

Independent Submission  
Internet-Draft  
Intended status: Experimental  
Expires: 14 July 2026

M.J. Arcan  
Arcan Consulting  
10 January 2026

The MyClerk Protocol: Tiered Security Communication for Distributed  
Family Systems  
draft-myclerk-protocol-01

## 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 and X25519 key exchange.

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 14 July 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 . . . . .	3
1.1. Requirements Language . . . . .	3
1.2. Terminology . . . . .	3
2. Protocol Overview . . . . .	4
2.1. Design Goals . . . . .	4
2.2. Security Tiers Overview . . . . .	4
3. Message Format . . . . .	5
3.1. Common Header Fields . . . . .	5
3.2. Tier 0 Header (1 byte) . . . . .	5
3.3. Tier 1 Header (4 bytes) . . . . .	6
3.4. Tier 2 Header (6 bytes) . . . . .	6
3.5. Tier 3 Header (12 bytes) . . . . .	6
3.6. Tier 4 Header (42 bytes) . . . . .	7
3.7. Tier 5 Header (58 bytes) . . . . .	7
4. Nonce Construction . . . . .	8
5. Key Derivation . . . . .	8
5.1. ECDH Key Exchange . . . . .	8
5.2. Key Rotation . . . . .	9
6. Protocol Operations . . . . .	9
6.1. Core Operations (0x0000-0x00FF) . . . . .	10
6.1.1. Session Management (0x0000-0x000F) . . . . .	10
6.1.2. Key Management (0x0010-0x001F) . . . . .	11
6.2. Resource Sharing Operations (0x0200-0x02FF) . . . . .	11
6.2.1. Device Management (0x0200-0x020F) . . . . .	11
6.2.2. Stream Operations (0x0210-0x021F) . . . . .	12
6.2.3. GPU Service Operations (0x0220-0x022F) . . . . .	12
6.2.4. Dynamic Routing and ACL (0x0240-0x024F) . . . . .	13
6.3. Hardware Passthrough Operations (0x0600-0x06FF) . . . . .	14
6.3.1. USB Operations (0x0600-0x060F) . . . . .	14
6.3.2. Serial Operations (0x0610-0x061F) . . . . .	15
6.3.3. GPIO Operations (0x0620-0x062F) . . . . .	16
6.3.4. I2C Operations (0x0630-0x063F) . . . . .	17
6.3.5. SPI Operations (0x0640-0x064F) . . . . .	18
6.3.6. CAN Bus Operations (0x0650-0x065F) . . . . .	19
6.3.7. 1-Wire Operations (0x0660-0x066F) . . . . .	19
7. Payload Formats . . . . .	20
7.1. SESSION_INIT Payload . . . . .	20
7.2. SESSION_ACK Payload . . . . .	21
7.3. SESSION_RESUME Payload . . . . .	21
8. Tier Compatibility . . . . .	21

8.1. Minimum Tier Requirements . . . . .	21
9. Error Handling . . . . .	22
9.1. Specific Error Codes . . . . .	22
10. Security Considerations . . . . .	23
10.1. Nonce Reuse . . . . .	23
10.2. Replay Protection . . . . .	23
10.3. Tier Downgrade Attacks . . . . .	23
10.4. Key Compromise . . . . .	23
10.5. Post-Quantum Considerations . . . . .	23
11. IANA Considerations . . . . .	24
12. References . . . . .	24
12.1. Normative References . . . . .	24
Acknowledgements . . . . .	25
Author's Address . . . . .	25

## 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.

### 2.2. Security Tiers Overview

Tier	Header Size	Encryption	Authentication	Use Case
0	1 byte	None (tunneled)	None	Inside secure session
1	4 bytes	None	None	Fire-and-forget commands
2	6 bytes	Optional	CRC-16	Home automation
3	12 bytes	ChaCha20-Poly1305	HMAC-32	Conversational

4	42 bytes	ChaCha20-Poly1305	HMAC-64	Key exchange
5	58+ bytes	ChaCha20-Poly1305	HMAC-256 + Poly1305	Maximum security

Table 1: Security Tier Summary

### 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:

```

0
0 1 2 3 4 5 6 7
+---+---+---+---+
|V V|T T T|C|F|E|
+---+---+---+---+

```

V: Protocol Version (2 bits) - Currently 0  
T: Security Tier (3 bits) - Values 0-5  
C: Compressed (1 bit) - Payload is compressed  
F: Fragmented (1 bit) - Message is fragmented  
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.

```

0
0 1 2 3 4 5 6 7
+---+---+---+---+
|0 0|0 0 0|C|F|E|
+---+---+---+---+

```

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)

```

      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-----+-----+-----+-----+
      |V V|0 0 1|C|F|E|           Operation Code           | Sequence |
      +-----+-----+-----+-----+

```

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)

```

      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-----+-----+-----+-----+
      |V V|0 1 0|C|F|E|           Operation Code           | Sequence |
      +-----+-----+-----+-----+
      |                               Session ID              |
      +-----+-----+-----+-----+

