📜 EMSG Protocol Specification (v0.1)

# 1. Overview

EMSG (Extended Messaging Service) is a decentralized, email-inspired communication protocol designed for modern messaging needs. It simplifies legacy email by removing unnecessary fields (subject, bcc), embraces structured JSON messaging, supports group messaging via cc, and ensures persistent group threads using a globally unique group\_id.

# 2. Addressing Format

Format: user#domain.com  
The '#' is used in place of '@' to distinguish EMSG from traditional email.  
Example:  
charu#magadhaempire.com  
king#empire.org

# 3. DNS Discovery

EMSG domains must publish discovery information via DNS:  
EMX Record (like MX in email):  
emx 10 emsg.magadhaempire.com.  
SRV Record (optional fallback):  
\_emsg.\_tcp.magadhaempire.com. 10 5 5888 emsg.magadhaempire.com.

# 4. Transport Protocol (ESMP)

Protocol Name: ESMP (Extended Simple Messaging Protocol)  
Default Port: 5888  
Transport: TCP (with TLS); optionally HTTP/WebSocket  
Payload Format: JSON

# 5. Message Schema

Common Message Format:  
{  
 "id": "uuid-v4",  
 "from": "king#empire.org",  
 "to": ["charu#magadhaempire.com"],  
 "cc": ["sandip#magadhaempire.com"],  
 "type": "chat",  
 "group\_id": "group-4b62a3f5.empire.org",  
 "timestamp": 1717850501,  
 "body": "Welcome to the court!",  
 "signature": "base64signature"  
}

# 6. Message Types

chat: Standard user message in one-to-one or group format.

system: System-level events that control group state or protocol operations.

Supported system Events:  
- group\_created: Initializes a persistent group.  
- joined: A user joined the group.  
- left: A user voluntarily left.  
- removed: A user was removed.  
- name\_updated: Optional name metadata.

# 7. Group ID Design

A globally unique group\_id ensures persistent group identity.  
Suggested format: group-<uuid>.<originating-domain>  
Example: group-4b62a3f5.empire.org  
Generated by the creator’s server using a UUID to avoid collisions.  
All group messages must reference the group\_id.

# 8. Authentication

- Use public-private key pairs for account identity.  
- Server-side JWT or OAuth2 tokens for session auth.  
- All messages must be signed using the sender’s private key.  
- Signature format: base64-encoded cryptographic signature of message content.

# 9. Encryption

- Transport: TLS for all communications.  
- End-to-End Encryption (E2EE) is optional and per-recipient.  
- Message-level encryption metadata should include key info per recipient.

# 10. Delivery & Routing

- Based on to and cc fields.  
- If destination domain is unreachable, the message is queued.  
- Servers may retry delivery or notify sender of failure.

# 11. Optional Features

- attachments: Referenced by URL or base64 with size limits.  
- typing indicator: Sent as system message.  
- read\_receipt: Optional acknowledgment messages.  
- history\_sync: New members may receive previous chat history via opt-in.

# 12. Error Codes

1001: Invalid signature  
1002: Group not found  
1003: Unauthorized action  
1004: Delivery failed  
1005: Group ID already exists

# 13. Versioning & Negotiation

- Message format includes version key (optional).  
- Clients and servers should declare version support in hello handshake.  
- Default version: 1.0

# 14. Federation Model

- Any domain may host an EMSG server and participate.  
- Identity is based on domain authority, not centralized registry.  
- Servers communicate using the ESMP protocol and resolve via DNS.

# 15. Implementation Considerations

- Server-to-server: TCP listener on port 5888, ESMP handler.  
- Client-to-server: WebSocket or HTTPS.  
- Minimal database schema: users, groups, messages, keys.

# 16. Roadmap

1. Define protocol spec  
2. Select existing repo or tech base (e.g., Stalwart, IronPigeon)  
3. Build PoC server-client pair with chat + system messages  
4. Add E2EE, group sync, DNS discovery  
5. Release EMSG v1.0 with web client and CLI