 requires a 96-bit (12-byte) nonce that MUST NOT be reused with the same key. The MyClerk Protocol constructs nonces as follows:



Timestamp (32 bits) Unix epoch seconds. Provides cross-session replay protection.

Random (32 bits) Cryptographically random value generated per session using a CSPRNG. Provides 2^32 possible values per second.

Counter (32 bits) Per-message counter starting at 0, incremented for each message. Allows 2^32 messages per session.

All fields are encoded in big-endian byte order.

This construction provides a collision probability of less than 2^-80 per year at 1 million operations per second, assuming proper CSPRNG implementation.

5. Key Derivation

Session keys are derived using HKDF as specified in [RFC5869] with SHA-256 as the hash function.

5.1. Hybrid Key Exchange

Tier 4+ sessions establish a shared symmetric key using a hybrid construction that combines classical X25519 ECDH [RFC7748] with post-quantum ML-KEM-768 [FIPS203]. The hybrid output is secure as long as at least one of the two primitives remains unbroken: an adversary that eventually breaks X25519 (e.g. by running Shor’s algorithm on a cryptographically relevant quantum computer) still has to defeat ML-KEM, and vice versa.

ML-KEM-768 (NIST category 3, approximately AES-192-equivalent security) is the default parameter set. ML-KEM-1024 MAY be offered by implementations requiring higher security margin; ML-KEM-512 is NOT RECOMMENDED for new deployments.

The initiator generates an ephemeral X25519 key pair and an ephemeral ML-KEM-768 decapsulation key pair, and sends both public keys in SESSION_INIT (see Section 7.1). The responder generates its own ephemeral X25519 key pair, encapsulates against the initiator's ML-KEM encapsulation key to obtain the PQ shared secret and its ciphertext, and returns the X25519 public key together with the ML-KEM ciphertext in SESSION_ACK.

Both sides then compute the combined shared secret:

```
ss_classical = X25519(local_x25519_priv, remote_x25519_pub)

; Responder: (ct, ss_pq) = ML-KEM-768.Encapsulate(initiator_mlkem_pub)
; Initiator: ss_pq       = ML-KEM-768.Decapsulate(local_mlkem_priv, ct)

combined_secret = ss_classical || ss_pq   ; 32 + 32 = 64 bytes

transcript_hash = SHA-256(
    SESSION_INIT_bytes  ||
    SESSION_ACK_bytes
)

session_key = HKDF-SHA256(
    IKM  = combined_secret,
    salt = nonce_initiator || nonce_responder,
    info = "myclerk-session-v1-hybrid" || transcript_hash,
    L    = 32
)
```

The concatenation order ss_classical || ss_pq is fixed and normative. Implementations that swap the order derive a different session key and will fail to communicate; this is a deliberate fail-closed property, not a bug.

The transcript_hash folded into the HKDF info parameter binds the session key to the exact bytes of the handshake as observed by the deriving endpoint. An on-path attacker who modifies any handshake field -- stripping a capability, downgrading kex-mode, swapping nonces, altering any declared parameter -- causes the two endpoints to derive different session keys, and the first post-handshake Tier 3 or Tier 5 frame fails its AEAD check. See Section 10.6 for the full security argument.

5.2. Classical-only Key Exchange (Legacy / Constrained)

Implementations that cannot afford the ~2.3 KB of wire overhead of the hybrid handshake (e.g. ESP32-class devices on low-bandwidth modem links, or legacy clients predating this draft) MAY negotiate classical-only mode by omitting the ML-KEM fields from SESSION_INIT and signalling kex-mode: classical-only. The responder MUST either accept and mirror classical-only in SESSION_ACK, or refuse the session if local policy requires PQ.

In classical-only mode the derivation reduces to:

```
combined_secret = X25519(local_x25519_priv, remote_x25519_pub)

transcript_hash = SHA-256(SESSION_INIT_bytes || SESSION_ACK_bytes)

session_key = HKDF-SHA256(
    IKM  = combined_secret,
    salt = nonce_initiator || nonce_responder,
    info = "myclerk-session-v1-classical" || transcript_hash,
    L    = 32
)
```

The info string intentionally differs between hybrid and classical-only derivations so that a key produced in one mode cannot be mistaken for a key in the other mode -- a domain-separation precaution against cross-protocol attacks.

Classical-only sessions MUST NOT be used for data requiring long-term confidentiality (conservatively: more than 10 years), because an attacker who records the traffic today and gains access to a cryptographically relevant quantum computer later can recover the session key. Implementations SHOULD log classical-only negotiations for audit purposes.

