Abracadabracadoo Protocol Specification
Version: 1.0
Date: May 5, 2025

---

1. Overview

The **Abracadabracadoo Protocol** is a nested-AEAD, hash-nonce end-to-end encryption scheme with server-mediated proof release. It lets a sender (A) bind a "proof token" (P) to the message ciphertext (EM) via hash-derived nonces, so that a recipient (B) can trigger the release of P—and anyone with server logs can later verify that B obtained a sound, decryptable message—without the server ever seeing the plaintext.

---

2. Terminology

- **A**: Sender (Alice)
- **B**: Recipient (Bob)
- **S**: Server
- **M**: Plaintext message payload
- **P**: Proof token (e.g. HMAC of `M‖timestamp`)
- **EM**: Encrypted message:

  EM=AEAD_Enc(Kmsg, nonce=HP[0..11], M) \text{EM} = \mathrm{AEAD_Enc}\bigl(K_{\text{msg}},\ \text{nonce}=H_P[0..11],\ M\bigr)EM=AEAD_Enc(Kmsg, nonce=HP[0..11], M)

- **EP**: Encrypted proof:

  EP=AEAD_Enc(Kproof, nonce=HEM, P) \text{EP} = \mathrm{AEAD_Enc}\bigl(K_{\text{proof}},\ \text{nonce}=H_{EM},\ P\bigr)EP=AEAD_Enc(Kproof, nonce=HEM, P)

- **H_P**: SHA-256 hash of P (`H_P = SHA256(P)`)
- **H_EM**: SHA-256 hash of EM (`H_EM = SHA256(EM)`)
- **msgID**: Unique message identifier
- **T1, T2**: Server-logged timestamps

---

3. Protocol Flow

3.1 Preparation

1. **Key Establishment**
   - A & B share `K_msg` (for EM) via any E2EE handshake (e.g. double-ratchet).
   - A & S share `K_proof` (for EP) via a separate secure channel.

3.2 Encryption by A

1. **Compute Proof Token**
   - `P = HMAC(K_proof, M ‖ timestamp)` (or similar).
2. **Hash & Encrypt Message**
   - `H_P = SHA256(P)` → derive 96-bit nonce prefix.
   - `EM = AEAD_Enc(K_msg, nonce=H_P[0..11], plaintext=M)`
3. **Hash & Encrypt Proof**
   - `H_EM = SHA256(EM)`
   - `EP = AEAD_Enc(K_proof, nonce=H_EM, plaintext=P)`
4. **Send to Server**
   - A transmits `(msgID, EM, EP)` to S.
   - S logs `(msgID, H_EM, T1)`.

3.3 Receipt Challenge by B

1. **B Receives** `(EM, EP)`.
2. **Compute Outer Hash**
   - `H_EM′ = SHA256(EM)`.
3. **Return Challenge**
   - B sends `(msgID, H_EM′)` to S.

3.4 Proof Release by S

1. **Verify** `H_EM′ == H_EM` (from log).
2. **Decrypt Proof**
   - `P = AEAD_Dec(K_proof, nonce=H_EM, ciphertext=EP)`
3. **Log & Deliver**
   - S logs `(msgID, P, T2)` (no plaintext).
   - S forwards `P` to B.

3.5 Verification

Any party (A, B, or auditor) can:

1. Recompute `EP` and its hash `H_EM`,
2. Check `H_EM` against the server log,
3. Verify that decrypting `EP` under `K_proof` yields `P`,
4. Decrypt `EM` under `nonce=SHA256(P)[0..11]` and `K_msg` to recover `M`.

---

4. Security Properties

- **End-to-End Encryption:** Only A & B (holding `K_msg`) see `M`.
- **Proof without Plaintext Exposure:** S never sees `M`, only hashes and proof tokens.
- **Ephemeral, User-Controlled Proofs:** Deleting `K_proof` or logs irreversibly destroys all proof tokens.
- **Hash-Derived Nonces:** Nested hash nonces bind message and proof together, preventing replay or tampering.

---

5. Implementation Notes

- **AEAD Ciphers:** Use AES-SIV (RFC 5297) or ChaCha20-Poly1305 with careful nonce handling.
- **Hash Function:** SHA-256 for nonce derivation.
- **Logging:** Store only `(msgID, H_EM, P)` in an append-only, tamper-evident log.
- **Nonce Length:** Truncate 256-bit hash to 96 bits for AEAD nonces where required.

Primary Differences between Abracadabra and Abracadabracadoo

Aspect	Abracadabra	Abracadabracadoo
Layering	Two symmetric layers: inner `C_inner` under `K_AB`, outer `C_locked` under hash of `C_inner`. Abracadabra Protocol Sp…	Two nested AEADs: inner EM under hash of P, outer EP under hash of EM.
Proof Token	None—relies on server releasing hash `h` for decryption.	Explicit `P` token encrypted and released.
Nonce Derivation	`h = SHA256(C_inner)` for outer AEAD; no nonce for inner.	`H_P = SHA256(P)` for EM nonce; `H_EM = SHA256(EM)` for EP nonce.
Server Work	Logs `(msgID, h, T1/T2)`, releases `h` to B.	Logs `(msgID, H_EM, T1)` then `(msgID, P, T2)`, releases `P`.

Aspect	Abracadabra	Abracadabracadoo
Verification	B uses `h` to decrypt `C_outer`, then `C_inner`.	Verifier recomputes nested hashes to decrypt EP → P, then EM → M.
Proof Granularity	Proves B could decrypt `C_inner`.	Proves B obtained exact `P` (hence exact `M`) via nested AEAD.
Ephemerality	Deleting `h` and logs destroys proof.	Deleting `K_proof`, `P`, and logs destroys proof.

Abracadabracadoo shifts from a simple hash-release of an AEAD key to a **fully nested AEAD** that encapsulates an explicit proof token, giving you stronger binding between the message and its proof while preserving erasability and deniability.