```

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)

```

      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-----+-----+-----+-----+
      |V V|0 1 1|C|F|E|           Operation Code           | Sequence |
      +-----+-----+-----+-----+
      |           Session ID           |           Timestamp           |
      +-----+-----+-----+-----+
      |           Timestamp (cont.)     |           Nonce             |
      +-----+-----+-----+-----+

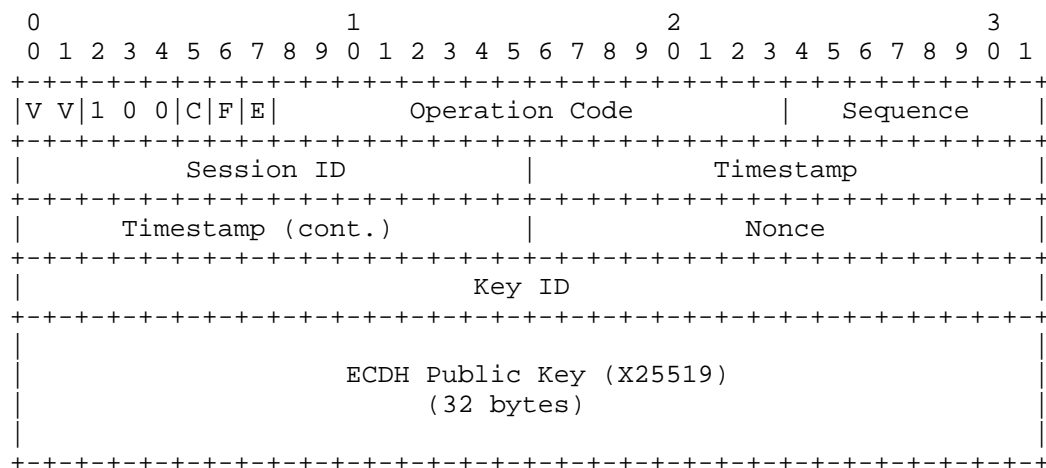
```

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 (42 bytes)

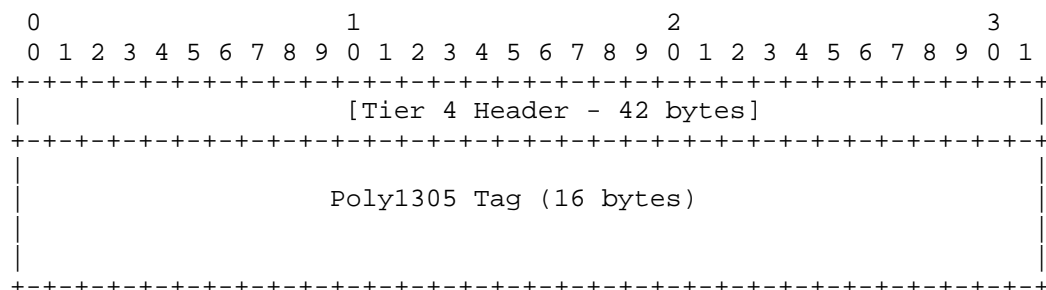


Key ID (32 bits) Identifier for the key being used or negotiated.

ECDH Public Key (256 bits) X25519 public key as specified in [RFC7748].

## 3.7. Tier 5 Header (58 bytes)

Tier 5 extends Tier 4 with a full Poly1305 authentication tag in the header:



Tier 5 messages MUST include a full HMAC-SHA256 (32 bytes) in the trailer, providing dual authentication: Poly1305 for AEAD integrity and HMAC-SHA256 for post-quantum resistance.

#### 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:

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Timestamp (32 bits)                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Random (32 bits)                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Counter (32 bits)                                    |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

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. ECDH Key Exchange

For Tier 4+ sessions:

```
shared_secret = X25519(local_private_key, remote_public_key)
```

```

session_key = HKDF-SHA256(
    IKM = shared_secret,
    salt = nonce_initiator || nonce_responder,
    info = "myclerk-session-v0",
    L    = 32
)

```

## 5.2. 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)
0xF000-0xFFFFE	Vendor Extensions
0xFFFF	Reserved

Table 2: 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 3: 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 4: Key Management Operations

## 6.2. 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.

## 6.2.1. 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
--------	---------------------	---------------------------

Table 5: Device Management Operations

6.2.2. 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 6: 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)

6.2.3. 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 7: 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)

#### 6.2.4. 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 8: Dynamic Routing Operations

Access modes:

- \* 0x00: None (no access)
- \* 0x01: Read-only
- \* 0x02: Write-only
- \* 0x03: Read-write
- \* 0x04: Exclusive (single client)

### 6.3. 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.

#### 6.3.1. 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 9: USB Operations

### 6.3.2. 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 10: Serial Operations

#### 6.3.3. 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 11: GPIO Operations

#### 6.3.4. 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 12: I2C Operations

#### 6.3.5. 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 13: SPI Operations

#### 6.3.6. 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 14: CAN Bus Operations

#### 6.3.7. 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 15: 1-Wire Operations

## 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

```

session-init = {
  nonce: bstr .size 8,
  timestamp: uint,
  ? capabilities: [* capability],
  ? device-id: bstr .size 16,
}

capability = &(
  compression-lz4: 0,
  compression-zstd: 1,
  encryption-chacha20: 2,
  encryption-aes256gcm: 3,
  fragmentation: 4,
  streaming: 5,
  federation: 6,
  vfs: 8,
)

```

## 7.2. SESSION\_ACK Payload

```
session-ack = {  
    session-id: uint .size 2,  
    nonce: bstr .size 8,  
    selected-tier: uint .le 5,  
    ? 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 16: 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 17: 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

The X25519 key exchange is not quantum-resistant. Tier 5 includes an additional HMAC-SHA256 trailer that provides some protection against future quantum attacks on Poly1305. Implementations concerned about long-term confidentiality SHOULD use Tier 5 for sensitive data.

## 11. IANA Considerations

This document has no IANA actions.

If this protocol were to be standardized, IANA would be requested to create a registry for MyClerk Protocol operation codes with the ranges defined in Section 6.

## 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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.

## 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](mailto:rfc@arcan-consulting.de)  
URI: <https://myclerk.eu>