5.3. Key Rotation

Keys MUST be rotated after any of the following conditions:

* 2^32 messages sent (nonce exhaustion)

* 24 hours elapsed (time-based)

* Explicit KEY_ROTATE operation received

Rotated keys are derived as:

```
new_key = HKDF-SHA256(
    IKM  = current_key,
    salt = "rotate",
    info = rotation_counter (4 bytes, big-endian),
    L    = 32
)
```

6. Protocol Operations

Operations are identified by a 16-bit operation code. The operation space is divided into ranges:

Range	Category
0x0000-0x00FF	Core Operations (Session, Key Management)
0x0100-0x01FF	Standard Operations (Device, Identity, Messaging)
0x0200-0x02FF	Resource Sharing (Streams, GPU, Routing)
0x0300-0x03FF	Federation
0x0400-0x04FF	Billing and Economics
0x0500-0x05FF	Virtual File System (VFS)
0x0600-0x06FF	Hardware Passthrough (USB, GPIO, I2C, SPI, CAN)
0x0700-0x07FF	Knowledge Base
0x0800-0x08FF	Settings & Configuration
0x0900-0x09FF	Room/Channel Management
0x0A00-0x0AFF	Media Handling
0x0B00-0x0BFF	Telephony (Voice, Video, SMS, IVR)
0x0C00-0x0CFF	Calendar & Events
0x0D00-0x0DFF	Tasks, Shopping & Gamification
0x0E00-0x0EFF	Smart Home Devices
0x0F00-0x0FFF	Automation Rules

0x1000-0x10FF	Maya AI Assistant
0x1100-0x11FF	Synchronization & VFS Federation
0x1200-0x12FF	Orchestration (Notifications, Finance)
0x1300-0x13FF	Presence & Status (Education)
0x1400-0x14FF	Health (Social, Community)
0x1500-0x15FF	Location & Mobility
0x1600-0x16FF	Debug, Testing & Plugins
0x1700-0x17FF	Mobile Device Management (MDM)
0x1800-0x18FF	Legal Documents & Care Directives
0x1900-0x19FF	Mobility & Vehicle Management
0x1A00-0x1AFF	Travel & Vacation
0x1B00-0x1BFF	Audit & Compliance
0x1C00-0x1CFF	Audio Pipeline
0x1D00-0x1DFF	Desktop Client
0x1E00-0xEFFF	Unassigned (reserved for future standard categories)
0xF000-0xFFFE	Vendor Extensions (third-party/vendor-specific use only)
0xFFFF	Reserved

Table 3: Operation Code Ranges

6.1. Core Operations (0x0000-0x00FF)

6.1.1. Session Management (0x0000-0x000F)

Code	Name	Description
0x0000	NOP	No operation
0x0001	KEEPALIVE	Session keepalive
0x0002	KEEPALIVE_ACK	Keepalive acknowledgment
0x0003	SESSION_INIT	Initialize session
0x0004	SESSION_ACK	Session acknowledgment
0x0005	SESSION_CLOSE	Close session
0x0006	SESSION_CLOSE_ACK	Close acknowledgment
0x0007	SESSION_RESUME	Resume session with ticket
0x0008	SESSION_RESUMED	Session resumed

Table 4: Session Management Operations

6.1.2. Key Management (0x0010-0x001F)

Code	Name	Description
0x0010	KEY_EXCHANGE_INIT	Initiate key exchange
0x0011	KEY_EXCHANGE_RESPONSE	Key exchange response
0x0012	KEY_EXCHANGE_COMPLETE	Key exchange complete
0x0016	SESSION_ROTATE	Rotate session key
0x0017	SESSION_REVOKE	Revoke session key

Table 5: Key Management Operations

6.2. Standard Operations (0x0100-0x01FF)

Standard Operations cover authentication, device registration, messaging, and identity management. The 0x0100-0x018F range is defined for authentication, key exchange, sessions, device registration, backup codes, and messaging. The 0x0190-0x01EF range is allocated to identity management operations.

6.2.1. Identity Management (0x0190-0x01EF)

Identity Management operations handle user lifecycle, profile management, authentication credentials, and two-factor authentication. All Identity Management operations MUST be sent at Tier 3 (Encrypted) or higher.

6.2.1.1. User Identity (0x0190-0x019F)

Code	Name	Description
0x0190	USER_CREATE	Create user account
0x0191	USER_GET	Get user information
0x0192	USER_UPDATE	Update user account
0x0193	USER_DELETE	Delete user account
0x0194	USER_LIST	List users
0x0195	USER_SEARCH	Search users

Table 6: User Identity Operations

6.2.1.2. User Moderation (0x01A0-0x01AF)

Code	Name	Description
0x01A0	USER_BLOCK	Block user
0x01A1	USER_UNBLOCK	Unblock user
0x01A2	USER_MUTE	Mute user
0x01A3	USER_UNMUTE	Unmute user

Table 7: User Moderation Operations

6.2.1.3. Profile and Avatar (0x01B0-0x01BF)

Code	Name	Description
0x01B0	PROFILE_GET	Get user profile
0x01B1	PROFILE_UPDATE	Update user profile
0x01B2	AVATAR_SET	Set user avatar
0x01B3	AVATAR_GET	Get user avatar

Table 8: Profile and Avatar Operations

6.2.1.4. Preferences (0x01C0-0x01CF)

Code	Name	Description
0x01C0	PREFERENCES_GET	Get user preferences
0x01C1	PREFERENCES_SET	Set user preferences

Table 9: Preferences Operations

6.2.1.5. Password and Email (0x01D0-0x01DF)

Code	Name	Description
0x01D0	PASSWORD_CHANGE	Change password
0x01D1	PASSWORD_RESET	Reset password
0x01D2	EMAIL_CHANGE	Change email address
0x01D3	EMAIL_VERIFY	Verify email address

Table 10: Password and Email Operations

PASSWORD_CHANGE requires the current password in the payload. PASSWORD_RESET initiates an out-of-band reset flow. EMAIL_CHANGE triggers a verification email to the new address; the change is not effective until EMAIL_VERIFY is completed.

