Zero Strategy Research

Zero Protocol Strategy Research

The Core Question

"Should I create a new protocol for Zero Gram?"

The Short Answer: YES, you must create a new **Application Layer Protocol** (The "Zero Protocol") to handle encryption, identity, and data structure. **NO**, you should not create a new **Transport Layer Protocol** (like a new TCP, SMTP, or custom P2P consensus). Use existing, battle-tested transports (JMAP, HTTP, WebSockets) to move your custom-protocol data.

---

1. The Two Types of "Protocols"

It is critical to distinguish between these two layers:

A. Transport Protocol (How data moves)

- **Examples:** SMTP, IMAP, JMAP, TCP, HTTP, QUIC.
- **Role:** Getting bytes from Alice to Bob.
- **Recommendation: DO NOT BUILD THIS.**
  - *Why?* It takes years to stabilize, firewalls will block it, and mobile networks (iOS/Android) break custom background connections.
  - *Strategy:* Use **JMAP (RFC 8620)**. It is efficient, mobile-friendly, and distinct from the legacy IMAP.

B. Application Protocol (What the data means)

- **Examples:** Signal Protocol, PGP, Atomic Mail Protocol, Matrix.
- **Role:** Ensuring those bytes are readable ONLY by Alice and Bob.
- **Recommendation: BUILD THIS.**
  - You need a standard way to package: `[Encrypted Body] + [Encrypted Metadata] + [Digital Signature] + [Key Exchange Header]`.
  - This is your "Zero Protocol".

---

2. Recommendation: The "Zero Protocol" Stack

Do not reinvent the wheel. Assemble your protocol from these standard primitives:

Layer	Component	Choice settings

Identity	Decentralized ID	**Ed25519** Public Keys (Users are identified by keys, not servers)
Key Exchange	Asymmetric	**X3DH** (Triple Diffie-Hellman) or **ECIES** (simpler, email-like)
Content	Symmetric	**AES-256-GCM** (Fast, secure, authenticated)
Transport	Delivery layer	**JMAP** (Phase 1) → **Nostr/Relays** (Phase 2)

Why this works?

1. **Mobile Support:** JMAP works perfectly with standard HTTP, so iOS/Android won't kill your app's background process.
2. **True Privacy:** The server (JMAP) only sees "blobs" of data. It doesn't know if it's an email or a chat. It just syncs encrypted blobs.
3. **Future Proof:** You can swap the "Transport" layer later (replace JMAP with P2P/IPFS) without changing your "Application" protocol (the encryption).

---

3. Comparison with Alternatives

Option	Pros	Cons	Verdict
Custom TCP Protocol	Total control, max performance	Blocked by firewalls, hard to maintain, security risks	❌ AVOID
Signal Protocol fork	Gold standard security (Forward Secrecy)	Very complex key management (PreKeys), hard for "offline" email	⚠️ TOO COMPLEX
PGP over SMTP	Standard email compat	Leaks metadata (Subject, Sender), bad UX, no forward secrecy	❌ OUTDATED
"Zero Protocol" Application Layer	Perfect privacy, modern UX, metadata encryption	Requires custom client (Zero Mail app)	✅ **WINNER**

4. Conclusion

You are building: An **Encrypted Application Protocol** that runs ON TOP OF **JMAP**.

Name it: "The Zero Protocol" **Spec:**

1. **Payload:** JSON packet containing AES-256 encrypted ciphertext.
2. **Metadata:** Encrypted headers (Subject, Sender, Time) visible only to client.
3. **Transport:** JMAP `blob/upload` and `blob/download`.

Decision: PROCEED with designing the "Zero Protocol" data structure (Application Layer) but use standard JMAP for moving the bits.

-
-
-
-
-

Zero Protocol Specification v1.0

Zero Protocol Specification v1.0

1. Overview

Zero Protocol is an application-layer encrypted messaging protocol designed for **Zero Mail**. It ensures that all data (content + metadata) is encrypted on the client device before being transported. It is transport-agnostic but designed to be carried over **JMAP**.