6.2.1.6. TOTP Two-Factor Authentication (0x01E0-0x01EF)

Code	Name	Description
0x01E0	TOTP_ENABLE	Enable TOTP for account
0x01E1	TOTP_DISABLE	Disable TOTP for account
0x01E2	TOTP_VERIFY	Verify TOTP code

Table 11: TOTP Operations

TOTP_ENABLE returns a shared secret and provisioning URI compatible with RFC 6238. The client MUST verify the secret by submitting a valid TOTP code via TOTP_VERIFY before the enablement is finalized. TOTP_DISABLE requires a valid TOTP code or backup code for confirmation.

6.3. Resource Sharing Operations (0x0200-0x02FF)

Resource Sharing operations enable devices to expose and consume resources across the network. These operations support video, audio, HID, and GPU services.

6.3.1. Device Management (0x0200-0x020F)

Code	Name	Description
0x0200	DEVICE_LIST	List available devices
0x0201	DEVICE_INFO	Get device information
0x0202	DEVICE_SUBSCRIBE	Subscribe to device events
0x0203	DEVICE_UNSUBSCRIBE	Unsubscribe from device
0x0204	DEVICE_LOCK	Lock device for exclusive use
0x0205	DEVICE_UNLOCK	Release device lock

0x0206	DEVICE_CONFIGURE	Configure device parameters
0x0207	DEVICE_CAPABILITIES	Query device capabilities
0x020B	DEVICE_DISCOVER	List all connected devices (including unshared)

Table 12: Device Management Operations

DEVICE_LIST (0x0200) returns only devices currently being shared. DEVICE_DISCOVER (0x020B) returns all connected devices regardless of sharing status, enabling provisioning workflows.

6.3.2. Stream Operations (0x0210-0x021F)

Code	Name	Description
0x0210	STREAM_START	Start data stream
0x0211	STREAM_STOP	Stop data stream
0x0212	STREAM_DATA	Stream data payload
0x0213	STREAM_CONFIGURE	Configure stream parameters
0x0214	STREAM_QUALITY	Adjust quality settings
0x0215	STREAM_PAUSE	Pause stream
0x0216	STREAM_RESUME	Resume paused stream

Table 13: Stream Operations

Stream types are indicated in the STREAM_START payload:

* 0x01: Video (H.264, MJPEG)
* 0x02: Audio (Opus, PCM)
* 0x03: HID (evdev events)
* 0x04: Serial (byte stream)

6.3.3. GPU Service Operations (0x0220-0x022F)

GPU operations enable queue-based access to inference services (LLM, TTS, STT, Image Generation).

Code	Name	Description
0x0220	GPU_REQUEST	Submit GPU request
0x0221	GPU_RESPONSE	GPU response (streaming)
0x0222	GPU_STATUS	Query service status
0x0223	GPU_CANCEL	Cancel pending request
0x0224	GPU_QUEUE_INFO	Queue position and ETA

Table 14: GPU Service Operations

GPU service types:

* 0x01: LLM (Large Language Model)
* 0x02: TTS (Text-to-Speech)
* 0x03: STT (Speech-to-Text)
* 0x04: ImageGen (Image Generation)

6.3.4. Dynamic Routing and ACL (0x0240-0x024F)

Dynamic routing enables Hub-controlled device assignment and per-client access control. A Hub can redirect device streams between clients dynamically.

Code	Name	Description
0x0240	ROUTE_CREATE	Create device route
0x0241	ROUTE_DELETE	Delete device route
0x0242	ROUTE_LIST	List active routes
0x0243	ROUTE_MODIFY	Modify existing route
0x0244	ACL_SET	Set access control
0x0245	ACL_GET	Get access control
0x0246	ACL_GRANT	Grant client access
0x0247	ACL_REVOKE	Revoke client access
0x0248	ROUTE_PRIORITY	Set routing priority

Table 15: Dynamic Routing Operations

Access modes:

* 0x00: None (no access)

* 0x01: Read-only

* 0x02: Write-only

* 0x03: Read-write

* 0x04: Exclusive (single client)

6.3.5. Client Assignment Operations (0x0250-0x025F)

Client Assignment operations enable Hub-managed device provisioning. A Hub can assign devices to clients, and servers can request approval for sharing new devices through a token-based workflow.

Code	Name	Description
0x0250	CLIENT_REGISTER	Register client with server
0x0251	CLIENT_HEARTBEAT	Client keepalive
0x0252	CLIENT_ASSIGN	Assign device to client
0x0253	CLIENT_REVOKE	Revoke device from client
0x0254	CLIENT_STATUS	Query client status
0x0255	CLIENT_LIST	List registered clients
0x0256	CLIENT_INFO	Get client information
0x0257	DEVICE_REQUEST_CREATE	Create device sharing request
0x0258	DEVICE_REQUEST_LIST	List pending device requests
0x0259	DEVICE_REQUEST_DETAILS	Get request details
0x025A	DEVICE_REQUEST_RESPOND	Approve/reject device request
0x025B	DEVICE_REQUEST_CANCEL	Cancel pending request

Table 16: Client Assignment Operations

Device Request Workflow:

1. Hub queries server with DEVICE_DISCOVER (0x020B) to list available devices

2. Hub administrator creates request via DEVICE_REQUEST_CREATE (0x0257)

3. Server queries pending requests via DEVICE_REQUEST_LIST (0x0258)

4. Server retrieves details via DEVICE_REQUEST_DETAILS (0x0259)

5. Server administrator approves/rejects via DEVICE_REQUEST_RESPOND (0x025A)

Request states:

* 0x00: Pending (awaiting server approval)

* 0x01: Approved (all devices accepted)

* 0x02: Partially Approved (subset of devices accepted)

* 0x03: Rejected (request denied)

* 0x04: Expired (token timeout)

* 0x05: Cancelled (hub cancelled request)

6.4. Hardware Passthrough Operations (0x0600-0x06FF)

Hardware Passthrough operations enable low-level access to hardware interfaces. Unlike service-based resource sharing, these operations provide direct protocol-level access to hardware buses.

6.4.1. USB Operations (0x0600-0x060F)

USB passthrough virtualizes USB at the URB (USB Request Block) level, allowing any USB device to appear as natively connected on the client.

Code	Name	Description
0x0600	USB_DEVICE_LIST	List USB devices
0x0601	USB_DEVICE_ATTACH	Attach USB device
0x0602	USB_DEVICE_DETACH	Detach USB device
0x0603	USB_CONTROL_TRANSFER	USB control transfer
0x0604	USB_BULK_TRANSFER	USB bulk transfer
0x0605	USB_INTERRUPT_TRANSFER	USB interrupt transfer
0x0606	USB_ISOCHRONOUS_TRANSFER	USB isochronous transfer
0x0607	USB_GET_DESCRIPTOR	Get USB descriptor
0x0608	USB_SET_CONFIGURATION	Set USB configuration
0x0609	USB_SET_INTERFACE	Set USB interface
0x060A	USB_CLEAR_HALT	Clear endpoint halt
0x060B	USB_RESET	Reset USB device

Table 17: USB Operations

6.4.2. Serial Operations (0x0610-0x061F)

Serial passthrough supports RS-232, RS-485, and UART interfaces.

Code	Name	Description
0x0610	SERIAL_PORT_LIST	List serial ports
0x0611	SERIAL_PORT_OPEN	Open serial port
0x0612	SERIAL_PORT_CLOSE	Close serial port
0x0613	SERIAL_PORT_CONFIGURE	Configure baud, parity, etc.
0x0614	SERIAL_DATA_WRITE	Write to serial port
0x0615	SERIAL_DATA_READ	Read from serial port
0x0616	SERIAL_CONTROL_SET	Set control lines (DTR, RTS)
0x0617	SERIAL_CONTROL_GET	Get control line status
0x0618	SERIAL_BREAK	Send break condition

Table 18: Serial Operations

6.4.3. GPIO Operations (0x0620-0x062F)

GPIO operations enable control of General Purpose Input/Output pins.

Code	Name	Description
0x0620	GPIO_CHIP_LIST	List GPIO chips
0x0621	GPIO_LINE_INFO	Get line information
0x0622	GPIO_LINE_REQUEST	Request GPIO line
0x0623	GPIO_LINE_RELEASE	Release GPIO line
0x0624	GPIO_LINE_SET	Set line value
0x0625	GPIO_LINE_GET	Get line value
0x0626	GPIO_LINE_WATCH	Watch for edge events
0x0627	GPIO_EVENT	GPIO edge event notification
0x0628	GPIO_PWM_CONFIGURE	Configure PWM output
0x0629	GPIO_PWM_SET	Set PWM duty cycle

Table 19: GPIO Operations

6.4.4. I2C Operations (0x0630-0x063F)

I2C (Inter-Integrated Circuit) bus operations for sensor and peripheral communication.

Code	Name	Description
0x0630	I2C_BUS_LIST	List I2C buses
0x0631	I2C_BUS_SCAN	Scan for devices on bus
0x0632	I2C_TRANSFER	I2C read/write transfer
0x0633	I2C_WRITE_BYTE	Write single byte
0x0634	I2C_READ_BYTE	Read single byte
0x0635	I2C_WRITE_BLOCK	Write data block
0x0636	I2C_READ_BLOCK	Read data block
0x0637	I2C_SMBUS_COMMAND	SMBus protocol command

Table 20: I2C Operations

6.4.5. SPI Operations (0x0640-0x064F)

SPI (Serial Peripheral Interface) operations for high-speed peripheral communication.

Code	Name	Description
0x0640	SPI_BUS_LIST	List SPI buses
0x0641	SPI_DEVICE_OPEN	Open SPI device
0x0642	SPI_DEVICE_CLOSE	Close SPI device
0x0643	SPI_CONFIGURE	Configure mode, speed, bits
0x0644	SPI_TRANSFER	Full-duplex SPI transfer
0x0645	SPI_WRITE	SPI write-only
0x0646	SPI_READ	SPI read-only

Table 21: SPI Operations

6.4.6. CAN Bus Operations (0x0650-0x065F)

CAN (Controller Area Network) bus operations for automotive and industrial protocols.