2. Cryptographic Primitives

Component	Primitive	Purpose
Identity Keys	**Ed25519**	Long-term user identity (Signing/Verification).
Key Exchange	**X25519** (ECDH)	Establishing shared secrets.
Encryption	**AES-256-GCM**	Symmetric content encryption with authentication tag.
Key Derivation	**HKDF-SHA256**	Deriving strong encryption keys from shared secrets.
Serialization	**JSON** (UTF-8)	Packet format (Header).

3. The "Zero Packet" Structure

Every message in Zero Mail is a "Zero Packet". The server sees only this JSON blob.

```
{
 "v": 1,
 "header": {
   "sender_id": "Base64(Ed25519_Public_Key)",
   "recipient_id": "Base64(Ed25519_Public_Key)",
   "ephemeral_key": "Base64(X25519_Public_Key)",
   "nonce": "Base64(12_bytes)",
   "timestamp": 1700000000
 },
 "ciphertext": "Base64(Encrypted_Payload)",
 "signature": "Base64(Ed25519_Signature)"
}
```

3.1 Payload (Inside `ciphertext`)

When decrypted, the `ciphertext` reveals the inner payload JSON:

```json
{
 "type": "email",
 "meta": {
  "subject": "Hello World",
  "received_at": 1700000000,
  "reply_to": "..."
 },
 "body": {
  "text": "This is a secret message.",
  "html": "<p>This is a secret message.</p>",
  "attachments": [...]
 }
}
```

4. Workflows

4.1 Encryption (Alice sends to Bob)

1. **Prepare**: Alice loads Bob's Identity Key (`Bob_ID_Pub` -> Convert to X25519 `Bob_X_Pub`).
2. **Ephemeral**: Alice generates a random ephemeral key pair (`Eph_Priv`, `Eph_Pub`).
3. **Key Exchange**: Calculate Shared Secret `S = X25519(Eph_Priv, Bob_X_Pub)`.
4. **Derive**: `Encryption_Key = HKDF(S, salt=nonce, info="ZeroProtocol_v1")`.
5. **Encrypt**: `Ciphertext = AES-256-GCM(Key=Encryption_Key, IV=nonce, Data=Payload_JSON)`.
6. **Construct Header**: `{ sender_id, recipient_id, ephemeral_key, nonce ... }`.
7. **Sign**: `Signature = Ed25519_Sign(Alice_ID_Priv, "Header + Ciphertext")`.
8. **Pack**: Assemble final JSON.

4.2 Decryption (Bob receives from Alice)

1. **Parse**: Extract `Header`, `Ciphertext`, `Signature`, `Sender_ID`.
2. **Verify**: Check `Ed25519_Verify(Sender_ID, Signature, "Header + Ciphertext")`. **STOP** if invalid.
3. **Prepare**: Identify `Ephemeral_Key` from Header.
4. **Key Exchange**: Calculate Shared Secret `S = X25519(Bob_Secret_Key, Ephemeral_Key)`.
5. **Derive**: `Decryption_Key = HKDF(S, salt=nonce, info="ZeroProtocol_v1")`.
6. **Decrypt**: `Payload = AES-256-GCM_Decrypt(Key=Decryption_Key, IV=nonce, Data=Ciphertext)`.

5. Implementation Roadmap (Rust)

We will implement this in `src-tauri/src/protocol/`.

Structs

- `ZeroIdentity` (Keypair management)
- `ZeroPacket` (The wire format)
- `ZeroPayload` (The inner content)

Dependencies `Cargo.toml`

```toml
[dependencies]
ed25519-dalek = "2.1"
x25519-dalek = "2.0"
aes-gcm = "0.10"
hkdf = "0.12"
sha2 = "0.10"
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"
base64 = "0.21"
rand = "0.8"
```