Code	Name	Description
0x0650	CAN_INTERFACE_LIST	List CAN interfaces
0x0651	CAN_INTERFACE_OPEN	Open CAN interface
0x0652	CAN_INTERFACE_CLOSE	Close CAN interface
0x0653	CAN_CONFIGURE	Configure bitrate, mode
0x0654	CAN_FRAME_SEND	Send CAN frame
0x0655	CAN_FRAME_RECEIVE	Receive CAN frame
0x0656	CAN_FILTER_SET	Set receive filter
0x0657	CAN_ERROR_STATUS	Get error counters

Table 22: CAN Bus Operations

6.4.7. 1-Wire Operations (0x0660-0x066F)

1-Wire bus operations for temperature sensors, iButtons, and similar devices.

Code	Name	Description
0x0660	ONEWIRE_BUS_LIST	List 1-Wire buses
0x0661	ONEWIRE_SEARCH	Search for devices
0x0662	ONEWIRE_RESET	Reset bus
0x0663	ONEWIRE_READ_ROM	Read device ROM
0x0664	ONEWIRE_MATCH_ROM	Select device by ROM
0x0665	ONEWIRE_SKIP_ROM	Skip ROM (single device)
0x0666	ONEWIRE_READ	Read bytes
0x0667	ONEWIRE_WRITE	Write bytes

Table 23: 1-Wire Operations

6.4.8. GSM Modem Operations (0x0670-0x067F)

GSM modem operations provide direct access to cellular modems for SMS, GNSS (GPS), voice calls, and USSD services. These operations abstract the underlying AT command interface.

Code	Name	Description
0x0670	GSM_MODEM_INIT	Initialize GSM modem
0x0671	GSM_MODEM_STATUS	Get modem status
0x0672	GSM_NETWORK_REG	Network registration status
0x0673	GSM_SIGNAL_QUALITY	Signal quality (RSSI, BER)
0x0674	GSM_SMS_SEND	Send SMS via AT commands
0x0675	GSM_SMS_RECEIVE	Receive SMS notification
0x0676	GSM_SMS_LIST	List SMS messages
0x0677	GSM_SMS_DELETE	Delete SMS from SIM
0x0678	GSM_USSD_SEND	Send USSD command
0x0679	GSM_USSD_RESPONSE	USSD response
0x067A	GSM_GNSS_ENABLE	Enable/disable GNSS module
0x067B	GSM_GNSS_POSITION	Get GNSS position fix
0x067C	GSM_GNSS_CONFIG	Configure GNSS
0x067D	GSM_VOICE_DIAL	Initiate voice call
0x067E	GSM_VOICE_HANGUP	Hangup voice call
0x067F	GSM_VOICE_ANSWER	Answer incoming call

Table 24: GSM Modem Operations

GSM modem types supported:

* SIM7600: 4G LTE Cat-4 with GNSS
* SIM800: 2G GSM/GPRS
* Quectel EC25: 4G LTE

6.5. Telephony: SMS Gateway (0x0B50-0x0B7F)

The SMS Gateway operations provide application-level SMS handling for the MyClerk platform. Unlike the low-level GSM Modem operations which deal with AT commands and hardware, these operations provide intelligent message routing, TAN/OTP detection, and emergency fallback services. These are standard protocol operations in the Telephony category (0x0B00-0x0BFF).

6.5.1. SMS Gateway Operations (0x0B50-0x0B5F)

Code	Name	Description
0x0B50	SMS_CREATE	Create outgoing SMS
0x0B51	SMS_SEND	Send SMS via gateway
0x0B52	SMS_RECEIVE	Incoming SMS notification
0x0B53	SMS_STATUS	Query message delivery status
0x0B54	SMS_LIST	List messages in gateway
0x0B55	SMS_DELETE	Delete message from gateway
0x0B56	SMS_ROUTE_ADD	Add routing rule
0x0B57	SMS_ROUTE_REMOVE	Remove routing rule
0x0B58	SMS_ROUTE_LIST	List routing rules
0x0B59	SMS_GATEWAY_STATUS	Gateway health status

Table 25: SMS Gateway Operations

6.5.2. SMS Parsing Operations (0x0B60-0x0B6F)

SMS Parsing operations handle intelligent classification and extraction of structured data from SMS messages.

Code	Name	Description
0x0B60	SMS_PARSED	Parsed SMS event
0x0B61	SMS_CLASSIFY	Classify SMS type
0x0B62	SMS_TAN_RECEIVED	TAN/OTP received event
0x0B63	SMS_TAN_ANNOUNCE	Announce TAN to family
0x0B64	SMS_TAN_CLAIM	Claim TAN ownership
0x0B65	SMS_TAN_EXPIRE	TAN expiration notification
0x0B66	SMS_DELIVERY_RECEIVED	Delivery notification received
0x0B67	SMS_APPOINTMENT_RECEIVED	Appointment SMS received
0x0B68	SMS_SPAM_DETECTED	Spam SMS detected
0x0B69	SMS_ALERT_RECEIVED	Alert/warning SMS received

Table 26: SMS Parsing Operations

SMS classification types:

* 0x01: TAN/OTP (banking, 2FA codes)

* 0x02: Delivery notification (package tracking)

* 0x03: Appointment reminder

* 0x04: Marketing/Spam

* 0x05: Alert/Warning (weather, emergency)

* 0x06: Personal message

* 0x07: Unknown

6.5.3. Emergency Gateway Operations (0x0B70-0x0B7F)

Emergency Gateway operations provide fallback communication when primary channels (internet, Matrix) are unavailable. The SMS gateway automatically activates emergency mode during outages.

Code	Name	Description
0x0B70	EMERGENCY_GATEWAY_STATUS	Gateway health status
0x0B71	EMERGENCY_INTERNET_DOWN	Internet outage detected
0x0B72	EMERGENCY_INTERNET_UP	Internet restored
0x0B73	EMERGENCY_SMS_ALERT	Emergency alert via SMS
0x0B74	EMERGENCY_SMS_COMMAND	Emergency command via SMS
0x0B75	EMERGENCY_LOCATION_REQUEST	Request device location
0x0B76	EMERGENCY_LOCATION_RESPONSE	Device location response
0x0B77	EMERGENCY_HEALTH_CHECK	Remote health check request
0x0B78	EMERGENCY_HEALTH_RESPONSE	Health check response
0x0B79	EMERGENCY_MODE_ACTIVATE	Activate emergency mode
0x0B7A	EMERGENCY_MODE_DEACTIVATE	Deactivate emergency mode

Table 27: Emergency Gateway Operations

Emergency mode features:

* Automatic activation when internet connectivity is lost

* Predefined SMS commands for remote device control

* Location sharing via GNSS coordinates in SMS

* Health monitoring with battery and connectivity status

* Graceful transition back to normal operation

Note: As of the reference implementation, SMS Gateway operations (0xF350-0xF35F), SMS Parsing operations (0xF360-0xF36F), and Emergency Gateway operations (0xF370-0xF37F) have been relocated to the Telephony category (0x0B50-0x0B7F) for consistency. The vendor extension range assignments are retained for backwards compatibility but implementations SHOULD use the Telephony range.

6.6. Application Operations (0x0700-0x1DFF)

Application-layer operations covering knowledge management, settings, messaging, telephony, calendar, tasks, smart home, automation, AI assistant, synchronization, orchestration, presence, health, location, plugins, MDM, legal documents, mobility, travel, audit, audio pipeline, and desktop client operations are defined in the companion document [MYCLERK-OPS].

The companion document serves as the authoritative registry for all application-layer operation codes in the 0x0700-0x1DFF range, following the "Assigned Numbers" pattern established by Bluetooth SIG specifications. This separation allows the core protocol specification to remain stable while application operations evolve independently.

7. Payload Formats

Payloads are encoded using MessagePack (a subset of CBOR). This section defines payload structures using CDDL [RFC8610].

7.1. SESSION_INIT Payload

The initiator MUST always include its X25519 public key. It SHOULD additionally include its ML-KEM-768 encapsulation key (and set kex-mode: hybrid-mlkem768) to negotiate the hybrid post-quantum KEX. If the initiator cannot afford the 1184-byte ML-KEM public, it MAY omit mlkem-public and set kex-mode: classical-only; servers that require PQ by policy will refuse such a session.

```
session-init = {
    nonce: bstr .size 8,
    timestamp: uint,
    kex-mode: kex-mode-type,
    x25519-public: bstr .size 32,
    ; mlkem-public REQUIRED when kex-mode = hybrid-mlkem768
    ? mlkem-public: bstr .size 1184,
    ? capabilities: [* capability],
    ? device-id: bstr .size 16,
}

kex-mode-type = &(
    classical-only:    0,  ; X25519 only (legacy / constrained)
    hybrid-mlkem768:   1,  ; ML-KEM-768 + X25519 (default)
    ; 2 reserved for future hybrid-mlkem1024
)

capability = &(
    compression-lz4:     0,
    compression-zstd:    1,
    encryption-chacha20: 2,
    encryption-aes256gcm:3,
    fragmentation:       4,
    streaming:           5,
    federation:          6,
    vfs:                 8,
    request-correlation: 11,
    pq-mlkem768:         12, ; signals ML-KEM-768 support
)
```

7.2. SESSION_ACK Payload

The responder MUST echo a selected-kex-mode. When that mode is hybrid-mlkem768, the responder MUST include mlkem-ciphertext: the 1088-byte ML-KEM ciphertext produced by encapsulating against the initiator’s encapsulation key.

```
session-ack = {
    session-id: uint .size 2,
    nonce: bstr .size 8,
    selected-tier: uint .le 5,
    selected-kex-mode: kex-mode-type,
    x25519-public: bstr .size 32,
    ; mlkem-ciphertext REQUIRED when
    ; selected-kex-mode = hybrid-mlkem768
    ? mlkem-ciphertext: bstr .size 1088,
    ? selected-capabilities: [* capability],
}
```

7.3. SESSION_RESUME Payload

```
session-resume = {
    old-session-id: uint .size 2,
    ticket: bstr .size 64,
}

; Ticket structure (encrypted with server key, AES-256-GCM):
; - session-key: 32 bytes
; - device-id: 16 bytes
; - issued-at: 4 bytes (Unix timestamp)
; - expires-at: 4 bytes (Unix timestamp)
; - ticket-nonce: 8 bytes
```

8. Tier Compatibility

When endpoints support different maximum tiers, they MUST negotiate to the highest common tier. The server MUST respond with the minimum of its supported tier and the client’s requested tier.

Servers MAY enforce minimum tier requirements for specific operations. If a client requests an operation at an insufficient tier, the server MUST respond with error code 0x12 (FORBIDDEN) and include the required tier in the error payload.

8.1. Minimum Tier Requirements

The following operations have minimum tier requirements:

Operation Category	Minimum Tier
Lock/Unlock (physical access)	3
Key Exchange	4
Federation Operations	4
Emergency Operations	3

Table 28: Minimum Tier Requirements

9. Error Handling

Error codes are 8-bit values organized into ranges:

Range	Category
0x00-0x0F	Success
0x10-0x1F	Client Errors
0x20-0x2F	Server Errors
0x30-0x3F	Federation Errors
0xF0-0xFF	Reserved

Table 29: Error Codes

9.1. Specific Error Codes

0x00 OK Operation completed successfully.

0x10 BAD_REQUEST Malformed message or invalid parameters.

0x11 UNAUTHORIZED Authentication required or failed.

0x12 FORBIDDEN Insufficient permissions or tier.

0x13 NOT_FOUND Requested resource does not exist.

0x17 INVALID_SESSION Session expired or invalid.

0x20 INTERNAL_ERROR Server encountered an unexpected error.

0x21 SERVICE_UNAVAILABLE Service temporarily unavailable.

0x22 TIMEOUT Operation timed out.

10. Security Considerations

10.1. Nonce Reuse

Reusing a nonce with the same key in ChaCha20-Poly1305 completely compromises the confidentiality and authenticity of all messages encrypted with that key-nonce pair. Implementations MUST ensure nonces are never reused by:

* Using cryptographically random values for the Random field

* Maintaining a monotonic counter per session

* Rotating keys before counter exhaustion

10.2. Replay Protection

Tier 3+ messages include timestamps for replay protection. Implementations SHOULD reject messages with timestamps more than 5 minutes in the past or future. Session resumption tickets include a nonce that MUST be tracked to prevent replay attacks.

10.3. Tier Downgrade Attacks

An attacker might attempt to force communication at a lower tier. Servers MUST enforce minimum tier requirements for sensitive operations. Clients SHOULD warn users when connecting at a lower tier than expected.

10.4. Key Compromise

If a session key is compromised, an attacker can decrypt all messages in that session. The 24-hour automatic key rotation limits the window of exposure. For forward secrecy, implementations SHOULD use ephemeral ECDH keys for each session.

10.5. Post-Quantum Considerations

Tier 4 and Tier 5 sessions use hybrid post-quantum key establishment by default, combining ML-KEM-768 [FIPS203] with X25519 [RFC7748] via concatenation and HKDF-SHA256 extraction (see Section 5.1). The hybrid output remains secure as long as at least one of the two primitives does, which defeats both a future cryptographically relevant quantum computer (CRQC) running Shor’s algorithm against X25519 and any currently unknown classical break of ML-KEM.

The symmetric layer (ChaCha20-Poly1305 with 256-bit keys) is considered post-quantum secure: Grover’s algorithm reduces the effective key strength to 128 bits, which remains computationally infeasible to brute-force. No change to the symmetric primitive is required and none is prescribed.

Classical-only mode (kex-mode: classical-only) MAY be used by resource-constrained implementations (e.g. deeply embedded devices where the 1184-byte ML-KEM encapsulation key is infeasible over a low-bandwidth modem link) or for interoperability with legacy endpoints predating this draft. Classical-only mode MUST NOT be used for data requiring long-term confidentiality (conservatively, more than 10 years) due to store-now-decrypt-later attacks.

Implementations MUST NOT negotiate classical-only mode when at least one endpoint signals the pq-mlkem768 capability and local policy requires PQ for the session's intended use.

10.6. KEX Mode Downgrade Protection

Like tier negotiation (Section 10.3), KEX-mode negotiation is vulnerable to active on-path downgrade attempts. An attacker could strip mlkem-public from SESSION_INIT or remove the pq-mlkem768 capability to force classical-only. This draft binds the full handshake to the session key to make such modifications detectable.

Specifically, both endpoints fold a transcript hash of the complete SESSION_INIT and SESSION_ACK byte sequences into the HKDF-SHA256 info parameter during session key derivation (see Section 5.1). Any modification of any handshake field by an intermediary causes the two endpoints to observe different transcript hashes, derive different session keys, and fail AEAD verification on the first Tier 3 or Tier 5 frame. The domain separator strings "myclerk-session-v1-hybrid" and "myclerk-session-v1-classical" additionally prevent a hybrid-derived key from being confused with a classical-derived key of the same inputs.

Implementations MUST:

1. Include the exact wire-form bytes of SESSION_INIT and SESSION_ACK in the transcript hash, in the order they were sent.

2. Refuse to complete a session where the responder's selected KEX mode is weaker than the mode the initiator offered AND the local policy requires the stronger mode.

3. Log classical-only negotiations for audit purposes so operators can detect systemic downgrade attempts.

10.7. ML-KEM Implementation Hazards

ML-KEM-768 implementations MUST observe the following safety properties. These are standard Fujisaki-Okamoto precautions but are called out explicitly because ahistorical implementations have repeatedly failed at them.

1. *Constant-time decapsulation.* The decapsulation routine MUST run in time independent of the secret key and the ciphertext contents, to prevent timing side channels that can recover the private key over repeated queries.

2. *Implicit rejection.* On a ciphertext-validity failure, decapsulation MUST return a pseudo-random shared secret derived from the private key and the ciphertext (the Fujisaki-Okamoto transform's "implicit rejection" branch), NOT an error. Returning an error leaks ciphertext-validity information to the attacker.

3. *Private-key hygiene.* ML-KEM decapsulation private keys MUST be cleared from memory (e.g. via crypto/subtle.ConstantTimeCompare-adjacent zero-wipe helpers) once the session key has been derived and the ephemeral has no further use.

Implementations SHOULD use a well-reviewed library rather than implementing ML-KEM from scratch. In the Go ecosystem the standard library crypto/mlkem package (available since Go 1.24) satisfies all three requirements and is the RECOMMENDED choice; the liboqs project, BoringSSL's ML-KEM implementation, and libcrux are also suitable alternatives.

11. IANA Considerations

11.1. Service Name and Port Number Registration

IANA is requested to register the following service name and port number in the "Service Name and Transport Protocol Port Number Registry" [RFC6335]:

Service Name: myclerk

Port Number: 5657

Transport Protocol(s): TCP, UDP

Description: MyClerk Protocol

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Reference: This document

TCP is the primary transport for control messages, synchronization, authentication, and VFS operations. UDP is used for real-time streaming (video, audio) and low-latency sensor telemetry where latency is prioritized over reliability.

11.2. Operation Code Registry

IANA is requested to create a new registry titled "MyClerk Protocol Operation Codes" with the following structure:

Registration Procedure: Expert Review per [RFC8126]

Reference: This document

The initial entries are the operation codes defined in Section 6 of this document, organized into the following ranges:

Range	Category	Reference
0x0001-0x00FF	Core Operations	Section 4
0x0100-0x01FF	Standard Operations (Device, Identity, Messaging)	Section 5
0x0200-0x02FF	Resource Sharing (Streams, GPU, Routing)	Section 6
0x0300-0x03FF	Federation	Section 6
0x0400-0x04FF	Billing and Economics	Section 6
0x0500-0x05FF	Virtual File System (VFS)	draft-myclerk-vfs
0x0600-0x06FF	Hardware Passthrough (USB, GPIO, I2C, SPI, CAN)	Section 6
0x0700-0x07FF	Knowledge Base	Section 6
0x0800-0x08FF	Settings & Configuration	Section 6
0x0900-0x09FF	Room/Channel Management	Section 6
0x0A00-0x0AFF	Media Handling	Section 6
0x0B00-0x0BFF	Telephony (Voice, Video, SMS, IVR)	Section 6
0x0C00-0x0CFF	Calendar & Events	Section 6
0x0D00-0x0DFF	Tasks, Shopping & Gamification	Section 6

0x0E00-0x0EFF	Smart Home Devices	Section 6
0x0F00-0x0FFF	Automation Rules	Section 6
0x1000-0x10FF	Maya AI Assistant	Section 6
0x1100-0x11FF	Synchronization & VFS Federation	Section 6
0x1200-0x12FF	Orchestration (Notifications, Finance)	Section 6
0x1300-0x13FF	Presence & Status (Education)	Section 6
0x1400-0x14FF	Health (Social, Community)	Section 6
0x1500-0x15FF	Location & Mobility	Section 6
0x1600-0x16FF	Debug, Testing & Plugins	Section 6
0x1700-0x17FF	Mobile Device Management (MDM)	[MYCLERK-OPS]
0x1800-0x18FF	Legal Documents & Care Directives	[MYCLERK-OPS]
0x1900-0x19FF	Mobility & Vehicle Management	[MYCLERK-OPS]
0x1A00-0x1AFF	Travel & Vacation	[MYCLERK-OPS]
0x1B00-0x1BFF	Audit & Compliance	[MYCLERK-OPS]
0x1C00-0x1CFF	Audio Pipeline	[MYCLERK-OPS]
0x1D00-0x1DFF	Desktop Client	[MYCLERK-OPS]
0x1E00-0xEFFF	Unassigned	This document
0xF000-0xFFFE	Vendor Extensions (third-party/vendor-specific use only)	This document
0xFFFF	Reserved	This document

Table 30

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8439] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 8439, DOI 10.17487/RFC8439, June 2018, <https://www.rfc-editor.org/info/rfc8439>.

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

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

[FIPS203] National Institute of Standards and Technology, "Module-Lattice-Based Key-Encapsulation Mechanism Standard", FIPS 203, August 2024, <https://doi.org/10.6028/NIST.FIPS.203>.

12.2. Informative References

[MYCLERK-OPS] Arcan, M.J., "MyClerk Protocol Application Operations Registry", Work in Progress, Internet-Draft, draft-myclerk-protocol-ops-00, 2026, <https://datatracker.ietf.org/doc/html/draft-myclerk-protocol-ops-00>.

Acknowledgements

This specification is part of the MyClerk project, a privacy-first family orchestration platform currently in development. For more information, visit https://myclerk.eu.

Author’s Address

Michael J. Arcan
Arcan Consulting
Email: rfc@arcan-consulting.de
URI: https://myclerk.eu